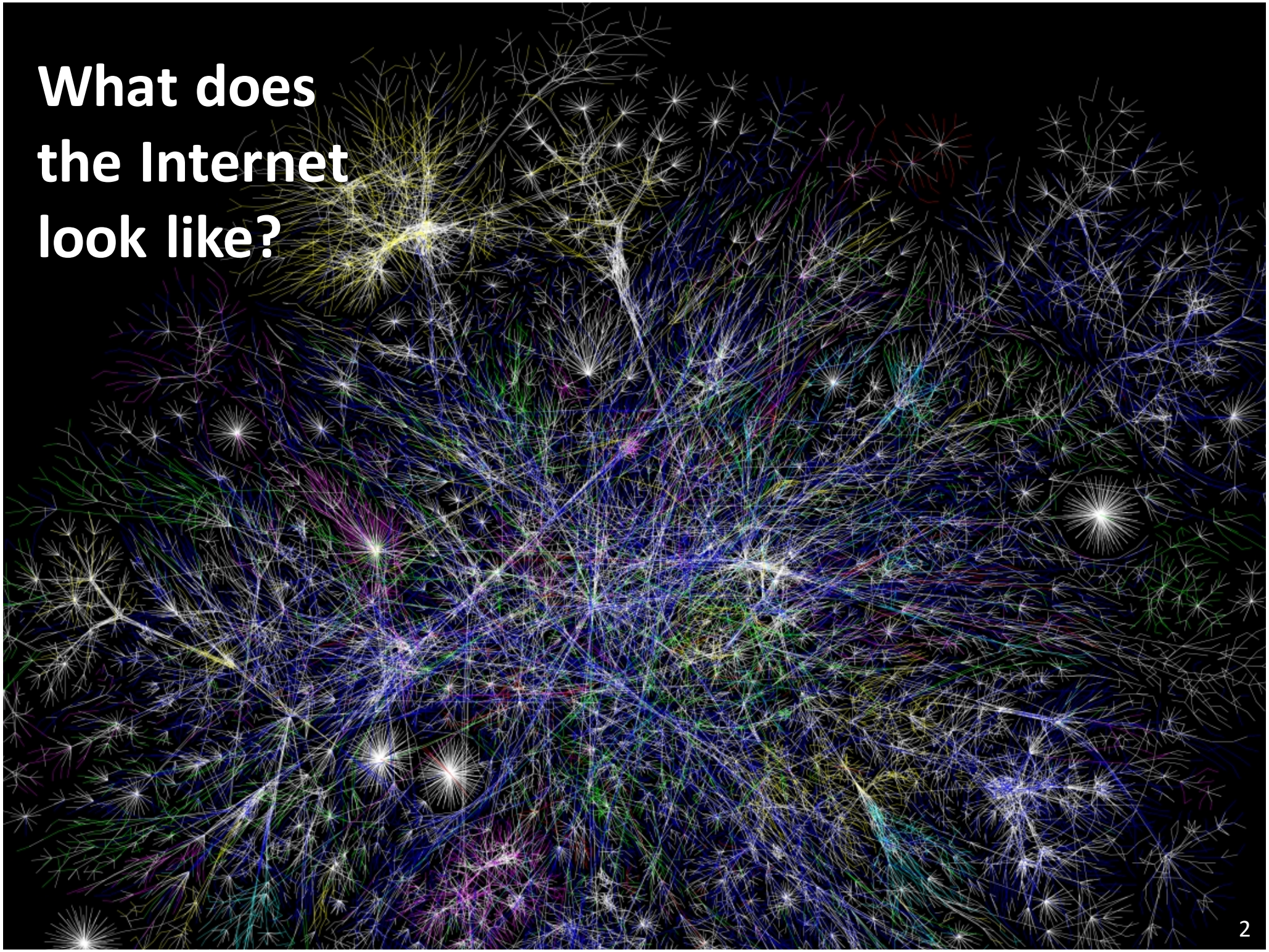
# Network Probing

*How global measurement can improve Internet security*

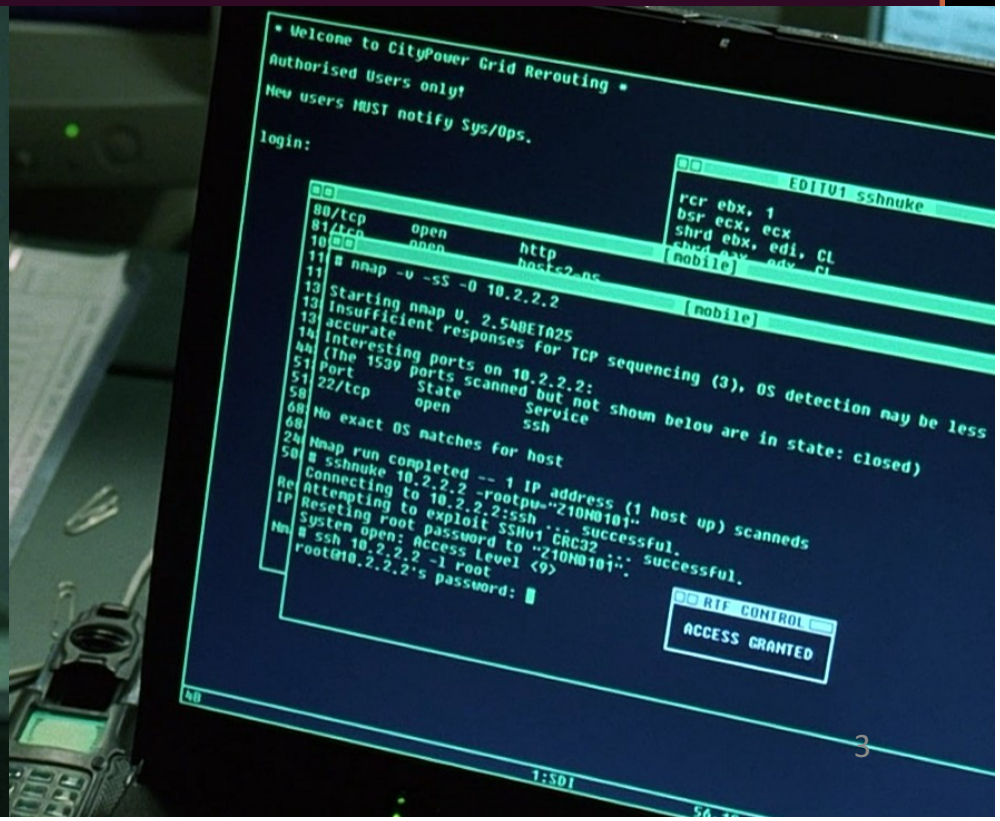
EECS 388: Intro to Security

University of Michigan

# What does the Internet look like?

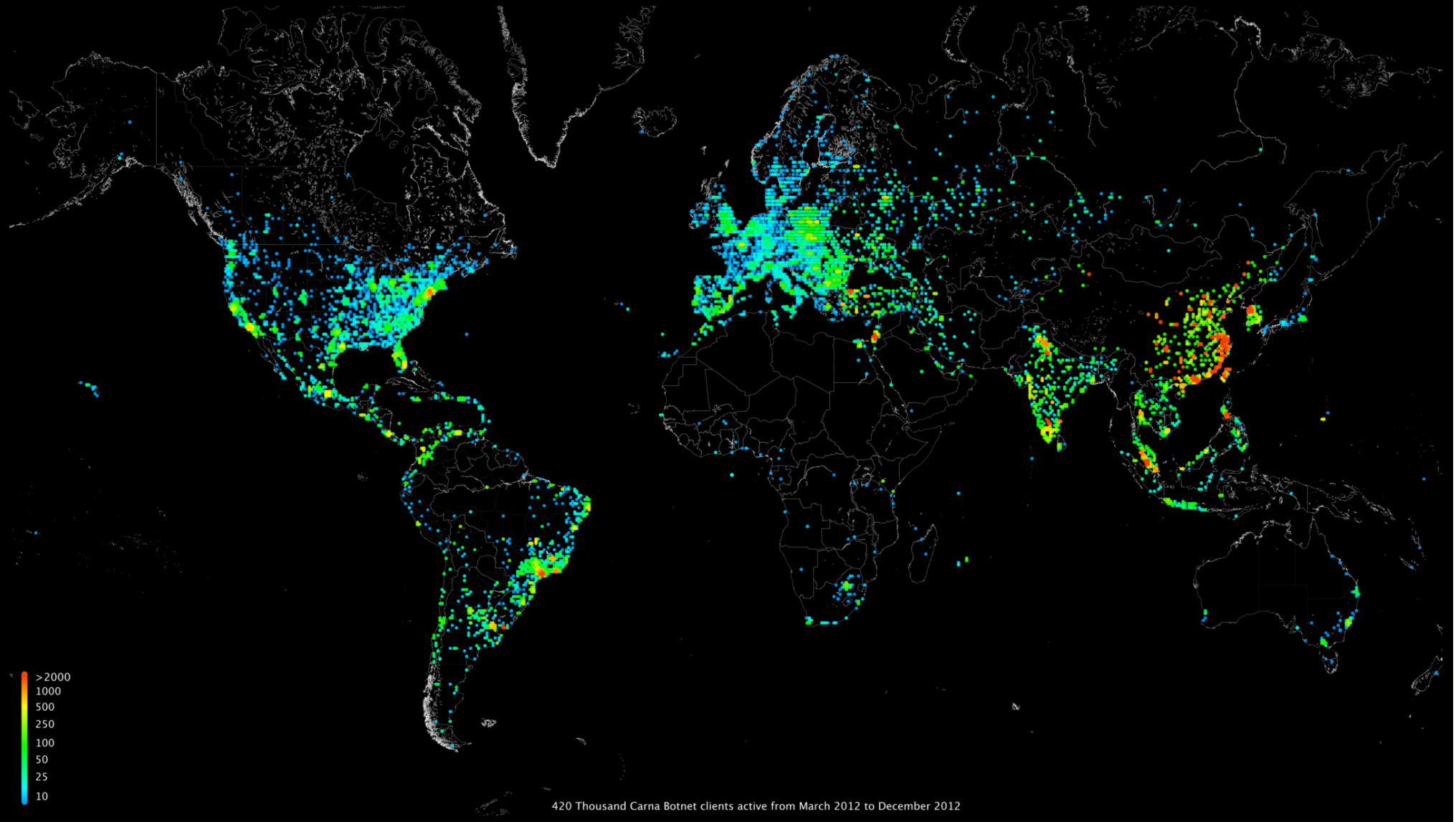



```
jhalderm@elsa-vm: ~
$ nmap 141.212.113.199

Starting Nmap 7.01 ( https://nmap.org ) at 2017-02-21 16:59 EST
Nmap scan report for www.eecs.umich.edu (141.212.113.199)
Host is up (0.00061s latency).
Not shown: 990 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
25/tcp   open  smtp
80/tcp   open  http
111/tcp  open  rpcbind
139/tcp  open  netbios-ssn
443/tcp  open  https
445/tcp  open  microsoft-ds
587/tcp  open  submission
2049/tcp open  nfs
3306/tcp open  mysql

Nmap done: 1 IP address (1 host up) scanned in 0.06 seconds
$
```




Electronic Frontier Foundation
SSL
Observatory

# Carna botnet Internet Census 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 acoss 25 Amazon EC2 Instances (625 CPU-hours)**

Carna botnet Internet Census (2012)

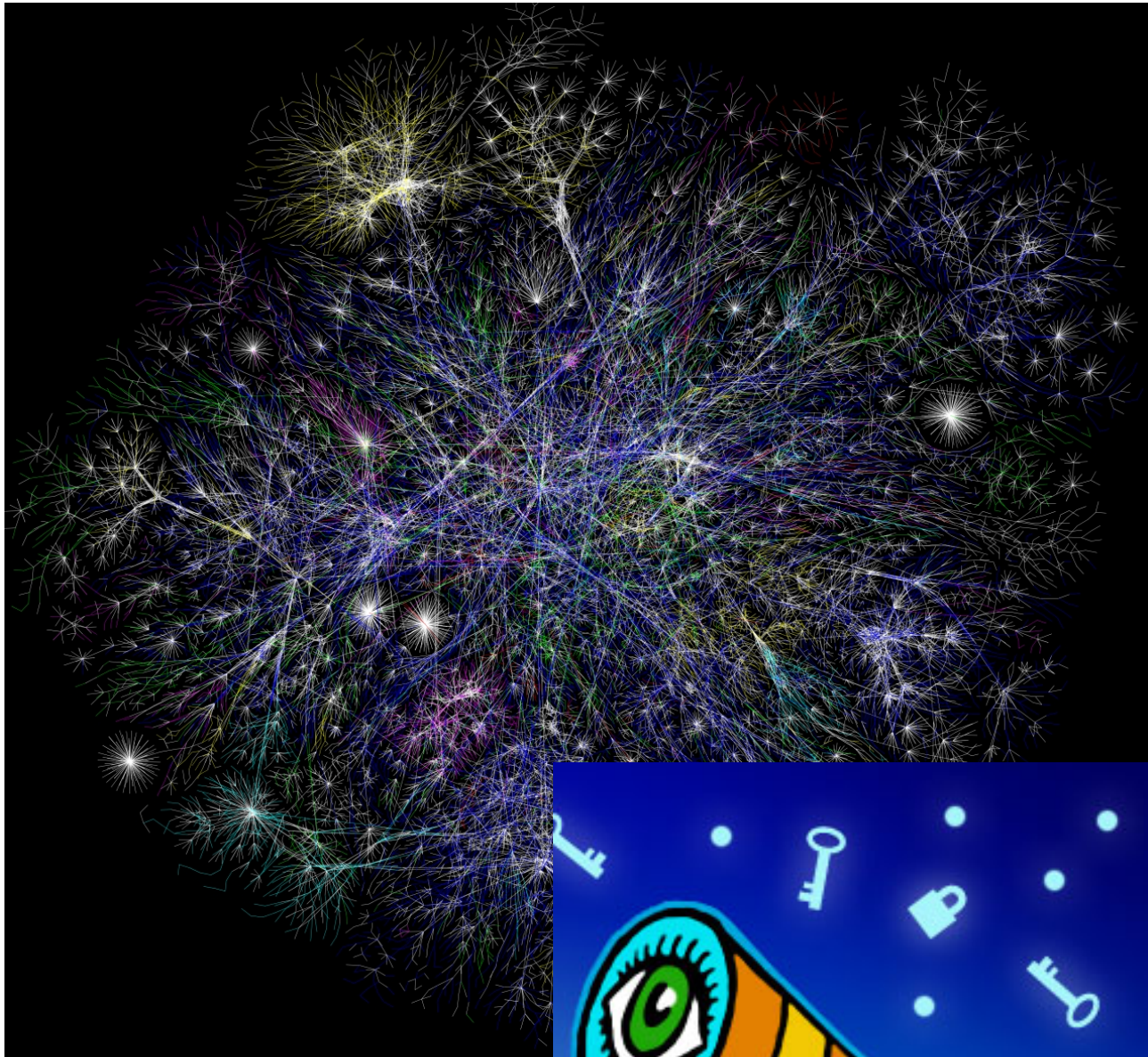
**420,000 usurped hosts**

# 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?**



# zmap

an open-source tool that can port scan the entire IPv4 address space from just one machine in minutes

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)
```

Gigabit Ethernet linespeed
(1200 x NMap)

# ZMap Architecture

## Previous Scanners (e.g. Nmap)

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's 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

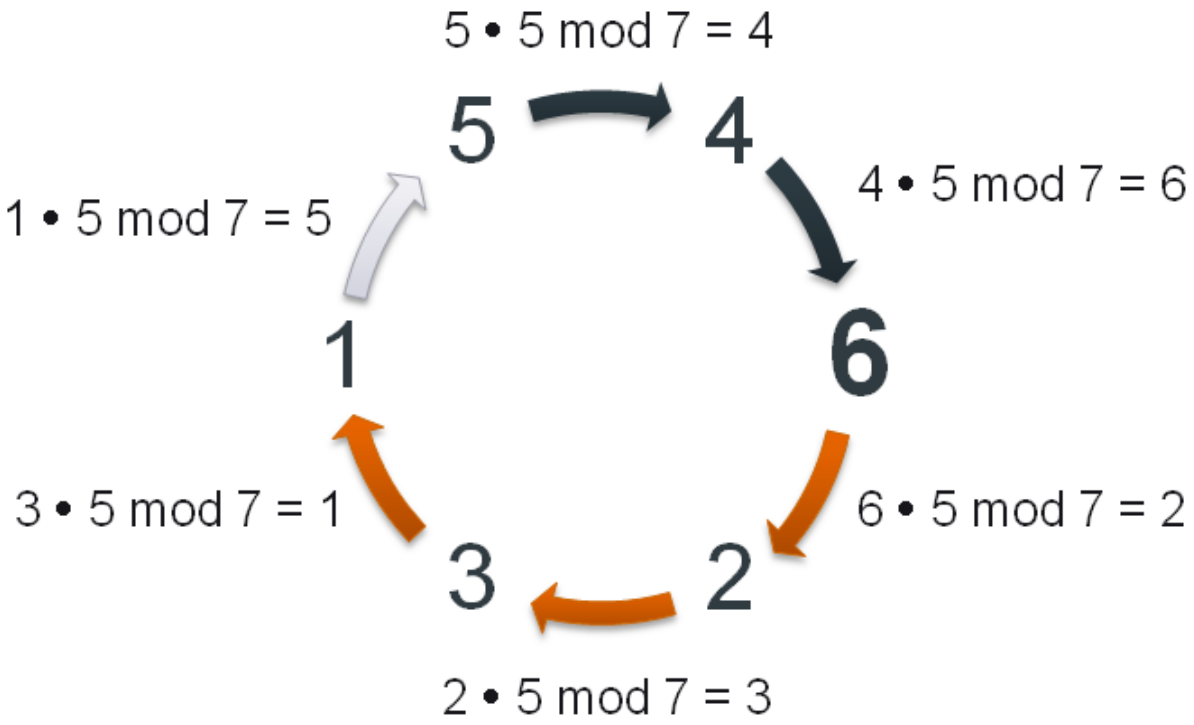
Probe-optimized network stack
- Bypass inefficiencies by generating Ethernet frames

# 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*.



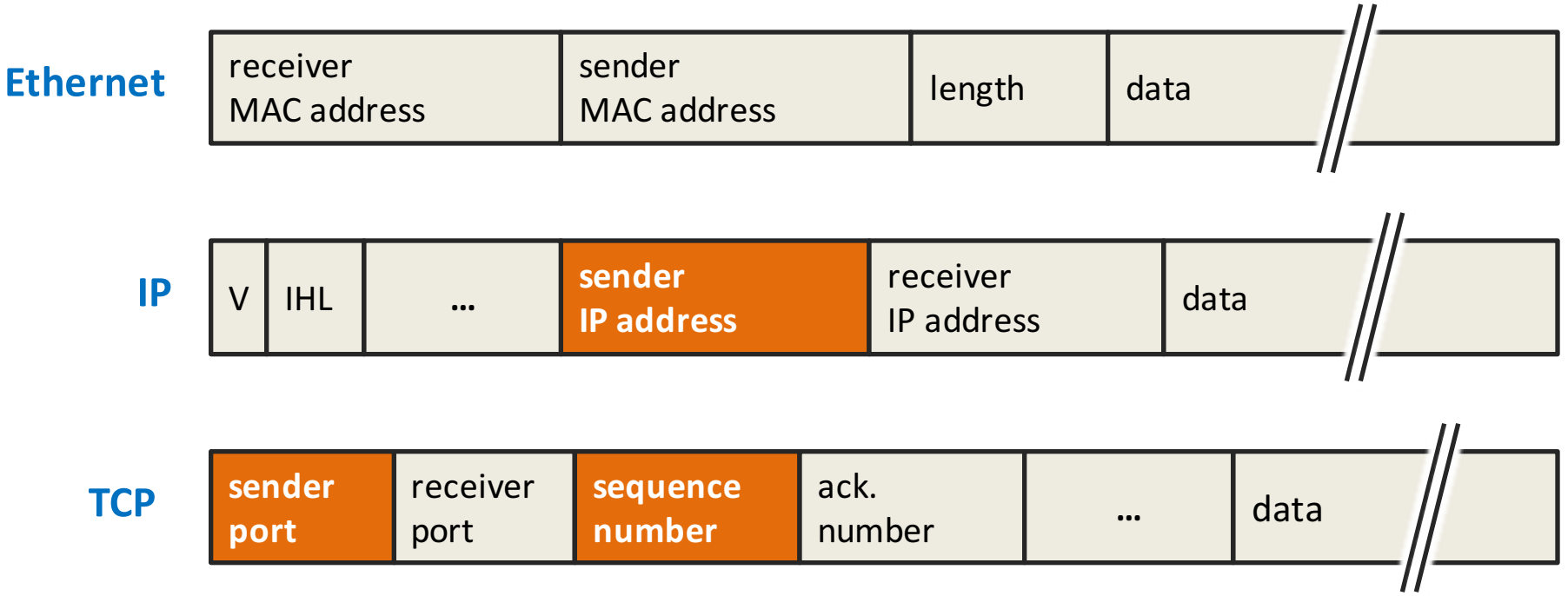

**Negligible State**

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

# Validating Responses

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

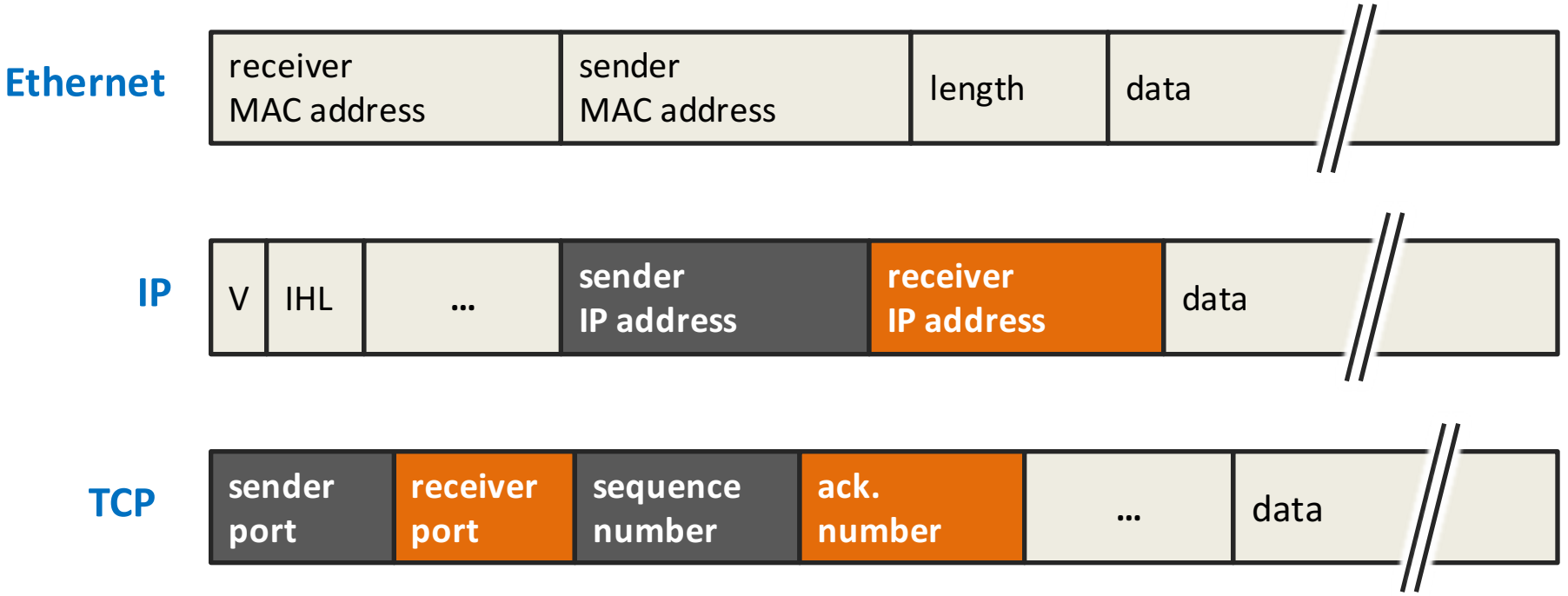
Encode secrets into mutable fields of probe packets that will have recognizable effect on responses.



# Validating Responses

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

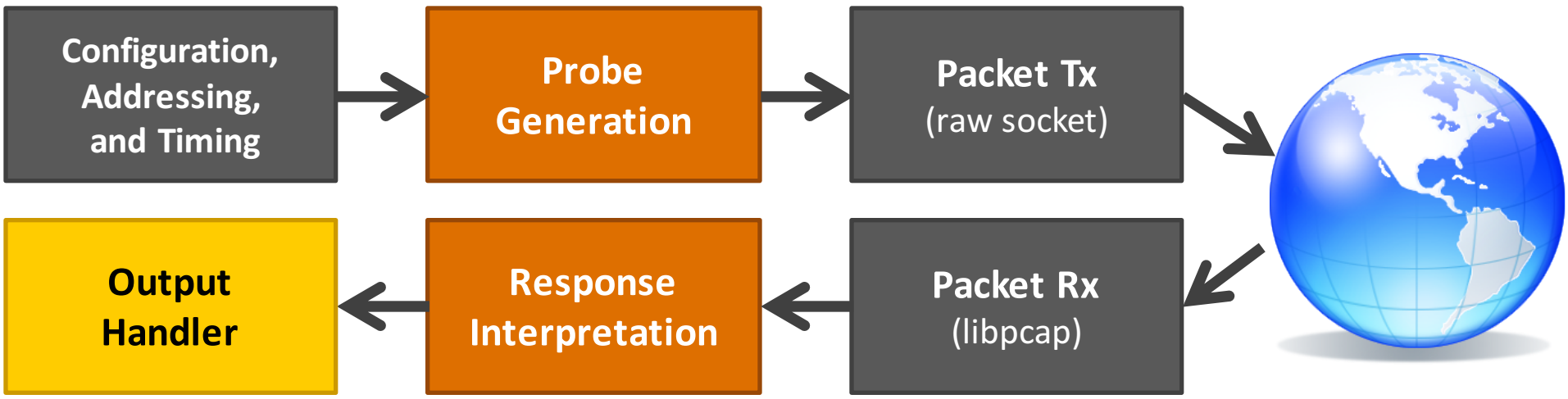
Encode secrets into mutable fields of probe packets that will have recognizable effect on responses.



# 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

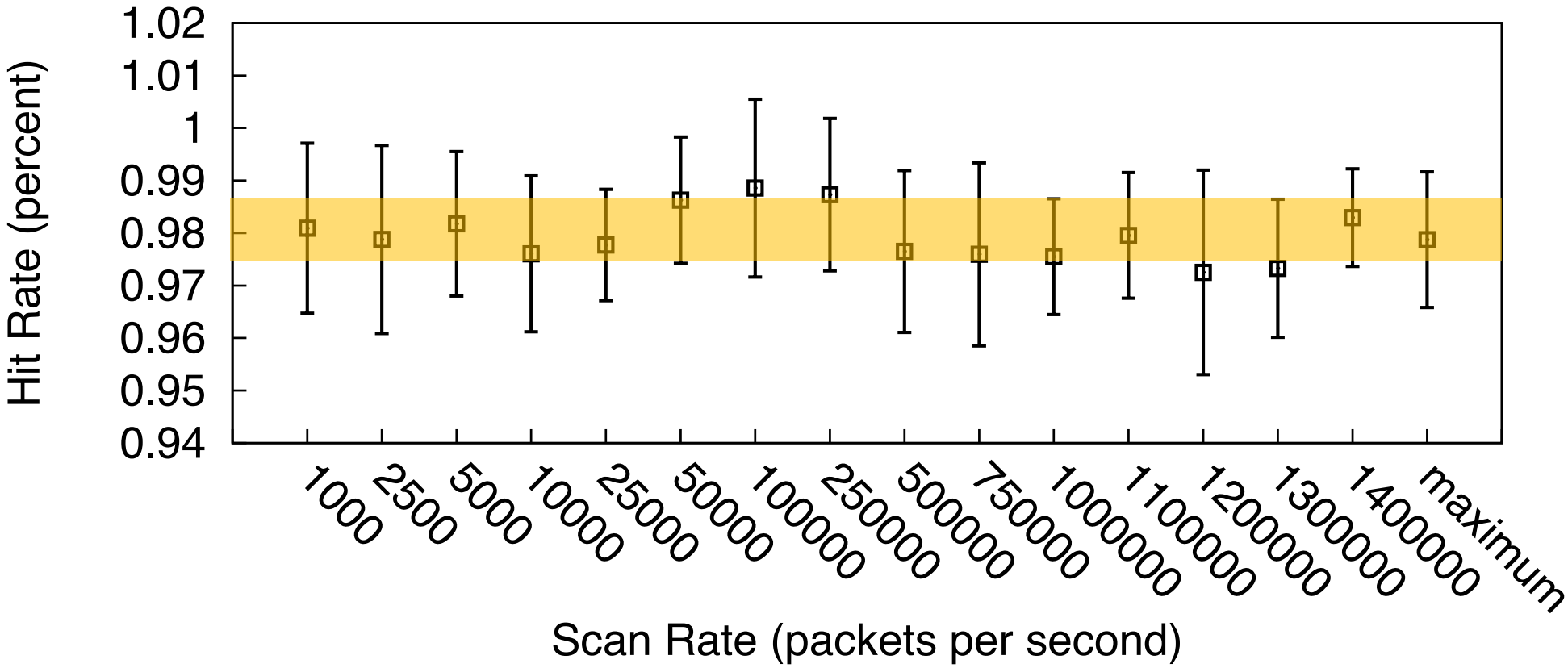


# Scan Rate – Up to 1GigE

## How fast is too fast?

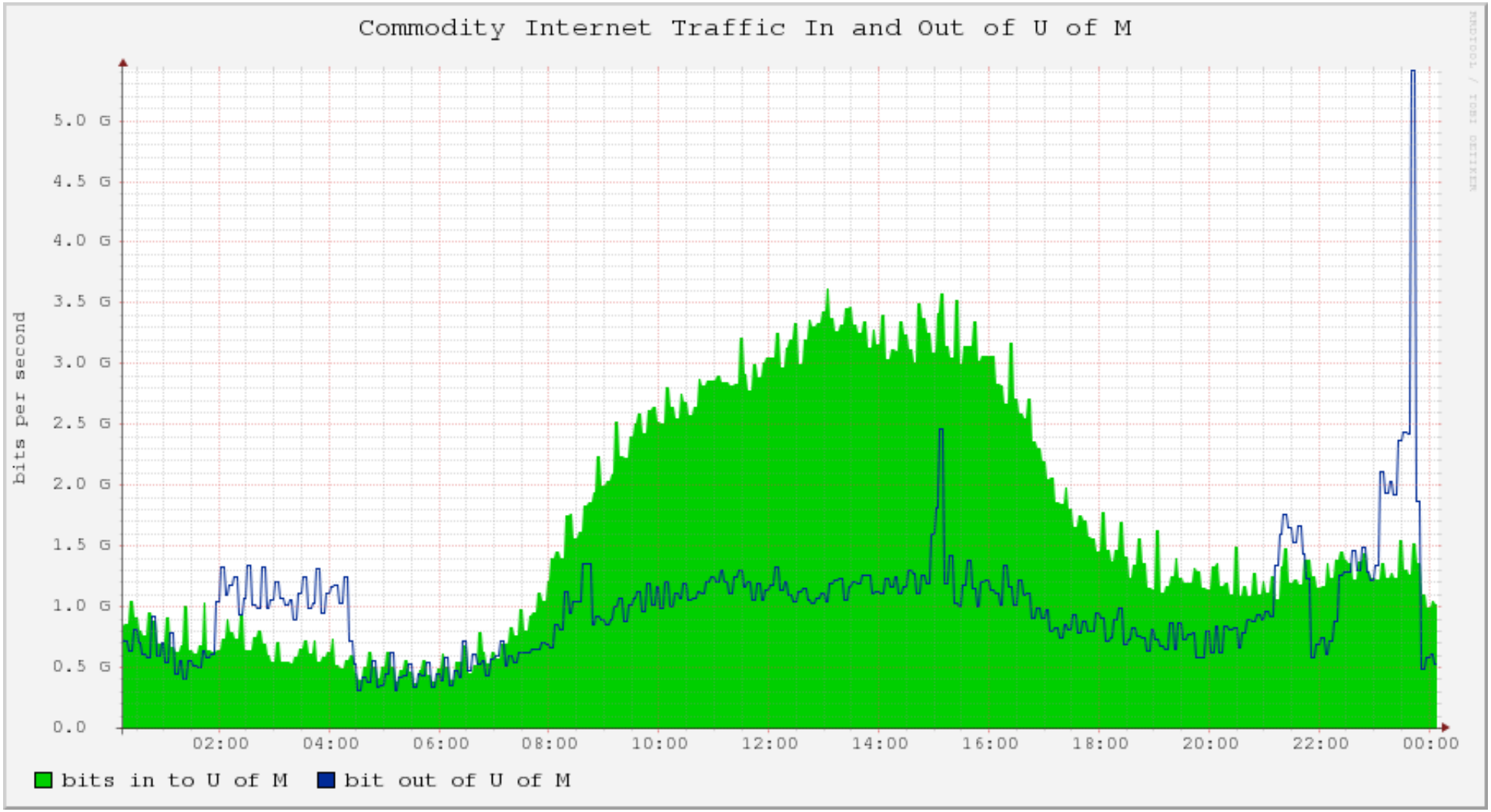
No meaningful correlation between speed and response rate.

Slower scanning does not reveal additional hosts.



# Scan Rate – 10GigE?

## How fast is too fast?



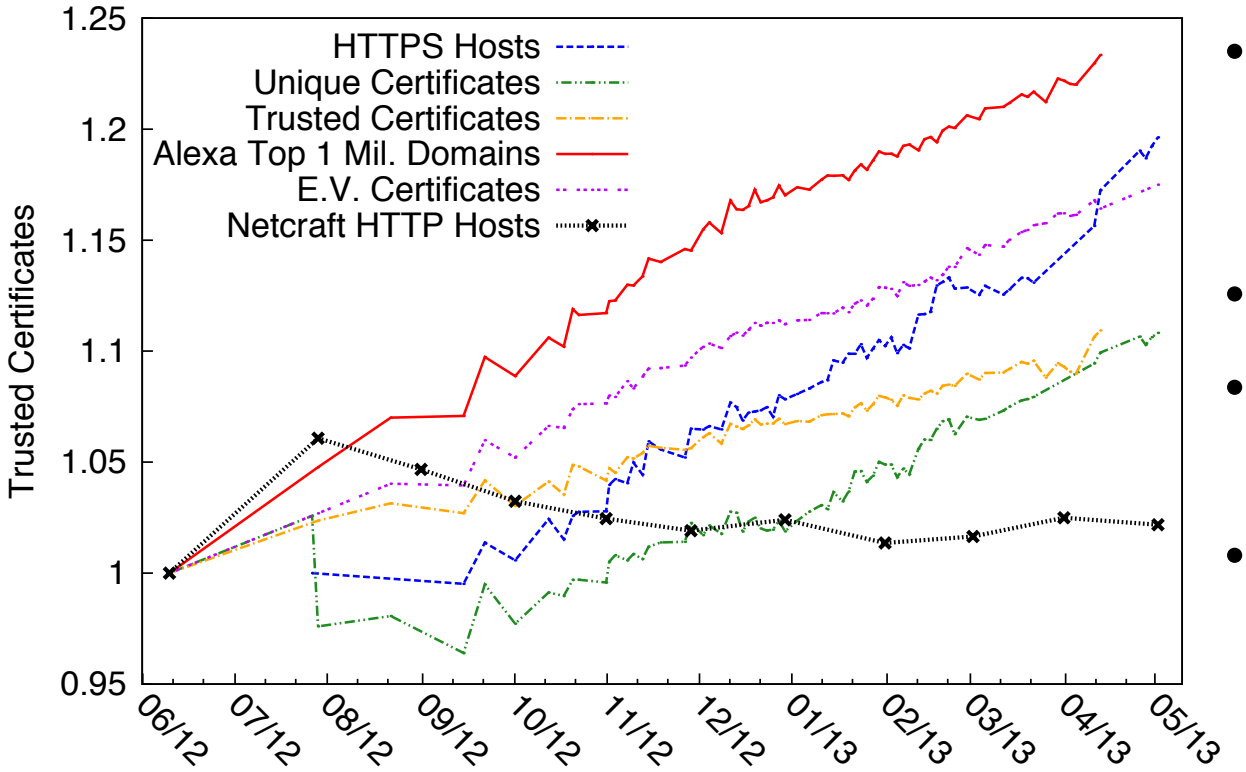
# ZMap: Applications

We did thousands of Internet-wide scans over 5 years (trillions of probes).

Please ignore probes from 141.212.121.0/24. It's just Prof. Halderman.

What can researchers do with ZMap?

## Track Adoption of Defenses




- Fine-grained analysis of HTTPS ecosystem.
  > 200 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 thousands of Internet-wide scans over 5 years (trillions of probes).

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

What can researchers do with ZMap?

**Discover Hidden Services**


Single scan reveals
86% of Tor bridge nodes

Historical scan data reveals
hidden Silk Road servers

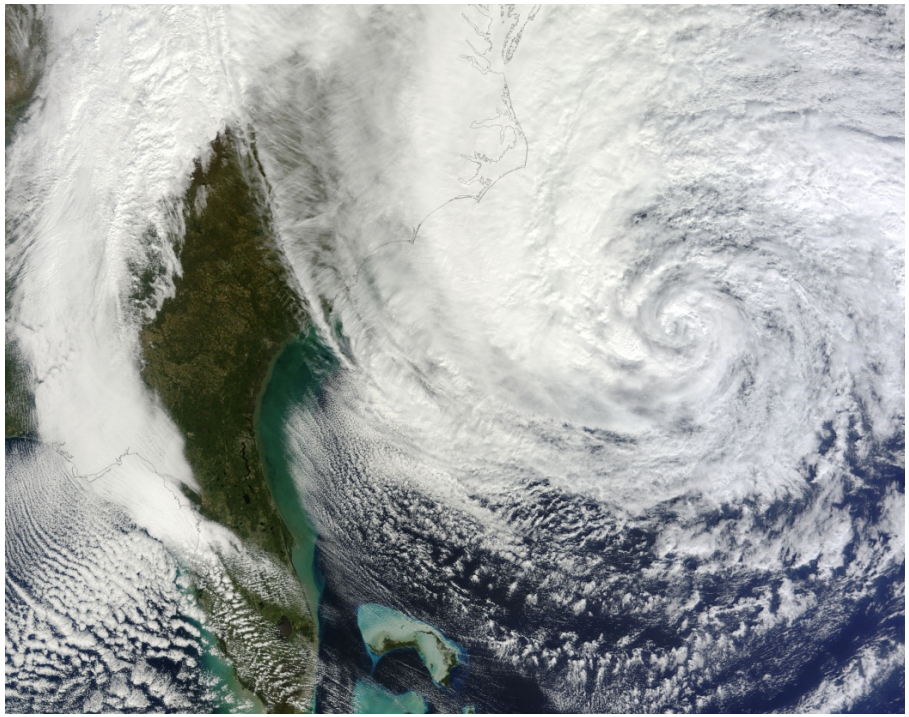
# ZMap: Applications

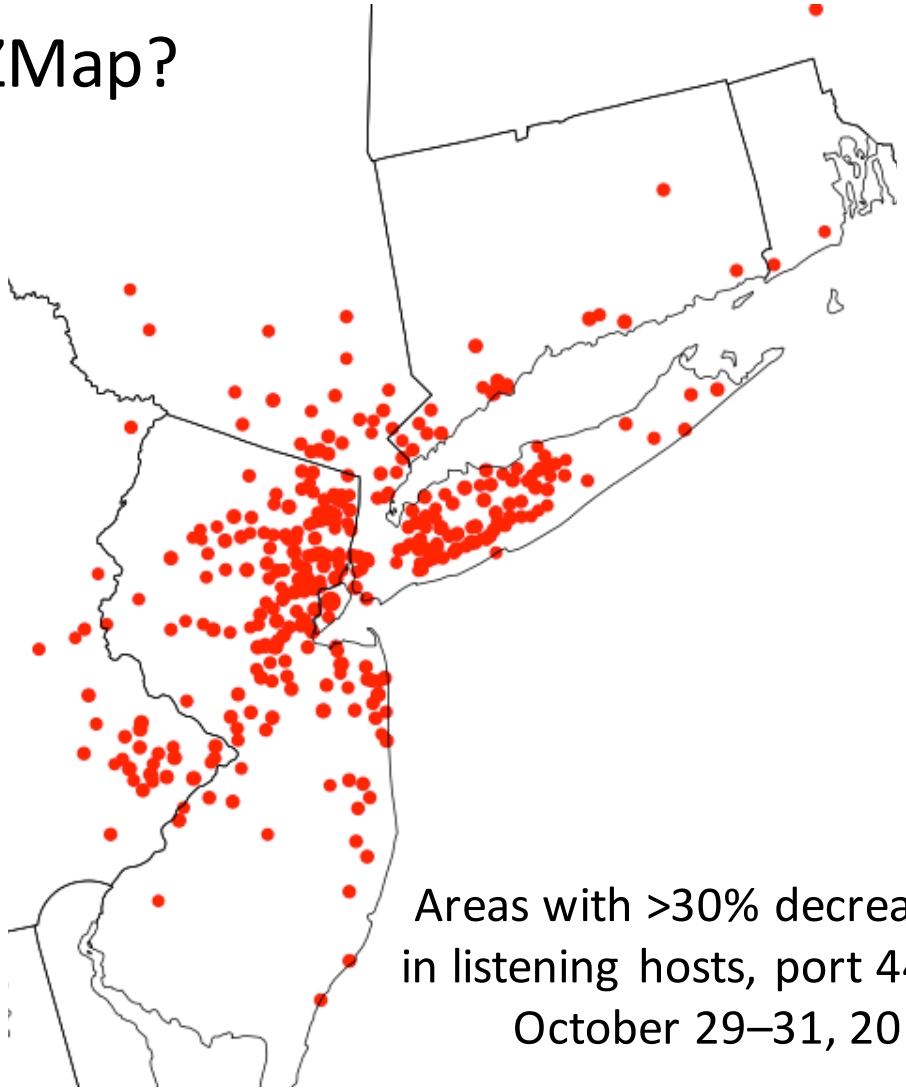
We did thousands of Internet-wide scans over 5 years (trillions of probes).

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

What can researchers do with ZMap?

## Detect Service Disruptions





Areas with >30% decrease in listening hosts, port 443
October 29–31, 2013

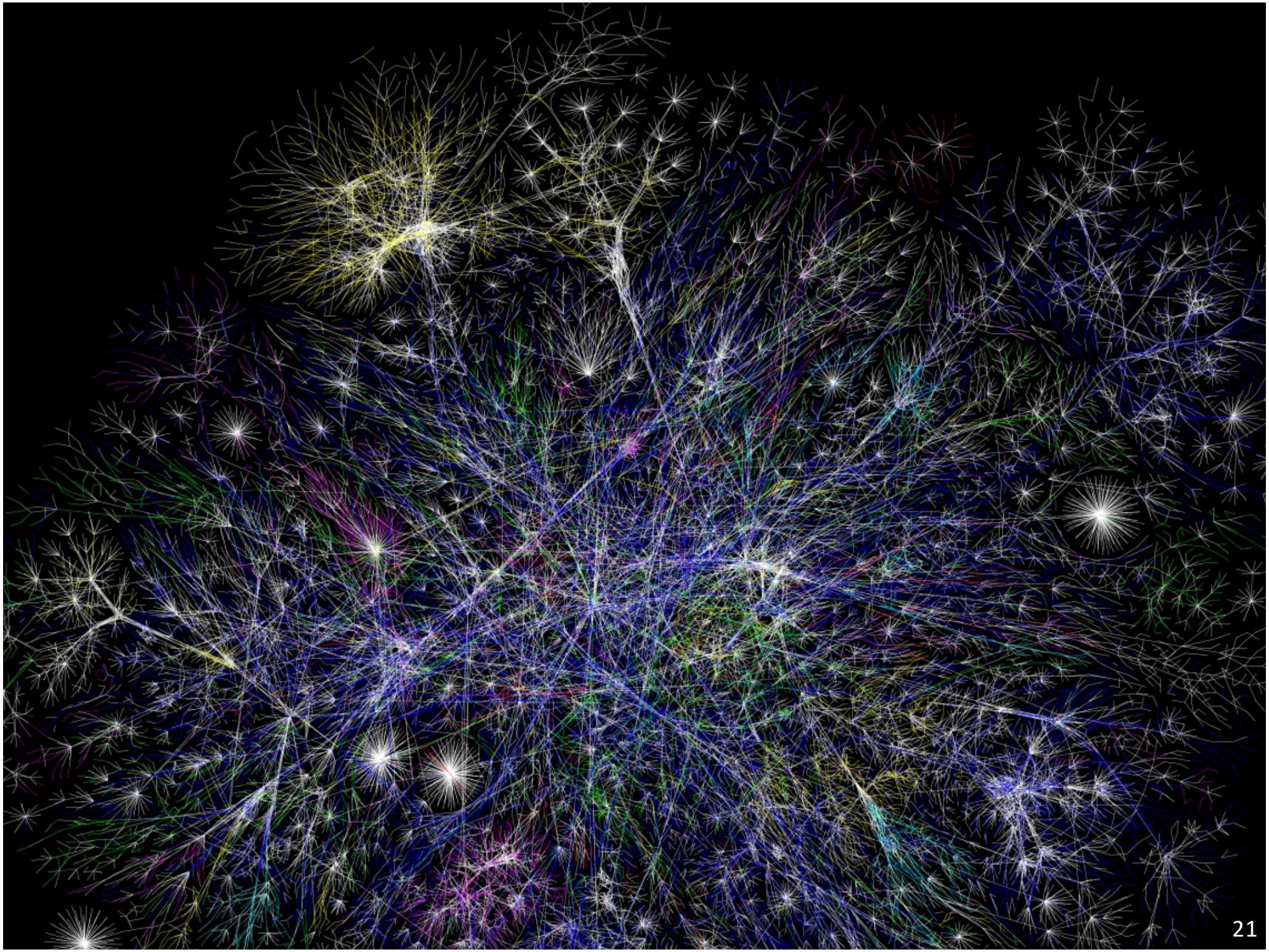
# ZMap: Applications

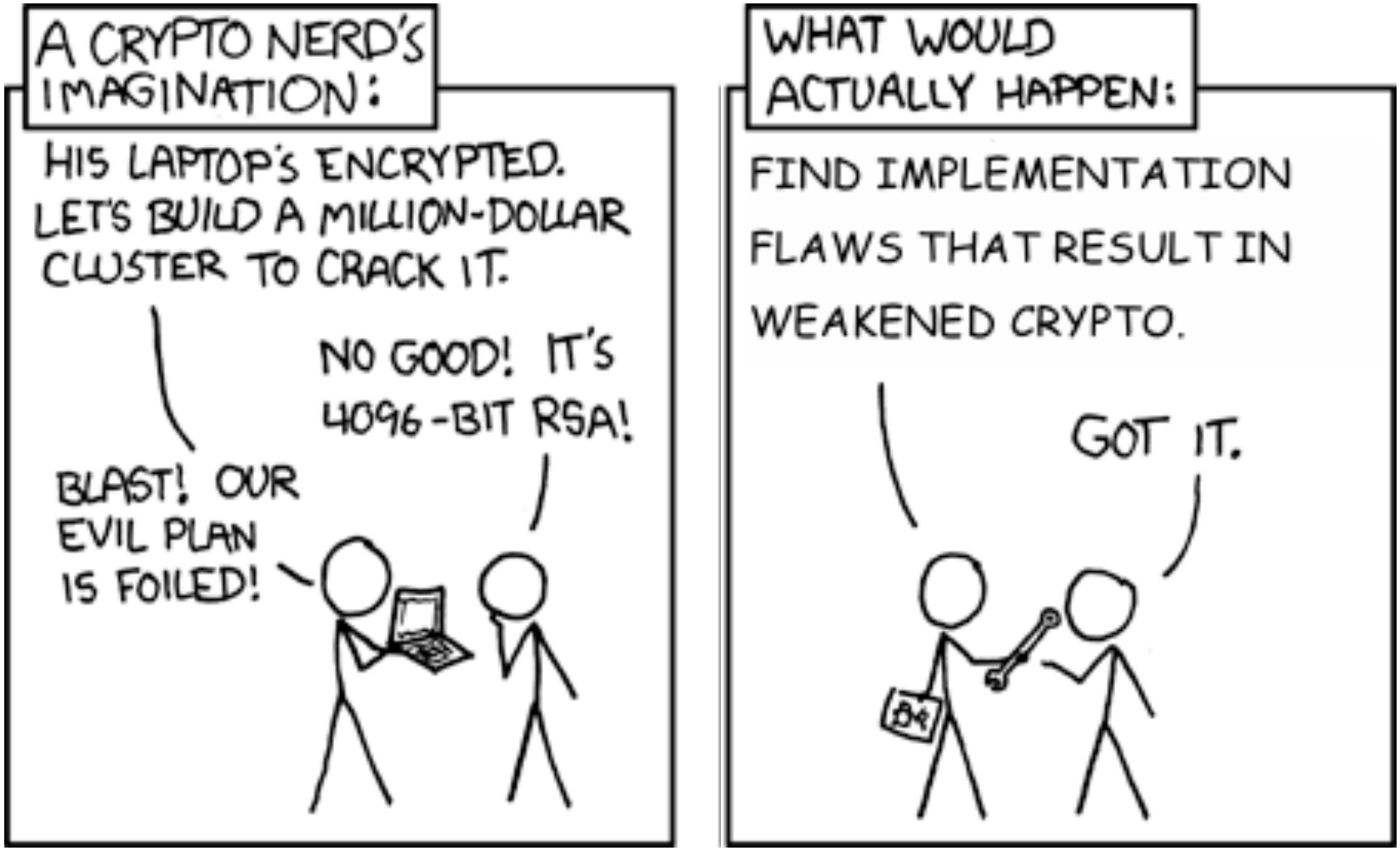
We did thousands of Internet-wide scans over 5 years (trillions of probes).

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

What can researchers do with ZMap?

**Expose and Fix Vulnerable Hosts**



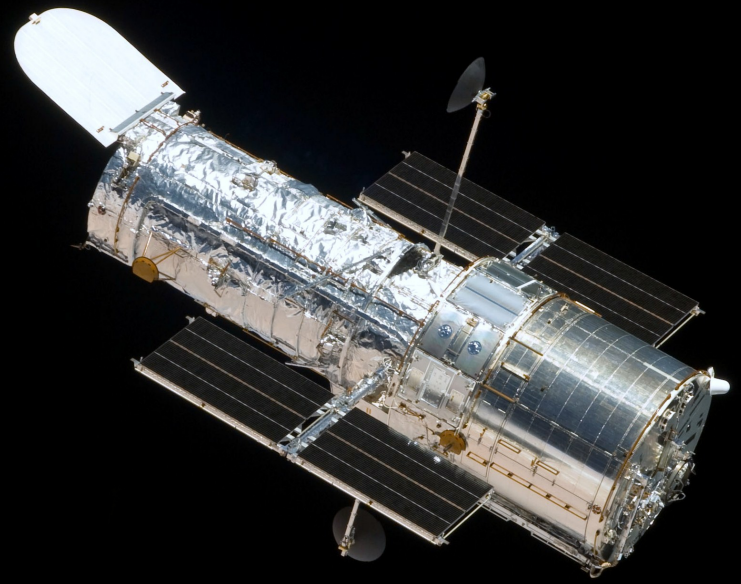

A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
FIND IMPLEMENTATION FLAWS THAT RESULT IN WEAKENED CRYPTO.
GOT IT.

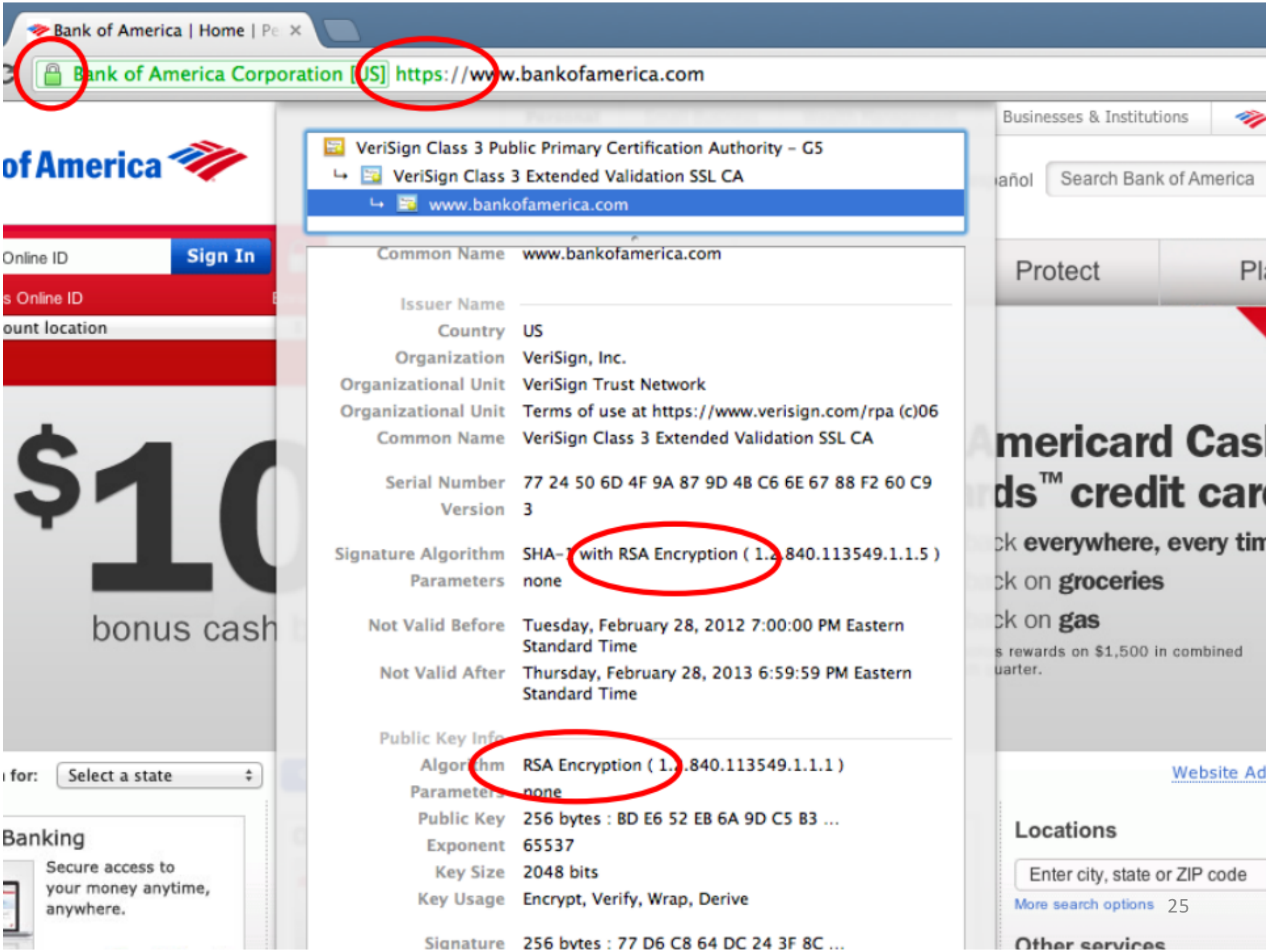
How can we identify problems with crypto implementations *at Internet scale*?

**Methodology:**

1) Collect public data from the network (public keys, signatures, ciphertext, …)
2) Mine the data for evidence of vulnerabilities
3) Investigate causes in representative implementations

# Data collection




Bank of America | Home | Pe
Bank of America Corporation [US] https://www.bankofamerica.com
VeriSign Class 3 Public Primary Certification Authority – G5
VeriSign Class 3 Extended Validation SSL CA
www.bankofamerica.com
Common Name www.bankofamerica.com
Issuer Name
Country US
Organization VeriSign, Inc.
Organizational Unit VeriSign Trust Network
Organizational Unit Terms of use at https://www.verisign.com/rpa (c)06
Common Name VeriSign Class 3 Extended Validation SSL CA
Serial Number 77 24 50 6D 4F 9A 87 9D 4B C6 6E 67 88 F2 60 C9
Version 3
Signature Algorithm SHA-1 with RSA Encryption ( 1.2.840.113549.1.1.5 )
Parameters none
Not Valid Before Tuesday, February 28, 2012 7:00:00 PM Eastern Standard Time
Not Valid After Thursday, February 28, 2013 6:59:59 PM Eastern Standard Time
Public Key Info
Algorithm RSA Encryption ( 1.2.840.113549.1.1.1 )
Parameters none
Public Key 256 bytes : BD E6 52 EB 6A 9D C5 B3 ...
Exponent 65537
Key Size 2048 bits
Key Usage Encrypt, Verify, Wrap, Derive
Signature 256 bytes : 77 D6 C8 64 DC 24 3F 8C ...
Businesses & Institutions
Search Bank of America
Online ID
Sign In
Protect
bonus cash
Banking
Secure access to your money anytime, anywhere.
Locations
Enter city, state or ZIP code
More search options

# HTTPS and Public Keys

Server presents a chain of X.509 certificates.

Public keys are used to:

- sign certificates, binding domain names to sites' keys
- encrypt session keys (in RSA key exchange, most common)
- sign key-exchange parameters (for Diffie–Hellman)

A compromised key could be used to:

- sign new trusted certificates (if it's a CA key)
- decrypt passively collected traffic (for RSA key exchange)
- man-in-the-middle connections

# Collecting Public Keys

**1. Finding Hosts**

Nmap from EC2

25 hosts * 25 hours

28.9 M hosts

listening on port 443

**2. TLS Handshaking**

Event Driven Process

3 hosts * 48 hours

12.8 M hosts

completed handshakes

**3. Parsing Certs**

OpenSSL, PostgreSQL

Extract public keys

5.8 M unique certs

(2 M browser trusted)

| Distinct keys by type: | RSA | DSA | ECDSA | GOST |
|---|---|---|---|---|
| SSL Observatory (12/2010) | 3,900,000 | 1,900 | 0 | 0 |
| Mining Ps and Qs (10/2011) | 5,600,000 | 6,200 | 7 | 12 |

# SSH Host Keys

SSH host keys are used to:

- authenticate servers to clients.

A compromised SSH host key could be used to:

- man-in-the-middle connections (for SSHv2, common case)

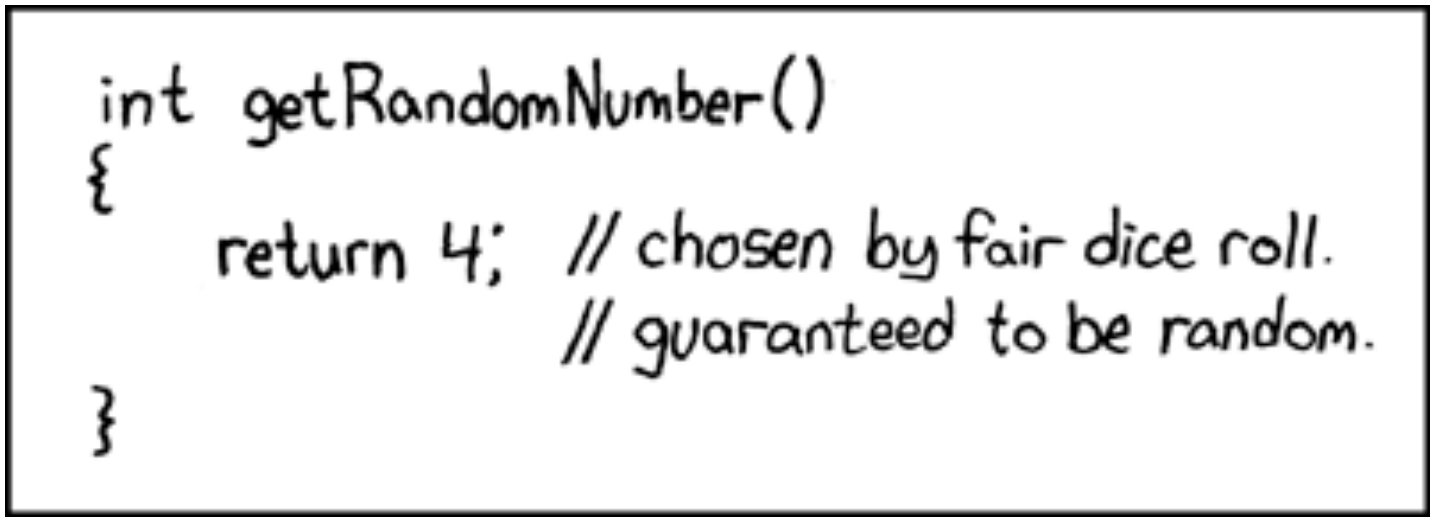
We scanned port 22 for SSH host keys and handshake signatures in February and April 2012.

| Open port | Handshake | RSA | DSA | ECDSA | GOST |
|---|---|---|---|---|---|
| 23,000,000 | 10,200,000 | 3,800,000 | 2,000,000 | 321,000 | 225 |

Similar number of keys to HTTPS, but far more DSA and ECDSA.

# Looking for problems

# What could go wrong?



```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

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

End of story?


# A Method for Obtaining Digital Signatures and Public-Key Cryptosystems

R.L. Rivest, A. Shamir, and L. Adleman*

# Textbook RSA

**Public Key**

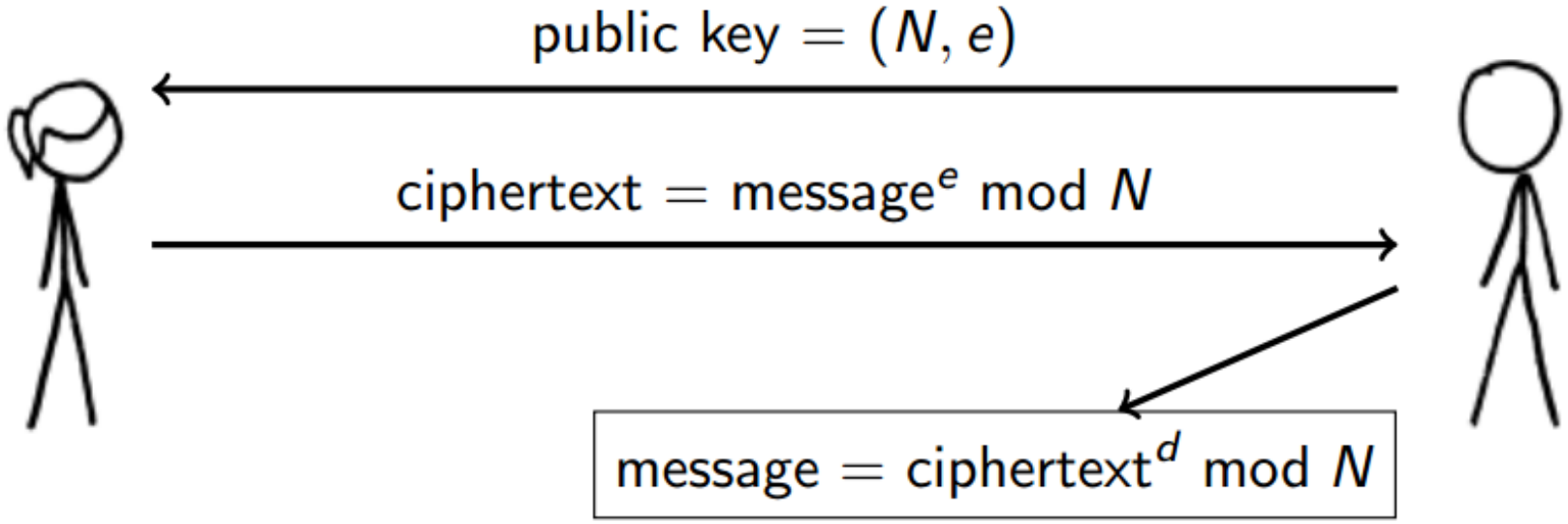
$N = pq$ modulus

$e$ encryption exponent

**Private Key**

$p, q$ primes

$d$ decryption exponent
(compute from $e$,$p$,$q$)



# What could go wrong?

1. Repeated keys

# 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.

  Security Implications:

  - Either host could man-in-the-middle the other.
  - Either host could decrypt traffic from TLS RSA key exchange.

# Looking for problems: Repeated 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.

# Looking for problems: Repeated keys

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

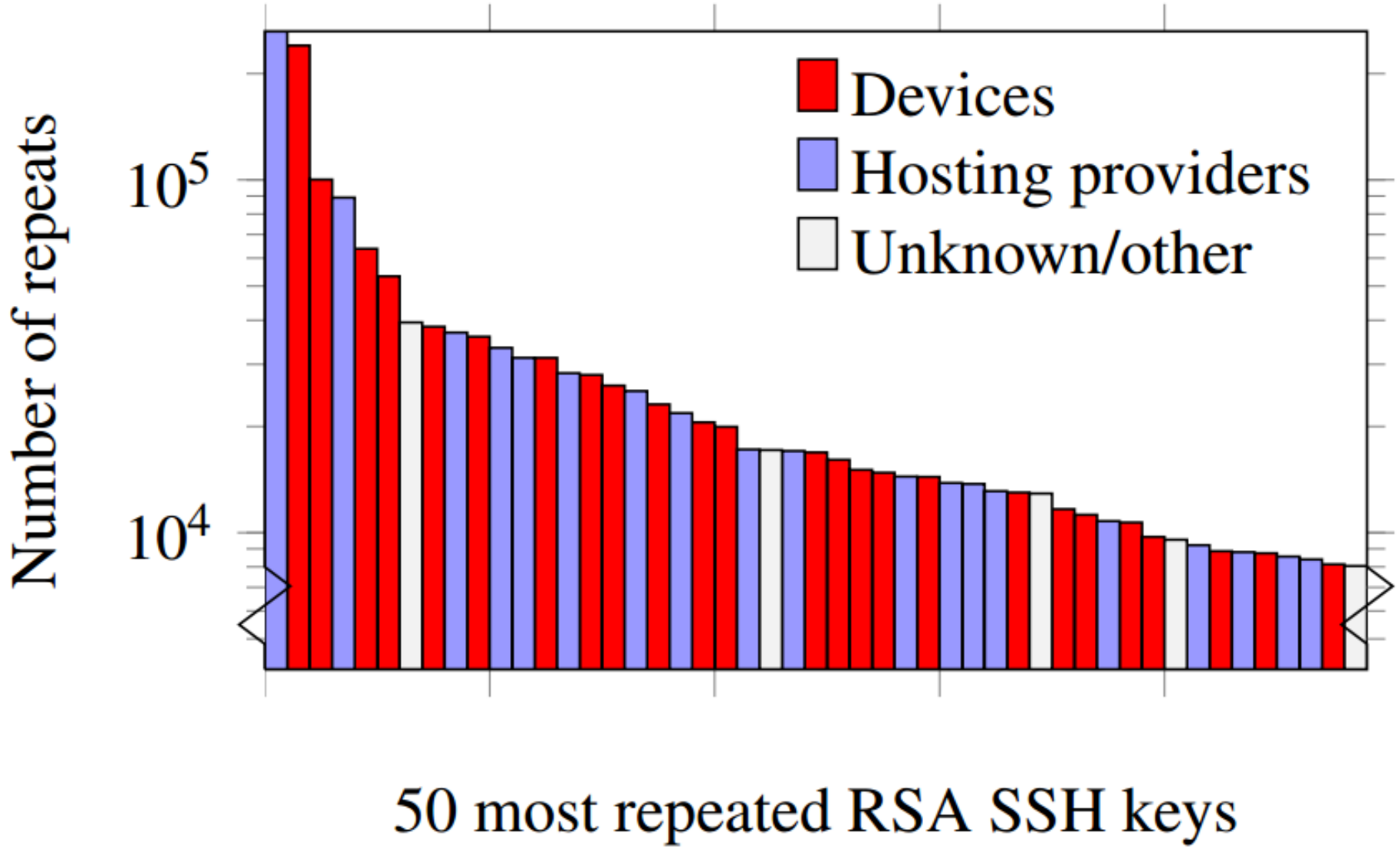
Common (and unwise) reasons to share keys:

- 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%)


Devices
Hosting providers
Unknown/other
Number of repeats
$10^5$
$10^4$
50 most repeated RSA SSH keys

# What could go wrong?

1. Repeated keys

2. Repeated factors in RSA keys

# What could go wrong: Shared RSA factors

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,

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

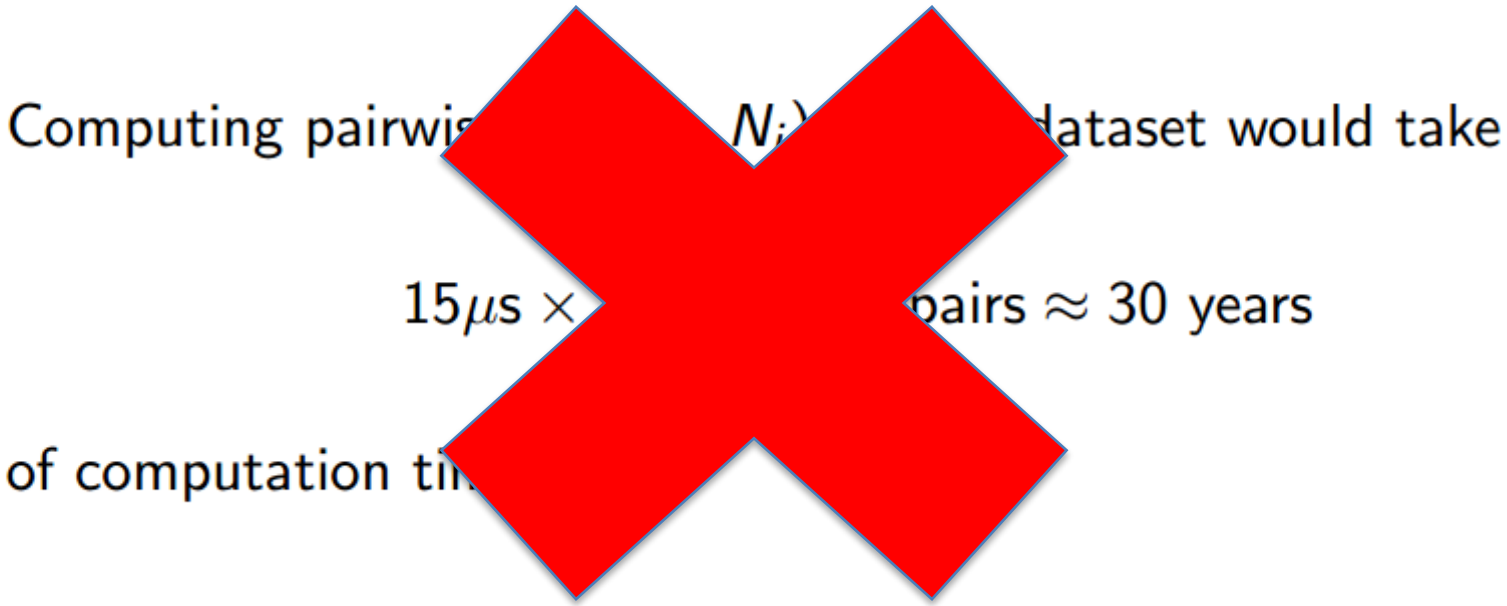
$$\gcd(N_1, N_2) = p$$

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

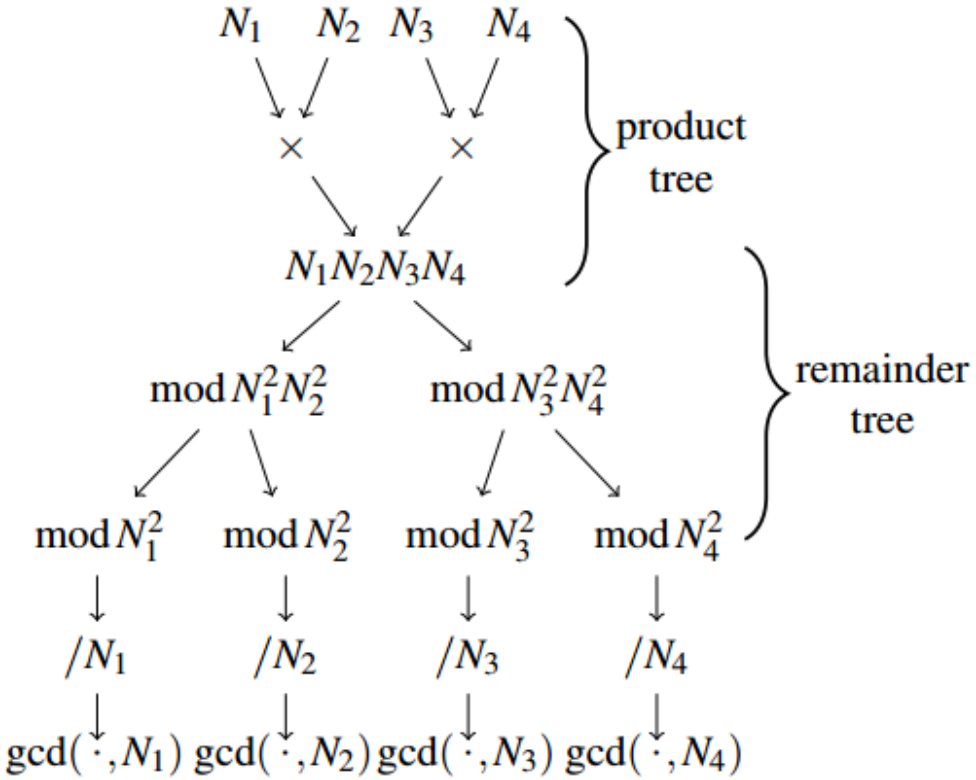


# Efficiently computing pairwise GCDs

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.

# What could go wrong?

1. Repeated keys
2. Repeated factors in RSA keys
3. Weak DSA signature nonces

**FIPS PUB 186-3**

---

**FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION**

# Digital Signature Standard (DSS)

**CATEGORY: COMPUTER SECURITY** **SUBCATEGORY: CRYPTOGRAPHY**

---

# What could go wrong: Weak DSA signature nonce

## DSA Public Key

$p, q, g$ domain parameters

$y = g^x \bmod p$

## Private Key

$x$ private key

DSA signatures require a random nonce.

- If the nonce is predictable
  $\rightarrow$ can easily compute long-term private key.
- Two distinct signatures use same nonce and private key
  $\rightarrow$ can easily compute nonce
  $\rightarrow$ can easily compute long-term private key.

Security Implications:

- Anyone could man-in-the-middle vulnerable hosts.

# Computing weak DSA keys

- Collected 9,114,925 DSA signatures from two SSH scans.
- 4,365 contained the same nonce as another signature.
- Computed 281 distinct private DSA keys.
- Keys were served by 105,728 (1.03%) of SSH DSA hosts.

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/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
```

# 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

# Research Methodology

1) Collect keys
2) Mine data for vulnerabilities
3) **Investigate causes**

# Linux random number generators

| /dev/random | /dev/urandom |
| --- | --- |
| "high-quality" randomness | pseudorandomness |
| blocks if insufficient entropy available | never blocks |

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

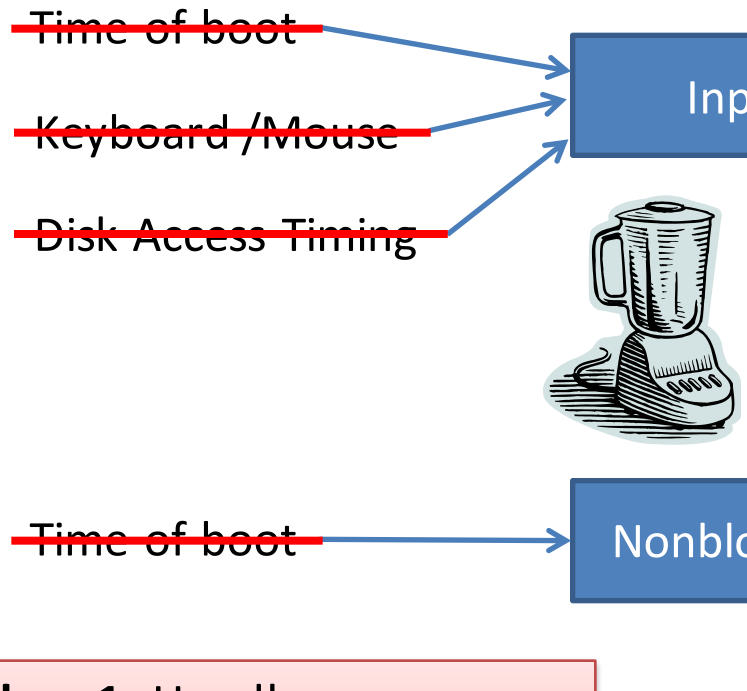
```
/* We'll use /dev/urandom by default,
since /dev/random is too much hassle. If
system developers aren't keeping seeds
between boots nor getting any entropy from
somewhere it's their own fault. */
#define DROPBEAR_RANDOM_DEV "/dev/urandom"
```

`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.




Only happens if Input Pool contains more than 192 bits…

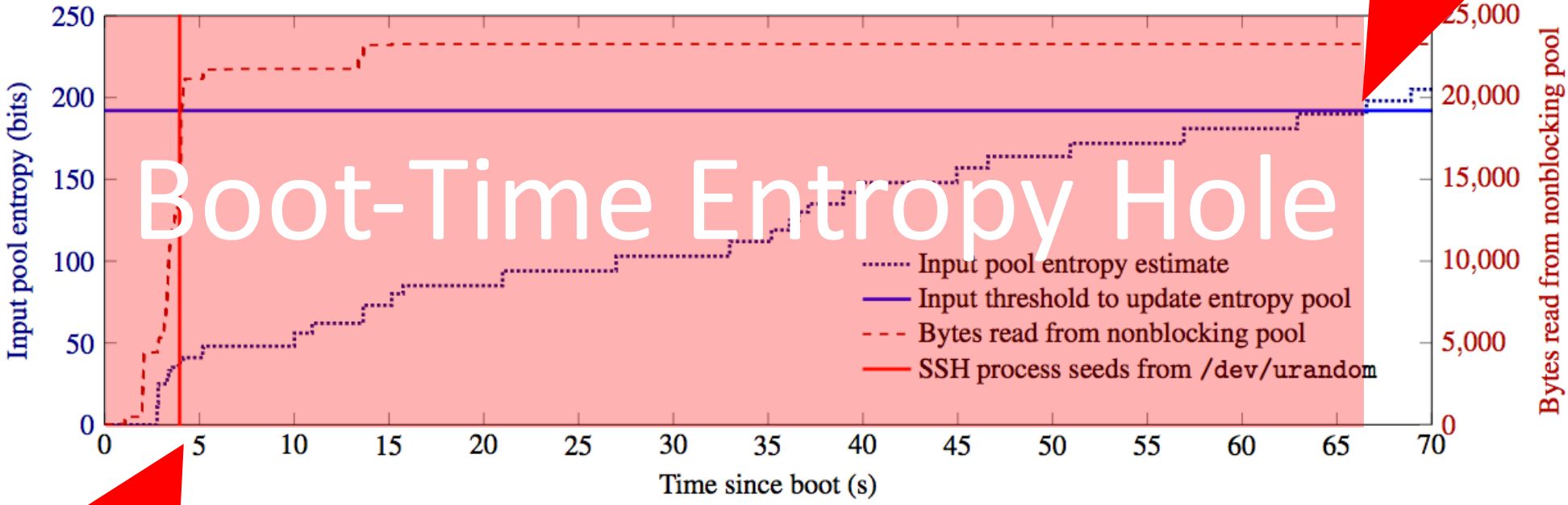
*"Anyone who considers arithmetical methods of producing random digits is, of course, in a state of sin."*

— John von Neumann

**Problem 1:** Headless or embedded devices may lack all these entropy sources

**Problem 2:** `urandom` may not have incorporated any entropy when queried by software

# Boot-Time Entropy Hole




Entropy first mixed into /dev/urandom

OpenSSH seeds from /dev/urandom

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

# Generating vulnerable RSA keys in software

- Insufficiently random seeds for pseudorandom number generator $\Longrightarrow$ 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.

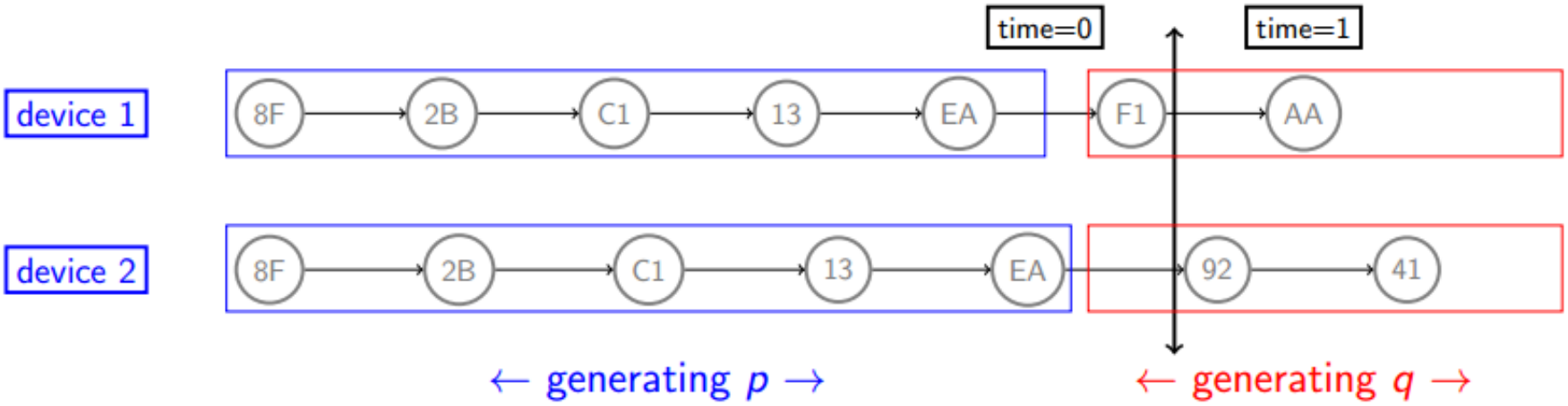
But why do we see factorable keys?

# Generating factorable RSA keys in software

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

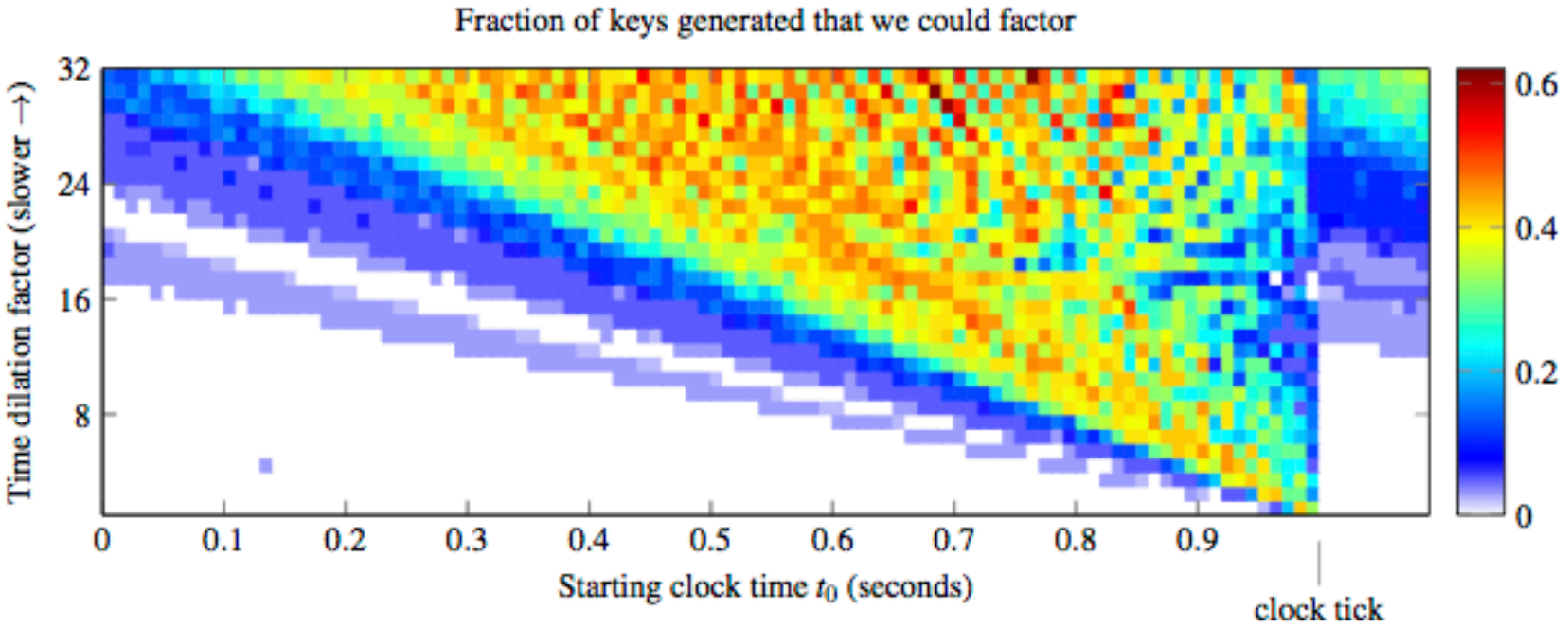
OpenSSL adds time in seconds

Insufficient randomness can lead to factorable keys.




Experimentally verified OpenSSL generates factorable keys in this situation.

# Factorable keys generated by OpenSSL




time as entropy source + asynchronous clocks → factorable keys

# 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.

Key-check service for end users.

# Practical Mitigations

**Developers and manufacturers:**

- Defense in depth: test, post-process, use multiple sources of randomness.
- Collect entropy more aggressively, add hardware sources.
- Seed devices with entropy at the factory.
- Run for a while before generating cryptographic keys.

**CAs:**

- Test for repeated, factorable, and other weak keys.

**Users:**

- Check against known weak keys. (See factorable.net)
- Replace default certificates.

# Is this just the tip of the iceberg?

Only looked at data from servers, not clients.

- TODO: Look at TLS CLIENT_HELLO nonces.
- TODO: Look at TCP ISNs, especially close to boot.
- TODO: Other protocols and ciphers.

Mainly see devices without real-time clocks.

- RTC may mask serious problems if time is the only entropy source.

On traditional PCs, margin of safety on first boot is slim.

- Not observed to be exploitable so far.

Mobile devices?

Many other security mechanisms need good randomness.

# Lessons

**Systems:**

- New insights from taking a macroscopic view of crypto practice.
- Cryptographic entropy is a systems problem—can't be solved at only one layer.
- Hard to get entropy in practice, especially on embedded devices.

**Cryptography:**

- Need cryptosystems that are more resilient to random number generation failures (e.g. DSA nonce deterministically generated from message)

# Contributions

Methodology for discovering crypto vulnerabilities at Internet scale, without having to reverse-engineer or study many heterogeneous implementations.

In-depth analysis of root causes of weak keys in widely deployed software.

Mitigations and suggestions for developers, manufacturers, users, and researchers.

# Heartbleed Vulnerability

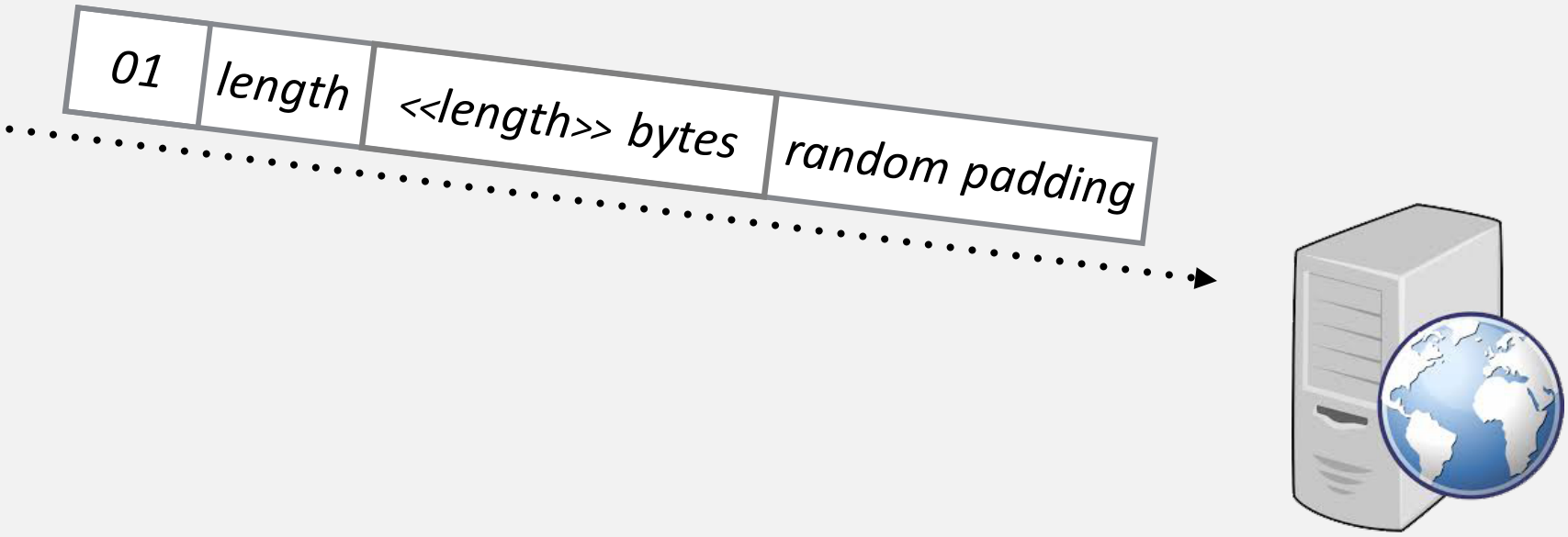
In April 2014, OpenSSL disclosed a catastrophic bug in their implementation of the TLS Heartbeat Extension

Vulnerability allowed attackers to dump private cryptographic keys, logins, and other private user data

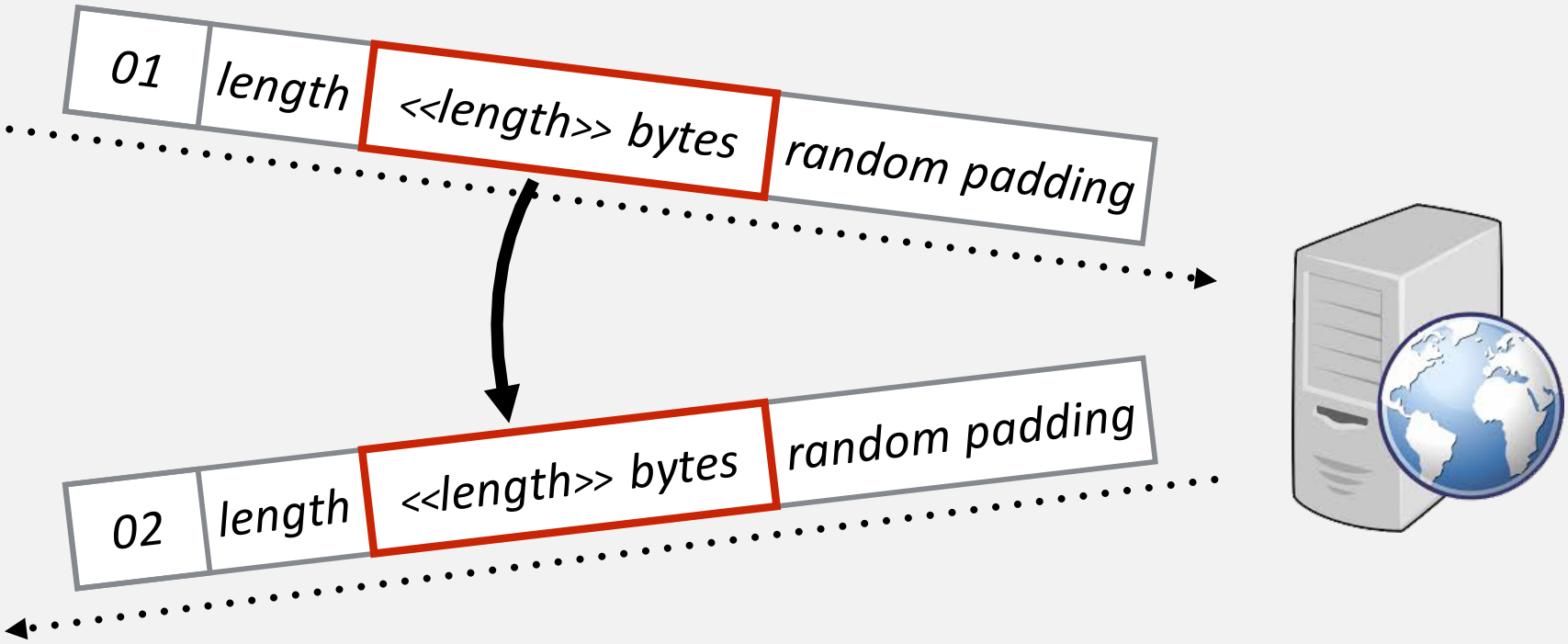
Potentially effected any service that used OpenSSL for TLS—including web, mail, messaging and database servers

An estimated **24-55%** of HTTPS websites were initially vulnerable

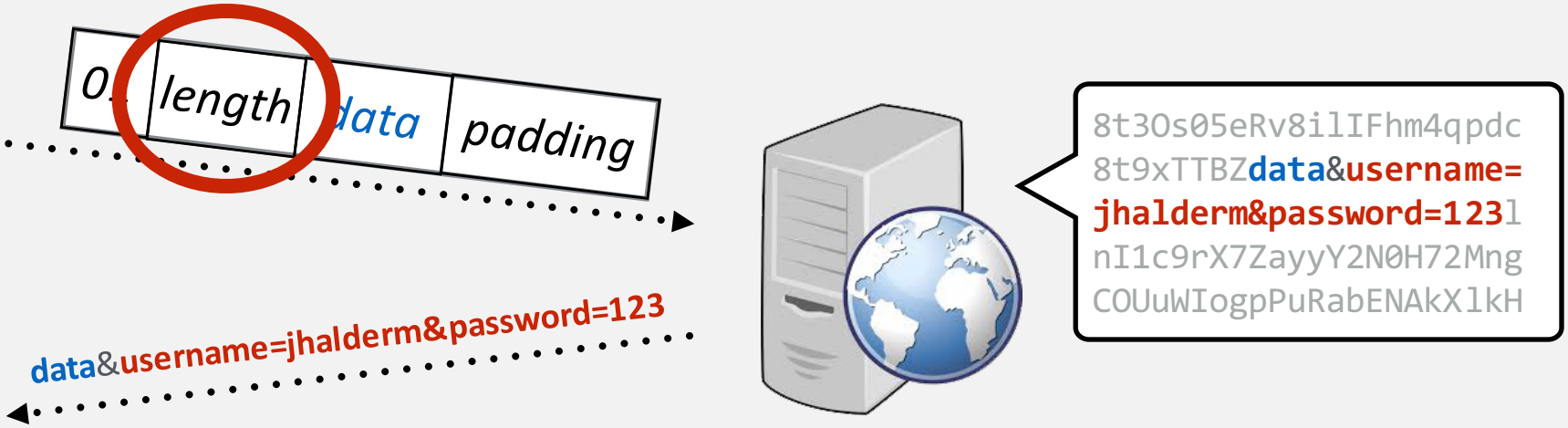
# TLS Heartbeat Extension



# TLS Heartbeat Extension



# Heartbleed Vulnerability



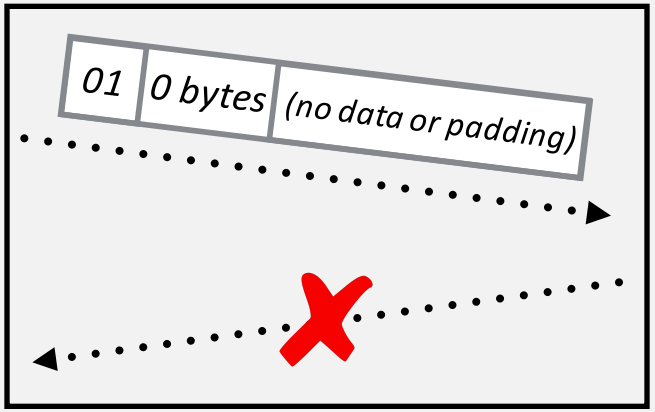

**Heartbleed Vulnerability:** OpenSSL trusts user provided length field and echoes back memory contents following request data

# Tracking the Vulnerability

**Data Collection**

- Began scanning 48 hours after public disclosure
- Scanned Alexa Top 1 Million and 1% samples of IPv4 every 8 hours

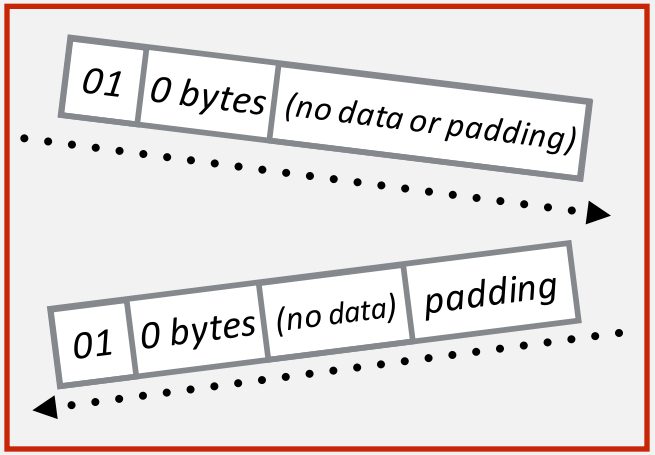
RFC 6520 Compliant



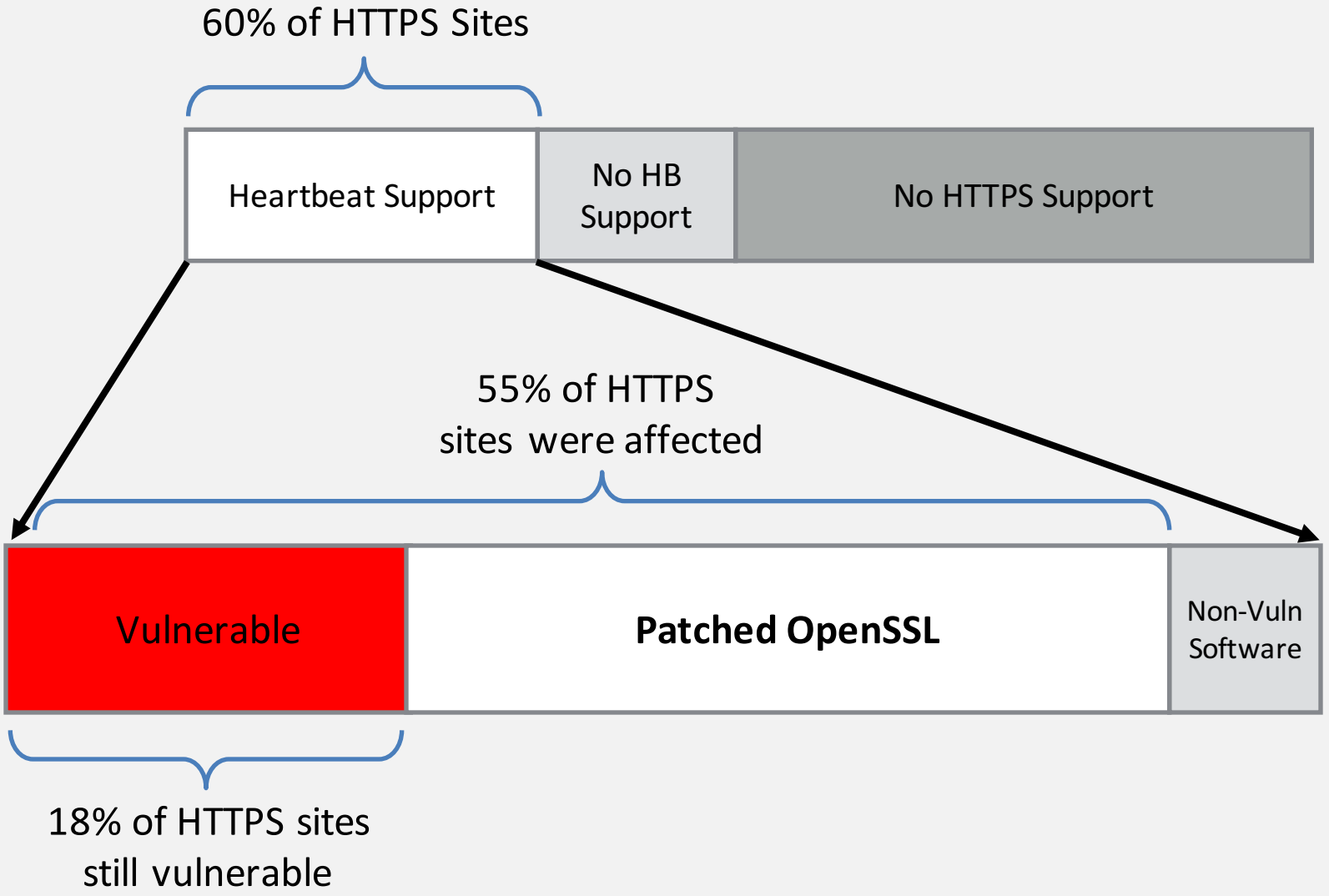

**Scanning for Heartbleed**

- Modified ZMap to scan for vulnerable versions of OpenSSL
- Instead of exploiting the vuln, we checked for non-compliant behavior of vulnerable version

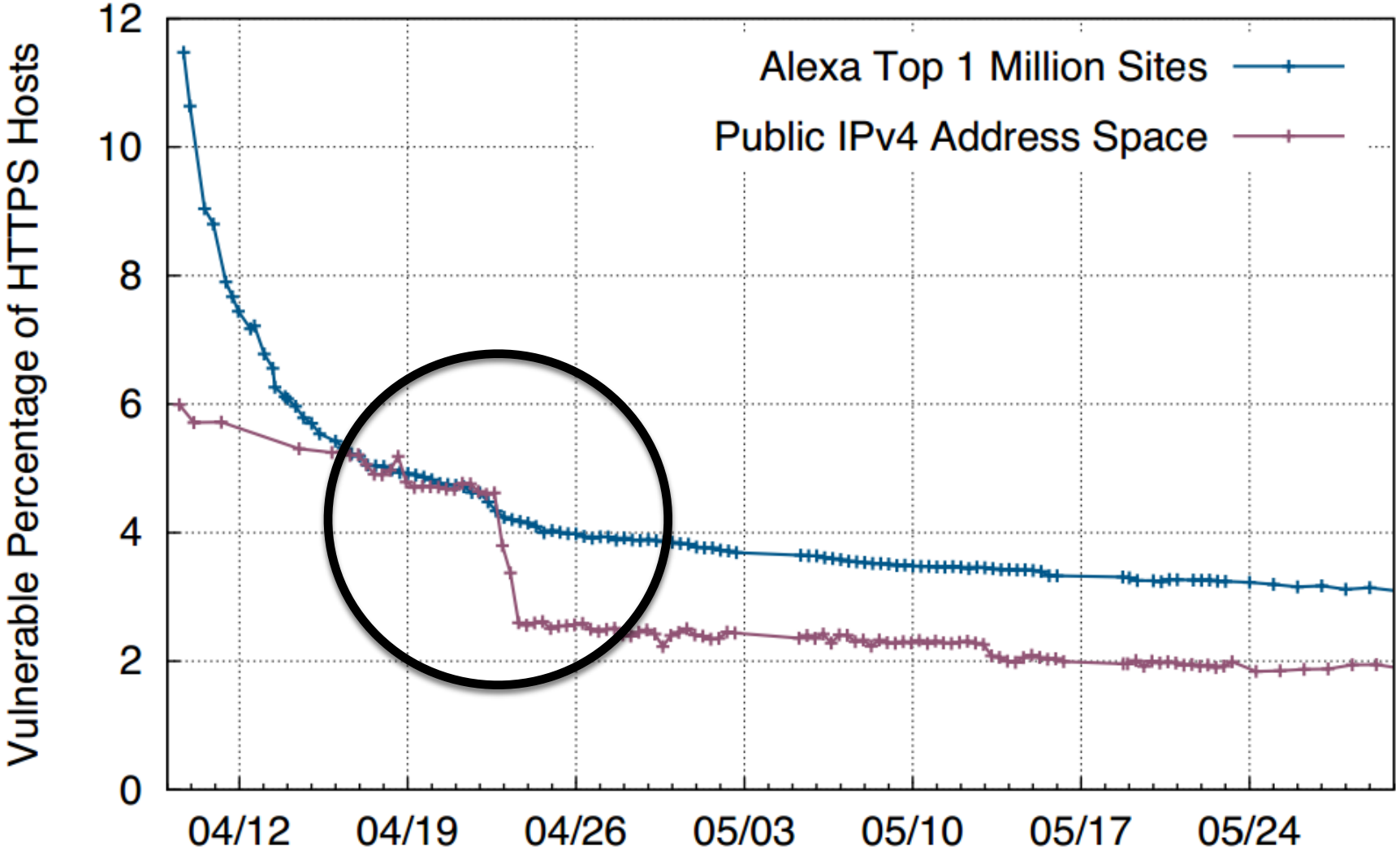
Vulnerable OpenSSL



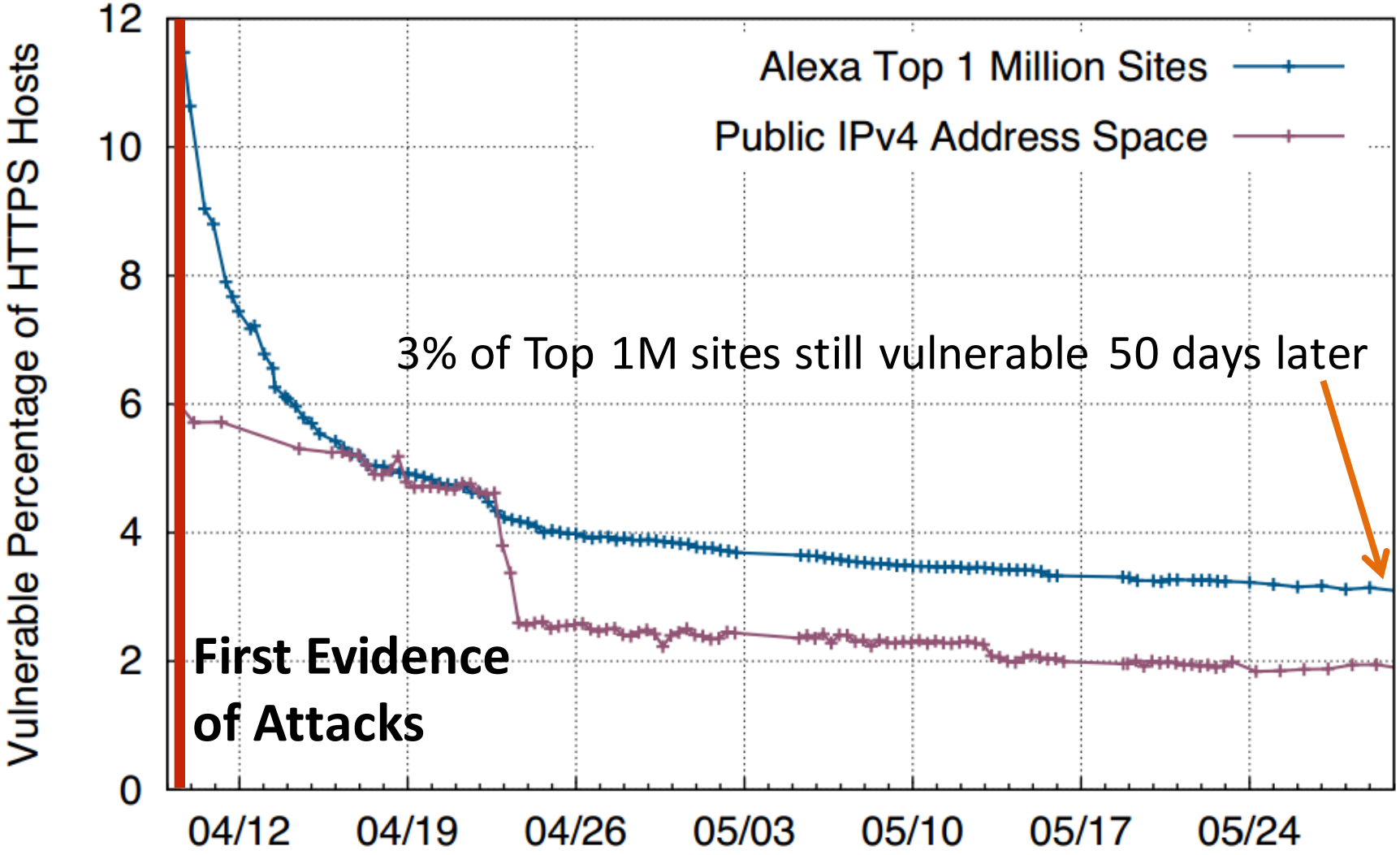
# Initial Results – Disclosure +2 Days



# Patching Over Time



# Patching Over Time



# 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.

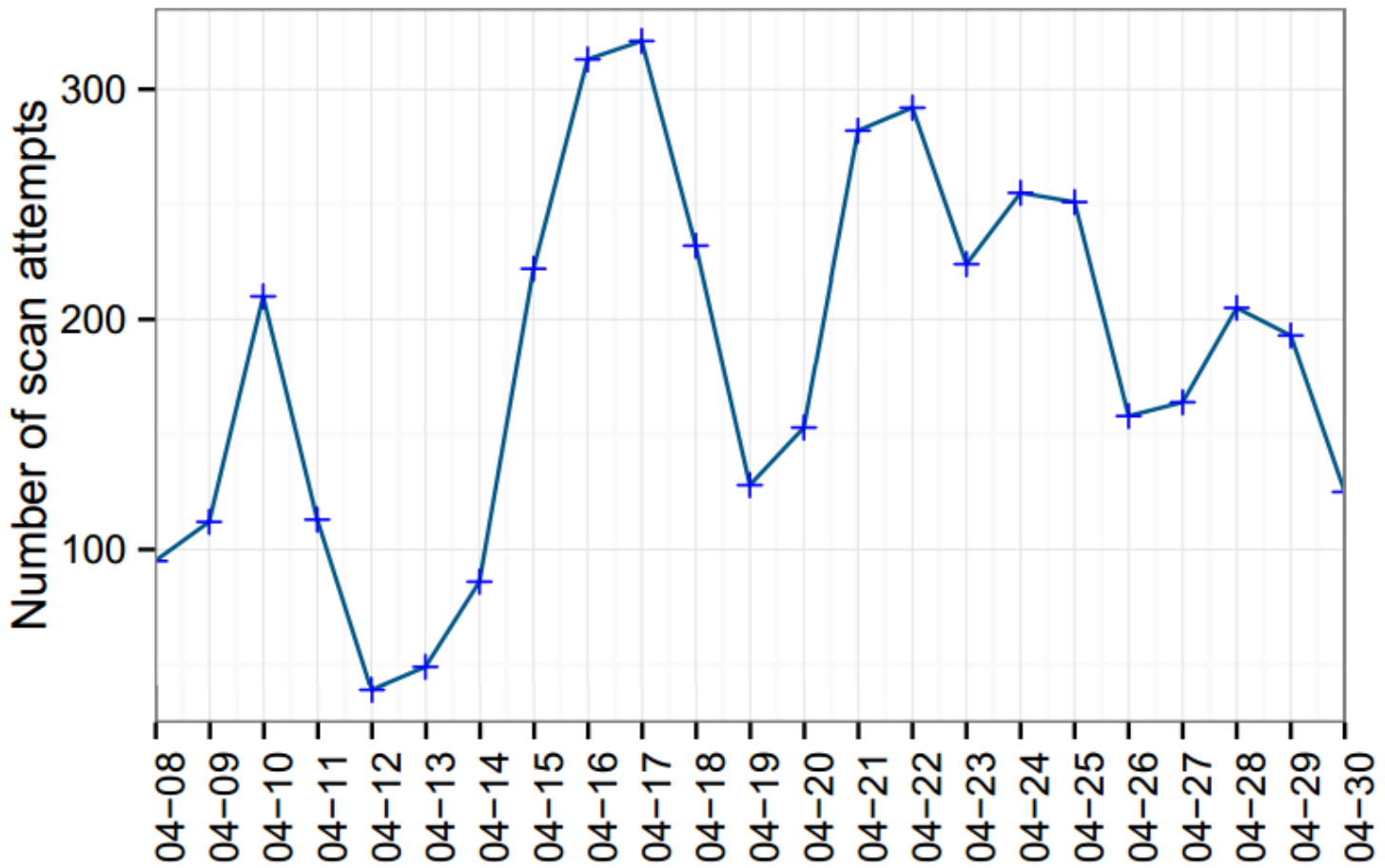
#### Web Servers

| 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 |

# Attack Attempts




In first week, 41 unique hosts scanning for Heartbleed, 59% from China.
First detected probe at 1539 GMT on April 8, 2014 – after public disclosure.

# Vulnerable TLS Certificates

Patching isn't enough. Private keys can be stolen.

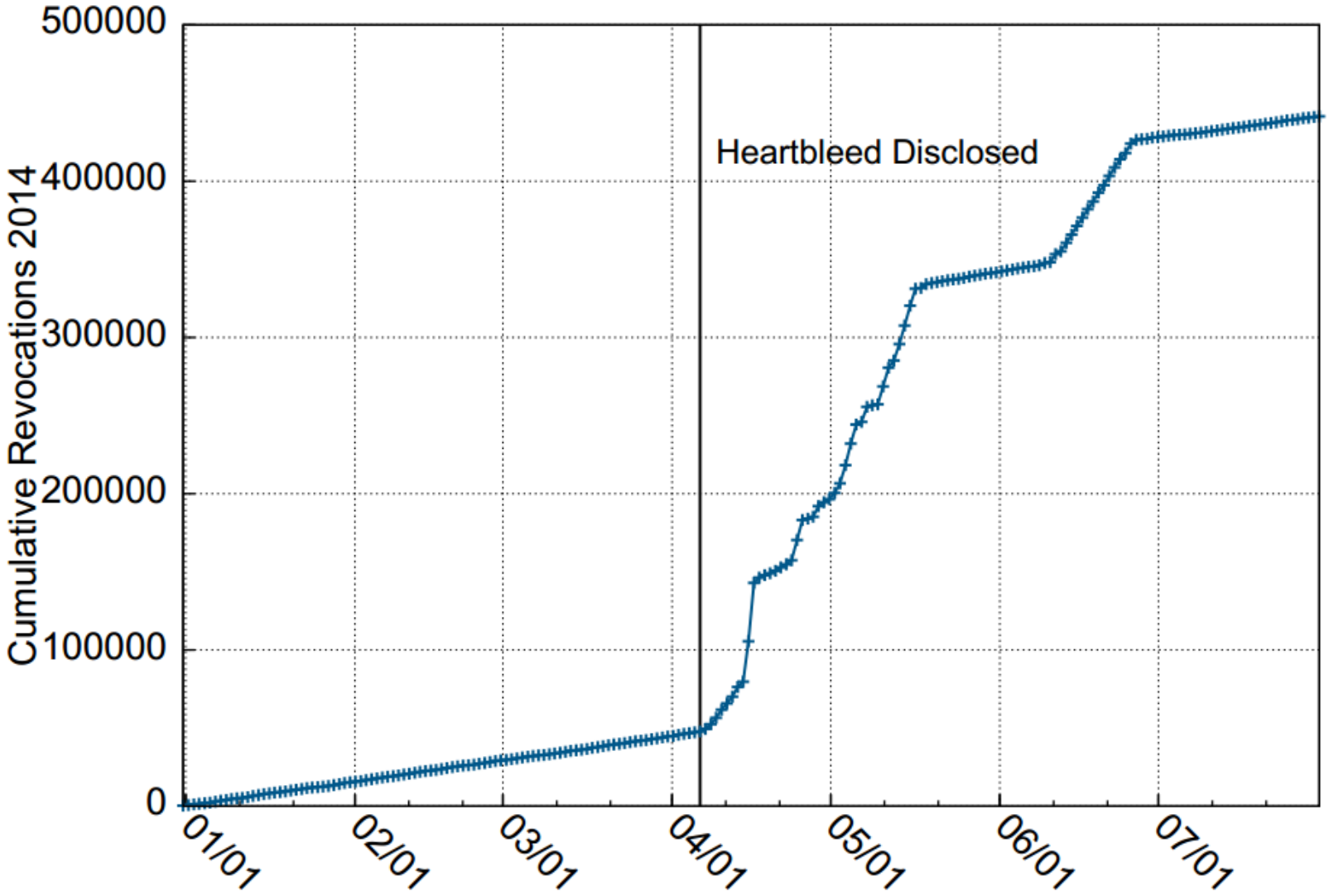
23% of Alex Top 1M replaced their TLS certificates in April!

But… only 10% of all vulnerable sites replaced their certificates.

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

… and 14% *reused the same private key!*

# Revoked TLS Certificates

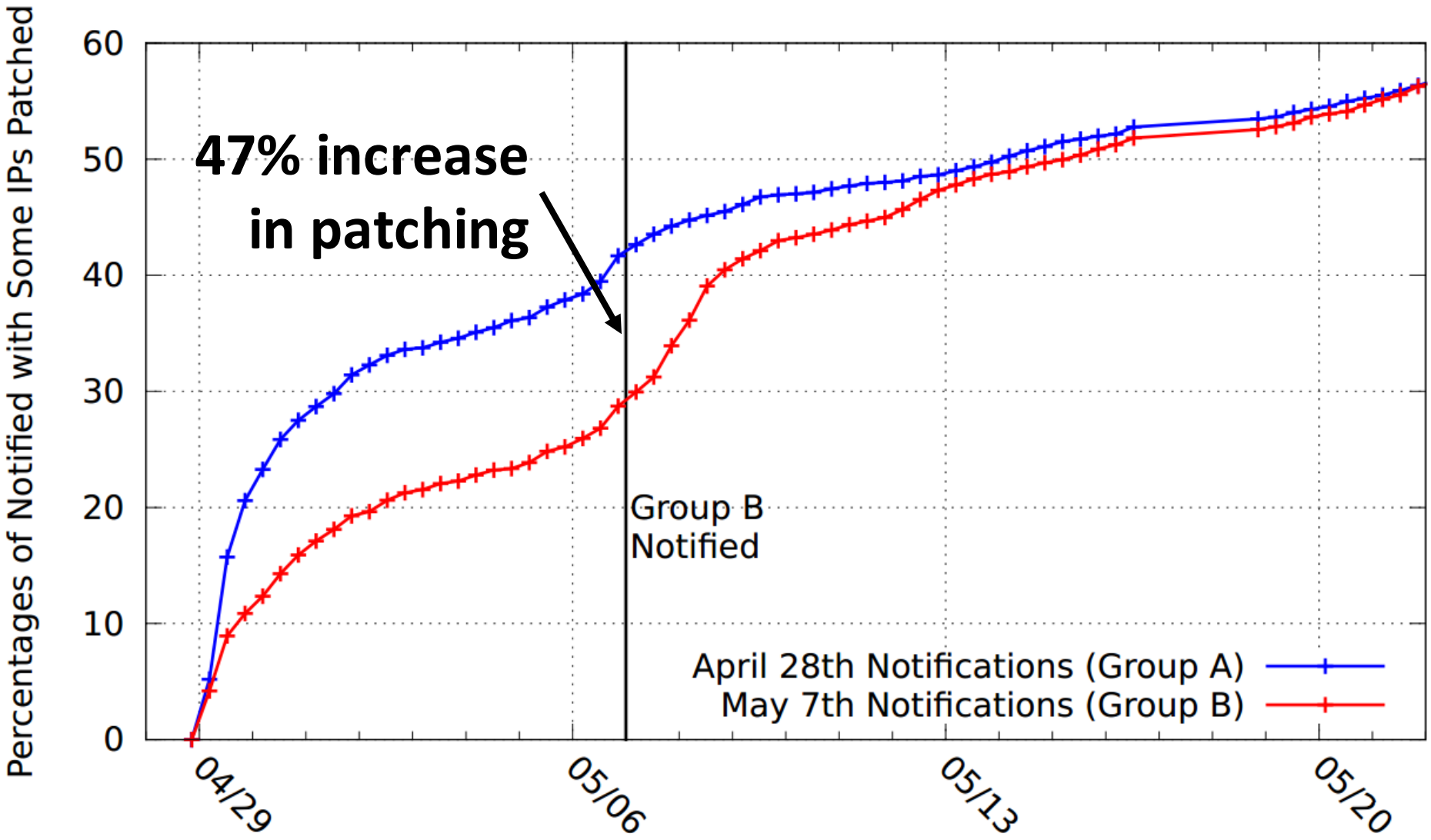


# Vulnerability Notifications

April 24 scan discovered 588,000 hosts still vulnerable.

What to do?

# Vulnerability Notifications



# Heartbleed Lessons

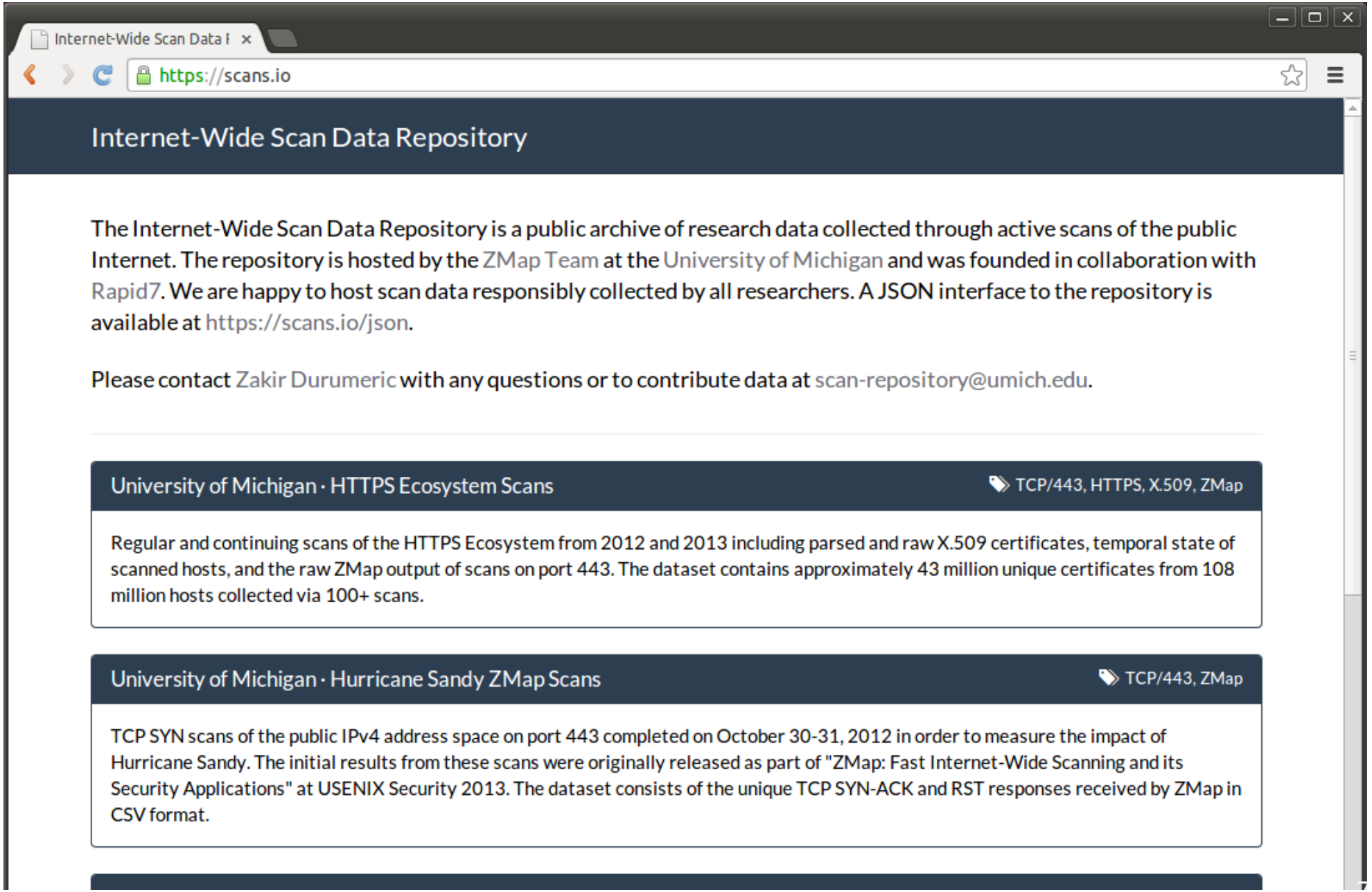
Critical software “plumbing” receives too little attention and support from the community

Coordinating disclosure for most severe vulnerabilities remains an unsolved problem

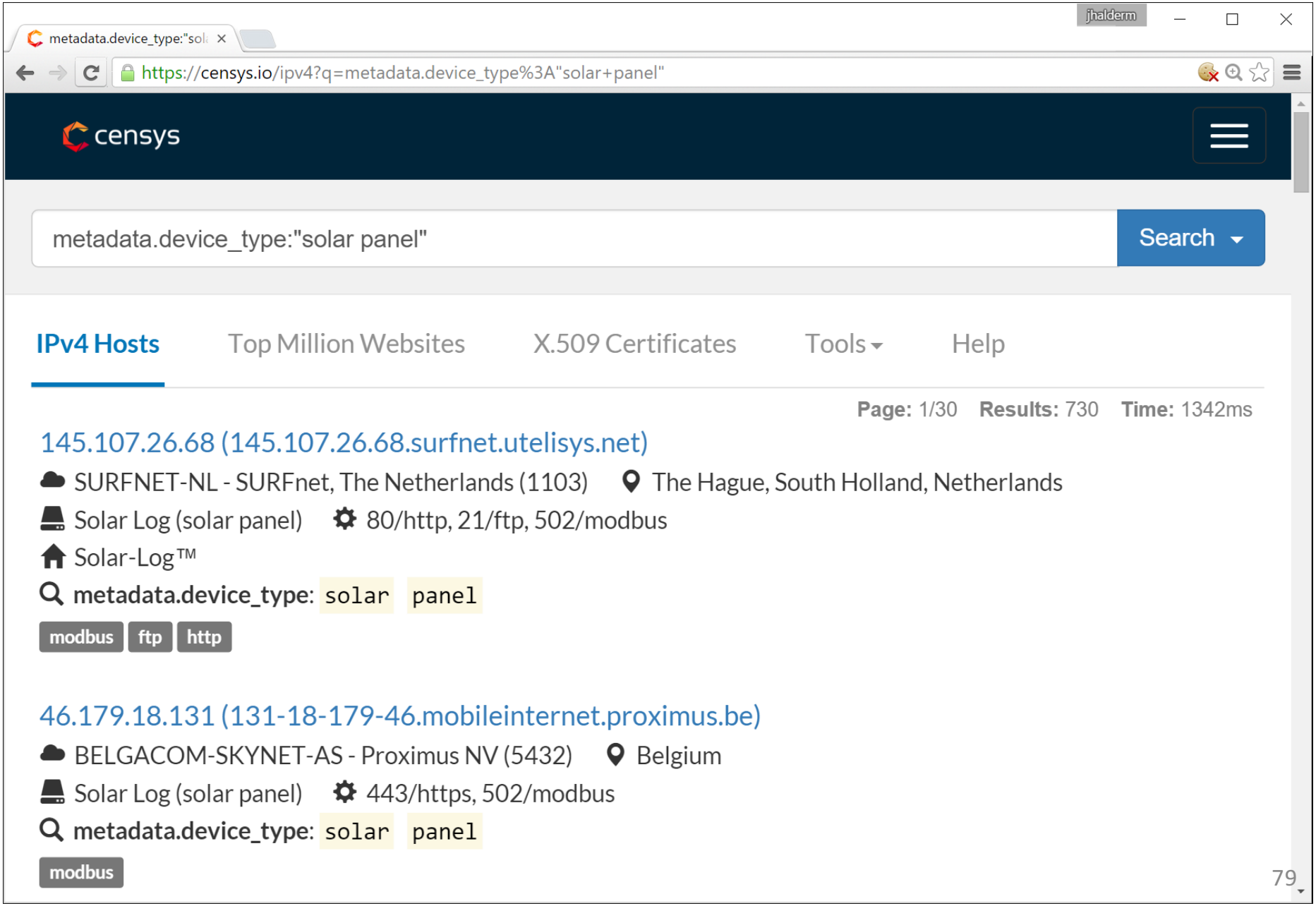
HTTPS administration suffers from high rate of human error – Don’t blame admins, it’s hard!

Measurement and notification can significantly impact patching rates, demands further study

# Getting Involved – Scans.io Data Repository



# Getting Involved – Censys Search Engine



# Learn more and experiment yourself:

ZMap Internet-wide scanner
https://**zmap.io**

Scan data repository
https://**scans.io**

Censys search engine
https://**censys.io**

Factorable key check service
https://**factorable.net**

Heartbleed health report
https://**zmap.io/heartbleed**