freedom-to-tinker.com

New research: There's no need to panic over factorable keys–just mind your Ps and Qs

Nadia Heninger

12-15 minutes

You may have seen the preprint posted today by Lenstra et al. about entropy problems in public keys. Zakir Durumeric, Eric Wustrow, Alex Halderman, and I have been waiting to talk about some similar results. We will be publishing a full paper after the relevant manufacturers have been notified. Meanwhile, we'd like to give a more complete explanation of what's really going on.

We have been able to remotely compromise about 0.4% of all the public keys used for SSL web site security. The keys we were able to compromise were generated incorrectly–using predictable "random" numbers that were sometimes repeated. There were two kinds of problems: keys that were generated with predictable randomness, and a subset of these, where the lack of randomness allows a remote attacker to efficiently factor the public key and obtain the private key. With the private key, an attacker can impersonate a web site or possibly

decrypt encrypted traffic to that web site. We've developed a tool that can factor these keys and give us the private keys to all the hosts vulnerable to this attack on the Internet in only a few hours.

However, there's no need to panic as this problem mainly affects various kinds of embedded devices such as routers and VPN devices, not full-blown web servers. (It's certainly not, as suggested in the New York Times, any reason to have diminished confidence in the security of web-based commerce.) Unfortunately, we've found vulnerable devices from nearly every major manufacturer and we suspect that more than 200,000 devices, representing 4.1% of the SSL keys in our dataset, were generated with poor entropy. Any weak keys found to be generated by a device suggests that the entire class of devices may be vulnerable upon further analysis.

We're not going to announce every device we think is vulnerable until we've contacted their manufacturers, but the attack is fairly easy to reproduce from material already known. That's why we are working on putting up a web site that you can use to determine whether your device is immediately vulnerable.

Read on for more details, and watch for our full paper soon.

Don't worry, the key for your bank's web site is probably safe

SSL is used to authenticate every major web site on the Internet, but in our analysis, these were not the keys that were vulnerable to the problems outlined in this blog post.

So which systems are vulnerable? Almost all of the vulnerable keys were generated by and are used to secure embedded hardware devices such as routers and firewalls, not to secure popular web sites such as your bank or email provider. Only one of the factorable SSL keys was signed by a trusted certificate authority and it has already expired. There are signed certificates using repeated keys; some of them are generated by vulnerable devices, some of them are due to website owners submitting known weak keys to be signed, and for some of them we have no good explanation.

Embedded devices are well known to have entropy problems. However, until now it wasn't apparent how widespread these problems were in real, Internet-connected devices.

Background: key generation

Websites and networked computers use public-key cryptography for authentication. The kind of authentication that we will be talking about here is a server certifying to a client that it really is the server that the client intended to connect to. An attacker who knows the private key to one of these systems would be able to impersonate the real system to a client or in many cases decrypt encrypted traffic between the client and server.

The most widely used cryptosystem for this purpose is RSA.

The RSA cryptosystem is intended to be based on the difficulty of factoring large numbers. An RSA public key consists of a pair of integers: an encryption exponent *e* and a modulus *N*, which is a large integer that itself is the product of two large primes, *p* and *q*. If an adversary can factor this integer *N* back into its prime factors *p* and *q*, then the adversary can decrypt any messages encrypted using this public key. However, even using the fastest known factoring algorithm, to public knowledge nobody has yet been able to factor a 1024-bit RSA modulus.

It is vitally important to the security of the keys that they are generated using random inputs. If the inputs used to generate the keys were not random, then an adversary may be able to guess those inputs and thus recover the keys without having to laboriously factor N.

On modern computers and servers, key generation software attempts to collect random information from physical sources (often through the underlying operating system): the movements of the mouse, keyboard, hard drive, network events, and other external sources of unpredictable information. However, if the keys are generated from a small set of possibilities, that is, using too little entropy, then the keys may be vulnerable to an attacker. Gathering strong entropy and verifying its strength is a very difficult problem that has given rise to multiple vulnerabilities over the years.

Two versions of the problem

We decided to investigate the prevalence of this issue by scanning the Internet for all SSL and SSH public keys. We scanned every IPv4 address on the Internet, collecting a copy of each SSL certificate and SSH host key. We were able to complete both scans in less than a day: we first used a standard tool called nmap to find hosts with the relevant ports open, and then used our own optimized software to query those hosts. In our SSL scan, we collected 5.8 million certificates. In our SSH scan, we collected 10 million host keys.

We found that entropy problems resulted in two different types of weaknesses:

Repeated public keys. We found that 1% of the RSA keys in our SSL scan data were repeated, apparently due to entropy problems. When two different devices have the same public key, it means they also have the same private key. In effect, the devices that share keys are “in the same boat” as one another–an attacker would only need to compromise the weakest one of these devices, in order to obtain the repeated private key that protects all of the devices. This has long been a known problem, but until now, none of the publicly available security literature has documented how widespread the problem was.

We manually verified that 59,000 duplicate keys were repeated due to entropy problems, representing 1% of all certificates, or 2.6% of self-signed certificates. We also found that 585,000

certificates, or 4.6% of all devices used the default certificates pre-installed on embedded devices. While these devices are not using keys generated with poor entropy, they are suspectible to the same attack as their private keys are found on every device of a given model. We manually verified these keys because a large number of websites may utilize repeated keys for legitimate reason; these provide no risk to users.

Factorable public keys. More surprisingly, we discovered that entropy problems can allow a remote attacker with no special access to factor a significant fraction of the RSA keys in use on the Internet. We were able to factor 0.4% of the RSA keys in our SSL scan. We did this by computing the greatest common divisor (GCD) of all pairs of moduli from RSA public keys on the Internet.

We identified 1724 common factors which allowed us to factor 24,816 SSL keys, and 301 common factors which allowed us to factor 2422 SSH host keys. This means we were able to calculate the private keys for almost half of 1% of the RSA keys in use for SSL. We will explain how we did this calculation below.

Specific vulnerable devices

Embedded devices often generate cryptographic keys on first boot, when their entire state may have been pre-determined in the factory. This can result in the kinds of entropy problems we observe in this study.

We were able to use information from the SSL certificates to

identify classes of devices that are prone to generating weak keys. Many more devices than the ones whose keys we factored are probably also producing weak keys that could be compromised by a determined attacker. The list of vulnerable devices that we have already identified includes more than thirty different manufacturers, including almost all of the biggest names in the computer hardware industry. The kinds of products that we identified include firewalls, routers, VPN devices, remote server administration devices, printers, projectors, and VOIP phones.

We're not going to list specific devices or brands until we've told the manufacturers, but here are some examples:

Firewall product X:
52,055 hosts in our SSL dataset
6,730 share public keys
12,880 have factorable keys

Consumer-grade router Y:
26,952 hosts in our SSL dataset
9,345 share public keys
4,792 have factorable keys

Enterprise remote access solution Z:
1,648 hosts in our SSL dataset
24 share public keys
0 factorable

How could this happen?

It wasn't obvious at first how these types of entropy problems might result in keys that could be factored. We'll explain now for the geekier readers.

Here's one way a programmer might generate an RSA modulus:

```
prng.seed(seed)

p = prng.generate_random_prime()

q = prng.generate_random_prime()

N = p*q
```

If the pseudorandom number generator is seeded with a predictable value, then that would likely result in different devices generating the same modulus *N*, but we would not expect a good pseudorandom number generator to produce different moduli that share a single factor.

However, some implementations add additional randomness between generating the primes *p* and *q*, with the intention of increasing security:

```
prng.seed(seed)

p = prng.generate_random_prime()
```

```
prng.add_randomness(bits)

q = prng.generate_random_prime()

N = p*q
```

If the initial seed to the pseudorandom number generator is generated with low entropy, this could result in multiple devices generating different moduli which share the prime factor p and have different second factors q. Then both moduli can be easily factored by computing their GCD: $p = \gcd(N_1, N_2)$.

OpenSSL's RSA key generation functions this way: each time random bits are produced from the entropy pool to generate the primes p and q, the current time in seconds is added to the entropy pool. Many, but not all, of the vulnerable keys were generated by OpenSSL and OpenSSH, which calls OpenSSL's RSA key generation code.

Computing the GCDs of all pairs of keys

If any pair of RSA moduli N_1 and N_2 share, say, the same prime factor p in common, but have different second factors q_1 and q_2, then we can easily factor the moduli by computing their greatest common divisor. On my desktop computer, computing the GCD of two 1024-bit RSA moduli took about 17µs.

For the mathematically inclined, I'll explain how we were able to use this idea to factor a large collection of keys.

The simplest way that one might try to factor keys is by computing the GCD of each pair of RSA moduli. A back of the envelope calculation shows that doing a GCD computation for all pairs of moduli in our data sets would take 24 years of computation time on my computer.

Instead, we used an idea Dan Bernstein published in the Journal of Algorithms in 2005 for factoring a group of integers into coprimes which allowed us to do the computation in a few hours on a desktop computer, in a few lines of Python. The algorithm is no great secret: a long stream of published papers has worked on improving these ideas.

The main mathematical insight is that one can compute the GCD of a single modulus N_1 with every other modulus $N_2,\ldots,N_m$ using the following equation:

$$\gcd(N_1,N_2 \ldots N_m) = \gcd(N_1, (N_1 * N_2 * \ldots * N_m \bmod N_1^2)/N_1)$$

The secret sauce is in making this run fast–note that the first step is to compute the product of all the keys, a 729 million digit number. We were able to factor the SSL data in eighteen hours on a desktop computer using a single core, and the SSH data in about four hours using four cores.

The bottom line

This is a problem, but it's not something that average users need to worry about just yet. However, embedded device

manufacturers have a lot of work to do, and some system administrators should be concerned. This is a wake-up call to the security community, and a reminder to all of how security vulnerabilities can sometimes be hiding in plain sight.