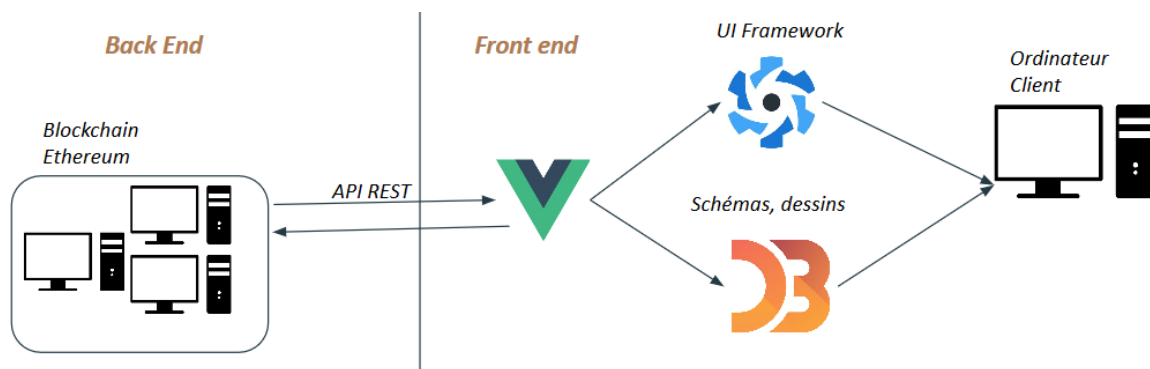


Illustration de scénario d'attaque sur la blockchain

Documentation

Ce document regroupe les informations nécessaires à l'opération du projet, ainsi que les opérations de version et de configuration.

Architecture du projet



Ce document exposera, en deux parties, la documentation relative à l'architecture de la partie blockchain et à l'architecture de la partie visualisation.

I – Partie Blockchain

a) Prérequis

Pour pouvoir opérer un nœud de blockchain complètement fonctionnel, il est nécessaire d'avoir à disposition 4Go de RAM pour les opérations de minage.

Si votre nœud n'est pas destiné à utiliser un mineur, cette considération n'a pas d'importance.

b) Version du client

La technologie choisie est Ethereum. Nous avons travaillé avec le client Go, dans sa version **1.8.0**. La version de Go utilisée était **1.7.4**.

Nous vous conseillons de repartir d'une installation du client Go-Ethereum et de Go à la dernière version. Les porteurs du projet Ethereum prévoient de finaliser une nouvelle bibliothèque

permettant l'interaction depuis du code javascript avec différents éléments de la blockchain, compatible avec celle utilisée au sein du projet. Il y a donc peu de chance pour que les API exposées par le client Go soient modifiées dans le futur proche.

Installation de Go : <https://golang.org/doc/install>

Installation de Go-Ethereum : <https://github.com/ethereum/go-ethereum> (see Building the Source)

c) Installation d'une blockchain privée

Pour ce projet, nous avons choisi d'opérer notre propre blockchain, pour ne pas dépenser d'Ether sur le vrai réseau Ethereum, et pour ne pas y publier de contrats vulnérables.

Ce paragraphe indique comment préparer et mettre en place un réseau privé de nœuds Ethereum.

- *Configuration et démarrage du premier nœud*

Les nœuds Ethereum nécessitent, pour fonctionner, un dossier dans lequel sera stocké l'état de la blockchain. Il est référencé par l'appellation « datadir » dans la documentation du client Ethereum. Lors du premier lancement de la blockchain, il faut l'initialiser par un processus particulier, à l'aide d'un fichier de configuration, nommé par défaut genesis.json.

En voici un exemple :

```
{
  "config":{
    "chainId":15,
    "homesteadBlock":0,
    "eip155Block":0,
    "eip158Block":0
  },
  "difficulty": "0x400",
  "gasLimit": "0x8000000",
  "alloc": {
    "d1916786838e320f09d5de3f8d5927f78bc6ce4":{"balance":"300000"}
  },
  "nonce": "0x0000042",
  "timestamp": "0",
  "parentHash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "mixhash": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "coinbase": "0x3333333333333333333333333333333333333333333333333333333333333333"
}
```

Les champs notables sont :

chainId : Identifiant de la chaîne. Sera utilisé pour le lancement de la blockchain en ligne de commande.

difficulty : difficulté à miner, pour les premiers blocs. Impacte la vitesse de minage. Avec la valeur fournie, un bloc vide est miné en environ 10 secondes.

gasLimit : Limite haute pour la valeur de gaz nécessaire pour miner un bloc. Si cette valeur dépasse gasLimit, la transaction est rejetée.

alloc : chaque entrée dans alloc permet de donner à un compte arbitraire (qui sera créé si il le faut), un montant de wei. Cependant un compte créé de cette façon sera inutilisable puisqu'il n'aura pas de mot de passe ! Nous proposons plus loin une solution à ce problème.

coinbase : adresse du compte qui recevra le gaz des transactions minées par les mineurs opérant sur le nœud.

Il faut ensuite initialiser la blockchain (peu importe les adresses cibles du champ alloc dans un premier temps).

```
geth init --datadir <datadir> genesis.json
```

Une fois la blockchain initialisée, nous allons la faire tourner et créer un compte que nous pourrons utiliser sur notre blockchain. Cela créera un fichier correspondant à ce compte dans le dossier « datadir », et si nous gardons ce fichier nous pouvons initialiser une nouvelle blockchain, dans laquelle ce compte sera utilisable et que nous pourrons créditer dans le fichier **genesis**.

Au travail :

```
geth --nodiscover --datadir <datadir> --networkid 15 --rpc --rpcapi  
personal,web3,eth,accounts,admin,miner,net,txpool --identity block1 --  
rpccorsdomain "*" --rpcaddr <ip> --mine --gasprice "1"
```

Cette commande permet de lancer la blockchain. Pour interagir avec, dans un nouveau terminal :

```
geth attach <datadir>/geth.ipc
```

Vous obtenez alors une console javascript permettant d'administrer votre blockchain. Créons donc un nouveau compte :

```
> personal.newAccount("yourPassword")
```

Une fois votre compte créé, notez l'adresse obtenue (celle de votre compte), puis coupez votre blockchain. Nous allons supprimer toutes les données sauf celles correspondant à notre compte, modifier le fichier **genesis** et initialiser à nouveau notre blockchain :

```
$ cd <datadir>  
$ rm -rf geth/
```

Pour modifier le fichier **genesis**, il suffit de remplacer l'adresse contenue dans le champ alloc par celle obtenue lors de la création de compte. Le fichier **genesis** ainsi obtenu est celui que vous allez garder et utiliser pour l'initialisation sur les autres nœuds.

Pour finir, réinitialisez et lancez votre blockchain :

```
$ geth init --datadir <datadir> genesis.json
```

```
$ geth --nodiscover --datadir <datadir> --networkid 15 --rpc --rpcapi  
personal,web3,eth,accounts,admin,miner,net,txpool --identity block1 --  
rpccorsdomain "*" --rpcaddr <ip> --mine --gasprice "1"
```

Votre premier nœud est prêt.

- *Configuration des nœuds suivant*

Pour les autres nœuds, il suffit de récupérer le fichier **genesis** utilisé à la dernière étape réalisée pour le premier nœud, et d'initialiser et lancer la blockchain.

```
$ geth init --datadir <datadir> genesis.json
```

```
$ geth --nodiscover --datadir <datadir> --networkid 15 --rpc --rpcapi
personal,web3,eth,accounts,admin,miner,net,txpool --identity block1 --
rpccorsdomain "*" --rpcaddr <ip> --mine --gasprice "1"
```

- *Connexion des nœuds*

Les nœuds ne sont pas connectés pour l'instant. Pour remédier à cela, il faut en récupérer les identifiants ou **enode**.

Pour cela, il faut se connecter à la console javascript d'administration comme précédemment :

```
$ geth attach <datadir>/geth.ipc
```

Une fois dans la console :

```
> admin.nodeInfo()
```

Le champ **enode** de la réponse est celui qui nous intéresse. Par exemple :

```
enode:
"enode://44826a5d6a55f88a18298bca4773fca5749cdc3a5c9f308aa7d810e9b31123f3e7
c5fba0b1d70aac5308426f47df2a128a6747040a3815cc7dd7167d03be320d@[::]:30303"
```

Pour finir de préparer la suite, pour chaque nœud, il faut remplacer dans l'enode les caractères `[::]` par l'adresse IP de l'interface réseau sur laquelle la blockchain sera exposée.

Notez le résultat pour chaque nœud du réseau.

Sur chaque nœud, il est possible de créer un fichier `<datadir>/geth/static-nodes.json` duquel seront tiré tous les pairs du nœud au démarrage de la blockchain.

Il faut y ajouter dans un tableau toutes les enodes des nœuds que vous voulez être pairs de votre nœud.

Exemple :

```
[
"enode://f4642fa65af50cfdea8fa7414a5def7bb7991478b768e296f5e4a54e8b995de102
e0ceae2e826f293c481b5325f89be6d207b003382e18a8ecba66fbaf6416c0@33.4.2.1:303
03",
"enode://pubkey@ip:port"
]
```

Pour un grand nombre de nœuds, il est possible de s'intéresser à la notion de **bootnode** dans la documentation d'Ethereum. Attention cependant, il faut laisser le nœud découvrir le reste de son réseau pour pouvoir utiliser cette méthode. Il vaut mieux avoir un identifiant de réseau unique pour être certain que votre nœud ne se connectera pas à un nœud inconnu.

d) Explication de la commande de lancement

Pour rappel, la commande de lancement est la suivante :

```
$ geth --nodiscover --datadir <datadir> --networkid 15 --rpc --rpcapi  
personal,web3,eth,accounts,admin,miner,net,txpool --identity block1 --  
rpccorsdomain "*" --rpcaddr <ip> --mine --gasprice "1"
```

Voici une explication rapide des options :

nodiscover : cette option garantit que le nœud ne va pas chercher à se connecter seul à des nœuds qui seraient potentiellement exposés, et dont le **networkid** est le même.

datadir : chemin vers le dossier de base de données de la blockchain

networkid : identifiant du réseau

rpc : ouverture de l'endpoint RPC-JSON

rpcapi : ouverture de modules disponibles via l'endpoint RPC-JSON

identity : surnom du nœud dans l'architecture

rpccorsdomain : domaine via lequel l'endpoint RPC-JSON sera accessible.

rpcaddr : adresse IP à laquelle sera disponible l'endpoint RPC-JSON (va de pair avec rpcport, pas utilisé ici)

mine : lancement automatique du mineur

gasprice : prix du gaz en wei.

e) Endpoint RPC-JSON et API

Ethereum propose une API RPC-JSON ouverte par défaut sur le port 8545. L'ouverture de cette API est facultative. Elle est entièrement documentée sur la page du client geth :

<https://github.com/ethereum/wiki/wiki/JSON-RPC>

II – Partie Visualisation

Comme spécifié dans le schéma d'architecture, l'interface est réalisée avec **VueJS** et **Quasar** pour la majorité du projet. **D3Js** est utilisé pour le tracé de graphes. La gestion des dépendances a été réalisée avec **NPM**.

a) Dépendances

Comme souvent avec le système de gestion de dépendances NPM, de nombreuses dépendances ont été installées dans le fichier package.json du projet de l'interface et ne seront donc pas toutes mentionnées.

Une partie du lien avec la blockchain a été réalisée à l'aide du module **Web3 (version 0.20.5)**. Ce module permet, au sein d'un script JS, de créer un objet permettant d'interagir avec différents objets de la blockchain (notamment les *smart contracts*) à partir d'un endpoint RPC-JSON ouvert.

La documentation de ce module est disponible ici :












<https://github.com/ethereum/web3.js/>

Attention ! La version **1.0** est en cours de préparation et sa documentation est déjà disponible pour que les développeurs puissent anticiper le changement. Cependant, il ne faut pas encore s'y fier.

b) Architecture du code

Le code de l'interface se trouve dans le dossier `web-plaform/src/`.

On y trouve les dossiers suivants :

| | |
|---|---|
|  <code>assets</code> | <u>Assets</u> : contient les ressources utilisées par l'interface (images, données factices de test...) |
|  <code>components</code> | <u>Components</u> : contient les composants VueJS, le code qui sera chargé suivant la page de l'interface chargée |
|  <code>services</code> | <u>Services</u> : services http, web3 utilisés par les composants pour communiquer |
|  <code>statics</code> | <u>Statics</u> : logos... |
|  <code>store</code> | <u>Store</u> : fichier où sont définis les variables qui définissent l'état de l'interface |
|  <code>structures</code> | <u>Structures</u> : structures de données notables dans le fonctionnement de l'interface |
|  <code>themes</code> | <u>Themes</u> : fichiers styl, utilisés par Quasar. |
|  <code>App.vue</code> | <u>App.vue</u> : container dans lequel seront chargés les composants |
|  <code>index.html</code> | <u>Index.html</u> : fichier d'entrée de l'application |
|  <code>main.js</code> | <u>Router.js</u> : permet la navigation dans l'interface |
|  <code>router.js</code> | |

Main.js : script gérant le chargement des composants, etc...

c) Lancement de l'interface

Pour pouvoir utiliser l'interface, il faut installer :

- Le client quasar ;
- Les dépendances.

Pour installer le client quasar :

```
$ sudo npm install -g quasar-cli
```

Ou

```
$ npm install quasar-cli
```

La première solution réalise l'installation du client quasar globalement sur la machine, ce qui permet de ne pas avoir à le refaire pour un autre projet utilisant quasar.

Pour installer les dépendances :

```
$ npm install
```

Enfin, pour lancer l'interface (sur le port 8080)

```
$ quasar dev
```