Random number generation done wrong

Nadia Heninger

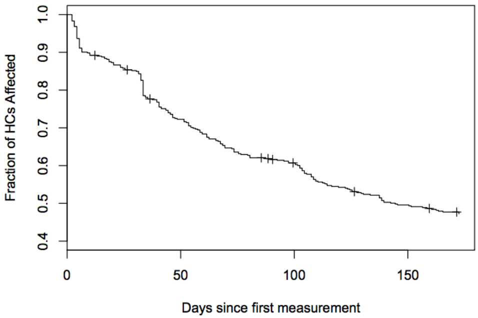
University of Pennsylvania

April 30, 2017

2008: The Debian OpenSSL entropy disaster

August 2008: Discovered by Luciano Bello

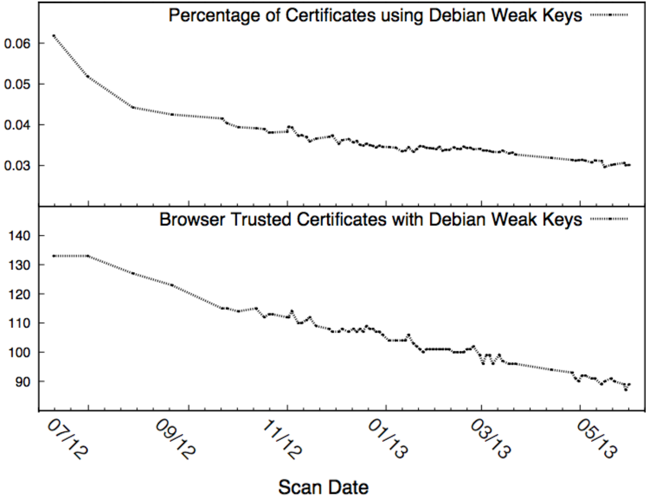
Keys dependent only on pid and machine architecture:
294,912 keys per key size.



[Yilek, Rescorla, Shacham, Enright, Savage 2009]

Debian OpenSSL weak keys in 2013

31,111 (0.34%) of RSA SSH hosts



[Durumeric Wustrow Halderman 2013]

[Heninger Durumeric Wustrow Halderman 2012], [Lenstra, Hughes, Augier, Bos, Kleinjung, Wachter 2012]

Motivating question:

What does cryptography look like on a broad scale?

Methodology:

1. Collect cryptographic data (keys, signatures...)
2. Look for interesting things.

Results:

Stumble upon random number generation flaws in the wild.

Public-key cryptography in practice.

End host cipher preference November 2016
(censys.io and custom Zmap scans)

		Key exchange			Signatures		
	Hosts	**RSA**	**DH**	**ECDH**	**RSA**	**DSA**	**ECDSA**
HTTPS	39M	39%	10%	51%	99%	≈ 0	1%
SSH	17M	≈ 0	52%	48%	93%	7%	0.3%
IKEv1	1.1M	-	97%	3%	-	-	-
IKEv2	1.2M	-	98%	2%	-	-	-

(* Preferences depend on client ordering.)

Cryptography relies on good randomness.

If you use bad randomness, an attacker might be able to guess your private key.

End of story?

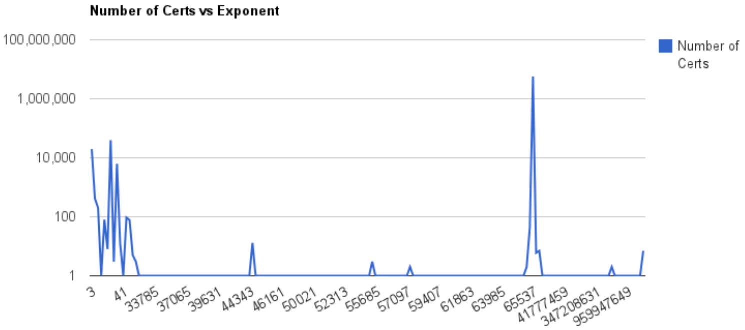
What could go wrong: Repeated keys

RSA Public Keys

$N = pq$ modulus

e encryption exponent

- Two hosts share e: not a problem.



What could go wrong: Repeated keys

RSA Public Keys

$N = pq$ modulus

e encryption exponent

- Two hosts share e: not a problem.
- Two hosts share N: $\rightarrow$ both know private key of the other.

Hosts share the same public and private keys, and can decrypt and sign for each other.

What happens if we look for repeated moduli?

$>60\%$ of HTTPS and SSH hosts served non-unique public keys.

What happens if we look for repeated moduli?

$>$ 60% of HTTPS and SSH hosts served non-unique public keys.

Many valid (and common) reasons to share keys:

- Shared hosting situations. Virtual hosting.
- A single organization registers many domain names with the same key.
- Expired certificates that are renewed with the same key.

What happens if we look for repeated moduli?

$>$ 60% of HTTPS and SSH hosts served non-unique public keys.

Common (and unwise) reasons to share keys:

- Device default certificates/keys.
- Apparent entropy problems in key generation.

What happens if we look for repeated moduli?

$>$ 60% of HTTPS and SSH hosts served non-unique public keys.

Common (and unwise) reasons to share keys:

- Device default certificates/keys.
- Apparent entropy problems in key generation.

HTTPS:
default certificates/keys:
670,000 hosts (5%)

low-entropy repeated keys:
40,000 hosts (0.3%)

SSH:
default or low-entropy keys:
1,000,000 hosts (10%)

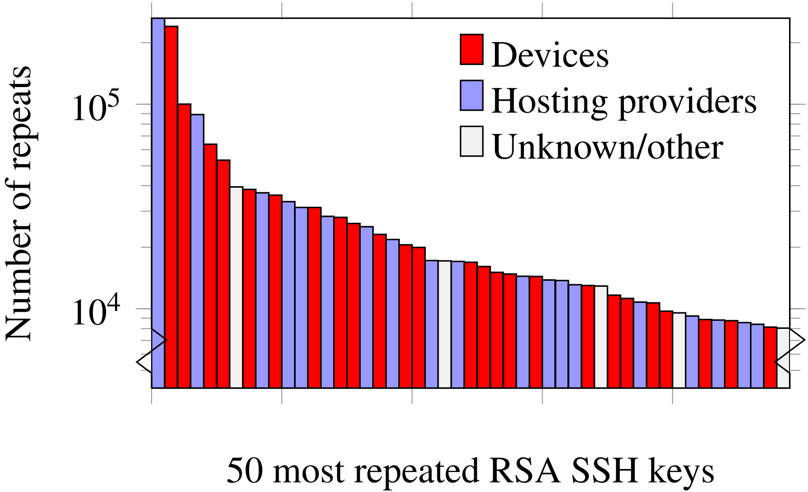
Subjects of most repeated TLS Certificates

```
C=TW, ST=HsinChu, L=HuKou, O=DrayTek Corp., OU=DrayTek Support, CN=Vigor Rou
C=UA, ST=Califonia, L=Irvine, O=Broadcom, OU=Broadband, CN=Daniel/emailAddre
C=US, ST=AL, L=Huntsville, O=ADTRAN, Inc., CN=NetVanta/emailAddress=tech.sup
C=CA, ST=Quebec, L=Gatineau, O=Axentraserver Default Certificate 863B4AB, CN
C=US, ST=California, L=Santa Clara, O=NETGEAR Inc., OU=Netgear Prosafe, CN=N
C=--, ST=SomeState, L=SomeCity, O=SomeOrganization, OU=SomeOrganizationalUni
C=US, ST=Texas, L=Round Rock, O=Dell Inc., OU=Remote Access Group, CN=iDRAC6
C=--, ST=SomeState, L=SomeCity, O=SomeOrganization, OU=SomeOrganizationalUni
C=IN, ST=WA, L=WA, O=lxlabs, OU=web, CN=*.lxlabs.com/emailAddress=sslsign@lx
C=TW, ST=none, L=Taipei, O=NetKlass Techonoloy Inc, OU=NetKlass, CN=localhos
C=--, ST=SomeState, L=SomeCity, O=SomeOrganization, OU=SomeOrganizationalUni
C=US, CN=ORname_Jungo: OpenRG Products Group
C=--, ST=SomeState, L=SomeCity, O=SomeOrganization, OU=SomeOrganizationalUni
C=LT, L=Kaunas, O=Ubiquiti Networks Inc., OU=devint, CN=ubnt/emailAddress=su
C=PL, ST=Some-State, O=Mini Webservice Ltd
C=US, ST=Texas, L=Round Rock, O=Dell Inc., OU=Remote Access Group, CN=DRAC5
C=AU, ST=Some-State, O=Internet Widgits Pty Ltd, CN=TS Series NAS
C=DE, ST=NRW, L=Wuerselen, O=LANCOM Systems, OU=Engineering, CN=www.lancom s
```

x509 Subject Alt Name of Repeated Trusted TLS Certificates

```
DNS:*.opentransfer.com, DNS:opentransfer.com
DNS:*.home.pl, DNS:home.pl
DNS:a248.e.akamai.net, DNS:*.akamaihd.net, DNS:*.akamaihd-staging.net
DNS:*.c11.hesecure.com, DNS:c11.hesecure.com
DNS:*.pair.com, DNS:pair.com
DNS:*.c12.hesecure.com, DNS:c12.hesecure.com
DNS:*.c10.hostexcellence.com, DNS:c10.hostexcellence.com
DNS:*.securesitehosting.net, DNS:securesitehosting.net
DNS:*.sslcert19.com, DNS:sslcert19.com
DNS:*.c11.ixsecure.com, DNS:c11.ixsecure.com
DNS:*.c9.hostexcellence.com, DNS:c9.hostexcellence.com
DNS:*.naviservers.net, DNS:naviservers.net
DNS:*.c10.ixwebhosting.com, DNS:c10.ixwebhosting.com
DNS:*.google.com, DNS:google.com, DNS:*.atggl.com, DNS:*.youtube.com, DNS:yo
DNS:*.hospedagem.terra.com.br
DNS:*.c8.ixwebhosting.com, DNS:c8.ixwebhosting.com
DNS:www.control.tierra.net, DNS:control.tierra.net
```

Classifying repeated SSH host keys



What could go wrong: Shared factors

If two RSA moduli share a common factor,

$$N_1 = pq_1 \qquad N_2 = pq_2$$

What could go wrong: Shared factors

If two RSA moduli share a common factor,

$$N_1 = pq_1 \qquad\qquad N_2 = pq_2$$

$$\gcd(N_1, N_2) = p$$

You can factor both keys with GCD algorithm.

Time to factor
768-bit RSA modulus:
2.5 calendar years
[Kleinjung et al. 2010]

Time to calculate GCD
for 1024-bit RSA moduli:
15μs

Should we expect to find key collisions in the wild?

Experiment: Compute GCD of each pair of M RSA moduli randomly chosen from P primes.

What *should* happen? **Nothing.**

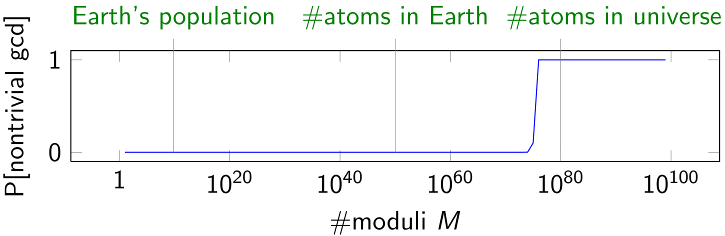
Should we expect to find key collisions in the wild?

Experiment: Compute GCD of each pair of M RSA moduli randomly chosen from P primes.

What *should* happen? **Nothing.**

Prime Number Theorem:
$\sim 10^{150}$ 512-bit primes

Birthday bound:
$\Pr[\text{nontrivial gcd}] \approx 1 - e^{-2M^2/P}$



How to efficiently compute pairwise GCDs

Computing pairwise $\gcd(N_i, N_j)$ the naive way on all of the unique RSA keys in a single set of scans would take

$$15\mu\text{s} \times \binom{14 \times 10^6}{2} \text{pairs} \approx 1100 \text{ years}$$

of computation time.

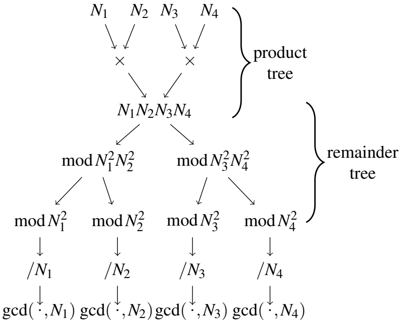
How to efficiently compute pairwise GCDs

Computing pairwise $\gcd(N_i, N_j)$ the naive way on all of the unique RSA keys in a single set of scans would take

$$15\mu\text{s} \times \binom{14 \times 10^6}{2} \text{pairs} \approx 1100 \text{ years}$$

of computation time.

Algorithm from (Bernstein 2004)
A few hours for 10M keys.
Implementation available at
https://factorable.net.



What happens if we compute GCDs of some RSA moduli?

What *did* happen when we GCDed all the keys in 2012?

What happens if we compute GCDs of some RSA moduli?

What *did* happen when we GCDed all the keys in 2012?

Computed private keys for

- 64,081 HTTPS servers (0.50%).
- 2,459 SSH servers (0.03%).
- 2 PGP users (and a few hundred invalid keys).

... only two of the factored https certificates were signed by a CA, and both were expired. The web pages weren't active.

... only two of the factored https certificates were signed by a CA, and both were expired. The web pages weren't active.

Subject information for certificates:

```
CN=self-signed, CN=system generated, CN=0168122008000024
CN=self-signed, CN=system generated, CN=0162092009003221
CN=self-signed, CN=system generated, CN=0162122008001051
C=CN, ST=Guangdong, O=TP-LINK Technologies CO., LTD., OU=TP-LINK SOFT, CN=TL-R478+1145D5C30089/emailAddres
C=CN, ST=Guangdong, O=TP-LINK Technologies CO., LTD., OU=TP-LINK SOFT, CN=TL-R478+139819C30089/emailAddres
CN=self-signed, CN=system generated, CN=0162072011000074
CN=self-signed, CN=system generated, CN=0162122009008149
CN=self-signed, CN=system generated, CN=0162122009000432
CN=self-signed, CN=system generated, CN=0162052010005821
CN=self-signed, CN=system generated, CN=0162072008005267
C=US, O=2Wire, OU=Gateway Device/serialNumber=360617088769, CN=Gateway Authentication
CN=self-signed, CN=system generated, CN=0162082009008123
CN=self-signed, CN=system generated, CN=0162072008005385
CN=self-signed, CN=system generated, CN=0162082008000317
C=CN, ST=Guangdong, O=TP-LINK Technologies CO., LTD., OU=TP-LINK SOFT, CN=TL-R478+3F5878C30089/emailAddres
CN=self-signed, CN=system generated, CN=0162072008005597
CN=self-signed, CN=system generated, CN=0162072010002630
CN=self-signed, CN=system generated, CN=0162032010008958
CN=109.235.129.114
CN=self-signed, CN=system generated, CN=0162072011004982
CN=217.92.30.85
CN=self-signed, CN=system generated, CN=0162112011000190
CN=self-signed, CN=system generated, CN=0162062008001934
CN=self-signed, CN=system generated, CN=0162112011004312
CN=self-signed, CN=system generated, CN=0162072011000946
C=US, ST=Oregon, L=Wilsonville, CN=141.213.19.107, O=Xerox Corporation, OU=Xerox Office Business Group,
CN=XRX0000AAD53FB7.eecs.umich.edu, CN=(141.213.19.107|XRX0000AAD53FB7.eecs.umich.edu)
CN=self-signed, CN=system generated, CN=0162102011001174
CN=self-signed, CN=system generated, CN=0168112011001015
CN=self-signed, CN=system generated, CN=0162012011000446
```

Attributing SSL and SSH vulnerabilities to implementations

Evidence strongly suggested *widespread implementation problems*.

Clue #1: Vast majority of weak keys generated by network devices:

- Juniper network security devices
- Cisco routers
- IBM server management cards
- Intel server management cards
- Innominate industrial-grade firewalls
- ...

Identified devices from > 50 manufacturers

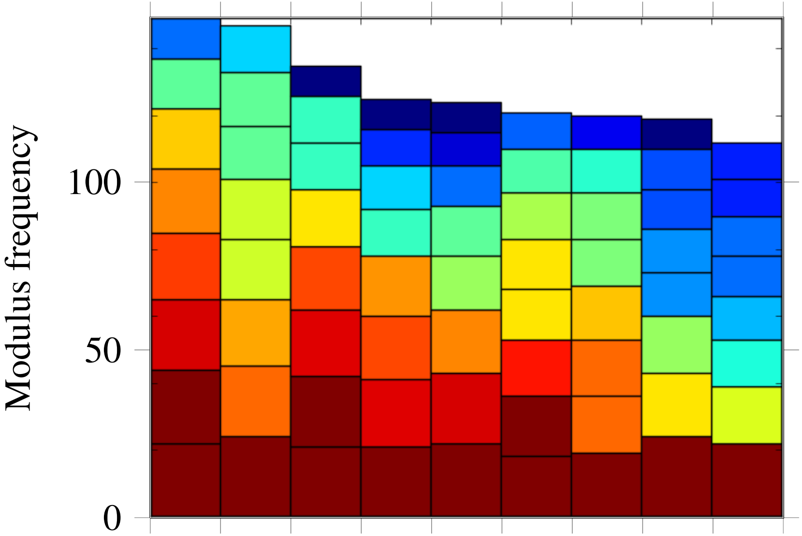
Attributing SSL and SSH vulnerabilities to implementations

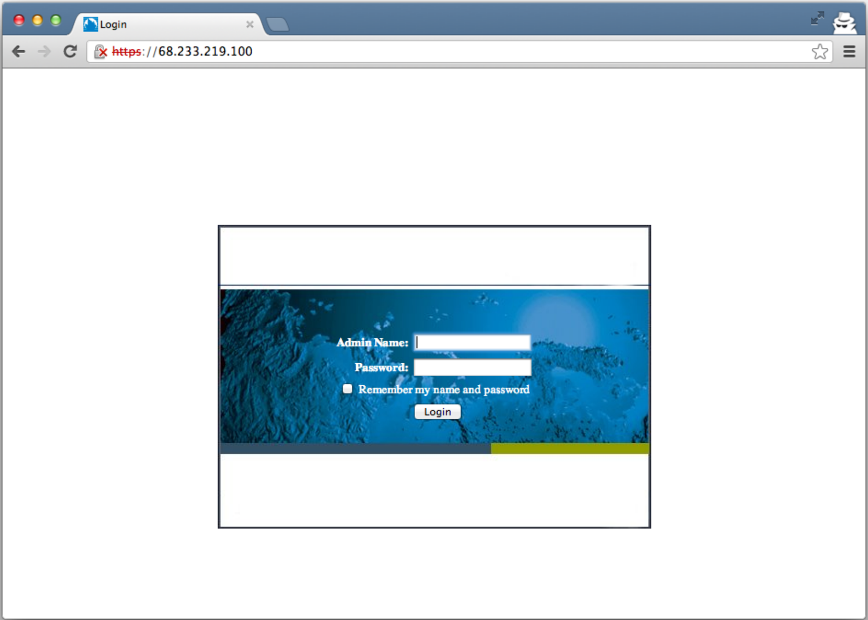
Evidence strongly suggested *widespread implementation problems*.

Clue #2: Very different behavior for different devices. Different companies, implementations, underlying software, distributions of prime factors.

Distribution of prime factors

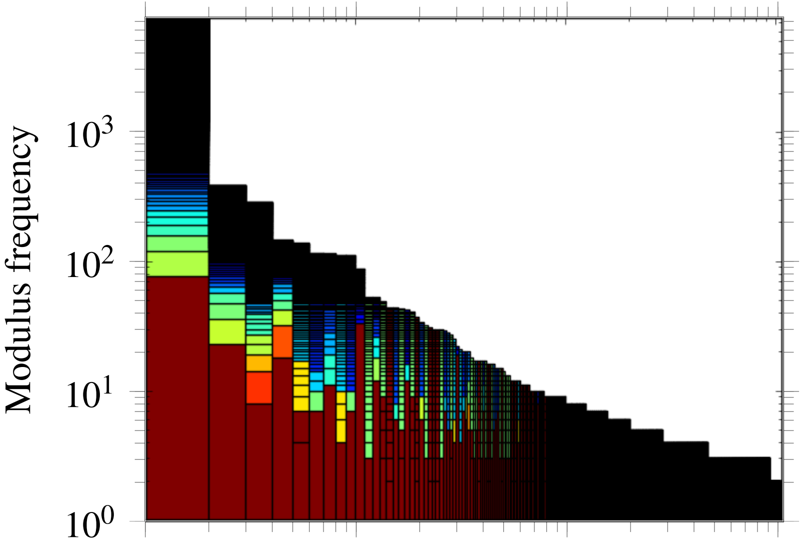
IBM Remote Supervisor Adapter II and Bladecenter Management Module



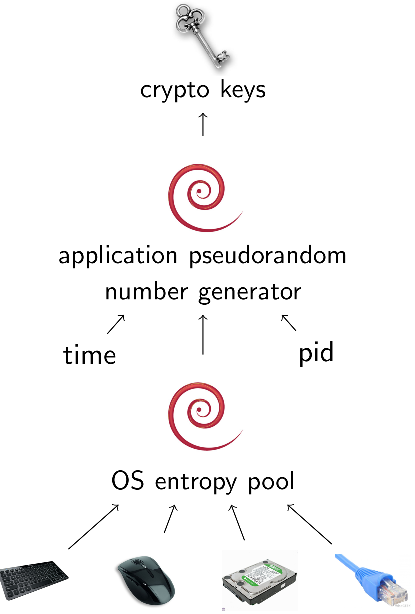

Login
https://68.233.219.100
Admin Name:
Password:
Remember my name and password
Login

Distribution of prime factors

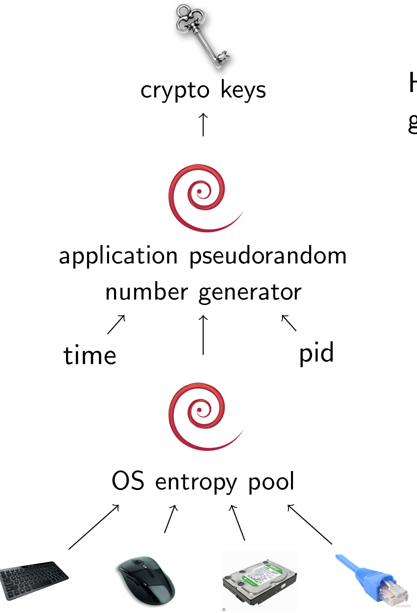
Juniper SRX branch devices



Random number generation in software

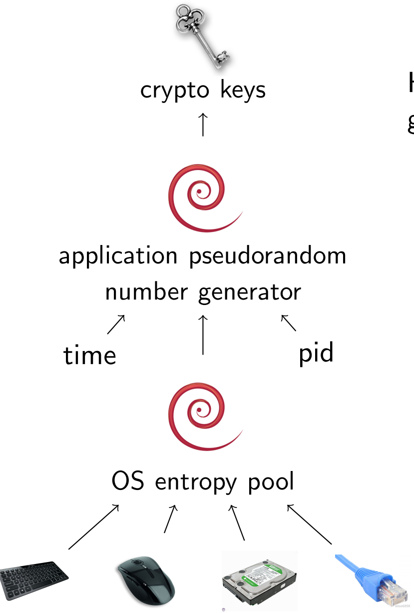


Random number generation in software



Hypothesis: Devices automatically generate crypto keys on first boot.

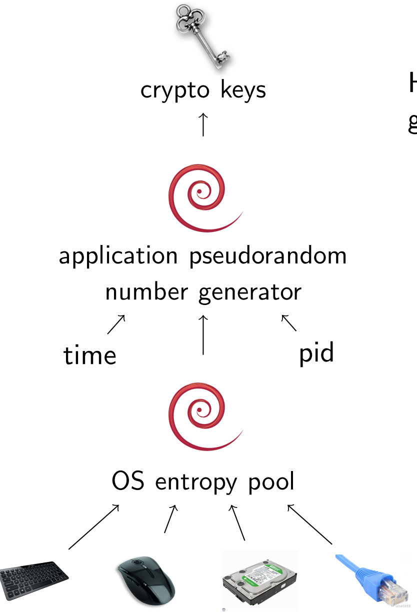
Random number generation in software



Hypothesis: Devices automatically generate crypto keys on first boot.

- Headless or embedded devices may lack these entropy sources.

Random number generation in software



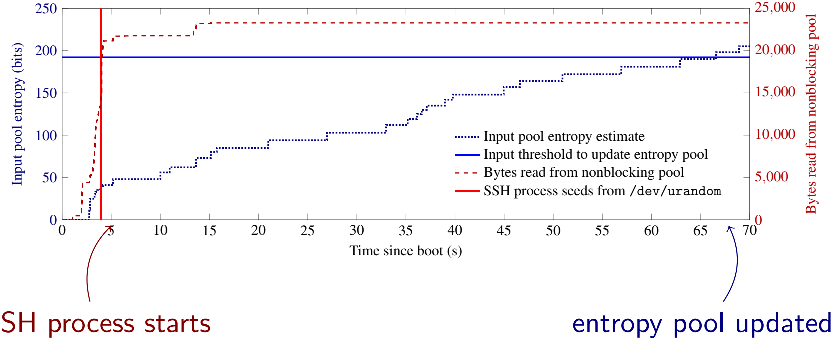
Hypothesis: Devices automatically generate crypto keys on first boot.

- OS random number generator may not have incorporated any entropy when queried by software.
- Headless or embedded devices may lack these entropy sources.

Linux boot-time entropy hole

Experiment: Instrument Linux kernel to track entropy estimates.

Ubuntu Server 10.04



Patched since July 2012.

Generating vulnerable RSA keys in software

- Insufficiently random seeds for pseudorandom number generator $\implies$ we should see repeated keys.

```
prng.seed()
p = prng.random_prime()
q = prng.random_prime()
N = p*q
```

- We do:
  - $>$ 60% of hosts share keys
  - At least 0.3% due to bad randomness.
- Repeated keys may be a sign that implementation is vulnerable to a targeted attack.

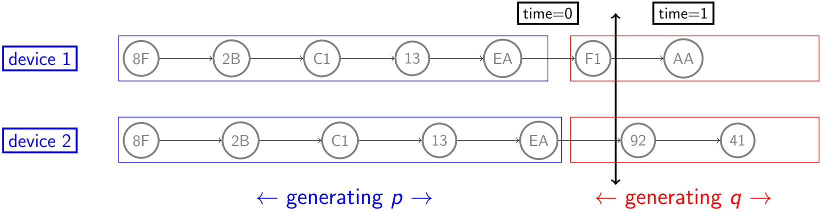
But why do we see factorable keys?

Generating factorable RSA keys in software

```
prng.seed()
p = prng.random_prime()
prng.add_randomness()
q = prng.random_prime()
N = p*q
```

OpenSSL adds time in seconds

Insufficient randomness can lead to factorable keys.



Experimentally verified OpenSSL generates factorable keys in this situation.

GCDing RSA keys is surprisingly fruitful...

2013 Factored 103 Taiwanese citizen smart card keys.
[Bernstein, Chang, Cheng, Chou, Heninger, Lange, van Someren 2013]

2015 Factored 90 export-grade HTTPS keys.
[Albrecht, Papini, Paterson, Villanueva-Polanco 2015]

2017 Factored 3,337 Tor relay RSA keys.
[Kadianakis, Roberts, Roberts, Winter 2017]

Were RNG issues fixed since 2012? A follow-up study.

[Hastings, Fried, Heninger 2016]

- Did vendors fix their broken implementations?
- Can we observe patching behavior in end users?



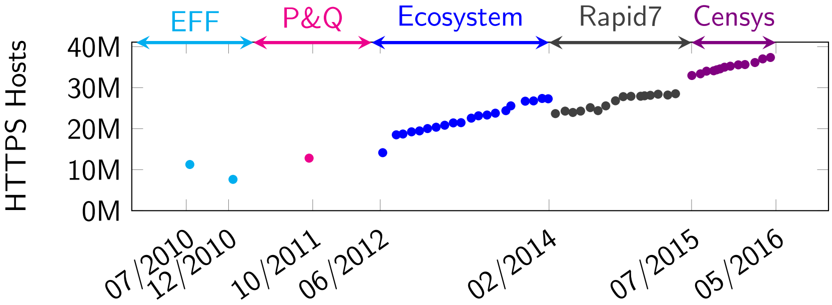
Methodology for this study

What happens when we ask vendors to fix a vulnerability?

1. Aggregated internet-wide TLS scans from 2010-2016
2. Computed batch GCD for 81.2 million RSA moduli
3. Identified vendors of vulnerable implementations
4. Examined results based on response to 2012 notification

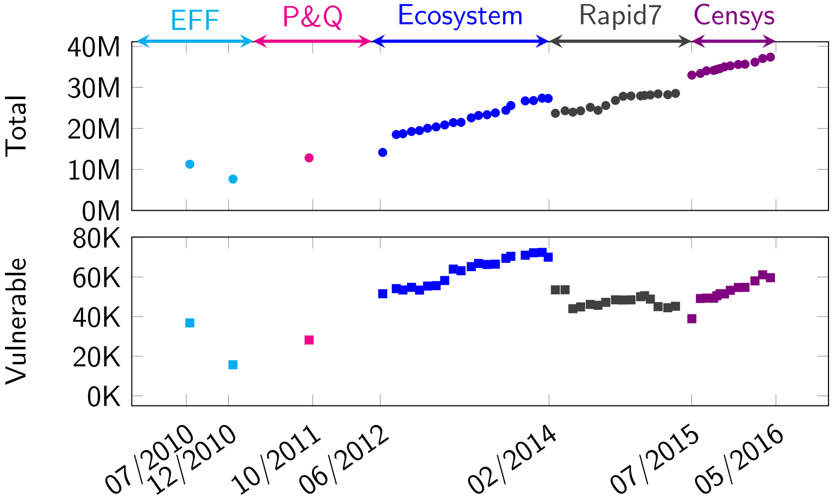
Data sources: how to read the plots

- Scan sources along top of plot
- Scan dates on x-axis
- Absolute counts on y-axis



Six years of factoring keys

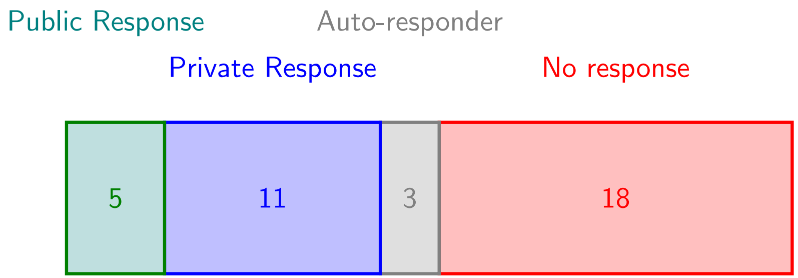
- 51 million distinct HTTPS RSA moduli : 0.43% vulnerable
- 65 million distinct HTTPS certificates : 2.2% vulnerable
- 1.5 billion HTTPS host records : 0.19% vulnerable



Original notification

- Low response rates from vendors
- Took place March-June 2012

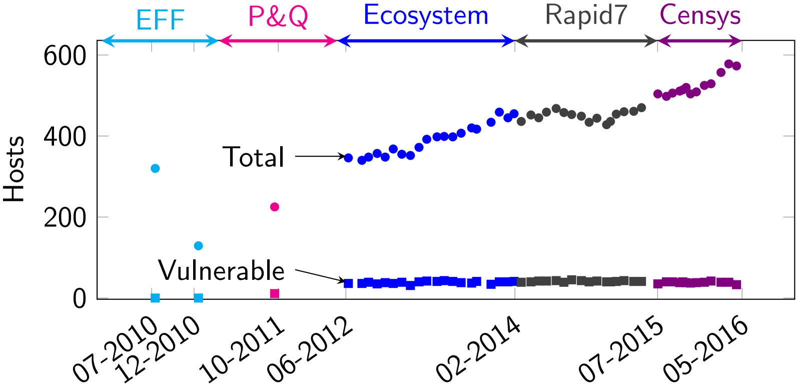
Vendor response to original notification



Innominate

mGuard network security devices (Smart, PCI, Industrial RS, Blade, Delta, EAGLE)

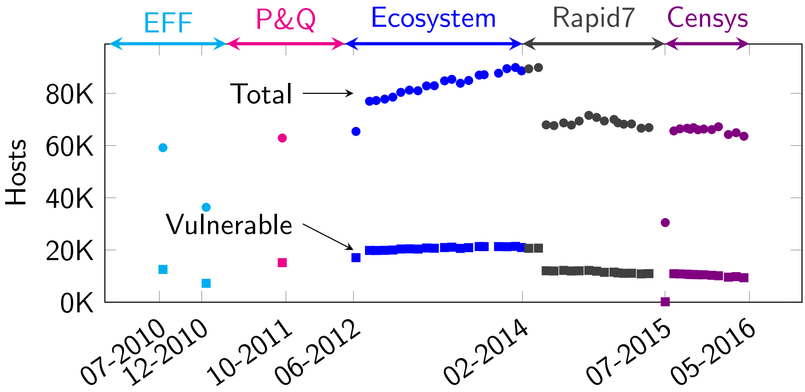
- Public advisory in June 2012
- Consistent population of vulnerable devices since 2012
- New devices not vulnerable, but old devices not patched



Juniper

SRX Series Service Gateways (SRX100, SRX110, SRX210, SRX220, SRX240, SRX550, SRX650), LN1000 Mobile Secure Router

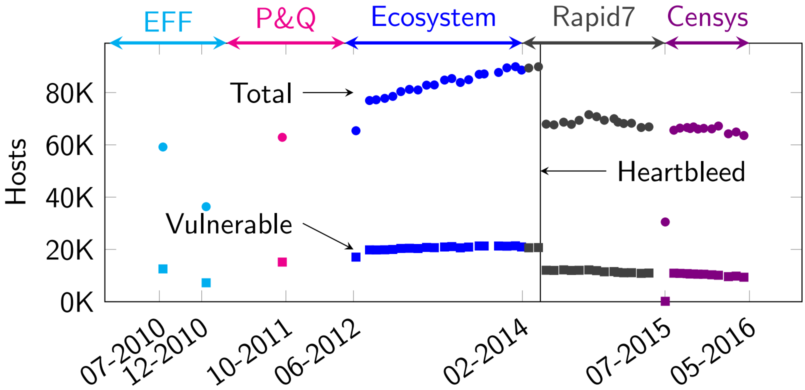
- Public security bulletin in April 2012, out-of-cycle security notice in July 2012
- Majority of factored keys in 2012 were Juniper hosts
- Weird behavior in April 2014



Juniper

SRX Series Service Gateways (SRX100, SRX110, SRX210, SRX220, SRX240, SRX550, SRX650), LN1000 Mobile Secure Router

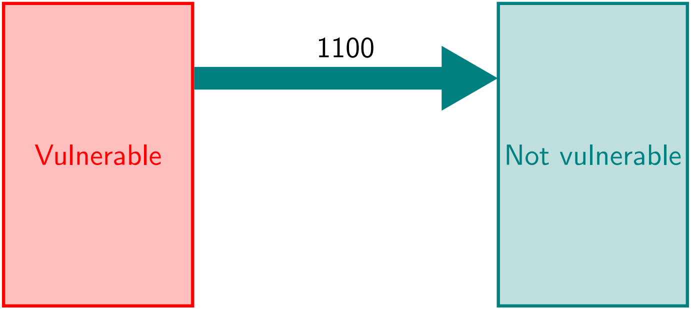
- 30,000 Juniper-fingerprinted hosts (9000 vulnerable) came offline after Heartbleed
- IPs do not reappear in later scans: TLS disabled, scans blocked, devices offline?



Juniper

SRX Series Service Gateways (SRX100, SRX110, SRX210, SRX220, SRX240, SRX550, SRX650), LN1000 Mobile Secure Router

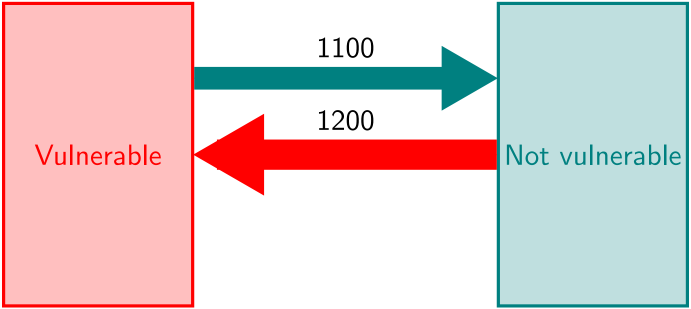
Did Juniper users ever patch?



Juniper

SRX Series Service Gateways (SRX100, SRX110, SRX210, SRX220, SRX240, SRX550, SRX650), LN1000 Mobile Secure Router

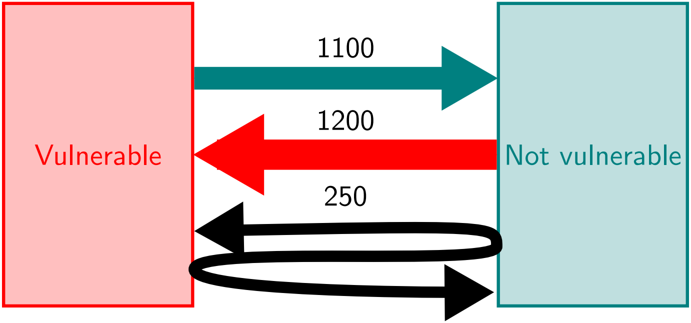
Did Juniper users ever patch?



Juniper

SRX Series Service Gateways (SRX100, SRX110, SRX210, SRX220, SRX240, SRX550, SRX650), LN1000 Mobile Secure Router

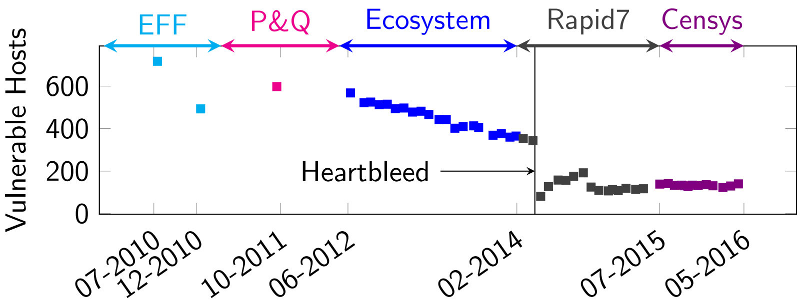
Did Juniper users ever patch?



IBM

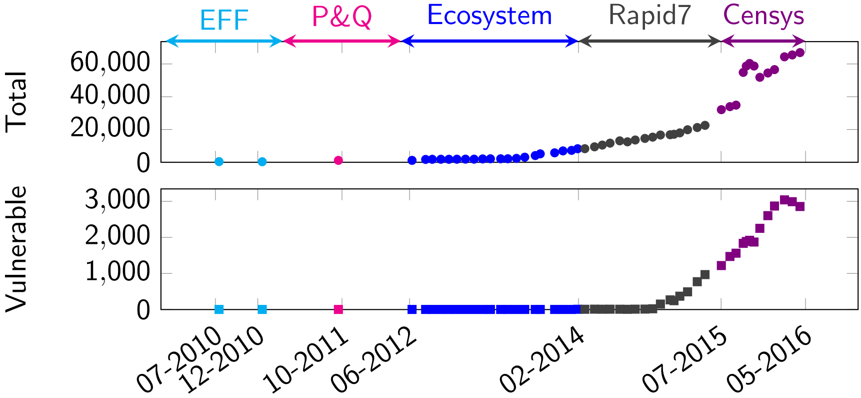
Remote Supervisor Adapter II, BladeCenter Management Module

- Public security advisory (CVE-2012-2187) in September 2012
- Prime generation bug: 36 possible public keys from 9 primes
- 100% of fingerprintable moduli are vulnerable



Huawei

- Introduced vulnerability in 2014
- Security advisory published Aug 2016



Non-RSA cryptographic RNG disasters

- DSA: 1% of SSH host private keys revealed from nonce collisions. [HDWH 2012]
- ECDSA: Android Bitcoin wallet vulnerability; dozens–hundreds of bitcoins stolen in 2013.
- AES-GCM: Fixed or colliding nonces. [Böck, Zauner, Devlin, Somorovsky, Joanovic 2016]
- Dual-EC: Juniper ScreenOS malicious code insertion.

Mining your Ps and Qs: Widespread Weak Keys in Network Devices
Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman
Usenix Security 2012 https://factorable.net

"Ron was wrong, Whit is right" published as
Public Keys Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter *Crypto 2012*

Elliptic Curve Cryptography in Practice Joppe W. Bos, J. Alex Halderman, Nadia Heninger, Jonathan Moore, Michael Naehrig, and Eric Wustrow. *Financial Cryptography 2014*

Factoring RSA keys from certified smart cards: Coppersmith in the wild
Daniel J. Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko van Someren, *Asiacrypt 2013.*

A Systematic Analysis of the Juniper Dual EC Incident. Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Matthew Green, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, and Hovav Shacham. *CCS 2016.*

Weak keys remain widespread in network devices Marcella Hastings, Joshua Fried, and Nadia Heninger. *IMC 2016*