Twitter Thread by Ryan Castellucci ■



Ryan Castellucci ■
@ryancdotorg



A thread on the Tasmota TLS MitM attack I found a few months ago since getting a proper blog post about it is taking me forever.

Tasmota is an open source firmware that runs on a number of ESP8266-based IoT home automation devices, which talks to an MQTT broker for management.

MQTT can be run over TLS to provide confidentiality and integrity, but given the constraints of running on an ESP8266 device, standard TLS certificate validation is rather heavy. As an alternative, Tasmota implements fingerprint based validation, like SSH

The fingerprint validation can do "trust on first use" (TOFU) and just remember the server's public key. This can work well if you're hosting your own MQTT server, and you can just use a self signed certificate. The fingerprint algorithm was even based on how SSH does it.

If you read RFC4253, it describes the RSA key format with the following encoding

string "ssh-rsa"
mpint e
mpint n

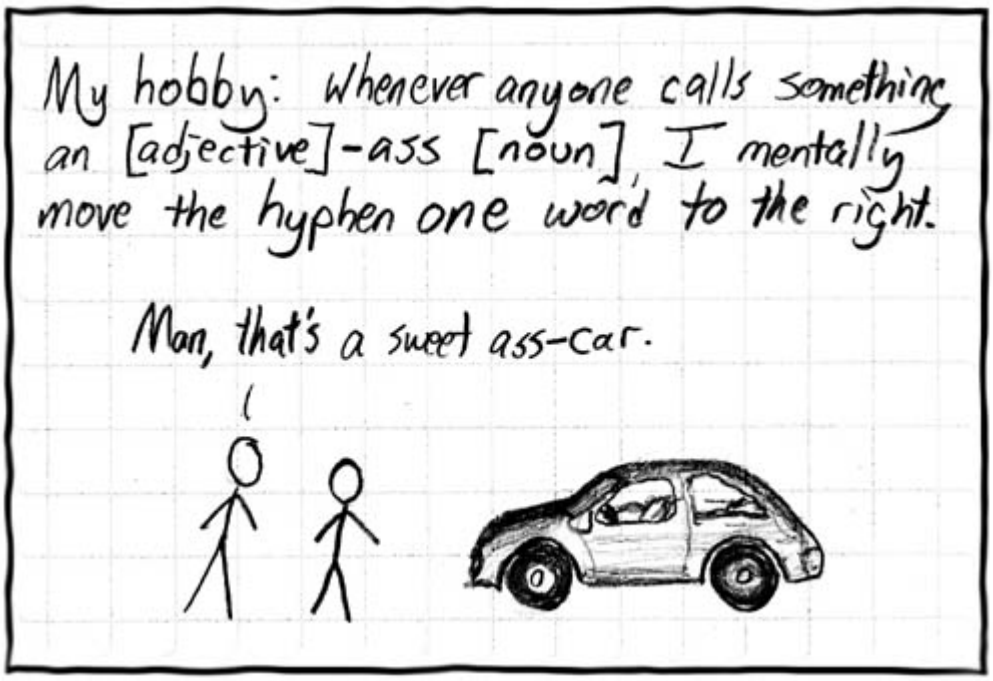
where e is the public exponent (usually 65537) and n is the modulus.

Slight problem. RFC4253 doesn't explain what an mpint is.

You have to go dig up RFC4251 for what an mpint is. The precise details don't really matter for this bug, but the critical detail is that an mpint is length-prefixed, which makes it unambiguous where when mpint ends and the next begins.

Tasmota's implementation missed this.

Instead, everything was simply concatenated together with no delimiter. That means each fingerprint actually matches a "family" of RSA keys with identical serialization with the public exponent ending and the modulus beginning in different places.



As an example (with tiny number), we could have e=17 and n=389436408973, with a serialization of "ssh-rsa17389436408973". But e=17389 and n=436408973 gives the same serialization and is much easier to factor - it's 7 * 11 * 13 * 435973.

A quick bit about RSA - keys are normally formed from two large prime numbers that are around the same size, which are multiplied together to form the modulus. The security of RSA depends on it being difficult to decompose the modulus into its prime factors.

The math behind RSA also works perfectly fine with more than two prime numbers - and this is even supported (though rarely used) by modern implementations. The question is, can factors actually be found for collisions of arbitrary RSA keys under Tasmota's scheme?

There are two ways to factor numbers large enough to use as RSA keys - general number field sieve and elliptic curve method.

GNFS's difficulty depends on the size of the number, and these are too big.

ECM, however, depends on the size of the second largest factor.

For a normal RSA number with two primes, GNFS would be faster, but if the primes are small enough, ECM is feasible.

So, then, to attack the key, the family of RSA keys that match the fingerprint are all generated, and then ECM is used to try to factor them.

The ECM attempt can be time bounded. I found a few minutes per key worked reasonably well, and gave about a 99% success rate in finding one usable factorization.

As a "nothing up my sleeve" demo, I picked this key here that covers nsa[.]gov (among other domains):

https://t.co/1AITBNXWIv

I was able to find a colliding key with a factorization of 13, 1,091, 15,032,926,429, and a 340 digit prime that won't fit in this tweet.

The attack also worked just fine on my LAN and I was able to pull off a MitM attack.

I supplied a patch to Tasmota to remediate this. My patch adds length prefixes to the RSA key serialization to make it unambigious, and will automatically update the fingerprint so long as the server's key isn't "suspicious".

https://t.co/quyAT5uzw4

I believe my code made it into Tasmota 8.4.0, so if you're running an up to date version you're protected - though I imagine most folks aren't bothering with TLS on their home networks.

I am the sort of person who thinks that running transport mode IPSec at home is a good time.

This was one of the coolest crypto exploits I've ever written.

I published a blog post a couple months ago about constructing multi-prime RSA keys in python, which was part of my attack for this.

I did need to patch OpenSSL to remove the limits on the maximum number of primes it allowed, but that was pretty simple.

I think that wraps this thread.