CHAINIAC: Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds

1Kiriill Nikitin, 1Eleftherios Kokoris-Kogias, 1Philipp Jovanovic, 1Linus Gasser, 1Nicolas Gailly, 2Ismail Khoffi, 3Justin Cappos, 1Bryan Ford

1École polytechnique fédérale de Lausanne (EPFL)
2University of Bonn
3New York University
Software Updates

A program tape for the 1944 Harvard Mark I, one of the first digital computers. [Wikipedia](https://en.wikipedia.org/wiki/Mark_I).

Hilary Mason’s Twitter
Software Updates

- Softwares updates are used to patch disclosed vulnerabilities, add new features, and improve security posture.

- If you *do not* update your system, things can go bad…
Software Updates

• But even if you *do* update your system regularly, things can go wrong too…

• Software-update systems are a lucrative attack target due to their centralized design and potential impact on users

How can we make software-update systems more secure and transparent?
Software Release Pipeline

**Development/Review** – Building release binaries – Sign-off – Release distribution

- Developers
- Distribution center
- Users
Software Release Pipeline

Development/Review – **Building release binaries** – Sign-off – Release distribution
Software Release Pipeline

Development/Review – Building release binaries – **Sign-off** – Release distribution

- Build server
- Developers
- Users
- Distribution center
Software Release Pipeline

Development/Review – Building release binaries – Sign-off – **Release distribution**
Challenges

1. Make software-update process resilient to partial key compromise
Challenges

1. Make software-update process resilient to partial key compromise

Build server

Developers

Distribution center

Users
Challenges

1. Make software-update process resilient to partial key compromise
Challenges

1. Make software-update process resilient to partial key compromise
2. Prevent malicious substitution of a release binary during building process
Challenges

2. Prevent malicious substitution of a release binary during a build process
Challenges

2. Prevent malicious substitution of a release binary during a build process
Challenges

2. Prevent malicious substitution of a release binary during a build process
Challenges

How many of you have reproducibly built software binaries for personal use?
Challenges

2. Prevent malicious substitution of a release binary during a build process

Building the Tor Browser bundle takes 32 hours on a modern laptop
Challenges

3. Protect users from targeted attacks by coerced or bribed developers
Challenges

3. Protect users from targeted attacks by coerced or bribed developers
3. Protect users from targeted attacks by coerced or bribed developers
Challenges

3. Protect users from targeted attacks by coerced or bribed developers
Challenges

3. Protect users from targeted attacks by coerced or bribed developers
Challenges

4. Enable developers to securely rotate their signing keys in case of renewal or compromise
Challenges

4. Enable developers to securely rotate their signing keys in case of renewal or compromise
Challenges

4. Enable developers to securely rotate their signing keys in case of renewal or compromise.
4. Enable developers to securely rotate their signing keys in case of renewal or compromise
Design of CHAINIAC
Roadmap to CHAINIAC

- Decentralized Release Approval
- Verified Builds
- Anti-equivocation
- Key Evolution
1. Make software-update process resilient to partial key compromise
Decentralized Release-Approval

1. Make software-update process resilient to partial key compromise
Decentralized Release-Approval

1. Make software-update process resilient to partial key compromise
Decentralized Release-Approval

1. Make software-update process resilient to partial key compromise
Decentralized Release-Approval

1. Make software-update process resilient to partial key compromise
Decentralized Release-Approval

1. Make software-update process resilient to partial key compromise
Background

- Collective Authority (Cothority), Collective Signing (CoSi), and BFT-CoSi

Authoritative statements: e.g. log records

1. record
2. record
3. record

Each statement collectively signed by both authority and all or most witnesses

References
Verified Builds

2. Prevent malicious substitution of a release binary during building process
Verified Builds

2. Prevent malicious substitution of a release binary during building process.
2. Prevent malicious substitution of a release binary during building process
2. Prevent malicious substitution of a release binary during building process
Verified Builds

Release Policy File
- List of individual developer public keys
- Signing threshold
- Cothority public key
- Supported platforms for verified builds
- ...

Decentralized Release Approval
Verified Builds
Anti-equivocation
Key Evolution
Anti-equivocation Measures

3. Protect users from targeted attacks by coerced or bribed developers

Developers

Cothority

Release Tree
<source code>
<binaries>
<previous head>

Developers’ signatures

Policy

Release 1
Co-signature

Release 2
Co-signature

Release 3
Co-signature

Transparency Release Log

Distribution center

User

Decentralized Release Approval

Verified Builds

Anti-equivocation

Key Evolution
3. Protect users from targeted attacks by coerced or bribed developers.
Anti-equivocation Measures

3. Protect users from targeted attacks by coerced or bribed developers

Developers

Cothority

Policy

Distribution center

Transparency Release Log

Release 1
Co-signature

Release 2
Co-signature

Release 3
Co-signature

Release 4
Co-signature

User
Evolution of Developer Keys

4. Enable developers to securely rotate their signing keys
Evolution of Developer Keys

4. Enable developers to securely rotate their signing keys
4. Enable developers to securely rotate their signing keys
Evolution of Developer Keys

4. Enable developers to securely rotate their signing keys
4. Enable cothority to securely rotate its collective key
Evolution of Cothority Configuration

4. Enable cothority to securely rotate its collective key
Skipchains
Skipchains

• Novel data structure: blockchain + skip lists

• Blocks have multi-hop two-way links:
  
  ▶ *Backward links* - hashes of past blocks
  
  ▶ *Forward links* - (collective) signatures

• Secure and efficient traversal of arbitrary long timelines
Implementation and Evaluation
Implementation

• CHAINIAC is implemented in Go
  ▶ Using the DEDIS Kyber crypto library and Onet networking framework
  ▶ Available open-source at https://github.com/dedis/paper_chainiac
Evaluation Methodology

What is the cost effect of CHAINIAC on cothority nodes and on clients?

• Cothority-node CPU cost of validating releases and maintaining transparency release log
  ‣ The average values for six Debian packages over two years
## Evaluation

1. Cothority-node CPU cost of validating releases and maintaining release log

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>3</th>
<th>15</th>
<th>127</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dev-signature verification</td>
<td>10^{-3}</td>
<td>10^{-2}</td>
<td>10^{-1}</td>
</tr>
<tr>
<td>Collective signing</td>
<td>10^{-1}</td>
<td>10^{0}</td>
<td>10^{1}</td>
</tr>
<tr>
<td>Reproducible build</td>
<td>10^{2}</td>
<td>10^{3}</td>
<td>10^{4}</td>
</tr>
<tr>
<td>CPU / Wall</td>
<td>10^{5}</td>
<td>10^{-1}</td>
<td>10^{1}</td>
</tr>
<tr>
<td>Creating timestamp</td>
<td>10^{2}</td>
<td>10^{3}</td>
<td>10^{4}</td>
</tr>
<tr>
<td>Wall-total over all nodes</td>
<td>10^{4}</td>
<td>10^{5}</td>
<td>10^{6}</td>
</tr>
</tbody>
</table>

### Diagram

- Cothority
  - Release 1
    - Signatures
  - Release 2
    - Signatures
  - Release 3
    - Signatures

- Co-signature
- Release 1
- Release 2
- Release 3
Evaluation

1. Cothority-node CPU cost of validating releases and maintaining release log

10$/month server is sufficient to validate and maintain the log of Debian-security repository

![Diagram](chart.png)
Evaluation Methodology

What is the cost effect of CHAINIAC on cothority nodes and on clients?

• Cothority-node CPU cost of validating releases and maintaining transparency release log
  ‣ The average values of six required Debian packages

• CPU cost of reproducing packages on cothority nodes
  ‣ From 1.5 to 30 minutes to reproduce a package

• Skipchain effect on communication cost
  ‣ Reducing the cost by the factor of 30 on 1.5 million update-requests from the PyPI repository

• CPU and bandwidth cost of securing a multi-package distribution
  ‣ ~20 sec to create a snapshot of >50k-packages Debian repository
Conclusion

• CHAINIAC decentralizes each step of the software-update process to increase trustworthiness and to eliminate single points of failure

• Skipchain structure for efficient logging and secure key evolution;
  See https://bford.github.io/2017/08/01/skipchain/ for more applications

• Verified builds as an improvement over reproducible builds

• Role-based architecture, multi-package Chainiac and more are in the paper

Kirill Nikitin
kirill.nikitin@epfl.ch
@nikir1ll