

Chat on #opensourcecornell!

Throughout the talk and cryptoparty, join us on

`#opensourcecornell` on `irc.freenode.net`

If you don't have an IRC client set up, just point your browser to

`https://webchat.freenode.net/?channels=opensourcecornell`

Trust is Key

Integrating PGP into your Free Software development workflow

Patrick M. Niedzielski

`pmn25@cornell.edu`

`pniedzielski@hummstrumm.org`

Open Source Cornell

April 22, 2014

- pmn25, pniedzielski
- PGP: 0xDEBFA176
- Freshman in Arts and Sciences
 - Prospective CS and Ling major
- Runs two Free/Open Source Software Projects
 - The Humm and Strumm Project, highly concurrent, cross-platform 3D game engine in C++11/14
 - cipra Unit Testing Framework, C++11 unit testing library based on Perl's Test::More
- C++, Perl, Haskell <3
- pniedzielski.wordpress.com



www.hummstrumm.org



cipra.sourceforge.net

I have trust issues.

- *Trust* is what you assume to be secure.
- Your axioms.
- Big problems when they aren't true!

I have a computer. I care a lot about it. I want to keep it secure.

Who am I trusting?

I have a computer. I care a lot about it. I want to keep it secure.

Who am I trusting?

- Myself.

I have a computer. I care a lot about it. I want to keep it secure.

Who am I trusting?

- Myself.
- Those with physical access to it.

I have a computer. I care a lot about it. I want to keep it secure.

Who am I trusting?

- Myself.
- Those with physical access to it.
- Those who developed the software on it.

I have a computer. I care a lot about it. I want to keep it secure.

Who am I trusting?

- Myself.
- Those with physical access to it.
- Those who developed the software on it.
- Those who distributed the software on it to me (!)

I have a computer. I care a lot about it. I want to keep it secure.

Who am I trusting?

- Myself.
- Those with physical access to it.
- Those who developed the software on it.
- Those who distributed the software on it to me (!)
- The medium on which the software was distributed (!!)

I have a computer. I care a lot about it. I want to keep it secure.

Who am I trusting?

- Myself.
- Those with physical access to it.
- Those who developed the software on it.
- Those who distributed the software on it to me (!)
- The medium on which the software was distributed (!!)
- Etc.

Trust

How can I trust all these people?

- Personal experience?
- Past work?
- Because everyone else trusts them?

How can I trust all these people?

- Personal experience?
- Past work?
- Because everyone else trusts them?

How can I trust that I'm trusting the right people?

- Names?
- Email addresses / usernames?
- ...?

Trust

How can I trust all these people?

- Personal experience?
- Past work?
- Because everyone else trusts them?

How can I trust that I'm trusting the right people?

- Names?
- Email addresses / usernames?
- ...?

There is no good way to know.

I have a Git repository. I care a lot about it. I want to keep it secure.
Who am I trusting?

I have a Git repository. I care a lot about it. I want to keep it secure.

Who am I trusting?

- Myself.

I have a Git repository. I care a lot about it. I want to keep it secure.

Who am I trusting?

- Myself.
- Ricardo Tiago
- Tim Walters

I have a Git repository. I care a lot about it. I want to keep it secure.

Who am I trusting?

- Myself.
- Ricardo Tiago
- Tim Walters
- Developers without commit bit.

I have a Git repository. I care a lot about it. I want to keep it secure.


Who am I trusting?

- Myself.
- Ricardo Tiago
- Tim Walters
- Developers without commit bit.
- Sourceforge (!)

Or GitHub

(or Gitorious, or Bitbucket, or Savannah, etc)


March 14, 2012:

 rails / rails

★ Star 21,363 Fork


wow how come I commit in master? O_o

master v4.1.0.rc2 v4.0.0

 **homakov** authored 2 years ago 1 parent [4d391a4](#) commit [b83965785db1eec019edf1fc272b1aa393e6dc57](#)

Showing 1 changed file with 3 additions and 0 deletions.

Show Diff Stats


3     hacked

 show inline notes

View

Patrick M. Niedzielski


Trust is Key



April 22, 2014

10 / 41


March 14, 2012:

 rails / rails

★ Star 21,363 Fork

wow how come I commit in master? O_o

master v4.1.0.rc2 v4.0.0

 homakov authored 2 years ago 1 parent 4d391a4 commit b83965785db1eec019edf1fc272b1aa393e6dc57

Showing 1 changed file with 3 additions and 0 deletions.

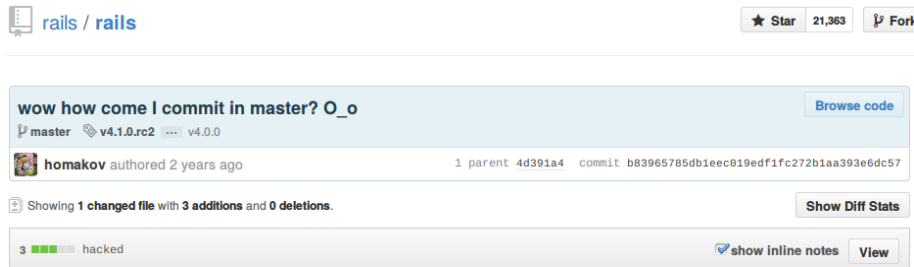
3 hacked

show inline notes View

You don't really control your project Git repository on GitHub.

Trust

March 14, 2012:



The screenshot shows a GitHub commit page for the `rails / rails` repository. At the top, it displays the repository name and navigation links for `Star` (21,363) and `Fork`. The commit message is "wow how come I commit in master? O_o". Below the message, it shows the commit details: `master` branch, commit `4d391a4`, and commit hash `b83965785db1eec019edf1fc272b1aa393e6dc57`. The commit was authored by `homakov` 2 years ago. A diff summary indicates "Showing 1 changed file with 3 additions and 0 deletions." and a "Show Diff Stats" button. At the bottom, there is a "hacked" status with 3 green squares and a "show inline notes" button.

You don't really control your project Git repository on GitHub.
Sometimes GitHub doesn't either.
This is trust.

I have issues with trust.

Observation:

Free Software development and distribution relies on trust.

Observation:

The Internet doesn't have a good, reliable, default way to facilitate trust.

Problem:

We need a way to ensure the *correctness* (i.e., comes from the trusted source) and *integrity* (i.e., was not modified between the trusted source and us) of arbitrary data.

Solution:

PGP.

What is PGP?

- **P**retty **G**ood **P**rivacy
- Cf. OpenPGP, GPG (GnuPG)
- Technology for *encrypting* and for *signing*.
- Collectively I'll call these *Crypto*.

What is PGP?

Encryption

- Obviously useful in certain cases.
- Personal communication: if I can, I encrypt.
- I won't be talking about it much.
 - Free/Open Source Software projects are generally done out in the open.
 - Encryption doesn't really help in this model.

What is PGP?

Signing

- If Alice signs her message, Bob should be able to tell that:
 - ① It's from Alice, and no one else. (correctness ✓)
 - ② It wasn't modified in transit, maliciously or otherwise. (integrity ✓)
- **PGP allows this.**

How can we trust PGP?

- Uses *public-key cryptography*.
- Remember that from CS2800?
 - Boils down to the fact that it's really easy (polynomial time) to multiply two prime numbers, but really hard (exponential time) to factor the result back to those prime numbers.
 - Especially when the numbers are very, very large.
 - We can make a *one-way function* that's fast to perform, but slow/practically impossible to undo.
 - $P=NP$?
- Cf. Shor's Algorithm, BQP time, $O((\log N)^3)$ in $3N$ qubits to factor binary number N .

How can we trust PGP?

The algorithm:

- We have two prime numbers p and q , and a number N such that $N = pq$.
- We choose d, e such that for any $A < N$,

$$A = (A^e \bmod N)^d \bmod N$$

- $M = (A^e \bmod N)$ is our encrypted message.
- $A = (M^d \bmod N)$ is our decrypted message.
- For messages larger than N , we either do it in blocks (encryption, guaranteed round-trip, but takes longer) or hash the message down to $< N$ bits (signing, depends on cryptographic security of hashing function).

How can we trust PGP?

We say the tuple (e, N) is the *public key*, and the tuple (d, N) is the *private key*. We allow everyone to know the public key, but keep the private key hidden so that no one else knows it.

This is common to all Public-Key Crypto: PGP, SSL/TLS, SSH, ...

How can we trust PGP?

Encryption

Anyone can encrypt to us

$$M = A^e \bmod N$$

but **only we can read it.**

$$A = M^d \bmod N$$

How can we trust PGP?

Signature:

Only we can sign

$$S = A^d \bmod N$$

but anyone can verify it

$$A = S^e \bmod N$$

How can we trust PGP?

As long as the private key is held private, we can trust.

How do we use PGP?

We can sign any data we want to allow the user to verify:

- Debian, Fedora, etc packages
- Upstream release tarballs
- Release announcements
- Git tags
- Emails
- Git Commits (!)
- ...?

How do we use PGP?

For GNU/Linux, (technically OS X, Windows, too):

```
# Generates a keypair
$ gpg --gen-key
# Generates a revocation certificate
$ gpg -o revcert.asc --gen-revoke 12345ABC
# Sign a file , makes filename.gpg
$ gpg -s filename
# Sign file in same file , makes filename.asc
$ gpg --clearsign -a filename
# Verify a signature or clearsigned message
$ gpg --verify filename.asc
# Encrypt a file , makes filename.gpg
$ gpg -e filename
# Decrypt a file and print its contents
$ gpg --d filename.gpg
```

Your mileage may vary (Debian, Ubuntu should use `gpg2`, GPG4Win has graphical tool for making keys, etc). Details on handout.

Great!

Great!
But wait...

Problem:

How can we trust that the private key owner is who they say they are?

Solution:

Key Signing.

Key Signing

- \neq Crypto signing
- When you sign a key, you're saying that you have verified that the key belongs to the person it says.
- You need to know the key information (names, email addresses, comments, photos attached to the key).
- You need to know their personal information (from government issued photo ids, other documentation)
- Do they match?
- **Only sign what you have verified! Only sign if you have verified!**

- **Names:** Does the name on the key match the name on the photo ID?

Key Signing

- **Names:** Does the name on the key match the name on the photo ID?
- **Emails:** Sign each email on the key separately, encrypted your signed key, and email that signed key to the email. The key will be uploaded only if it can be decrypted (i.e., only if the key owner also controls the email account).

Key Signing

- **Names:** Does the name on the key match the name on the photo ID?
- **Emails:** Sign each email on the key separately, encrypted your signed key, and email that signed key to the email. The key will be uploaded only if it can be decrypted (i.e., only if the key owner also controls the email account).
- **Photo:** Does it match the photo ID and the person? (*There isn't really a need for this, because it should already be verified by the name check. Also, people's appearances change much more often than their names → less trust.*)

- **Names:** Does the name on the key match the name on the photo ID?
- **Emails:** Sign each email on the key separately, encrypted your signed key, and email that signed key to the email. The key will be uploaded only if it can be decrypted (i.e., only if the key owner also controls the email account).
- **Photo:** Does it match the photo ID and the person? (*There isn't really a need for this, because it should already be verified by the name check. Also, people's appearances change much more often than their names → less trust.*)
- **Comment:** Other documentation needed! (*Again, seldom needed, because name verification should be sufficient. Also makes it much harder to establish trust.*)

Key Signing

There's hidden trust in this! You're trusting:

- Yourself, to follow this process.

Key Signing

There's hidden trust in this! You're trusting:

- Yourself, to follow this process.
- The other person, to keep their private key secret.

There's hidden trust in this! You're trusting:

- Yourself, to follow this process.
- The other person, to keep their private key secret.
- The photo ID authority, to have verified this person's information. *

Key Signing

There's hidden trust in this! You're trusting:

- Yourself, to follow this process.
- The other person, to keep their private key secret.
- The photo ID authority, to have verified this person's information. *

A good rule of thumb is two photo IDs to verify, and more documentation if there is a comment. One photo ID can be sufficient. Use good judgment.

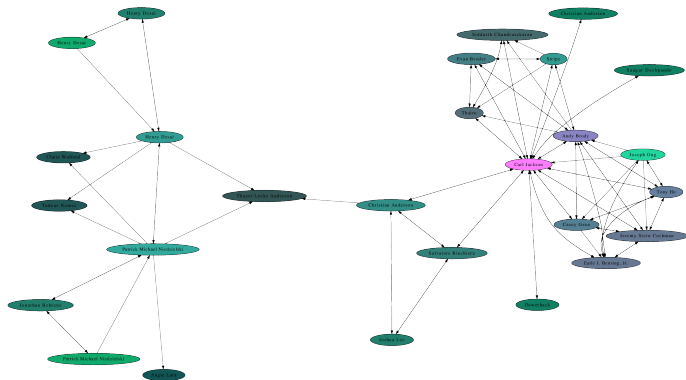
Problem:

I personally have to verify every single person I trust.

Solution:

Web of Trust

Web Of Trust (WoT)



- Alice signed Bob's key, and Bob signed Carol's key. Why can't Alice trust Carol?
 - She can, if she trusts that Bob followed the proper procedure in signing Carol's key.
 - Trust both in the procedure and in others to follow the procedure.
- Carol is said to be in Alice's *web of trust*.
- **This is what distinguishes PGP from SSL/TLS and SSH.**

Web of Trust (WoT)

- Maybe Bob seemed a little fishy.
- Let's add a *trust level* to each key we know about:
 - *Untrusted*
 - *Marginal* (need two paths through marginal keys to trust a key)
 - *Complete* (need one path through complete key to trust a key)
 - *Ultimate* (you own the key)
- This is not transitive, nor published!
 - If Alice signed Bob, Bob signed Carol, and Carol signed Dave, then Alice needs to assign a trust level to Carol to trust Dave.
 - If she doesn't know Carol, she can't trust Dave.
- It's up to you whether you rely only on signature paths or on trust levels. Signature paths give a bigger web of trust, but there's also a lot more trust involved!

Expand your web of trust!

- Have people sign your keys!
- Attend Cryptoparties (like this one!)

- Trust is at the center of computer security, but also at the center of Free Software development.
- PGP gives us a way to formalize trust through Crypto and transform trust into true security.
- It does this by allowing us to sign keys we trust, associating people with keys.
- With a Web of Trust, we have far more people we can trust.

Questions? Comments?

- These slides (including source code) will be posted on my website following the talk.
- Cryptoparty!
 - There are papers up at the front for setting up keys and signing others' keys on GNU/Linux and UNIX-y systems, OS X, and Windows. Two sheets for each.
 - If you need a key set up, ask on IRC or come talk to one of us.
 - If you want to help set up keys, go for it!
 - Sign keys! Make sure you follow the keysigning procedure!
- If you're interested in computer security or in the details of how PGP works, come up to me and chat.



Copyright © 2014, Patrick M. Niedzielski. This work is released under the Creative Commons Attribution 4.0 United States License.

I give NO WARRANTY to the validity or truth of the statements made within this presentation. This presentation represent my views, plans, opinions, and ideas, not those of my employer, university, or any organizations to which I belong. The opinions expressed are solely my own.

All of the images in this presentation I either created, have a license to use in this manner, am using under the United States Fair Use doctrine, or are trademarks which I use under U.S. Trademark law. If you are the rights holder of any one of the images, and you believe that I do not have the rights to use that image, please contact me.