badkeys

Hanno Böck

https://badkeys.info/

About me

- Writing for the IT news publication Golem.de
- Author of monthly Bulletproof TLS Newsletter
- Have discovered various TLS vulnerabilities in the past (e.g. ROBOT)

Funding/Support for the badkeys project

- Center for Information Security and Trust (CISAT) at the IT University of Copenhagen
- CIDI project (Cybersecure IOT in Danish Industry) / Industriens Fond

Security

GitHub security update: revoking weakly-generated SSH keys

On September 28, 2021, we received notice from the developer Axosoft regarding a vulnerability in a dependency of their popular git GUI client - GitKraken. An underlying issue with a dependency, called `keypair`, resulted in the GitKraken client generating weak SSH keys.

"There is no haveibeenpwned for public keys as far as I know"

user jornane on lobste.rs, 10/2021

Good News: Now there is!

badkeys.info

A website, tool and library to check cryptographic keys for known vulnerabilities

Public Key Cryptography

We have a public and a private key

Sometimes things go wrong and private keys are no longer private

Key Generation Vulnerabilities

- Debian OpenSSL Bug
- Common prime factors
- Return of Coopersmith's attack / ROCA
- keypair / Gitkraken bug
- Fermat attack
- "Public Private Keys"

Debian OpenSSL bug

In 2008 it was discovered that Debian's OpenSSL package contained a patch that broke the random number generator, effectively making the PID the only source of randomness

Attack: Generate all possible keys

There are a number of possible variations to consider, like the architecture, the key size, OpenSSL vs. OpenSSH

This is 14 years ago, surely no such keys are in use any more!

In March 2020 ssl.com issued a TLS certificate with a key affected by the Debian weak key bug

Matt Palmer on mozilla.dev.security.policy

Common Prime Factors

In 2012 two research teams independently discovered a large number of RSA keys with common prime factors

An RSA public key consists of two values N and e

The N value is the product of two secret primes p and q

$$N = p * q$$

If you know these primes p and q you can calculate the private key

What if...?

There are two RSA keys with

$$N_1 = p_1 * q_1$$

$$N_2 = p_1 * q_2$$

If

$$N_1 = p_1 * q_1$$

$$N_2 = p_1 * q_2$$

then

$$GCD(N_1, N_2) = p_1$$

GCD = Greatest Common Divisor

This works for two keys, but there exists an efficient algorithm to do this for millions of keys

Why would such keys exist?

Early boot time entropy problem

Something like this:

1. Linux boots, random number generator is not yet initialized
2. Prime p is generated with bad random numbers
3. Random number generator is initialized
4. Prime q is generated with good random numbers

Linux implemented countermeasures and with any kernel released after 2012 this shouldn't happen again

In 2016 a paper analyzed the patch status of this vulnerability, it was still very prevalent

Weak Keys Remain Widespread in Network Devices

How to check for that?

We obviously can't check whether a key shares a prime factor with all other keys on the fly, but we can check against a product of known common prime factors

Return of Coppersmith's attack / ROCA

In 2017 researchers discovered that chips from Infineon generated RSA keys that could be broken with a variant of Coopersmith's attack

ROCA affected devices like TPM chips, smart cards, Yubikeys, ...

Estonia had to replace 750,000 ID cards

These cards where certified according to FIPS and Common Criteria

The attack is expensive (140 CPU years for 2048 bit keys), but checking keys for the vulnerability is fast

keypair / Gitkraken bug

In 2021 it was discovered that a javascript library called "keypair" that is used by the Gitkraken software sometimes created duplicate keys

The disclosure - by Gitkraken, Github, Gitlab and others - was very scarce in details

I asked Gitkraken for further information or the tool they used to identify affected keys, they refused to share anything

Due to a bad conversion of bytes to strings the random number generator would be initialized only with values 0-9, while 0 was the most common value

Unfortunately it is not feasible to generate all affected keys, as the key space is still too large, but some keys are more likely to appear than others

Fermat Attack

In 1643 the matematician Pierre de Fermat described a factoring algorithm that is efficient for products of "close" primes

Remember: RSA public keys contain a value N that is the product of the primes p and q

If p and q are "close" (meaning the difference between the two is small) then we can use Fermat's factorization algorithm to break the key

The Safezone library by Rambus, used in printers from Canon and Fujifilm, was vulnerable to this attack

"Public Private Keys"

When your private key can be found on the Internet then it is no longer secure

What are "Public Private Keys"?

- Static Keys in Firmware
- Example keys used in software, standards or documentation
- Keys accidentally uploaded / leaked

Kompromat Repository

https://github.com/SecurityFail/kompromat

badkeys checks against a list of hashes of "public private keys"

Implementation Details

badkeys contains three kinds of checks:

- Algorithmic checks (Fermat, ROCA)
- Hash-Blocklist ("Public Private Keys", Debian bug, keypair bug)
- A mix of both (Common prime factors)

How to best implement a blocklist for RSA keys?

A common strategy is to have a list of hashes of the public key, but it is better to just hash the modulus

Why is that?

If the key (N, e_1) is broken then the key (N, e_2) is also broken

Example Debian OpenSSL bug

The OpenSSL command line genrsa can generate keys with *e=3* and *e=65537*, so if we hash the full key and wand to cover both our blocklist would be twice as large

Most checks in badkeys are about RSA

What about other key types (Elliptic Curves, DSA, ...)?

Supported by blocklist, but currently no further checks

Applying badkeys to collections of keys

Now we have a tool to check for vulnerable keys, we can apply that to large collections of public keys

SSH host keys (~16 million)

- 18,848 Static Keys from Firmware
- 821 Debian OpenSSL bug
- 697 Common prime factors

TLS keys (Rapid7 data, ~28 million, biased due to removal of duplicate certs!)

- 1,422 Static Keys from Firmware
- 1,418 Keys with Common Prime Factors
- 137 Keys vulnerable to Fermat Attack

Certificate Transparency (since Jan 2021, ~2,5 billion!)

- 262 Keys with Common Prime Factors
- 22 Keys vulnerable to Fermat attack
- 20 Keys vulnerable to ROCA (certificates used by Yahoo)

Certificate Transparency contains publicly trusted certificates, vulnerable keys in there are far less likely, but sometimes you still find things

Certificate authorities check keys for known vulnerabilities and must revoke certificates with compromised keys

I also found two certificates using an OpenSSL example key, one of them still valid

Demo time

What is it good for?

Obviously: Pentests, Audits etc.

If you are in a position where users provide public keys to you (e.g. certificate authorities, code hosting plattforms) - it's a good idea to check them for known vulnerabilities

The code is Open Source (Python, MIT license), please use it!

https://github.com/badkeys/badkeys

badkeys.info

Thank you for listening!

Any questions?