



Smart contracts security assessment

Final report

[Tariff: Standard](#)

Pulse Drip Launch Pad

October 2024



0xguard.com



hello@0xguard.com

Contents

1. Introduction	3
2. Contracts checked	4
3. Procedure	5
4. Known vulnerabilities checked	5
5. Classification of issue severity	6
6. Issues	7
7. Conclusion	23
8. Disclaimer	24

Introduction

The report has been prepared for **Pulse Drip Launch Pad**.

The audited project consists of:

- 3 ERC20 tokens: fixed supply token and 2 tokens with taxable transfers, one of which burns collected fees, and other adds them to token's DEX pair;
- factory contract to deploy cloned versions of abovementioned tokens;
- locker and vesting contract;
- airdrop contract for an arbitrary ERC20 tokens;
- unified farming contract that allows users to stake fixed token for multiple types of rewards;
- presale contract for ERC20 tokens;
- ERC20 token receiver contract to collect fees and swap them for H2O tokens;
- governance contract that allows owner to set multiple managers with separate modifier for restricted functions.

All project contracts are upgradable.

The md5 sums of the files under investigation:

3c4c9e1c6026cf01be8d1c0d6cdcf41c - Airdrop.sol

d6055da2d472f633a568bc3830c97752 - ERC20_Burn_Tax.sol

dda627bfe892f6c24c2c36426ccd1255- ERC20_Liquidity_Tax.sol

be6d489d16ea7dfa3707376a10af3bdb - ERC20_Simple.sol

6b042bee61d3d5fae110b8322274ebf6 - Factory.sol

346fe33cc9e6ea5a557f4bc26c492645 - Farms.sol

69967fe47fcab0e1417e7a9263f09d26 - FeeHandler.sol

c301ef42d3002eada54cc1e79a11c519 - Locker.sol

8465406f98639d79e9eb4afc7ce840da - ManageableUpgradeable.sol

c0fde9e6c46de36075b38a0a53e7092b - PresaleManager.sol

Name	Pulse Drip Launch Pad
Audit date	2024-10-03 - 2024-10-09
Language	Solidity
Platform	Pulse Chain

Contracts checked

Name	Address
------	---------

ERC20_Simple.sol

ERC20_Burn_Tax.sol

ERC20_Liquidity_Tax.sol

Factory.sol

Farms.sol

Locker.sol

Airdrop.sol

FeeHandler.sol

ManageableUpgradeable.sol

PresaleManager.sol

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

<u>Incorrect Constructor Name</u>	passed
<u>Block values as a proxy for time</u>	passed
<u>Authorization through tx.origin</u>	passed
<u>DoS with Failed Call</u>	passed
<u>Delegatecall to Untrusted Callee</u>	passed
<u>Use of Deprecated Solidity Functions</u>	passed
<u>Assert Violation</u>	passed
<u>State Variable Default Visibility</u>	passed
<u>Reentrancy</u>	passed
<u>Unprotected SELFDESTRUCT Instruction</u>	passed
<u>Unprotected Ether Withdrawal</u>	passed
<u>Unchecked Call Return Value</u>	passed
<u>Floating Pragma</u>	passed
<u>Outdated Compiler Version</u>	passed
<u>Integer Overflow and Underflow</u>	passed
<u>Function Default Visibility</u>	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. Restriction of direct transfers (ERC20_Burn_Tax.sol)

Status: Open

Transfers by the **transfer** function are prohibited unless the sender is excluded from tax. At the same time, transfers via the **transferFrom** function are allowed, but requires additional transaction for approval.

```
function transfer(
    address _to,
    uint256 _value
) public override returns (bool success) {
    if (isExcludedFromTax[_msgSender()]) {
        _transfer(_msgSender(), _to, _value);
        return true;
    }
    return true;
}
```

Recommendation: Consider allowing direct transfers.

2. Sell and transfer taxes can be avoided (ERC20_Burn_Tax.sol)

Status: Open

Buy tax is applied first if **from** address has **token0** or **token1** methods (returning token address), other taxes can be evaded by using special contract with view methods of Uniswap pair. If the buy tax is lower than sell and transfer taxes, then this vulnerability can be used for trading optimizations.

```
function _transfer(address _from, address _to, uint256 _value) private {
```

```
. . .
if (!isExcludedFromTax[_from] && !isExcludedFromTax[_to]) {
    if (isUniswapV2Pair(_from)) {
        taxAmount = (_value * taxRates.buy) / TAX_DIVISOR;
    } else if (isUniswapV2Pair(_to)) {
        taxAmount = (_value * taxRates.sell) / TAX_DIVISOR;
    } else {
        taxAmount = (_value * taxRates.transfer) / TAX_DIVISOR;
    }
    . . .
}

function isUniswapV2Pair(address target) public view returns (bool) {
    if (target.code.length == 0) {
        return false;
    }

    IUniswapV2Pair pairContract = IUniswapV2Pair(target);

    address token0;
    address token1;

    try pairContract.token0() returns (address _token0) {
        token0 = _token0;
    } catch (bytes memory) {
        return false;
    }

    try pairContract.token1() returns (address _token1) {
        token1 = _token1;
    } catch (bytes memory) {
        return false;
    }

    return token0 == address(this) || token1 == address(this);
}
```

Recommendation: Consider using list of known pairs or list of Uniswap factories to avoid detecting trading pair relying on data provided by the user.

3. Sell and transfer taxes can be avoided (ERC20_Liquidity_Tax.sol)

Status: Open

Buy tax is applied first if **from** address has **token0** or **token1** methods (returning token address), other taxes can be evaded by using special contract with view methods of Uniswap pair. If the buy tax is lower than sell and transfer taxes, then this vulnerability can be used for trading optimizations.

```
function _transfer(address _from, address _to, uint256 _value) private {
    . . .
    bool isBuying = isUniswapV2Pair(_from);
    if (!isExcludedFromTax[_from] && !isExcludedFromTax[_to]) {
        if (isBuying) {
            taxAmount = (_value * taxRates.buy) / TAX_DIVISOR;
        } else if (isUniswapV2Pair(_to)) {
            taxAmount = (_value * taxRates.sell) / TAX_DIVISOR;
        } else {
            taxAmount = (_value * taxRates.transfer) / TAX_DIVISOR;
        }
    }
    . . .
}

function isUniswapV2Pair(address target) public view returns (bool) {
    if (target.code.length == 0) {
        return false;
    }

    IUniswapV2Pair pairContract = IUniswapV2Pair(target);

    address token0;
    address token1;

    try pairContract.token0() returns (address _token0) {
        token0 = _token0;
    } catch (bytes memory) {
        return false;
    }

    try pairContract.token1() returns (address _token1) {
        token1 = _token1;
    } catch (bytes memory) {
```

```
        return false;
    }

    return token0 == address(this) || token1 == address(this);
}
```

Recommendation: Consider using list of known pairs or list of Uniswap factories to avoid detecting trading pair relying on data provided by the user.

4. Not all ERC-20 tokens are supported (Locker.sol)

Status: Open

Locked token is an arbitrary address provided by user. Rebasing tokens or tokens with taxable transfers can cause locked funds problem (if contract's balance has increased during the locking period) or lack of liquidity problem (if the balance has decreased or lock creation transaction was subjected to transfer tax).

Recommendation: Use whitelist for supported tokens and check actual transferred amounts, or store locked tokens in individual contracts.

5. Not all ERC-20 tokens are supported (PresaleManager.sol)

Status: Open

Sale **baseToken** is an arbitrary address provided by user. Rebasing tokens or tokens with taxable transfers can cause locked funds problem (if contract's balance has increased during the token holding period) or lack of liquidity problem (if the balance has decreased or initial **transferFrom** was subjected to transfer tax).

Recommendation: Use whitelist for supported tokens and check actual transferred amounts, or store sale tokens in individual contracts.

6. Incorrect calculations of presale amount (PresaleManager.sol)

Status: Open

Required for presale token amount is calculated in the **calculateTokensForPresale** function. The **hardCap** amount (in quote tokens) is multiplied by **presaleRate**, resulting amount should be nominated in base tokens, but then it is divided by base token decimals factor. The post-sale claiming

on the other hand, uses traditional calculation: user's contribution (in quote tokens) is multiplied by **presaleRate** and divided by constant precision (meaning that the presale rate should be presented with that precision).

```
function calculateTokensForPresale(
    PresaleParams memory params
) public view returns (uint256) {
    uint256 amountForIdo = (params.presaleRate * params.hardCap) /
        10 ** IERC20(params.baseToken).decimals();
    uint256 amountForLiquidity = (params.listingRate *
        params.hardCap *
        params.liquidityPercentage) /
        (PRECISION * 10 ** IERC20(params.baseToken).decimals());
    uint256 amountForBalancing = (params.presaleRate *
        params.hardCap *
        FEE) / (PRECISION * 10 ** IERC20(params.baseToken).decimals());

    return amountForIdo + amountForLiquidity + amountForBalancing;
}

function claimTokens(uint256 presaleId) external nonReentrant notPaused {
    Presale storage presale = presales[presaleId];
    require(
        presale.status == PresaleStatus.COMPLETED,
        "CLAIM_TOKENS: Presale is completed"
    );

    uint256 contribution = presaleIdAddressToContribution[presaleId][
        _msgSender()
    ];
    require(contribution > 0, "CLAIM_TOKENS: No contribution found");

    presaleIdAddressToContribution[presaleId][_msgSender()] = 0;

    IERC20Upgradeable(presale.baseToken).transfer(
        _msgSender(),
        (contribution * presale.presaleRate) / PRECISION
    );
}
```

Recommendation: Fix and test calculations.

7. Incorrect parameters for liquidity adding (PresaleManager.sol)

Status: Open

The sale finalization in the function `completePresale` requires interaction with the external Uniswap router contract. To perform `IUniswapV2Router02.addLiquidity` call, an approval must be made, but the approval amount is different than the `addLiquidity` parameters.

The `to` parameter of `addLiquidity` call is set to `address(this)`, meaning that minted LP tokens are locked in the PresaleManager contract.

The deadline parameter is set to `block.timestamp + presale.liquidityLockupTime`, but any value greater than `block.timestamp` is ignored and treated as the `block.timestamp`.

```
function completePresale(
    uint256 presaleId
) external nonReentrant notPaused {
    . . .
    uint256 liquidityQuoteAmount = (presale.raisedQuoteAmount *
        presale.liquidityPercentage) / PRECISION;
    uint256 liquidityBaseAmount = (presale.listingRate *
        liquidityQuoteAmount) / PRECISION;

    IERC20Upgradeable(presale.quoteToken).approve(
        presale.router,
        presale.tokensForPresale
    );

    IERC20Upgradeable(presale.baseToken).approve(
        presale.router,
        presale.tokensForPresale
    );

    IUniswapV2Router02(presale.router).addLiquidity(
        presale.baseToken,
        presale.quoteToken,
        liquidityBaseAmount,
```

```

        liquidityQuoteAmount,
        0,
        0,
        address(this),
        block.timestamp + presale.liquidityLockupTime
    );
    . . .
}

```

Recommendation: Fix parameters, increase testing coverage.

8. Incorrect parameters for token lock (PresaleManager.sol)

Status: Open

The function `completePresale` locks sale tokens that are meant to be claimed by users. The `LOCKER.lock` call is made without beforehand approval that is mandatory for successful external call that uses `transferFrom` method.

```

function completePresale(
    uint256 presaleId
) external nonReentrant notPaused {
    . . .
    LOCKER.lock(
        presale.baseToken,
        presale.tokensForPresale,
        block.timestamp + presale.liquidityLockupTime,
        presaleIdToOwner[presaleId]
    );
    . . .
}

```

Recommendation: Fix the logic, increase the testing coverage.

9. No tests provided (All audited contracts)

Status: Open

For the project, there were no testing offered. We advise covering the core business logic scenarios with integration tests with adequate coverage until we can no longer ensure the project works in line

with the documentation, as the audit uncovered several high severity issues and erroneous operation of many methods.

Medium severity issues

1. Standard violation (ERC20_Simple.sol)

Status: Open

Zero amount transfers are prohibited. This restriction contradicts the EIP-20 requirement "Transfers of 0 values MUST be treated as normal transfers and fire the **Transfer** event".

```
function _transfer(address _from, address _to, uint256 _value) private {
    require(
        _from != address(0),
        "TRANSFER: Transfer from the dead address"
    );
    require(_to != address(0), "TRANSFER: Transfer to the dead address");
    require(_value > 0, "TRANSFER: Invalid amount");
    require(balances[_from] >= _value, "TRANSFER: Insufficient balance");
    balances[_from] -= _value;
    balances[_to] += _value;
    emit Transfer(_from, _to, _value);
}
```

Recommendation: Consider allowing zero amount transfers.

2. Standard violation (ERC20_Burn_Tax.sol)

Status: Open

Zero amount transfers are prohibited. This restriction contradicts the EIP-20 requirement "Transfers of 0 values MUST be treated as normal transfers and fire the **Transfer** event".

```
function _transfer(address _from, address _to, uint256 _value) private {
    . . .
    require(
        _value > 0,
        "TRANSFER: Transfer amount must be greater than zero"
    );
}
```

```
);
    . . .
}
```

Recommendation: Consider allowing zero amount transfers.

3. Transfers without event (ERC20_Burn_Tax.sol)

Status: Open

Transfer of **toBurn** amount from **_from** address to **address(this)** is not accompanied with **Transfer** event. Such transfers directly violate ERC-20 token standard.

```
function _transfer(address _from, address _to, uint256 _value) private {
    . . .
    balances[owner()] += toDev;
    emit Transfer(_from, owner(), toDev);

    balances[address(this)] += toBurn;
    _burn(address(this), toBurn);

    balances[_from] -= _value;
    balances[_to] += tokensToTransfer;
    emit Transfer(_from, _to, tokensToTransfer);
}
```

Recommendation: Emit corresponding event to allow full token tracking with events.

4. Owner tax can reach 100% of all taxes (ERC20_Burn_Tax.sol)

Status: Open

The variable **devPercentage** is checked to be not greater than 20%, when set, but can reach 100% of tax amount.

```
function setDevPercentage(uint256 _devPercentage) public onlyOwner {
    require(
        _devPercentage <= MAX_TAX_RATE,
        "TAX: Dev percentage must be less than or equal to 20%"
    );
}
```

```

        devPercentage = _devPercentage;
    }

    function _transfer(address _from, address _to, uint256 _value) private {
        . . .
        uint256 toDev = (taxAmount * devPercentage) / MAX_TAX_RATE;
        uint256 toBurn = taxAmount - toDev;
        . . .
    }

```

Recommendation: Consider using `toDev=(taxAmount*devPercentage)/TAX_DIVISOR` calculation or fix the error message in the `setDevPercentage` and `setInitialDevPercentage` functions.

5. Owner tax can reach 100% of all taxes (ERC20_Liquidity_Tax.sol)

Status: Open

The variable `devPercentage` is checked to be not greater than 20%, when set, but can reach 100% of tax amount.

```

function setDevPercentage(uint256 _devPercentage) public onlyOwner {
    require(
        _devPercentage <= MAX_TAX_RATE,
        "TAX: Dev percentage must be less than or equal to 20%"
    );
    devPercentage = _devPercentage;
}

function _transfer(address _from, address _to, uint256 _value) private {
    . . .
    uint256 toDev = (taxAmount * devPercentage) / MAX_TAX_RATE;
    uint256 toBurn = taxAmount - toDev;
    . . .
}

```

Recommendation: Consider using `toDev=(taxAmount*devPercentage)/TAX_DIVISOR` calculation or fix the error message in the `setDevPercentage` and `setInitialDevPercentage`

functions.

6. Standard violation (ERC20_Liquidity_Tax.sol)

Status: Open

Zero amount transfers are prohibited. This restriction contradicts the EIP-20 requirement "Transfers of 0 values MUST be treated as normal transfers and fire the **Transfer** event".

```
function _transfer(address _from, address _to, uint256 _value) private {  
    . . .  
    require(  
        _value > 0,  
        "TRANSFER: Transfer amount must be greater than zero"  
    );  
    . . .  
}
```

Recommendation: Consider allowing zero amount transfers.

7. Adding liquidity with 100% slippage (Factory.sol)

Status: Open

Adding liquidity to Uniswap pair is made with 100% slippage (**amountAmin** and **amountBmin** parameters are not set). Both pair tokens are added with the price in the pair that is calculated at the moment of transaction mining. Any user with access to pair's tokens can create pair or manipulate its price before execution of **addLiquidity** function. Any unspent tokens remain at the Factory contract's balance.

```
function addLiquidity(  
    address _token,  
    address _pairToken,  
    address _router,  
    uint256 _tokenAmount,  
    uint256 _pairTokenAmount  
) public payable onlyAllowedRouter(_router) nonReentrant {  
    . . .  
}
```

```

        IUniswapV2Router02(_router).addLiquidity(
            _token,
            _pairToken,
            _tokenAmount,
            _pairTokenAmount,
            0,
            0,
            _msgSender(),
            block.timestamp
        );
    }

```

Recommendation: Add min amounts of pair tokens from the input parameters. Return any unspent tokens to the user.

8. Fee amount of sale tokens is transferred twice (PresaleManager.sol)

Status: Open

The sale tokens are subjected to fee when the sale is created, but this fee amount is transferred twice from the sale creator: the first amount is transferred to a special fee receiver address, and the second time it's transferred to the PresaleManager contract itself to then be, presumably, refunded after the end of the sale.

```

function createPresale(
    PresaleParams memory params
) external payable nonReentrant notPaused {
    require(msg.value > FIXED_FEE, "CREATE_PRESALE: Insufficient fee");
    (bool success, ) = FEE_RECEIVER.call{value: FIXED_FEE}("");
    require(success, "CREATE_PRESALE: Transfer failed");

    uint256 refund = msg.value - FIXED_FEE;
    if (refund > 0) {
        (success, ) = _msgSender().call{value: refund}("");
        require(success, "CREATE_PRESALE: Refund failed");
    }

    validatePresaleParams(params);
    uint256 tokensForPresale = calculateTokensForPresale(params);

```

```
uint256 fee = (tokensForPresale * FEE) / PRECISION;
uint256 tokensForPresaleWithFee = tokensForPresale + fee;

IERC20Upgradeable(params.baseToken).transferFrom(
    _msgSender(),
    address(this),
    tokensForPresaleWithFee
);

IERC20Upgradeable(params.baseToken).transfer(FEE_RECEIVER, fee);

createPresaleInternal(params, tokensForPresale);
}

function calculateTokensForPresale(
    PresaleParams memory params
) public view returns (uint256) {
    uint256 amountForIdo = (params.presaleRate * params.hardCap) /
        10 ** IERC20(params.baseToken).decimals();
    uint256 amountForLiquidity = (params.listingRate *
        params.hardCap *
        params.liquidityPercentage) /
        (PRECISION * 10 ** IERC20(params.baseToken).decimals());
    uint256 amountForBalancing = (params.presaleRate *
        params.hardCap *
        FEE) / (PRECISION * 10 ** IERC20(params.baseToken).decimals());

    return amountForIdo + amountForLiquidity + amountForBalancing;
}
```

Recommendation: Remove one of the fees or add it to documentation.

Low severity issues

1. Possible math underflow (ERC20_Simple.sol)

Status: Open

The function `transferFrom` can be reverted with underflow exception if spender's allowance is exceeded or not set at all.

```
function transferFrom(
    address _from,
    address _to,
    uint256 _value
) public override returns (bool success) {
    if (allowances[_from][_msgSender()] < type(uint256).max) {
        allowances[_from][_msgSender()] -= _value;
    }
    _transfer(_from, _to, _value);
    return true;
}
```

Recommendation: Add safety check with human-readable error message.

2. Contract deployer is excluded from taxes (ERC20_Burn_Tax.sol)

Status: Open

The deployer address is excluded from taxes, even though the deployer is supposed to be a factory.

```
function setInitialExclusions(address _owner) internal {
    isExcludedFromTax[address(this)] = true;
    isExcludedFromTax[_msgSender()] = true;
    isExcludedFromTax[_owner] = true;
}
```

Recommendation: Consider removing the deployer exclusion.

3. Possible math underflow (ERC20_Burn_Tax.sol)

Status: Open

The function `transferFrom` can be reverted with underflow exception if spender's allowance is exceeded or not set at all.

```
function transferFrom(
    address _from,
    address _to,
    uint256 _value
) public override returns (bool success) {
    if (allowances[_from][_msgSender()] < type(uint256).max) {
        allowances[_from][_msgSender()] -= _value;
    }
    _transfer(_from, _to, _value);
    return true;
}
```

Recommendation: Add safety check with human-readable error message.

4. Contract deployer is excluded from taxes (ERC20_Liquidity_Tax.sol)

Status: Open

The deployer address is excluded from taxes, even though the deployer is supposed to be a factory.

```
function setInitialExclusions(address _owner, address _router) internal {
    isExcludedFromTax[address(this)] = true;
    isExcludedFromTax[_msgSender()] = true;
    isExcludedFromTax[_owner] = true;
    isExcludedFromTax[_router] = true;
}
```

Recommendation: Consider removing the deployer exclusion.

5. Possible math underflow (ERC20_Liquidity_Tax.sol)

Status: Open

The function `transferFrom` can be reverted with underflow exception if spender's allowance is exceeded or not set at all.

```
function transferFrom(
    address _from,
    address _to,
    uint256 _value
) public override returns (bool success) {
    if (allowances[_from][_msgSender()] < type(uint256).max) {
        allowances[_from][_msgSender()] -= _value;
    }
    _transfer(_from, _to, _value);
    return true;
}
```

Recommendation: Add safety check with human-readable error message.

6. Inefficient search (Farms.sol)

Status: Open

Search over all farms' reward tokens is ineffective due to **farmInfo** array can only increase over time.

Recommendation: Use a mapping for active reward tokens.

7. Incorrect event parameter (Airdrop.sol)

Status: Open

The event **TokensAirdroppedTokensAirdropped** in the **sharedAirdrop** function is emitted with zeroed array of amounts.

Conclusion

Pulse Drip Launch Pad ERC20_Simple.sol, ERC20_Burn_Tax.sol, ERC20_Liquidity_Tax.sol, Factory.sol, Farms.sol, Locker.sol, Airdrop.sol, FeeHandler.sol, ManageableUpgradeable.sol, PresaleManager.sol, All audited contracts contracts were audited. 9 high, 8 medium, 7 low severity issues were found.

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.

