Threat model of blockchain applications

Simone Bronzini
CTO @ Chainside
Overview

- What is a blockchain
- Why is it important for IoT
- What are the security challenges
- Some past exploits
What is the scope of a blockchain?
Create an electronic cash-like payment system
Goals

• p2p transactions
Goals

• p2p transactions

• Censorship resistant
Goals

• p2p transactions

• Censorship resistant

• Permission-less
Goals

- p2p transactions
- Censorship resistant
- Permission-less
- Bonus: store of value
Goals

• p2p transactions

• Censorship resistant

• Permission-less

• Bonus: store of value

• Bonus 2: privacy-aware
Decomposing the problem

• WHO is transacting?
Decomposing the problem

• **WHO** is transacting?

• **WHERE** do I keep track of the transaction?
Decomposing the problem

• **WHO** is transacting?

• **WHERE** do I keep track of the transaction?

• **WHAT** is the object of the transaction?
Decomposing the problem

• **WHO** is transacting?

• **WHERE** do I keep track of the transaction?

• **WHAT** is the object of the transaction?

• **WHEN** is the transaction occurring?
The PayPal model

• Everything is controlled by a centralized server
The PayPal model

• Everything is controlled by a centralized server

• Easy to be censored and shut down
The PayPal model

• Everything is controlled by a centralized server

• Easy to be censored and shut down

• Security hole and reliability risks
The PayPal model

• Everything is controlled by a centralized server

• Easy to be censored and shut down

• Security hole and reliability risks

• No control on the total supply
Using digital signatures you can easily identify the actors in the system
WHO - Digital signatures

• Using digital signatures you can easily identify the actors in the system

• The receiver of the transaction can verify that the sender is actually the owner of a balance
WHO - Digital signatures

• Using digital signatures you can easily identify the actors in the system

• The receiver of the transaction can verify that the sender is actually the owner of a balance

• No need to rely on a central party to manage identities
WHERE - The distributed ledger

• Instead of keeping everything on a central server, each user keeps a local copy of the transactions that are relevant to him
WHERE - The distributed ledger

• Instead of keeping everything on a central server, each user keeps a local copy of the transactions that are relevant to him

• Less trust required through the central party, more difficult to censor transactions
WHAT - The Proof of Work

• Originally applied by Dr. Adam Back to solve the problem of email spamming
WHAT - The Proof of Work

• Originally applied by Dr. Adam Back to solve the problem of email spamming

• To send an email, you were required to calculate the hash of the text + a nonce, and keep hashing changing the nonce until an hash with the required number of 0 was found
WHAT - The Proof of Work

• Originally applied by Dr. Adam Back to solve the problem of email spamming

• To send an email, you were required to calculate the hash of the text + a nonce, and keep hashing changing the nonce until an hash with the required number of 0 was found

• The receiver could easily verify that a work was done in order to send the email
WHAT - The Proof of Work to issue money

• Instead of having a central party issuing new digital money, the PoW can be use to create new coins
WHAT - The Proof of Work to issue money

• Instead of having a central party issuing new digital money, the PoW can be used to create new coins.

• Users have the guarantee that their wealth cannot be easily diluted without an effective work done.
WHAT - The Proof of Work to issue money

• Instead of having a central party issuing new digital money, the PoW can be use to create new coins

• Users have the guarantee that their wealth cannot be easily diluted without an effective work done

• The object of the transaction has a value since a work was done to create a coin
WHAT - The Proof of Work to issue money

50 btc to Alice/ nonce

SHA256

4389051bbab1a0a176caca15c7fc33bbeb8b04a060436a1685b68e20e311b364
34da93ee3b196a998cab98b889c041ad503bf21af46059a84adcefeeedf53e905
C06b4e4b8e35e49f09fe2d5cef82eb56666a98b00097cf4c785918c54bca7fc5
A7de28cfe908ec9801c5e42220545e9d93c6113344166611ee70f3f73705f485
3439c0226bc243d4fa1e816b560851df1c9ecb4ad5bb356d333d6e78716be996

028511490652c7a4ff0a0a4c7b4a6e764600ea96682f54d15beba5e6293edf2
WHEN - The double spending problem

• The user A can create a valid transaction towards the user B, and later a second valid transaction spending the same coins towards the user C
WHEN - The double spending problem

• The user A can create a valid transaction towards the user B, and later a second valid transaction spending the same coins towards the user C

• Both B and C when validating the transaction believe they received the coins, creating two different versions of the ledger
WHEN - The double spending problem

• The user A can create a valid transaction towards the user B, and later a second valid transaction spending the same coins towards the user C

• Both B and C when validating the transaction believe they received the coins, creating two different versions of the ledger

• If the double spending is not forbidden, users can create new money at 0 costs
WHEN - Looking for coordination

• An easy solution would be to have a central coordinator to decide which transaction came first
WHEN - Looking for coordination

• An easy solution would be to have a central coordinator to decide which transaction came first

• This of course causes censorship resistance and reliability problems
WHEN - Looking for coordination

• An easy solution would be to have a central coordinator to decide which transaction came first

• This of course causes censorship resistance and reliability problems

• Everybody needs to have a full copy of the ledger (to detect double spending) and a consensus on the state of the ledger has to be achieved
WHEN - PoW for transaction validation

• The PoW is already applied for issuing transactions (mining), to decide which transactions are valid it can be applied also to other people non-issuing transactions
WHEN - PoW for transaction validation

• The PoW is already applied for issuing transaction (mining), to decide which transactions are valid it can be applied also to other people non-issuing transactions

• A PoW algorithm is applied to a block of transaction, the first one to find an hash with the required difficulty get the issuing transaction (reward) and decides the validity of other people transactions
WHEN - The Proof of Work to issue money

50 btc to Alice
1 btc from Bob to Carol
5 btc from Dave to Frank
2.3 btc from Ted to Chuck

SHA256

0de8c5eadad506ec3e777a25ebc5982d22364e987ac759d5ce685731b798e627
Consensus rules

In any consensus-based system, nodes follow the same set of rules.

What happens if nodes follow different rules?
As soon as a blue block appears, there is no way to recover the split.
Blockchain and IoT

There are 3 main use cases for the blockchain in the IoT industry:
Blockchain and IoT

There are 3 main use cases for the blockchain in the IoT industry:

- **Machine to machine payments**: differently by any other payment system, Bitcoin propriety is defined by the control of a private key, making it the only currency machine can really control and own. For this reason Bitcoin is perfect for m2m transactions
Blockchain and IoT

There are 3 main use cases for the blockchain in the IoT industry:

- **Machine to machine payments:** differently by any other payment system, Bitcoin propriety is defined by the control of a private key, making it the only currency machine can really control and own. For this reason Bitcoin is perfect for m2m transactions

- **Transaction verification:** machine are able to independently verify Bitcoin transactions, making it perfect for actuators that wish to execute an action after a payment has occurred
Blockchain and IoT

There are 3 main use cases for the blockchain in the IoT industry:

- **Machine to machine payments**: differently by any other payment system, Bitcoin propriety is defined by the control of a private key, making it the only currency machine can really control and own. For this reason Bitcoin is perfect for m2m transactions.

- **Transaction verification**: machines are able to independently verify Bitcoin transactions, making it perfect for actuators that wish to execute an action after a payment has occurred.

- **Data notarization**: IoT sensors generated data can be notarized on the blockchain making them verifiable for future auditing, something extremely valuable in the insurance industry.
Smart contracts
Smart contracts
Smart contracts accepted by yellow nodes

Smart contracts accepted by blue nodes
Smart contracts

- Smart contracts accepted by yellow nodes
- Smart contracts accepted by blue nodes
Smart contracts

Smart contracts accepted by yellow nodes

Smart contracts accepted by blue nodes
Smart contracts

- Smart contracts accepted by yellow nodes
- Smart contracts accepted by blue nodes
Smart contracts

- Smart contracts accepted by yellow nodes
- Smart contracts accepted by blue nodes
Smart contracts

Smart contracts accepted by yellow nodes

Smart contracts accepted by blue nodes
Turing completeness - General concept

The diagram represents a state machine with states $s_0, s_1, s_2, s_3, s_4, s_5$. The transitions are labeled with 0, 1, and 0, 1, as indicated by the arrows. The sequence at the bottom shows the output sequence of the state machine.
Turing completeness - In real life

```python
1  a = int(input())
2
3  if a > 10:
4     while a < 100:
5        a = a + 1
6        print(a)
7
8  else:
9     print(a + 10)
10```

Turing completeness - Bitcoin formalisation

\[ s_0 \rightarrow \begin{array}{c}
0, 1
\end{array},
\begin{array}{c}
1
\end{array},
\begin{array}{c}
0
\end{array},
\begin{array}{c}
1
\end{array},
\begin{array}{c}
1
\end{array},
\begin{array}{c}
0
\end{array},
\begin{array}{c}
1
\end{array} \]

\[ s_1 \rightarrow \begin{array}{c}
1
\end{array},
\begin{array}{c}
0
\end{array},
\begin{array}{c}
1
\end{array} \]

\[ s_2 \rightarrow \begin{array}{c}
0
\end{array},
\begin{array}{c}
1
\end{array} \]

\[ s_3 \rightarrow \begin{array}{c}
0
\end{array},
\begin{array}{c}
1
\end{array} \]

\[ s_4 \rightarrow \begin{array}{c}
1
\end{array},
\begin{array}{c}
0
\end{array} \]

\[ s_5 \rightarrow \begin{array}{c}
1
\end{array},
\begin{array}{c}
0
\end{array} \]

\[ s_6 \rightarrow \begin{array}{c}
1
\end{array},
\begin{array}{c}
0
\end{array} \]
Turing completeness - Bitcoin in real life

1 IF
2 HASH160 <hashed_data> EQUAL
3 <pubk1> CHECKSIG
4
5 ELSE
6 <timestamp> CHECKLOCKTIMEVERIFY
7
8 2 <pubk2> <pubk3> <pubk4> 3 CHECKMULTISIG
9
10 ENDIF
Turing completeness - Problems

```python
sorted_currencies = sorted(currencies)
for gateway in [g.replace('/', '-') for g in Account.known_users]:
    for field in ('from_users', 'from_gateways'):
        with open('trust_by_gateway/%s/%s' % (field, gateway), 'w') as outfile:
            outfile.write('time
')
            for curr in sorted_currencies:
                outfile.write('0.0 %0.0
')
                outfile.write('overall_amount
')

for ledger in ledger_gen(START, LAST+1, STEP, True):
    output = {user: c: 0.0 for c in currencies + ['overall']},
    'from_gateways': {c: set() for c in currencies + ['overall']}
    for user in Account.known_users:
        received = {user: c: set() for c in currencies + ['overall']},
        'from_gateways': {c: set() for c in currencies + ['overall']}
        for edge in ledger.trustlines(lambda x: x.dest.known and
                                      converter.convert_in_time(x.amount.currency)):
            name = edge.dest.get_name().replace('/', '-')
            currency = edge.amount.currency
            if edge.orig.known:
                field = 'from_gateways'
            else:
                field = 'from_users'
            converted_value = converter.convert_in_time(edge.amount.value, currency, conversion, time)
            if currency in currencies:
                output[name][field][currency] += converted_value
                received[name][field][currency].add(edge.orig.get_name())
            output[name][field]['overall'] += converted_value
            received[name][field]['overall'].add(edge.orig.get_name())

for gateway in output:
    for field in ('from_gateways', 'from_users'):
        with open('trust_by_gateway/%s/%s' % (field, gateway), 'a') as outfile:
            outfile.write('time
')
            for i in x in range(len(sorted_currencies))
                outfile.write('0.0 %0.0
')
                outfile.write('overall
')
    len(received[gateway][field][sorted_currencies])
    len(received[gateway][field]['overall'])
```
Turing completeness - Problems

- Classical software development process
Turing completeness - Problems

- Classical software development process

Diagram:
- Development
- Testing
- Debugging

Looks like all bugs are fixed...
Turing completeness - Problems

- Classical software development process

- Development
- Deployment
- Testing
- Debugging

Looks like all bugs are fixed...
Turing completeness - Problems

- Classical software development process

Looks like all bugs are fixed...

But then...
Immutability!

- Software development process on smart contracts

Diagram:

- Development
- Testing
- Debugging

往返流程图
Immutability!

- Software development process on smart contracts

Looks like all bugs are fixed...
Immutability!

- Software development process on smart contracts

![Diagram showing software development process]

- Development
- Debugging
- Testing
- Deployment

Looks like all bugs are fixed...
Immutability!

- Software development process on smart contracts

Development → Testing → Development

Looks like all bugs are fixed...

Development → Deployment

But then...
Immutability!

- Software development process on smart contracts

Development → Testing → Debugging

Looks like all bugs are fixed...

Development → Deployment

But then...

WHOOPS!
### Attack surfaces

<table>
<thead>
<tr>
<th>Application Layer</th>
<th>Wallet Layer</th>
<th>Interpreter</th>
<th>Core</th>
<th>Smart Contract</th>
<th>Consensus</th>
<th>Network</th>
</tr>
</thead>
</table>

**Diagram:**

- Application Layer
- Wallet Layer
- Interpreter
- Core
- Smart Contract
- Consensus
- Network
Brainwallets

- Brainwallets are wallets that generate the private keys using few words chosen by the user as source of entropy.
- With this technique users don’t have to securely store their private keys as they are able to regenerate them at anytime.
- Unfortunately the lack of entropy in this key generation system make it easy for attackers to brute force the private keys and steal the funds.
Mt Gox

- In Bitcoin transactions are identified by the TXID, but unfortunately it was possible for third parties to change the TXID of a transaction without making it invalid
- MtGox used the TXID to track deposits and withdrawals of its users
- Attacker could request a withdraw, malleate the withdrawing transaction and contact MtGox support claiming that the transaction didn’t occur
- MtGox would then send a second withdraw transaction to the user
On December 8th 2014, for few hours the random number generator responsible for private keys creation was mistakenly altered.

The key generated in that period were exposed, compromising the bitcoins they held.

Few hundreds bitcoins were stolen, but Blockchain.info reimbursed the users affected by the loss.
On August 15 2010, it was discovered that block 74638 contained a transaction that created 184,467,440,737.09551616 bitcoins. This happened because the code used for checking transactions before including them in a block didn't account for the case of outputs so large that they overflowed when summed. A new version of the client was published, containing a soft forking change to the consensus rules that rejected output value overflow transactions. The blockchain forked until the “good” chain eventually became the longest.
Bitcoin - Multisig evaluation

- In its first version, bitcoin operation `OP_CHECKMULTISIG` expected N+1 values as an input to validate N signatures
- This meant that providing the correct number of signatures had the interpreter crash
- The solution was actually easy (just provide one more dummy parameter)
- To this day we still add a dummy parameter to multisig scripts
Parity provided a multisig wallet smart contract which was probably overcomplicated (hundreds of lines of code)
In the contract there was a bug that allowed third parties to change the ownership of the contract, stealing the funds inside
$31M dollars were stolen, and other $100M were taken by white hat hackers and later given back to the owners
Parity multisig wallet contract refers to a “library” contract to save gas on the deployment of the wallet logic.

The “library” contract however was actually an unutilized wallet contract, and it could be initialized by anyone.

Somebody did it, and as the new owner of the contract was able to kill it, freezing the funds of all the other wallets depending on it.
The DAO was a complex Ethereum-based smart contract that was supposed to act as a decentralised investment fund and collected over $150M. Due to a bug in its code, it was possible for an attacker to withdraw more money than deposited, draining funds out of the DAO.
DAO (2)

● To fix the issues, miners agreed to operate a hard fork and give the money back to the DAO investors

● As a result of the hard fork, the Ethereum network split and still today there are two incompatible versions of the chain
devops199 @devops199
will i get arrested for this? 😁
0x642483b7936b505dbe2e735cc140f29ddfdddb3f3e39efa549707d9e0298de6e4dea99
a30da4872869e36322c9dfcdd06d9a389e746dc6b92f720414e0b18421b
0xae716d8d6b525862f4f3f37d987a971b385b96952

Tienus @Tienus
@devops199 you are the one that called the kill tx?

devops199 @devops199
yes
i'm eth newbie..just learning

qx133 @qx133
you are famous now haha

devops199 @devops199
sending kill() destroy() to random contracts
you can see my history

Xavier @n3xco
can't make an omelet without breaking some eggs
i guess

Tienus @Tienus
Let me know next time you decide to kill some contracts so I can sell my E
Conclusions

● Blockchains are a very powerful tool, but:
  ○ Using them is very complex
  ○ Mistakes can cost millions
  ○ There is a huge need for experts
  ○ There is a huge lack of experts
QUESTIONS?