Smart contract security audit FractalCash

v.1.2

Table of Contents

1.0 Introduction **3**
1.1 Project engagement 3
1.2 Disclaimer 3
2.0 Coverage **4**
2.1 Target Code and Revision 4
2.2 Attacks made to the contract 5
3.0 Security Issues **7**
3.1 High severity issues [1] 7
3.2 Medium severity issues [2] 7
3.3 Low severity issues [2] 8
3.4 Informational issues [1] 8
4.0 Testing coverage - python **9**
5.0 Annexes **10**
6.0 Summary of the audit **26**

1.0 Introduction

1.1 Project engagement

During June of 2023, FractalCash team engaged CTDSec to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. FractalCash provided CTDSec with access to their code repository and whitepaper. Fractalcash use solidity and circom to provide their services of anonymous balance/transfers.

1.2 Disclaimer

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the network's fast-paced and rapidly changing environment, we at CTDSec recommend that FractalCash team put in place a bug bounty program to encourage further and active analysis of the smart contract. Contract audits are performed to help the community keep the ecosystem safe in a decentralized way.

2.0 Coverage

2.1 Target Code and Revision

For this audit, we performed research, investigation, and review of the FractalCash contract followed by issue reporting, along with mitigation and remediation instructions outlined in this report. The following code files are considered in-scope for the review:

Source file:

Contracts_fractal.zip [SHA256] -

7563b222529e400efccdaecbc47c4803cdd15f2e92905739bd82c31313c8008a

circuits.zip [SHA256] - 6a348e6aad07a52d95bc67e1fe28c09494e2be64587653fe34e0e9afa2e577ed

Fixed version:

Contracts_fix_fractal.zip [SHA256] -

c4ef925c03a14f301607a0cda53af68a8ad9ed26dc6dba571b1adb90bcb9d522

2.2 Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

№	Issue description.	Checking status
1	Compiler warnings.	PASSED
2	Race conditions and Reentrancy. Cross-function race conditions.	FIXED ISSUES
3	Possible delays in data delivery.	PASSED
4	Oracle calls.	PASSED
5	Front running.	PASSED
6	Timestamp dependence.	PASSED
7	Integer Overflow and Underflow.	PASSED
8	DoS with Revert.	PASSED
9	DoS with block gas limit.	PASSED
10	Methods execution permissions.	PASSED
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	PASSED
12	The impact of the exchange rate on the logic.	PASSED
13	Private user data leaks.	PASSED
14	Malicious Event log.	PASSED
15	Scoping and Declarations.	PASSED

16	Uninitialized storage pointers.	PASSED
17	Arithmetic accuracy.	PASSED
18	Design Logic.	FIXED ISSUES
19	Cross-function race conditions.	PASSED
20	Safe Zeppelin module.	PASSED
21	Fallback function security.	PASSED
22	Overpowered functions / Owner privileges	PASSED

3.0 Security Issues

3.1 High severity issues [1]

1. Reentrancy attack

Contract: WrappedPrivateETH

Function: deposit(), withdraw(), depositAndWrap(), unwrapAndWithdraw() and completePrivateTransfer()

Problem: The functions can be called from an untrusted contract caller causing a reentrancy attack.

Fix: Import from OZ contracts [ReentrancyGuard.sol] and use the modifier nonReentrant in the described functions.

Dev update: Fixed on package [c4ef925c03a14f301607a0cda53af68a8ad9ed26dc6dba571b1adb90bcb9d522]. Reentrancy guards were added.

3.2 Medium severity issues [2]

1. Unexpected behavior from the relayer

Contract: WrappedPrivateETH

Function: Unwrap() and completePrivateTransfer()

Problem: The relayer can get a big amount of fees in the transfer function and unwrap the procedure. The contract is only checking that the hash balance produced before is bigger than the amount + fees.

Fix: Set a maximum fee amount on the relayer.

Dev update: Fee limit will not be included per contract instead the interface will show the % and if it's big it will alert a Warning.

2. Incorrect function visibility

Contract: MerkleTreeWithHistory

Function: _insert()

Problem: Function should be internal and not accessible externally.

Fix: Change the function visibility to ‘internal’.

Dev update: Fixed on package [c4ef925c03a14f301607a0cda53af68a8ad9ed26dc6dba571b1adb90bcb9d522]. Visibility has changed.

3.3 Low severity issues [2]

1. Encrypted balances are not controlled

Contract: WrappedPrivateETH

Function: Unwrap() and completePrivateTransfer()

Problem: It’s possible to set random amounts in encrypted balances.

Fix: Verify when encryptedReceiverNewBalance is set or not use _privateEncryptedBalances.

Dev update: Fixed on package [c4ef925c03a14f301607a0cda53af68a8ad9ed26dc6dba571b1adb90bcb9d522].

2. Functions mistakenly placed in the wrong contract

Contract: MerkleTreeWithHistory

Function: dualMux() and verify

Problem: Both functions must be deleted from solidity contract and instead moved to merkletree.circom

Fix: Move to merkletree.circom

3.4 Informational issues [1]

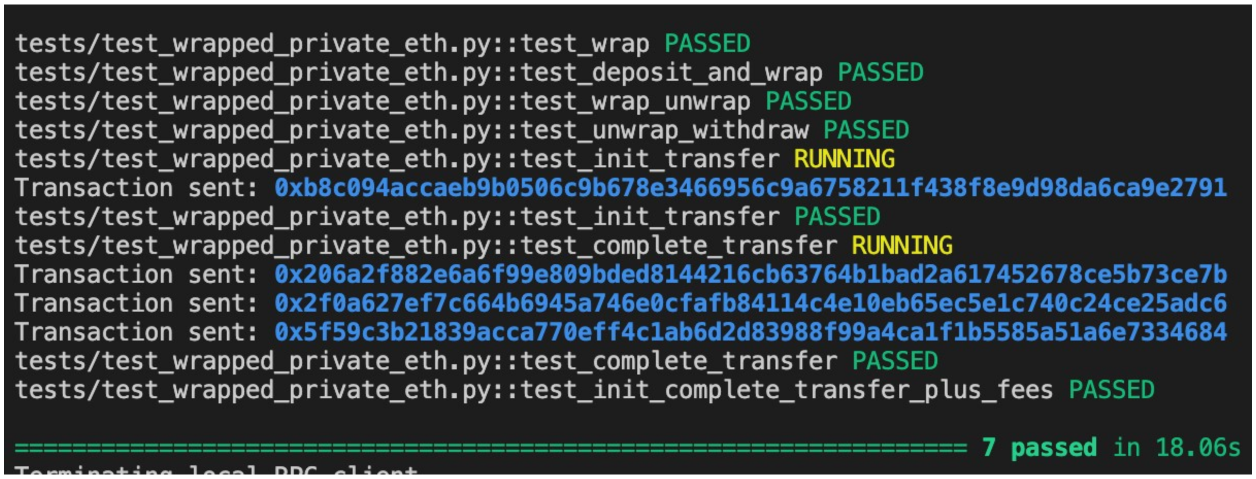
1. Malicious actors

We recommend having a way to block interactions with malicious actors whose funds come from hacks, scams or any type of illegal activity as well as monitor incoming requests.

4.0 Testing coverage - python

During the testing phase, custom use cases were written to cover all the logic of contracts in python language. *<u>*Check "5 Annexes" to see the testing code.</u>*

<u>FractalCash tests</u>



```
tests/test_wrapped_private_eth.py::test_wrap PASSED
tests/test_wrapped_private_eth.py::test_deposit_and_wrap PASSED
tests/test_wrapped_private_eth.py::test_wrap_unwrap PASSED
tests/test_wrapped_private_eth.py::test_unwrap_withdraw PASSED
tests/test_wrapped_private_eth.py::test_init_transfer RUNNING
Transaction sent: 0xb8c094accaeb9b0506c9b678e3466956c9a6758211f438f8e9d98da6ca9e2791
tests/test_wrapped_private_eth.py::test_init_transfer PASSED
tests/test_wrapped_private_eth.py::test_complete_transfer RUNNING
Transaction sent: 0x206a2f882e6a6f99e809bded8144216cb63764b1bad2a617452678ce5b73ce7b
Transaction sent: 0x2f0a627ef7c664b6945a746e0cfafb84114c4e10eb65ec5e1c740c24ce25adc6
Transaction sent: 0x5f59c3b21839acca770eff4c1ab6d2d83988f99a4ca1f1b5585a51a6e7334684
tests/test_wrapped_private_eth.py::test_complete_transfer PASSED
tests/test_wrapped_private_eth.py::test_init_complete_transfer_plus_fees PASSED

================================================================== 7 passed in 18.06s
Terminating local RPC client
```

5.0 Annexes

FractalCash testing:

```
from brownie import (
    reverts
)
from brownie.network import accounts
from eth_abi.packed import encode_packed

from scripts.helpful_scripts import (
    ZERO_ADDRESS,
    get_account,
)

from scripts.deploy import (
    deploy_wrapped_private_eth
)

def test_wrap(only_local):
    #arrange
    owner = get_account(0)
    other_account = get_account(1)
    weth = deploy_wrapped_private_eth(owner)

    deposit_amount = 2e18
    #assert
    tx = weth.deposit({"from": other_account, "value": deposit_amount})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"]["from"] == ZERO_ADDRESS
    assert tx.events["Transfer"]["to"] == other_account
    assert tx.events["Transfer"]["value"] == deposit_amount
    assert weth.balanceOf(other_account) == deposit_amount

    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x28eb372afa8304a706b729ec9c41569c04fe688917d8a25275e0cee7f309613d,0x02df4
80fd862c3371a2aceda1e7a46c81a045301c9ab0ea46be66960f55e285d],
[0x080e58a341061c09eb0a8d2e56467d07aa3aff7d5ec58f78e9be3478668324c2,
0x0a7c941f0d87ab9f47ae4d92cd264b26d2f1a9d0660158d4abc0593530933c26],

[0x14815b507f7f5e7127f8f0237496474d4e57c9f28b8876793dbfdbf4021d74a7,
0x04ee616476070e96a1c5671c724c9284d7041cc37e458c76c30288f054d0bddf],

[0x1195f7b91f179dc2c95ecea75b5cfc655c286bd1bc5963d4b9c1da10019ac917,
0x2b7d853d9137e54358b11e1ce6f0225b898a5f071890b28c72e7593030eb6376]
        )
    )
    inputs = [
        0x0000000000000000000000000000000000000000000000000de0b6b3a7640000,
# amount = 1000000000000000000
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceBefore
        0x26947269d91f6269642a7009cf49805a9cca0015d20ea976ab2e112e98b38ecf
# hashBalanceAfter
    ]
    new_encrypted_balance = encode_packed(['uint32'], [80000])
    assert weth.verifyProofWrap(proof, inputs) == True

    tx = weth.wrap(1e18, inputs[2], new_encrypted_balance, proof, {"from":
other_account})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"]["from"] == other_account
    assert tx.events["Transfer"]["to"] == ZERO_ADDRESS
    assert tx.events["Transfer"]["value"] == 1e18

def test_deposit_and_wrap(only_local):
    #arrange
    owner = get_account(0)
    other_account = get_account(1)
    weth = deploy_wrapped_private_eth(owner)

    deposit_amount = 2e18
    #assert
    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x15afd1e53782a1bc80d6f80de5761b6c6b7b1c411c6069cb435f969560fa656c,
0x259f37a35299fa75c17064f1b6be5bf8491b783ae3c9dbc0c590f4b2f706a125],
[0x0e1874c3eb46bc0cb0cacbee17d1011bc27281bffce741ffa3dccdb85ec586d3,
0x26d0e97865394032cf9f4ea694a826dc7b3595322414a6bde0bd0fb400c81d86],

[0x0cfd32c44321d073d9efb704970700265a3ea6e33455b1c72344e29f36ce6010,
0x17ddc30fc0a79cd46d2880ffa3608d58dca6f7a20c09d4b9a056f62f8bef7fea],

[0x1ff1fa4a54722992441aea668c21af9d6636ca42cc23bc14329cefe98c536080,
0x213a509356b365a45be632a26b6476345eb9037d44d8bebf782f04fee6fd2dac]
        )
    )
    inputs = [
        0x0000000000000000000000000000000000000000000000001bc16d674ec80000,
# amount = 2000000000000000000
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceBefore
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980
# hashBalanceAfter
    ]
    new_encrypted_balance = encode_packed(['uint32'], [80000])
    assert weth.verifyProofWrap(proof, inputs) == True

    tx = weth.depositAndWrap(inputs[2], new_encrypted_balance, proof,
{"from": other_account, "value": deposit_amount})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]["from"] == ZERO_ADDRESS
    assert tx.events["Transfer"][0]["to"] == other_account
    assert tx.events["Transfer"][0]["value"] == deposit_amount
    assert tx.events["Transfer"][1]["from"] == other_account
    assert tx.events["Transfer"][1]["to"] == ZERO_ADDRESS
    assert tx.events["Transfer"][1]["value"] == deposit_amount

def test_wrap_unwrap(only_local):
    #arrange
    owner = get_account(0)
    other_account = get_account(1)
    new_account = accounts.at('0x79B2f0CbED2a565C925A8b35f2B402710564F8a2',
force=True) #694778392256214003851701702023377900431359670434
    other_account.transfer(new_account, "2 ether")
    weth = deploy_wrapped_private_eth(owner)

    deposit_amount = 2e18
    #assert
    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x15afd1e53782a1bc80d6f80de5761b6c6b7b1c411c6069cb435f969560fa656c,
0x259f37a35299fa75c17064f1b6be5bf8491b783ae3c9dbc0c590f4b2f706a125],

[0x0e1874c3eb46bc0cb0cacbee17d1011bc27281bffce741ffa3dccdb85ec586d3,
0x26d0e97865394032cf9f4ea694a826dc7b3595322414a6bde0bd0fb400c81d86],

[0x0cfd32c44321d073d9efb704970700265a3ea6e33455b1c72344e29f36ce6010,
0x17ddc30fc0a79cd46d2880ffa3608d58dca6f7a20c09d4b9a056f62f8bef7fea],

[0x1ff1fa4a54722992441aea668c21af9d6636ca42cc23bc14329cefe98c536080,
0x213a509356b365a45be632a26b6476345eb9037d44d8bebf782f04fee6fd2dac]
        )
    )
    inputs = [
        0x0000000000000000000000000000000000000000000000001bc16d674ec80000,
# amount = 2000000000000000000
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceBefore = 0
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980
# hashBalanceAfter = 2000000000000000000
    ]
    new_encrypted_balance = encode_packed(['uint32'], [80000])
    weth.depositAndWrap(inputs[2], new_encrypted_balance, proof, {"from":
new_account, "value": deposit_amount})

    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x120de1588d6f1ad6efd233c8027cb5791f23418900569ee08fae62c281abca6c,
0x0d39197e91b5302d14634dae41f7807c19e4dc59808f2eefe9f182964d828972],

[0x087501aadd3c0a8b8392033a5434de1eb24f4be3fd4728d12a4144dbca16d0af,
0x1130bbebef9fd15f7ae0b2f6af0b39623bc0f0156d86c9d9765c903d8ba7b066],

[0x18115d6bff791404503b7fe7b4c0a7f8137f5c3bd9f49fa9b94314e97a51611f,
0x0be6c57759c89ddc168fc4cb70bedbd55914d9bcc2cfb2c8287ffe72c4dcd1a8],

[0x150ca5973512d055847ffd3ef87c8783a62fe67501445342303cd8780bf03500,
0x1aadceaed225ef2e6dff84cf1c21196b6b15e545687fb413aa4ea2bcf1994e61]
```

```
        )
    )
    inputs = [
        0x0000000000000000000000000000000000000000000000001bc16d674ec80000,
# amount = 2000000000000000000
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980,
# hashBalanceBefore = 2000000000000000000
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceAfter = 0
        0x0000000000000000000000000000000000000000000000000000000000000000,
# relayerFeeAmount = 0
        0x00000000000000000000000079b2f0cbed2a565c925a8b35f2b402710564f8a2,
# receiver = new_account
        0x00000000000000000000000034cbf414a4fb3ac1c61b49a80105e5721c9532b4
# encryptedAmount = uint(ripemd160(abi.encodePacked(abi.encode(amount)))
    ]
    assert weth.verifyProofUnwrap(proof, inputs) == True
    encrypted_balance = encode_packed(['uint'], [1000000000000000000])
    tx = weth.unwrap(deposit_amount, inputs[2], encrypted_balance, 0,
ZERO_ADDRESS, new_account, proof, {"from": other_account})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]["from"] == ZERO_ADDRESS
    assert tx.events["Transfer"][0]["to"] == new_account
    assert tx.events["Transfer"][0]["value"] == deposit_amount
    tx = weth.withdraw(deposit_amount//2, {"from": new_account})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]["from"] == new_account
    assert tx.events["Transfer"][0]["to"] == ZERO_ADDRESS
    assert tx.events["Transfer"][0]["value"] == deposit_amount//2

def test_unwrap_withdraw(only_local):
    #arrange
    owner = get_account(0)
    other_account = get_account(1)
    new_account = accounts.at('0x79B2f0CbED2a565C925A8b35f2B402710564F8a2',
force=True) #694778392256214003851701702023377900431359670434
    other_account.transfer(new_account, "2 ether")
    weth = deploy_wrapped_private_eth(owner)

    deposit_amount = 2e18
    #assert
    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x15afd1e53782a1bc80d6f80de5761b6c6b7b1c411c6069cb435f969560fa656c,
0x259f37a35299fa75c17064f1b6be5bf8491b783ae3c9dbc0c590f4b2f706a125],

[0x0e1874c3eb46bc0cb0cacbee17d1011bc27281bffce741ffa3dccdb85ec586d3,
0x26d0e97865394032cf9f4ea694a826dc7b3595322414a6bde0bd0fb400c81d86],

[0x0cfd32c44321d073d9efb704970700265a3ea6e33455b1c72344e29f36ce6010,
0x17ddc30fc0a79cd46d2880ffa3608d58dca6f7a20c09d4b9a056f62f8bef7fea],

[0x1ff1fa4a54722992441aea668c21af9d6636ca42cc23bc14329cefe98c536080,
0x213a509356b365a45be632a26b6476345eb9037d44d8bebf782f04fee6fd2dac]
        )
    )
    inputs = [
        0x00000000000000000000000000000000000000000000000001bc16d674ec80000,
# amount = 2000000000000000000
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceBefore = 0
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980
# hashBalanceAfter = 2000000000000000000
    ]
    new_encrypted_balance = encode_packed(['uint32'], [80000])
    weth.depositAndWrap(inputs[2], new_encrypted_balance, proof, {"from":
new_account, "value": deposit_amount})

    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x120de1588d6f1ad6efd233c8027cb5791f23418900569ee08fae62c281abca6c,
0x0d39197e91b5302d14634dae41f7807c19e4dc59808f2eefe9f182964d828972],

[0x087501aadd3c0a8b8392033a5434de1eb24f4be3fd4728d12a4144dbca16d0af,
0x1130bbebef9fd15f7ae0b2f6af0b39623bc0f0156d86c9d9765c903d8ba7b066],

[0x18115d6bff791404503b7fe7b4c0a7f8137f5c3bd9f49fa9b94314e97a51611f,
0x0be6c57759c89ddc168fc4cb70bedbd55914d9bcc2cfb2c8287ffe72c4dcd1a8],

[0x150ca5973512d055847ffd3ef87c8783a62fe67501445342303cd8780bf03500,
0x1aadceaed225ef2e6dff84cf1c21196b6b15e545687fb413aa4ea2bcf1994e61]
        )
    )
    inputs = [
        0x00000000000000000000000000000000000000000000001bc16d674ec80000,
# amount = 2000000000000000000
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980,
# hashBalanceBefore = 2000000000000000000
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceAfter = 0
        0x0000000000000000000000000000000000000000000000000000000000000000,
# relayerFeeAmount = 0
        0x00000000000000000000000079b2f0cbed2a565c925a8b35f2b402710564f8a2,
# receiver = new_account
        0x00000000000000000000000034cbf414a4fb3ac1c61b49a80105e5721c9532b4
# encryptedAmount = uint(ripemd160(abi.encodePacked(abi.encode(amount)))
    ]
    assert weth.verifyProofUnwrap(proof, inputs) == True
    encrypted_balance = encode_packed(['uint'], [1000000000000000000])
    tx = weth.unwrapAndWithdraw(deposit_amount, inputs[2],
encrypted_balance, 0, ZERO_ADDRESS, new_account, proof, {"from":
other_account})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]["from"] == ZERO_ADDRESS
    assert tx.events["Transfer"][0]["to"] == new_account
    assert tx.events["Transfer"][0]["value"] == deposit_amount
    assert tx.events["Transfer"][1]["from"] == new_account
    assert tx.events["Transfer"][1]["to"] == ZERO_ADDRESS
    assert tx.events["Transfer"][1]["value"] == deposit_amount

def test_init_transfer(only_local):
    #arrange
    owner = get_account(0)
    other_account = get_account(1)
    weth = deploy_wrapped_private_eth(owner)

    deposit_amount = 2e18
    #assert
    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x15afd1e53782a1bc80d6f80de5761b6c6b7b1c411c6069cb435f969560fa656c,
0x259f37a35299fa75c17064f1b6be5bf8491b783ae3c9dbc0c590f4b2f706a125],
[0x0e1874c3eb46bc0cb0cacbee17d1011bc27281bffce741ffa3dccdb85ec586d3,
0x26d0e97865394032cf9f4ea694a826dc7b3595322414a6bde0bd0fb400c81d86],

[0x0cfd32c44321d073d9efb704970700265a3ea6e33455b1c72344e29f36ce6010,
0x17ddc30fc0a79cd46d2880ffa3608d58dca6f7a20c09d4b9a056f62f8bef7fea],

[0x1ff1fa4a54722992441aea668c21af9d6636ca42cc23bc14329cefe98c536080,
0x213a509356b365a45be632a26b6476345eb9037d44d8bebf782f04fee6fd2dac]
        )
    )
    inputs = [
        0x000000000000000000000000000000000000000000000001bc16d674ec80000,
# amount = 2000000000000000000
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceBefore
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980
# hashBalanceAfter
    ]

    new_encrypted_balance = encode_packed(['uint32'], [80000])
    weth.depositAndWrap(inputs[2], new_encrypted_balance, proof, {"from":
other_account, "value": deposit_amount})

    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x0e882a6296af7e6174eb30002f38cf50ea94168b2224454b3eee0799a03df350,
0x1112c7c97b5140ea5057fe72e28036fbed98c3e01f0ae1d83f96acb4edeae284],

[0x2045ea8bddd71a91f7167e0ade7f488cc039a169be83937ef7cab8e4e01da502,
0x170944eb3af8effd8cc9d2b16bdd29b54795acb6c2b9b19edf992472eee71ae9],

[0x039a96d28b8497a83f1c4de9b53df855b894232c73af5c207225c716eb78c1fc,
0x0dbbb9465807f5cb047116efc819a95b6a4c748650d4000487f09c2685078184],

[0x157521de2f9488d3ec02916b03e0bc1a70b12b7db4ec6931101cf273d5eb3a03,
0x231976505f5f9787f69ab9880b3f1dc99c019b200057865e310087de30c40002]
        )
    )
    inputs = [
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980,
# hashBalanceBefore
```

```
        0x26947269d91f6269642a7009cf49805a9cca0015d20ea976ab2e112e98b38ecf,
# hashBalanceAfter
        0x2ee91d220fa314a75ac580b364e763e51816e33658c18e5382c2f1a0b8a3f8fc
# commitment
    ]
    assert weth.verifyProofInitTransfer(proof, inputs) == True

    secret = encode_packed(['uint32'], [1122334455])
    tx = weth.initiatePrivateTransfer(inputs[-1], secret,
new_encrypted_balance, inputs[1], proof, {"from": other_account})
    assert tx.events["PrivateTransferInitiated"] is not None

    with reverts("The commitment has been submitted"):
        weth.initiatePrivateTransfer(inputs[-1], secret,
new_encrypted_balance, inputs[1], proof, {"from": other_account})

def test_complete_transfer(only_local):
    #arrange
    owner = get_account(0)
    other_account = get_account(1)
    new_account = accounts.at('0x79B2f0CbED2a565C925A8b35f2B402710564F8a2',
force=True) #694778392256214003851701702023377900431359670434
    weth = deploy_wrapped_private_eth(owner)

    deposit_amount = 2e18
    #assert

    # DEPOSIT & WRAP

    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x15afd1e53782a1bc80d6f80de5761b6c6b7b1c411c6069cb435f969560fa656c,
0x259f37a35299fa75c17064f1b6be5bf8491b783ae3c9dbc0c590f4b2f706a125],

[0x0e1874c3eb46bc0cb0cacbee17d1011bc27281bffce741ffa3dccdb85ec586d3,
0x26d0e97865394032cf9f4ea694a826dc7b3595322414a6bde0bd0fb400c81d86],

[0x0cfd32c44321d073d9efb704970700265a3ea6e33455b1c72344e29f36ce6010,
0x17ddc30fc0a79cd46d2880ffa3608d58dca6f7a20c09d4b9a056f62f8bef7fea],

[0x1ff1fa4a54722992441aea668c21af9d6636ca42cc23bc14329cefe98c536080,
0x213a509356b365a45be632a26b6476345eb9037d44d8bebf782f04fee6fd2dac]
        )
    )
    inputs = [
        0x000000000000000000000000000000000000000000000001bc16d674ec80000,
# amount = 2000000000000000000
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceBefore
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980
# hashBalanceAfter
    ]

    new_encrypted_balance = encode_packed(['uint32'], [80000])

    weth.depositAndWrap(inputs[2], new_encrypted_balance, proof, {"from":
other_account, "value": deposit_amount})

    # INITITIATE PRIVATE TRANSFER

    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x0e882a6296af7e6174eb30002f38cf50ea94168b2224454b3eee0799a03df350,
0x1112c7c97b5140ea5057fe72e28036fbed98c3e01f0ae1d83f96acb4edeae284],

[0x2045ea8bddd71a91f7167e0ade7f488cc039a169be83937ef7cab8e4e01da502,
0x170944eb3af8effd8cc9d2b16bdd29b54795acb6c2b9b19edf992472eee71ae9],

[0x039a96d28b8497a83f1c4de9b53df855b894232c73af5c207225c716eb78c1fc,
0x0dbbb9465807f5cb047116efc819a95b6a4c748650d4000487f09c2685078184],

[0x157521de2f9488d3ec02916b03e0bc1a70b12b7db4ec6931101cf273d5eb3a03,
0x231976505f5f9787f69ab9880b3f1dc99c019b200057865e310087de30c40002]
        )
    )
    inputs = [
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980,
# hashBalanceBefore = 2 eth
        0x26947269d91f6269642a7009cf49805a9cca0015d20ea976ab2e112e98b38ecf,
# hashBalanceAfter = 1 eth
        0x2ee91d220fa314a75ac580b364e763e51816e33658c18e5382c2f1a0b8a3f8fc
# commitment
```

```
    ]

    secret = encode_packed(['uint32'], [1122334455])
    tx = weth.initiatePrivateTransfer(inputs[-1], secret,
new_encrypted_balance, inputs[1], proof, {"from": other_account})
    assert tx.events["PrivateTransferInitiated"] is not None

    # COMPLETE TRANSFER

    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x2f03023382e4ce65ee5d23ae6c7012ebe06e2f5b3137fd57aea6241f0d955df9,
0x070b5938e2fc702b3efb3c4d4699021ffa79bde4e742d2aa92fdff27cc93ed11],

[0x1000b7bcd1dc0c87b729c12c43212f4c866ec15c866e0c9fc08ae16aa5fcc747,
0x2d7bc5a07d22b70869cd75d6f419991496a6efa390f14c74d2fc74385affa326],

[0x0e3742697bf027bacb24946047242b349a3cb7c3273fbda096e2c5b5d6f72da8,
0x11c80994347864ef9e7f1dd2d58a06e8de553b02da96a1a31e0b6225dd8420ba],

[0x018501bbddc7eeade3d5377ad334cdf6a96496198f4f7363c7d6daef605eac90,
0x04a8d53c77c8da3b4c3bec68b5c6967dbe675d0637e6ad0c031bc0bb1fad5fa0]
        )
    )
    inputs = [
        0x2a9a0de14b2190f10507071335c2a853c9a47e837341c60475a97750a5eafdf0,
# root
        0x02b81f0bc0f07e7877a57ce9ea0ba293164c6d7a1fa1a1c7211bfc191b50313c,
# nullifierHash
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceBefore
        0x26947269d91f6269642a7009cf49805a9cca0015d20ea976ab2e112e98b38ecf,
# hashBalanceAfter
        0x0000000000000000000000000000000000000000000000000000000000000000,
# relayerFeeAmount
        0x00000000000000000000000079b2f0cbed2a565c925a8b35f2b402710564f8a2,
# receiver
        0x00000000000000000000000034cbf414a4fb3ac1c61b49a80105e5721c9532b4
# encryptedAmount
    ]
    assert weth.verifyProofCompleteTransfer(proof, inputs) == True

    invented_root =
0x0000000000000000000000000000000000000000000000000000000000000001
    with reverts("Cannot find your merkle root"):
        weth.completePrivateTransfer(invented_root, inputs[1], inputs[3],
new_encrypted_balance,
                                            inputs[4], owner, new_account, proof,
{"from": other_account})

    encrypted_balance = encode_packed(['uint'], [1000000000000000000])
    tx = weth.completePrivateTransfer(inputs[0], inputs[1], inputs[3],
encrypted_balance,
                                            inputs[4], owner, new_account, proof,
{"from": other_account})
    assert tx.events["PrivateTransferCompleted"] is not None
    assert tx.events["PrivateTransferCompleted"][0]["receiver"] ==
new_account
    assert
weth.bytes32ToUint(tx.events["PrivateTransferCompleted"][0]["nullifierHash"
]) == inputs[1]
    assert weth.bytes32ToUint(weth.getHashBalance(new_account)) ==
inputs[3]

    with reverts("This transfer has already been completed"):
        weth.completePrivateTransfer(inputs[0], inputs[1], inputs[3],
encrypted_balance,
                                            inputs[4], owner, new_account, proof,
{"from": other_account})

    # UNWRAP + FEES
    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x0ae04b125604525ef1e5f00fb24d7d538bf247684ca58baf56317e17378409d3,
0x0b59569f4540215374bf838235918f6485d17eb0d77d8a2fab2b2c81a379a50a],

[0x154edf318d0fc1f554710628f0b257ea55d438107906c7aac6edb65ca2d00964,
0x092e90d7c09771173397744c9a19362a536b0609379fc900af3463286901f9f1],

[0x2ec6b4ee5650d37294b8b9fa1b203a0be96ab971d304e938bcc834c4490952bc,
0x1a5b65c5a3521188e1aba7dae676b639565952a2f800df01190a10fab32fb3b7],
[0x1717dc2468d76a4e6aeb91a042e50121d650c8521e526e5f65c49a526ee90441,
0x1a2809e9a7fb2eb1d7fc8051927a823c7b0553cd37a523b6697258246e7604fd]
        )
    )
    inputs = [
        0x000000000000000000000000000000000000000000000000c7d713b49da0000,
# amount = 900000000000000000
        0x26947269d91f6269642a7009cf49805a9cca0015d20ea976ab2e112e98b38ecf,
# hashBalanceBefore = 1000000000000000000
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceAfter = 0
        0x00000000000000000000000000000000000000000000000016345785d8a0000,
# relayerFeeAmount = 100000000000000000
        0x00000000000000000000000079b2f0cbed2a565c925a8b35f2b402710564f8a2,
# receiver = new_account
        0x00000000000000000000000034cbf414a4fb3ac1c61b49a80105e5721c9532b4
# encryptedAmount = uint(ripemd160(abi.encodePacked(abi.encode(amount)))
    ]
    encrypted_balance = encode_packed(['uint'], [1000000000000000000])
    with reverts(): # invalid amount
        weth.unwrap(1e18, inputs[2], encrypted_balance, 100000000000000000,
owner, new_account, proof, {"from": other_account})
    tx = weth.unwrap(900000000000000000, inputs[2], encrypted_balance,
100000000000000000, owner, new_account, proof, {"from": other_account})
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]["from"] == ZERO_ADDRESS
    assert tx.events["Transfer"][0]["to"] == new_account
    assert tx.events["Transfer"][0]["value"] == 900000000000000000
    assert tx.events["Transfer"][1]["from"] == ZERO_ADDRESS
    assert tx.events["Transfer"][1]["to"] == owner
    assert tx.events["Transfer"][1]["value"] == 100000000000000000

def test_init_complete_transfer_plus_fees(only_local):
    #arrange
    owner = get_account(0)
    other_account = get_account(1)
    new_account = accounts.at('0x79B2f0CbED2a565C925A8b35f2B402710564F8a2',
force=True) #694778392256214003851701702023377900431359670434
    weth = deploy_wrapped_private_eth(owner)

    deposit_amount = 2e18
    #assert

    # DEPOSIT & WRAP
    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x15afd1e53782a1bc80d6f80de5761b6c6b7b1c411c6069cb435f969560fa656c,
0x259f37a35299fa75c17064f1b6be5bf8491b783ae3c9dbc0c590f4b2f706a125],

[0x0e1874c3eb46bc0cb0cacbee17d1011bc27281bffce741ffa3dccdb85ec586d3,
0x26d0e97865394032cf9f4ea694a826dc7b3595322414a6bde0bd0fb400c81d86],

[0x0cfd32c44321d073d9efb704970700265a3ea6e33455b1c72344e29f36ce6010,
0x17ddc30fc0a79cd46d2880ffa3608d58dca6f7a20c09d4b9a056f62f8bef7fea],

[0x1ff1fa4a54722992441aea668c21af9d6636ca42cc23bc14329cefe98c536080,
0x213a509356b365a45be632a26b6476345eb9037d44d8bebf782f04fee6fd2dac]
        )
    )
    inputs = [
        0x0000000000000000000000000000000000000000000000001bc16d674ec80000,
# amount = 2000000000000000000
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceBefore
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980
# hashBalanceAfter
    ]
    new_encrypted_balance = encode_packed(['uint32'], [80000])
    weth.depositAndWrap(inputs[2], new_encrypted_balance, proof, {"from":
other_account, "value": deposit_amount})

    # INITITIATE PRIVATE TRANSFER
    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x0e882a6296af7e6174eb30002f38cf50ea94168b2224454b3eee0799a03df350,
0x1112c7c97b5140ea5057fe72e28036fbed98c3e01f0ae1d83f96acb4edeae284],

[0x2045ea8bddd71a91f7167e0ade7f488cc039a169be83937ef7cab8e4e01da502,
0x170944eb3af8effd8cc9d2b16bdd29b54795acb6c2b9b19edf992472eee71ae9],

[0x039a96d28b8497a83f1c4de9b53df855b894232c73af5c207225c716eb78c1fc,
0x0dbbb9465807f5cb047116efc819a95b6a4c748650d4000487f09c2685078184],

[0x157521de2f9488d3ec02916b03e0bc1a70b12b7db4ec6931101cf273d5eb3a03,
0x231976505f5f9787f69ab9880b3f1dc99c019b200057865e310087de30c40002]
        )
    )
    inputs = [
        0x0f9d335866f9c29a56f0ae916ac94c96f2e328eb5534618ce5ec7672011cf980,
# hashBalanceBefore = 2 eth
        0x26947269d91f6269642a7009cf49805a9cca0015d20ea976ab2e112e98b38ecf,
# hashBalanceAfter = 1 eth
        0x2ee91d220fa314a75ac580b364e763e51816e33658c18e5382c2f1a0b8a3f8fc
# commitment
    ]
    secret = encode_packed(['uint32'], [1122334455])
    tx = weth.initiatePrivateTransfer(inputs[-1], secret,
new_encrypted_balance, inputs[1], proof, {"from": other_account})
    assert tx.events["PrivateTransferInitiated"] is not None

    # COMPLETE TRANSFER + FEES
    proof = encode_packed(['int[]', 'int[]','int[]','int[]'],
        (

[0x26ae95c97f5db34317bd92f0723da6dbb4675a827faf2fbd8cff6076717a10df,
0x2daa2bd3a70d644a9c20ac8a80c5535fa228fb28ebcc4e26d4c047096f878346],

[0x16812cb8f22adab2f9cfbb54e4e49a322749e442c55cfdc7651c167638bb6f1e,
0x19f3aec4a2911f5bb72e97c4df4f59d0de62f2a0b6cf3122928a824376b2ae96],

[0x06b86f4d9ff60cf5fc9b6dc83953d22ec7649538f1e8e7710ba291412bfe2cab,
0x0cb91d0c2e9926fdf23a26a438cf9738036dc1a94036927b503a2ab96c5bc122],

[0x17c00d9cbc397d7c223115e6593c55b51b1b599694c727fa8d8cb8640af1800a,
0x2b9955b39a454520da209e2b916307dc204810dc52f4d42b5292c773e941dc42]
        )
    )
    inputs = [
        0x2a9a0de14b2190f10507071335c2a853c9a47e837341c60475a97750a5eafdf0,
# root
        0x02b81f0bc0f07e7877a57ce9ea0ba293164c6d7a1fa1a1c7211bfc191b50313c,
# nullifierHash
        0x0c0be0bd5964e86230fcd3b9db389a99857870d8e23c4cbd737a1d451239bf5d,
# hashBalanceBefore
        0x21f768d01121fd05f94d65f4d6f672de3e31650de2a5cd735bc2fee76d747697,
# hashBalanceAfter = 900000000000000000
        0x0000000000000000000000000000000000000000000000000016345785d8a0000,
# relayerFeeAmount = 100000000000000000
        0x000000000000000000000000079b2f0cbed2a565c925a8b35f2b402710564f8a2,
# receiver
        0x000000000000000000000000034cbf414a4fb3ac1c61b49a80105e5721c9532b4
# encryptedAmount
    ]
    assert weth.verifyProofCompleteTransfer(proof, inputs) == True

    encrypted_balance = encode_packed(['uint'], [1000000000000000000])
    tx = weth.completePrivateTransfer(inputs[0], inputs[1], inputs[3],
encrypted_balance,
                                      inputs[4], owner, new_account, proof,
{"from": other_account})
    assert tx.events["PrivateTransferCompleted"] is not None
    assert tx.events["PrivateTransferCompleted"][0]["receiver"] ==
new_account
    assert
weth.bytes32ToUint(tx.events["PrivateTransferCompleted"][0]["nullifierHash"
]) == inputs[1]
    assert weth.bytes32ToUint(weth.getHashBalance(new_account)) ==
inputs[3]
    assert tx.events["Transfer"] is not None
    assert tx.events["Transfer"][0]["from"] == ZERO_ADDRESS
    assert tx.events["Transfer"][0]["to"] == owner
    assert tx.events["Transfer"][0]["value"] == 100000000000000000
```

6.0 Summary of the audit

High/medium vulnerabilities were fixed by the development team. Contracts can be deployed to mainnet.