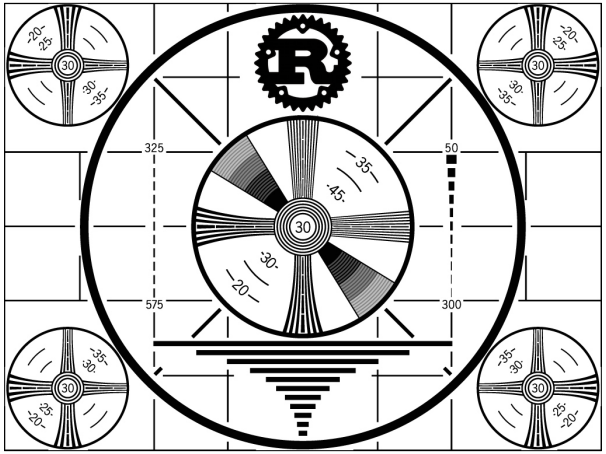
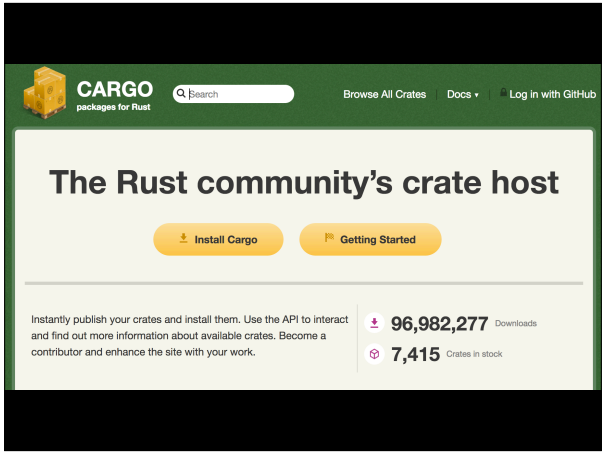
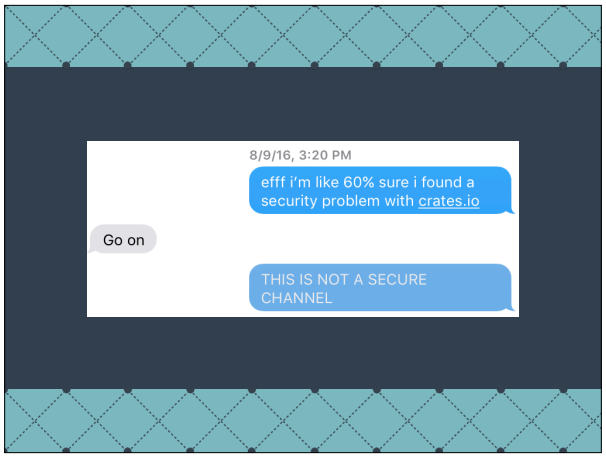




Picture this:
- working on OSS, crates.io
- hosts Rust libraries
- found something strange
- followed to a security issue; circumstances where someone could publish to someone else's crate
- what did I do?



I kinda freaked out.

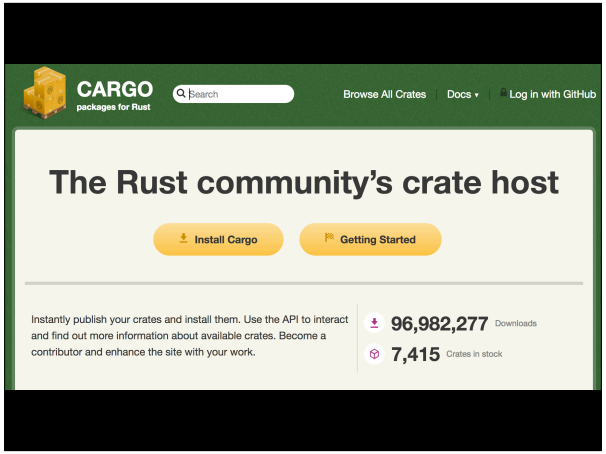
- wasn't prepared
- lots of googling
- actually more likely than i had thought - security isn't my whole job, but it is part, i do know more than the average web user
- you should be prepared too

HOW TO DISCLOSE A SECURITY VULNERABILITY

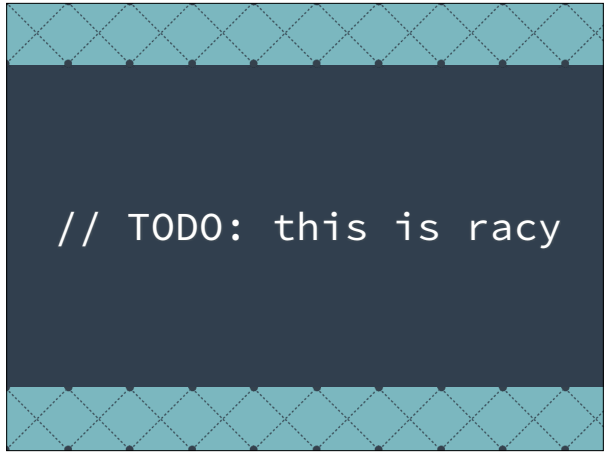
Carol (Nichols || Goulding)
@carols10cents

AGENDA

- Security issue I found
- Steps to make a disclosure
- Kinds of disclosures
- Case Studies
- To-do list



back to crates io. web app, rust backend

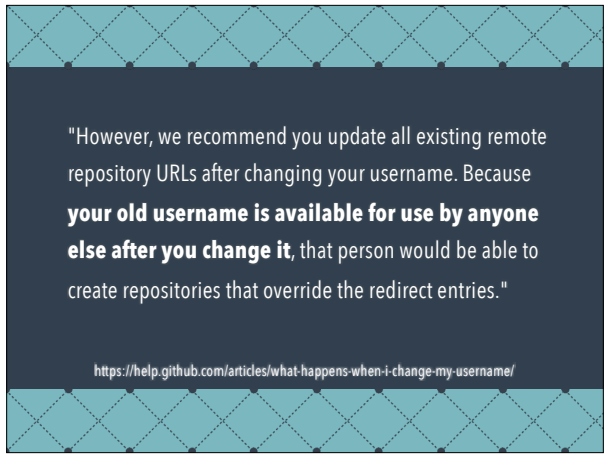


- improving comments like this
- changing places that did 2 queries (select + insert/update, update + insert)
- to use 9.5's UPSERT feature that does this in 1 query

```
impl User {
    /// Updates a user or inserts a new user
    /// into the database.

    fn find_or_insert(…, username: &str, …) {
        …
        try!(conn.query("\
            UPDATE users
            SET gh_access_token = $1,
                email = $2,
                name = $3,
                gh_avatar = $4
            WHERE gh_username = $5
            RETURNING *",
            &[&access_token, &email, &name,
              &avatar, &username]));
        …
```

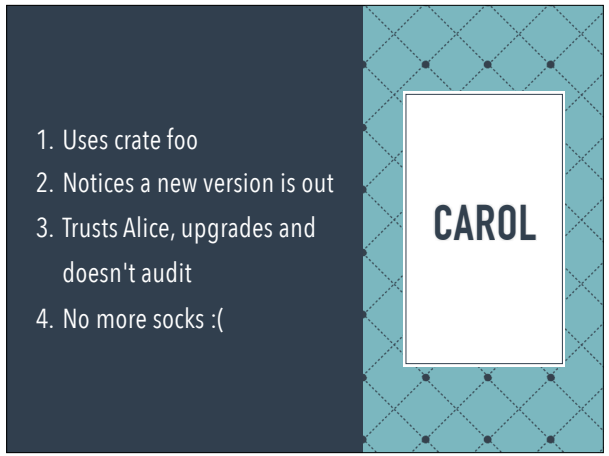
- gets info from github, tries to match to a user record on github username

- upsert needs a unique index
- users table didn't have a unique index on github username
- concerned with inconsistent, incorrect data
- started thinking up scenarios to get 2 user records with the same username
- end up on github help page for changing your gh username

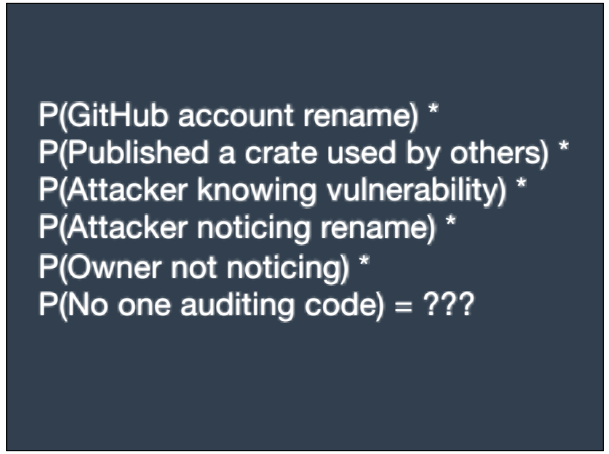


Github help for renaming your account
we had a problem, nothing to do with duplicate rows though
just due to looking up user records by github username

– will come back to forgetting in a minute



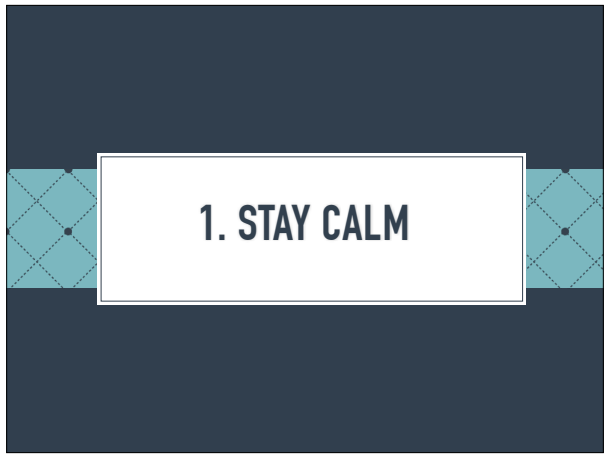
let's do a little threat modeling to think about how bad this is



- core team member had renamed their github account!
- some people had renamed their account and lost access to their crates, contacted admins who fixed the db manually
- scale of 0 to heartbleed, pretty low but not 0

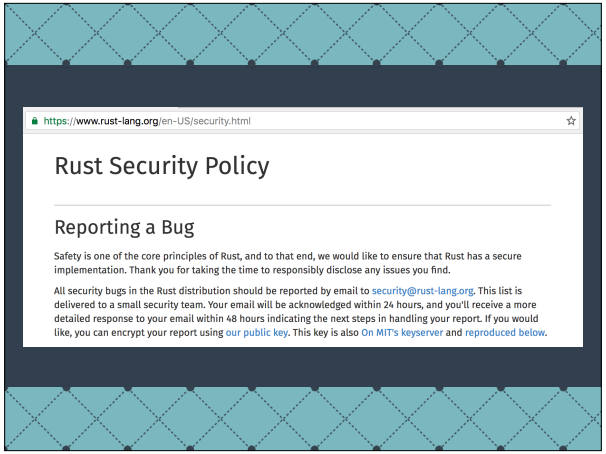
so once i found this, what did i do, what should i have done?
my experience was pretty easy, going to add in tips i learned along the way

---



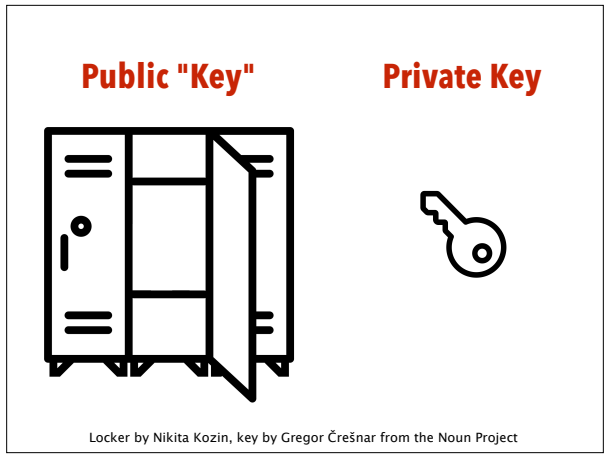
insecure: email, irc, slack, public forums, twitter
secure: signal, encrypted email

– want them to believe you and understand the severity



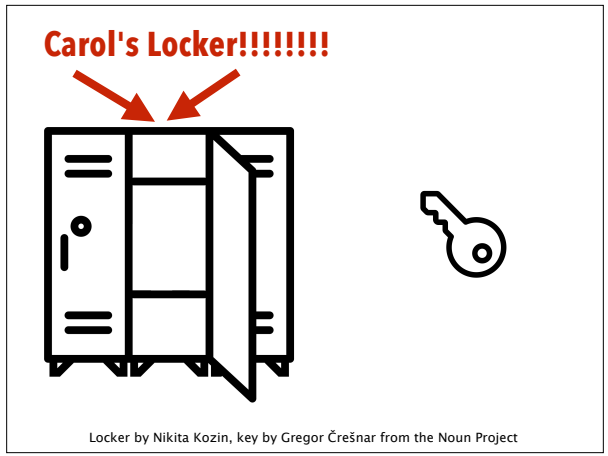
search for security, disclosure, IT dept

- quick, high level overview of public key encryption

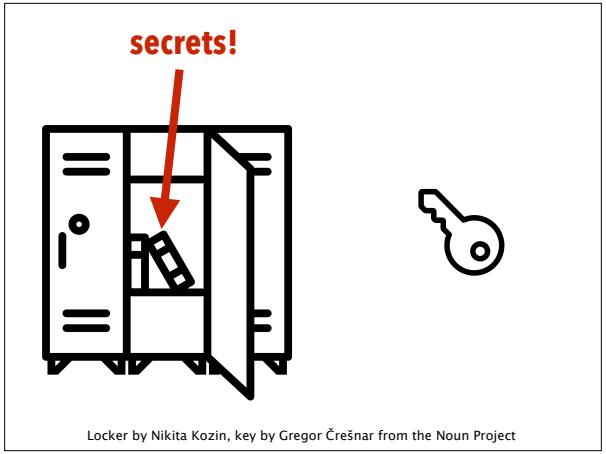


public key is more like an open locker, to which you hold the private key

---



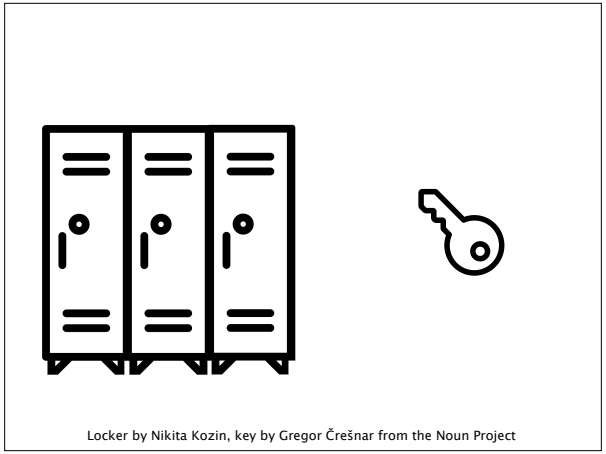
You tell everyone on the internet where your locker is and that it's yours, and leave it open. You don't ever give out your key.



Locker by Nikita Kozin, key by Gregor Črešnar from the Noun Project

If someone wants to give something secret to you, they use your public key to secure the secrets, which is kind of like putting the secrets in your locker

---



Locker by Nikita Kozin, key by Gregor Črešnar from the Noun Project

and then shutting the door, so that only the person with the private key can get the secrets.

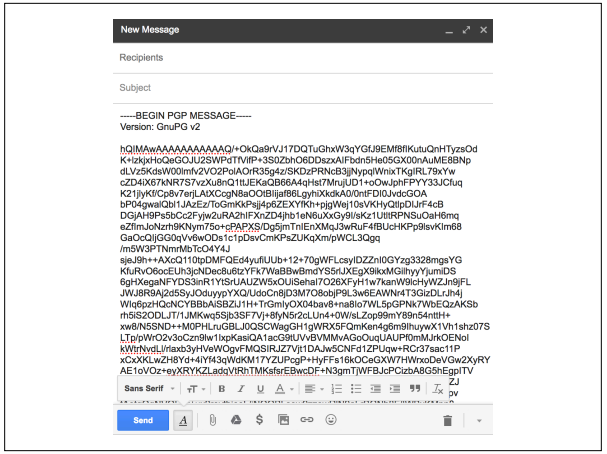
not going to go through how to generate keys, will be in the reading list

So we want to take the really good bug report and use the security team's public key to email it to them securely.

one thing I didn't realize is I didn't need to be able to encrypt ALL my email by integrating it into an email client - could just encrypt and

---

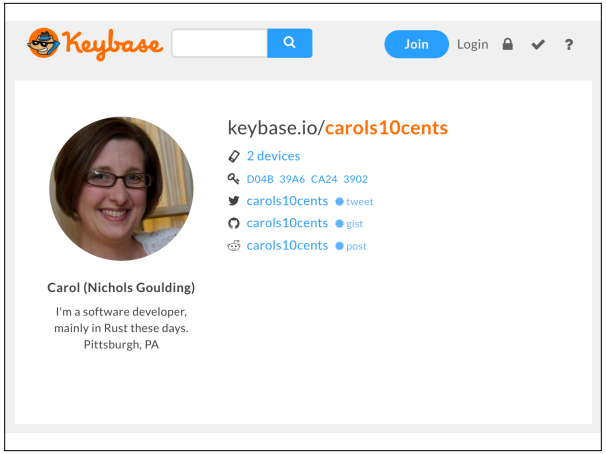
```
$ gpg2 --import recipients-public-key.asc

# write the body of your email in message.txt
# e = encrypt, a = ascii, r = recipient
$ gpg2 -ea -r Recipient message.txt

$ pbcopy < message.txt.asc
# paste into an email!
```



- keep in mind subject is not encrypted
- Rust team requested a meaningful subject because of spam, check guidelines
- feel like a spy

so that they can respond and ask followup questions
good idea for them to be able to verify it belongs to you

MIT key server
rust security team found it, i had forgotten but still had private key
why was i doing this on xmas 2014



- proof that person who owns this public key controls these accounts
- not perfect but more than nothing
- once you have an encrypted reply, copy all the garbage

```
# Copy encrypted reply
# from your email into reply.txt.asc
$ pbpaste > reply.txt.asc

# d = decrypt
$ gpg2 -d reply.txt.asc

# enter your password and receive plain text!
```

- when fix will be available, when you or they will publish
- at least when they'll give you an update
- more important with less cooperative vendors

don't dream of hackers
hard to not talk about it on github, irc

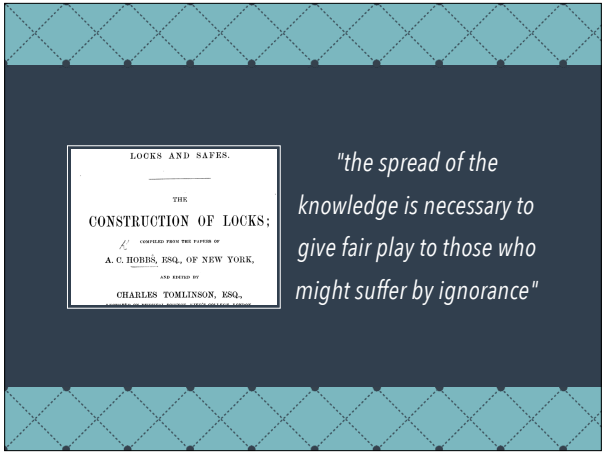
- software needs to be tested
- Bug Hunters Diary – sometimes needed iterations with the vendor
– fix for my issue: add github ID to user records (doesn't change on account rename), backfill, add unique index

---

- You, or the vendor, or both
- inform users of risks, mitigation, any action they need to take
- share what you learned
– thought i was done learning – i was wrong

TYPES OF DISCLOSURES

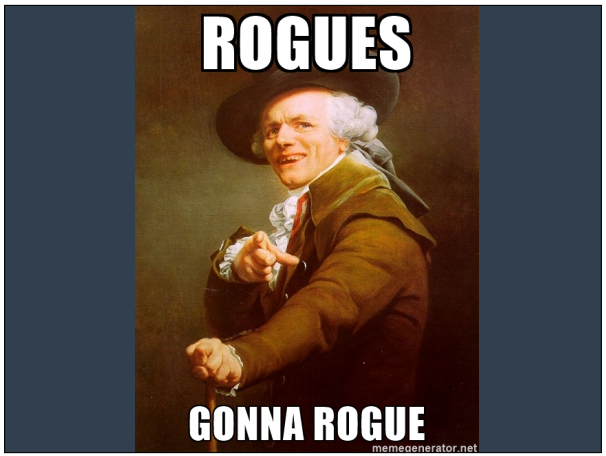
And with that, I walked into a controversy that I had no idea about: when you know about a vulnerability, what's the best thing to do with that information?



Which is a controversy that's been going on since at LEAST 1868, as documented in this book about The Construction of Locks

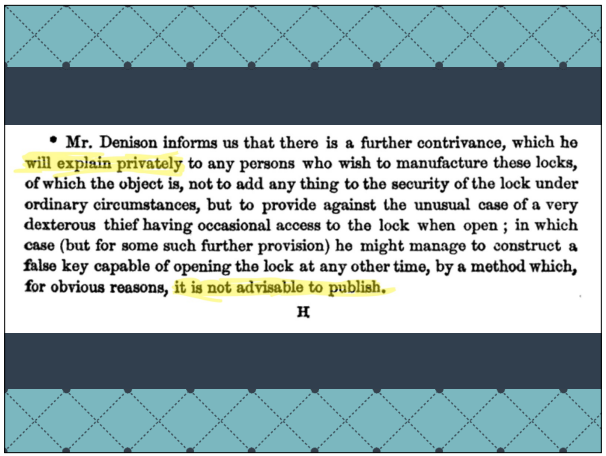
Great Exhibition of 1851 – picked "unpickable" locks, bug bounties, embarrassed the vendor, vendor tried to get out of paying, better locks were made

to summarize and translate to today's parlance,



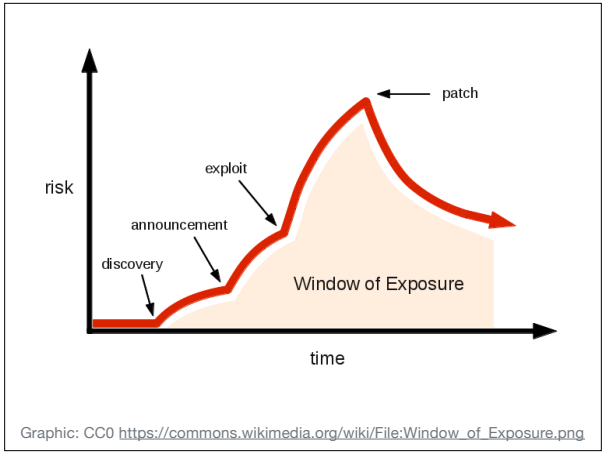
this debate was not settled in the 1800s, nor was it even settled in this book

---


• Mr. Denison informs us that there is a further contrivance, which he will explain privately to any persons who wish to manufacture these locks, of which the object is, not to add any thing to the security of the lock under ordinary circumstances, but to provide against the unusual case of a very dexterous thief having occasional access to the lock when open ; in which case (but for some such further provision) he might manage to construct a false key capable of opening the lock at any other time, by a method which, for obvious reasons, it is not advisable to publish.

H

later part refused to print some details, contradicting its earlier statement

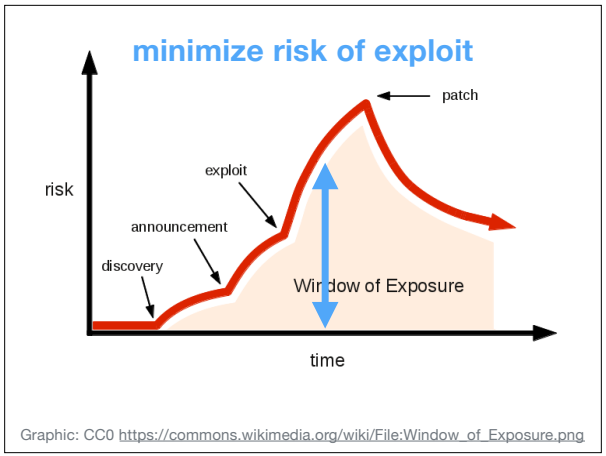


Graphic: CC0 https://commons.wikimedia.org/wiki/File:Window_of_Exposure.png

- Bruce Schneier coined term called window of exposure
- to minimize amount of time and risk users are exposed, minimize the area under the curve.
- anything trying to do that could be described as responsible
- two main ways of doing that: coordinated disclosure and full disclosure

- Scott Culp at MS coined responsible, microsoft later asked for a change to coordinated bc people argued that full disclosure could be the most responsible thing

- vendors like this most: control, save face
- bug bounty programs = carrot, we wont sue you = stick
- bug clearing houses like CERT
- sometimes vendors aren't motivated to fix

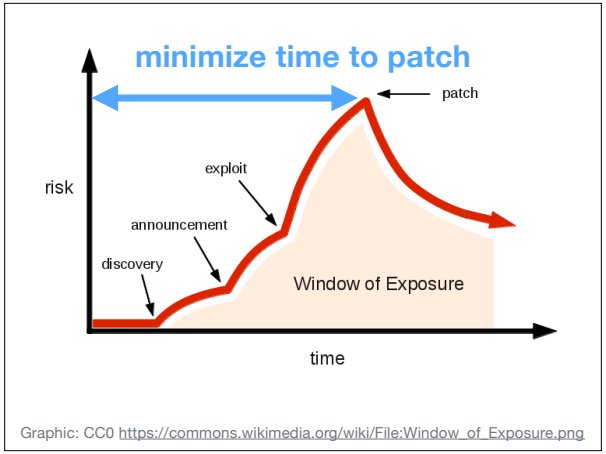


Graphic: CC0 https://commons.wikimedia.org/wiki/File:Window_of_Exposure.png

- if people don't know about it, there won't be an exploit of it
- might increase time to patch

FULL DISCLOSURE

- Tell everyone all the details immediately
- Most pressure on vendor
- Most info to users*
- Most info to exploiters**



Graphic: CC0 https://commons.wikimedia.org/wiki/File:Window_of_Exposure.png

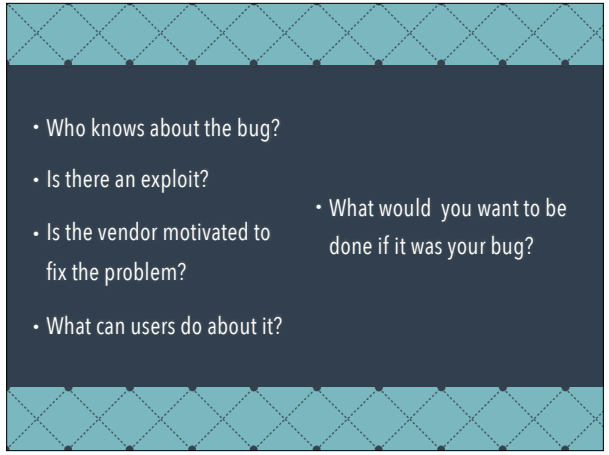
- full disclosure: trying to minimize time to patch

NON DISCLOSURE

- Don't publish any details, ever
- Least information to everyone

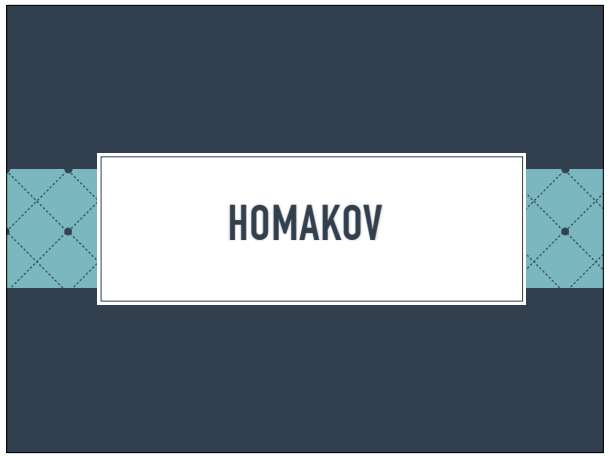
NSA likes it

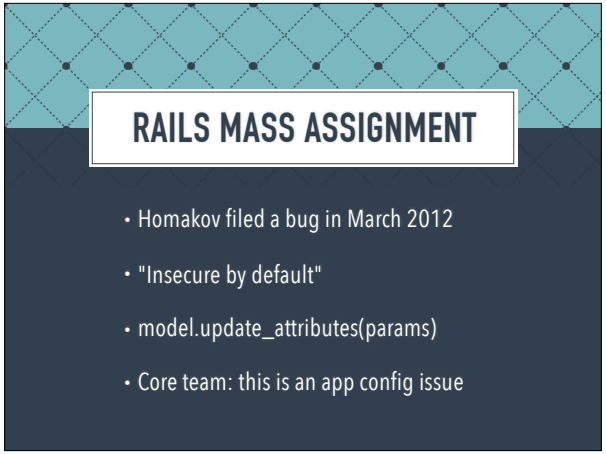
deciding which kind of disclosure is an ethical decision depending on many factors



i had it easy, the vendor was people i knew and trusted

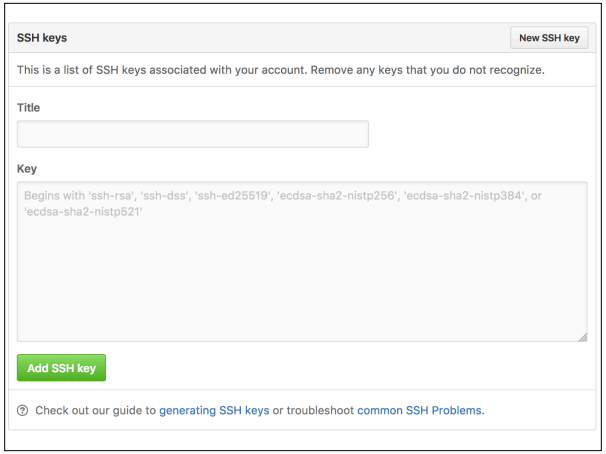
Picked other open source projects





homakov's point: no one does this app config, rails should change the default

---

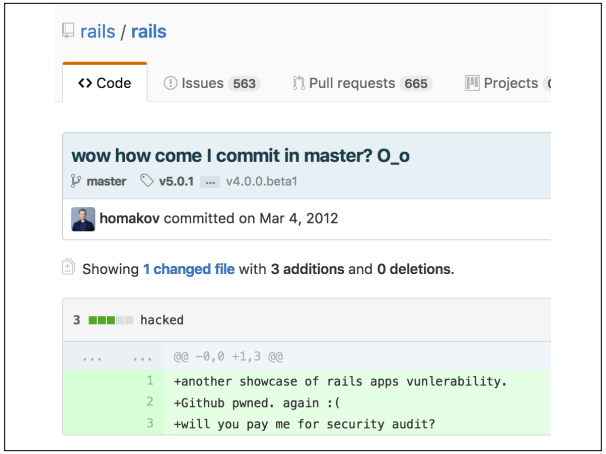


- Homakov tested on github since they were a rails app
- tried all the forms, including the one to edit a public key in your account
- this form doesn't even exist anymore, you can only add or delete keys

---

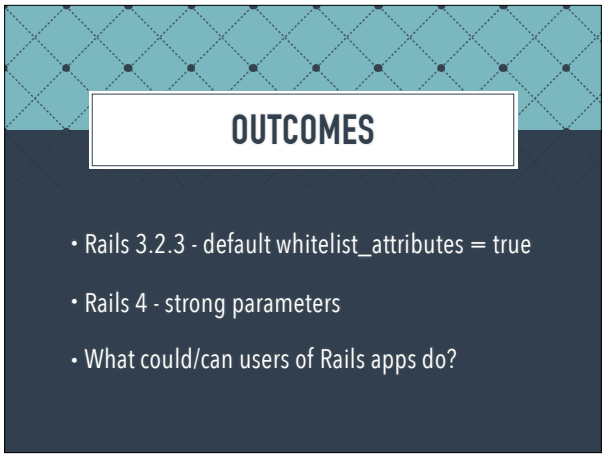
```
PublicKey
  :id, :user_id, :value


class PublicKeyController < ApplicationController
  def update
    @current_key = PublicKey.find_by_id params['key']['id']
    @current_key.update_attributes(params['key'])
  end
end


PUT github.com/public_keys/123 // <- Homakov's key
    {key: { user_id: 555 }}// <- Rails committer's user id
```

- guessed at models/params
- normally shouldn't be editing user_id, but this code happily let you
- changed a rails committer's key to homakov's



- which let him commit to master of rails. this got some attention!

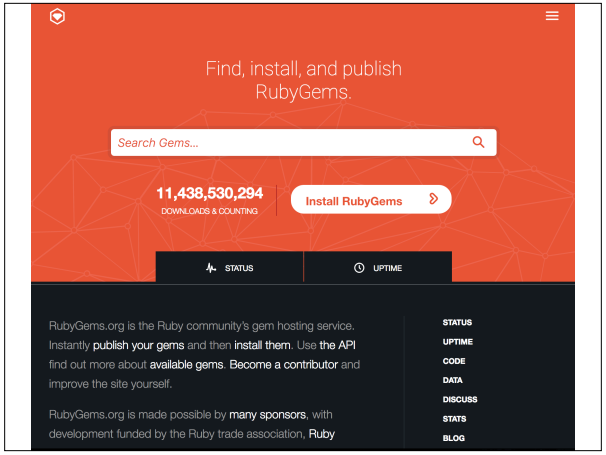
- github suspended his account for a few days, gave it back
- in 2014, he updated his blog post with regrets



prompted the core team to address the underlying issue and change the defaults

whitelist attrs = model level
strong params = controller level

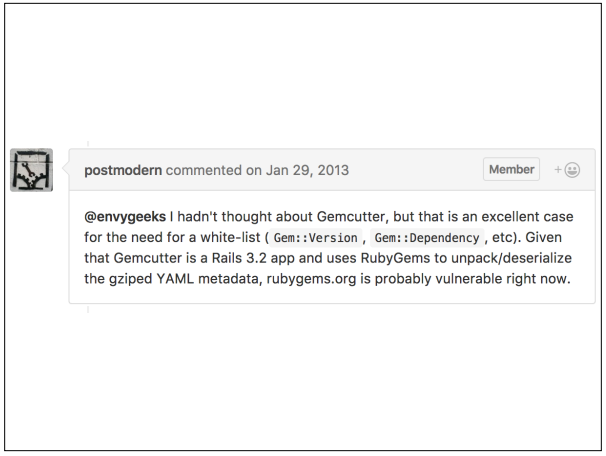
Strong parameters available as a gem later in March 2012
3.2.3: End of March 2012
Rails 4.0: June 2013
long tail of upgrading



rubygems::ruby as crates.io::rust

- yaml is yet another markup language
- psych is a gem that deserializes yaml into ruby objects
- ANY ruby objects
- security announcement involving Rails parameter parsing - early Jan 2013
- by mid jan, discussion on how to make loading yaml safer on a psych issue

---

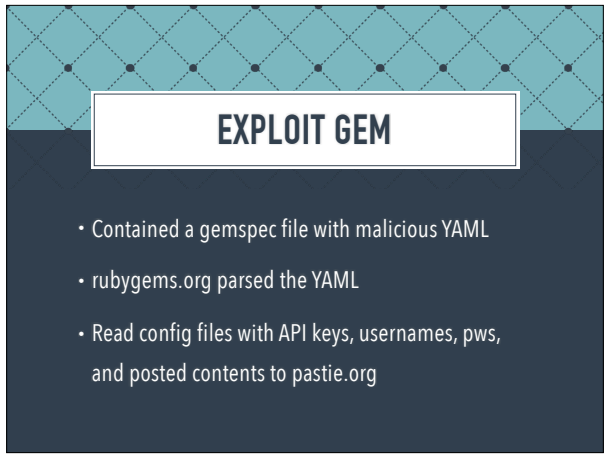


by jan 29, there were public discussions about the fact that rubygems.org parses user-supplied yaml

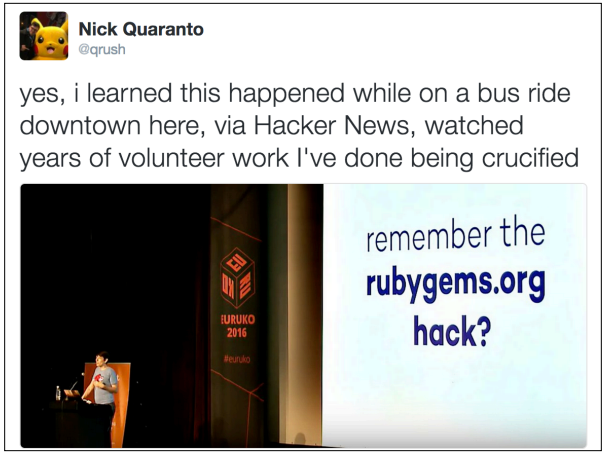
```
--- !ruby/object:Gem::Specification
name: rails
version: !ruby/object:Gem::Version
  hash: 3
  prerelease:
  segments:
  - 3
  - 1
  - 0
  version: 3.1.0
platform: ruby
```

- when you package a gem to be uploaded to rubygems.org, the gem specification gets serialized as yaml.
- rubygems.org then needs to parse that gempspec.

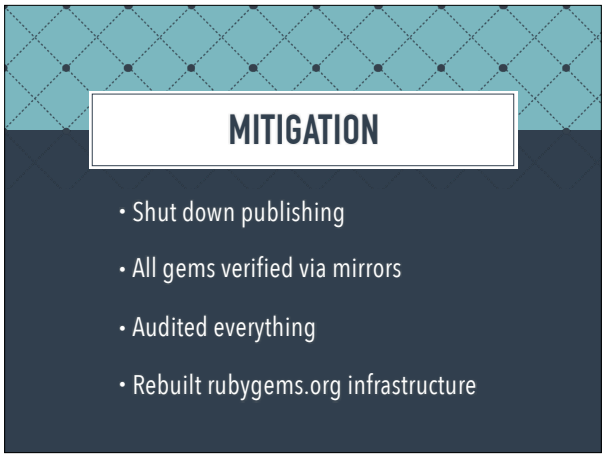
- discussions about how the rubygems.org devs had been notified
- this was at 1am EST



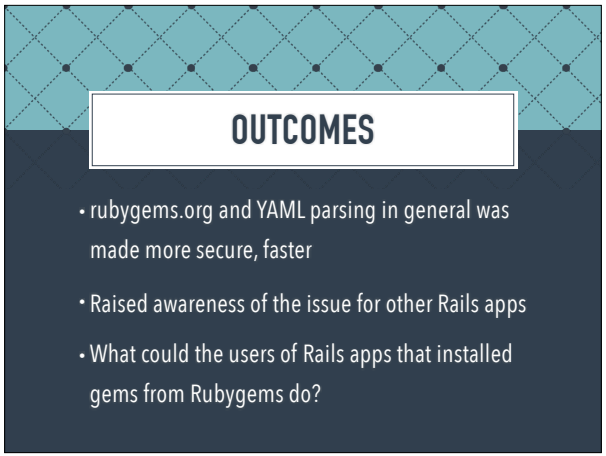
- by 9:30am est, there was a gem uploaded to rubygems.org
- literally named it exploit, so it wasn't trying to be sneaky, but it was still trying to do nefarious things
-

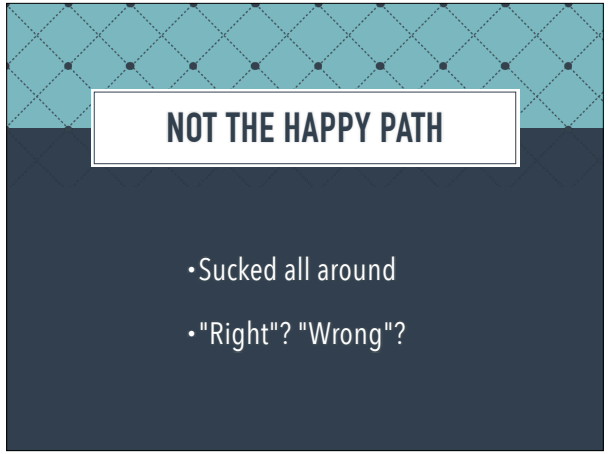


Especially with open source, think about the maintainers.



at least 2 long days of a lot of work





- could this have been worse? could this have been better?

- what would i have done if i knew crates.io was vulnerable to a worse bug and i was having trouble getting hold of the maintainers?
- practice hypotheticals with yourself before you're panicking