Filippo Valsorda

08 Jan 2023

SSH WHOAMI.FILIPPO.IO

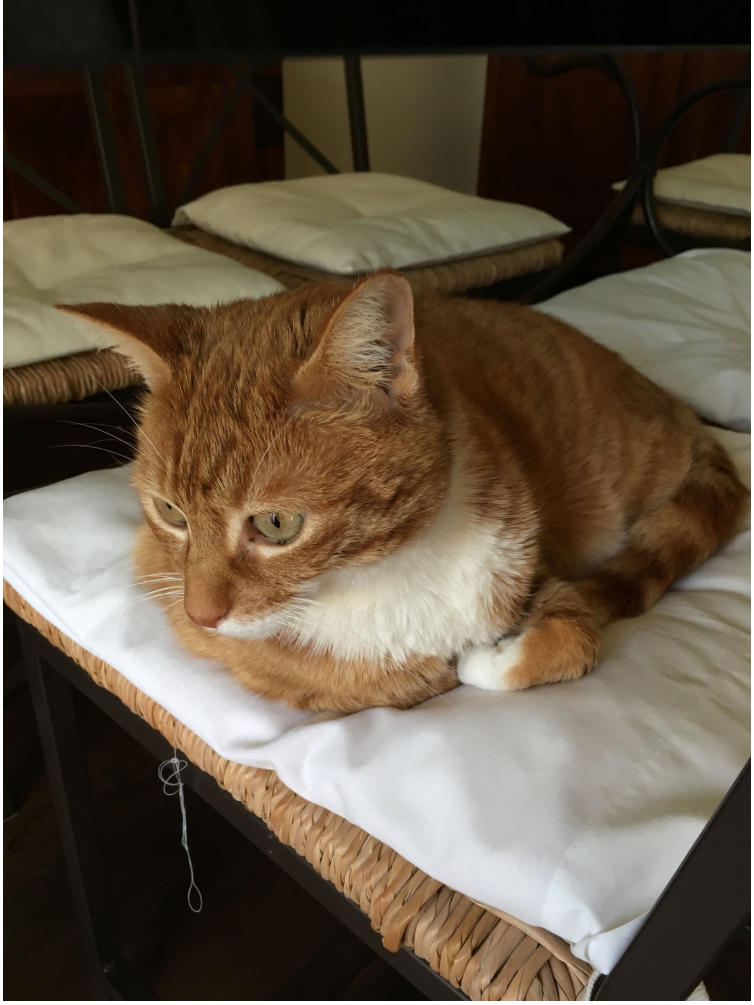
This is an issue of Cryptography Dispatches, my lightly edited newsletter on cryptography engineering. Subscribe via email or RSS.

I updated the `whoami.filippo.io` dataset over the holidays, so it should be pretty accurate at least for a little while. If you already know what I'm talking about, below are some tidbits about how I fetched the new dataset and how it's stored.

If you don't, stop reading, and run this. I'll wait.

```
$ ssh whoami.filippo.io
```

Here's a picture of my grandmother's cat, to avoid spoilers.



What?!

There are two things going on here that might be unexpected.

The first is that the SSH protocol provides to the server all public keys the client is willing to produce signatures for, which by default are all the public keys in your `ssh-agent` and in your `~/.ssh/id_*`. This is somewhat unavoidable. You could technically make an authentication protocol that doesn't disclose public keys to a peer that didn't already know them, but it would be annoying: for example, you couldn't use ECDSA or RSA without

first having the server prove knowledge of the public keys, because it's usually possible to recover the public key from those signatures. Having the server prove knowledge of the public keys before the client uses them is also kinda annoying, because you also don't want the client to learn about the public keys accepted by the server. You end up with a complex interactive protocol that wastes round-trips.[1]

(This is also why we can do `git clone git@github.com` instead of `git clone FiloSottile@github.com`. GitHub relies on the client sharing its public keys to know who is trying to authenticate. Which is also why you can't use the same SSH key for two accounts.)

How it technically works is that the client sends public keys to the server until the server answers that it likes one of them, and then the client sends a signature from that key.[2] The client is allowed to skip the first part and start sending signatures right away, but it doesn't because producing a signature might require user interaction (for example to decrypt the private key or to enter the PIN of a hardware token) and it's bad UX to require that for a key the server will reject. This is also why age ciphertexts encrypted to SSH keys carry a hash of the public key: to let the client know if it should bother the user to decrypt an encrypted private key.

(A neat consequence is that you can test what public keys a server accepts even without having the corresponding private key.)

The second thing going on that might be unexpected is that your GitHub SSH keys are, well, public. For example, you can see mine at https://github.com/FiloSottile.keys.

I knew this, what's new?

`whoami.filippo.io` has been running since 2015, originally on a dataset collected by Ben Cox. What's new is that I now have a faster way to refresh its keys database, and that it runs on new architecture.

The GitHub GraphQL API now includes users public keys, and since it allows fetching 100 users per request and 5000 requests per hour[3] it's significantly faster than using the REST API and the `.keys` endpoint. What it lacks though is the ability to iterate through all users.

You can make a search for all users, which will tell you there are 97,616,627 users at the time of this writing, but you can only fetch at most 1000 results from a search, and they don't come in any clear order, so you can't just make the next search start where the previous one left off (or I didn't figure out how).

What you can do though is request accounts created in a certain time range. If you get the time range right, so that it has less than 1000 entries, you can paginate through it, and then request the next time range. This was a little easier said than done, because of course registrations come in waves and the rate changes over the years, but I eventually built a simple adaptive algorithm that rarely overshot, and that went through all users in less than a couple weeks without ever hitting the rate-limits. (That means it could have been a little faster with some concurrency, but good enough.) This is how the final GraphQL query looked like:

```
{
  search(
    type: USER
    query: "type:user created:{{ .From }}..{{ .To }}"
    first: 100
    {{ if .After }}after: "{{ .After }}"{{ end }}
  ) {
    userCount
```

```
      pageInfo {
        hasNextPage
        endCursor
      }
      edges {
        node {
          ... on User {
            databaseId
            publicKeys(first: 100) {
              nodes {
                key
              }
            }
          }
        }
      }
    }
  }
```

Once I had all the keys as a nice ~5GB JSON Lines file, I wanted to find a way to deploy this that was simpler than the previous PostgreSQL database. I played with some more complex ideas, but eventually I tried making a two column SQLite database from a `SHA-256(key)[:16]` PRIMARY KEY to the user ID and it was less than 400MB, small enough to just embed in a Docker image deployed on Fly.io. (Hell, some base images are that large.)

That's how this runs now. No moving parts, and no need for me to sysadmin anything. Bliss. You can see all the source at https://github.com/FiloSottile/whoami.filippo.io, although I have not published the dataset. If you have a compelling analysis you want to run, feel free to reach out. I am not too concerned about sharing it because it's all easily fetched public data anyway.

Have fun, and consider following me at @filippo@abyssdomain.expert.

1. Figuring out what formal properties you need from your signatures for privacy here is kinda interesting. Schnorr signatures like Ed25519 are probably ok, but the formal formulation is left as an exercise to the reader. ↩

2. Even more technically, the client sends the same message both to ask if a key is good and to authenticate with it. In the former case without a signature, in the latter with a signature. This always scared the hell out of me, because there must be an implementation out there that can be tricked to take the message-with-signature code path even if the signature is not there, which would let an attacker authenticate by just knowing the right *public* key. ↩

3. Sort of, it's 5000 points per hour, but our requests cost one point each. ↩