Tópicos em Segurança da Informação
Aula Mineração(Cont.)

Ivan Sendin

FACOM - Universidade Federal de Uberlandia
ivansendin@yahoo.com,sendin@ufu.br

4 de junho de 2024

- Mineradores mantem a blockchain
- Corretude e imutabilidade
- Ganham para isso
- Não precisamos confiar na honestidade deles

- Fork
- CPU,GPU, ASIC e Nuvem
- Pool
- PoS

Selfish Mining

- Desvio do Protocolo
- "Incentive Compatible"
  Teoria dos Jogos
- "block withhold"
- Minero o bloco n e não conto para ninguem!!!
- ..até minerar o bloco $n + 1$
- Consequencias?!?

Selfish Mining

- Consequencias?!?
- Aumento as chances de minerar o $n + 1$
- "Arrisco" o bloco n
- Diversos estudos para detectar e impedir este tipo de comportamento....

TCC do Guilherme Henrique Santos

```
import random as R

class SimpleMiner:
    def __init__(self,id,p):
        self.power =p
        self.id = id
        self.mined=0

    def tryMine(self,wpf):
        if R.random()<=(self.power/wpf):
            self.mined+=1

class World:
    def __init__(self,powers,F=10.0):
        self.miners = [SimpleMiner(i+1,powers[i]) for i in range(len(powers))]
        self.wpf = sum(powers)*F

    def step(self):
        for m in self.miners:
            m.tryMine(self.wpf)

    def printStat(self):
        for m in self.miners:
            print(m.id,m.power,m.mined,m.mined/m.power)
```

```
powers = [1,2,3,4,5,6,7,8,9,10,10,10,10,10,10,10,10,10,10,5]
world = World(powers)

for i in range(100000):
    world.step()

world.printStat()
```

```
1 1 76 76.0
2 2 114 57.0
3 3 176 58.666666666666664
4 4 261 65.25
5 5 299 59.8
6 6 412 68.66666666666667
7 7 448 64.0
8 8 488 61.0
9 9 555 61.666666666666664
10 10 620 62.0
11 10 616 61.6
12 10 632 63.2
13 10 604 60.4
14 10 620 62.0
15 10 598 59.8
16 10 637 63.7
17 10 599 59.9
18 10 628 62.8
19 10 602 60.2
20 10 619 61.9
21 5 302 60.4
```

```
#for i in range(10000000):
1 1 6236 6236.0
2 2 12600 6300.0
3 3 18742 6247.333333333333
4 4 24982 6245.5
5 5 31206 6241.2
6 6 37654 6275.666666666667
7 7 43785 6255.0
8 8 49793 6224.125
9 9 56298 6255.333333333333
10 10 62698 6269.8
11 10 62573 6257.3
12 10 62815 6281.5
13 10 62481 6248.1
14 10 62416 6241.6
15 10 63156 6315.6
16 10 62552 6255.2
17 10 62388 6238.8
18 10 62089 6208.9
19 10 62497 6249.7
20 10 62546 6254.6
21 5 31201 6240.2
```

Esta faltando...

- A rede
- A blockchain

```
class SimpleMiner:

    def __init__(self,id,p):
        self.power =p
        self.id = id
        self.ng = []
        self.blockchain = [0]
        self.needupdate = False

    def tryMine(self,wpf):
        if R.random()<=(self.power/wpf):
            self.blockchain = self.blockchain[:]
            self.blockchain.append(self.id)
            self.needupdate = True
```

```
def broadcast(self):
        if self.needupdate:
            for _ng in self.ng:
                _ng.receive(self.blockchain)
            self.needupdate = False

def receive(self,blockchain):
        if (len(blockchain)> len(self.blockchain)):
            self.blockchain = blockchain
            self.needupdate = True

def addNeighbor(self, m):
        self.ng.append(m)
```

```
class World:
    def __init__(self,powers,F=10.0):
        self.miners = [SimpleMiner(i+1,powers[i]) for i in range(len(powers))]
        self.wpf = sum(powers)*F*1.0
        for m in self.miners:
            for i in range(5):
                vz = R.choice(self.miners)
                while (vz.id == m.id):
                    vz = R.choice(self.miners)
                m.addNeighbor(vz)

    def step(self):
        for m in self.miners:
            m.tryMine(self.wpf)
        for m in self.miners:
            m.broadcast()


    def balance(self):
        r = []
        for m in self.miners:
            r.append([m.id,m.power, m.blockchain.count(m.id)])
        return r
```

```
def getFork(self):
      forks = {}
      longest = 3
      for m in self.miners:
          if len(m.blockchain)>longest:
              forks={}
              longest  = len(m.blockchain)
          if len(m.blockchain)==longest:
              # a simple trick to concat 2 integers
              forks[m.blockchain[-1] + m.blockchain[-2]*1000]=True
      return forks.keys()
```

Função bem simples...qualitativo. Uma análise quantitativa seria interessante.

```
 powers = [1,2,3,4,5,6,7,8,9,10,10,10,10,10,10,10,10,10,10,5]
world = World(powers)

for i in range(100000):
    world.step()
    forks =  world.getFork()
    if (len(forks)>1):
        print i,forks

b = world.balance()
for _b in b:
    print _b[0],_b[1],_b[2],_b[2]/(_b[1]*1.0)


for m in world.miners:
    print(m.blockchain[-10:])
```

Inúmeras bifurcações e "trifurcações"...com diversas profundidades...

```
1 1 6199 6199.0
2 2 12153 6076.5
3 3 17333 5777.666666666667
4 4 24502 6125.5
5 5 30544 6108.8
6 6 35446 5907.666666666667
7 7 41575 5939.285714285715
8 8 47533 5941.625
9 9 52892 5876.888888888889
10 10 59927 5992.7
11 10 58760 5876.0
12 10 59144 5914.4
13 10 58418 5841.8
14 10 59322 5932.2
15 10 58501 5850.1
16 10 58272 5827.2
17 10 62443 6244.3
18 10 57555 5755.5
19 10 57808 5780.8
20 10 57391 5739.1
21 5 28549 5709.8
```

```
class World:
    def __init__(self,powers,F=10.0):
        self.miners = [SimpleMiner(i+1,powers[i][0],powers[i][1]) for i in range(len(powers))
        self.wpf = sum([p[0] for p in powers])*F*1.0
        for m in self.miners:
            for i in range(m.ngsize):
                vz = R.choice(self.miners)
                while (vz.id == m.id):
                    vz = R.choice(self.miners)
                m.addNeighbor(vz)

powers = [ [10,5], [10,5], [10,5], [10,5], [10,5], [10,5], [10,2], [10,2]]
```

```
1 10 11705 1170.5
2 10 12229 1222.9
3 10 12174 1217.4
4 10 11724 1172.4
5 10 11639 1163.9
6 10 11437 1143.7
7 10 11323 1132.3
8 10 11521 1152.1
```

Convencidos??

```
1 10 11941 1194.1
2 10 12102 1210.2
3 10 12151 1215.1
4 10 11600 1160.0
5 10 12020 1202.0
6 10 11894 1189.4
7 10 10452 1045.2
8 10 11478 1147.8
```

OK...falta um teste estatístico!

```
class FraudulentMiner:

...
    def tryMine(self,wpf):
        if R.random()<=(self.power/wpf):
            self.blockchain = self.blockchain[:]
            self.blockchain.append(-self.id)
            self.needupdate = True


class SimpleMiner:

   def receive(self,blockchain):
        if (len(blockchain)> len(self.blockchain) and blockchain[-1]>0):
            self.blockchain = blockchain
            self.needupdate = True
```

```
class World:
    def __init__(self,powers,fraudulent=[],F=10.0):
        self.miners = [SimpleMiner(i+1,powers[i][0],powers[i][1]) for i in range(len(powers))
        lastid = len(self.miners)+1
        self.miners.extend([FraudulentMiner(i+lastid,fraudulent[i][0],fraudulent[i][1]) for i
        ...

powers = [ [10,5], [10,5], [10,5], [10,5], [10,5], [10,5], [10,5], [10,5]]
fraud = [ [20,5]]
world = World(powers,fraud)
```

```
1 10 9944 994.4
2 10 9998 999.8
3 10 9350 935.0
4 10 9633 963.3
5 10 9348 934.8
6 10 9582 958.2
7 10 9324 932.4
8 10 9463 946.3
9 20 0 0.0
[5, 6, 8, 2, 2, 6, 3, 1, 4, 4]
[5, 6, 8, 2, 2, 6, 3, 1, 4, 4]
[5, 6, 8, 2, 2, 6, 3, 1, 4, 4]
[5, 6, 8, 2, 2, 6, 3, 1, 4, 4]
[5, 6, 8, 2, 2, 6, 3, 1, 4, 4]
[5, 6, 8, 2, 2, 6, 3, 1, 4, 4]
[5, 6, 8, 2, 2, 6, 3, 1, 4, 4]
[5, 6, 8, 2, 2, 6, 3, 1, 4, 4]
[5, 6, 8, 2, 2, 6, 3, 1, 4, -9]
```

- Simular mineração egoista (IC)
  Modificar o codigo...stubborn,PoS
  Impacto de poder diferente? Varios egoistas?
- Procurar mineração egoista
- Analisar os suspeitos

- Procurar por mineração egoista
- “Pegadas” ??

- Procurar por mineração egoista
- “Pegadas” ??
- Mineração em sequencia
- Como diferenciar do acaso ??

- Dada uma lista de id de mineradores

$$[5, 6, 8, 2, 2, 6, 3, 1, 4, 4]$$

- Essas repetições vieram do acaso?? (acaso = processo estocástico)
- Pergunta mais fácil:
  Qual o poder dos mineradores ?
  (Valor que maximiza a verossimilhança)

- Dada uma lista de id de mineradores

$$[5, 6, 8, 2, 2, 6, 3, 1, 4, 4]$$

- Gero muitas "minerações" para o poder computacional assumido
  Permutação
- Conto as repetições em cada uma delas
- Comparo o dado real com o as contagens "Posição"
- p-value
- (CUIDADO: multiplos testes!!!)

- Analisar os suspeitos
- Analise financeira
- Conjunto de endereços?? Quantidade de transações ?? alguma característica??