



**PALADIN**  
BLOCKCHAIN SECURITY

# Smart Contract Security Assessment

Final Report

For PIP

07 September 2022



[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 BscPaymentSplitterDeploy	6
2 Findings	7
2.1 BscPaymentSplitterDeploy	7
2.1.1 Privileged Functions	8
2.1.2 Issues & Recommendations	9

# 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.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

# 1 Overview

This report has been prepared for PIP's payment splitter contract on the BNB Smart Chain. 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>	PIP
<b>URL</b>	<a href="https://getpip.com">https://getpip.com</a>
<b>Network</b>	BNB Smart Chain
<b>Language</b>	Solidity

## 1.2 Contracts Assessed

Name	Contract	Live Code Match
BSPaymentSplitter	proxy: 0x11454268cb62e0E574a08eC83be1dAed1813b240  implementation: 0x783b45978671d1148482980a9bb10552f2794016	

## 1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
● High	3	3	-	-
● Medium	3	2	-	1
● Low	3	1	-	2
● Informational	5	4	1	-
<b>Total</b>	<b>14</b>	<b>10</b>	<b>1</b>	<b>3</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 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 BscPaymentSplitterDeploy

ID	Severity	Summary	Status
01	HIGH	Contract does not support tokens with a fee on transfer	✓ RESOLVED
02	HIGH	A malicious to parameter can drain the contract	✓ RESOLVED
03	HIGH	Lack of safeguards for feeAmount and gasAmount	✓ RESOLVED
04	MEDIUM	Bnb sent to the contract via a fallback function can never be withdrawn	✓ RESOLVED
05	MEDIUM	Using t transfer for the native token does not work for contracts or multi-signature wallets	ACKNOWLEDGED
06	MEDIUM	Gas token may not be withdrawable	✓ RESOLVED
07	LOW	Contract logic can be abused	ACKNOWLEDGED
08	LOW	Checks-effects-interactions pattern is not adhered to	ACKNOWLEDGED
09	LOW	Certain variables should be public	✓ RESOLVED
10	INFO	Unnecessary transfer	✓ RESOLVED
11	INFO	Unused declarations and events	✓ RESOLVED
12	INFO	Various functions can be made external	✓ RESOLVED
13	INFO	Typographical errors	PARTIAL
14	INFO	Lack of safeTransfer	✓ RESOLVED

# 2 Findings

---

## 2.1 BscPaymentSplitterDeploy

BscPaymentSplitterDeploy is a distributor-like contract. Users have the ability to send tips via this contract to other addresses either by a direct transfer or via an escrow service.

For the direct transfer, the user can either call `receiveNative` or `receiveToken` with `isEscrow != 1`. In the case of escrow (`isEscrow == 1`), the user has to pay a `feeAmount` and a `gasAmount` in addition to the user's tip. If the escrow service is not used, the user simply pays the `feeAmount`.

For the escrow service, the user's funds will be deposited into the contract and can be sent via `sendEscrow` by the admin to any receiver address. There is no further validation.

It is crucial to mention that none of these fees are validated. We assume that the frontend is automatically calculating the `gasAmount` and the `feeAmount`. This of course results in a security risk where the user can simply input an arbitrary `feeAmount` and `gasAmount`. Furthermore, the user also has the ability to call `receiveNativeByPipService` and `receiveTokenByPipService` since none of these functions are safeguarded, thus they can execute what `receiveNative` and `receiveToken` does in the case of a non-escrow transaction without any fee.



## 2.1.1 Privileged Functions

- `setGasFeeAddress`
- `getGasFeeAddress`
- `setPipFeeAddress`
- `getPipFeeAddress`
- `chkGasFee`
- `chkPipFee`
- `chkEscrowBalance`
- `withdrawGasFee`
- `withdrawPipFee`
- `sendEscrow`





## 2.1.2 Issues & Recommendations

<b>Issue #01</b>	<b>Contract does not support tokens with a fee on transfer</b>
<b>Severity</b>	 HIGH SEVERITY
<b>Description</b>	<p>Within receiveToken, the contract will increment <code>_pipFees</code> and <code>_escrowBalances</code> with the initial amount. However, the contract does not receive the full amount due to the tax deducted.</p> <p>This will later result in issues when the admin sends the escrow amount out via <code>sendEscrow</code> or transfers funds out via <code>withdrawPipFee</code>. Both functions rely on the accounting variables <code>_pipFees</code> and <code>_escrowBalances</code> which limits the amount that can be transferred out.</p> <p>If these two variables are being increased with the initial amount but the contract only receives the amount after the tax, executing both functions with the initial value will slowly drain the contract.</p>
<b>Recommendation</b>	Consider switching to a logic that supports tokens with a fee on transfer or simply consider to not accept such tokens.
<b>Resolution</b>	 RESOLVED
	<p>The client added a whitelist modifier for accepted tokens as well as a <code>safeTransferFromAndCheckBalance</code> function which ensures that the received amount is always the initial amount.</p>

**Issue #02**      **A malicious to parameter can drain the contract**

**Severity**       HIGH SEVERITY

**Description**      Within the functions `withdrawPipFee` and `sendEscrow`, there is an approval made to the `to` address before a standard transfer. Since the transfer does not adjust the approval, the `to` address can then drain the contract via `transferFrom`.

This is especially risky for the `sendEscrow` function, since the `to` address is most likely a third-party receiver.

**Recommendation**      Consider removing the unnecessary approval.


**Resolution**       RESOLVED

**Issue #03**      **Lack of safeguards for `feeAmount` and `gasAmount`**

**Severity**       HIGH SEVERITY

**Description**      `receiveToken` and `receiveNative` both lack validation for the input parameters. Therefore, users can circumvent the fee by inputting a very low `feeAmount` and `gasAmount`.

**Recommendation**      We recommend to determine a standard `gasAmount` and to calculate the `feeAmount` based on the `tipAmount`.

**Resolution**       RESOLVED

We suggested a different fix for this issue, however, the client indicated that their fix logic works as desired.

<b>Issue #04</b>	<b>Bnb sent to the contract via a fallback function can never be withdrawn</b>
<b>Severity</b>	<span>MEDIUM SEVERITY</span>
<b>Description</b>	The fallback functions receive and fallback allow anyone to send BNB directly to the contract, however, there is no way to withdraw that amount because it was not accounted for.
<b>Recommendation</b>	Consider removing the fallback functions
<b>Resolution</b>	<span>RESOLVED</span> The client will remove these functions in the final version.

<b>Issue #05</b>	<b>Using transfer for the native token does not work for contracts or multi-signature wallets</b>
<b>Severity</b>	<span>MEDIUM SEVERITY</span>
<b>Description</b>	The use of transfer for the native gas token just forwards 2100 gas to the recipient. If the recipient is a smart contract or a multi-signature wallet, there is usually contract logic that gets executed as a fallback function when receiving the gas token. However, in the case of transfer, this will simply run out of gas and revert.
<b>Recommendation</b>	Consider using call instead of transfer.
<b>Resolution</b>	<span>ACKNOWLEDGED</span>

**Issue #06**      **Gas token may not be withdrawable**

**Severity**      MEDIUM SEVERITY

**Description**      Within receiveNative and receiveToken, the gasAmount can be chosen arbitrarily. If a user decides to pay a gasAmount > 0 for the else cases, \_gasFee will not get increased accordingly.

This exposes an issue with withdrawGasFees because this function relies on the correct accounting of \_gasFee.

**Recommendation**      Consider accounting for \_gasFee correctly if it is > 0.

**Resolution**      RESOLVED

receiveNative and receiveToken both require gasAmount to be zero for nonEscrow cases. Additionally, an upper limit was introduced to the setGasFee function and the bug within the requirement was fixed.

**Issue #07**      **Contract logic can be abused**

**Severity**      LOW SEVERITY

**Description**      Since there are no safeguards for receiveTokenByPipService and receiveNativeByPipService, users can simply call this function instead of receiveToken and receiveNative and circumvent the fee logic.

**Recommendation**      Consider either acknowledging this or rethinking the contract logic.

**Resolution**      ACKNOWLEDGED

**Issue #08**      **Checks-effects-interactions pattern is not adhered to**

**Severity**      ● LOW SEVERITY

**Description**      Within the whole contract, the checks-effects-interactions pattern is not adhered to. Even if all functions are safeguarded with the nonReentrant modifier, Paladin always recommends adhering to the checks-effects-interactions pattern ([https://fravoll.github.io/solidity-patterns/checks\\_effects\\_interactions.html](https://fravoll.github.io/solidity-patterns/checks_effects_interactions.html)).

**Recommendation**      Consider changing the contract logic to adhere to the checks-effects-interactions pattern.

**Resolution**      ● ACKNOWLEDGED

**Issue #09**      **Certain variables should be public**

**Severity**      ● LOW SEVERITY

**Description**      Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts.

The following variables should be marked as public:

- \_escrowBalances
- \_pipFees
- \_gasFee
- \_pipFeeAddress
- \_gasFeeAddress

**Recommendation**      Consider either making these variables public, or remove the onlyAdmin modifier for the view functions.

**Resolution**      ✓ RESOLVED

<b>Issue #10</b>	<b>Unnecessary transfer</b>
<b>Severity</b>	<span>INFORMATIONAL</span>
<b>Location</b>	<p><u>Line 249</u> payable(address(this)).call{value: msg.value};</p> <p><u>Line 257</u> payable(address(this)).call{value: feeAmount};</p> <p><u>Line 283</u> payable(address(this)).call{value: gasAmount};</p>
<b>Description</b>	Within several functions, the contract tries to send itself the gas token via call . However, during the function call itself, msg.value is automatically sent to the contract itself. This call is unnecessary and just consumes gas.
<b>Recommendation</b>	Consider removing the unnecessary calls.
<b>Resolution</b>	<span>RESOLVED</span>



**Issue #11****Unused declarations and events****Severity** INFORMATIONAL**Description**

Declarations, variables, functions, events, etc. defined in a contract but not used within said contract could confuse third-party auditors. They also increase the contract length unnecessarily.

Lines 18-19

```
using SafeERC20Upgradeable for IERC20Upgradeable;  
using StringsUpgradeable for string;
```

These declarations are not actively used within the contract, although SafeERC20Upgradeable should be used.

Line 55-60

```
event Approve(string approveType, address indexed  
toContract, address indexed spender, uint256 indexed  
amount );
```

This event is not used.

**Recommendation**

Consider removing all unused or unnecessary events and declarations.

**Resolution** RESOLVED

**Issue #12****Various functions can be made external****Severity** INFORMATIONAL**Description**

Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.

The following functions can be made external:

- setGasFeeAddress
- getGasFeeAddress
- setPipFeeAddress
- getPipFeeAddress
- chkGasFee
- chkPipFee
- chkEscrowBalance
- withdrawGasFee
- withdrawPipFee
- sendEscrow
- receiveNative
- receiveToken
- receiveNativeByPipService
- receiveTokenByPipService

**Recommendation**

Consider making the above functions external.

**Resolution** RESOLVED



**Description**

We have consolidated the typographical errors into a single issue to keep the report brief and readable.

Line 176

```
function chkEscrowBalance(address target)
```

The parameter should be token.

Line 183

```
function withdrawGasFee(address payable to, uint256 amount)
public payable
```

The payable keyword for the function can be removed. Additionally, the to parameter is unnecessary since it must always be `_gasFeeAddress`.

Line 196

```
function withdrawPipFee(address symbol, address payable to,
uint256 amount) public payable
```

The payable keyword for the function can be removed.

symbol should be renamed to token, which would make it more readable for third-party reviewers.

Additionally, the to parameter is unnecessary since it must always be `_pipFeeAddress`.

Line 196

```
_pipFees[symbol] -= amount;
```

This can be done at the beginning of the function — there is no need to do this two times for each case.

---

Line 216

```
function sendEscrow(address symbol, address payable to,  
uint256 amount) public payable
```

The payable keyword for the function can be removed, we do not expect that the admin wants to add any msg.value here.

symbol should be renamed to token, which would make it more readable for third-party reviewers.

Line 223

```
payable(to).transfer(amount);
```

The to address is already wrapped with payable.

Line 241

```
function receiveNative(uint256 isEscrow, address payable  
recipient, uint256 tipAmount, uint256 feeAmount, uint256  
gasAmount)
```

isEscrow should be a boolean value. The same issue exists with the receiveToken function.

Line 316

```
function receiveTokenByPipService(address toContract,  
address recipient, uint256 amount, string memory payload)  
public payable
```

The payable keyword for the function can be removed.

---


**Recommendation** Consider fixing the above typographical errors.

**Resolution**

 PARTIALLY RESOLVED

Not all the errors have been fixed.

---

**Issue #14****Lack of safeTransfer****Severity** INFORMATIONAL**Description**

Even if SafeERC20Upgradeable is imported correctly, it is not being used. All transfers are made with transferFrom or transfer instead of using safeTransferFrom or transferFrom. This does not work for tokens that will return false on transfer (or malformed tokens that do not have a return value).

**Recommendation**

Consider using safeTransfer instead of transfer.

**Resolution** RESOLVED

The contract now uses safeTransfer and safeTransferFrom.





**PALADIN**  
BLOCKCHAIN SECURITY