



**PALADIN**  
BLOCKCHAIN SECURITY

# Smart Contract Security Assessment

Final Report

For CryptEx Tokens

21 October 2021



[paladinsec.co](http://paladinsec.co)



[info@paladinsec.co](mailto:info@paladinsec.co)

# Table of Contents

Table of Contents	2
Disclaimer	3
1 Overview	4
1.1 Summary	4
1.2 Contracts Assessed	4
1.3 Findings Summary	5
1.3.1 TokenConstructorFactory	6
1.3.2 ReflectToken	7
1.3.3 RfiTokenDeployCode	8
1.3.4 TokenDeployCode	8
1.3.5 DefaultToken	8
2 Findings	9
2.1 TokenConstructorFactory	9
2.1.1 Privileged Roles	9
2.1.2 Issues & Recommendations	10
2.2 ReflectToken	14
2.2.1 Privileged Roles	14
2.2.2 Issues & Recommendations	15
2.3 RfiTokenDeployCode	24
2.3.1 Privileged Roles	24
2.3.2 Issues & Recommendations	25
2.4 TokenDeployCode	26
2.4.1 Privileged Roles	26
2.4.2 Issues & Recommendations	27
2.5 DefaultToken	28
2.5.1 Issues & Recommendations	28

# Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocacy for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

# 1 Overview

This report has been prepared for CryptEx's token contract on the Binance Smart Chain (BSC). Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

## 1.1 Summary

<b>Project Name</b>	CryptEx by HashEx
<b>URL</b>	<a href="https://cryptexlock.me/">https://cryptexlock.me/</a>
<b>Platform</b>	Binance Smart Chain
<b>Language</b>	Solidity

## 1.2 Contracts Assessed

Name	Contract	Live Code Match
TokenConstructorFactory	TokenConstructorFactory.sol	
ReflectToken	ReflectToken.sol	
RfiTokenDeployCode	RfiTokenDeployCode.sol	
TokenDeployCode	TokenDeployCode.sol	
DefaultToken	DefaultToken.sol	

## 1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	1	1	-	-
● Medium	4	4	-	-
● Low	4	3	-	1
● Informational	13	12	-	1
<b>Total</b>	<b>22</b>	<b>20</b>	<b>-</b>	<b>2</b>

### Classification of Issues

Severity	Description
● High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
● Medium	Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
● Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
● Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

## 1.3.1 TokenConstructorFactory

ID	Severity	Summary	Status
01	LOW	No maximum payment amount can be set which might cause purchasers to overpay if governance changes the prices before the frontend updates	ACKNOWLEDGED
02	INFO	Typographical errors	RESOLVED
03	INFO	Unused ERC20 import	RESOLVED
04	INFO	rfiTokenDeployCodeAddress and tokenDeployCodeAddress should be marked as public	RESOLVED
05	INFO	Lack of events for setDeployCodeAddresses and collectPayments	RESOLVED

## 1.3.2 ReflectToken

ID	Severity	Summary	Status
06	HIGH	Token router is swappable, which could allow the owner to steal transfer taxes or turn the token into a honeypot	RESOLVED
07	MEDIUM	The transaction limit can be set infinitesimally small, making any transaction fail	RESOLVED
08	MEDIUM	Precision issue with reflection rate	RESOLVED
09	MEDIUM	Exclusion logic is flawed which could lead to transfers failing	RESOLVED
10	MEDIUM	Token could turn into a partial honeypot if the liquidity threshold is ever set to zero	RESOLVED
11	LOW	feeLimit can be made public	RESOLVED
12	LOW	Referral fee is sent to msg.sender and the referral of msg.sender instead of the from address	RESOLVED
13	LOW	While liquidity is not added to the pair, the token might turn into a honeypot	RESOLVED
14	INFO	Lack of parameter validation on liquidityAddress	RESOLVED
15	INFO	_updateSwapPair contains unused isPair parameter	RESOLVED
16	INFO	Distribute insufficient amount error is ambiguous	RESOLVED
17	INFO	Lack of events for distribute, excludeFromReward, includeInReward, excludeFromFee, includeInFee and recoverLockedTokens	RESOLVED
18	INFO	_decimals, BRN_ENABLED, MRK_ENABLED, REF_ENABLED and feeLimit can be made immutable	RESOLVED
19	INFO	Events are wrongly emitted within the constructor and setFee function	ACKNOWLEDGED

### 1.3.3 RfiTokenDeployCode

ID	Severity	Summary	Status
20	INFO	Gas optimization: Usage of memory instead of calldata	RESOLVED

### 1.3.4 TokenDeployCode

ID	Severity	Summary	Status
21	INFO	Gas optimization: Usage of memory instead of calldata	RESOLVED
22	INFO	Typographical error	RESOLVED

### 1.3.5 DefaultToken

No issues found.





# 2 Findings

---

## 2.1 TokenConstructorFactory

The TokenConstructorFactory is the main interface for users to create both simple ERC-20 tokens and reflection tokens. It levies a fee in either the native chain token or CRX. In addition, the users can opt to pay for an audit as well, which we expect will be taken care of off-chain, although we are not sure what this audit would include since the code is already audited. If the audit option is chosen during token creation, an extra fee is levied. All fees are freely configurable in the smart contract.

### 2.1.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `setDeployCodeAddresses`
- `updatePrices`
- `setBnbToCrxRatio`
- `collectPayments`
- `changeIsAddressAGoodRouter`



## 2.1.2 Issues & Recommendations

Issue #01

No maximum payment amount can be set which might cause purchasers to overpay if governance changes the prices before the frontend updates

Severity

LOW SEVERITY

Description

The creation functions do not set a maximum price for the purchase which means that users might for example see a quote of 10 tokens on the frontend and create the transaction. However, if the governance updates the price in the meantime, their transaction might execute at a higher price than expected which might cause user frustration and confusion in case this happens.

Recommendation

Consider adding a maximum price to the createToken functions that is automatically set to the current price on the frontend. If a price is updated before the frontend incorporates the new price, those createToken transactions will then fail.

Resolution

ACKNOWLEDGED



**Description**

The contract contains the following typographical errors:

Line 60

```
address owner,
```

This address is not the owner but the receiver of the initial mint.

Line 122

```
@return address oft the created token
```

This should be "of".

Line 197

```
(address tokenAddress, address issuer) = _createRFIToken(
```

The second parameter should be the owner which is not necessarily equal to the issuer of the token.

Line 228

```
Issuer
```

This address is not the owner but the receiver of the initial mint.

Line 278

```
* @return address of the created token
```

This function returns the token price.

Line 303

```
* @notice Updates prices in native and CRX and customize the price ration between ERC20 and RFI tokens
```

This comment looks outdated since this function only updates the native price directly, and ration might be a typographical error.

Line 348

```
* @param _bnbToCrxRatioBP BNB to CRX ratio for payments multiplied by 1000, i.e. 1:1 BNB/CRX ratio is _bnbToCrxRatioBP = 1000
```

This multiplier is 10000 in production.

---

**Recommendation** Consider fixing the above errors.

---

**Resolution** 

---

**Issue #03** **Unused ERC20 import**

**Severity** 

---

**Location** Line 26  
import "@openzeppelin/contracts/token/ERC20/ERC20.sol";

Line 34  
using SafeERC20 for ERC20;

---

**Description** The code contains unused code sections which might be confusing to third-party reviewers.

---

**Recommendation** Consider removing the unused code sections.

---

**Resolution** 

---



**Issue #04** **rfiTokenDeployCodeAddress and tokenDeployCodeAddress should be marked as public**

**Severity** INFORMATIONAL

**Description** Variables that are essential to the functioning of the contract should be marked as public to signal this to third parties.

**Recommendation** Consider making these variables public.

**Resolution** RESOLVED

**Issue #05** **Lack of events for setDeployCodeAddresses and collectPayments**

**Severity** INFORMATIONAL

**Description** Functions that affect the status of sensitive variables should emit events as notifications.

**Recommendation** Add events to the above functions.

**Resolution** RESOLVED



---

## 2.2 ReflectToken

The ReflectToken contract is a template token contract deployed for all reflective tokens created with the CryptEx system. It allows for a reflection fee, liquidity generation fee, a burn fee, a marketing fee and a referral fee split of 50/50 to the referral and referee.

The contract improves upon the design of SafeMoon in several regards by having more readable code and more sound include/exclude logic. It is therefore not a straightforward SafeMoon fork; instead, it is inspired by the reflection mechanism. The fees can be reconfigured up to the `feeLimit` which can be set to a maximum of 50% during deployment but cannot be changed afterwards.

### 2.2.1 Privileged Roles

The following functions can be called by the owner of the contract:

- `excludeFromReward`
- `includeInReward`
- `recoverLockedTokens`
- `excludeFromFee`
- `includeInFee`
- `setFee`
- `setLiquifyStatus`
- `setLiquifyThreshold`
- `setMarketingAddress`
- `setReferral`
- `setTxLimit`
- `setSwapRouter`



## 2.2.2 Issues & Recommendations

### Issue #06

**Token router is swappable, which could allow the owner to steal transfer taxes or turn the token into a honeypot**

#### Severity

 HIGH SEVERITY

#### Description

The contract uses a router to exchange the tokens to WETH for automatic liquidity generation, however, this router is exchangeable which could lead to the owner exchanging it for a malicious router which can be any contract of their choosing. This could be a contract that simply steals the fees instead of swapping or worse it could be a contract which reverts, which would effectively turn the token into a honeypot as purchases would be the only transactions which still work.

#### Recommendation

Consider making the router immutable and removing the swap functions.

#### Resolution

 RESOLVED

The token router can now only be changed to routers approved by CryptEx.

### Issue #07

**The transaction limit can be set infinitesimally small, making any transaction fail**

#### Severity

 MEDIUM SEVERITY

#### Description

There is no lower bound on the txLimit, the variable that limits the maximum amount of tokens that can be transferred in a single transaction. This allows the governance to set this variable to as low as zero to effectively disable transfers. The risk that this might occur could cause mistrust among investors.

#### Recommendation

Consider adding a realistic lower bound to the aforementioned function.

#### Resolution

 RESOLVED

A basic limit of 0.0001% of the total supply has been added. It should be noted that this might very well be too low for transfers.

**Issue #08****Precision issue with reflection rate****Severity** MEDIUM SEVERITY**Location**Lines 475-480

```
function _getRate() public view returns (uint256) {  
    uint256 totalRatedBalance_ = _totalRatedBalance;  
  
    if (totalRatedBalance_ == 0) return (_totalReflection /  
_totalSupply);  
    return (_totalRatedReflection / totalRatedBalance_);  
}
```

**Description**

The `_getRate` function does a division. However, as Solidity does not have decimals, there could be severe rounding errors if `totalRatedReflection_` ever gets close to `_totalRatedBalance`. This is slightly exaggerated by the fact that `_totalRatedReflection` decreases over time, while `totalRatedBalance` generally stays constant.

**Recommendation**

Consider returning both `totalReflection` and `totalBalance` so derivative functions can use a multiply-divide pattern to maintain precision.

This recommendation however leads to a second issue that the multiplication might have a high chance of overflowing. For this, the Uniswap `mulDiv` function could be considered, which does a multiply-divide pattern without overflow risk.

As this is all quite a compromise, it might suffice to add validation that `totalReflection` will always be orders of magnitude higher than `totalSupply`. This issue will be resolved when the client provides sufficient motivation and safeguards that these two variables can never get close to each other. It should be noted that `totalRatedReflections` decreases over time and it might therefore be difficult to guarantee this aspect.

**Resolution** RESOLVED

The recommendation has been implemented.



**Issue #09****Exclusion logic is flawed which could lead to transfers failing****Severity** MEDIUM SEVERITY**Location**

Lines 593-594  
\_takeLiquidity(liqAmount, rate);  
\_updateBalances(from, to, amount, rate, feesAmount);

**Description**

The contract contains logic to exclude and include accounts in receiving reflection rewards. Traditionally, within SafeMoon, this functionality was highly flawed and the client has already resolved it to a great extent. However, there are still certain edge cases which cause the token to malfunction due to this behavior like when all accounts are excluded.

We've provided statements for this that will fail and could be used to build a test case:

```
await token.connect(owner).transfer(token.address,  
parseEther('1'));  
await token.connect(owner).excludeFromReward(owner.address);  
await token.connect(owner).excludeFromReward(token.address);  
await token.connect(owner).setLiquifyThreshold(1000);  
await token.connect(owner).transfer(alice.address,  
parseEther('1'));
```

**Recommendation**

The reason why this logic fails is because \_takeLiquidity is called before \_updateBalances, while \_updateBalances is the operation that mints the reflectionary tokens that would be taken again.

Consider explaining to use why \_takeLiquidity has to go before \_updateBalances, if there is no such explanation, consider carefully validating whether the following resolution has no side-effects.

```
_updateBalances(from, to, amount, rate, feesAmount);  
_takeLiquidity(liqAmount, rate);
```

**!** WARNING: This same issue presents itself for the \_takeFee function. Consider re-organizing this function as well in case it can be executed after \_updateBalances without side-effects.

**Resolution** RESOLVED

The client has adjusted the functionality so both \_takeLiquidity and \_takeFee are ordered below \_updateBalances.

**Issue #10****Token could turn into a partial honeypot if the liquify threshold is ever set to zero****Severity** MEDIUM SEVERITY**Description**

The token will attempt to swap liquidity once the liquify threshold is reached in fees collected. However, if this variable is set to zero, this threshold will be reached even though there are no tokens within the router. Therefore, the contract will currently attempt a swap and liquidity addition and revert Uniswap-like AMMs will revert due to the lack of input tokens.

We've provided statements for this that will fail and could be used to build a test case:


```
await token.connect(owner).setLiquifyThreshold(0);  
await token.connect(owner).transfer(alice.address,1);
```

**Recommendation**

Consider adding a minimum to the liquify threshold and furthermore wrapping the uniswap operations within try-catch statements.

**Resolution** RESOLVED

The Uniswap transactions are wrapped into try-catch statements.


**Issue #11****feeLimit can be made public****Severity** LOW SEVERITY**Description**

Variables that are essential to the safety of the contract should be marked as public so third party reviewers can easily inspect them.

**Recommendation**

Consider making the variable public.

**Resolution** RESOLVED

**Issue #12****Referral fee is sent to msg.sender and the referral of msg.sender instead of the from address****Severity** LOW SEVERITY**Location**Line 537

```
address referralAddress = referrals[msg.sender];
```

Line 542

```
_takeFee(msg.sender, ref / 2, rate);
```

**Description**


The referral fee is sent to msg.sender. In practice, this will however make the referral mechanism near completely useless, since pretty much every transaction is a contract interaction where the msg.sender is equal to the contract executing transferFrom.

**Recommendation**

Consider using sender, recipient or tx.origin instead.

**Resolution** RESOLVED

The referral fee is now granted to tx.origin.

**Issue #13****While liquidity is not added to the pair, the token might turn into a honeypot****Severity** LOW SEVERITY**Description**

While there is no liquidity within the pair, the liquidity mechanism will revert. Since this mechanism is not called on token purchases, this essentially turns the token into a honeypot which might seriously mislead investors.

**Recommendation**

Consider wrapping the AMM operations in solidity try-catch statements to always allow sales to proceed, even when the liquidity generation mechanism does not function.

**Resolution** RESOLVED

The recommendation has been implemented.

**Issue #14****Lack of parameter validation on liquidityAddress****Severity** INFORMATIONAL**Location**Line 422

```
function setLiquidityAddress(address newLiquidityAddress) external  
onlyOwner {
```

**Description**

The setLiquidityAddress function currently does not validate that the liquidityAddress is not equal to the zero address. Many tokens revert when they are sent to the zero address and this could cause contract malfunction.

This issue is marked as informational severity as Uniswap-forked exchanges without modifications generally do not revert when their LP tokens are transferred to the zero address. We however still raise this issue since there is no guarantee that the router will be a straightforward fork.

**Recommendation**

Consider validating the newLiquidityAddress.

```
require(newLiquidityAddress != address(0));
```

**Resolution** RESOLVED

The recommended validation has been implemented.



**Issue #15****\_updateSwapPair contains unused isPair parameter****Severity** INFORMATIONAL**Location**Lines 469-470

```
function _updateSwapPair(address pair, bool isPair) internal {  
    swapPairs[pair] = isPair;  
}
```

**Description**

The \_updateSwapPair function contains an isPair parameter which is never set to false throughout the contract. This might confuse third party reviewers into thinking swapPair addresses can be unset while this is in fact not possible.

**Recommendation**

Consider simplifying the function to have it indicate the true behavior.

```
function _setSwapPair(address pair) internal {  
    swapPairs[pair] = true;  
}
```

**Resolution** RESOLVED

The parameter has been removed.

**Issue #16****Distribute insufficient amount error is ambiguous****Severity** INFORMATIONAL**Location**Lines 333-334

```
_reflections[msg.sender] -= rAmount;  
_totalReflection -= rAmount;
```

**Description**

The distribute function emits an ambiguous error when the account has insufficient funds to distribute. This might cause confusion for users that call this function.

**Recommendation**

Consider adding a requirement to explicitly handle the case:

```
require(_reflections[msg.sender] >= rAmount, "Insufficient  
balance");
```

**Resolution** RESOLVED

**Issue #17**

**Lack of events for `distribute`, `excludeFromReward`, `includeInReward`, `excludeFromFee`, `includeInFee` and `recoverLockedTokens`**

**Severity**

 INFORMATIONAL

**Description**

Functions that affect the status of sensitive variables should emit events as notifications.

**Recommendation**

Consider adding events to the above functions.

**Resolution**

 RESOLVED

**Issue #18**

**`_decimals`, `BRN_ENABLED`, `MRK_ENABLED`, `REF_ENABLED` and `feeLimit` can be made `immutable`**

**Severity**

 INFORMATIONAL

**Description**

Variables that are set within the constructor and remain unchanged throughout the contract can be marked as such using the keyword `immutable`. This not only signals to third-party reviewers that these variables will remain unchanged but it furthermore saves gas.

**Recommendation**

Consider making the above variables `immutable`.

**Resolution**

 RESOLVED




**Issue #19****Events are wrongly emitted within the constructor and setFee function****Severity** INFORMATIONAL**Description**

The constructor and setFee function contain events with parameters that do not necessarily match the actual stored state variable related to the parameter. This is because both fees and the marketing address are not always set depending on other parameters.

**Recommendation**

Consider only emitting the UpdateMarketingAddress if the marketing address was actually set and UpdateFees with the fee values that were actually set.

**Resolution** ACKNOWLEDGED

---

## 2.3 RfiTokenDeployCode

The RfiTokenDeployCode is a simple deployer for ReflectTokens (reflective tokens similar to SafeMoon). It contains a single function that takes in token parameters, then deploys the token and returns the address. This function can only be called by the main TokenConstructorFactory.



### 2.3.1 Privileged Roles

- `deployNewToken` (callable by `TokenConstructorFactory`)





## 2.3.2 Issues & Recommendations

<b>Issue #20</b>	<b>Gas optimization: Usage of memory instead of calldata</b>
<b>Severity</b>	 INFORMATIONAL
<b>Description</b>	The contract uses memory to signify a string instead of calldata. Using calldata could be advantageous for gas usage in this case.
<b>Recommendation</b>	Consider using calldata instead.
<b>Resolution</b>	 RESOLVED



---

## 2.4 TokenDeployCode

The TokenDeployCode is a simple deployer for DefaultTokens (basic ERC-20 tokens). It contains a single function that takes in token parameters, then deploys the token and returns the address. This function can only be called by the main TokenConstructorFactory.

### 2.4.1 Privileged Roles

- `deployNewToken` (callable by `TokenConstructorFactory`)



## 2.4.2 Issues & Recommendations

<b>Issue #21</b>	<b>Gas optimization: Usage of memory instead of calldata</b>
<b>Severity</b>	<span>INFORMATIONAL</span>
<b>Description</b>	The contract uses memory to signify a string instead of calldata. Using calldata could be advantageous for gas usage in this case.
<b>Recommendation</b>	Consider using calldata instead.
<b>Resolution</b>	<span>RESOLVED</span>

<b>Issue #22</b>	<b>Typographical error</b>
<b>Severity</b>	<span>INFORMATIONAL</span>
<b>Description</b>	The contract contains the following typographical error:  Line 39 issuer  This address is not the issuer but the receiver of the initial mint.
<b>Recommendation</b>	Consider fixing the typographical error.
<b>Resolution</b>	<span>RESOLVED</span>



---

## 2.5 DefaultToken

The DefaultToken is a simple ERC-20 token that can be deployed with a custom name, symbol, total supply and decimals. The total supply is minted to the configured receiver address and no further minting can occur.

### 2.5.1 Issues & Recommendations

No issues found.





**PALADIN**  
BLOCKCHAIN SECURITY