Security

The computer kind, mostly.

- Why you probably shouldn't use a wildcard certificate

Why you probably shouldn't use a wildcard certificate

> This article was originally published at https://gist.github.com/joepie91/7e5cad8c0726fd6a5e90360a754fc568.

Recently, Let's Encrypt launched free wildcard certificates. While this is good news in and of itself, as it removes one of the last remaining reasons for expensive commercial certificates, I've unfortunately seen a lot of people dangerously misunderstand what wildcard certificates are for.

Therefore, in this brief post I'll explain why **you probably shouldn't use a wildcard certificate, as it will put your security at risk.**

A brief explainer

It's generally pretty poorly understood (and documented!) how TLS ("SSL") works, so let's go through a brief explanation of the parts that are important here.

The general (simplified) idea behind how real-world TLS deployments work, is that you:

1. Generate a cryptographic keypair (private + public key)
2. Generate a 'certificate' from that (containing the public key + some metadata, such as your hostname and when the certificate will expire)
3. Send the certificate to a Certificate Authority (like Let's Encrypt), who will then validate the metadata - this is where it's ensured that you actually *own* the hostname you've created a certificate for, as the CA will check this.
4. Receive a *signed* certificate - the original certificate, plus a cryptographic signature proving that a given CA validated it
5. Serve up this signed certificate to your users' clients

The client will then do the following:

1. Verify that the certificate was signed by a Certificate Authority that it trusts; the keys of all trusted CAs already exist on your system.
2. If it's valid, treat the public key included with the certificate as the legitimate server's public key, and use that key to encrypt the communication with the server

This description is somewhat simplified, and I don't want to go into too much detail as to *why* this is secure from many attacks, but the general idea is this: nobody can snoop on your traffic or impersonate your server, so long as 1) no Certificate Authorities have their own keys compromised, and 2) your keypair + signed certificate have not been leaked.

So, what's a wildcard certificate *really?*

A typical TLS certificate will have an explicit hostname in its metadata; for example, Google might have a certificate for `mail.google.com`. That certificate is **only** valid on `https://mail.google.com/` - not on `https://google.com/`, not on `https://images.google.com/`, and not on `https://my.mail.google.com/` either. In other words, the hostname has to be an *exact match*. If you tried to use that certificate on `https://my.mail.google.com/`. you'd get a certificate error from your browser.

A wildcard certificate is different; as the name suggests, it uses a *wildcard* match rather than an exact match. You might have a certificate for `*.google.com`, and it would be valid on `https://mail.google.com/` *and* `https://images.google.com/` - but **still not** on `https://google.com/` or `https://my.mail.google.com/`. In other words, the asterisk can match any one single 'segment' of a hostname, but nothing with a full stop in it.

There are some situations where this is very useful. Say that I run a website builder from a single server, and every user gets their own subdomain - for example, my website might be at `https://joepie91.somesitebuilder.com/`, whereas your website might be at `https://catdogcat.somesitebuilder.com/`.

It would be very impractical to have to request a new certificate for *every single user that signs up*; so, the easier option is to just request one for `*.somesitebuilder.com`, and now that single certificate works for *all* users' subdomains.

So far, so good.

So, why can't I do this for *everything* with subdomains?

And this is where we run into trouble. Note how in the above example, all of the sites are hosted *on a single server*. If you run a larger website or organization with lots of subdomains that host different things - say, for example, Google with their `images.google.com` and `mail.google.com` - then these subdomains will probably be hosted on *multiple servers*.

And that's where the security of wildcard certificates breaks down.

Remember how one of the two requirements for TLS security is "your keypair + signed certificate have not been leaked". Sometimes certificates *do* leak - servers sometimes get hacked, for

example.

When this happens, you'd want to *limit the damage* of the compromise - ideally, your certificate will expire pretty rapidly, and it doesn't affect anything other than the server that was compromised anyway. After fixing the issue, you then revoke the old compromised certificate, replace it with a new, non-compromised one, and all your other servers are unaffected.

In our single-server website builder example, this is not a problem. We have a single server, it got compromised, the stolen certificate only works for that one single server; we've limited the damage as much as possible.

But, consider the "multiple servers" scenario - maybe just the `images.google.com` server got hacked, and `mail.google.com` was unaffected. However, the certificate on `images.google.com` was a wildcard certificate for `*.google.com`, and now the thief can use it to impersonate the `mail.google.com` server and intercept people's e-mail traffic, even though the `mail.google.com` server was never hacked!

Even though originally only one server was compromised, we didn't correctly limit the damage, and now the e-mail server is at risk too. If we'd had two certificates, instead - one for `mail.google.com` and one for `images.google.com`, each of the servers only having access to their own certificate - then this would not have happened.

The moral of the story

Each certificate should only be used for one server, or one homogeneous cluster of servers. Different services on different servers should have their own, usually non-wildcard certificates.

If you have a lot of hostnames pointing at the same service on the same server(s), then it's fine to use a wildcard certificate - so long as that wildcard certificate doesn't *also* cover hostnames pointing at *other* servers; otherwise, each service should have its own certificates.

If you have a few hostnames pointing at unique servers and everything else at one single service - eg. `login.mysite.com` and then a bunch of user-created sites - then you may want to put the wildcard-covered hostnames under their own prefix. For example, you might have one certificate for `login.mysite.com`, and one (wildcard) certificate for `*.users.mysite.com`.

In practice, you will *almost never* need wildcard certificates. It's nice that the option exists, but unless you're automatically generating subdomains for users, a wildcard certificate is probably an unnecessary and insecure option.

(To be clear: this is in no way specific to Let's Encrypt, it applies to wildcard certificates *in general*. But now that they're suddenly not expensive anymore, I think this problem requires a bit more attention.)