**Bug ID#1 [Fixed]**

## Incorrect Inheritance

**Vulnerability Type**
Incorrect inheritance

**Severity**
High

**Description:**
The smart contracts were using a contract called "`BlockListed.sol`" to blacklist or whitelist a user. This contract was inherited by "`Airdrop.sol`", `Crowdsale.sol`", "`MasterContract.sol`", and "`TokenVesting.sol`".
Due to a spelling error in the contract name inside "`BlockListed.sol`", the compilation of the whole project failed and gave errors. This broke the contract and the blacklisting feature.

**Vulnerable Code:**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2;

contract BlockListed {
    mapping(address=>bool) isBlacklisted;

    function blackList(address _user) public {
        require(!isBlacklisted[_user], "user already
blacklisted");
        isBlacklisted[_user] = true;
    }

    function removeFromBlacklist(address _user) public {
        require(isBlacklisted[_user], "user already
whitelisted");
        isBlacklisted[_user] = false;
    }

}
```

**PoC:**

1. Go to the contract "`BlockListed.sol`" and note the error on Line 4. https://github.com/web3dev3/bck3-za2f/blob/main/contracts/BlockListed.sol#L4

**Impacts:**

This vulnerability may lead to compilation errors and inconclusive and erroneous logic which may have adverse effects on the flow of the contract.

**Remediation:**

Correct the name of the contract used so that the inheritance works and the blacklist feature can function.

**Retest:**

A fix was deployed by correcting "`BlockLkisted`" to "`BlockListed`"

**Bug ID#2**

## Use of Floating Pragma

**Vulnerability Type**
Floating pragma - SWC-103

**Severity**
Low

**Description:**
Solidity allows developers to specify a pragma version or a range of pragma versions with which the contracts can be compiled.
The contracts were found to be using a floating pragma which is not considered safe as it can be compiled with all the versions described.

**Vulnerable Code:**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2; //Floating pragma
```

**Vulnerable Contracts:**

- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L1
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/AllowanceCrowdsale.sol#L2
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/BlockListed.sol#L2
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Crowdsale.sol#L2
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L2
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TimedCrowdsale.sol#L2

**PoC:**

1. Go to the line numbers mentioned in the contracts and note the floating pragma "`^0.8.2`".

**Impacts:**
Compiling contracts with a floating pragma allows the contracts to be compiled with all the compiler versions specified in the floating pragma. This may cause issues because of

unforeseen breaking changes usually implemented in solidity compiler versions and/or various bug fixes introduced in later versions.

**Remediation:**
It is recommended to use a static pragma version, as future compiler versions may handle certain language constructions in a way the developer did not foresee. The developers should always use the exact Solidity compiler version when designing their contracts as it may break the changes in the future.
Use pragma solidity `0.8.2`; instead.

**Retest:**

**Bug ID#3**

## Outdated Compiler Version

**Vulnerability Type**
Outdated compiler version SWC-102

**Severity**
Low

**Description:**
The smart contracts were found to be using Solidity Compiler version `0.8.2` and above. Version `0.8.2` is affected by multiple security issues and bugs as specified in the Solidity repository.

**Vulnerable Code:**

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.2; //Vulnerable compiler version
```

**PoC:**

1. Open any contract and note the solidity pragma version specified at the top of the file.

**Impacts:**
The Solidity compiler version `0.8.2` is affected by vulnerabilities like "`SignedImmutables`", "`ABIDecodeTwoDimensionalArrayMemory`", and "`KeccakCaching`".

**Remediation:**
Use the latest version or most stable version of the solidity compiler with fewer known bugs. Anything above version `0.8.4` can be used.

**Retest:**

**Bug ID#4**

## Redundant SafeMath

**Vulnerability Type**
Code With No Effects - SWC-135

**Severity**
Medium

**Description:**
Starting from Solidity version `0.8.0+`, it is not recommended to use `SafeMath` for overflow or underflow checks because the Solidity compiler has default built-in validations for these making the usage of `SafeMath` redundant. Also, `SafeMath` is gas inefficient.

**Vulnerable Code:**

```
contract Airdrop is Ownable, BlockListed {
   using SafeMath for uint;
```

**PoC:**

1. Go to the line number and contracts mentioned above to see the redundant use of the "`SafeMath`" library.

**Impacts:**
`SafeMath` is generally not needed starting with Solidity `0.8`, since the compiler now has built-in overflow checking, making it redundant.

**Remediation:**
Remove the `SafeMath` library because Solidity `0.8.0+` automatically handles overflow and underflow checks.

**Retest:**

**Bug ID#5**

## Missing SPDX License

**Vulnerability Type**
Best practices

**Severity**
Informational

**Description:**
The contract "`Airdrop.sol`" was missing an SPDX License identifier in the source code. A smart contract whose source code is available can better establish trust. In order to minimize legal problems relating to copyright, Solidity encourages the use of machine-readable SPDX license identifiers.

**Vulnerable Code:**
https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol

```
pragma solidity ^0.8.2;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/utils/math/Math.sol";
import "@openzeppelin/contracts/utils/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "./BlockListed.sol";
```

**PoC:**

1. Open the file "`Airdrop.sol`" and note that the file is missing an SPDX License.

**Impacts:**
SPDX Licenses help in identifying the legal owner of the software, therefore, helping in issues like copyright infringement.

**Remediation:**

Every source file should start with a comment indicating its license. Add a necessary license identifier in the contract code like the one shown below:

```
// SPDX-License-Identifier: MIT
```

**Retest:**

**Bug ID#6**

## Unindexed Events

**Vulnerability Type**
Misconfiguration

**Severity**
Low

**Description:**
Events in solidity contain two kinds of parameters - indexed and non-indexed. These indexes are also known as "topics" and are the searchable parameters used in events. In the Ethereum system, events must be easily searched for, so that applications can filter and display historical events without undue overhead.
It was noticed that none of the event parameters were indexed in the contract "`Airdrop.sol`". This makes searching for past events difficult.

**Vulnerable Code:**
https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L17

```
contract Airdrop is Ownable, BlockListed {
    using SafeMath for uint;

    address public tokenAddr;

    ERC20 public Token;

    event EtherTransfer(address beneficiary, uint amount);
...
```

**PoC:**

1. Go to the contract "`Airdrop.sol`" and note the unindexed event on Line 17.

**Impacts:**
This does not impact the security aspect of the Smart contract but affects the ease of use when searching for past events.

**Remediation:**
It should be noted that indexed event parameters take up more gas than non-indexed

ones. Keeping that in mind, the contract should add indexed keywords to the searchable parameters to make searching efficient using an event filter.
You can add an attribute indexed to up to three parameters. Then, they appear in the structure of topics, not the data portion of the log.

**Retest:**

**Bug ID#7**

## Use of Large Number Literals

**Vulnerability Type**
Missing best practices.

**Severity**
Low

**Description:**
Integer literals are formed from a sequence of digits in the range 0-9. They are interpreted as decimals. The use of very large numbers with too many digits was detected in the code that could have been optimized using a different notation also supported by Solidity.

**Vulnerable Code:**

- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L24
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L27
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L34
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L37
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L40
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L43
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L46
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L49
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L52
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L55
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L58

- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L61
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L64
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L67

```
    uint256 public _totalSupply = 1000000000;

    //Max Transfer
    uint256 public MAX_TRANSFER = 10000000;

    //Cooling Period
    uint public cooling_period = 12 hours;

    // Token Allocation
    string public AirdropRate = "3%";
    uint256 public AirdropToken = 30000000;

    string public seadSaleRate = "16%";
    uint256 public seadSaleToken = 160000000;

    string public publicSaleRate = "4%";
    uint256 public publicSaleToken = 40000000;

    string public operationRate = "6%";
    uint256 public operationToken = 60000000;

    string public developmentRate = "10%";
    uint256 public developmentToken = 100000000;

    string public marketingRate = "10%";
    uint256 public marketingToken = 100000000;

    string public teamRate = "10%";
    uint256 public teamToken = 100000000;

    string public advisorRate = "5%";
    uint256 public advisorToken = 50000000;

    string public communityRate = "8%";
    uint256 public communityToken = 80000000;
```

```
string public reserveRate = "7%";
uint256 public reserveToken = 70000000;

string public stakingRate = "6%";
uint256 public stakingToken = 60000000;

string public liquidityRate = "15%";
uint256 public liquidityToken = 150000000;
```

**PoC:**

1. Go to the contract "`MasterContract.sol`" and note the large number literals from lines 24 to 67.

**Impacts:**

Literals with many digits are difficult to read and review. This may also introduce errors in the future if one of the zeroes is omitted while doing code modifications.

**Remediation:**

Scientific notation in the form of `2e10` is also supported, where the mantissa can be fractional but the exponent has to be an integer. The literal `MeE` is equivalent to `M * 10**E`. Examples include `2e10`, `2e10`, `2e-10`, `2.5e1`.

**Reference:**

https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals

**Retest:**

**Bug ID#8**

## Missing Constant Attribute in Variables

**Vulnerability Type**
Gas Optimization

**Severity**
Low

**Description:**
State variables can be declared as constant or immutable. In both cases, the variables cannot be modified after the contract has been constructed. For constant variables, the value has to be fixed at compile time.
The compiler does not reserve a storage slot for these variables, and every occurrence is replaced by the respective value.
Compared to regular state variables, the gas costs of constant and immutable variables are much lower since no `SLOAD` is executed to retrieve constants from storage because they're interpolated directly into the bytecode.

**Vulnerable Code:**

| Function Name | Vulnerable Code |
|---|---|
| AirdropRate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L33 |
| `AirdropToken` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L34 |
| `MAX_TRANSFER` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L27 |
| `_decimals` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L22 |
| `_name` | https://github.com/web3dev3/bck3-za2f/ |

| | |
|---|---|
| | blob/main/contracts/MasterContract.sol#L20 |
| _symbol | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L21 |
| _totalSupply | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L24 |
| advisorRate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L54 |
| advisorToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L55 |
| communityRate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L57 |
| communityToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L58 |
| cooling_period | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L30 |
| developmentRate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L45 |
| developmentToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L46 |
| liquidityRate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L66 |
| liquidityToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L67 |
| marketingRate | https://github.com/web3dev3/bck3-za2f/ |

| | |
|---|---|
| | blob/main/contracts/MasterContract.sol#L48 |
| marketingToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L49 |
| operationRate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L42 |
| operationToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L43 |
| publicSaleRate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L39 |
| publicSaleToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L40 |
| reserveRate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L60 |
| reserveToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L61 |
| seadSaleRate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L36 |
| seadSaleToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L37 |
| stakingRate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L63 |
| stakingToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L64 |
| teamRate | https://github.com/web3dev3/bck3-za2f/ |

| | |
|---|---|
| | blob/main/contracts/MasterContract.sol#L51 |
| teamToken | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L52 |

```
string public _name = "CalibToken";
  string public _symbol = "CALIB";
  uint8 public _decimals = 18;
  uint8 public _taxFee = 0;
  uint256 public _totalSupply = 1000000000;

  //Max Transfer
  uint256 public MAX_TRANSFER = 10000000;

  //Cooling Period
  uint public cooling_period = 12 hours;

  // Token Allocation
  string public AirdropRate = "3%";
  uint256 public AirdropToken = 30000000;

  string public seadSaleRate = "16%";
  uint256 public seadSaleToken = 160000000;

  string public publicSaleRate = "4%";
  uint256 public publicSaleToken = 40000000;

  string public operationRate = "6%";
  uint256 public operationToken = 60000000;

  string public developmentRate = "10%";
  uint256 public developmentToken = 100000000;

  string public marketingRate = "10%";
  uint256 public marketingToken = 100000000;

  string public teamRate = "10%";
  uint256 public teamToken = 100000000;
```

```
string public advisorRate = "5%";
uint256 public advisorToken = 50000000;

string public communityRate = "8%";
uint256 public communityToken = 80000000;

string public reserveRate = "7%";
uint256 public reserveToken = 70000000;

string public stakingRate = "6%";
uint256 public stakingToken = 60000000;

string public liquidityRate = "15%";
uint256 public liquidityToken = 150000000;
```

**PoC:**

1. Go to the contract "`MasterContract.sol`" and note the large number literals from lines 20 to 67.

**Impacts:**

Gas usage is increased if the variables that should be constants are not set as constants.

**Remediation:**

A "`constant`" attribute should be added in the parameters that never change to save the gas.

**Retest:**

**Bug ID#9**

## Missing Zero-Address Validations

**Vulnerability Type**
Input validation

**Severity**
Medium

**Description:**
Every critical address change, sensitive functions handling addresses and tokens, and all the other address type parameters should include a zero-address check otherwise contract functionality may become inaccessible or tokens burned forever if an invalid or a zero address is supplied by mistake or due to contract errors.

**Vulnerable Code:**

```
    constructor(address _tokenAddr) public {
        Token = ERC20(_tokenAddr);
    }
...

    function updateTokenAddress(address newTokenAddr) public
onlyOwner {
        tokenAddr = newTokenAddr;
    }

    function withdrawTokens(address beneficiary) public onlyOwner
{
        require(Token.transfer(beneficiary,
Token.balanceOf(address(this))));
    }

    function withdrawEther(address payable beneficiary) public
onlyOwner {
        beneficiary.transfer(address(this).balance);
    }
}
...
```

```
    function _deliverTokens(address beneficiary, uint256
tokenAmount)
        internal
        virtual
        override
    {
        token().safeTransferFrom(_tokenWallet, beneficiary,
tokenAmount);
    }

...
```

**PoC:**

1. The following functions are missing a zero-address check in their address type input parameters -

| Function Name | Function Parameter | Vulnerable Code |
|---|---|---|
| `_tokenAddr` | `_tokenAddr` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L19-L21 |
| `updateTokenAddress` | `newTokenAddr` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L57-L60 |
| `withdrawTokens` | `beneficiary` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L61-L63 |
| `withdrawEther` | `beneficiary` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L65-L67 |
| `_deliverTokens` | `beneficiary` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/AllowanceCrowdsale.sol#L55-L61 |
| `createVestingSchedule` | `_beneficiary` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L149 |

**Impacts:**

Missing a zero-address validation on sensitive address changes and token transfers may cause contract ownership to be los, and the tokens may be burned forever.

**Remediation:**

Add a zero-address validation to the functions specified above.

**Retest:**

**Bug ID#10**

## Dead/Unreachable Code

**Vulnerability Type**
Code With No Effects - SWC-135

**Severity**
Informational

**Description:**
It is recommended to keep the production repository clean to prevent confusion and the introduction of vulnerabilities. The functions and parameters that are never used or called externally or from inside the contracts should be removed when the contract is deployed on the mainnet.

**Vulnerable Code:**
https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TimedCrowdsale.sol#105-115

```
    function _extendTime(uint256 newClosingTime) internal {
        require(!hasClosed(), "TimedCrowdsale: already closed");
        // solhint-disable-next-line max-line-length
        require(
            newClosingTime > _closingTime,
            "TimedCrowdsale: new closing time is before current
closing time"
        );

        emit TimedCrowdsaleExtended(_closingTime,
newClosingTime);
        _closingTime = newClosingTime;
    }
```

**PoC:**

1. Go to the contract "`TimedCrowdsale.sol`" and note the function "`_extendTime`" is an internal function.
2. This function is not called throughout the code in any of the contracts and therefore should be removed.

**Impacts:**
This does not impact the security aspect of the Smart contract but prevents confusion when the code is sent to other developers or auditors to understand and implement. This reduces the overall size of the contracts and also helps in saving gas.

**Remediation:**
Delete the internal function "`_extendTime`" if it is not supposed to be used.

**Retest:**

**Bug ID#11**

## Missing Input Validation on Amount

**Vulnerability Type**
Input Validation

**Severity**
Low

**Description:**
Input validation is a frequently-used technique for checking potentially dangerous inputs in order to ensure that the inputs are safe for processing within the code, or when communicating with other components.
When the smart contract does not validate the inputs properly, it may introduce a range of vulnerabilities.
The contracts "`AllowanceCrowdsale.sol`" and "`TokenVesting.sol`" were missing input validation on the amount parameters in various functions.

**Vulnerable Code:**
https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TimedCrowdsale.sol#105-115

```
    function withdraw(uint256 amount)
        public
        nonReentrant
        onlyOwner{
        require(this.getWithdrawableAmount() >= amount,
"TokenVesting: not enough withdrawable funds");
        _token.safeTransfer(owner(), amount);
    }

...

    function release(
        bytes32 vestingScheduleId,
        uint256 amount
    )
        public
        nonReentrant
```

```
        onlyIfVestingScheduleNotRevoked(vestingScheduleId){
        VestingSchedule storage vestingSchedule =
vestingSchedules[vestingScheduleId];
        bool isBeneficiary = msg.sender ==
vestingSchedule.beneficiary;
        bool isOwner = msg.sender == owner();
        require(
            isBeneficiary || isOwner,
            "TokenVesting: only beneficiary and owner can release
vested tokens"
        );
        uint256 vestedAmount =
_computeReleasableAmount(vestingSchedule);
        require(vestedAmount >= amount, "TokenVesting: cannot
release tokens, not enough vested tokens");
        vestingSchedule.released =
vestingSchedule.released.add(amount);
        address payable beneficiaryPayable =
payable(vestingSchedule.beneficiary);
        vestingSchedulesTotalAmount =
vestingSchedulesTotalAmount.sub(amount);
        _token.safeTransfer(beneficiaryPayable, amount);
    }

...

    function _deliverTokens(address beneficiary, uint256
tokenAmount)
        internal
        virtual
        override
    {
        token().safeTransferFrom(_tokenWallet, beneficiary,
tokenAmount);
    }
```

**PoC:**

1. Go to the contracts and line numbers mentioned below and note that the amount parameters don't have an input validation and can even take 0 as an input.

| Function Name | Function Parameter | Vulnerable Code |
|---|---|---|
| `_deliverTokens` | `tokenAmount` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/AllowanceCrowdsale.sol#L55-L61 |
| `withdraw` | `amount` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L209-L215 |
| `release` | `amount` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L222-L242 |

**Impacts:**
Accepting amount as 0 in a transaction might cause the transaction to fail or it'll cause useless consumption of gas.

**Remediation:**
Have an input validation on the functions and parameters mentioned above to check that the amount accepted is more than zero and is within the necessary limits.

**Retest:**

**Bug ID#12**

# Missing Input Validation on Vesting Start Time

**Vulnerability Type**
Input validation

**Severity**
Medium

**Description:**
Input validation is a frequently-used technique for checking potentially dangerous inputs in order to ensure that the inputs are safe for processing within the code, or when communicating with other components.
When the smart contract does not validate the inputs properly, it may introduce a range of vulnerabilities.
The contracts "`TokenVesting.sol`" was missing input validation in the function "`createVestingSchedule`" in which the start time of the vesting period - "`_start`", has no validation and can be set in the past.

**Vulnerable Code:**
https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L150

```
    function createVestingSchedule(
        address _beneficiary,
        uint256 _start,
        uint256 _cliff,
        uint256 _duration,
        uint256 _slicePeriodSeconds,
        bool _revocable,
        uint256 _amount
    )
        public
        onlyOwner{
        require(
            this.getWithdrawableAmount() >= _amount,
            "TokenVesting: cannot create vesting schedule because
not sufficient tokens"
        );
        require(_duration > 0, "TokenVesting: duration must be >
```

```
0");
        require(_amount > 0, "TokenVesting: amount must be > 0");
        require(_slicePeriodSeconds >= 1, "TokenVesting:
slicePeriodSeconds must be >= 1");
        bytes32 vestingScheduleId =
this.computeNextVestingScheduleIdForHolder(_beneficiary);
        uint256 cliff = _start.add(_cliff);
        vestingSchedules[vestingScheduleId] = VestingSchedule(
            true,
            _beneficiary,
            cliff,
            _start,
            _duration,
            _slicePeriodSeconds,
            _revocable,
            _amount,
            0,
            false
        );
        vestingSchedulesTotalAmount =
vestingSchedulesTotalAmount.add(_amount);
        vestingSchedulesIds.push(vestingScheduleId);
        uint256 currentVestingCount =
holdersVestingCount[_beneficiary];
        holdersVestingCount[_beneficiary] =
currentVestingCount.add(1);
    }
```

**PoC:**

1. Go to the contract "TokenVesting.sol" and note that the function accepts the parameter "`_start`" on line number 150.
2. This is used again on Line 172 without any validation on the timestamp.

**Impacts:**

This issue allows a malicious contract owner to manipulate the release of tokens in the beneficiary's account or maybe by mistake if there is an error in epoch value leading to past dates of vesting period.

**Remediation:**

Use a `require()` validation in the function that validates if the value of the `_start` parameter is not too far in the past.

**Retest:**

**Bug ID#13**

## Functions should be declared External

**Vulnerability Type**
Gas Optimization

**Severity**
Low

**Description:**
A function with `public` visibility modifier was detected that is not called internally.

`public` and `external` differs in terms of gas usage. The former use more than the latter when used with large arrays of data. This is due to the fact that Solidity copies arguments to memory on a `public` function while `external` read from calldata which a cheaper than memory allocation.
Multiple public functions were found throughout the contracts which were never called from inside the contracts. These should be changed to `external`.

**Vulnerable Code:**

| Function Name | Vulnerable Code |
|---|---|
| **dropTokens** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L23 |
| **dropEther** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L34 |
| **updateTokenAddress** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L57 |
| **withdrawTokens** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L61 |
| **withdrawEther** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Airdrop.sol#L65 |
| tokenWallet | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/AllowanceCrowdsale.sol#L34 |
| remainingTokens | https://github.com/web3dev3/bck3-za2f/ |

| | |
|---|---|
| | blob/main/contracts/AllowanceCrowdsale.sol#L42 |
| **blackList** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/BlockListed.sol#L7 |
| **removeFromBlacklist** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/BlockListed.sol#L12 |
| wallet | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Crowdsale.sol#L100 |
| rate | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Crowdsale.sol#L107 |
| **mint** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L93 |
| **setTaxFee** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L107 |
| **allocateTokens** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/MasterContract.sol#L112 |
| openingTime | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TimedCrowdsale.sol#L57 |
| closingTime | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TimedCrowdsale.sol#L64 |
| **createVestingSchedule** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L148 |
| **revoke** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L190 |
| **withdraw** | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L209 |
| computeReleasableAmount | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L2 |

| | |
|---|---|
| | 59 |
| `getLastVestingScheduleForHolder` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L303 |

**PoC:**

1. Check the functions mentioned in the table above.
2. Search those functions in the contracts and note that they are not called from within any of the contracts and therefore should be marked as external.

**Impacts:**

Making `public` functions as `external` if not used inside the contracts will save gas because `external` functions use lesser gas than `public` functions.

**Remediation:**

If you know the function you create only allows for `external` calls, use the `external` visibility modifier instead of `public`. It provides performance benefits and you will save on gas.

**Retest:**

**Bug ID#14**

# Missing Access Control in Blacklisting Users

**Vulnerability Type**
Improper access control

**Severity**
Critical

**Description:**
The contract "`BlockListed.sol`" has functions to add a user to the blacklist and remove them from the blacklist. These functions do not have any access control validations on them and they can be called by any external malicious user.
This will allow them to add or remove any addresses from the blacklist.

**Vulnerable Code:**
https://github.com/web3dev3/bck3-za2f/blob/main/contracts/BlockListed.sol#L7-L10
https://github.com/web3dev3/bck3-za2f/blob/main/contracts/BlockListed.sol#L12-L15

```
    function blackList(address _user) public {
        require(!isBlacklisted[_user], "user already
blacklisted");
        isBlacklisted[_user] = true;
    }

    function removeFromBlacklist(address _user) public {
        require(isBlacklisted[_user], "user already
whitelisted");
        isBlacklisted[_user] = false;
    }
```

**PoC:**

1. Go to the contract "`BlockList.sol`" and note that the functions are public which means they can be called externally as well.
2. They also lack any access control validation like `onlyOwner`.

**Impacts:**
This is a critical vulnerability allowing any user to add or remove any user's address from the blacklist in the contracts.

**Remediation:**
Add an access control validation on the critical functions such that they can only be called by owners or administrators. Modifiers can be used for this.

**Retest:**

**Bug ID#15**

## Critical Functions Should Validate if Contract is Pasued

**Vulnerability Type**
Improper access control

**Severity**
High

**Description:**

Openzeppelin's `Pausable` Library is used as a modifier to check if a contract is paused or not. This is typically used in contracts to protect sensitive functions in case there's malicious activity going on or if the contract is compromised.

The contracts "`TokenVesting.sol`", and "`Crowdsale.sol`" was found to be missing the "`whenNotPaused`" modifier on several sensitive functions mentioned below.

**Vulnerable Code:**

| Function Name | Vulnerable Code |
|---|---|
| `release` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L222-L242 |
| `buyTokens` | https://github.com/web3dev3/bck3-za2f/blob/main/contracts/Crowdsale.sol#L124-L141 |

```
function buyTokens(address beneficiary) public payable
nonReentrant {...}

...

function release(
        bytes32 vestingScheduleId,
        uint256 amount
    )
```

```
        public
        nonReentrant
        onlyIfVestingScheduleNotRevoked(vestingScheduleId){...}
```

**PoC:**

1. Go to the contracts mentioned above and note that the functions "`release`" and "`buyTokens`" do not have the "`whenNotPaused`" modifiers set.

**Impacts:**

Missing the "`whenNotPaused`" modifier on these functions may prove fatal if the contract is compromised and the owners need to pause everything. If the `whenNotPaused` modifier is missing from a vulnerable function, the attackers may call that function without restrictions.

**Remediation:**

It is recommended to implement the `whenNotPaused` modifier on all the sensitive functions that deal with Ether or tokens or sensitive access roles and their modifications.

**Retest:**

**Bug ID#16**

## Unnecessary Code

**Vulnerability Type**
Code optimization

**Severity**
Informational

**Description:**
The contract "`TokenVesting.sol`" has a function "`getCurrentTime()`" to get the time using "`block.timestamp`".
This is only used at one place inside the function "`_computeReleasableAmount`" on Line 328.
Since this function is only used at one place, it is recommended to directly call "`block.timestamp`" rather than define a function for it.

**Vulnerable Code:**

- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L328
- https://github.com/web3dev3/bck3-za2f/blob/main/contracts/TokenVesting.sol#L344-L352

```
function getCurrentTime()
    internal
    virtual
    view
    returns(uint256){
    return block.timestamp;
}
```

**PoC:**

1. Go to the contract "`TokenVesting.sol`" and note the function is defined on Line 344.
2. This is then called on Line 328.

**Impacts:**

Defining a complete function for just a call "`block.timestamp`" which is only used at one other place is not advisable as this will take up contract space and will also use more gas.

**Remediation:**
If the function "`getCurrentTime`" is not supposed to be used at many places, remove the function and use the direct call to "`block.timestamp`" instead.

**Retest:**

**Bug ID#17**

## Missing Access Control in Blacklisting Users

**Vulnerability Type**
Access Control

**Severity**
High

**Description:**
The contract “BlockListed.sol” has functions to add a user to blacklist and remove them from the blacklist. These functions do not have any access control validations on them and they can be called by any external malicious user.
This will allow them to add or remove any addresses from the blacklist.

**Vulnerable Code:**
https://github.com/web3dev3/bck3-za2f/blob/main/contracts/BlockListed.sol#L7-L10
https://github.com/web3dev3/bck3-za2f/blob/main/contracts/BlockListed.sol#L12-L15

```
    function blackList(address _user) public {
        require(!isBlacklisted[_user], "user already
blacklisted");
        isBlacklisted[_user] = true;
    }

    function removeFromBlacklist(address _user) public {
        require(isBlacklisted[_user], "user already
whitelisted");
        isBlacklisted[_user] = false;
    }
```

**PoC:**

3. Go to the contract “BlockList.sol” and note that the functions are public which means they can be called externally as well.
4. They also lack any access control validation like onlyOwner.

**Impacts:**
This is a critical vulnerability allowing any user to add or remove any user’s address from the blacklist in the contracts.

**Remediation:**
Add an access control validation on the critical functions such that they can only be called by owners or administrators. Modifiers can be used for this.

**Retest:**