Weak Keys Remain Widespread in Network Devices

Marcella Hastings
University of Pennsylvania
mhast@cis.upenn.edu

Joshua Fried
University of Pennsylvania
fjosh@cis.upenn.edu

Nadia Heninger
University of Pennsylvania
nadiah@cis.upenn.edu

ABSTRACT

In 2012, two academic groups reported having computed the RSA private keys for 0.5% of HTTPS hosts on the internet, and traced the underlying issue to widespread random number generation failures on networked devices. The vulnerability was reported to dozens of vendors, several of whom responded with security advisories, and the Linux kernel was patched to fix a boot-time entropy hole that contributed to the failures.

In this paper, we measure the actions taken by vendors and end users over time in response to the original disclosure. We analyzed public internet-wide TLS scans performed between July 2010 and May 2016 and extracted 81 million distinct RSA keys. We then computed the pairwise common divisors for the entire set in order to factor over 313,000 keys vulnerable to the flaw, and fingerprinted implementations to study patching behavior over time across vendors. We find that many vendors appear to have never produced a patch, and observed little to no patching behavior by end users of affected devices. The number of vulnerable hosts increased in the years after notification and public disclosure, and several newly vulnerable implementations have appeared since 2012. Vendor notification, positive vendor responses, and even vendor-produced public security advisories appear to have little correlation with end-user security.

Keywords

Security vulnerabilities; networked devices

1. INTRODUCTION

In February of 2012, Lenstra, Hughes, Augier, Bos, Kleinjung, and Wachter [26] and Heninger, Durumeric, Wustrow, and Halderman [21] announced that they had factored tens of thousands of public RSA keys from TLS certificates. The latter group traced the vulnerability to widespread failures in random number generators on headless, embedded, and low-resource network devices.

While this vulnerability revealed the private keys for 0.5% of HTTPS hosts on the internet at the time, the immediate security impact on those particular hosts was likely limited by the fact that the vast majority of compromised TLS certificates were not browser trusted, and were automatically-generated certificates protecting web login interfaces on consumer and small-business grade routers and firewalls, remote server administration interfaces, and embedded device such as cooling systems, building monitors, cameras, projectors, and printers. Rather, the presence of weak keys was an externally visible signal that the random number generation subsystems on entire classes of devices and across multiple implementations had failed, likely impacting these systems and many other hosts in a variety of ways.

In this paper, we study the disclosure process undertaken by the authors of [21] and the aftermath of this family of random number generation failures. We aggregated publicly available internet-wide TLS scan data dating back almost six years, including monthly scans during the past four years from multiple sources. We give more details on our data sources in Section 3.1. In order to measure the incidence of vulnerable keys, we extracted the 81 million distinct RSA moduli found in these scans and ran a batch GCD algorithm to factor improperly generated RSA keys. We were able to factor over 313,000 moduli. (See Table 1.) To carry out such a large computation efficiently for such a large number of keys, we modified the algorithm to run in parallel across a distributed cluster. We describe our computation in detail in Section 3.2. We then manually fingerprinted vulnerable implementations based on available metadata, including certificate subjects, certificate subject alternative names, port scans, and known private key behavior, as described in Section 3.3, and examine vulnerability rates over time for user populations associated with identified vendors in Section 4.

Previous studies of vulnerability patching have focused on patching behavior from system administra-

IMC '16, November 14–16, Santa Monica, CA, USA.
© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4526-2/16/11...$15.00
DOI: http://dx.doi.org/10.1145/2987443.2987486

tors in response to highly publicized vulnerabilities in OpenSSL [40, 15] or in response to targeted vulnerability notifications [27, 34]. In contrast, the vulnerability disclosures undertaken by the authors of [21] were targeted mainly at vendors of embedded and headless devices, which are rarely updated by end users.

The widespread nature of the flaw provides us with an accidental historical experiment allowing us to study the industry's response to disclosure by comparing responses across many vendors. We describe in detail for the first time the results from the industry-wide disclosure process undertaken by the authors of [21]. In particular, fewer than half of the organizations contacted about the vulnerabilities responded at all to a best-effort attempt at vulnerability notification.

We also give data on the post-notification internet-wide vulnerability rates for different vendors in response to the same vulnerability. Many vendors apparently continued to ship vulnerable products for years after receiving notification of and even acknowledging the security vulnerability, and in some cases after putting out security advisories for their products. Since the targets of our study are primarily *devices*, rather than software packages, end users must rely on vendors to distribute updates in order to patch their own devices. We observe that for this vulnerability, end-user patching did not appear to occur at all.

The single largest drop in the number of vulnerable keys occurred shortly after the disclosure of the Heartbleed vulnerability in April 2014. The decrease in vulnerable keys is confined to a handful of devices, for which there was an even larger concurrent drop in the total population of fingerprinted devices, suggesting that while in some cases the publicity may have prompted device users to regenerate certificates or apply a patch that also fixed the weak key vulnerability, many device HTTPS interfaces may instead have simply been taken offline. In the case of at least one device family, Heartbleed vulnerability scans were reported to render the device non-responsive. This suggests that end-user device security can improve when vendors patch vulnerabilities and end users are encouraged to apply patches or change configuration by widespread media attention and targeted vulnerability notifications, but that this process did not occur in response to the vulnerabilities we study in this paper.

2. BACKGROUND

2.1 TLS security

In this paper, we focus on RSA as used in the Transport Layer Security (TLS) protocol, which is used for HTTPS. Several studies have examined the HTTPS [16], email [14], and other TLS application infrastructures [22]. In the TLS protocol, a server's RSA public keys are included in a server's certificate, which is transmitted to the client in response to a client hello message. When client and server negotiate a cipher suite that includes RSA key exchange, the client uses the server's public key to encrypt session key information; when they negotiate a Diffie-Hellman or elliptic-curve Diffie-Hellman cipher suite, the server uses its RSA public key to sign its key exchange message to prove authenticity. RSA digital signatures are used to authenticate TLS certificates; a certificate may be signed by a trusted certificate authority's public key or by a certificate chaining up to a trusted root key, or self-signed by its own public key.

HTTPS host records	1,526,222,329
Distinct HTTPS certificates	65,285,795
Distinct HTTPS moduli	50,677,278
Total distinct RSA moduli	81,228,736
Vulnerable RSA moduli	313,330
Vulnerable HTTPS host records	2,964,447
Vulnerable HTTPS certificates	1,441,437

Table 1: We analyzed six years of internet-wide HTTPS scans in order to study server behavior over time. Each host record represents a HTTPS handshake on a given date with a server at some IP address. We augmented our batch GCD computation with RSA public keys extracted from IMAPS, POP3S, SMTPS, and SSH scans, but excluded these other protocols from further study.

A compromised public key from a TLS certificate could be used to passively decrypt TLS sessions that had negotiated RSA key exchange. An active attacker could use a compromised certificate key to impersonate a trusted server whose certificate used that key, or to perform a man-in-the-middle attack to decrypt or modify traffic using Diffie-Hellman or RSA cipher suites. 74% of the 61,240 vulnerable devices present in our most recent scan data from April 2016 only support RSA key exchange, making them vulnerable to passive decryption by an attacker who is able to observe network traffic. In order for an attacker to intercept network traffic, they would need to be on the network path between the victim client and the true server, or use a network attack like DNS or BGP hijacking [8] to force the victim to connect to the server via the client.

Among the vulnerable devices that we examined, HTTPS is used primarily to serve remote management interfaces. Many of the vulnerable firewalls also use TLS to encrypt SSL VPN connections. An attacker who is able to decrypt connections to one of these devices may obtain administrative credentials for the device or view remote user traffic to internal network resources.

2.2 RSA and factoring

RSA [31] is the most commonly used public-key cryptosystem on the internet. RSA keys can be used both for encryption and digital signatures. An RSA public key consists of a *modulus* N, which is the product of two large primes p and q, and a public exponent e.

Public Advisory	Private Response		Auto-Response	No Response		
IBM	Cisco	Pogoplug	Brocade	ZyXEL	Emerson	McAfee
Innominate	Sentry	HP	NTI	TP-Link	Fortinet	Dell
Intel	Technicolor	Hillstone Networks	Haivision	2-Wire	Sinetica	D-Link
Juniper	AudioCodes	Motorola		Xerox	SkyStream	Pronto
Tropos	Ruckus	Kronos		Kyocera	BelAir	Simton
	Linksys			AVM	JDSU	MRV

Table 2: 37 vendors were notified via email in February and March 2012 about weak TLS or SSH RSA key generation in their products. Only five released a public security advisory. About half of the vendors acknowledged receipt.

The corresponding RSA private key is a private exponent $d = e^{-1} \bmod (p-1)(q-1)$, which is efficient to compute if the prime factorization of N is known. If the prime factorization of N is not known, the most efficient method known in general to compute an RSA private key is to factor the modulus N into its prime factors and use the factors to calculate d.

The most efficient general-purpose factoring algorithm is the number field sieve, which has asymptotic complexity $\exp((1.923 + o(1))(\log N)^{1/3}(\log\log N)^{2/3})$ [25]. Factoring a 512-bit RSA key using the number field sieve is within the range of attackers of even modest resources today [38]; the current public factorization record is a 768-bit RSA modulus factored by an academic team over several years, announced in 2009 [24], and 1024-bit factorization is believed to be within the range of governments.

2.3 Factoring weak RSA keys

Implementation flaws can allow an attacker to break RSA much more efficiently. The family of implementation flaws we study in this paper results in the generation of RSA moduli that share a single common prime factor. In that case, one might have two RSA moduli $N_1 = pq_1$ and $N_2 = pq_2$ that share the prime factor p but have different second factors q_1 and q_2. The two moduli will appear distinct, but an attacker who can find such a pair can easily factor both of them by computing $p = \gcd(N_1, N_2)$ and dividing to discover q_1 and q_2. These two operations can be performed in less than one second on a standard modern laptop.

The authors of [26] and [21] discovered such flaws were widespread by collecting several million RSA public keys from publicly available datasets, including HTTPS certificate scans, PGP key servers, and SSH host scans, and computing the pairwise common divisors of every pair of RSA moduli in the datasets. (See Section 3.2.)

Since these original publications, the same technique has been used to find further vulnerable implementations: Bernstein et al. [6] found a random number generation flaw in the Taiwanese national smart card infrastructure, and Albrecht, Papini, Paterson, and Villanueva-Polanco [3] found a large number of weak keys among export-grade RSA keys for TLS.

2.4 The random number generation flaw

The implementation flaws resulting in these weak keys appear to have arisen independently in many different implementations, but [21] points to a common pattern: on many headless, embedded, or low-resource devices, the operating system random number generator may not have incorporated any external sources of entropy when it is used by an application to generate a cryptographic key. If the key-generation process incorporates additional low-entropy sources of randomness during key generation—for example the current time or arriving network packets—applications running on different systems may have identical states during the generation of the first prime factor of the RSA modulus, and diverge during the generation of the second prime factor, resulting in the exact situation described above.

The authors of [21] discovered a boot-time "entropy hole" vulnerability in the Linux random number generator, where the output of `/dev/urandom` could be deterministic on boot in real usage, and noted that in this situation OpenSSL's key generation behavior could result in factorable keys. This combination was visible on many but not all of the affected product families.

The vulnerable factored keys were almost exclusively confined to network devices, rather than general purpose web servers. Only a handful of the vulnerable HTTPS certificates were signed by a browser-trusted CA. While most of the vulnerable hosts were likely low-value targets, their weak keys are an externally visible sign of the underlying flaw in the operating system, with broader security implications for any service on the system requiring randomness. This type of flaw is difficult to detect in a standard unit test framework, making it difficult for vendors to discover and mitigate in normal software development practices.

2.5 Responsible disclosure

The authors of [21] notified 61 vendors of the vulnerability between February and June 2012. Of these, 37 concerned vulnerable RSA keys, and the remainder produced vulnerable DSA signatures only. Since our main dataset consists only of RSA keys, we exclude the DSA vulnerabilities from further analysis. Among the RSA vulnerabilities, 28 vendor devices produced vulnerable TLS certificates, and the rest produced vulnerable

	7/2010 (EFF)	4/2016 (Censys)
TLS Handshakes	11,261,212	38,014,832
Distinct Certificates	5,479,537	10,670,210
Distinct RSA Keys	5,333,832	10,457,597

Table 3: Our dataset includes nearly six years of scans; we summarize the earliest and latest above.

RSA SSH host keys. A complete list of affected vendors was never published. The authors first attempted to contact each vendor individually via email. They were able to find a security contact, either an email address or web form, for 13 vendors by searching company web sites, and in two cases via personal connections. For the rest, they attempted to email `security@domain.com` and `support@domain.com`. Later, CERT/CC and ICS-CERT aided in contacting and coordination between companies. Table 2 lists the vendors who were notified about weak RSA keys and their responses.

In addition, the authors of [21] notified the maintainers for the Linux kernel, who published a patch in July 2012 including several mitigations against the failures [36]. In July 2014, Linux introduced the `getrandom()` system call, which outputs nonblocking data only after it has been properly seeded. This is the desirable functionality for cryptographically secure random number generation. [37]

3. METHODOLOGY

In order to examine vulnerability rates over time, we collected historical HTTPS scan data, extracted the RSA public keys from server certificates, and performed a massive batch GCD computation across all of the distinct RSA keys ever seen in our scans. We fingerprinted certificates to identify the server implementation.

3.1 Data sources

While weak keys have been found across many public-key infrastructures, we chose to focus on HTTPS data, as previous and ongoing measurement studies have provided a historical record to draw from.

The earliest internet-wide HTTPS scans we are aware of are the EFF SSL Observatory [18], collected by Burns and Eckersley in July 2010 and December 2010. These two scans were performed over the course of two to three months each, using Nmap [28] to scan for hosts with port 443 open and a custom Python client to send a client hello and collect a server's certificate. We refer to this dataset as "EFF".

Heninger, Durumeric, Wustrow, and Halderman [21] performed an internet-wide HTTPS scan over five days in October 2011, first using Nmap to scan hosts with open TCP port 443, and using a custom Python client to collect certificates over the course of several days. We refer to this dataset as "P&Q". Durumeric, Kasten, Bailey, and Halderman began taking regular scans of

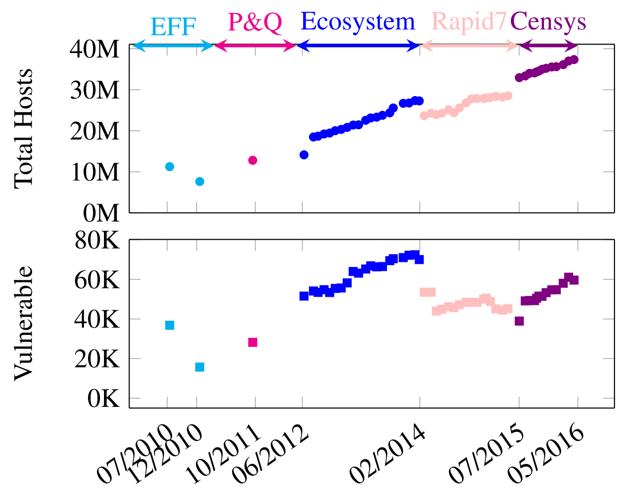

Figure 1: We aggregated HTTPS scan data from several sources. The total number of hosts in each scan is shown above, and the number of hosts serving keys we were able to factor below.

the HTTPS ecosystem [16] in June 2012 and continued through January 2014; we refer to this dataset as "Ecosystem". These scans used the much faster Zmap [17] port scanner and a custom certificate fetcher, and the authors report that a single scan took 18 hours on average to complete. The scans vary in frequency from one to over 20 scans per month. Beginning in October 2013, Rapid7's Project Sonar [30] took weekly scans over IPv4 space, also using the Zmap port scanner with their own custom client implementation to collect HTTPS certificates. We use their HTTPS data through May 2015. Finally, Durumeric, Adrian, Mirian, Bailey, and Halderman [13] have performed daily scans from July 2015 through April 2016, made available through the Censys search engine, using Zmap and an integrated toolchain to collect TLS handshake data on port 443. The Censys team also performs regular scans of IMAPS, POP3S, and SMTPS using TLS and SSLv2; we extracted the RSA moduli for these datasets for use in our batch GCD calculation, but did not include the certificates or host records in the rest of our results. We also included Censys SSH host key data.

In total, our dataset includes 1.5 billion host records, representing an IP address and certificate pair on a given date, and 131 million unique certificates. There are 81.2 million RSA moduli, of which 313,330, or 0.37% were factored using the methods described in the following section. The data is hosted in a MySQL database on a machine with a fast 6TB SSD cache layer and 28TB of RAIDed HDDs for backup storage of the scan data.

Figure 1 illustrates the number of hosts seen over time on port 443 and the number of hosts on port 443 whose public keys we were able to factor. Artifacts from the different scan methodologies used by each team are clearly visible. Although most of these sources provided

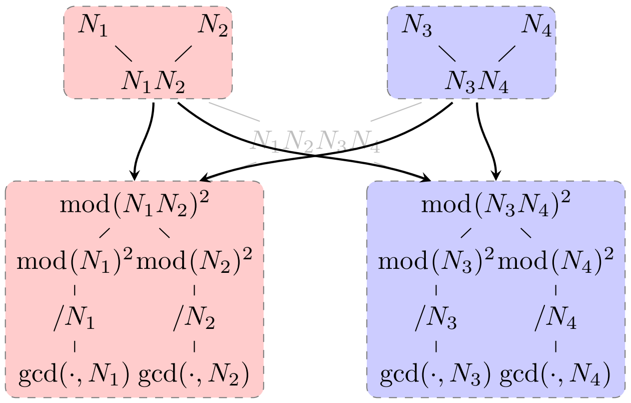

Figure 2: To parallelize the batch GCD computation over a cluster, we divide the input into k subsets and compute a product only over each subset. For each subset, we compute the remainder tree with respect to all subset products. The total amount of computation is higher, but we achieve a speedup in practice by avoiding the bottleneck of computing with respect to the very large products in the center of the tree.

more frequent scans, we chose one representative scan per month over the time period we examined. In all figures, each point represents a single scan.

We also found that the Rapid7 data included sets of intermediate certificates without explicitly chaining them, while the other scan data either excluded the issuer certificates or explicitly chained them. In order to better correlate our results across datasets, we excluded these intermediate certificates from our analysis by reconstructing the chains using common names among all certificates associated with each IP address and including only the lowest certificate in the chain.

3.2 Batch GCD

In order to compute pairwise GCDs across all collected moduli, we adapted the batch GCD implementation described in [21] [1]. This code was originally reported to perform the batch GCD computation on a set of 11.1 million RSA moduli in 1.6 hours on a 16-core Amazon cloud cluster compute instance with 60.5 GB of RAM. The batch GCD algorithm is fundamental to the feasibility of our study: it runs in quasilinear time over the number of input moduli, while a naive implementation that simply computes the GCD of all pairs of moduli runs in quadratic time in the number of moduli. The computation time required by the latter is not feasible for the dataset sizes used in this paper.

The algorithm, which is adapted from an algorithm described by Bernstein [5], has two main phases. First, it uses a product tree to compute the product of all input moduli $P = \prod N_i$. (A product tree computes the product of its inputs using a binary tree of multiplications.) Then it uses a remainder tree to compute $z_i = P \bmod N_i^2$ for every N_i. Finally, for each N_i, it outputs $\gcd(N_i, z_i/N_i)$. If this value is not 1, then N_i shares some common factor with at least one other modulus in the database.

Scaling this calculation to the 81 million moduli posed some computational challenges. First, the code as written was limited to running on a single shared-memory system, and used threads to run arithmetic operations in parallel at each level of the tree. Second, the implementation uses GMP [20] for bignum arithmetic operations, which are single threaded. This resulted in a bottleneck at the central node in the middle of the tree, the product of all 81 million RSA moduli.

To solve both of these problems, we made several adaptations to the batch GCD algorithm that raise the asymptotic complexity of the calculation but allow a speedup in practice. The modified algorithm works as follows. Instead of computing the product $\prod N_i$ of $N_1...N_n$ moduli, we divide the moduli into k subsets and compute the product $\{P_1, \ldots, P_k\}$ for each subset. We then compute a remainder tree for each product with respect to each subset. Since we pair every product with every subset, we are guaranteed to have coverage of all possible pairs of moduli. This part of the computation scales quadratically in the number of subsets k. For each modulus N_i, this yields k expressions $\{P_1 \bmod N_i^2, P_2 \bmod N_i^2, \ldots, P_k \bmod N_i^2\}$. The algorithm then completes the final step as before, reporting a modulus N_i as vulnerable if any of the subproducts returned nontrivial common divisors.

This modification allowed us to parallelize the computation across a cluster. We ran the computation across a cluster of six machines with dual 18-core 2.30GHz Intel Xeon E5-2699 v3 processors and 128GB of RAM each, eight machines with dual 22-core 2.20GHz Intel Xeon E5-2699 v4 processors and 512GB of RAM each, and eight machines with dual 12-core 2.50GHz Intel Xeon E5-2680 v3 processors and 512GB of RAM each. We were additionally able to speed up the computation by storing the entirety of the product and remainder trees in RAM, where the original hardware used for the computation had limited memory, requiring that the trees be written to disk.

For the 81 million RSA keys in our database, we chose $k = 16$. The entire computation took 86 minutes of wall-clock time to finish on our cluster, representing 1089 CPU hours. The product and remainder trees required between 70 and 100 GB of storage on each node.

In contrast, running the original algorithm without modifications entirely in memory on a machine with quad 6-core 3.40GHz Intel Xeon E7-8893 processors with 3 TB RAM took 500 minutes and required over 500 GB of memory to store the product and remainder trees.

3.3 Fingerprinting Implementations

Once we had identified the weak RSA keys in our database, we linked them to the set of certificates con-

[1]Available publicly on https://factorable.net.

	HTTPS	SSH	POP3S	IMAPS	SMTPS
Date scanned	*04/11/2016*	*10/29/2015*	*04/25/2016*	*04/25/2016*	*04/25/2016*
Total hosts with public keys	38,014,832	10,730,527	4,533,094	4,544,158	3,292,031
Hosts with RSA keys	37,931,417	6,257,106	4,533,094	4,544,158	3,292,031
Vulnerable hosts	59,628	723	0	0	0

Table 4: We included datasets of RSA public keys on different protocols into our batch GCD computation, but found that the majority of vulnerable keys were associated with HTTPS. We excluded these other protocols from further analysis. The scan data above is from the Censys project.

taining these moduli and attempted to identify the underlying implementations using available information.

3.3.1 Certificate subject fingerprints

For many implementations, the certificate subjects clearly identified a device manufacturer and model. We identified the majority of host records using certificate subjects. We observed that for vulnerable implementations end users typically did not alter the default certificate values provided by the device, so it was straightforward to map from distinguished name to vendor.

For example, the Hewlett-Packard, Xerox, TP-LINK, and Conel s.r.o. devices generating weak keys in our dataset all generate certificates containing "O=*vendor*" in the distinguished name. We also used the distinguished name to directly identify models. Cisco certificates in particular used the organizational unit section of the distinguished name to refer to the model of the device. There were 5,274,287 certificates that identified a vendor in this fashion.

In other cases, sets of certificates had consistent distinguished or common names, but did not name a vendor. In some cases, the content being served via HTTPS for a host of interest was a login or informational page identifying the vendor and model of the device. For example, McAfee's SnapGear 300 certificate distinguished name had fields "CN=Default Common Name, O=Default Organization, OU=Default Unit…", etc. The content served over HTTPS for these hosts was the home page for a SnapGear Management Console. Similarly, every Juniper certificate contained the field "CN=system generated". We labeled 26,272,330 certificates from 18 vendors using this heuristic.

3.3.2 Fingerprinting prime behavior

We found that in the vast majority of cases, devices sharing prime factors were identified as the same vendor. We used this information to extrapolate vendors for some certificates we could not otherwise identify.

We used a heuristic to identify shared prime factors. For a vendor with clearly identifiable certificates, we created a pool of all prime factors generated by the vendor. We then examined our set of moduli and set aside all moduli produced using a prime from the pool. Any certificate containing one of these moduli, we labeled as the original vendor. Excepting a few special circumstances which we discuss below, this heuristic uniquely labeled certificates.

In a special case, a bug in the prime-generation code of certain IBM Remote Server Administration cards and BladeServer Management Modules led to only nine possible primes being generated. Every public key associated with these devices was the product of two of these primes. [21]. Most of these certificates did not contain information identifying IBM. We identified 3,229 certificates using an IBM prime, and labeled them all IBM.

In examining the IBM data, we found an overlap between a modulus in the vulnerable IBM clique and devices identified as Siemens Building Automation, an interface for building-wide operations such as alarms, time schedules, and HVAC systems. The Siemens-identified modulus began appearing in February 2013 and continued to appear in scans throughout the study. This affected 2,441 certificates. There were 18 vulnerable certificates identified as Siemens that did not use an IBM modulus. There were about 15,000 total Siemens certificates, which we identified using certificate subject information.

We used extrapolation via shared prime factors to label certificates associated with Fritz!Box DSL modems. Many Fritz!Box certificates had a common name with some subdomain of `myfritz.net`. Others filled in the alternative name with identifiable domain names: `DNS:fritz.fonwlan.box, DNS:fritz.box, DNS:www.fritz.box, DNS:myfritz.box, DNS:www.myfritz.box`. These were easy to identify. However, there were tens of thousands of certificates whose certificate subject identified only an IP address in octets. We labeled these as Fritz!Box if they shared a common prime factor with certificates fingerprinted via certificate data or Nmap host identification. We were able to label 20,717 certificates as Fritz!Box using these heuristics.

We also found an overlap in shared primes between Xerox and Dell certificates. On closer examination, we found that a set of machines with a certificate subject containing "OU=Dell Imaging Group" all shared primes with Xerox devices. In 2004, Dell announced a partnership with Fuji Xerox, which specializes in imaging technologies [23]. Later, reviews of Dell printers claim they are manufactured by Xerox [11]. This overlap affected 416 certificates.

3.3.3 Internet Rimon Man-in-the-Middle

In attempting to identify shared implementations via shared cryptographic keys, we discovered that an Israeli ISP, Internet Rimon, was substituting its own RSA modulus into the self-signed certificates being served by the internet-facing services on devices belonging to its customers. Only the public key and the signature (as well as the choice of hash function used in the signature) were changed; the rest of the certificate remained unchanged. Internet Rimon offers an opt-in content filtering service for its customers via installation of a root certificate on the customer's machine, but this is the first case we have seen of an ISP man-in-the-middling *inbound* encrypted connections to its customers' devices by substituting their public keys for a fixed key of its own. We saw 922 distinct IP addresses with this key, appearing in scans throughout the entire study period.

We did not factor the 1024-bit RSA modulus used by Internet Rimon.

3.3.4 Fingerprinting OpenSSL primes

Mironov [29] observes that OpenSSL uses a distinctive method of prime generation that eliminates primes p such that $p - 1$ is divisible by any of the first 2048 primes. He estimates that the probability that a randomly chosen 512-bit prime satisfies this property is about 7.5%. This property would also be satisfied if an implementation only generated "safe" primes (a prime p where $(p - 1)/2$ is also prime). We found that none of our labeled vulnerable vendors produced exclusively safe primes, although they do arise in the dataset by chance. We did not find any vulnerable implementations producing exclusively safe primes, suggesting that the property is a true OpenSSL fingerprint.

We can use this property to identify implementations as likely OpenSSL and definitely not OpenSSL by examining the fraction of prime factors of weak keys that satisfy this property. If an implementation uses OpenSSL to generate primes, every prime factor p_i for every modulus associated with this implementation should satisfy $p_i \neq 1 \bmod q_i$ for each of the 2048 primes checked by OpenSSL. If an implementation is not OpenSSL, we would expect that only a small fraction of the prime factors associated with this implementation would satisfy this property.

We found that 71,417 of the certificates with weak keys were not OpenSSL, about 5% of all certificates with broken moduli. Table 5 categorizes vendors according to this OpenSSL fingerprint. Because the fingerprint requires the private key, it only covers models generating vulnerable keys. For vendors like Juniper where we cannot distinguish different product models, we can only state that there exists a non-OpenSSL implementation generating weak keys.

Satisfy OpenSSL fingerprint		**Do not satisfy**
2Wire	IBM	DrayTek
AdTran	Innominate	Fortinet
Allegro	Linksys	Huawei
BridgeWave	McAfee	Juniper
Cisco	MitraStar	Kronos
Conel s.r.o.	Netgear	Siemens
D-Link	NTI	Xerox
Dell	Sangfor	ZyXEL
Fritz!Box	Schmid Telecom	
HP	ServerTech	
TP-LINK	SkyStream Networks	
	Thomson	

Table 5: OpenSSL uses a distinctive prime generation process that allows us to use the primes in a private RSA key to distinguish vendors who are likely using OpenSSL to generate keys from those that are not. We classified device vendors from our TLS scan data.

3.3.5 Bit errors

107 (0.03%) of the 313,330 vulnerable moduli in our database are not the product of two equal-sized primes, that is, they are not well-formed RSA moduli. These corresponded to 113 certificates in our database. In many cases, this appeared to be due to bit errors. For example, we found 11 different "certificate authority" certificates in our database with non-well-formed RSA keys that were at least one bit error from a nearly identical valid certificate. (In the case of the certificates with errors, the signatures of course will fail to verify.) One or more bit flips in a valid RSA modulus would be expected to produce a random integer; this integer would be divisible by some prime q_i with probability $1/q_i$. Thus such bit errors are likely to show up in the results of our batch GCD computations with divisors that are the product of many small prime factors. Anecdotally, such errors have been observed to appear in other large key corpuses such as the PGP keyservers. [7]

We set these cases aside and did not treat them as indicators of a flawed implementation, since they could have resulted from memory errors on the host, errors on the wire, or errors during storage or download of the datasets. A small portion of these certificates were presented multiple times by the same hosts across scans. Most, however, were seen exactly once, indicating some error with the transmission or storage of the key.

4. RESULTS

In this section, we examine the behavior of vendor devices based on the company response to the vulnerability disclosures by the authors of [21].

4.1 Vendors that released a public disclosure

Five companies released public security advisories for TLS vulnerabilities[2].

[2]https://factorable.net/advisories.html

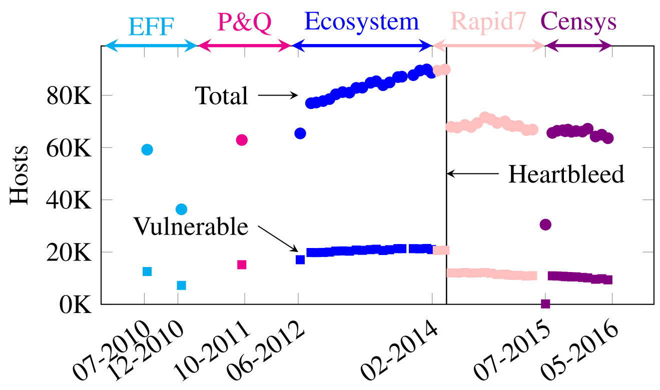

Figure 3: The number of vulnerable Juniper hosts continued to rise for years after Juniper's published security advisories and public disclosure. The single largest drop in the number of both vulnerable and total number of fingerprinted hosts occurred during April 2014, which correlates with the disclosure of Heartbleed, marked with a vertical line.

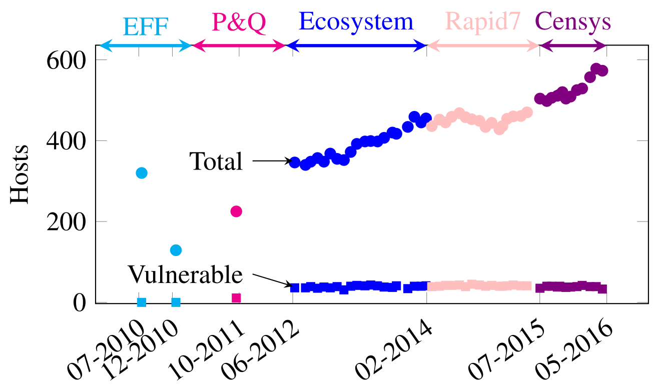

Figure 4: The number of broken Innominate mGuard hosts increased from the original 2012 notification and has stayed mostly consistent during the four years since the public security advisory.

Juniper, whose products represented the majority of factored keys in 2012 and the second highest number over all scans, released a public Security Bulletin in April 2012 and an Out-of-Cycle Security Notice in July 2012 announcing that the weak key vulnerability had been internally discovered and fixed. The exact device model is not apparent from the certificate data or the HTTPS login page, but Juniper has identified the vulnerable models as SRX branch security devices. Juniper certificates do not identify device models, and the default certificate matching our fingerprint can be generated by different Juniper models. We generated a certificate on a Juniper SSG5 security platform running ScreenOS that has the same fingerprint as the Juniper certificates in our scan.

Our data shows that that the number of vulnerable hosts continued to increase for two years following disclosure, and end-user patching was minimal at best. (Figure 3.) In order to better understand patching behavior, we examined the vulnerability status of fingerprinted Juniper hosts in more detail. Over the entire course of our scan data, we observed 169,000 IP addresses serving a fingerprinted Juniper certificate, of which 34,000 hosts served vulnerable keys. Of these, 1,100 hosts transitioned from a certificate containing a vulnerable key to a certificate containing a non-vulnerable key, 1,200 transitioned from a certificate containing a non-vulnerable key to a certificate containing a vulnerable key, and 250 transitioned between vulnerable and non-vulnerable certificates multiple times.

There is an abrupt decrease in the number of fingerprinted Juniper hosts, both total and those generating weak keys, between April 2014 and May 2014. Juniper released four security advisories in April 2014 for various vulnerabilities, including a patch for the OpenSSL "Heartbleed" issue on April 23, 2014 [1]. The security advisory states that the SRX branch devices identified as generating vulnerable keys were not vulnerable to Heartbleed.

During the aftermath of Heartbleed, nearly 30,000 Juniper-fingerprinted hosts went offline entirely, including over 9,000 vulnerable devices. This suggests that device owners disabled TLS on port 443, blocked external scans, or took devices offline in response to Heartbleed. Anecdotal reports suggest that Juniper NetScreen firewalls crashed when scanned for Heartbleed [39].

Innominate, now Phoenix Contact Cyber Security, manufactures security appliances for industry settings. In June 2012, they released a security advisory about weak keys in mGuard network security devices. Since the notification and advisory, the number of vulnerable mGuard hosts has remained roughly fixed. Out of 561 IP addresses that ever served vulnerable Innominate certificates, two hosts served non-vulnerable Innominate certificates and transitioned to a vulnerable public key in subsequent scans, three served vulnerable certificates and transitioned to non-vulnerable certificates in subsequent scans, and one host transitioned from non-vulnerable to vulnerable and back again. Over the time period covered by our scan data, the total number of Innominate-fingerprinted devices has risen (Figure 4), suggesting that the vulnerability has been fixed in newer devices, but the population of originally vulnerable devices continue to serve vulnerable certificates.

IBM BladeCenter Management Module and Remote Supervisor Adapter II cards generated only 36 possible public keys from 9 possible prime factors, as described in Section 3.3. 99.5% of the certificates we were able to identify as one of these IBM products contained these moduli. IBM published a security advisory (CVE-2012-2187) in September 2012. Nearly all certificates contained non-fingerprintable identifying information from the organizations themselves, although we found 51 cer-

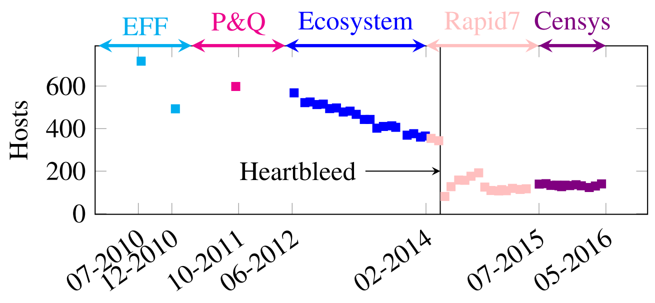

Figure 5: We fingerprinted IBM Remote Supervisor Adapter II and BladeCenter Management Modules using the 36 possible public keys that could be generated by these implementations. The vulnerable population appears to have been decreasing already by 2012, and shows a marked decrease at the time of the Heartbleed vulnerability in April 2014.

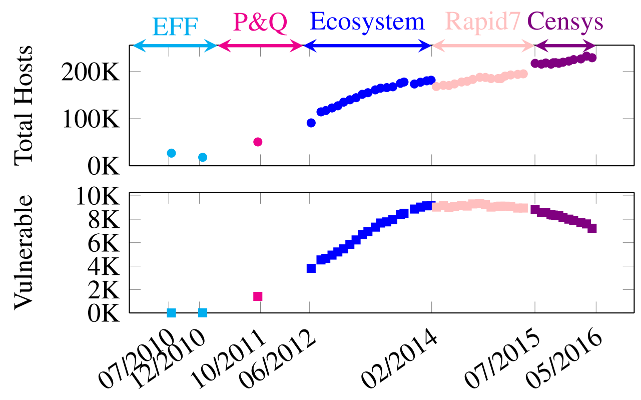

Figure 6: The number of broken Cisco hosts (bottom) increased steadily through 2014, although it has begun to decrease in the past year.

tificates labeled as Remote Supervisor Adaptors. For this reason, we do not have an estimate of the total population for these devices. Figure 5 shows the number of vulnerable hosts over time; the vulnerable population appears to have already been decreasing at the time of disclosure in 2012, and drops sharply in April 2014, corresponding to the Heartbleed disclsosure.

Examining certificate lifetimes and replacement on each host suggests that the vulnerable population of IBM devices was decreasing because devices (or their publicly accessible web interfaces) were taken offline altogether, and not because users patched the vulnerability and renewed their HTTPS certificates on the same device with a non-vulnerable public key. Among the 1,728 IP addresses that ever served a certificate containing one of the vulnerable IBM primes in any of our scans, 350 served a certificate during any of the scans that did not contain a vulnerable public key. The varying subjects of these new certificates indicated that these new certificates were due to IP churn.

Intel and Tropos released public vulnerability disclosures in the aftermath of [21], but these vulnerabilities concerned vulnerable SSH host keys on port 22, for which we do not have comprehensive data. Moxa also released a public disclosure for a vulnerability in DSA signatures, which we do not address in this study.

4.2 Vendors that responded privately

Several companies responded substantively to the vulnerability notification but did not release a public announcement.

Multiple lines of Cisco small business products generated weak keys. As mentioned in Section 3.3, most Cisco certificates clearly identified full model names in the certificate distinguished name, so we were able to label 81% of certificates (and 91.5% of broken certificates) with a model. Although Cisco responded in private to the vulnerability notice, they never released a public security advisory and do not appear to have patched the vulnerability for several years, as the number of vulnerable hosts continued to increase for the next three years. (Figure 6.)

The detailed model information in Cisco certificates allows us to study the effect of end-of-life announcements on device populations in the wild. Of the 14 vulnerable models we labeled, 10 have released an end-of-life announcement with a final sale date before May 6, 2016. We found that the end-of-life announcements marked the beginning of a slow decrease in the total number of devices online. We also note that the end-of-life announcement typically preceded the end-of-sale date by several months. (Figure 7.)

HP's Integrated Lights Out out-of-band management cards generated weak keys. The number of vulnerable hosts appears to have peaked in 2012, and has been decreasing steadily since, as shown in Figure 8. There is a notable drop in the number of vulnerable hosts as well as the total number of HP hosts in the months after the Heartbleed disclosure in April 2014. HP iLO cards were reported to crash when scanned for Heartbleed [39].

The population of other vendors' devices in this category was too small to draw conclusions from.

4.3 Vendors that never responded

Figure 9 illustrates the fingerprinted populations and number of vulnerable hosts for ten vendors who did not respond to the vulnerability notification. In most cases, the population of vulnerable hosts is declining gradually, outside of apparent differences due to different scanning methodology across our datasets. Four of these vendors (Thomson, Linksys, ZyXEL, and McAfee) show a decline in the vulnerable population that closely tracks the decline in the overall number of hosts with that device fingerprint. Fritz!Box shows a marked increase in the vulnerable population before an eventual decline, suggesting that the vulnerability may have been fixed for new devices at some point in 2014.

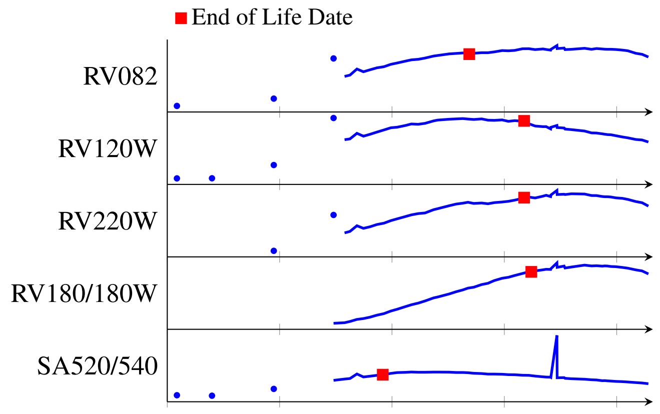

Figure 7: We found that Cisco's end-of-life announcements for their most popular small business routers generally mark the start of a gradual decline in the device population. We identified vulnerable hosts associated with all the device models in this figure except the RV082.

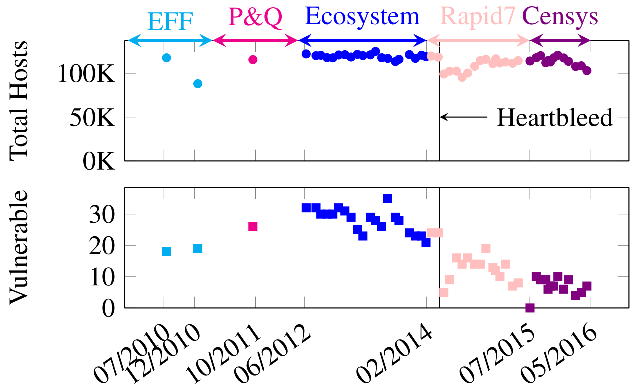

Figure 8: HP iLO out-of-band management cards reportedly crashed when scanned for the Heartbleed vulnerability. There is a noticeable decrease in the total population of HP products in the months following the Heartbleed disclosure.

4.4 Newly vulnerable products since 2012

Several companies had no or very few vulnerable devices in 2012, but appear to have since released products with the random number generation vulnerability. These newly vulnerable hosts are responsible for the rising number of vulnerable hosts visible over the past year in Figure 1.

The first vulnerable Huawei hosts appeared April 2015, and the population has increased dramatically (Figure 10). An examination of their certificates did not reveal a specific model. However, the certificates all identify a business unit in India, while 84% of the total population of Huawei certificates matching our fingerprint identify India. We contacted Huawei via their vulnerability reporting web page in May 2016. They responded to our notification and released a security advisory and software update in August 2016.

D-Link was contacted about HTTPS RSA vulnerability in 2012 and did not respond to the original notification. In 2012, the population of vulnerable devices was small, but has since increased dramatically. We contacted D-Link in May 2016 via their vulnerability reporting web form and have not received a response.

ADTRAN was notified in 2012 about DSA vulnerabilities in SSH and responded at the time. The vulnerability for HTTPS RSA keys seems to have been newly introduced in 2015. We contacted ADTRAN using the customer support form in May 2016. They responded substantively to our new notification, but have not yet released a public advisory or informed us of a software update.

We attempted to contact Sangfor using the customer support web form, but the request was closed.

The only contact we could find for Schmid Telecom was an Information Request web form; we did not receive a response to our notification. Schmid Telecom is a multinational company, but all their vulnerable certificates identify an Indian subsidiary.

5. DISCUSSION

Our data, viewed as a collection of individual anecdotes on vendor responses to a security vulnerability, illustrate the challenges faced by all parties in the vulnerability disclosure and mitigation process. From the perspective of the reporting party, the disclosure received no substantive response and no acknowledgement of a deployed fix for years for most vendors who were contacted. While the decline of the vulnerable population for many of these vendors suggests that their newer product versions are no longer naively vulnerable, this decline took years to become visible in the data. From the vendor side, the recent introduction of several newly vulnerable products illustrates how difficult it can be to keep old bugs out of new products.

5.1 Vendor notification processes

There is considerable debate in the security research community as well as among policymakers about the ethics of public disclosure of security vulnerabilities. The norm in the academic research community is generally "responsible disclosure" [2, 12] in which the researcher notifies the vendor before going public. However, the attempted notification of several dozen vendors studied in this paper shows how this process can fail at multiple stages. Most of the vendors did not have security contact information publicly available. We found, between our disclosure process and in [21], 16 out of 42 vendors had a discoverable point of contact for reporting vulnerabilities (see Sections 2.5 and 4.4 for details). The majority of vendors who were contacted in [21] never responded to the notification at all. The majority of those vendors who did respond in some way never released a

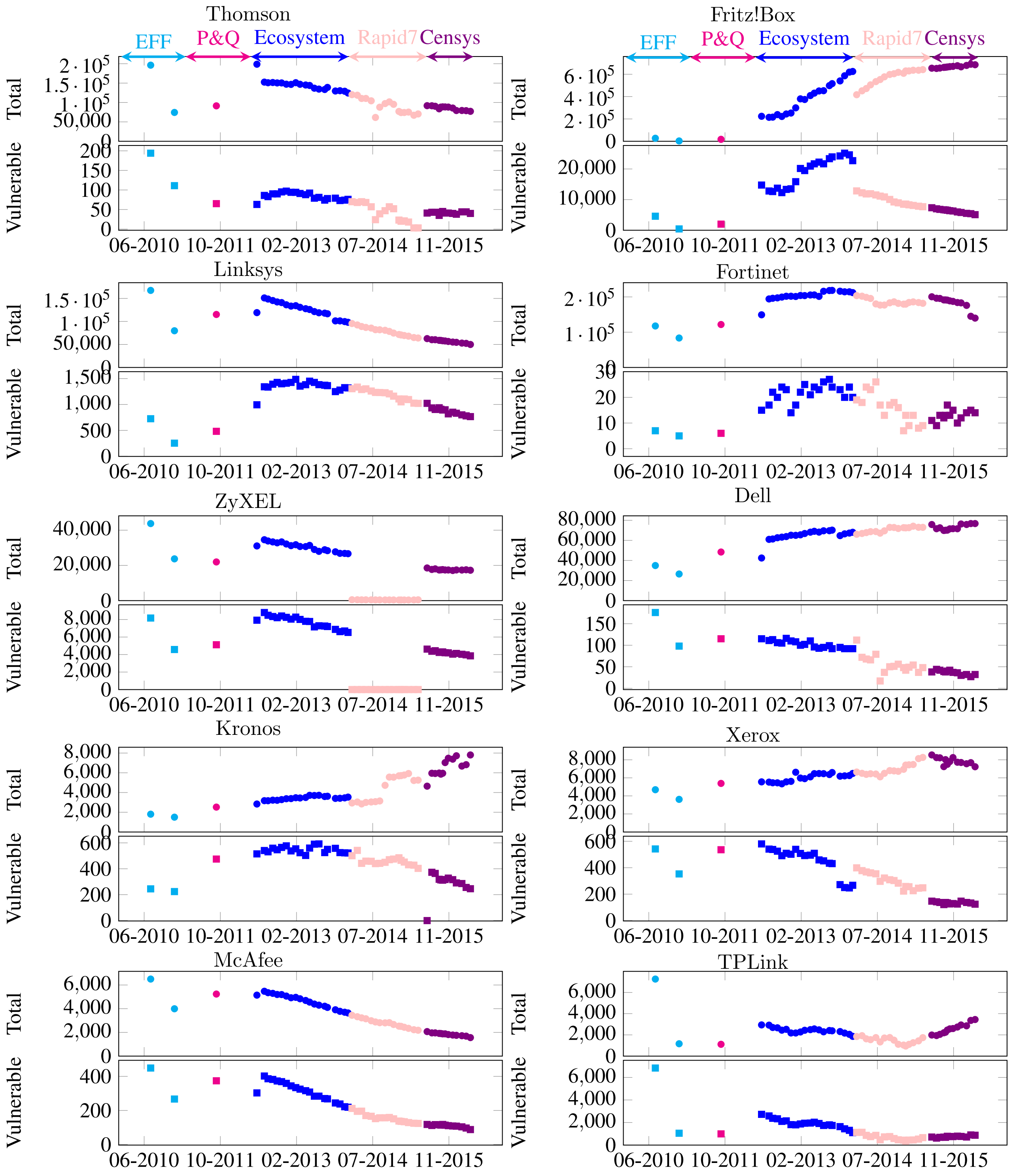

Figure 9: Many companies never responded to the vulnerability disclosure at all. In the above cases, the population of vulnerable hosts seems to be declining over the past several years.

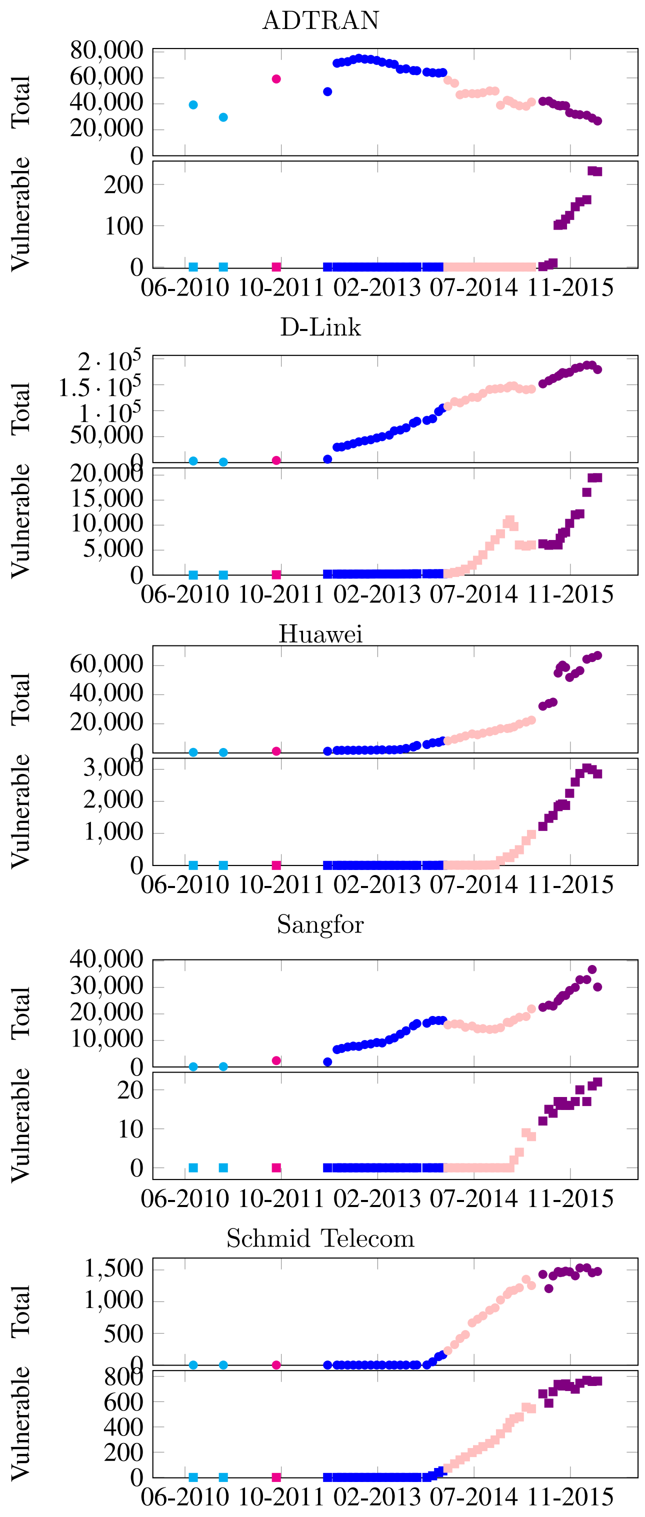

Figure 10: Some vendors had few or no vulnerable hosts in 2012, but appear to have released a vulnerable product version since then.

public security advisory. These figures were repeated on a smaller scale four years later in the more limited security disclosure process we undertook for this paper.

Reporting vulnerabilities through an organization such as CERT/CC is a potential alternative to directly contacting vendors. Arora et al. [4] found that response time from vendors decreases when reporting through such an organization. However, Li et al. [27] found that some CERT organizations did not follow up on the endpoint vulnerability disclosure information they shared, and concluded that reporting vulnerabilities through CERT organizations appeared to be of limited utility.

Despite the lack of organized vendor response, the eventual declining vulnerability rates for most vendors suggest that ultimately the underlying vulnerabilities were fixed for most new devices after several years. We hypothesize that this is likely due to newer products using updated versions of the Linux kernel that incorporate the software mitigations introduced by the maintainers in 2012. It remains an open problem to design an experiment to test this hypothesis.

Recommendations.

Software and hardware vendors should provide a dedicated means of contact for security vulnerability notifications, and when possible, an explicit statement of procedures concerning security vulnerabilities. Such a process should include a default timeline for public disclosure, acknowledgments, and a bug bounty policy for certain classes of flaws [19]. Alternatively, researchers can take advantage of the well-defined public disclosure process and vendor coordination provided by organizations like CERT/CC. In addition to providing a dedicated contact point, CERT/CC vulnerability disclosure policy includes a default 45-day deadline for public disclosure. In the case of the process we studied, coordination via CERT resulted in at least two additional public security advisories.

We leave it as an open problem to design mechanisms to incentivize vendors to use the most up-to-date versions of operating systems and libraries in their products, in order to avoid old vulnerabilities reappearing in new products, as we observed in Section 4.4.

5.2 Scoring and predicting vendor responses

Our data does not appear to show any correlation between company size or customer population and response to vulnerability notification, nor between vendor response and end-user vulnerability rates.

Future work might study in more detail the variables leading to different vendor responses, and design regulatory, economic, or commercial incentives to promote more effective security policies among vendors.

5.3 End-user patching behavior

In principle, our dataset should allow us to study end-user patching behavior, but this requires software and hardware vendors to make patches available. Only three vendors with HTTPS RSA vulnerabilities (Juniper, Innominate, and IBM) released a public security advisory and patch for the vulnerability in 2012, so our visibility into end-user patching behavior is limited.

Nevertheless, our data suggests that end user patching rates were low to nonexistent, and that decreases in the vulnerable populations were due to devices or their web interfaces being taken offline altogether. This does not seem to be due to devices being entirely unmaintained, as we see clear evidence of a drop in the vulnerable populations around the time of the Heartbleed disclosure, suggesting that publicity campaigns around vulnerabilities are both effective at changing user behavior, as observed previously [15], and that such campaigns can have unintended consequences. Since we do not see such significant events elsewhere in the dataset, nor even a declining trend in the vulnerable populations for Juniper or Innominate devices, it appears that most users do not have a regular patching process in place, and that widespread patching or mitigation for security flaws in these types of devices is a rare event.

We hypothesize that a major contributing factor to the lack of observed patching is that this vulnerability largely affected web interfaces for *devices* rather than web services. The low patch rates have distressing implications for the security of the "internet of things". Yu et al. [41] survey a number of critical vulnerabilities that remain unpatched across hundreds of thousands of devices in the wild.

Previous work on end-user patching rates [40, 15] for TLS vulnerabilities and vulnerability notifications [27, 34] have focused on system administrator behavior for web sites. In contrast, the families of devices affected by the random number generation vulnerabilities in our study fall into three broad classes: industrial control systems and critical infrastructure, router and firewall products aimed largely at the small business market, and cable and DSL modems and WiFi routers largely aimed at individual consumers.

Recommendations.

Our results show that public disclosure and patching of security vulnerabilities does not suffice to ensure end-user security. Additional processes will be necessary to ensure patches are deployed to consumers. The design of appropriate and effective software update mechanisms and incentives will likely depend heavily on the end users of the device.

For consumer-grade products, multiple studies have shown that individual users are unable to effectively evaluate security concerns (e.g. [9, 33]), and will tend not discover or apply patches. For these cases, actively pushing updates to devices is likely the most effective security update policy. Such a policy has significant drawbacks, however: software updates may produce unintended side effects or even brick devices [10]. There is evidence that most users will eventually apply optional patches that are pushed to devices they interact with regularly: Thomas et al. [35] found that half of all Android devices were updated within 30 days of a patch becoming available via Android's over-the-air push mechanism, and 95% within a year. However, many internet-of-things devices will likely lack a good interface for users to manage such updates, so designing a usable update mechanism remains an open problem.

In the case of devices aimed at small or medium business consumers, we recommend that vendors encourage customer IT staff to adopt a "Patch Tuesday"-type policy for device updates as well as software products. This requires vendors to remain on top of upstream software updates for the software stacks used in their products, and push or make available updated software versions in a standardized manner to customers. The positive responses by system administrators to active notification of software vulnerabilities [27, 34] and the effects of end-of-life policies on vulnerable device populations discussed in Section 4.2 suggests that they may be receptive to active update notifications from vendors.

The process of applying software updates to industrial control systems and critical infrastructure can be complicated by operational, regulatory, and legal considerations. It remains an open problem to design an effective software update mechanism that balances security and reliability of critical systems in this framework.

5.4 Ethics and responsible disclosure

Detailed information on the random number generation vulnerabilities studied in this paper, the efficient batch GCD implementation, and the data sets used in this paper have all been publicly available for years.

The majority of the vendors discussed in this paper were directly notified of the vulnerability in 2012. In the course of this research, we discovered five vendors who appeared to have introduced newly vulnerable products since 2012. We attempted to notify these vendors of the new vulnerability in May 2016. Only two companies have acknowledged receipt of our disclosure. Huawei released a public security advisory and software update in August 2016. The vulnerability was assigned CVE-2016-6670. We give more details in Section 4.4.

6. ACKNOWLEDGEMENTS

We thank Joshua Pearlstein for his contributions to the early stages of the project, and the anonymous reviewers for their extremely helpful and detailed feedback on our submission. This material is based upon work supported by the National Science Foundation under grants CNS-1513671, CNS-1408734, and a gift from Cisco. We are grateful to Cisco for donating the UCS servers we used for our computations.

7. REFERENCES

[1] Junos Pulse/IC (UAC): Details on fixes for OpenSSL "Heartbleed" issue (CVE-2014-0160)/JSA10623.

https://kb.juniper.net/InfoCenter/index?page=content&id=KB29007&pmv=print&actp=LIST, April 2014.

[2] Submitting papers: Human subjects and ethical considerations. https://www.usenix.org/conference/usenixsecurity16/submitting-papers, 2016. Accessed: 2016-09-01.

[3] Martin R. Albrecht, Davide Papini, Kenneth G. Paterson, and Ricardo Villanueva-Polanco. Factoring 512-bit RSA moduli for fun (and a profit of $9,000). https://martinralbrecht.files.wordpress.com/2015/03/freak-scan1.pdf, 2015.

[4] Ashish Arora, Ramayya Krishnan, Rahul Telang, and Yubao Yang. An empirical analysis of software vendors' patch release behavior: Impact of vulnerability disclosure. *Info. Sys. Research*, 21(1):115–132, March 2010.

[5] D. J. Bernstein. How to find smooth parts of integers. http://cr.yp.to/papers.html#smoothparts.

[6] Daniel J. Bernstein, Yun-An Chang, Chen-Mou Cheng, Li-Ping Chou, Nadia Heninger, Tanja Lange, and Nicko van Someren. *Advances in Cryptology - ASIACRYPT 2013: 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part II*, chapter Factoring RSA Keys from Certified Smart Cards: Coppersmith in the Wild, pages 341–360. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[7] Hanno Bock. About the supposed factoring of a 4096 bit RSA key. https://blog.hboeck.de/archives/872-About-the-supposed-factoring-of-a-4096-bit-RSA-key.html.

[8] Kevin Butler, Toni R. Farley, Patrick McDaniel, and Jennifer Rexford. A survey of BGP security issues and solutions. In *Proceedings of the IEEE*, volume 98, October 2013.

[9] Hong Chan and Sameera Mubarak. Article: Significance of information security awareness in the higher education sector. *International Journal of Computer Applications*, 60(10):23–31, December 2012.

[10] Amit Chowdhry. Apple confirms existence of the 'Error 56' iOS 9.3.2 bug. http://www.forbes.com/sites/amitchowdhry/2016/05/18/apple-ios-9-3-2-bug, 2016. Accessed: 2016-09-01.

[11] Edward J. Correia. Review: Dell C3765dnf workgroup color printer. http://www.crn.com/reviews/components-peripherals/240154599/review-dell-c3765dnf-workgroup-color-printer.htm, 2013.

[12] D. Dittrich and E. Kenneally. The Menlo Report: Ethical Principles Guiding Information and Communication Technology Research. Technical report, U.S. Department of Homeland Security, August 2012.

[13] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J. Alex Halderman. A search engine backed by Internet-wide scanning. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, October 2015.

[14] Zakir Durumeric, David Adrian, Ariana Mirian, James Kasten, Elie Bursztein, Nicolas Lidzborski, Kurt Thomas, Vijay Eranti, Michael Bailey, and J. Alex Halderman. Neither snow nor rain nor MITM...: An empirical analysis of email delivery security. In *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*, IMC '15, pages 27–39, New York, NY, USA, 2015. ACM.

[15] Zakir Durumeric, James Kasten, David Adrian, J. Alex Halderman, Michael Bailey, Frank Li, Nicolas Weaver, Johanna Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. The matter of Heartbleed. In *Proceedings of the 2014 Conference on Internet Measurement Conference*, IMC '14, pages 475–488, New York, NY, USA, 2014. ACM.

[16] Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman. Analysis of the HTTPS certificate ecosystem. In *Proceedings of the 13th Internet Measurement Conference*, October 2013.

[17] Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. ZMap: Fast Internet-wide scanning and its security applications. In *Proceedings of the 22nd USENIX Security Symposium*, August 2013.

[18] Peter Eckersley and Jesse Burns. An observatory for the SSLiverse. https://www.eff.org/files/DefconSSLiverse.pdf, 2010.

[19] Chris Evans, Eric Grosse, Neel Mehta, Matt Moore, Tavis Ormandy, Julien Tinnes, Michal Zalewski, and Google Security Team. Rebooting responsible disclosure: a focus on protecting end users. https://security.googleblog.com/2010/07/rebooting-responsible-disclosure-focus.html, July 2010. Accessed: 2016-09-01.

[20] Torbjörn Granlund and the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*, 5.0.5 edition, 2012.

[21] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Proceedings of the 21st USENIX Security Symposium*, August 2012.

[22] Ralph Holz, Johanna Amann, Olivier Mehani, Matthias Wachs, and Mohamed Ali Kaafar. TLS in the wild: an Internet-wide analysis of TLS-based protocols for electronic

communication. *Proceedings of NDSS 2016*, February 2016.

[23] Cynthia B. Johnston and Darlene Caldarelli. Xerox Corp.: Xerox International Partners and Fuji Xerox align with Dell to expand imaging and printing marketplace. http://www.businesswire.com/news/home/20040108005690/en/Xerox-Corp.-Xerox-International-Partners-Fuji-Xerox, 2004.

[24] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K Lenstra, Emmanuel Thomé, Joppe W Bos, Pierrick Gaudry, Alexander Kruppa, Peter L Montgomery, Dag Arne Osvik, et al. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology*, CRYPTO '10, pages 333–350. Springer, 2010.

[25] A. K. Lenstra and H. W. Lenstra, Jr., editors. *The Development of the Number Field Sieve.* Springer, 1993.

[26] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. Ron was wrong, Whit is right. In *32nd International Cryptology Conference*, CRYPTO '12, August 2012. http://eprint.iacr.org/2012/064.

[27] Frank Li, Zakir Durumeric, Jakub Czyz, Mohammad Karami, Michael Bailey, Damon McCoy, Stefan Savage, and Vern Paxson. You've got vulnerability: Exploring effective vulnerability notifications. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1033–1050, Austin, TX, August 2016. USENIX Association.

[28] Gordon Fyodor Lyon. *Nmap network scanning: The official Nmap project guide to network discovery and security scanning.* Insecure, 2009.

[29] Ilya Mironov. Factoring RSA moduli. part ii. https://windowsontheory.org/2012/05/17/factoring-rsa-moduli-part-ii/, May 2012.

[30] Rapid7. Project Sonar SSL Certificates. https://scans.io/study/sonar.ssl, April 2016.

[31] Ronald L Rivest, Adi Shamir, and Len Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[32] Hanna Smigala and Amanda Naiman. Siemens and IBM team on next generation of cloud-based building energy management solutions. http://www-03.ibm.com/press/us/en/pressrelease/49159.wss, February 2016.

[33] Jeffrey M. Stanton, Kathryn R. Stam, Paul Mastrangelo, and Jeffrey Jolton. Analysis of end user security behaviors. *Comput. Secur.*, 24(2):124–133, March 2005.

[34] Ben Stock, Giancarlo Pellegrino, Christian Rossow, Martin Johns, and Michael Backes. Hey, you have a problem: On the feasibility of large-scale web vulnerability notification. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 1015–1032, Austin, TX, August 2016. USENIX Association.

[35] Daniel R. Thomas, Alastair R. Beresford, and Andrew Rice. Security metrics for the Android ecosystem. In *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '15, pages 87–98, New York, NY, USA, 2015. ACM.

[36] Theodore Ts'o. /dev/random fixups. https://lkml.org/lkml/2012/7/5/414.

[37] Theodore Ts'o. random: introduce getrandom(2) system call. https://lwn.net/Articles/605828/, 2014.

[38] Luke Valenta, Shaanan Cohney, Alex Liao, Joshua Fried, Satya Bodduluri, and Nadia Heninger. Factoring as a service. In *Financial Cryptography.* Springer, 2016.

[39] Rob VandenBrink. Be careful what you scan for! https://isc.sans.edu/forums/diary/Be+Careful+what+you+Scan+for/18017/.

[40] Scott Yilek, Eric Rescorla, Hovav Shacham, Brandon Enright, and Stefan Savage. When private keys are public: Results from the 2008 Debian OpenSSL vulnerability. In Anja Feldmann and Laurent Mathy, editors, *Proceedings of IMC 2009*, pages 15–27. ACM Press, November 2009.

[41] Tianlong Yu, Vyas Sekar, Srinivasan Seshan, Yuvraj Agarwal, and Chenren Xu. Handling a trillion (unfixable) flaws on a billion devices: Rethinking network security for the Internet-of-things. In *Proceedings of the 14th ACM Workshop on Hot Topics in Networks*, HotNets-XIV, pages 5:1–5:7, New York, NY, USA, 2015. ACM.