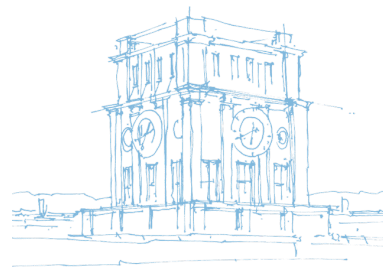
Software Transparency:
package security beyond signatures and reproducible builds

MiniDebconf Hamburg, 19 May 2018

Benjamin Hof
hof@in.tum.de



Overview

- propose add-on security feature on top of Secure APT and reproducible builds
- investigate where infrastructure modifications are required
- solicit feedback

Introduction

Design

Feedback

Motivation

Attacks against software distribution

- Juniper backdoors, discovered 2016
- developers phished $\rightarrow$ Chrome extensions backdoored, 2017
- 100 banks infected by signed banking software update (ShadowPad/NetSarang), 2017

Goals

relax trust in archive

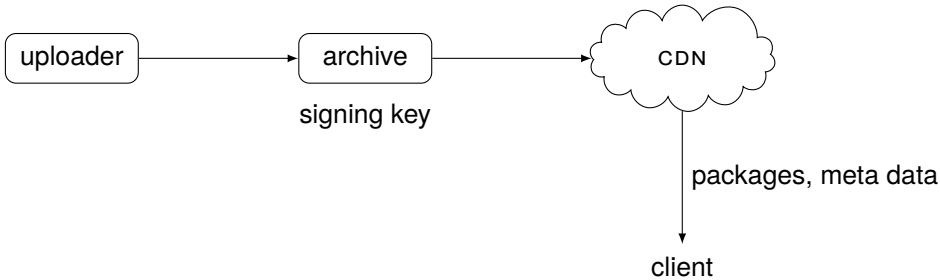
- detect targeted backdoors under compromised archive

forensic auditability

- inspectable source code for every binary
- verified mapping between source and binary
- identify maintainer responsible for distribution
- proof attributing misbehaviour

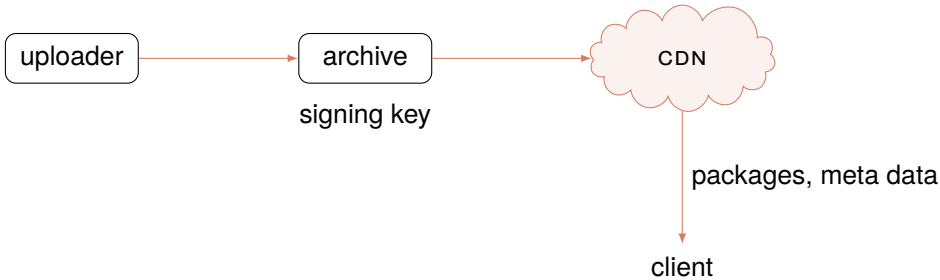
Software distribution in APT

Current approach



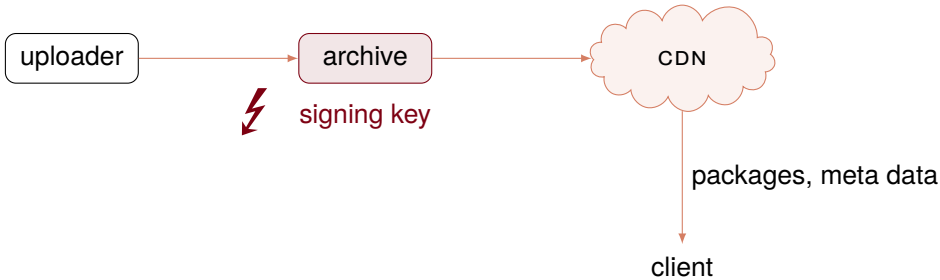
Software distribution in APT

Security model



Software distribution in APT

Security model



Goals:

- resistance against key compromise and targeted backdoors
- audit support

Approach

Make sure everybody runs the exact same software

- analysis carries over
- no targeted backdoors
- generate cryptographic proof attributing misbehaviour

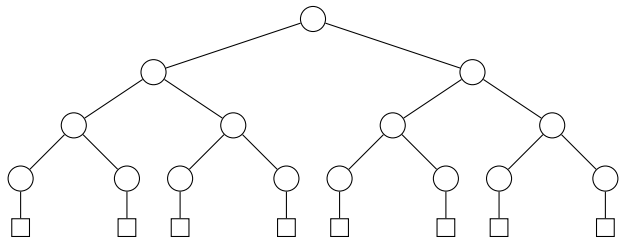
Introduction

Design

Feedback

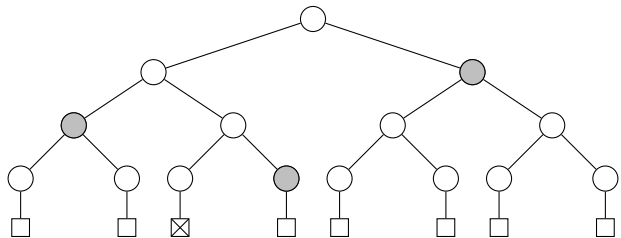
Authenticating meta data and source code

Hash tree over a list of items: Merkle tree



Authenticating meta data and source code

Hash tree over a list of items: Merkle tree



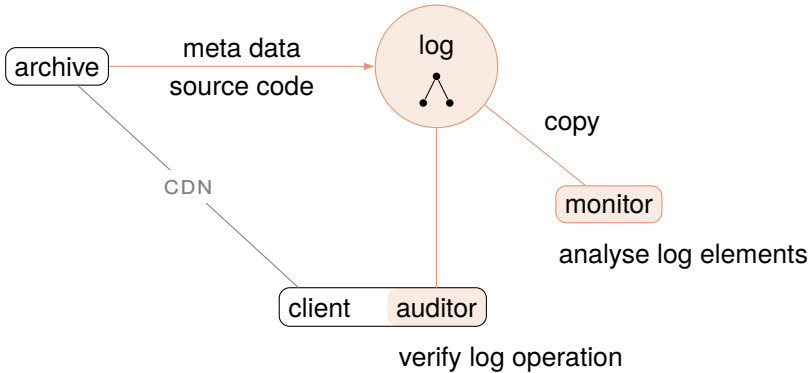
Log can efficiently and cryptographically prove:

- inclusion of a given element in the list
- append-only operation of the list

$\Rightarrow$ no need to trust log, can be verified

Design

Software Transparency architecture for APT



Transparency overlay

Log can efficiently and cryptographically prove:

- inclusion of a given element in the list
- append-only operation of the list

Auditor and monitor:

- verify log inclusion of the release file
- verify consistent operation of the log
- monitor additionally investigates log entries

Discussion

Archive

- log submission
- distribution of log proofs/promises

- consistent release frequency
- responsible for distributing reproducible builds

- buildinfo covered by release file (source meta data)
- source-only uploads
- respect an append-only key ring source package as authoritative

Discussion

Log and auditor

log

- standalone server (probably more than one)
- can have caching read-only front ends

auditor

- APT integration for proof verification: crypto, file format, network
- log proofs can also be distributed over mirrors

Discussion

Verification functions of monitor

- append-only operation
- exchange tree roots with monitors and auditors
- meta data
  - completeness
  - source available
  - version increment: change in package meta data or buildinfo requires version bump
- release frequency
- upload ACL: source signatures, key ring
- reproducible builds

Monitor features relevant to different groups

Independent of existing infrastructure

- maintainers
  - reproducible binary
  - all uploads using my key
  - who did changes on my packages
- key ring maintainer, DAM
  - key ring
  - uploads
- reproducible builds
  - assurance of reproducible binary packages
- archive
  - assurance of correct meta data
  - assurance of reproducible binary packages
  - upload ACL

Implementation and evaluation

- prototypes for all components exist
- replayed two years of Debian stretch (mostly pre-release)
  - log size: 270 k tree leaves
  - log storage: 396 GB mainly source packages
- monitor cost dominated by reproducible builds verification
- unexpected events in meta data
  - source missing
  - version not incremented

Related work

- Chase, Meiklejohn: Transparency Overlays and Applications (2016)
- Nikitin et al.: CHAINIAC (2017)
- Barnes: Binary Transparency (Mozilla Wiki, 2017)
- Al-Bassam, Meiklejohn: Contour (preprint)

Summary

- detection of targeted backdoors
- forensic auditability
  - inspectable source code for every binary
  - verified mapping between source and binary
  - identify maintainer responsible for distribution
  - proof attributing misbehaviour
- equivocation attacks not discussed today

Introduction

Design

Feedback

Questions and feedback