A Censorship-Resistant Web

December 21, 2010

Original link

Imagine someone put a document up at `http://pentagonpapers.com/volumes/1.html` that a) some people want to read and b) some people want to keep you from reading.

Step one: How it works now

On the current Web, the way you request such a document is like this:

1. You ask one of your pre-programmed root servers who is in charge of `.com`
2. They respond with VeriSign, so you ask VeriSign who is in charge of `pentagonpapers.com`
3. They respond with Acme ISP, so you ask ACME ISP where to find `pentagonpapers.com`
4. It responds with an IP address, so you request the page from that IP

The censors can ask VeriSign to give them control of pentagonpapers.com, they can try to shut down Acme ISP, they can try to prevent you from getting hosting, and they can try to shut down your IP. All of these have been used recently, with some success. You need a backup plan.

Let's imagine we want this URL to resolve in an uncensorable way. How would we do it?

Step one: Domain name ownership

First we would have a *certificate authority* (CA) which would sign statements of the form: "As of [DATE], [DOMAIN NAME] is owned by the holder of [PUBLIC KEY]." (Let's call this a *certificate.*) Conveniently, there's already a whole industry of trusted businesses that make these statements — they're called SSL certificates.

The problem is that CAs are presumably just as subject to attack as the registrars (in fact, in some cases they are the registrars!). One possibility is to set up a certificate authority that will not sign such statements for people attempting to engage in censorship. It seems probable that such a policy would be protected by the First Amendment in the US. However, "people engaging in censorship" is a somewhat subjective notion. Also, it's always possible a court could order the certificate authority to turn over the private signing key (or the key could be obtained in some other way).

Another possibility is some kind of "rollback UI". If you know vaguely when the censorship attempts started, you can only trust certifications made before that date. This is a somewhat difficult feature to implement in a way that makes sense to users, though. The best case scenario is one in which the user can clearly distinguish between a censored and uncensored page. In that case, if the page appears censored they can hit a "go back a month" button and the system will only trust certifications made more than a month prior to the certification it's currently using. The user can hit this button repeatedly until they get an uncensored version of the page.

Step two: Web page authentication

Next the owner of the website will need to sign statements of the form "The content of [URL] had the hash [HASH] on [DATE]." (Let's call this an *authenticator*.) Now given a page, a corresponding valid authenticator, and a corresponding valid certificate (call this trio an *authentic page*), browsers can safely display a page even if it can't access the actual web server. The digital signatures work together to prove that the page is what the website owner wanted to publish. If a browser gets back multiple authentic pages, it can display the latest one (modulo the effects of the "go back a month" button).

Step three: Getting authentic pages

Set up a series of domain-to-certificate servers. These servers take a domain names (e.g. `pentagonpapers.com`) and returns back any certificates for it. Certificates can be obtained by crawling the Web or by being submitted by website owners or by being submitted by the CAs themselves.

Set up a series of URL-to-hash servers. These servers take a URL and return back any valid authenticators for that URL. Authenticators are very small, so each URL-to-hash server can probably store all of them. If spam becomes a problem, a little bit of hashcash could be required for storage. Website owners submit their authenticators to the URL-to-hash servers.

Set up a series of hash-to-URL servers. These servers take a hash and return a series of URLs which can be dereferenced in the normal way to obtain a file with that hash. People can submit hash-to-URL mappings to these servers and they can attempt to automatically verify them by downloading the file and seeing if the hash matches.[1][2] Again, these mappings are very small so each server can probably store all of them.[3]

Then there are a series of servers that host controversial files. Perhaps they saved a copy before the site was censored, perhaps they received it thru some out-of-band channel[4]. However they got it, they put them up on their website and then submit the URL to the hash-to-URL servers. Meanwhile, the site publisher submits an authenticator to the URL-to-hash servers.

Now, if a browser cannot obtain the `pentagonpapers` page through normal means it can:

1. Ask each domain-to-cetificate server it knows for certificates for `pentagonpapers.com`
2. Ask each URL-to-hash server it knows for authenticators for the URL
3. Ask each hash-to-URL server it knows for alternative URLs
4. Download from the alternative URLs[5]

This can be implemented through a browser plugin that you click when a page appears to be unavailable. If it takes off, maybe it can be built in to browsers. (While I've been assuming the worst-case-scenario of censorship here, the system would be equally useful for sites that are just down because their servers couldn't handle the load or some other innocent failure.)

This system should work unless our adversary can censor every well-known CA, every well-known URL-to-hash server, every well-known hash-to-URL server, or every alternative URL.

Step four: Beyond the Web

We can help ensure this by operating at least one of each as a Tor hidden service. Because the operator of the service is anonymous, they are immune to legal threats.[6] If the user doesn't have Tor, they can access them through tor2web.org.

Similarly, if you know your document is going to get censored, you can skip steps 1 and 2. Instead of distributing a `pentagonpapers.com` URL which is going to go down, you can just distribute the hash. For users whose browsers don't support this system, you can embed the hash in a URL like:

```
https://hash2url.org/sha1/284219ea93827cdd26f5a697112a029b515dc9a4
```

where `hash2url.org` is a hash-to-URL server that redirects you to a valid URL.

And, of course, if you somehow have access to a working P2P system, you can just it to obtain authentic pages.

Conclusions

What's nice about this system is that it gets you censorship resistance without introducing anything wildly new. There are already certificate authorities. There are already hash-to-URL servers. There are already mirrors. There's already Tor. (There's already tor2web.) The only really new thing specific to censorship resistance is URL-to-hash servers of the form I described, but they're very simple and hopefully uncontroversial.

There is some work to be done stitching all of these together and improving the UI, but unlike with some other censorship-resistance systems, there's nothing you can point to as having no good purpose except for helping bad guys. It's all pretty basic and generally useful stuff, just put together in a new way.

If you're interested in helping build something like this, please send me an email: me@aaronsw.com.

---

1. Any server will have finite bandwidth, so an attacker could try to fool the hash-to-URL server by submitting a URL which when dereferenced never stops sending the data. The hash-to-URL servers should stop after a certain limit and mark the URL as unverified due to max file size. If the server ever obtains a file whose size is under the limit with that hash, it can toss all such URLs.

2. URLs can go out of date so perhaps upon receiving sufficient complaints about a URL being "bad", the server should attempt to reverify. Again, hashcash can be used throughout to avoid spam.

3. A possible protocol for the above two servers is provided in RFC 2169.

4. I have ideas on how to automate this, naturally, but this essay is already far too long.

5. *Optional bonus:* Use HTTP Range headers to download 1/n of the file from each of the n URLs. There are some circumstances where this could speed things up. Or maybe it's just annoying.

6. This moves the censorship weak link to the distribution of introduction points to hidden services.[7] But instead of being published by a DHT, introduction points can be distributed through a flood protocol[8]. Or maybe the DHT can be modified so that there's no obvious censorship point?

7. The introduction points themselves can't be censored because they don't know who they're talking to. (I think they do in the current implementation of Tor, but this doesn't seem necessary. The hidden service can generate a new keypair for each introduction point and send the public key to the introduction point and to Alice.)

8. Is this too chatty? Probably. But remember, it's a last-case resort in some kind of insane police-state world where every country prevents people from running servers that give out the IP addresses of other servers that let you talk to a third server which will give you illegal content.