Biconomy

TransferHandler Smart Contract Audit

Prepared by: **Halborn**

Date of Engagement: **August 19th, 2021 - September 1st, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	08/31/2021	Ataberk Yavuzer
0.9	Document Edits	08/31/2021	Ataberk Yavuzer
1.0	Final Version	08/31/2021	Gabi Urrutia
1.1	Remediation Plan	09/06/2021	Ataberk Yavuzer

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Biconomy engaged Halborn to conduct a security assessment on a smart contract beginning on August 19th, 2021 and ending September 1st, 2021. The security assessment was scoped to the TransferHandlerCustom.sol smart contract provided in the Biconomy Repository. Halborn conducted this audit to measure security risk and identify any new vulnerabilities introduced during the final stages of development before the production release.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week for the engagement and assigned a full time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverables set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure smart-contract development.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the smart contract audit. While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of smart contracts and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.
- Smart Contract manual code review and walkthrough.
- Graphing out functionality and contract logic/connectivity/functions(solgraph)
- Manual Assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes.
- Static Analysis of security for scoped contract, and imported functions.(Slither)
- Test Net deployment (RemixIDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident, and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to the smart contracts:

- TransferHandlerCustom.sol

Commit ID: 4bc2eb1559b8dfcf3983a4085472f2b7d440ba30

OUT-OF-SCOPE:

Other smart contracts in the repository, external libraries and economics attacks.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	2	3

LIKELIHOOD

IMPACT

🟨	🟨	🟥	🟥	🟪
🟩	🟨	🟨 (HAL-01)	🟥	🟥
🟩	🟩 (HAL-03) (HAL-04)	🟨 (HAL-02)	🟨	🟥
⬜ (HAL-05)	🟩	🟩	🟨	🟨
⬜ (HAL-07)	⬜ (HAL-06)	🟩	🟩	🟨

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL01) - USERS CAN MANIPULATE TRANSFER FEE AMOUNT	Medium	SOLVED: 09/05/2021
(HAL02) - OWNER CAN RENOUNCE OWNERSHIP	Medium	SOLVED: 09/05/2021
(HAL03) - LACK OF ZERO ADDRESS CHECK ON CONSTRUCTOR	Low	SOLVED: 09/05/2021
(HAL04) - IMPROPER IMPLEMENTATION OF FEE RECEIVER	Low	SOLVED: 09/05/2021
(HAL05) - EXPERIMENTAL FEATURES ENABLED	Informational	SOLVED: 09/05/2021
(HAL06) - PRAGMA VERSION	Informational	ACKNOWLEDGED
(HAL07) - MISSING EVENTS EMITTING	Informational	SOLVED: 09/05/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) USERS CAN MANIPULATE THE TRANSFER FEE - MEDIUM

Description:

The _feeTransferHandler function calculates gas charge for transfers. It uses the following mathematical operation to calculate the charge:

Listing 1

```
tokenGasPrice.mul(executionGas).mul(feeMultiplier).div(10000);
```

According to this function, the charge result could be **0** if users supply one of the tokenGasPrice, executionGas or feeMultiplier values as **0** while using the external transfer() function. Also, it is not possible to manipulate feeMultiplier variable to **0** without having admin privileges. In addition, it is not possible to manipulate executionGas variable since it is using the gasleft() function second time while calculating the executionGas variable. However. Any user can set tokenGasPrice variable to **0** while using the transfer() function. As a result, the charge will be zero since 0.mul(executionGas).mul(feeMultiplier).div(10000) equals to **0** and token will not be charged during the transfer.

Code Location:

Listing 2: TransferHandlerCustom.sol (Lines 193)

```
191 function _feeTransferHandler(uint256 tokenGasPrice, address _payer
        , address token, uint256 executionGas) internal returns(uint256
         charge){
192         // optional checks if token is allowed could be added
193         charge = tokenGasPrice.mul(executionGas).mul(feeMultiplier
                ).div(10000);
194         //Needs safe transfer from to support USDT SafeERC20.
                safeTransferFrom(IERC20(token),_payer, feeReceiver,
                charge)
195          SafeERC20.safeTransferFrom(IERC20(token), _payer,
                feeReceiver,charge);
        /*require(IERC20(token).transferFrom(
            _payer,
            feeReceiver,
            charge));*/
    }
```

Recommendation:

There is a pre-flight check for controlling the tokenGasPrice variable on the JS side. However, these controls are also should be applied to the related contract. It is recommended to implement a zero value check to the control for validating the tokenGasPrice variable.

Remediation Plan:

SOLVED: Biconomy Team implemented an additional security check which interacts with the Price Oracle. This new check controls the tokenGasPrice value if tokenGasPrice is **0** or not. It has been observed that this new implementation resolved the issue.

3.2 (HAL-02) OWNER CAN RENOUNCE OWNERSHIP - MEDIUM

Description:

The Owner of the contract is usually the account that deploys the contract. As a result, the Owner is able to perform some privileged functions like setFeeReceiver(), setBaseGas() and etc. In the TransferHandlerCustom.sol smart contract, the renounceOwnership function is used to renounce the Owner permission. Renouncing ownership before transferring would result in the contract having no Owner, eliminating the ability to call privileged functions.

Risk Level:

Likelihood - 3
Impact - 3

Code Location:

Listing 3: TransferHandlerCustom.sol Ownable (Lines 12)

```
12 contract TransferHandlerCustom is EIP712MetaTransaction("
       ERC20Transfer","1"), Ownable{
13
14 ...
```

Recommendation:

It is recommended that the Owner is not able to call renounceOwnership without transferring the Ownership to other address before. In addition, if a multi-signature wallet is used, calling renounceOwnership function should be confirmed for two or more users. As an other solution, Renounce Ownership functionality can be disabled with the following line.

Listing 4: Disable RenounceOwnership (Lines 1)

```
function renounceOwnership () public override onlyOwner {
    revert("can 't renounceOwnership here "); // not possible with
        this smart contract
}
```

Remediation Plan:

SOLVED: Biconomy Team replaced the default Ownable.sol library with customized OwnableWithoutRenounce.sol library and eliminated the RenounceOwnership function. It has been confirmed by the Halborn Team that this vulnerability has been fixed.

3.3 (HAL-03) LACK OF ZERO ADDRESS CHECK ON CONSTRUCTOR - LOW

Description:

The TransferHandlerCustom.sol contract includes an Owner role. For example, the Owner role can change the baseGas value. It is very important to provide valid addresses to this role. Also, the Owner role should be driven by people. There are too many address checks in the TransferHandlerCustom contract to keep this role safe. For example, it is not possible to set Owner or FeeReceiver addresses to address(0) after initialization of the contract. However, it is possible to set Owner address to address(0) because of the lack of address control on constructor.

Risk Level:

Likelihood - 2
Impact - 3

Code Location:

Listing 5: TransferHandlerCustom.sol (Lines 54)

```
constructor(address _owner) public Ownable(_owner){
    }
```

Recommendation:

It is recommended to implement zero address check on constructor.

Remediation Plan:

SOLVED: Biconomy Team solved this issue by implementing new zero address checks on the constructor. It is not possible to set owner or feeReceiver to address(0) anymore. It has been confirmed by the Halborn Team that this vulnerability has been fixed.

3.4 (HAL-04) IMPROPER IMPLEMENTATION OF FEE RECEIVER - LOW

Description:

Roles are used on the developed contracts to provide ease of use or to separate the tasks on the contract from each other. It is very important that these roles should be sharply separated from each other during the deployment phase of the contract and assigned to the right accounts.

In some cases, unexpected conditions may occur as a result of incorrect programming of these roles. During the tests, it was seen that the Fee Receiver role on the TransferHandlerCustom.sol contract was not initialized on the constructor while deploying the contract. As a result of this situation, there will be no feeReceiver address on the contract and transactions will not be performed.

Code Location:

Listing 6: TransferHandlerCustom.sol (Lines 54)

```
constructor(address _owner) public Ownable(_owner){
    }
```

Recommendation:

It is recommended to implement the following code for the fees:

Listing 7: TransferHandlerCustom.sol (Lines 54,55)

```
constructor(address _owner, address _feeReceiver) public Ownable(
   _owner){
        require(
            _feeReceiver != address(0),
            "Transfer Handler: the fee receiver can not be a zero
                address"
        );
        feeReceiver = _feeReceiver;
    }
```

Remediation Plan:

SOLVED: It has been confirmed by the Halborn Team that this vulnerability has been fixed due to the Biconomy Team applied the recommendation on the constructor.

3.5 (HAL-05) EXPERIMENTAL FEATURES ENABLED - INFORMATIONAL

Description:

ABIEncoderV2 is enabled and the use of experimental features could be dangerous on live deployments. The experimental ABI encoder does not handle non-integer values shorter than 32 bytes properly. This applies to bytesNN types, bool, enum and other types when they are part of an array or a struct and encoded directly from storage. This means these storage references have to be used directly inside `abi.encode(...)` as arguments in external function calls or in event data without prior assignment to a local variable. The types **bytesNN** and **bool** will result in corrupted data while enum might lead to an invalid revert.

Risk Level:

Likelihood - 1
Impact - 2

Code Location:

Listing 8: TransferHandlerCustom.sol (Lines 3)

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.0;
pragma experimental ABIEncoderV2;
```

Recommendation:

When possible, do not use experimental features in the final live deployment. Validate and check that all the conditions above are true for integers and arrays (i.e. all using uint256).

Remediation Plan:

SOLVED: Biconomy Team solved this issue by removing pragma experimental ABIEncoderV2 from the contract. It has been confirmed by the Halborn Team that this vulnerability has been fixed.

3.6 (HAL-06) PRAGMA VERSION - INFORMATIONAL

Description:

The TransferHandlerCustom.sol contract uses one of the newest pragma versions (0.8.0), which was released on March 10, 2020. Since this version is not one of the generally used reliable versions, we do not recommend using it. It is recommended to use an older but secure version, as this version is a newer version and there may still be new security vulnerabilities on it.

Code Location:

Listing 9: TransferHandlerCustom.sol (Lines 2)

```
pragma solidity 0.8.0;
```

Recommendation:

In the Solidity Github repository, there is a “json” file listing the bugs reported for each compiler version. A security vulnerability has been found in the pragma version 0.7.6. The latest stable version is pragma 0.6.12. Furthermore, pragma 0.6.12 is widely used by Solidity developers and has been extensively tested in many security audits. We recommend using the latest stable version.

Please check the list below:
https://github.com/ethereum/solidity/blob/develop/docs/bugs_by_version.json

Remediation Plan:

ACKNOWLEDGED: Biconomy Team decided to continue using pragma 0.8.0 and accepts the risk.

3.7 (HAL-07) MISSING EVENTS EMITTING - INFORMATIONAL

Description:

It has been observed that critical functionality is missing emitting event for setDefaultFeeMultiplier and setFeeReceiver functions. These functions should emit events after completing the transactions.

Risk Level:

Likelihood - 1
Impact - 1

Code Location:

Listing 10: TransferHandlerCustom.sol

```
function setDefaultFeeMultiplier(uint16 _bp) external onlyOwner{
        require(_bp <= maximumMarkup, "fee multiplier is too high"
            );
        feeMultiplier = _bp;
    }
```

Listing 11: TransferHandlerCustom.sol

```
function setFeeReceiver(address _feeReceiver) external onlyOwner{
        require(
            _feeReceiver != address(0),
            "Transfer Handler: new fee receiver can not be a zero
               address"
        );
        feeReceiver = _feeReceiver;
    }
```

Recommendation:

Consider emitting an event when calling setDefaultFeeMultiplier and setFeeReceiver functions.

Listing 12

```
event setDefaultFeeMultiplier(uint16 _bp);
event setFeeReceiver(address _feeReceiver);
```

Remediation Plan:

SOLVED: Biconomy Team solved this issue by implementing new events and emitting these events on the necessary functions. It has been seen by the Halborn team that the specified vulnerability has been fixed.

AUTOMATED TESTING

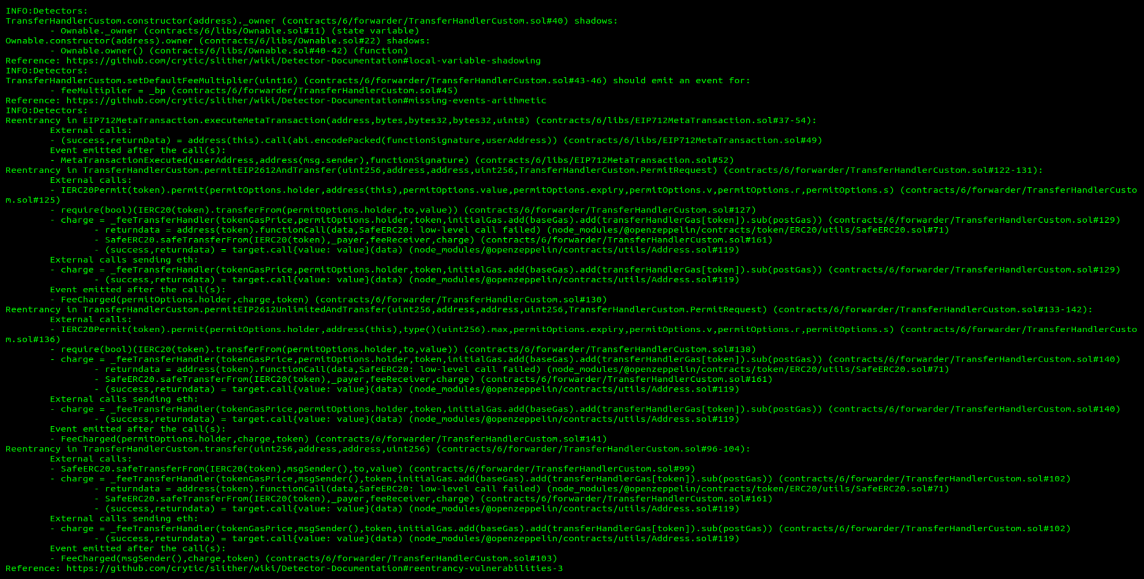
4.1 STATIC ANALYSIS REPORT

Description:

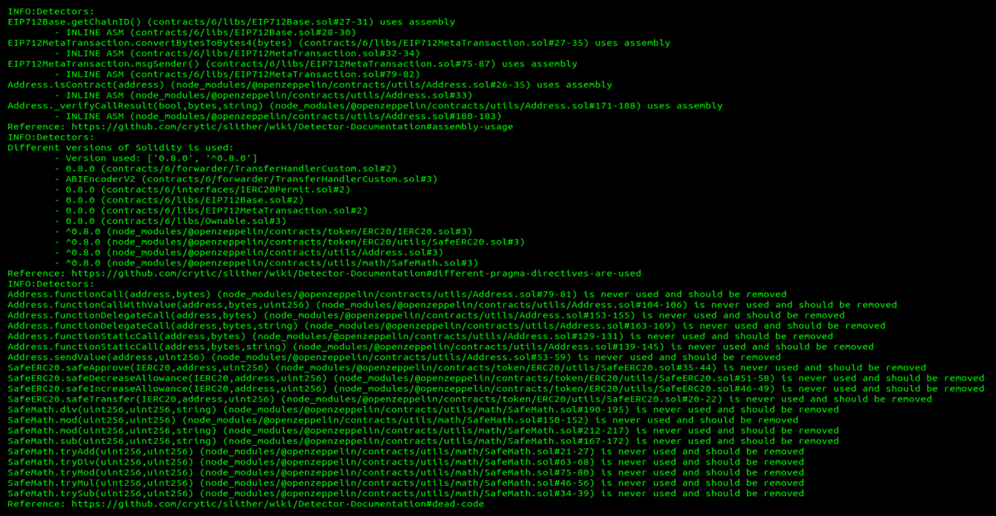
Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

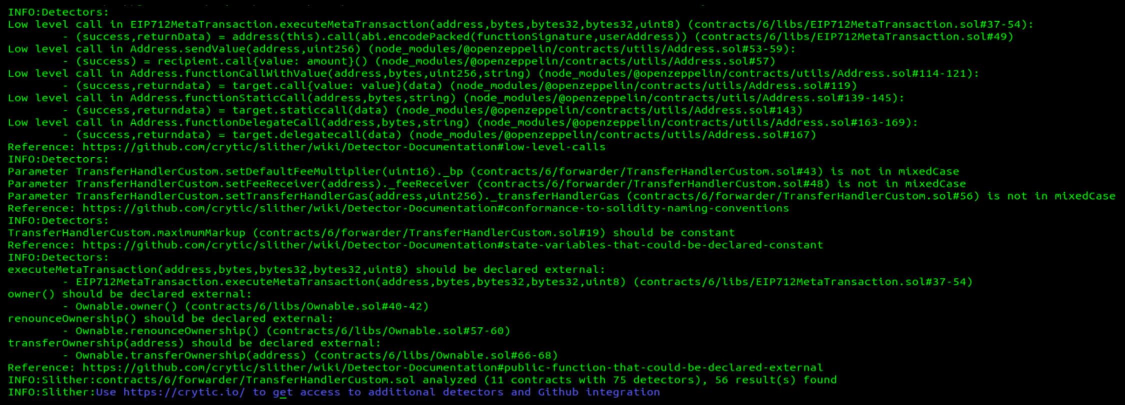
TransferHandlerCustom.sol



```
INFO:Detectors:
TransferHandlerCustom.constructor(address)._owner (contracts/6/forwarder/TransferHandlerCustom.sol#40) shadows:
        - Ownable._owner (contracts/6/libs/Ownable.sol#11) (state variable)
Ownable.constructor(address).owner (contracts/6/libs/Ownable.sol#22) shadows:
        - Ownable.owner() (contracts/6/libs/Ownable.sol#40-42) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
TransferHandlerCustom.setDefaultFeeMultiplier(uint16) (contracts/6/forwarder/TransferHandlerCustom.sol#43-46) should emit an event for:
        - feeMultiplier = _bp (contracts/6/forwarder/TransferHandlerCustom.sol#45)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
Reentrancy in EIP712MetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (contracts/6/libs/EIP712MetaTransaction.sol#37-54):
        External calls:
        - (success,returnData) = address(this).call(abi.encodePacked(functionSignature,userAddress)) (contracts/6/libs/EIP712MetaTransaction.sol#49)
        Event emitted after the call(s):
        - MetaTransactionExecuted(userAddress,address(msg.sender),functionSignature) (contracts/6/libs/EIP712MetaTransaction.sol#52)
Reentrancy in TransferHandlerCustom.permitEIP2612AndTransfer(uint256,address,address,uint256,TransferHandlerCustom.PermitRequest) (contracts/6/forwarder/TransferHandlerCustom.sol#122-131):
        External calls:
        - IERC20Permit(token).permit(permitOptions.holder,address(this),permitOptions.value,permitOptions.expiry,permitOptions.v,permitOptions.r,permitOptions.s) (contracts/6/forwarder/TransferHandlerCusto
m.sol#125)
        - require(bool)(IERC20(token).transferFrom(permitOptions.holder,to,value)) (contracts/6/forwarder/TransferHandlerCustom.sol#127)
        - charge = _feeTransferHandler(tokenGasPrice,permitOptions.holder,token,initialGas.add(baseGas).add(transferHandlerGas[token]).sub(postGas)) (contracts/6/forwarder/TransferHandlerCustom.sol#129)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#71)
                - SafeERC20.safeTransferFrom(IERC20(token),_payer,feeReceiver,charge) (contracts/6/forwarder/TransferHandlerCustom.sol#161)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
        External calls sending eth:
        - charge = _feeTransferHandler(tokenGasPrice,permitOptions.holder,token,initialGas.add(baseGas).add(transferHandlerGas[token]).sub(postGas)) (contracts/6/forwarder/TransferHandlerCustom.sol#129)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
        Event emitted after the call(s):
        - FeeCharged(permitOptions.holder,charge,token) (contracts/6/forwarder/TransferHandlerCustom.sol#130)
Reentrancy in TransferHandlerCustom.permitEIP2612UnlimitedAndTransfer(uint256,address,address,uint256,TransferHandlerCustom.PermitRequest) (contracts/6/forwarder/TransferHandlerCustom.sol#133-142):
        External calls:
        - IERC20Permit(token).permit(permitOptions.holder,address(this),type()(uint256).max,permitOptions.expiry,permitOptions.v,permitOptions.r,permitOptions.s) (contracts/6/forwarder/TransferHandlerCusto
m.sol#136)
        - require(bool)(IERC20(token).transferFrom(permitOptions.holder,to,value)) (contracts/6/forwarder/TransferHandlerCustom.sol#138)
        - charge = _feeTransferHandler(tokenGasPrice,permitOptions.holder,token,initialGas.add(baseGas).add(transferHandlerGas[token]).sub(postGas)) (contracts/6/forwarder/TransferHandlerCustom.sol#140)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#71)
                - SafeERC20.safeTransferFrom(IERC20(token),_payer,feeReceiver,charge) (contracts/6/forwarder/TransferHandlerCustom.sol#161)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
        External calls sending eth:
        - charge = _feeTransferHandler(tokenGasPrice,permitOptions.holder,token,initialGas.add(baseGas).add(transferHandlerGas[token]).sub(postGas)) (contracts/6/forwarder/TransferHandlerCustom.sol#140)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
        Event emitted after the call(s):
        - FeeCharged(permitOptions.holder,charge,token) (contracts/6/forwarder/TransferHandlerCustom.sol#141)
Reentrancy in TransferHandlerCustom.transfer(uint256,address,address,uint256) (contracts/6/forwarder/TransferHandlerCustom.sol#96-104):
        External calls:
        - SafeERC20.safeTransferFrom(IERC20(token),msgSender(),to,value) (contracts/6/forwarder/TransferHandlerCustom.sol#99)
        - charge = _feeTransferHandler(tokenGasPrice,msgSender(),token,initialGas.add(baseGas).add(transferHandlerGas[token]).sub(postGas)) (contracts/6/forwarder/TransferHandlerCustom.sol#102)
                - returndata = address(token).functionCall(data,SafeERC20: low-level call failed) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#71)
                - SafeERC20.safeTransferFrom(IERC20(token),_payer,feeReceiver,charge) (contracts/6/forwarder/TransferHandlerCustom.sol#161)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
        External calls sending eth:
        - charge = _feeTransferHandler(tokenGasPrice,msgSender(),token,initialGas.add(baseGas).add(transferHandlerGas[token]).sub(postGas)) (contracts/6/forwarder/TransferHandlerCustom.sol#102)
                - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
        Event emitted after the call(s):
        - FeeCharged(msgSender(),charge,token) (contracts/6/forwarder/TransferHandlerCustom.sol#103)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3
```



```
INFO:Detectors:
EIP712Base.getChainID() (contracts/6/libs/EIP712Base.sol#27-31) uses assembly
        - INLINE ASM (contracts/6/libs/EIP712Base.sol#28-30)
EIP712MetaTransaction.convertBytesToBytes4(bytes) (contracts/6/libs/EIP712MetaTransaction.sol#27-35) uses assembly
        - INLINE ASM (contracts/6/libs/EIP712MetaTransaction.sol#32-34)
EIP712MetaTransaction.msgSender() (contracts/6/libs/EIP712MetaTransaction.sol#75-87) uses assembly
        - INLINE ASM (contracts/6/libs/EIP712MetaTransaction.sol#79-82)
Address.isContract(address) (node_modules/@openzeppelin/contracts/utils/Address.sol#26-35) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#33)
Address._verifyCallResult(bool,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#171-188) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#180-183)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Different versions of Solidity is used:
        - Version used: ['0.8.0', '^0.8.0']
        - 0.8.0 (contracts/6/forwarder/TransferHandlerCustom.sol#2)
        - ABIEncoderV2 (contracts/6/forwarder/TransferHandlerCustom.sol#3)
        - 0.8.0 (contracts/6/interfaces/IERC20Permit.sol#2)
        - 0.8.0 (contracts/6/libs/EIP712Base.sol#2)
        - 0.8.0 (contracts/6/libs/EIP712MetaTransaction.sol#2)
        - 0.8.0 (contracts/6/libs/Ownable.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Address.sol#3)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#3)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
Address.functionCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#79-81) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#104-106) is never used and should be removed
Address.functionDelegateCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#153-155) is never used and should be removed
Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#163-169) is never used and should be removed
Address.functionStaticCall(address,bytes) (node_modules/@openzeppelin/contracts/utils/Address.sol#129-131) is never used and should be removed
Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#139-145) is never used and should be removed
Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#53-59) is never used and should be removed
SafeERC20.safeApprove(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#35-44) is never used and should be removed
SafeERC20.safeDecreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#51-58) is never used and should be removed
SafeERC20.safeIncreaseAllowance(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#46-49) is never used and should be removed
SafeERC20.safeTransfer(IERC20,address,uint256) (node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol#20-22) is never used and should be removed
SafeMath.div(uint256,uint256,string) (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#190-195) is never used and should be removed
SafeMath.mod(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#150-152) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#212-217) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#167-172) is never used and should be removed
SafeMath.tryAdd(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#21-27) is never used and should be removed
SafeMath.tryDiv(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#63-68) is never used and should be removed
SafeMath.tryMod(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#75-80) is never used and should be removed
SafeMath.tryMul(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#46-56) is never used and should be removed
SafeMath.trySub(uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/SafeMath.sol#34-39) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
```



```
INFO:Detectors:
Low level call in EIP712MetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (contracts/6/libs/EIP712MetaTransaction.sol#37-54):
        - (success,returnData) = address(this).call(abi.encodePacked(functionSignature,userAddress)) (contracts/6/libs/EIP712MetaTransaction.sol#49)
Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#53-59):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#57)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#114-121):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#119)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#139-145):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#143)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#163-169):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#167)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Parameter TransferHandlerCustom.setDefaultFeeMultiplier(uint16)._bp (contracts/6/forwarder/TransferHandlerCustom.sol#43) is not in mixedCase
Parameter TransferHandlerCustom.setFeeReceiver(address)._feeReceiver (contracts/6/forwarder/TransferHandlerCustom.sol#48) is not in mixedCase
Parameter TransferHandlerCustom.setTransferHandlerGas(address,uint256)._transferHandlerGas (contracts/6/forwarder/TransferHandlerCustom.sol#56) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
TransferHandlerCustom.maximumMarkup (contracts/6/forwarder/TransferHandlerCustom.sol#19) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) should be declared external:
        - EIP712MetaTransaction.executeMetaTransaction(address,bytes,bytes32,bytes32,uint8) (contracts/6/libs/EIP712MetaTransaction.sol#37-54)
owner() should be declared external:
        - Ownable.owner() (contracts/6/libs/Ownable.sol#40-42)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (contracts/6/libs/Ownable.sol#57-60)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (contracts/6/libs/Ownable.sol#66-68)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:contracts/6/forwarder/TransferHandlerCustom.sol analyzed (11 contracts with 75 detectors), 56 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

As a result of the tests completed with the Slither tool, some results were obtained and these results were reviewed by Halborn. In line with the reviewed results, it was decided that some vulnerabilities were false-positive and these results were not included in the report. The actual vulnerabilities are already included in the findings on the report.

THANK YOU FOR CHOOSING

// HALBORN