

Healing Heartbleed

Vulnerability Mitigation with Internet-wide Scanning

J. Alex Halderman

Based on joint work:

Mining Your Ps and Qs: Widespread Weak Keys in Network Devices

Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman

21st Usenix Security Symposium (Sec '12), August 2012

ZMap: Fast Internet-Wide Scanning and Its Security Applications

Zakir Durumeric, Eric Wustrow, and J. Alex Halderman

22nd Usenix Security Symposium (Sec '13), August 2013

Analysis of the HTTPS Certificate Ecosystem

Zakir Durumeric, James Kasten, Michael Bailey, and J. Alex Halderman

13th Internet Measurement Conference (IMC '13), October 2013

An Internet-Wide View of Internet-Wide Scanning

Zakir Durumeric, Michael Bailey, and J. Alex Halderman

23rd USENIX Security Symposium (Sec '14), August 2014

Zipper ZMap: Internet-Wide Scanning at 10Gbps

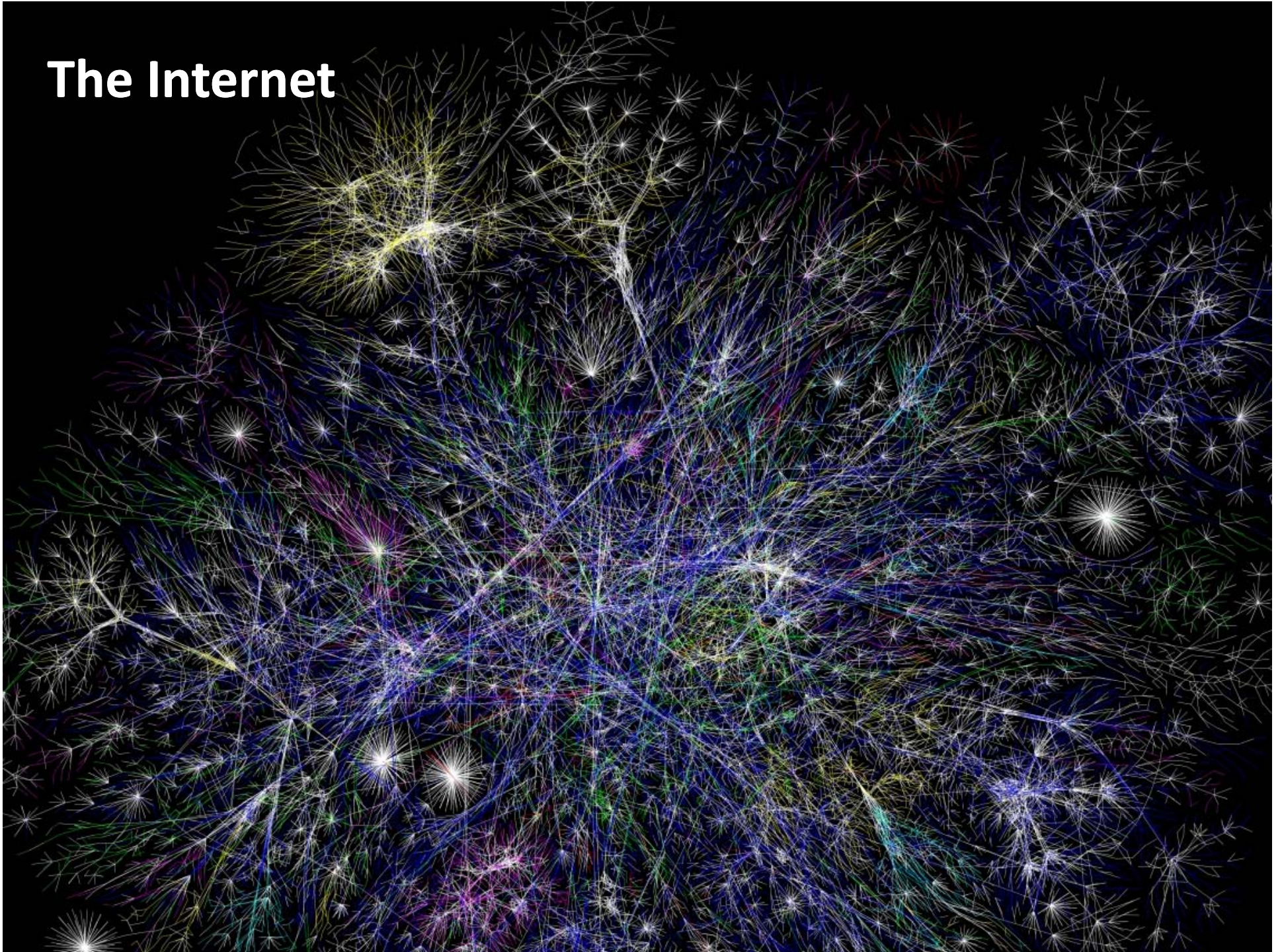
David Adrian, Zakir Durumeric, Gulshan Singh, and J. Alex Halderman

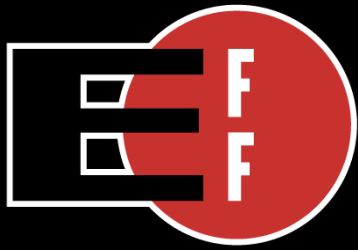
8th USENIX Workshop on Offensive Technologies (WOOT '14), August 2014

The Matter of Heartbleed

Zakir Durumeric, James Kasten, J. Alex Halderman, Michael Bailey, Frank Li, Nicholas Weaver, Bernhard Amann, Jethro Beekman, Mathias Payer, and Vern Paxson. *In submission.*

The Internet



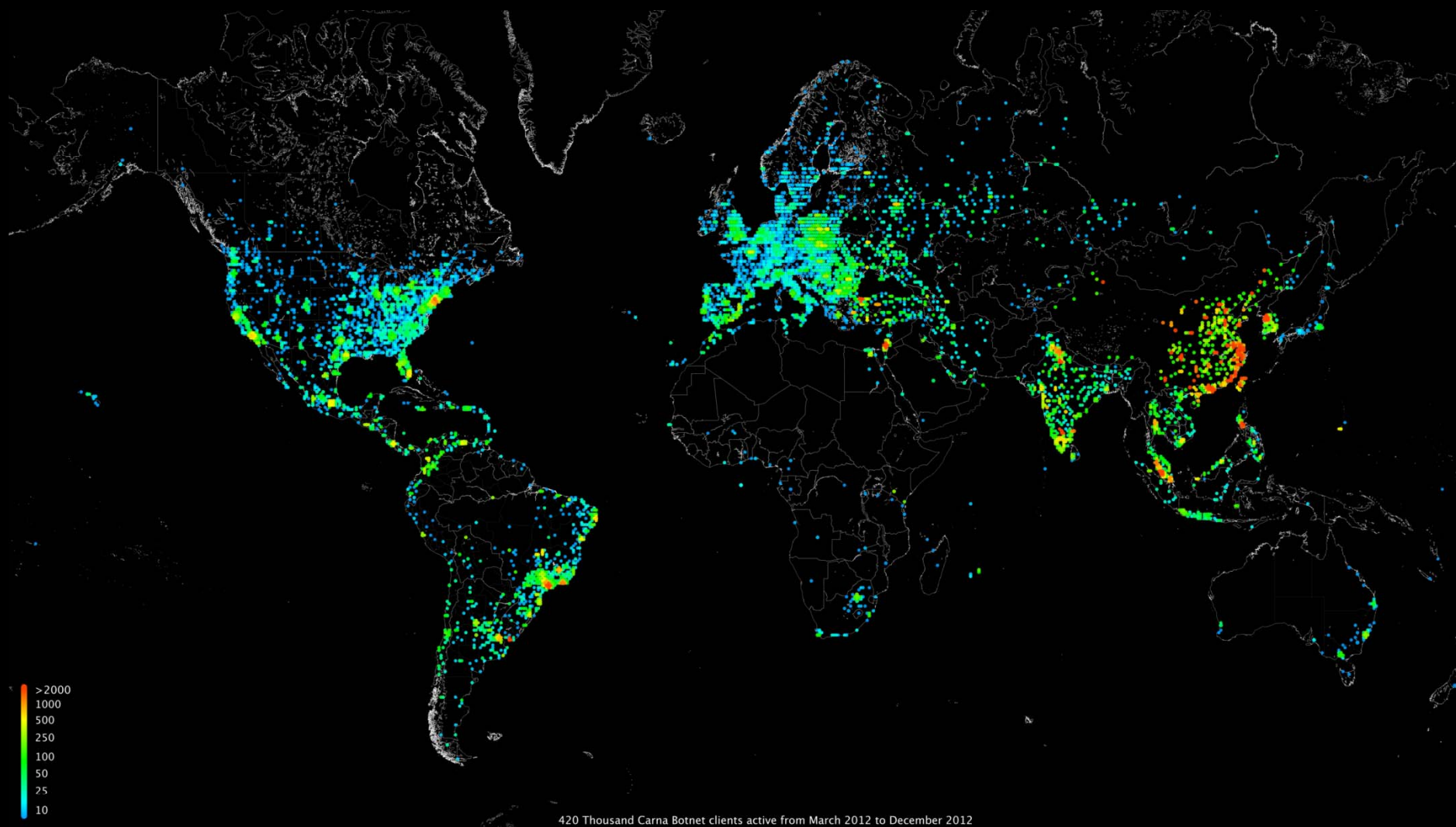


Electronic Frontier Foundation

SSL Observatory



Carna botnet Internet Census 2012



420 Thousand Carna Botnet clients active from March 2012 to December 2012

Barriers to using Internet-wide scans?

Census and Survey of the Visible Internet (2008)

3 months to complete ICMP census (2200 CPU-hours)

EFF SSL Observatory: A glimpse at the CA ecosystem (2010)

3 months on 3 Linux desktop machines (6500 CPU-hours)

Mining Ps and Qs: Widespread weak keys in network devices (2012)

25 hours across 25 Amazon EC2 Instances (625 CPU-hours)

Carna botnet Internet Census (2012)

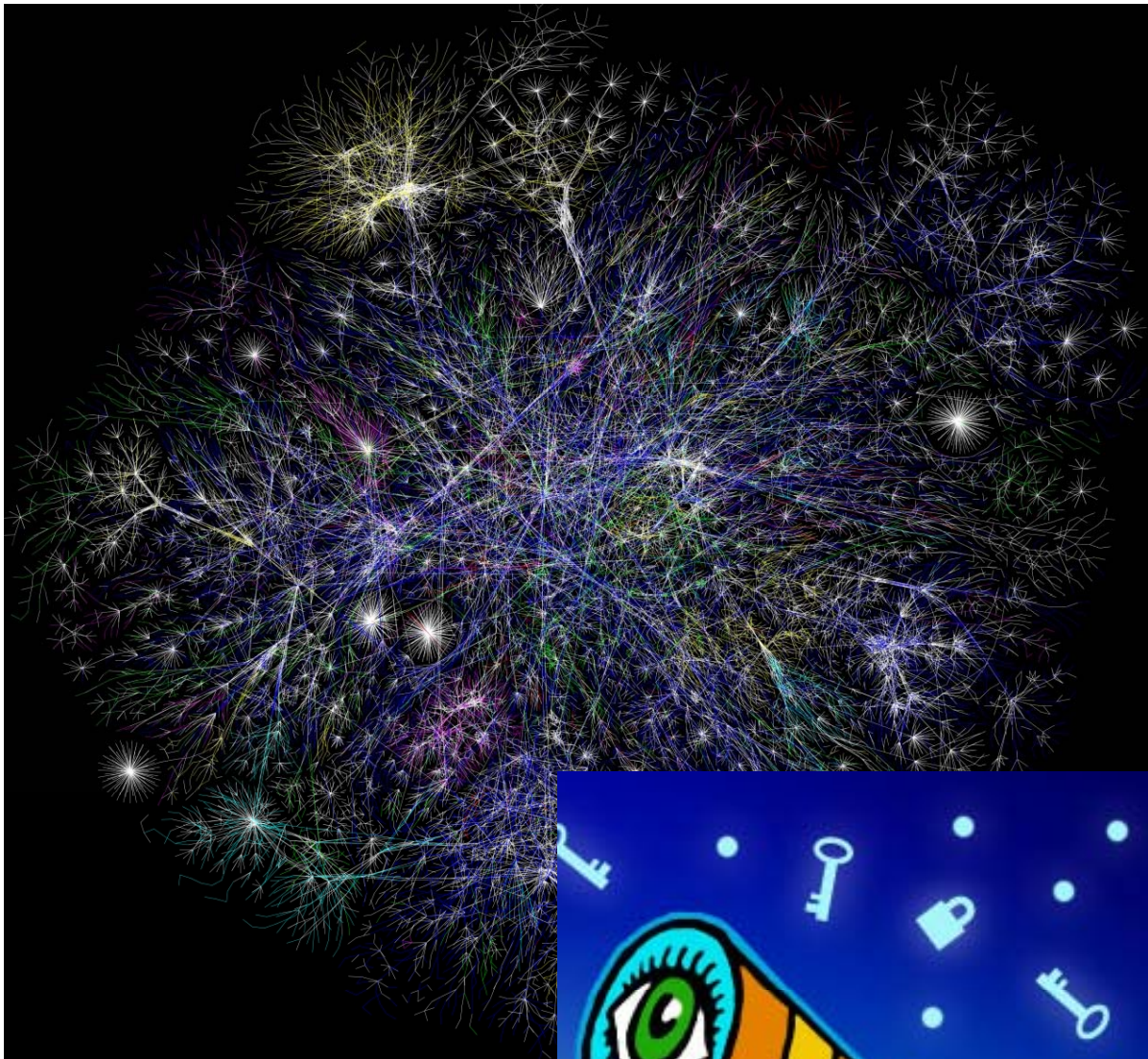
420,000 usurped hosts

What if...?

What if Internet-wide surveys didn't require heroic effort?

What if scanning the IPv4 address space took under an hour?

What if we wrote a whole-Internet scanner from scratch?





an **open-source tool** that can port scan the entire IPv4 address space from just **one machine** in **under 45 minutes** with 98% coverage



With ZMap, an Internet-wide TCP SYN scan on port 443 is as easy as:

```
$ sudo apt-get install zmap
$ zmap -p 443 -o results.csv
found 34,132,693 listening hosts
(took 44m12s)
```

97% of gigabit Ethernet
linespeed (1200 x NMAP)

Ethics of Active Scanning

Considerations

- Impossible to request permission from all owners

- No IP-level equivalent to robots exclusion standard

- Administrators may believe that they are under attack

Reducing Scan Impact

- Scan in random order to avoid overwhelming networks

- Signal benign nature over HTTP and w/ DNS hostnames

- Honor all requests to be excluded from future scans

Be a good neighbor!

ZMap Architecture

Typical Port Scanners

Reduce state by scanning in batches

- Time lost due to blocking
- Results lost due to timeouts

Track individual hosts and retransmit

- Most hosts will not respond

Avoid flooding through timing

- Time lost waiting

Utilize existing OS network stack

- Not optimized for immense number of connections

ZMap Approach

Eliminate local per-connection state

- Fully asynchronous components
- No blocking except for network

Shotgun scanning approach

- Always send n probes per host

Scan widely dispersed targets

- Send as fast as network allows

Probe-optimized network stack

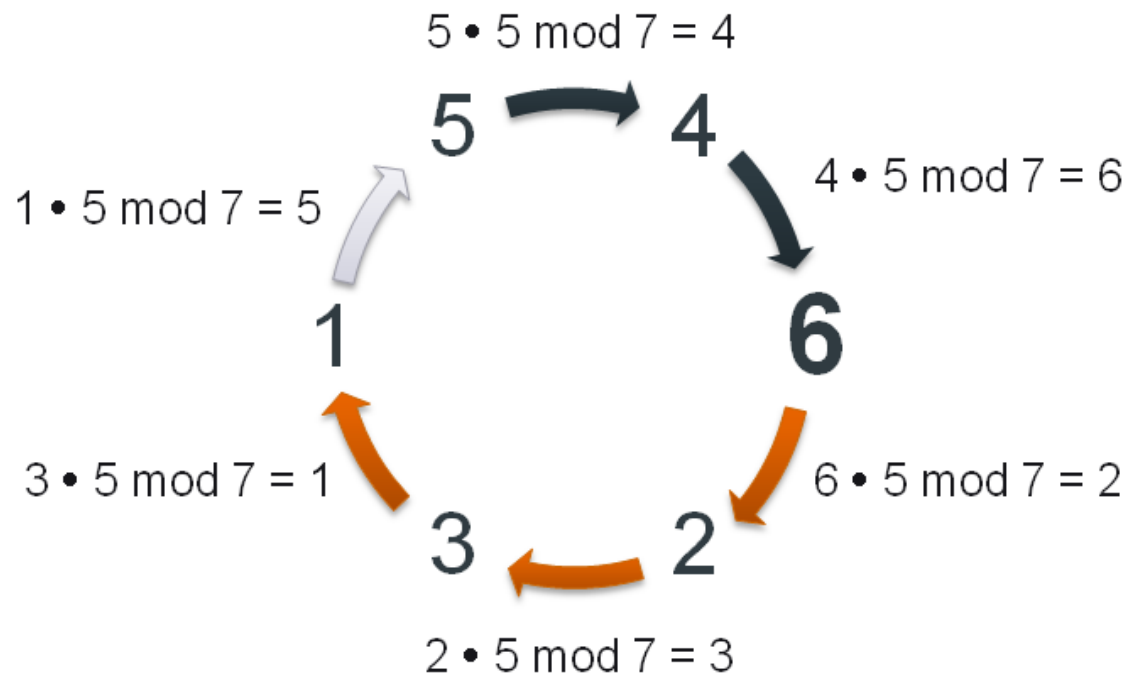
- Bypass inefficiencies by generating Ethernet frames

Addressing Probes

How do we randomly scan addresses without excessive state?

Scan hosts according to random permutation.

Iterate over multiplicative group of integers modulo p .



Negligible State

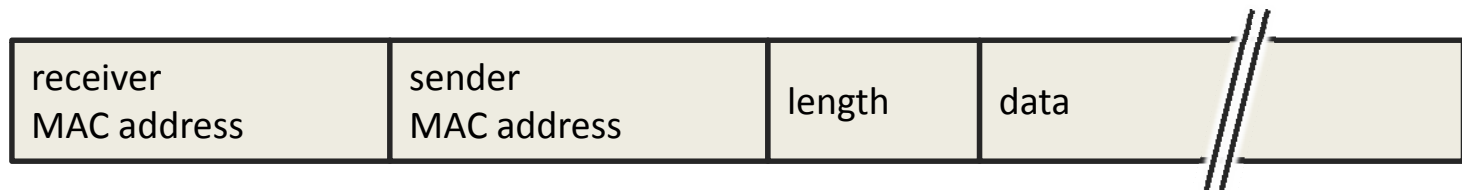
1. Primitive Root
2. Current Location
3. First Address

Validating Responses

How do we validate responses without local per-target state?

Encode secrets into mutable fields of probe packets that will have recognizable effect on responses.

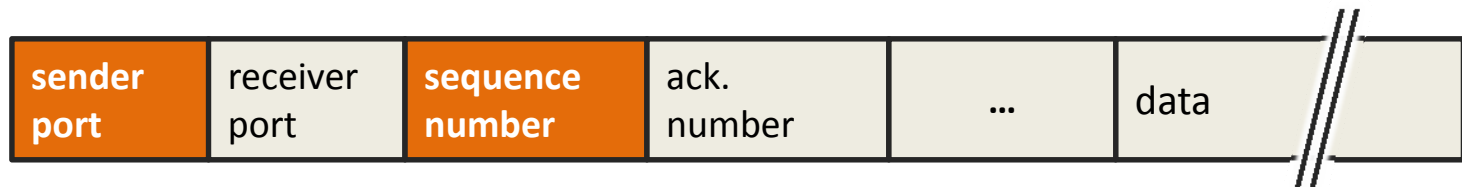
Ethernet



IP



TCP

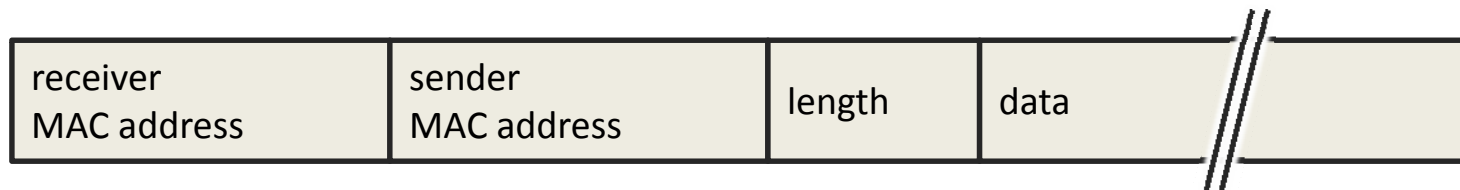


Validating Responses

How do we validate responses without local per-target state?

Encode secrets into mutable fields of probe packets that will have recognizable effect on responses.

Ethernet



IP



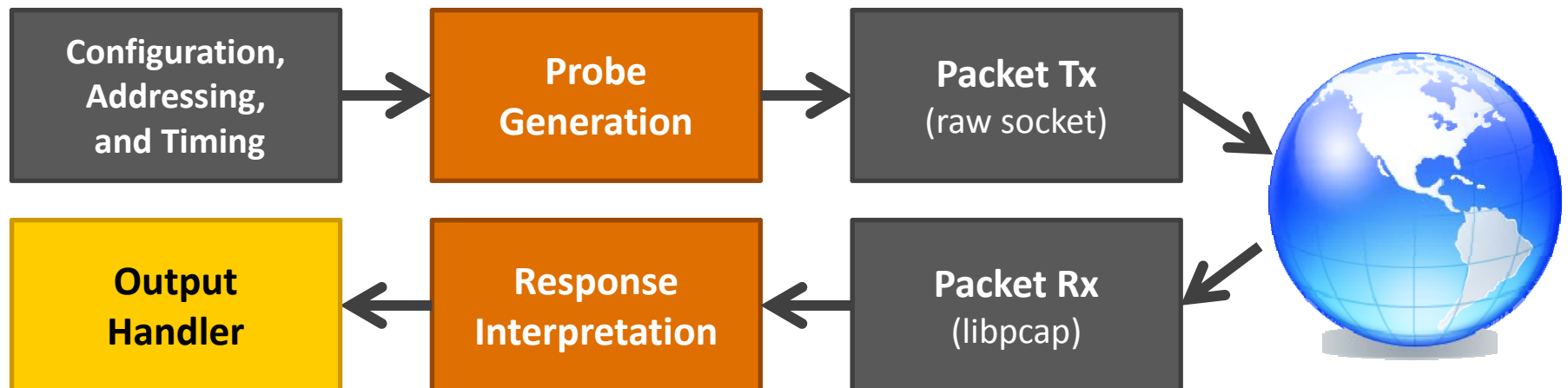
TCP



Packet Transmission and Receipt

How do we make processing probes easy and fast?

1. **ZMap Framework** handles the hard work
2. **Probe Modules** fill in packet details, interpret responses
3. **Output Modules** allow follow-up or further processing

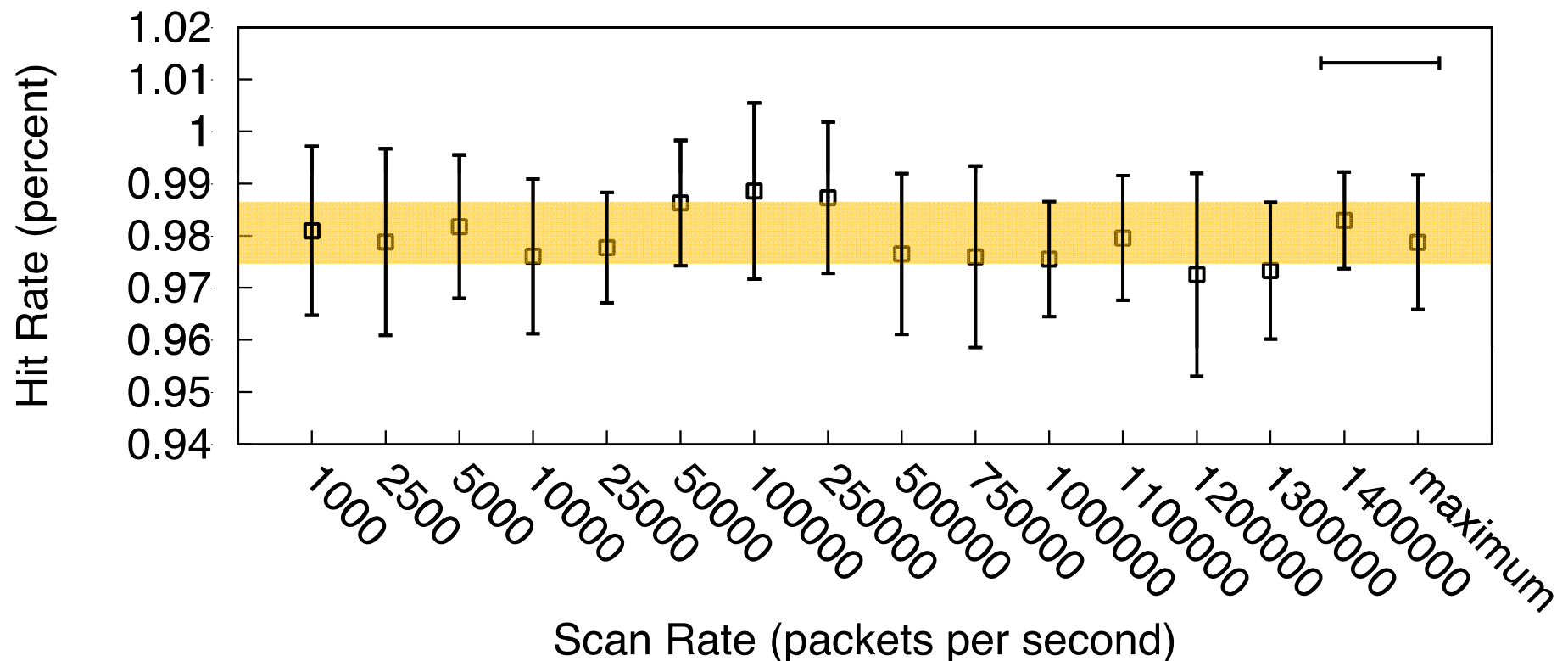


Scan Rate

How fast is too fast?

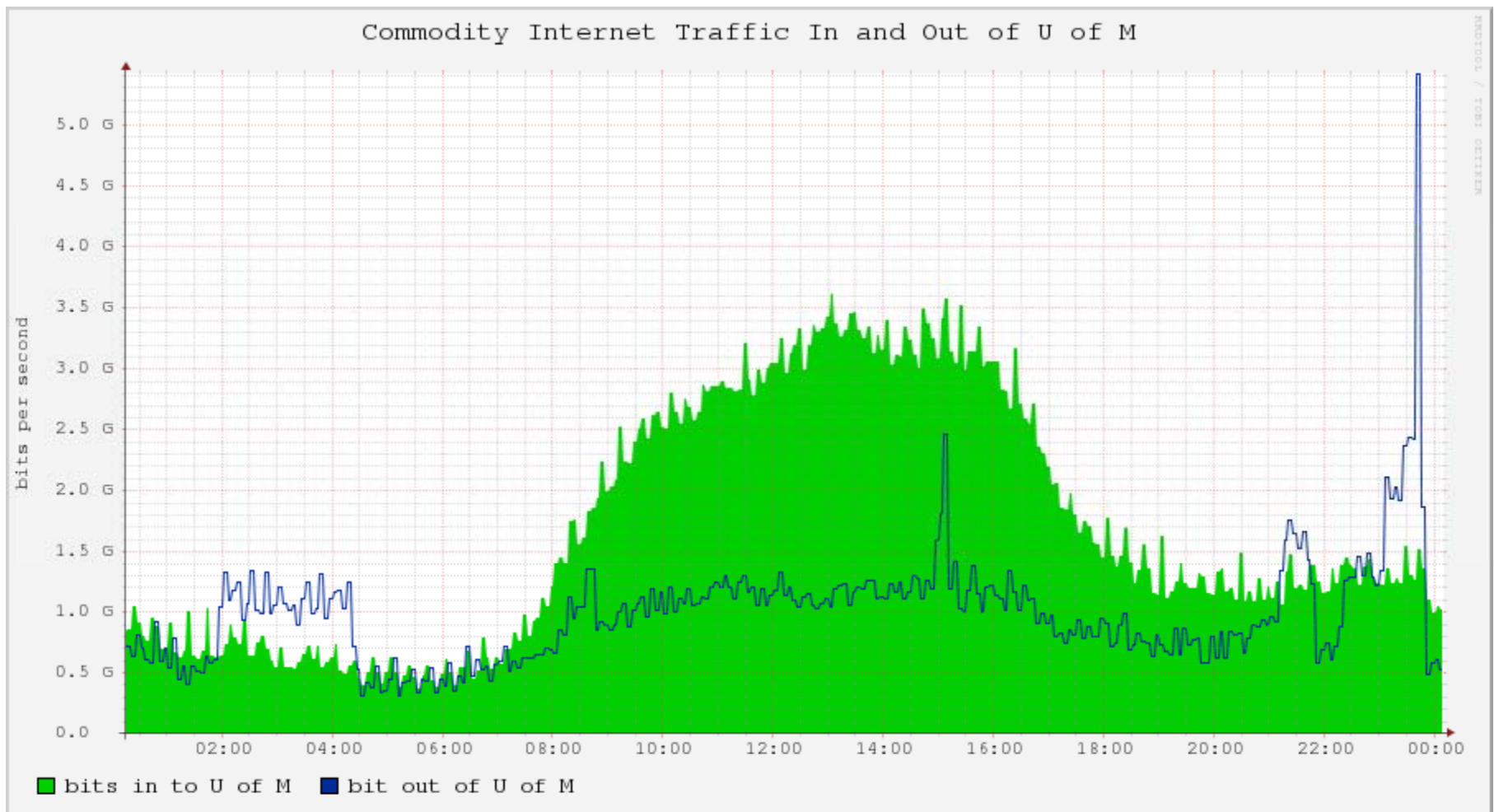
No meaningful correlation between speed and hit rate.
Slower scanning does not reveal additional hosts.

*Your mileage
may vary!*



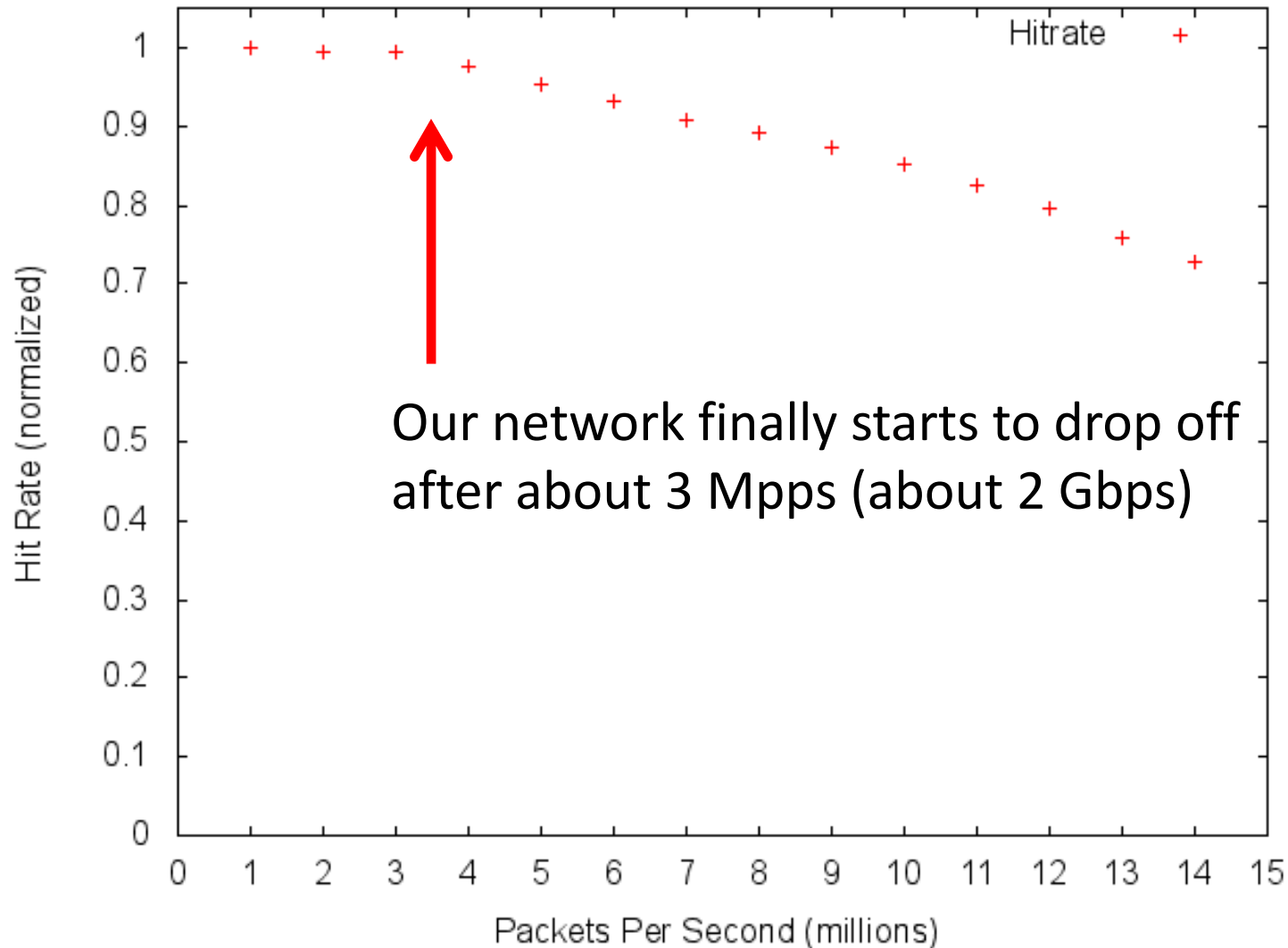
Scan Rate – 10 Gbps?

How fast is too fast?



Scan Rate – 10 Gbps?

How fast is too fast?



ZMap vs. Nmap

Averages for scanning 1 million random hosts:

	Normalized Coverage	Duration (mm:ss)	Est. Internet Wide Scan
Nmap (1 probe)	81.4%	24:12	62.5 days
Nmap (2 probes)	97.8%	45:03	116.3 days
ZMap (1 probe)	98.7%	00:10	1:09:35
ZMap (2 probes)	100.0%	00:11	2:12:35

ZMap can scan more than **1300 times faster** than the most aggressive Nmap default configuration (“insane”)

Surprisingly, ZMap also finds more results than Nmap

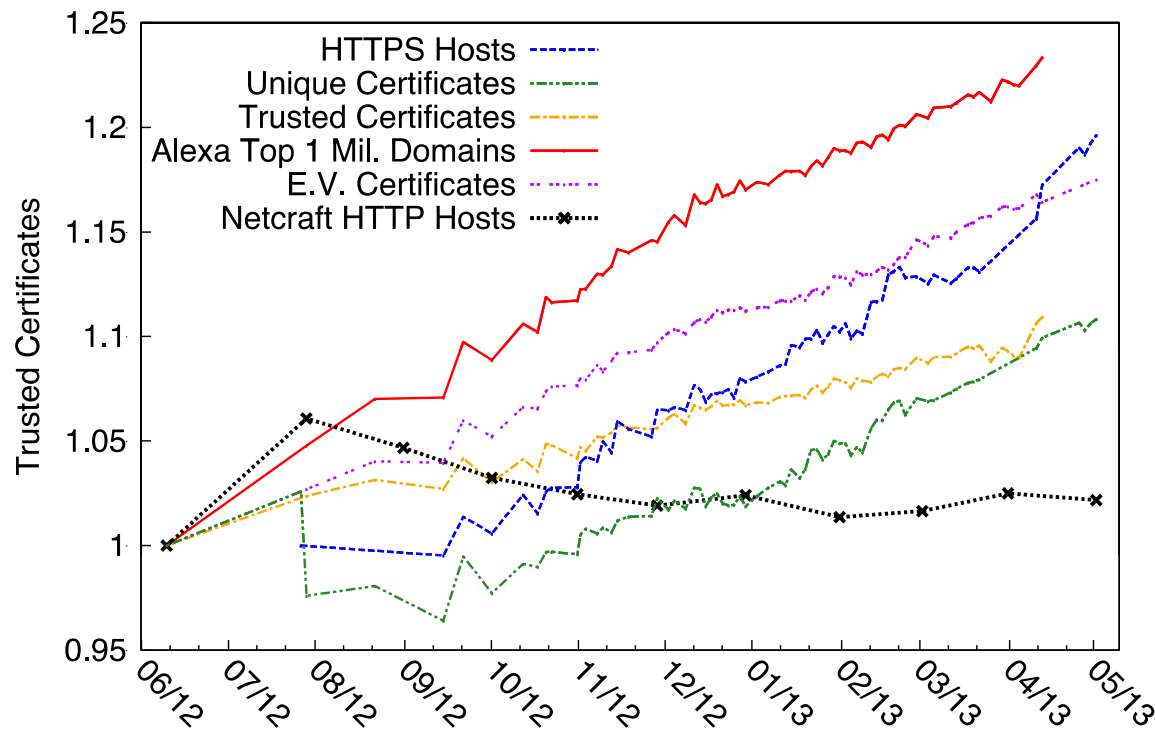
ZMap: Applications

We did > 300 Internet-wide scans over 2 years (> 1 trillion probes).

Please ignore probes from 141.212.121.0/24. It's just our desktop.

What else can researchers do with ZMap?

Track Adoption of Defenses



- Fine-grained analysis of HTTPS ecosystem.
> 100 full scans over a year
- Many vulnerabilities!
- 10% growth in HTTPS sites
23% among Alexa Top 1 M.
- Historical data useful for tracking botnets and APTs.

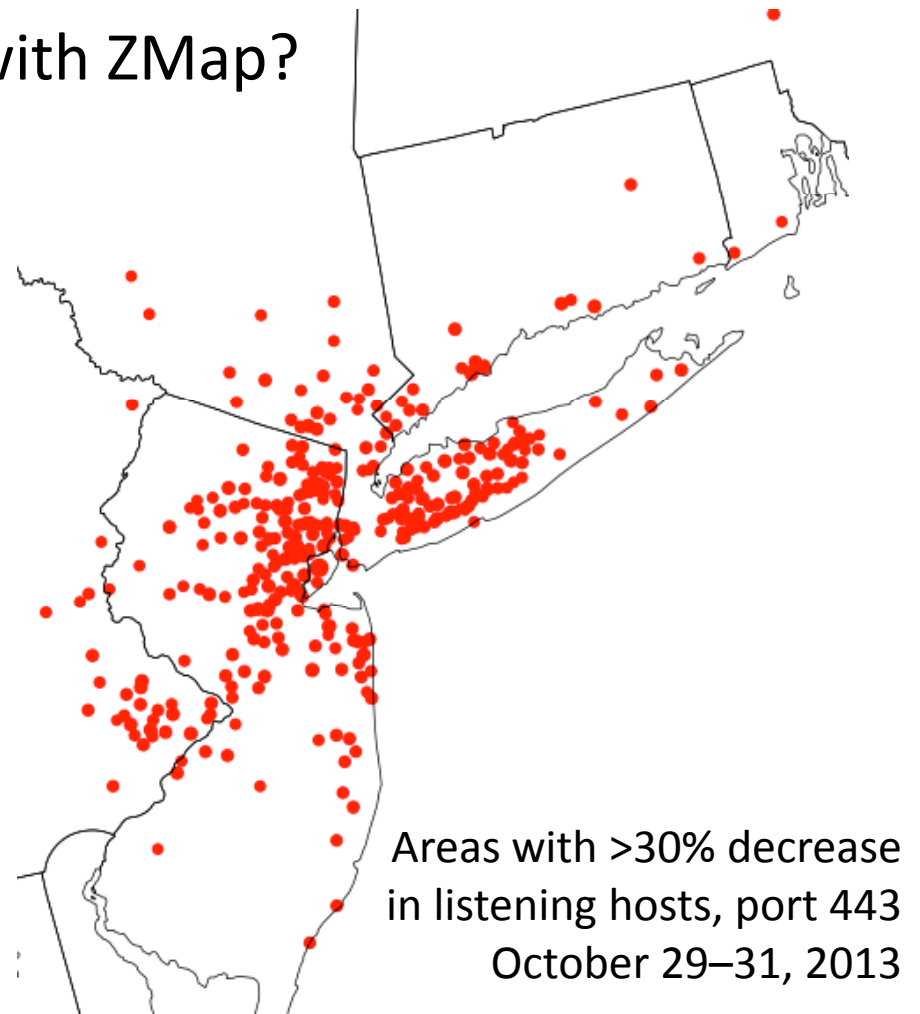
ZMap: Applications

We did > 300 Internet-wide scans over 2 years (> 1 trillion probes).

Please ignore probes from 141.212.121.0/24. It's just our desktop.

What else can researchers do with ZMap?

Detect Service Disruptions



ZMap: Applications

We did > 300 Internet-wide scans over 2 years (> 1 trillion probes).

Please ignore probes from 141.212.121.0/24. It's just our desktop.

What else can researchers do with ZMap?

Expose Vulnerable Hosts

- Took < 4 hours to code and run UPnP discovery scan, 150 SLOC.
- Found 3.34 million devices vulnerable to HD Moore's attacks.
- Compromise possible with a single UDP packet!



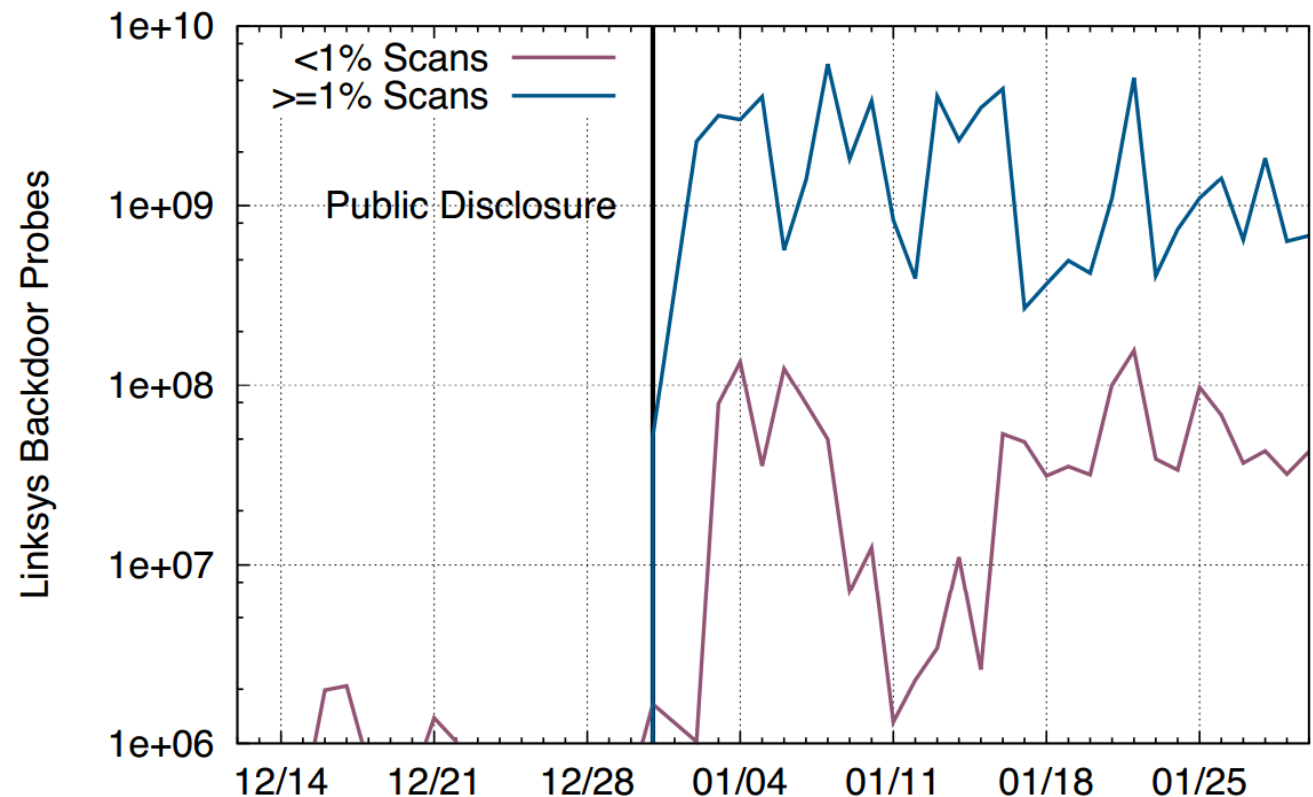
Tracking the Scanners

Data from large darknet allow us to see scans as they happen.

Both [researchers](#) and [attackers](#) using scanning to spot vulnerable hosts.



Backdoor on
TCP/32764
December 2013



43 large scans, starting within 24 hours:

Shodan, Rapid7, academics, bullet-proof hosting

Broken Cryptographic Keys

Why are a large fraction of hosts sharing cryptographic keys?

> 60% of hosts served non-unique public keys.

	Port 443 (HTTPS)	Port 22 (SSH)
Live Hosts	12.8 M	10.2 M
Distinct RSA public keys	5.6 M	3.8 M
Distinct DSA public keys	6,241	2.8 M

Many valid (and common) reasons to share keys:

- ▶ Shared hosting situations. Virtual hosting.
- ▶ A single organization registers many domain names with the same key.

Broken Cryptographic Keys

Why are a large fraction of hosts sharing cryptographic keys?

> 60% of hosts served non-unique public keys.

Common (and unwise) reasons to share keys:

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

TLS:

default certificates/keys:

670,000 hosts (5%)

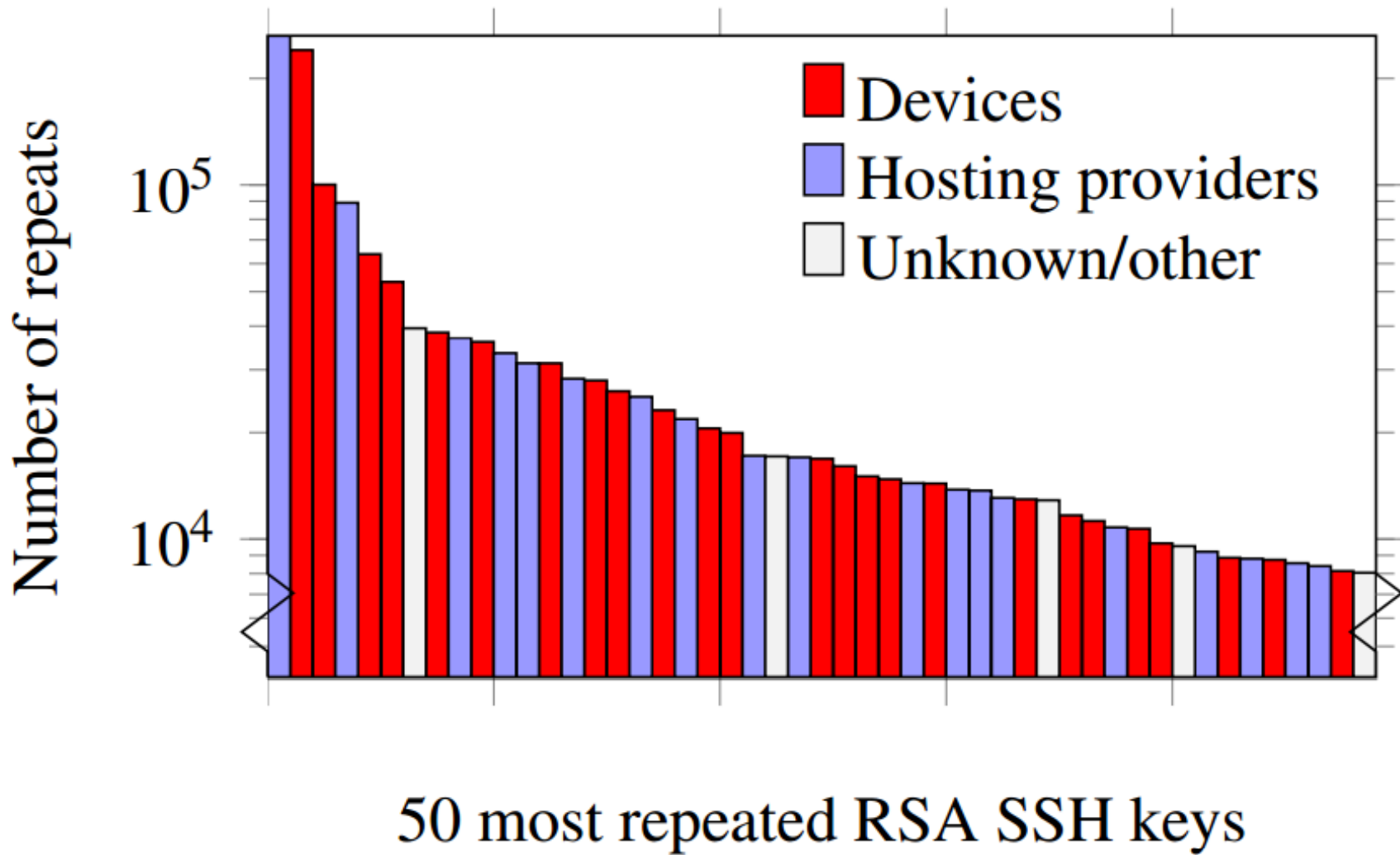
low-entropy repeated keys:

40,000 hosts (0.3%)

SSH:

default or low-entropy keys:

1,000,000 hosts (10%)



Factorable Cryptographic Keys

Public key modulus $N = pq$. Factoring N reveals the private key.
Factoring 1024-bit RSA not known to be feasible.

However... if two RSA moduli share a prime factor in common,

$$N_1 = pq_1 \quad N_2 = pq_2$$

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

Outside observer can factor both with GCD algorithm.

Security Implications:

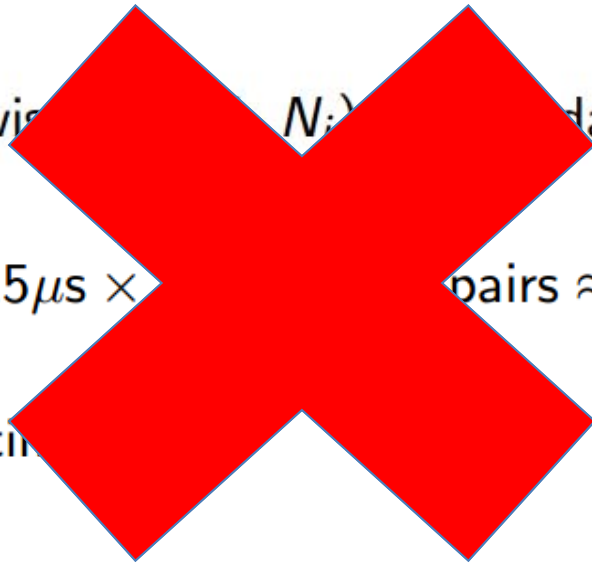
- ▶ Anyone could man-in-the-middle vulnerable hosts.
- ▶ Anyone can decrypt traffic from TLS RSA key exchange.

Computing pairwise GCDs

Computing pairwise GCDs on a dataset would take

$$15\mu\text{s} \times \text{pairs} \approx 30 \text{ years}$$

of computation time

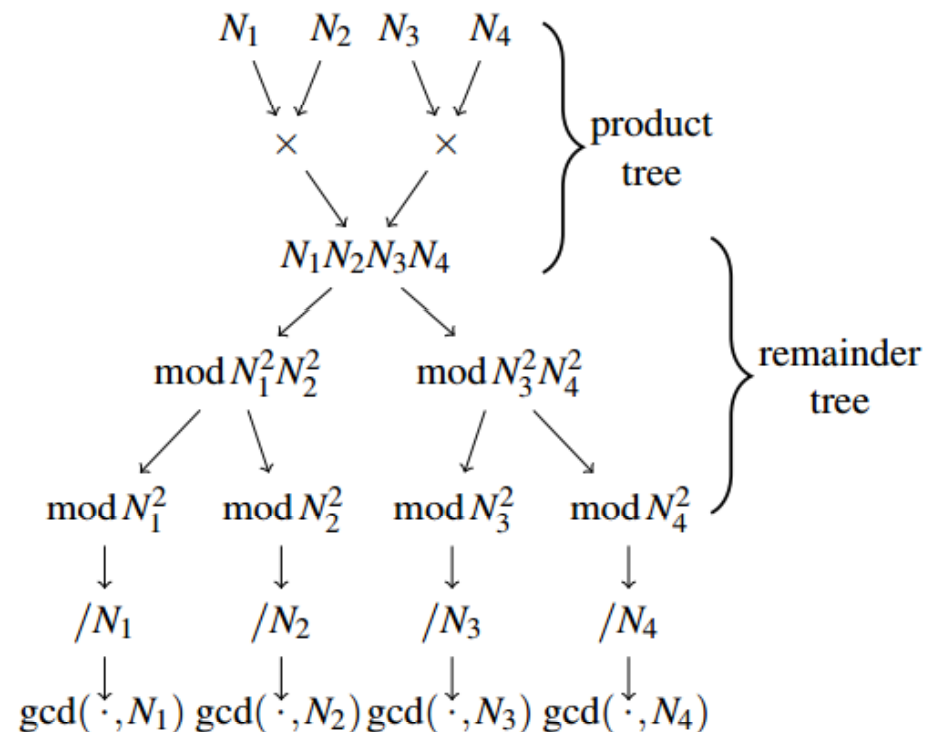


Efficiently computing pairwise GCDs

Adapted efficient algorithm due to [Bernstein 2004].

Ran on 11,170,883 distinct moduli in SSL, SSH, and EFF Observatory datasets

- ▶ 1.5 hours on 16 cores.
- ▶ \$5 of Amazon ec2 time.



What happens when we GCD all the keys?

- ▶ 2,314 distinct prime factors factored 16,717 moduli.
- ▶ Private keys for 64,081 (0.50%) of HTTPS servers.
- ▶ Private keys for 2,459 (0.03%) of SSH servers.

Private keys for 0.5% of all TLS hosts!? 1% of SSH hosts!?

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

Look at 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/emailAddress
C=CN, ST=Guangdong, O=TP-LINK Technologies CO., LTD., OU=TP-LINK SOFT, CN=TL-R478+139819C30089/emailAddress
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/emailAddress
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
---
```

Why do we find vulnerable keys?

Evidence strongly suggested *widespread implementation problems*.

Clue #1: Vast majority 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



Linux random number generators

`/dev/random`

“high-quality” randomness

blocks if insufficient entropy
available

`/dev/urandom`

pseudorandomness

never blocks

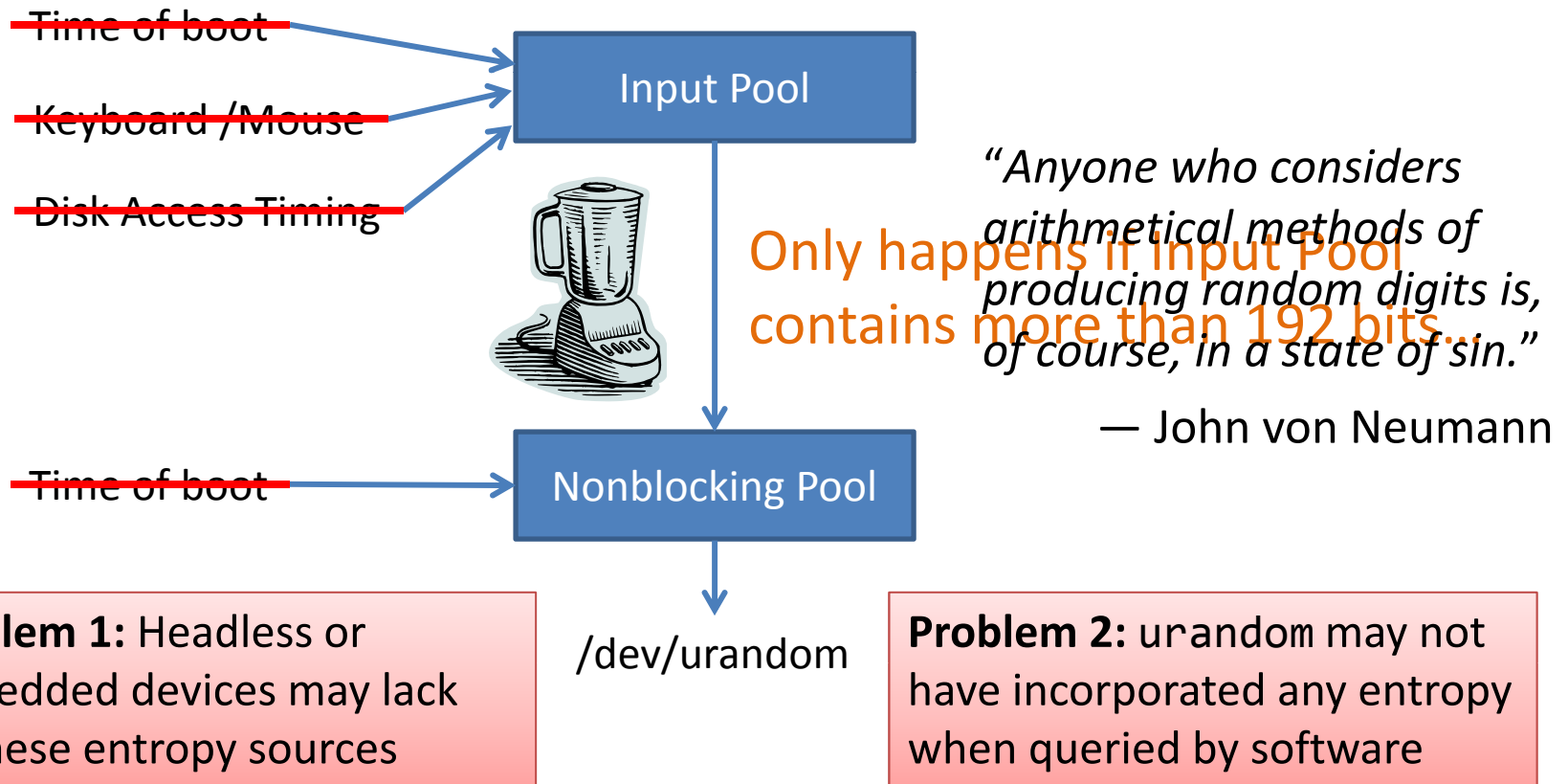
As a general rule, `/dev/urandom` should be used for everything except long-lived GPG/SSL/SSH keys.—`man random`


```
/* We'll use /dev/urandom by default,  
since /dev/random is too much hassle. If  
system developers aren't keeping seeds  
between boots nor getting any entropy from  
somewhere it's their own fault. */  
#define DROPBEAR_RANDOM_DEV "/dev/urandom"
```

random's conservative blocking behavior is a usability problem.
This results in many developers using urandom for cryptography.

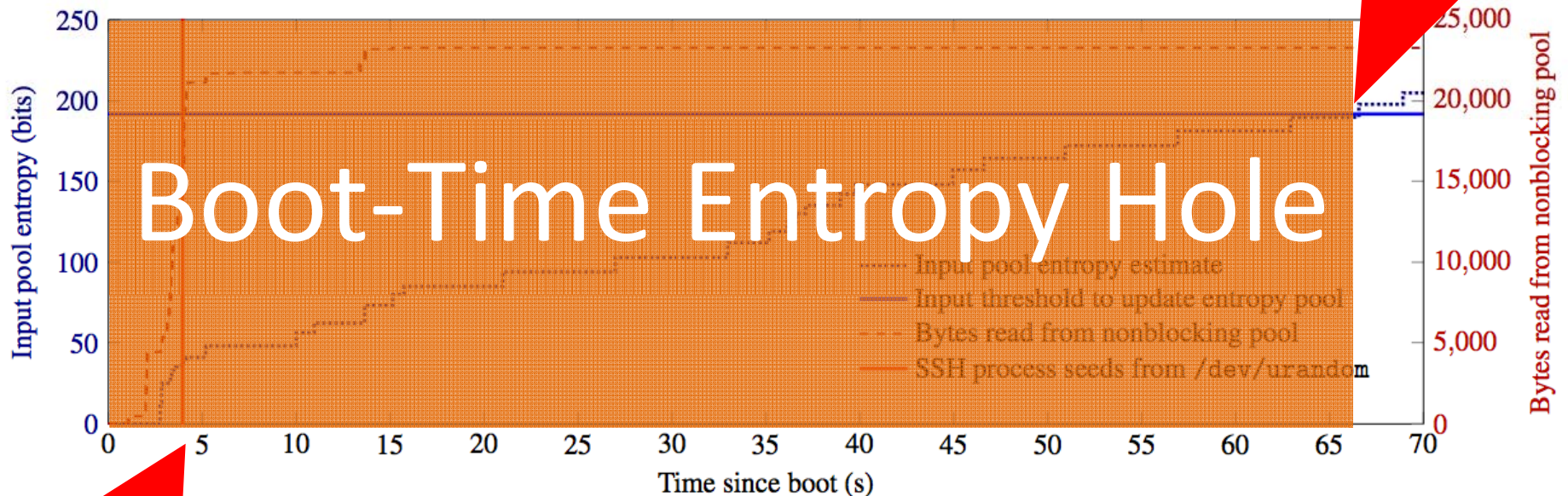
Inside Linux /dev/urandom

Hypothesis: Devices are using /dev/urandom to automatically generate crypto keys on first boot.



Detected Problem in Linux Kernel

Why are embedded systems generating broken keys?



OpenSSH seeds
from /dev/urandom

/dev/urandom may be predictable
for a period after boot.

Responsible Disclosure

Wrote disclosures to **61 companies**.

Coordinated through US-CERT, ICS-CERT, JP-CERT.

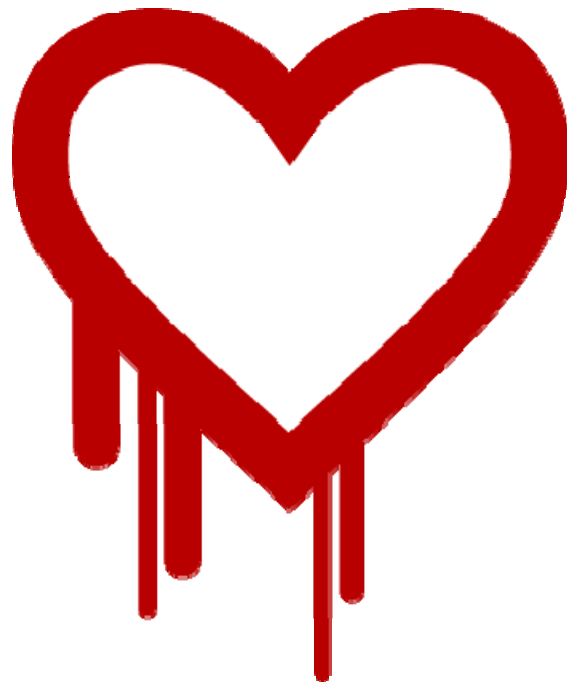
13 had Security Response Team contact information available.

28 sent us a response from a human.

13 told us they fixed the problem.

5 informed us of security advisories.

Linux kernel has been patched.



TLS Heartbeats

TLS Heartbeat Request:



TLS Heartbeat Response:



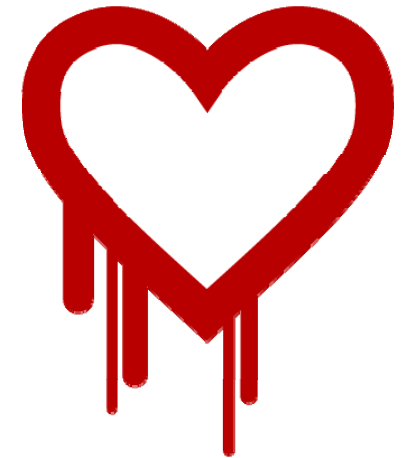
(Based on joint work with Zakir Durumeric, James Kasten, J. Alex Halderman, Michael Bailey, Frank Li, Nicholas Weaver, Bernhard Amann, Jethro Beekman, Mathias Payer, and Vern Paxson.)

OpenSSL Heartbleed Vulnerability

Malformed TLS Heartbeat Request:



Vulnerable OpenSSL Heartbeat Response:



Discovered March 2014

Publicly disclosed April 7, 2014

Detecting Heartbleed Hosts



How can we detect vulnerable hosts without exploiting them?

TLS Heartbeat Request:



type *length 0 (invalid per RFC)*

Vulnerable OpenSSL Response:



type *length* *padding (16 bytes)*

Patched OpenSSL Response:

Error

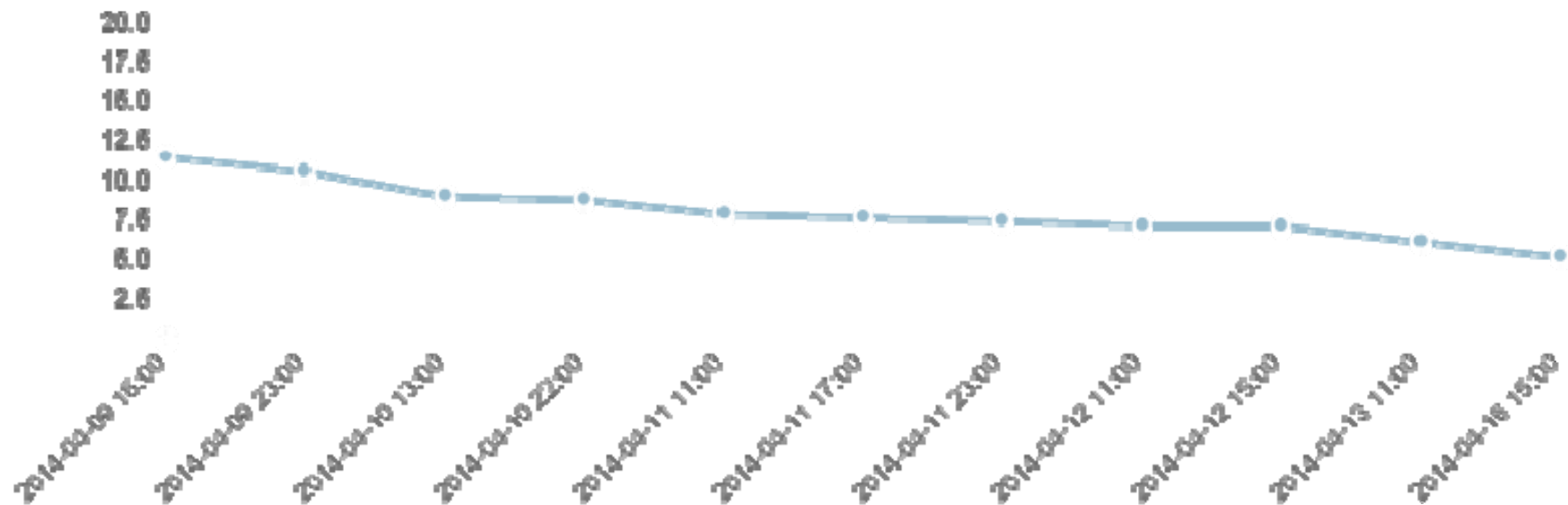
Patching Observations



Track patching used large-scale scans

Prior to disclosure, problem affected ~27% of Alexa Top 1M

Most quickly patched. The rest?



Public Health Report

<https://zmap.io/heartbleed>



Heartbleed Bug Health Report

The Heartbleed Bug is a vulnerability in the OpenSSL cryptographic library that allows attackers to invisibly read sensitive data from a web server. This potentially includes cryptographic keys, usernames, and passwords. More information and frequently asked questions can be found in the initial disclosure. Information on popular websites that were impacted, but are no longer vulnerable can be found on Mashable's [The Heartbleed Hit List: The Passwords You Need to Change Right Now](#). If you are concerned that a specific website is vulnerable, you can test that website using the [Qualys SSL Server Test](#). If you are a Systems Administrator, the EFF has published [Heartbleed Recovery for System Administrators](#) with information on how to protect services.

Most Popular Vulnerable Domains

Below, we list the top 1,000 most popular web domains and mail servers that remain vulnerable to the heartbleed vulnerability as of 4:00 PM EDT on April 16, 2014. More comprehensive lists of vulnerable [web servers](#) and [mail servers](#) are also available.

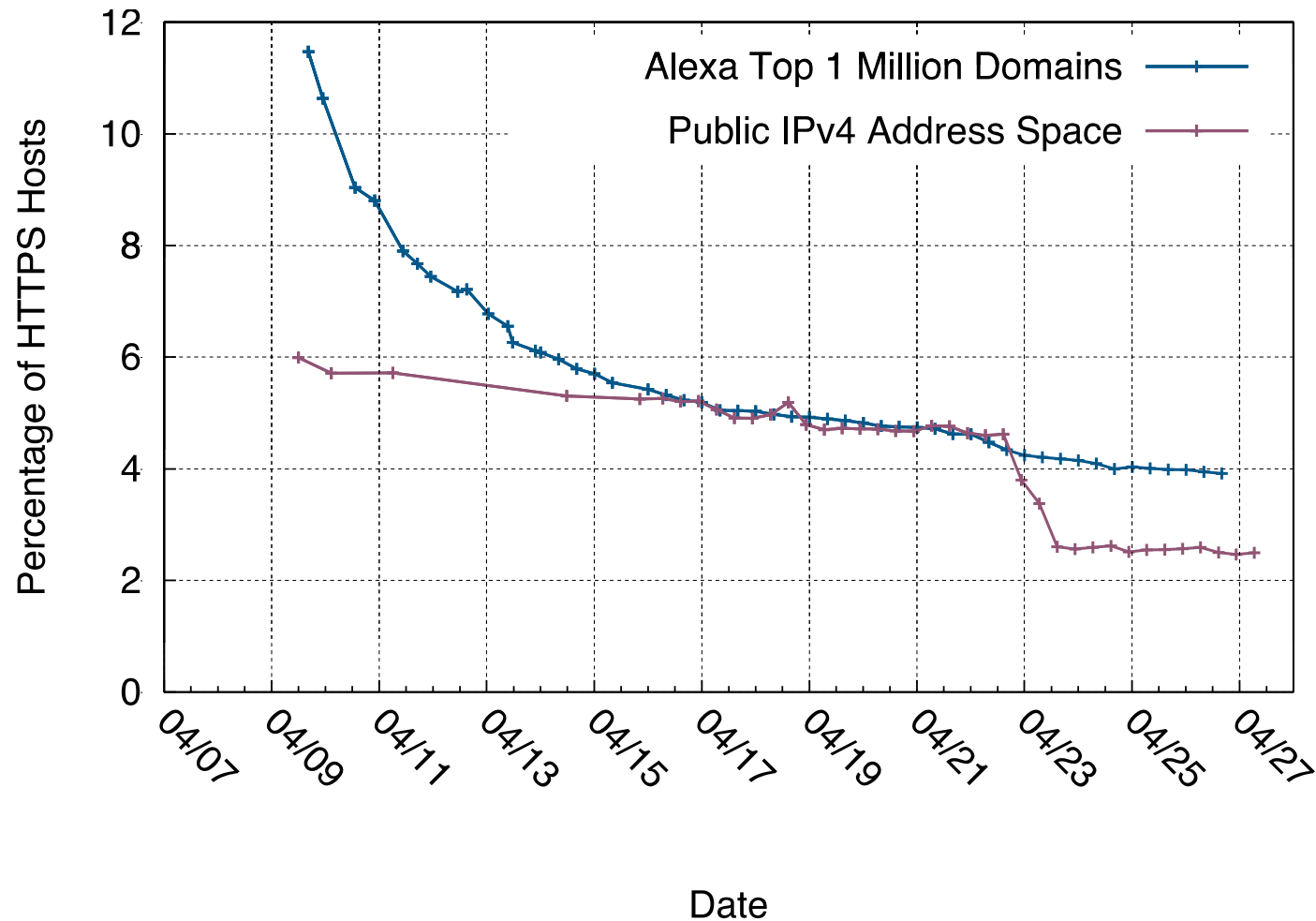
Web Servers

Rank	Domain	Vulnerable
1829	gi-akademie.com	vulnerable
1863	prezentacya.ru	vulnerable
1873	wallstcheatsheet.com	vulnerable
1907	semalt.com	vulnerable
2700	gazzetta.gr	vulnerable
3159	protothema.gr	vulnerable
3428	text.ru	vulnerable
3451	haodf.com	vulnerable

Mail Servers

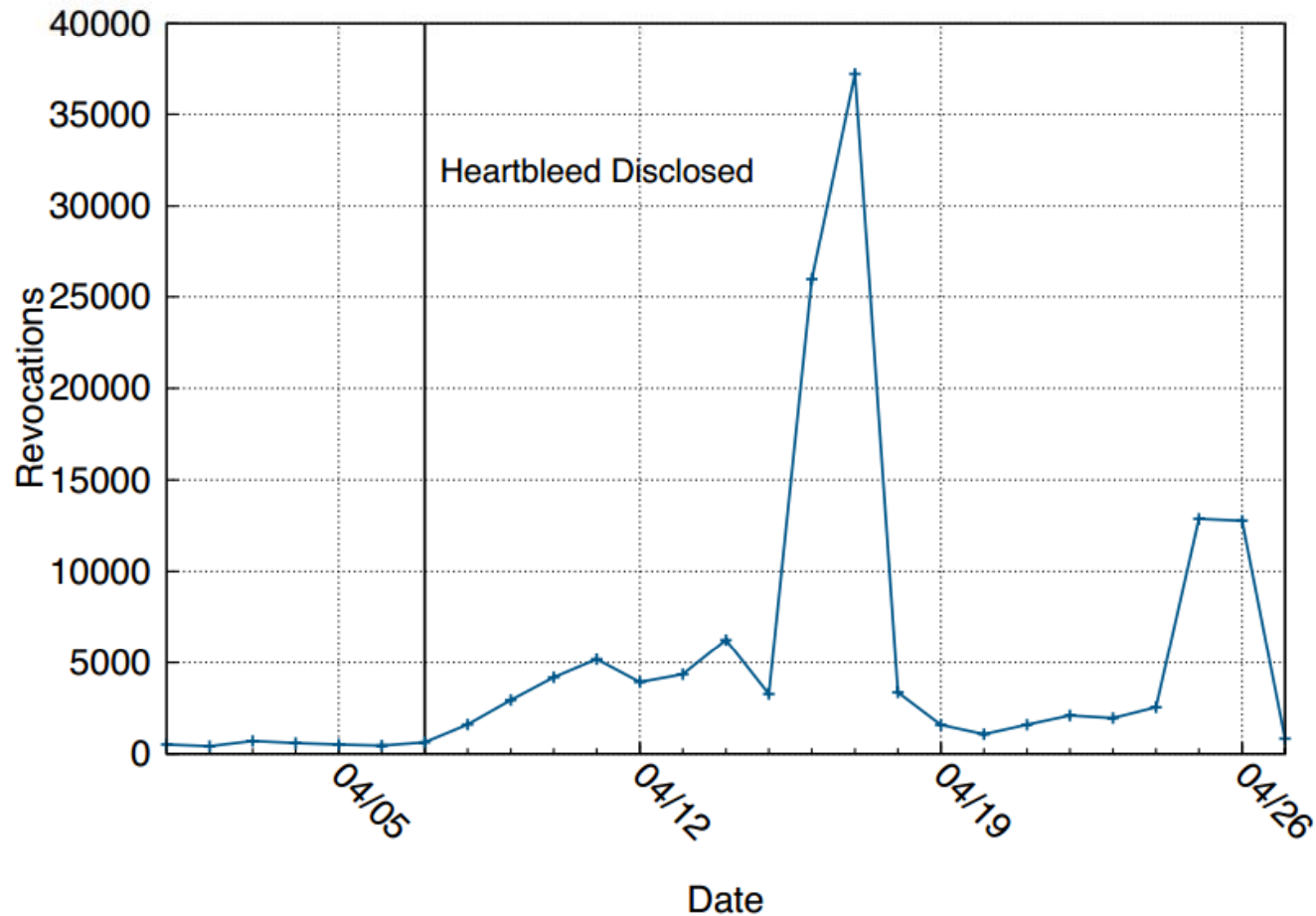
Rank	Domain	Vulnerable
727	turbobit.net	vulnerable
1700	nmisr.com	vulnerable
2100	boerse.bz	vulnerable
2951	ubi.com	vulnerable
3277	filmifullizle.com	vulnerable
3992	uline.com	vulnerable
4081	elektroda.pl	vulnerable
5186	memecenter.com	vulnerable

Patching Observations



~4% of Top 1M
still vulnerable
20 days later

Revoked TLS Certs



Spikes are
GlobalSign
(revoked 55k certs)
GoDaddy
(revoked 22.5k)

Revoked TLS Certs



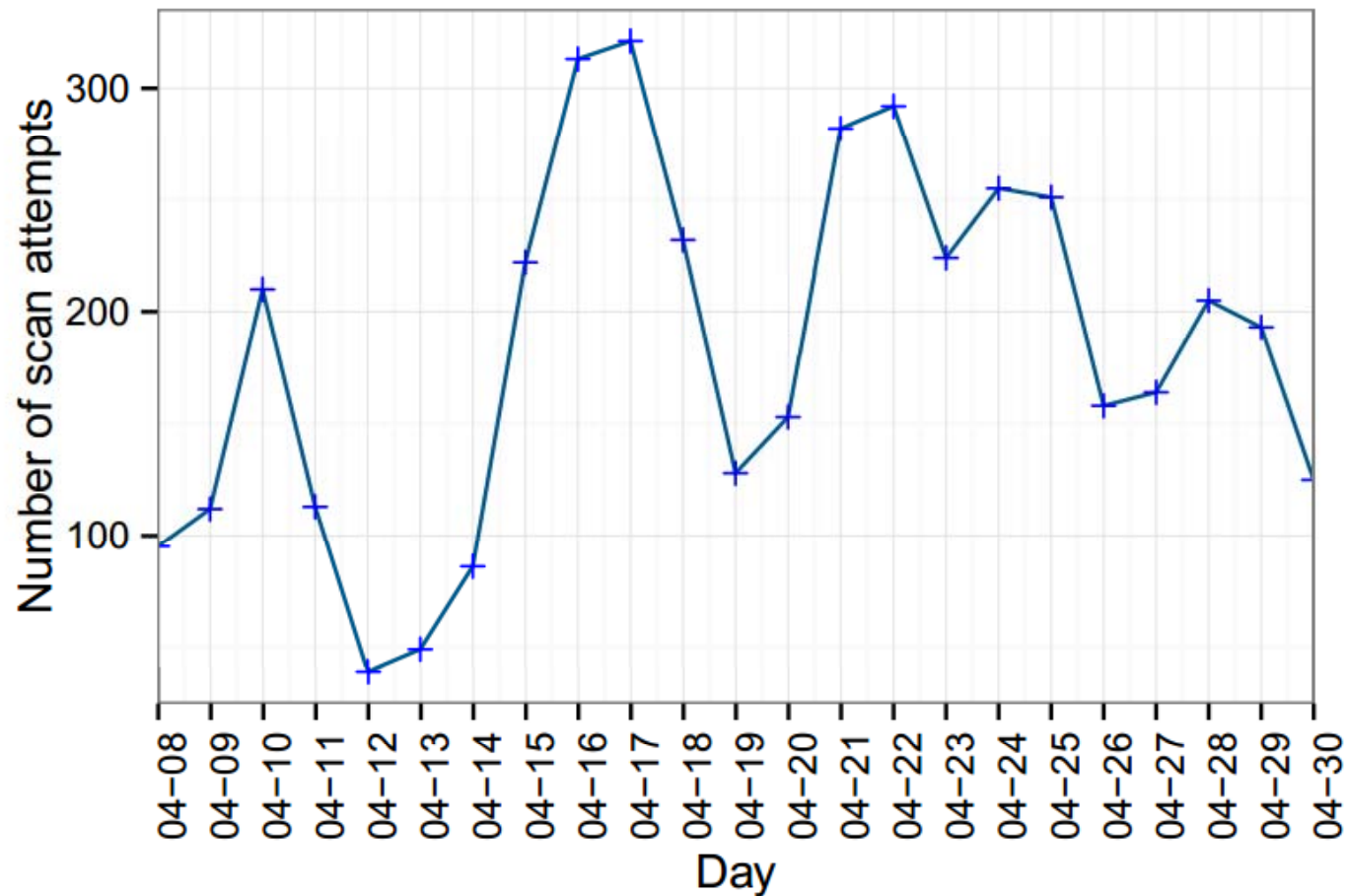
23% of Alex Top 1M replaced their TLS certs in April!

But... of sites still vulnerable on April 9, only 10% replaced certs.

... of those, only 19% revoked their original certificate.

... and 14% re-used the same private key!

Who Scanned for Heartbleed?



In first week, 41 unique hosts scanning for Heartbleed, 59% from China
The detected probe at 1539 GMT on April 8, 2014

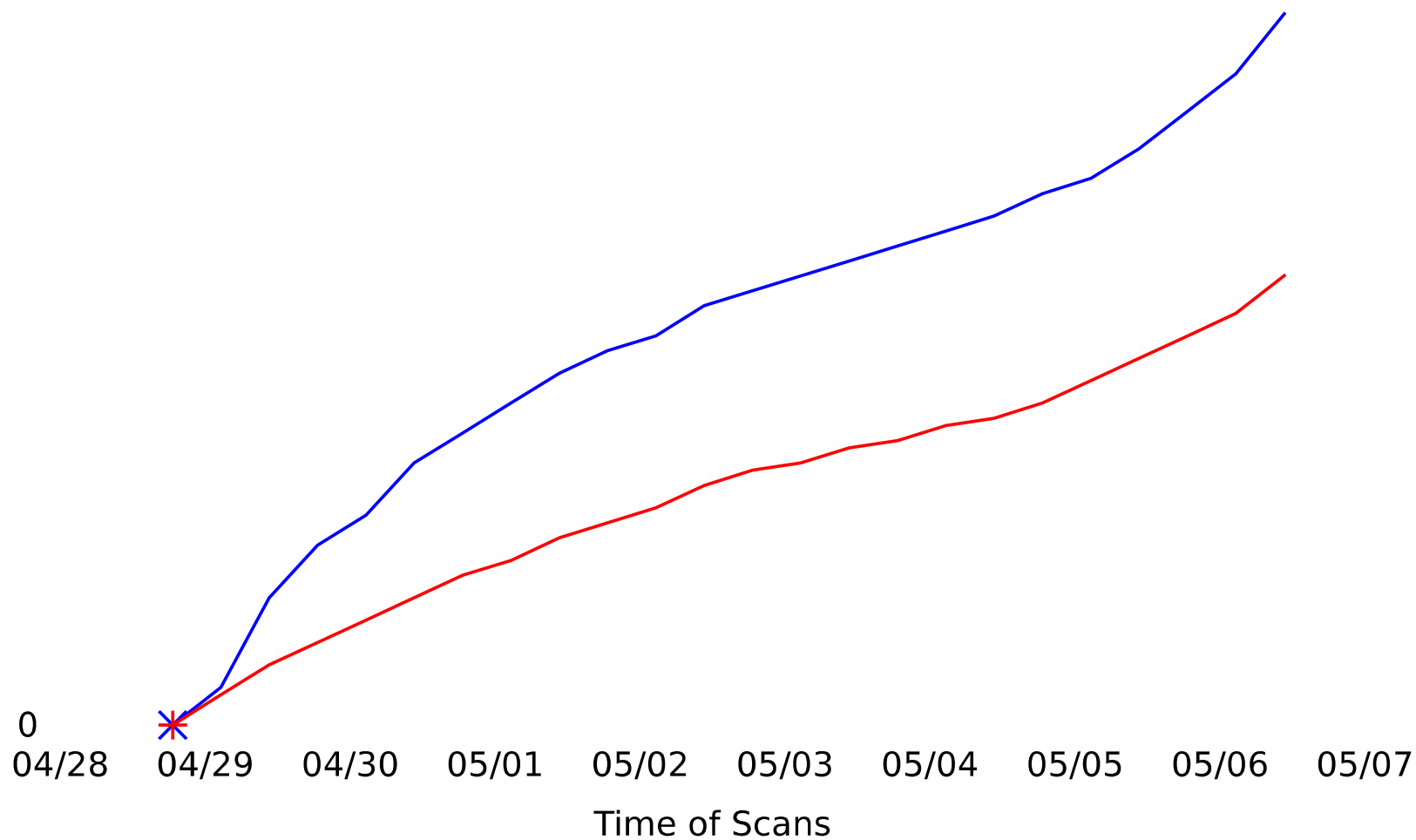
Vulnerability Notifications



April 24 scan discovered 588,000 hosts still vulnerable.

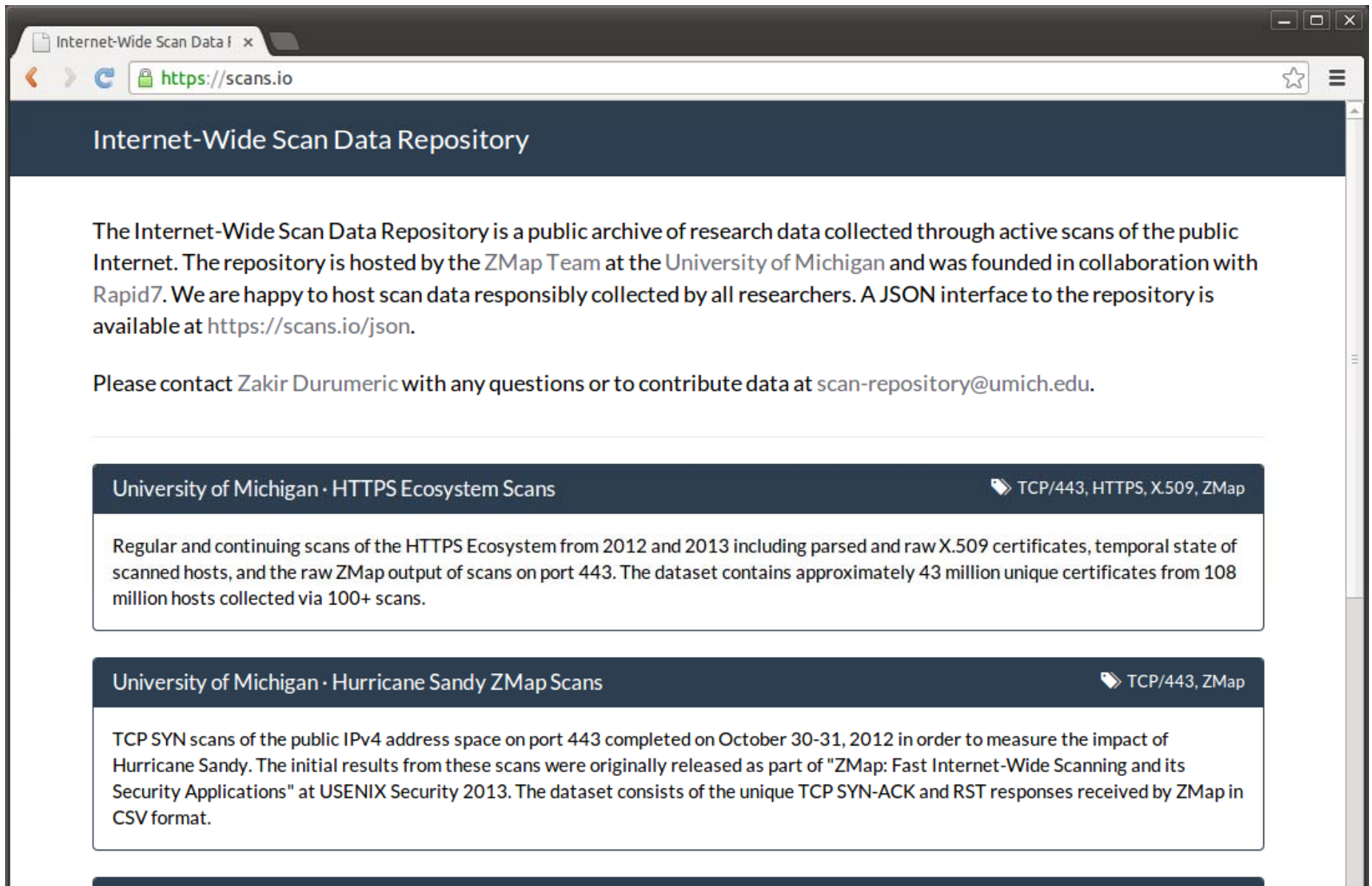
What to do?

Vulnerability Notifications



Conclusions

Scans.io Data Repository



The screenshot shows a web browser window with the address bar displaying <https://scans.io>. The page title is "Internet-Wide Scan Data Repository". The main content area contains a paragraph about the repository, a contact information line, and two data scan listings. The first listing is for "University of Michigan · HTTPS Ecosystem Scans" with tags "TCP/443, HTTPS, X.509, ZMap". The second listing is for "University of Michigan · Hurricane Sandy ZMap Scans" with tags "TCP/443, ZMap".

Internet-Wide Scan Data f x

<https://scans.io>

Internet-Wide Scan Data Repository

The Internet-Wide Scan Data Repository is a public archive of research data collected through active scans of the public Internet. The repository is hosted by the ZMap Team at the University of Michigan and was founded in collaboration with Rapid7. We are happy to host scan data responsibly collected by all researchers. A JSON interface to the repository is available at <https://scans.io/json>.

Please contact Zakir Durumeric with any questions or to contribute data at scan-repository@umich.edu.

University of Michigan · HTTPS Ecosystem Scans TCP/443, HTTPS, X.509, ZMap

Regular and continuing scans of the HTTPS Ecosystem from 2012 and 2013 including parsed and raw X.509 certificates, temporal state of scanned hosts, and the raw ZMap output of scans on port 443. The dataset contains approximately 43 million unique certificates from 108 million hosts collected via 100+ scans.

University of Michigan · Hurricane Sandy ZMap Scans TCP/443, ZMap

TCP SYN scans of the public IPv4 address space on port 443 completed on October 30-31, 2012 in order to measure the impact of Hurricane Sandy. The initial results from these scans were originally released as part of "ZMap: Fast Internet-Wide Scanning and its Security Applications" at USENIX Security 2013. The dataset consists of the unique TCP SYN-ACK and RST responses received by ZMap in CSV format.

Thank You!

J. Alex Halderman

<https://jhalderm.com>

ZMap Internet-wide scanner

<https://zmap.io>

Scan data repository

<https://scans.io>

Heartbleed reports

<https://zmap.io/heartbleed>