#HashEx
BLOCKCHAIN SECURITY

PurpleSale Mint

smart contracts final audit report

May 2024

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the PurpleSale team to perform an audit of their smart contract. The audit was conducted between 02/05/2024 and 06/05/2024.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code was provided directly in .sol files. SHA-1 hashes of the audited files are:

ERC20.sol `5e576eed578e9b4d8efa1c9f90a891271f81090e`

Mint.sol `d255ebcf983f59f307dccdd1c1df19ccaf6308be`

StandardToken.sol `d34ddeaf726afbe2627b707a7cfefadd901e2ad9`

IUniswapV2Router01.sol `6557811a6e10e9373f216981864dbc37316f080b`

IUniswapV2Factory.sol `e632113e513be591210dbae86e27187730df6682`

Update. The PurpleSale team has responded to this report. SHA-1 hashes of the updated files are:

Mint.sol `b7dbb2784c6ef7849db38827d3733bc82734ef28`

StandardToken.sol `9e801899398ac6f3054263e11935c0a4d1b05ae3`

Please note that no code verification for deployed contracts was made. Deployed code may not be the same as what was audited. To ensure that the deployed contracts are the same as those audited, users must independently verify the SHA-1 hashes of the verified contract code.

2.1 Summary

Project name	PurpleSale Mint
URL	https://purplesale.finance
Platform	Binance Smart Chain
Language	Solidity
Centralization level	High
Centralization risk	High

2.2 Contracts

Name	Address
Mint	
PurpleToken01	
PurpleToken02	
ERC20	
Interfaces	
Multiple contracts	

3. Project centralization risks

Cf7CR21 Owner privileges

- The token owner can abstain from enabling the trading, blocking all transfers to and from the selected address.
- The owner can update the pair address. If the trading switch is not enabled, all transfers to and from pair address are reverted.

Cf8CR22 Owner privileges

- The token owner can update the pair address. If it's set to a wrong or malicious address, all transfers to and from pair address are reverted.
- The owner can update the swap router address. If it's set to a wrong or malicious address, all transfers to and from pair address are reverted.
- The owner can abstain from enabling the trading, blocking all transfers to and from the selected address.
- The owner can transfer out collected fees, blocking recent pair interactions.
- The owner can update marketing and burn fees within limit set during the contract deployment.
- The owner can update the marketing address, which can block or filter token transfers.
- The owner can update the whitelist of addresses that are exempt from fees.

4. Found issues

High	2 (11%)
Medium	4 (22%)
Low	9 (50%)
Info	3 (17%)

Cf6. Mint

ID	Severity	Title	Status
Cf6Ia5	Medium	Gas limit problem	Resolved
Cf6Ia4	Low	Gas optimizations	Resolved
Cf6Ia2	Low	Default visibility of state variables	Resolved
Cf6Ia6	Low	Inconsistent code	Acknowledged
Cf6Ia3	Low	Type of created token is not tracked	Acknowledged

Cf7. PurpleToken01

ID	Severity	Title	Status
Cf7Ia9	Medium	Complicated pair creation	Resolved
Cf7Ia7	Low	Gas optimizations	Partially fixed
Cf7Ia8	Info	Limited ERC20 token support	Resolved

Cf8. PurpleToken02

ID	Severity	Title	Status
Cf8Ib0	High	Wrong swap function	Resolved
Cf8Iac	Medium	Complicated pair creation	Resolved
Cf8Ib1	Medium	Pair interactions may fail	Partially fixed
Cf8Iab	Low	Lack of events	Resolved
Cf8Iad	Low	Swaps with 100% slippage	Acknowledged
Cf8Iaa	Low	Gas optimizations	Partially fixed
Cf8Iae	Low	Lack of input validation	Acknowledged
Cf8Ib3	Info	Typographical error	Acknowledged
Cf8Iaf	Info	Limited ERC20 token support	Resolved

Cfb. Multiple contracts

ID	Severity	Title	Status
CfbIb2	High	Lack of tests and documentation	Open

5. Contracts

Cf6. Mint

Overview

A factory contract to deploy PurpleToken01 and PurpleToken02 token contracts. The Mint contract has no owner or governance functions.

Issues

Cf6Ia5 Gas limit problem

Medium Resolved

The `getAllTokensCreated()` function returns full array and may become malfunctioned in future.

Recommendation

Return slice of the `tokenDetailsArray`.

Cf6Ia4 Gas optimizations

Low Resolved

- Duplicated data in storage: the `tokenDetailsArray.length` equals the `tokenCount` variable

Cf6Ia2 Default visibility of state variables

Low Resolved

Several state variables have been declared without explicit visibility. In Solidity, the default visibility for state variables is `internal`. However, relying on default visibility can lead to misunderstandings and potential security vulnerabilities if not carefully considered and documented.

- `uint256 tokenCount`

- `TokenDetails[] tokenDetailsArray`
- `mapping(address => TokenDetails[]) personalTokenDetails`

Cf6la6 Inconsistent code

Low Acknowledged

The `createNewFeeToken()` function includes a requirement for input parameters to ensure that the sum of marketing and burn fees is not greater than 10%. But no function in the PurpleToken02 token applies both fees at once, so it's unclear why there's a need to check the sum of the fees.

```
function createNewFeeToken(
    address user,
    string calldata tokenName,
    string calldata tokenSymbols,
    uint8 decimals__,
    uint totalSupply,
    fessAndWallets memory _feesStruct
) external returns (address) {
    require(
        (_feesStruct.marketingFeeBps_ + _feesStruct.burnFee_) <= 1000,
        "Fee Limit reached"
    );
    ...
}
```

Cf6la3 Type of created token is not tracked

Low Acknowledged

The Mint contract allows users to create 2 types of ERC-20 tokens by calling either `createNewToken()` or `createNewFeeToken()` functions. Parameters of created tokens are stored in the `TokenDetails` structs and also emitted with the `TokenCreated` event. However, both these struct and event don't contain information about the type of created tokens.

```
struct TokenDetails {
    address creator;
    address tokenaddress;
```

```
        string name;
        string symbols;
        uint8 decimals;
        uint256 __totalSupply;
    }

    event TokenCreated(
        address indexed owner,
        address tokenaddress,
        string name,
        string symbols,
        uint8 decimals,
        uint totalSupply
    );
```

Cf7. PurpleToken01

Overview

An ERC-20 standard token with trading switch for a single address of DEX pair.

Issues

Cf7Ia9 Complicated pair creation

Medium Resolved

The `_transfer()` function requires the `enabledTrading` bool switch to be enabled in order to transfer tokens to or from the DEX pair address. It does not allow the owner to add the initial liquidity after the pair creation, but not before the public trading starts.

Recommendation

Allow owner's transfers:

```
    function _transfer(address from, address to, uint256 amount) internal virtual
override{
```

```
        address _pair = pair;
        bool pairTransfer = from == _pair || to == _pair;

        if (!enabledTrading) {
            if (pairTransfer && _msgSender() != owner()) {
                revert("Trading not started");
            }
        }

        super._transfer(from, to, amount);

    }
```

Cf7Ia7 Gas optimizations

Low Partially fixed

- Duplicated functions `pair()/getPair()` and `enabledTrading()/isTradingEnabled()`.
- Multiple reads from storage in the `_transfer()` function: `pair` variable.
- Unnecessary overridden functions `renounceOwnership()` and `transferOwnership()`.

Cf7Ia8 Limited ERC20 token support

Info Resolved

The `claimStuckTokens()` function doesn't use the SafeERC20 library for transferring arbitrary ERC20 tokens. Non-standard tokens may be unsupported.

Cf8. PurpleToken02

Overview

An ERC-20 standard token with transfer and burn fees. Collected fees are swapped to native EVM coins once a threshold is reached, which are transferred out to an address controlled by the token owner.

Issues

Cf8Ib0 Wrong swap function

High Resolved

The `swap()` function calls a UniswapV2-like router to swap tokens to native EVM coins. The PurpleToken02 token has a marketing fee on transfers, therefore, the `IUniswapV2Router02.swapExactTokensForETHSupportingFeeOnTransferTokens()` function should be called. Otherwise, the router calls DEX pair contract with a wrong input amount that will be reverted.

```
    function swap(uint tokenAmount) internal lockTheSwap {
        ...
        lpSwapRouter.swapExactTokensForETH(
                tokenAmount,
                0,
                path,
                address(this),
                block.timestamp
            );
    }
```

Recommendation

Use `lpSwapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens()` call.

Cf8Iac Complicated pair creation

Medium Resolved

The `_transfer()` function requires the `enabledTrading` bool switch to be enabled in order to transfer tokens to or from the DEX pair address. It does not allow the owner to add the initial liquidity after the pair creation, but not before the public trading starts.

Recommendation

Allow owner's transfers:

```
function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal virtual override {
        address _pair = pair;
        bool pairTransfer = from == _pair || to == _pair;
        if (!enabledTrading) {
            if (pairTransfer && _msgSender() != owner()) {
                revert("Trading not started");
            }
        }
        uint256 _marketingFeeBps = marketingFeeBps;
        if (pairTransfer && _marketingFeeBps > 0 && !isExcludedFromFees[from] && !
isExcludedFromFees[to]) {
            uint256 feetaken;
            feetaken = amount * marketingFeeBps / 10000;
            uint256 newTotalFees = totalFees + feetaken;
            totalFees = newTotalFees;

            super._transfer(from, address(this), feetaken);

            amount -= feetaken;

            if ((newTotalFees >= swapTokensAtAmount) && !inSwap) {
                swap(newTotalFees);
                totalFees = 0;
            }
        }
        super._transfer(from, to, amount);
    }
```

Cf8Ib1 Pair interactions may fail

Medium Partially fixed

User's interactions with the DEX pair are occasionally reverted.

Token purchases from the pair and liquidity removals may fail, if the `swap()` function is triggered in pair outgoing transfer due to pair's external state-modifying functions being protected from reentrancy.

Adding liquidity actions may fail if it's called directly via pair's `mint()` function or router's `addLiquidity()` with WETH as first token, and if the `swap()` function is triggered in pair incoming transfer. In that case WETH tokens are consumed during the triggered fees' swap, and then liquidity minting is reverted with 'INSUFFICIENT_LIQUIDITY_MINTED' error.

Recommendation

Don't trigger fees' swap when the transfer is made from the pair.

Caution users from adding liquidity by unusual methods.

Cf8lab Lack of events

Low Resolved

The functions `setMarketingWallet()`, `setMarketingFees()`, `setSwapTokensAtAmount()`, `setLpSwapRouter()`, `setTokenLPPair()`, `setBurnFee()`, `excludeFromFees()`, `includeInFees()` change important variable in the contract storage, but no event is emitted.

We recommend adding events to these functions to make it easier to track their changes offline.

Cf8lad Swaps with 100% slippage

Low Acknowledged

DEX swapping in the `swap()` function is called with 100% slippage. A large `swapTokensAtAmount` amount may result in successful MEV actions.

Cf8laa Gas optimizations

Low Partially fixed

- Multiple approvals for the router are made with each swap of the accumulated fees. A single `type(uint256).max` approval may be used.

- Unnecessary unspent allowance is granted for the `pair` address in the `swap()` function.
- Unnecessary external view calls to itself is made in the `swap()` function.
- The `totalFeePercentage` is checked in the `_transfer()` function, but only the marketing fee is applied.
- Multiple reads from storage in the `_transfer()` function: `pair` variable.
- Multiple reads from storage in the `_transfer()` and `burn()` functions: `totalFees` variable.
- Unnecessary reads from storage in the `setMarketingWallet()` function: `marketingWallet` variable.
- Duplicated functions `totalFeePercentage()/getTotalFeeBPS()`, `marketingWallet()/getMarketingWallet()`, `marketingFeeBps()/getMarketingFees()`, `swapTokensAtAmount()/getSwapTokenAtAmount()`, `lpSwapRouter()/getRouter()`, `pair()/getPair()`, `burnfeebps()/getBurnFee()`, `isExcludedFromFees()/getIsExcludedFromFees()`, and `enabledTrading()/isTradingEnabled()`.
- Unnecessary reads from storage in the `setMarketingFees()` and `setBurnFee()` functions: `marketingFeeBps`, `burnfeebps` variables.
- Unnecessary reads from storage in the `setLpSwapRouter()` function: `lpSwapRouter` variable.

Cf8Iae Lack of input validation

Low Acknowledged

Parameters of the constructor are not validated in any kind, possibly causing math overflow for unreasonably big decimals or total supply, or errors due to malfunctioning DEX router.

Cf8Ib3 Typographical error

Info Acknowledged

The smart contract code contains a typographical error in the event name `reciever`, which should be correctly spelled as `receiver` to align with standard English grammar conventions.

Cf8Iaf Limited ERC20 token support

Info Resolved

The `claimStuckTokens()` function doesn't use the SafeERC20 library for transferring arbitrary ERC20 tokens. Non-standard tokens may be unsupported.

Cf9. ERC20

Overview

An ERC-20 standard token with Ownable extensions, but without any owner restricted functions. Part of inheritance scheme of the PurpleToken01 and PurpleToken02 contracts.

Cfa. Interfaces

Overview

The IUniswapV2Router01 and IUniswapV2Factory interface contracts are used for DEX interaction, i.e., pair creation and fees swaps.

Cfb. Multiple contracts

Issues

CfbIb2 Lack of tests and documentation

High Open

The project doesn't contain any tests and documentation. We urgently recommend increasing test coverage. We also suggest providing the documentation section.

6. Conclusion

2 high, 4 medium, 9 low severity issues were found during the audit. 1 high, 3 medium, 3 low issues were resolved in the update. The reviewed contracts are highly dependent on the owner's account. See the centralization risks chapter.

This audit includes recommendations on code improvement and the prevention of potential attacks.

Please note that no code verification for deployed contracts was made. Deployed code may not be the same as what was audited. To ensure that the deployed contracts are the same as those audited, users must independently verify the SHA-1 hashes of the verified contract code.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. Issue status description

Resolved. The issue has been completely fixed.

Partially fixed. Parts of the issue have been fixed but the issue is not completely resolved.

Acknowledged. The team has been notified of the issue, no action has been taken.

Open. The issue remains unresolved.

Appendix C. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

Appendix D. Centralization risks classification

Centralization level

- **High.** The project owners can manipulate user's funds, lock user's funds on their will (reversible or irreversible), or maliciously update contracts parameters or bytecode.
- **Medium.** The project owners can modify contract's parameters to break some functions of the project contract or contracts, but user's funds remain withdrawable.
- **Low.** The contract is trustless or its governance functions are safe against a malicious owner.

Centralization risk

- **High.** Lost ownership over the project contract or contracts may result in user's losses. Contract's ownership belongs to EOA or EOAs, and their security model is unknown or out of scope.
- **Medium.** Contract's ownership is transferred to a contract with not industry-accepted parameters, or to a contract without an audit. Also includes EOA with a documented security model, which is out of scope.
- **Low.** Contract's ownership is transferred to a well-known or audited contract with industry-accepted parameters.

 contact@hashex.org

 @hashex_manager

 blog.hashex.org

 linkedin

 github

 twitter

#HashEx
BLOCKCHAIN SECURITY