Snapping Snap Sync:
Practical Attacks on Go Ethereum Synchronising Nodes

Massimiliano Taverna
*ETH Zurich*

Kenneth G. Paterson
*ETH Zurich*
What is Ethereum?

• Most popular blockchain for decentralized applications
What is Ethereum?

• Most popular blockchain for decentralized applications
• Go Ethereum: most widely used Ethereum client
  – 80% of all the nodes, before the Merge
What is Ethereum?

- Most popular blockchain for decentralized applications

- Go Ethereum: most widely used Ethereum client
  - 80% of all the nodes, before the Merge

- The Merge: switch from proof-of-work (PoW) to proof-of-stake (PoS)
  - Sept 2022
What is Ethereum?

• PoW still in use:
  – Ethereum Classic (ETC)
  – EthereumPoW (ETHW)
What is Ethereum?

- PoW still in use:
  - Ethereum Classic (ETC)
  - EthereumPoW (ETHW)
- Go Ethereum even more dominant here
  - 95% of all ETC nodes
What is Ethereum?

• PoW still in use:
  – Ethereum Classic (ETC)
  – EthereumPoW (ETHW)

• Go Ethereum even more dominant here
  – 95% of all ETC nodes

• We have found attacks on Go Ethereum nodes which apply to PoW
Peer-to-Peer Network

- Distributed system
Peer-to-Peer Network

- Distributed system
- Every node has the blockchain
Peer-to-Peer Network

• Distributed system
• Every node has the blockchain
• The blockchain defines the Ethereum state
  • Account balances
  • Smart contracts
Peer-to-Peer Network

• Distributed system
• Every node has the blockchain
• The blockchain defines the Ethereum state
  • Account balances
  • Smart contracts
• Ethereum client
  – Creates, propagates and verifies blocks
  – Propagates, verifies, executes transactions
  – Overall, manages the Ethereum blockchain and state
Proof of Work

• A block needs a PoW in order to be valid
Proof of Work

• A block needs a PoW in order to be valid
• Goal: find PoW s.t. the block hash satisfies a constraint \( \pi \)
Proof of Work

• A block needs a PoW in order to be valid
• Goal: find PoW s.t. the block hash satisfies a constraint $\pi$
  – ETHash is a slow hash function
  – No better strategy than random
Proof of Work

- A block needs a PoW in order to be valid
- Goal: find PoW s.t. the block hash satisfies a constraint $\pi$
  - ETHash is a slow hash function
  - No better strategy than random

$$\pi(\text{ETHash}(B)) = \text{true}$$

B

PoW: 0xabc
Proof of Work

• A block needs a PoW in order to be valid
• Goal: find PoW s.t. the block hash satisfies a constraint $\pi$
  – ETHash is a slow hash function
  – No better strategy than random
• Evidence that computational effort has been put into the block creation

$\pi(\text{ETHash}(B)) = \text{true}$

PoW: 0xabc
Longest chain rule

• Assumption:
  the majority of the computational power is held by honest nodes

• Consequence:
  the longest chain is the honest chain
Synchronisation process

• A new node joins the network
Synchronisation process

- A new node joins the network
- Download and verify all blocks
Synchronisation process

- A new node joins the network
- Download and verify all blocks
  - PoW verification is slow
    - ETHash
Synchronisation process

- A new node joins the network
- Download and verify all blocks
- PoW verification is slow
  - ETHash
- Go Ethereum solution
  - Verify one PoW in every 64 blocks
  - Random choice
Intro to the attacks

- Our attacks target synchronising nodes
Intro to the attacks

- Our attacks target synchronising nodes
- Take a victim node onto a malicious fork
- Arbitrarily modify the Ethereum state
  - Account balances
  - Smart contracts

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>20 $</td>
</tr>
<tr>
<td>Eve</td>
<td>10 $</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>20 $</td>
</tr>
<tr>
<td>Eve</td>
<td>1000 $</td>
</tr>
</tbody>
</table>
Cashing Out

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>20 $</td>
</tr>
<tr>
<td>Eve</td>
<td>10 $</td>
</tr>
</tbody>
</table>
Cashing Out

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>20 $</td>
</tr>
<tr>
<td>Eve</td>
<td>10 $</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>20 $</td>
</tr>
<tr>
<td>Eve</td>
<td>1,000,000 $</td>
</tr>
</tbody>
</table>
Cashing Out

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>20 $</td>
</tr>
<tr>
<td>Eve</td>
<td>10 $</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>20 $</td>
</tr>
<tr>
<td>Eve</td>
<td>1,000,000 $</td>
</tr>
</tbody>
</table>
Cashing Out

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>20 $</td>
</tr>
<tr>
<td>Eve</td>
<td>10 $</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>40,020 $</td>
</tr>
<tr>
<td>Eve</td>
<td>960,000 $</td>
</tr>
</tbody>
</table>

40,000 $
Cashing Out

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>20 $</td>
</tr>
<tr>
<td>Eve</td>
<td>10 $</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>40,020 $</td>
</tr>
<tr>
<td>Eve</td>
<td>960,000 $</td>
</tr>
</tbody>
</table>
Attack 1: Adversarial Model

- Fraction of the total mining power: 1.6%
- 2 malicious peers in the victim’s peer-set
- Victim still has to sync
Attack 1: Security Issues

• Go Ethereum picks a random integer by using crypto/rand 😊
Attack 1: Security Issues

- Go Ethereum picks a random integer by using `crypto/rand`
- This is used to seed `math/rand`
Attack 1: Security Issues

• Go Ethereum picks a random integer by using `crypto/rand` 😊
• This is used to seed `math/rand` 😞
• `math/rand` internally reduces the seed to a 31-bit integer
Attack 1: Security Issues

- Go Ethereum picks a random integer by using `crypto/rand` 😊
- This is used to seed `math/rand` 😞
- `math/rand` internally reduces the seed to a 31-bit integer
- The random PoWs to verify are chosen by using `math/rand`
Attack 1: Security Issues

• Go Ethereum picks a random integer by using crypto/rand 😊
• This is used to seed math/rand 😞
• math/rand internally reduces the seed to a 31-bit integer
• The random PoWs to verify are chosen by using math/rand
• A synchronising node leaks information about its PRNG’s outputs
Attack 1: Security Issues

- Go Ethereum picks a random integer by using `crypto/rand` 😊
- This is used to seed `math/rand` 😞
- `math/rand` internally reduces the seed to a 31-bit integer
- The random PoWs to verify are chosen by using `math/rand`
- A synchronising node leaks information about its PRNG’s outputs

- Consequence: an attacker can recover the seed and build a longer chain than the honest one
Attack 1: Execution

- While $\mathcal{V}$ downloads blocks, $\mathcal{A}$ creates new ones
Attack 1: Execution

- While $\mathcal{V}$ downloads blocks, $\mathcal{A}$ creates new ones
- $\mathcal{A}$ knows which blocks $\mathcal{V}$ will verify and computes PoWs only for those
Attack 1: Execution

- While $\mathcal{V}$ downloads blocks, $\mathcal{A}$ creates new ones
- $\mathcal{A}$ knows which blocks $\mathcal{V}$ will verify and computes PoWs only for those
Attack 1: Execution

- While $\mathcal{V}$ downloads blocks, $\mathcal{A}$ creates new ones.
- $\mathcal{A}$ knows which blocks $\mathcal{V}$ will verify and computes PoWs only for those.
- $\mathcal{A}$ needs to mine one block in less time than is needed by honest miners to mine 64 blocks (on average).
Attack 1: Execution

• While $\mathcal{V}$ downloads blocks, $\mathcal{A}$ creates new ones
• $\mathcal{A}$ knows which blocks $\mathcal{V}$ will verify and computes PoWs only for those
• $\mathcal{A}$ needs to mine one block in less time than is needed by honest miners to mine 64 blocks (on average)

\[- R_A \geq \frac{R_H}{64} \Rightarrow \frac{R_A}{R_H} \geq \frac{1}{64} \approx 1.6\% \]
Attack 2: Adversarial Model

- Fraction of the total mining power: 0.23%
- 1 malicious peer in the victim’s peer-set
- Victim still has to sync
Attack 2: Adversarial Model

• Fraction of the total mining power: 0.23%
• 1 malicious peer in the victim’s peer-set
• Victim still has to sync

Interestingly enough, this attack is enabled by the countermeasure to another attack
Attack 3: Combining both flaws

- We can build a unique attack exploiting both flaws
Attack 3: Combining both flaws

- We can build a unique attack exploiting both flaws
- Outcome: divert the victim onto a malicious chain at a surprisingly low cost
  - 5 GPU for Ethereum
  - 1 GPU for Ethereum Classic
Attack 3: Adversarial Model

- Fraction of the total mining power: $5.5 \times 10^{-7}$
- 2 malicious peers in the victim’s peer-set
- Victim still has to sync
Coordinated Disclosure

• Ethereum
  • Contacted after the Merge
  • Vulnerabilities no longer part of their bug bounty program
Coordinated Disclosure

• Ethereum
  • Contacted after the Merge
  • Vulnerabilities no longer part of their bug bounty program

• Ethereum Classic
  • 90-day disclosure period
  • Collaborative team of developers
Coordinated Disclosure

• Ethereum
  • Contacted after the Merge
  • Vulnerabilities no longer part of their bug bounty program

• Ethereum Classic
  • 90-day disclosure period
  • Collaborative team of developers

• EthereumPoW
  • No reply despite multiple attempts to contact them
Conclusion: A lack of security awareness

• Usage of a weak PRNG for a security-critical operation
• Closing a vulnerability opens a new one
• Web3 is a new, dynamic environment
Thanks for listening
Kenneth G. Paterson
Email: kenny.paterson@inf.ethz.ch

Massimiliano Taverna
Email: massi.taverna@gmail.com