Code reviews and governance with Codechain

Frank Braun

@thefrankbraun

2020-03-10

Reflections on Trusting Trust

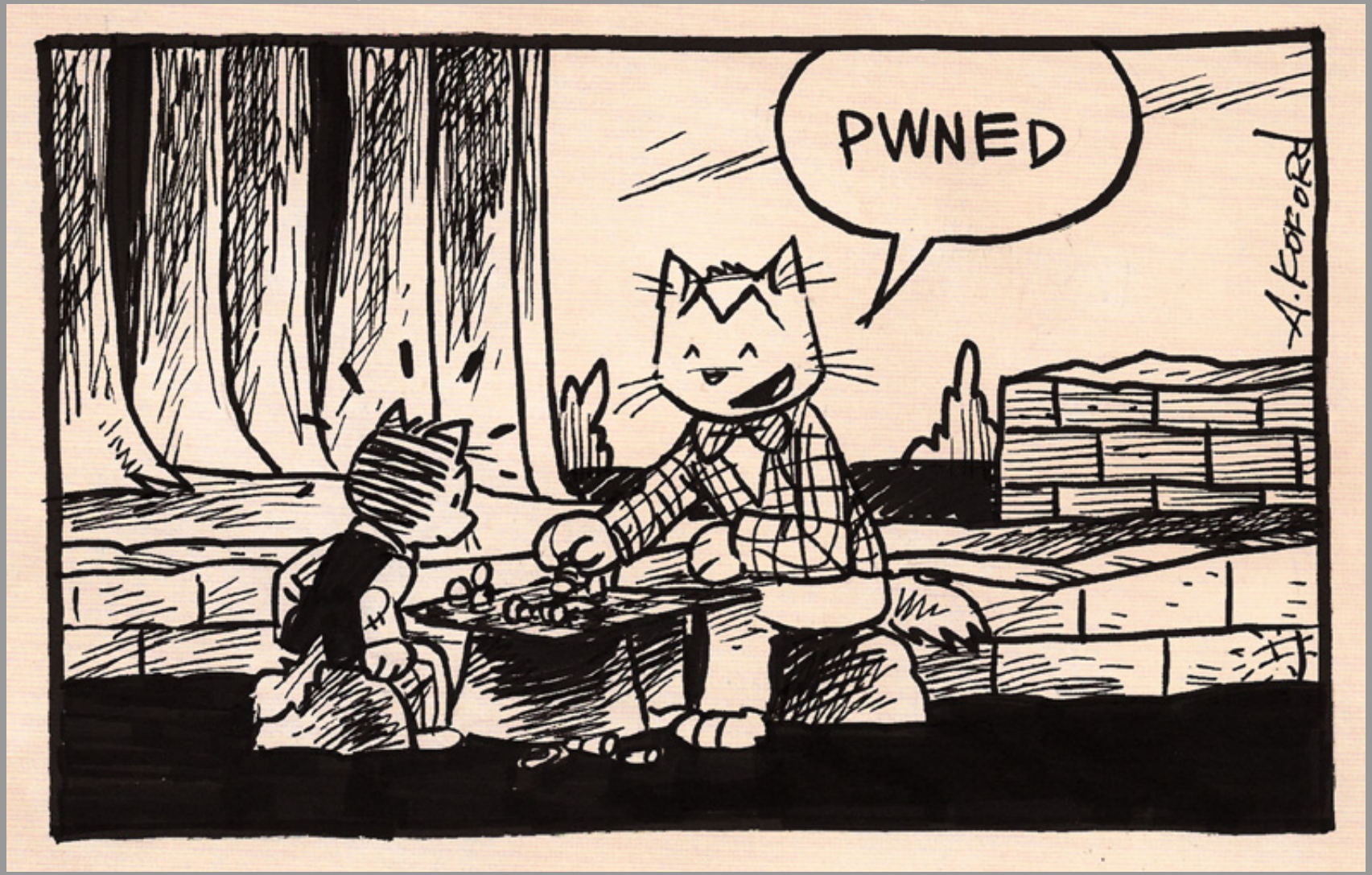
> *"To what extend should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software." — Ken Thompson, Turing Award Lecture, 1984*

questions:

- how can we trust the people who wrote the software?
- how can we make sure we actually run the code they wrote?

$\Rightarrow$ this talk is not about making sure the code you execute is right, but making sure you execute the right code!

what happens if you execute the wrong code?



what are the problems (with code)?

- manual verification of the code
- a single signing key
- targeted backdoors

software signed by GPG keys

- has to be verified manually
- one compromised key (or developer!) is enough to break it
- no easy key rotation
- targeted updates are usually possible

That's how it works for Bitcoin Core[1]

[1] `https://bitcoin.org/en/download`

Git version control system

- data integrity via Git's data structure (Merkle trees)
- Git allows to sign tags and commits with GPG

Git: signing & verifying tags

```
$ git tag -s v1.5 -m 'my signed 1.5 tag'
$ git tag -v v1.5
```

problem:

- tags are not unmodifiable

Git: signing & verifying commits

```
$ git commit -a -S -m 'signed commit'
$ git merge --verify-signatures signed-branch
```

only merging "fast-forwarding" branches gives some protection against regression (given one knows the `HEAD`)

problem:

- every commit needs to be signed
- user has to trust all developer keys

$\Rightarrow$ hard to deploy in practice

summary of possible attacks

The current solutions to sign software do not protect against:

- key compromise
- developer coercion (wrench attack), blackmailing, or bribing
- regression (suppressing of updates)

A developer being forced to give up his signing key or a stolen repository signing key would be disastrous.

GPG has no automatic mechanism for key rotation, likely reason why many GPG keys are quite old.

proposed solution

design for secure software distribution and development which mitigates these attacks:

1. Establishing code trust via multi-party code reviews recorded in unmodifiable hash chains. This prevents that a single developer can include a generic backdoor into software.
2. A single source of truth (SSOT) mechanism which makes sure every user of the software gets the same version of the software. This prevents targeted backdoors and the suppression of security updates.

Together this builds a secure software delivery and update mechanism which cannot be compromised by a single developer or for a specific user, thereby preventing targeted backdoors.

Codechain design

(joined work with Jonathan "Smuggler" Logan)

- the "unit" of code are directory trees
- the hash of a directory tree is a tree hash
- the code history is a sequence of unique tree hashes, starting from the hash of the empty tree
- the sequence of tree hashes and their signatures are recorded in a hash chain file
- the signatures contributes towards a m-of-n threshold
- code is distributed as a set of patch files which transform a directory tree *a* into a directory tree *b*
- patch files are named after the outgoing tree hash *a*

tree hashes

```
$ cd $GOPATH/src/github.com/frankbraun/codechain/doc/helloproject

$ codechain treehash -l
f ab81f3080f71a034c90dc0ca64b62295d3a75a23ec1b0f498dfda4a34325ae3a README.md
f ad125cc5c1fb680be130908a0838ca2235db04285bcdd29e8e25087927e7dd0d hello.go

$ codechain treehash
d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321c7aa55629716

$ codechain treehash -l | sha256sum
d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321c7aa55629716
```

patch files

```
codechain patchfile version 1
treehash e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495
+ f ab81f3080f71a034c90dc0ca64b62295d3a75a23ec1b0f498dfda4a34
dmppatch 2
@@ -0,0 +1,45 @@
+## Example project for Codechain walkthrough%0A
+ f ad125cc5c1fb680be130908a0838ca2235db04285bcdd29e8e2508792
dmppatch 2
@@ -0,0 +1,78 @@
+package main%0A%0Aimport (%0A%09%22fmt%22%0A)%0A%0Afunc main
%7B%0A%09fmt.Println(%22hello world!%22)%0A%7D%0A
treehash d844cbe6f6c2c29e97742b272096407e4d92e6ac7f167216b321
```

hash chain format

- a hash chain is stored in a simple newline separated text file
- each hash chain entry corresponds to a single line of the form:

```
hash-of-previous current-time type type-fields ...
```

where:

- hash-of-previous is the SHA256 hash of the previous line (without newline)
- the fields are separated by single white spaces
- the current-time is encoded as an ISO 8601 string in UTC
- all hashes in a hash chain are SHA256 hashes encoded in hex notation
- hex encodings have to be lowercase
- all public keys are Ed25519 keys and they and their signatures are encoded in base64 (URL encoding without padding)
- comments are arbitrary UTF-8 sequences (without newlines)

hash chain types

there are six different types of hash chain entries:

```
cstart
source
signtr
addkey
remkey
sigctl
```

- a hash chain must start with a `cstart` entry
- that is the only line where this type must appear

type `cstart`

A `cstart` entry starts a new hash chain.

```
hash-of-prev cur-time cstart pubkey nonce signature [comment]
```

type source

Marks a new source tree state for publication (from developer).

```
hash-of-prev cur-time source source-hash pubkey sig [comment]
```

Signature is over the source-hash and the optional comment.

type `signtr`

Signs a previous entry and approves all code changes and changes to the set of signature keys and `m` up to that point.

```
hash-of-prev cur-time signtr hash-of-chain-entry pubkey sig
```

It does not have to sign the previous line ($\rightarrow$ detached signatures).

type addkey

Marks a pubkey for addition to the list of approved signature keys.

```
hash-of-prev cur-time addkey w pubkey sig [comment]
```

The weight (towards `m`) is denoted by `w`.

type `remkey`

A `remkey` entry marks a signature pubkey for removal.

```
hash-of-prev cur-time remkey pubkey
```

type `sigctl`

Denotes an update of `m`, the minimum number of necessary signatures to approve state changes (the threshold).

```
hash-of-prev cur-time sigctl m
```

distributing the current head

- The current head of a hash chain is all one need to fully verify the entire code history and recreate the most current code version with enough signatures, given that one has access to the hash chain and the corresponding patch files.

- But in order to prevent the suppression of updates to certain users, a form of targeted updates, one has to ensure that all users have access to the most current head.

$\Rightarrow$ Employ a so-called single source of truth (SSOT) where every user has access to the same authentic version of a data object.

Single source of truth (SSOT) via DNS

- use widely deployed SSOT system DNS
- store a signed head in TXT record of fully qualified domain name
- the head is signed, allows clients updates to it (trust on first use)
- due to distributed caching of DNS it is not possible for publishers to send different signed heads to different users (no targeted updates by publishers)
- distributing false signed heads through DNS spoofing is prevented **iff** client has seen signed head before (clients cache)
- if valid head not seen before, vulnerable to DNS spoofing
- can be mitigated by deploying the SSOT on a domain with DNSSEC

Signed head specification

- `PUBKEY` (32-byte), Ed25519 public key of SSOT head signer
- `PUBKEY_ROTATE` (32-byte), pubkey to rotate to (0 if unused)
- `VALID_FROM` (8-byte), signed head valid from given Unix time
- `VALID_TO` (8-byte), signed head is valid to the given Unix time
- `COUNTER` (8-byte), strictly increasing signature counter
- `HEAD`, the Codechain head to sign
- `SIGNATURE`, signature with `PUBKEY`

concatenate (integers in big-endian) and encode as base64

Secure package (`.secpkg`) specification

```
{
  "Name": "the project's package name",
  "Head": "head of project's Codechain",
  "DNS": "fully qualified domain name",
}
```

Example '.secpkg' file for Codechain itself

```
{
  "Name": "codechain",
  "Head": "a9712fa5666da7391dc9401e72b8a1a2a94c440b28cf4
  "DNS": "codechain.secpkg.net",
}
```

CreatePkg & SignHead

- CreatePkg: Publish HEAD & distribution HEAD.tar.gz for the first time
- SignHead: Sign current HEAD regularly and update package if necessary
- Signing should happen regularly (timeouts), even if the HEAD didn't change.

The administrator has to upload the distribution HEAD.tar.gz to the download URL and publishes the new DNS TXT record in the defined zone. DNSSEC should be enabled.

Install & Update

```
$ dig -t TXT _codechain-head.codechain.secpkg.net
[...]
_codechain-head.codechain.secpkg.net. 1754 IN TXT "49_tm45BwAdDitsidwlFWXsTOd-PZ71e
yskggnXuECAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABeZrSGAAAAAF6OQYYAAAAAAAAAA
KKlxL6Vmbac5HclAHnK4oaKpTEQLKM9NZXLrLS5k6_nSJP_-S9EmdW0k6a__CM3yYYMjTu8DJBDqr82DPfu
MVuIzeSzYZOSEvFoDJi0LZYV_NzeHlD4DicVCppsttExjDQ"
[...]
```

- Install: install software described by a .secpkg file for the first time.
- Update: Query for updates and perform them, if necessary.

```
$ dig -t TXT _codechain-url.codechain.secpkg.net
[...]
_codechain-url.codechain.secpkg.net. 1800 IN TXT "https://frankbraun.org/secpkg/codechai
_codechain-url.codechain.secpkg.net. 1800 IN TXT "https://taz0.org/secpkg/codechain"
[...]
```

Installing the Codechain package

```
$ secpkg install codechain.secpkg
.secpkg: written
signed head found: 53f2c26d92e173306e83d54e3103ef2e0bd87a561315bc4b49e1ee6c78dfb583
/home/frank/.config/secpkg/pkgs/codechain/signed_head: written
download
http://frankbraun.org/codechain/53f2c26d92e173306e83d54e3103ef2e0bd87a561315bc4b49e1ee6c
env GO111MODULE=on go build -mod vendor -v ./...
env GO111MODULE=on GOBIN=/home/frank/.config/secpkg/local/bin go install \
  -mod vendor -v ./...
```

three Codechain tools for different user roles

1. `codechain` for developers to record code changes and corresponding multi-party reviews in a unmodifiable hash chain.

2. `ssotpub` for admins to publish the head of a hash chain created by `codechain` with a SSOT using DNS TXT records, creating a `.secpkg` file in the process.

3. `secpkg` for users to securely install and update software distributed as `.secpkg` files.

what are the problems (with governance)?

How to enforce m-of-n decisions?

$\Rightarrow$ *Put decision in machine readable file under Codechain's control*

example: server access control with SSH

```
# vi /etc/ssh/sshd_config

Match Group cccontrolled
AuthorizedKeysFile /codechaindir/%u/authorized_keys
```

Mint coordination in Scrit

- Scrit[2] (secure, confidential, reliable, instant transactions) is a federated Chaumian ecash.
- Coins in Scrit are so-called digital bearer certificates (DBCs) issued by mints.
- Scrit mitigates the issuer risk common in other DBC systems by employing n mints in parallel.

$\Rightarrow$ Coordination of mints (m-of-n) by putting mint configuration file under Codechain's control (with m-of-n signers)

[2] `https://github.com/scritcash`

future work: supressing updates

- new hash chain entry to activate interval signatures (devs have to sign regularly)
- multiple DNS heads (every signer has one)

Codechain

- Codechain is available now (v1.1.2)

→ `https://github.com/frankbraun/codechain`

- minimal code base, Go only, cross-platform (tested on Linux)
- ≈9500 lines of code (plus vendored dependencies)
- public domain (`http://unlicense.org/`)
- Codechain depends on the `git` binary (for `git diff`), but that's optional
- Codechain is reviewed and signed with Codechain (2-of-3)

current head of Codechain's hash chain:

`a9712fa5666da7391dc9401e72b8a1a2a94c440b28cf4d6572eb2d2e64ebf9d2`

conclusion

While it is good that code signing is widely deployed now it doesn't solve important attack vectors.

Codechain mitigates:

- key compromise (with multiparty signatures & key rotation)
- developer coercion (with multiparty signatures & key rotation)
- mitigates regression / suppression of updates (with SSOT)

This gives us

- globally identical,
- verifiable,
- reproducible, and
- attributable

binaries build from source, which mitigates targeted updates

conclusion (cont.)

The same mechanism can be used for governance!

acknowledgments: Jonathan "Smuggler" Logan

contacts:

- Email: `frank@cryptogroup.net` (use PGP, key on keyserver)
- `94CC ADA6 E814 FFD5 89D0 48D7 35AF 2AC2 CEC0 0E94`
- Twitter: @thefrankbraun

slides: `https://frankbraun.org/code-reviews-and-governance-with-codechain.pdf`

thank you very much for your attention! questions?