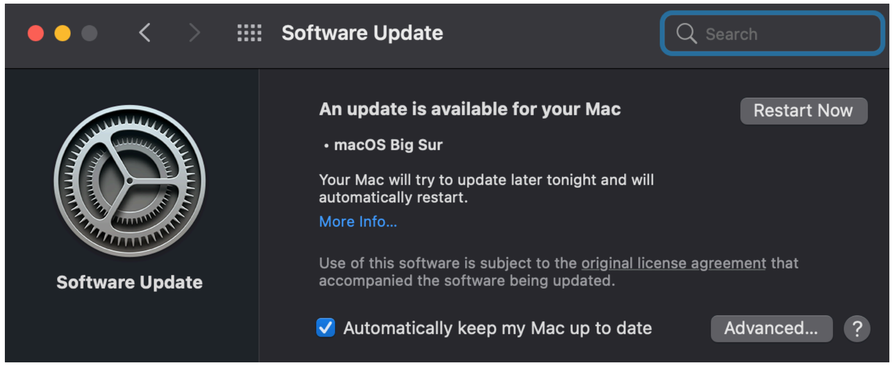
CHAINIAC

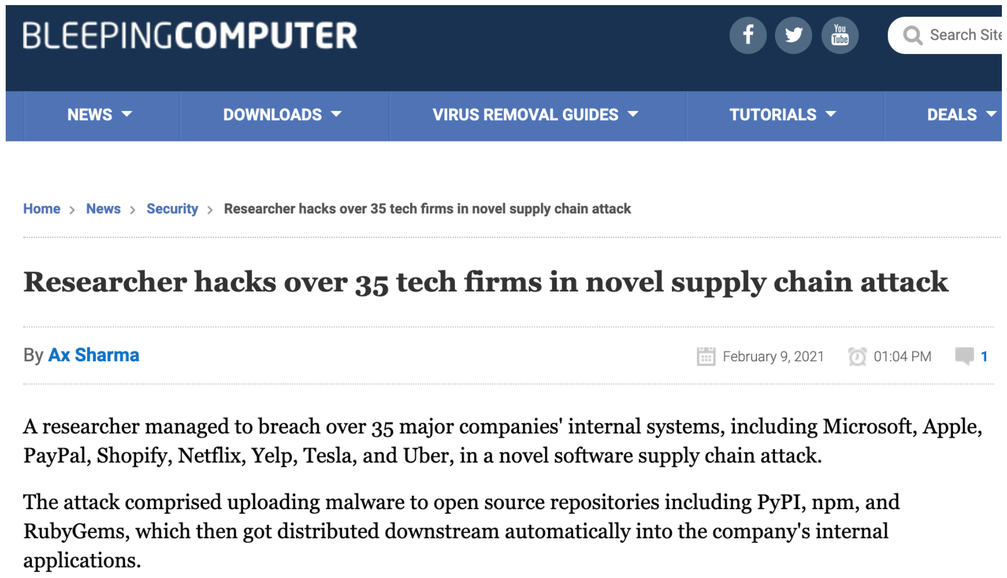
Proactive Software-Update Transparency via Collectively Signed Skipchains and Verified Builds

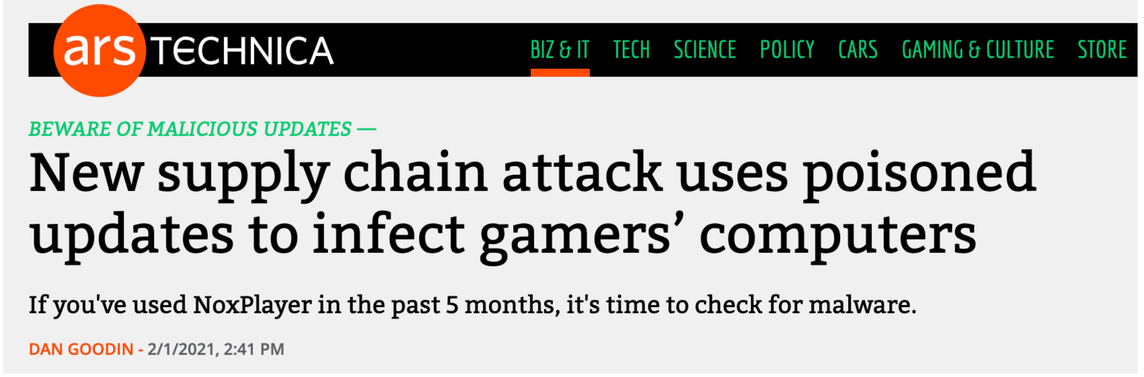
Rick Barber

Motivation


Software Update
Search
An update is available for your Mac
Restart Now
• macOS Big Sur
Your Mac will try to update later tonight and will automatically restart.
More Info...
Use of this software is subject to the original license agreement that accompanied the software being updated.
Software Update
Automatically keep my Mac up to date
Advanced...
?

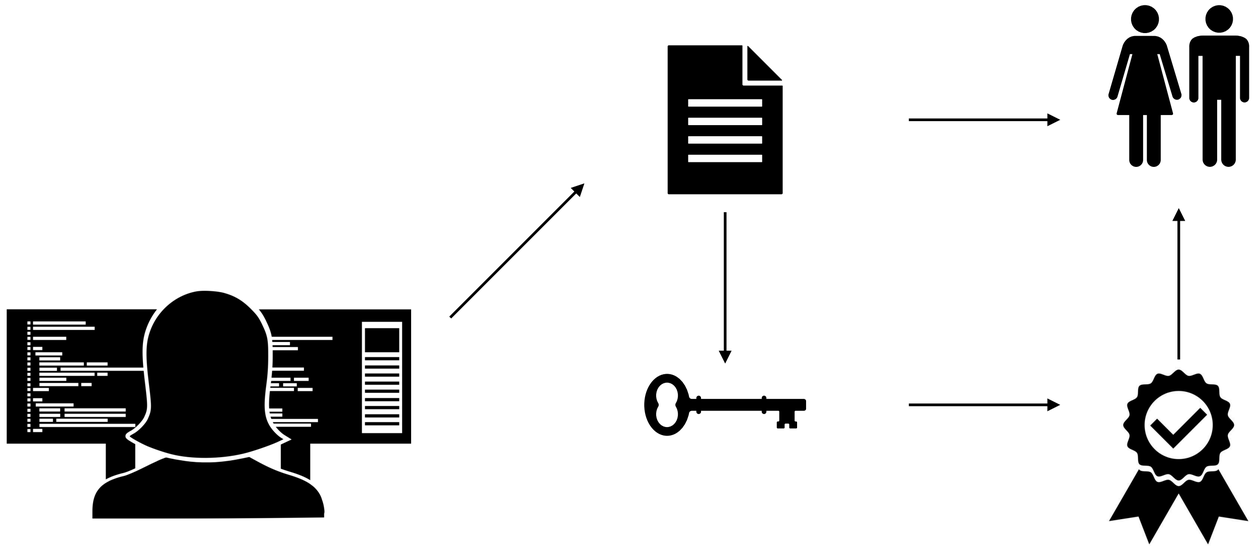

REUTERS
World
Business
Markets
Breakingviews
Video
More
MEDIA AND TELECOMS
FEBRUARY 14, 2021 / 7:50 PM / UPDATED 17 HOURS AGO
SolarWinds hack was 'largest and most sophisticated attack' ever: Microsoft president
By Reuters Staff
2 MIN READ


BLEEPINGCOMPUTER
Search Site
NEWS
DOWNLOADS
VIRUS REMOVAL GUIDES
TUTORIALS
DEALS
Home › News › Security › Researcher hacks over 35 tech firms in novel supply chain attack
Researcher hacks over 35 tech firms in novel supply chain attack
By Ax Sharma
February 9, 2021
01:04 PM
1
A researcher managed to breach over 35 major companies' internal systems, including Microsoft, Apple, PayPal, Shopify, Netflix, Yelp, Tesla, and Uber, in a novel software supply chain attack.
The attack comprised uploading malware to open source repositories including PyPI, npm, and RubyGems, which then got distributed downstream automatically into the company's internal applications.


ars TECHNICA
BIZ & IT
TECH
SCIENCE
POLICY
CARS
GAMING & CULTURE
STORE
BEWARE OF MALICIOUS UPDATES —
New supply chain attack uses poisoned updates to infect gamers' computers
If you've used NoxPlayer in the past 5 months, it's time to check for malware.
DAN GOODIN - 2/1/2021, 2:41 PM

Why are we seeing so many “supply chain” attacks?

Software update ideal



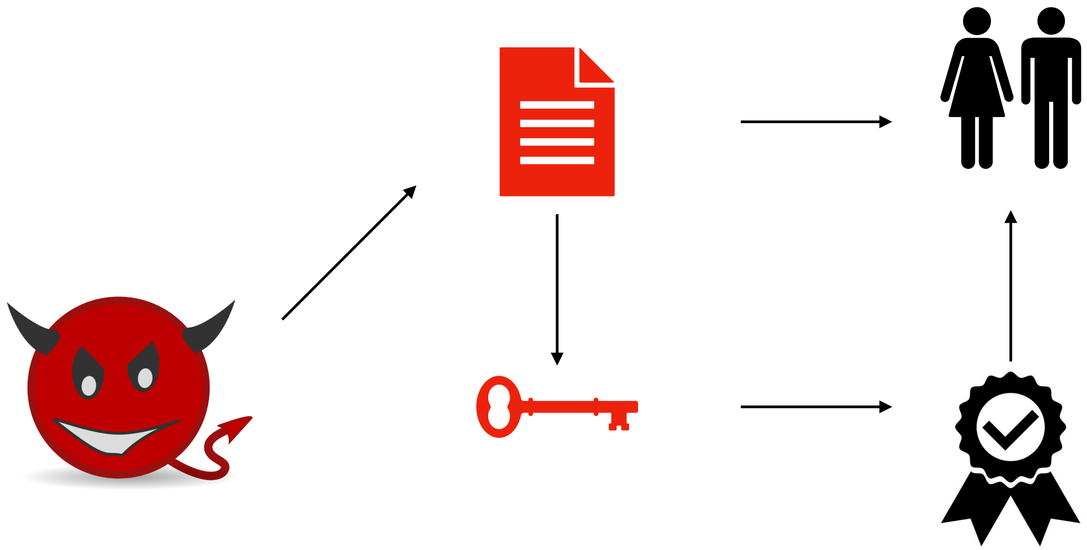
Developers write update

A key is used to sign the update

Users get update and signature

Why are we seeing so many “supply chain” attacks?

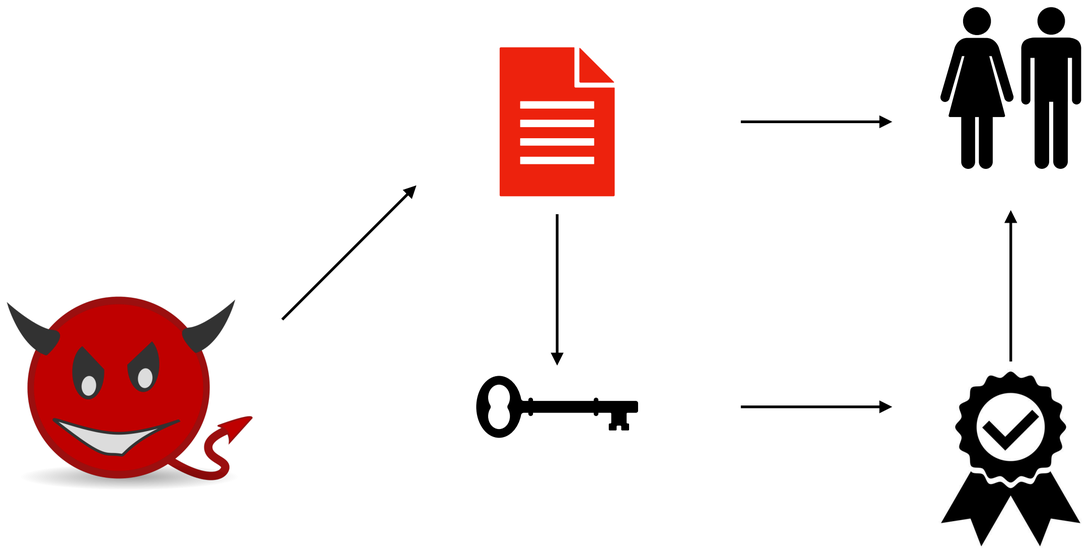
One failure mode



Attacker compromises key and signs a malicious update

Why are we seeing so many “supply chain” attacks?

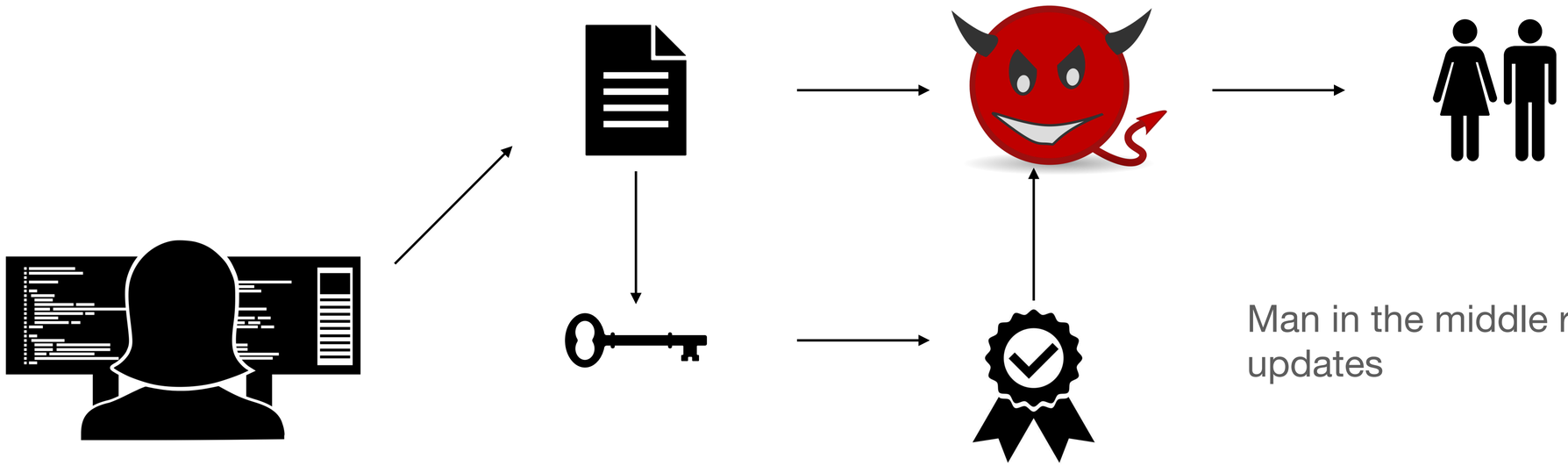
Another failure mode



Attacker compromises source or compilation and binary is signed by unaware key holder

Why are we seeing so many “supply chain” attacks?

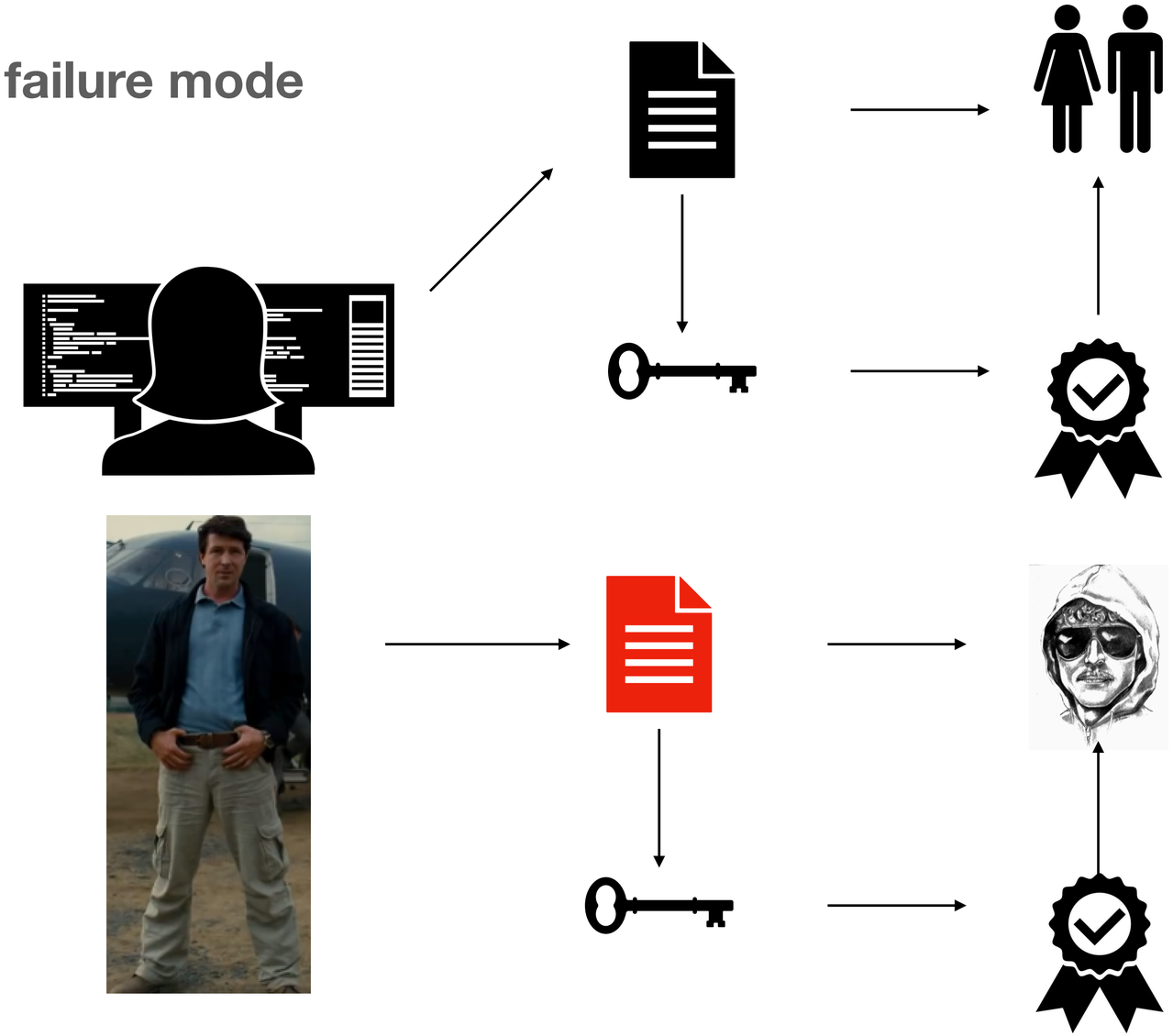
Yet another failure mode



Man in the middle replays stale, vulnerable updates

Why are we seeing so many “supply chain” attacks?

Yet another failure mode



Someone compels targeted malicious update

CHAINIAC Solution

Expand trust with a multi-signature cothority

Verifiable builds within the cothority

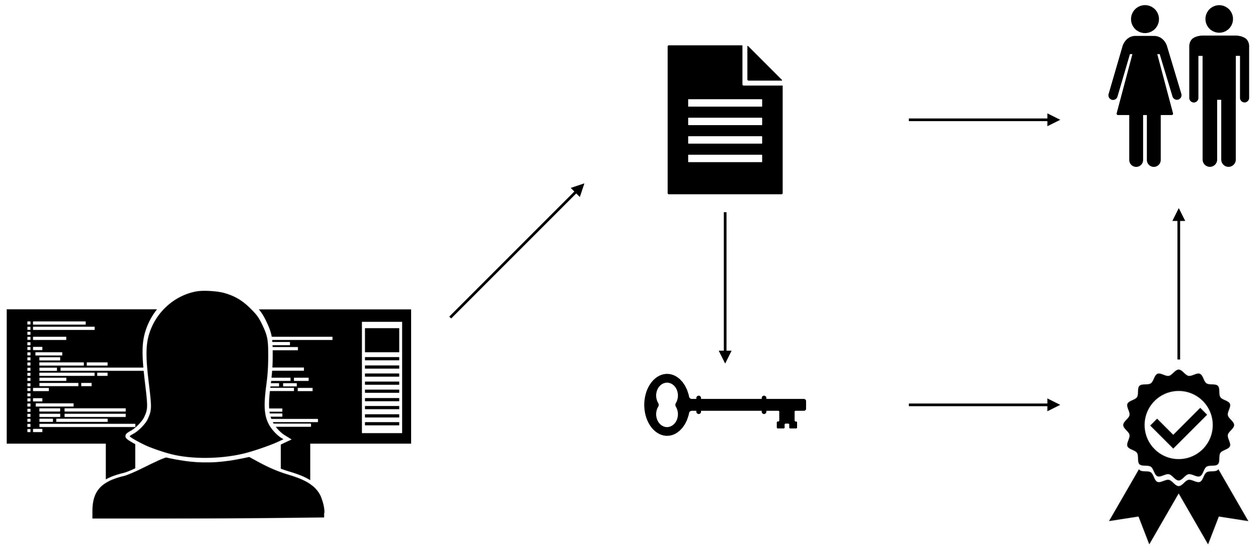
Facilitate key rotation

Efficient, tamper-evident update timeline to ensure timeliness & integrity

Building CHAINIAC

Step 0

Software update ideal



Developers write update

A key is used to sign the update

Users get update and signature

Building CHAINIAC

Step 1: Decentralized release approval

Software has a **policy file** containing developer public keys

For each new release, each developer signs the source and a user can accept if number of signers greater than threshold

User builds the binary from signed source

But this sucks for the user….

Building CHAINIAC

Step 2: Build transparency via developers

Each candidate release is a binary + source

Each developer compiles the source to a binary using reproducible build techniques and signs if their binary matches the release target binary

User trusts code if threshold of signatures

But this sucks for the developers

Discussion

What happens if a piece of software does not provide reproducible builds?

Is it necessary / important for developers wishing to incorporate CHANIAC for future updates to migrate their entire update history to the timeline?

Building CHAINIAC

Step 3: Release validation with cothority

Binary + source are sent to third party witness servers who are trusted collectively but not individually

These witnesses build the binary from source and witness the correspondence

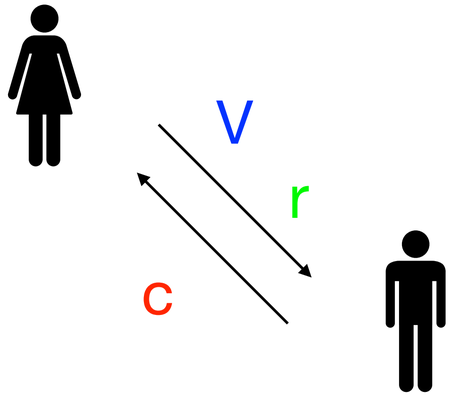
User trusts code if threshold of witness signatures

Policy file contains witness public keys

Building CHAINIAC

Interlude: BFT-CoSi

Schnorr signatures



(r,c) is called the Schnorr signature

Anyone can verify X is Alice's public key by computing $V' = G^r X^c$ and checking c = H(V' || S)

Alice wants to sign S, Bob has Alice's public key X and wants to be sure it is Alice who signed S

They've agreed on a group of prime order with a generator G ahead of time, and a cryptographic hash function H

$X = G^x$ where x is Alice's secret key

Alice selects a secret v and computes a commit $V = G^v$, which she sends to Bob

Bob responds with a challenge c = H(V || S) (S is what's being signed, recall)

Alice responds with $r = v - cx$

Building CHAINIAC

Interlude: BFT-CoSi

Fairly straightforward to get multi-signatures from this

Public key is the product of everyone's public key $X = \prod_i X_i$

Each witness comes up with their own secret v_i and commit $V_i = G^{v_i}$

Verifier issues a collective challenge $c = H(V||S)$ aggregating commits $V = \prod_i V_i$

Each witness responds with $r_i = v_i - cx_i$

Schnorr signature is (r, c) with $r = \sum_i r_i$

BFT-CoSi builds a tree from Schnorr multi signatures where aggregate commitments and responses flow up and the message to be signed and challenges flow down from root

Discussion

How can we ensure the integrity of witness servers?

Can witness servers be incentivized to collude and inject malware before verifying the build?

Are witnesses shared across many packages or does each package have a unique set of witness servers?If so, could this be used to leak information about proprietary source code?

Building CHAINIAC

Step 4: Anti-equivocation measures

Goal: to prevent targeting of specific users and to discourage attempts to compromise developers.

The cothority will build a hash chain of releases with each block containing Merkle tree of the software version and other metadata.

Backward links will be hashes of prior blocks, forward links will be the BFT-CoSi signatures, witnessing the next release

Discussion

The paper mentions how even if a faulty / backdoor'd build gets added to the log, it stays present for future scrutiny. Is there an elegant method to prevent clients from using this update?

Building CHAINIAC

Step 5: Key rotation

Developer and cothority keys will need to rotate from time to time, likely on staggered schedules in order to present a moving target for attackers

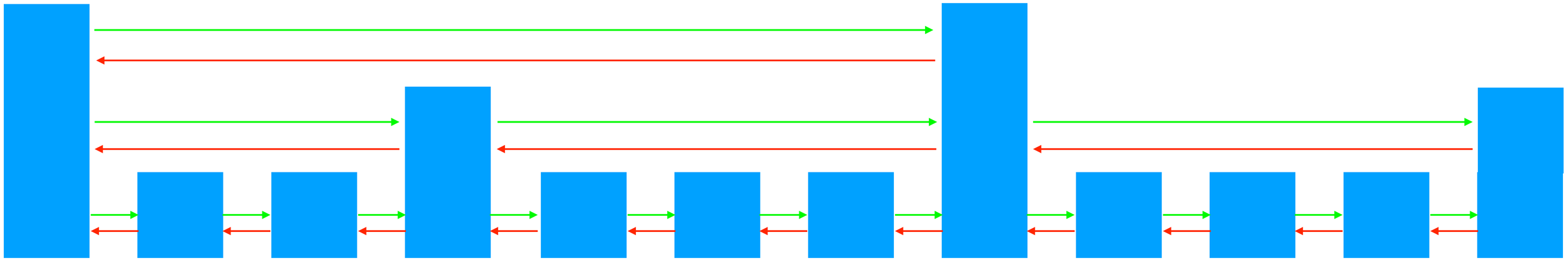
Create a block chain whose blocks are cothority configurations and have developers include key rotation in their release Merkle tree

A large cothority means frequent key turnover from the user’s POV, so we need an efficient data structure

Building CHAINIAC

Interlude: Skipchains

Hash chains meet skip lists to achieve O(log n) search in a sorted linked list



Blocks are cothority configurations

Backlinks are hashes of prior blocks

Forward links are collective signatures, potential problem?

Discussion

The immediate discussion question that comes to mind is what applications besides Chaniac can the skipchain data structure be used for?

Experiments

Debian: built packages reproducibly and timed it. 90% of a sample of popular and random packages built in 3 minutes. The number was 5.5 minutes for 27 required packages.

Tested time to add a new release block for cothorities of various sizes. CoSI performed well. Communication overhead grew modestly with network size.

PyPI: compared skipchains to linear updates and diff between now and last update. Skipchain performs similar to the latter.

For the client: CHAINIAC added an overhead of 16% to APT manager.

Discussion

How does Chaniac compare to other software update protection dissemination work such as overlay and peer to peer based approaches?