MALWARE: WORMS

GRAD SEC

OCT 12 2017

TODAY'S PAPERS


How to Own the Internet in Your Spare Time

Stuart Staniford* — *Silicon Defense* — stuart@silicondefense.com
Vern Paxson† — *ICSI Center for Internet Research* — vern@icir.org
Nicholas Weaver‡ — *UC Berkeley* — nweaver@cs.berkeley.edu

Abstract

The ability of attackers to rapidly gain control of vast numbers of Internet hosts poses an immense risk to the overall security of the Internet. Once subverted, these hosts can not only be used to launch massive denial of service floods, but also to steal or corrupt great quantities of sensitive information, and confuse and disrupt use of the network in more subtle ways.

We present an analysis of the magnitude of the threat. We begin with a mathematical model derived from empirical data of the spread of Code Red I in July, 2001. We discuss techniques subsequently employed for achieving greater virulence by Code Red II and Nimda. In this context, we develop and evaluate several new, highly virulent possible techniques: hit-list scanning (which creates a *Warhol* worm), permutation scanning (which enables self-coordinating scanning), and use of Internet-sized hit-lists (which creates a *flash* worm).

We then turn to the threat of *surreptitious* worms that spread more slowly but in a much harder to detect "contagion" fashion. We demonstrate that such a worm today could arguably subvert upwards of 10,000,000 Internet hosts. We also consider robust mechanisms by which attackers can control and update deployed worms.

In conclusion, we argue for the pressing need to develop a "Center for Disease Control" analog for virus- and worm-based threats to national cybersecurity, and sketch some of the components that would go into such a Center.

1 Introduction

If you can control a million hosts on the Internet, you can do enormous damage. First, you can launch distributed denial of service (DDOS) attacks so immensely diffuse that mitigating them is well beyond the state-of-the-art for DDOS traceback and protection technologies. Such attacks could readily bring down e-commerce sites, news outlets, command and coordination infrastructure, specific routers, or the root name servers.

Second, you can access any sensitive information present on any of those million machines—passwords, credit card numbers, address books, archived email, patterns of user activity, illicit content—even blindly searching for a "needle in a haystack," i.e., information that might be on a computer somewhere in the Internet, for which you trawl using a set of content keywords.

Third, not only can you access this information, but you can sow confusion and disruption by corrupting the information, or sending out false or confidential information directly from a user's desktop.

In short, if you could control a million Internet hosts, the potential damage is truly immense: on a scale where such an attack could play a significant role in warfare between nations or in the service of terrorism.

Unfortunately it is reasonable for an attacker to gain control of a million Internet hosts, or perhaps even ten million. The highway to such control lies in the exploitation of *worms*: programs that self-propagate across the Internet by exploiting security flaws in widely-used services.[1] Internet-scale worms are not a new phenomenon [Sp89, ER89], but the severity of their threat has rapidly grown with (i) the increasing degree to which the In-

*Research supported by DARPA via contract N66001-00-C-8045
†Also with the Lawrence Berkeley National Laboratory, University of California, Berkeley.
‡Additional support from Xilinx, ST Microsystems, and the California MICRO program

[1] We distinguish between the worms discussed in this paper—*active worms*—and *viruses* (or *email worms*) in that the latter require some sort of user action to abet their propagation. As such, they tend to propagate more slowly. From an attacker's perspective, they also suffer from the presence of a large anti-virus industry that actively seeks to identify and control their spread.

Inside the Slammer Worm

David Moore — *Cooperative Association for Internet Data Analysis and University of California, San Diego*
Vern Paxson — *International Computer Science Institute and Lawrence Berkeley National Laboratory*
Stefan Savage — *University of California, San Diego*
Colleen Shannon — *Cooperative Association for Internet Data Analysis*
Stuart Staniford — *Silicon Defense*
Nicholas Weaver — *Silicon Defense and University of California, Berkeley*

The Slammer worm spread so quickly that human response was ineffective. In January 2003, it packed a benign payload, but its disruptive capacity was surprising. Why was it so effective and what new challenges do this new breed of worm pose?

Slammer (sometimes called Sapphire) was the fastest computer worm in history. As it began spreading throughout the Internet, the worm infected more than 90 percent of vulnerable hosts within 10 minutes, causing significant disruption to financial, transportation, and government institutions and precluding any human-based response. In this article, we describe how it achieved its rapid growth, dissect portions of the worm to study some of its flaws, and look at our defensive effectiveness against it and its successors.

Slammer began to infect hosts slightly before 05:30 UTC on Saturday, 25 January 2003, by exploiting a buffer-overflow vulnerability in computers on the Internet running Microsoft's SQL Server or Microsoft SQL Server Desktop Engine (MSDE) 2000. David Litchfield of Next Generation Security Software discovered this underlying indexing service weakness in July 2002; Microsoft released a patch for the vulnerability before the vulnerability was publicly disclosed (www.microsoft.com/security/slammer.asp). Exploiting this vulnerability, the worm infected at least 75,000 hosts, perhaps considerably more, and caused network outages and unforeseen consequences such as canceled airline flights, interference with elections, and ATM failures (see Figure 1).

Slammer's most novel feature is its propagation speed. In approximately three minutes, the worm achieved its full scanning rate (more than 55 million scans per second), after which the growth rate slowed because significant portions of the network had insufficient bandwidth to accommodate more growth.

Although Stuart Staniford, Vern Paxson, and Nicholas Weaver had predicted rapid-propagation worms on theoretical grounds,[1] Slammer provided the first real-world demonstration of a high-speed worm's capabilities. By comparison, Slammer was two orders of magnitude faster than the Code Red worm, which infected more than 359,000 hosts on 19 July 2001,[2] and had a leisurely 37 minutes of population doubling time.

While Slammer had no malicious payload, it caused considerable harm by overloading networks and disabling database servers. Many sites lost connectivity as local copies of the worm saturated their access bandwidths. Although most backbone providers appeared to remain stable throughout the epidemic, there were several reports of Internet backbone disruption. For a single snapshot of the activity, see www.digitaloffense.net/worms/mssql_udp_worm/internet_health.jpg. Additionally, Tim Griffin of AT&T Research, has plotted Internet routing data (an overall view of Internet routing behavior) that shows a substantial perturbation in network connectivity resulting from Slammer's spread (www.research.att.com/~griffin/bgp_monitor/sql_worm.html).

If the worm had carried a malicious payload, attacked a more widespread vulnerability, or targeted a more popular service, its effects would likely have been far more severe.

For more on how we tracked Slammer, see the "Network telescope operations" sidebar.

How Slammer chooses its victims

The worm's spreading strategy uses random scanning—it randomly selects IP addresses, eventually finding and infecting all susceptible hosts. Random-scanning worms initially spread exponentially, but their rapid new-host

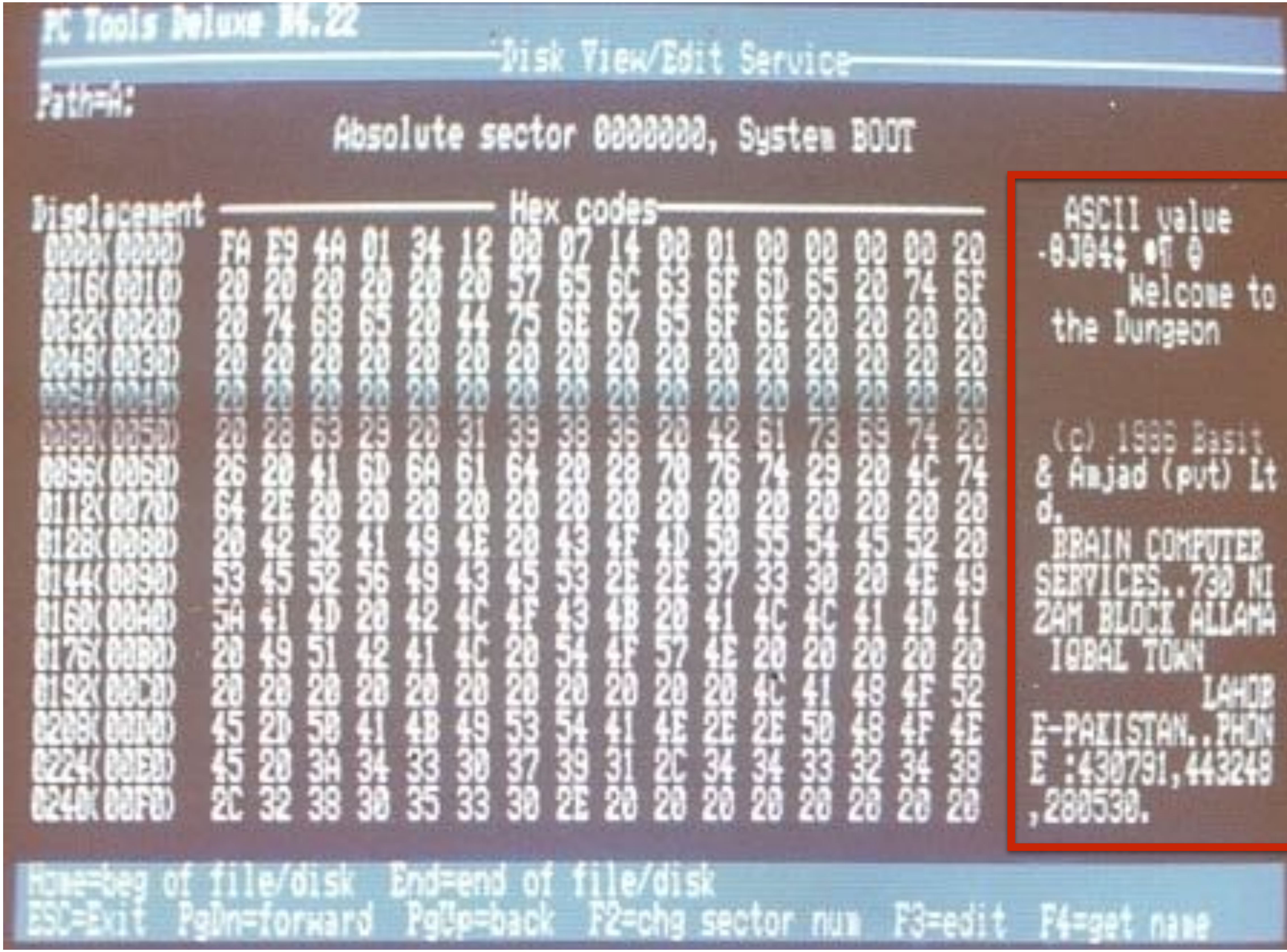
VIRUS
CASE STUDIES

BRAIN

First IBM PC virus (1987)

- Propagation method
  - Copies itself into the boot sector
  - Tells the OS that all of the boot sector is “faulty” (so that it won’t list contents to the user)
    - Thus also one of the first examples of a **stealth** virus
  - Intercepts disk read requests for 5.25” floppy drives
    - Sees if the 5th and 6th bytes of the boot sector are 0x1234
    - If so, then it’s already infected, otherwise, infect it

- Payload:
  - Nothing really; goal was just to spread (to show off?)
  - However, it served as the template for future viruses


PC Tools Deluxe R4.22
Disk View/Edit Service
Path=A:
Absolute sector 0000000, System BOOT
Displacement
Hex codes
ASCII value
0000(0000) FA E9 4A 01 34 12 00 07 14 00 01 00 00 00 00 20
0016(0010) 20 20 20 20 20 20 57 65 6C 63 6F 6D 65 20 74 6F
0032(0020) 20 74 68 65 20 44 75 6E 67 65 6F 6E 20 20 20 20
0048(0030) 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0064(0040) 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0080(0050) 20 28 63 29 20 31 39 38 36 20 42 61 73 69 74 20
0096(0060) 26 20 41 6D 6A 61 64 20 28 70 76 74 29 20 4C 74
0112(0070) 64 2E 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0128(0080) 20 42 52 41 49 4E 20 43 4F 4D 50 55 54 45 52 20
0144(0090) 53 45 52 56 49 43 45 53 2E 2E 37 33 30 20 4E 49
0160(00A0) 5A 41 4D 20 42 4C 4F 43 4B 20 41 4C 4C 41 4D 41
0176(00B0) 20 49 51 42 41 4C 20 54 4F 57 4E 20 20 20 20 20
0192(00C0) 20 20 20 20 20 20 20 20 20 20 20 4C 41 48 4F 52
0208(00D0) 45 2D 50 41 4B 49 53 54 41 4E 2E 2E 50 48 4F 4E
0224(00E0) 45 20 3A 34 33 30 37 39 31 2C 34 34 33 32 34 38
0240(00F0) 2C 32 38 30 35 33 30 2E 20 20 20 20 20 20 20 20
Welcome to the Dungeon
(c) 1986 Basit & Amjad (pvt) Ltd.
BRAIN COMPUTER SERVICES..730 NIZAM BLOCK ALLAMA IQBAL TOWN
LAHORE-PAKISTAN..PHONE :430791,443248,280530.
Home=beg of file/disk End=end of file/disk
ESC=Exit PgDn=forward PgUp=back F2=chg sector num F3=edit F4=get name

ROOTKITS

Malicious code that hides from discovery

- Ways to hide:
  - By intercepting system calls, patching the kernel, etc.
  - Often effectively done by a man in the middle attack
- Rootkit revealer: analyzes the disk offline and through the online system calls, and compares
- Mark Russinovich ran a rootkit revealer and found a rootkit in 2005…

SONY XCP ROOTKIT

Detected 2005

SONY XCP ROOTKIT

Detected 2005

- Goal: keep users from copying copyrighted material

SONY XCP ROOTKIT

Detected 2005

- Goal: keep users from copying copyrighted material
- How it worked:
  - Loaded thanks to autorun.exe on the CD
  - Intercepted read requests for its music files
  - If anyone but Sony's music player is accessing them, then garble the data
  - Hid itself from the user (to avoid deletion)

SONY XCP ROOTKIT

Detected 2005

- Goal: keep users from copying copyrighted material

- How it worked:
  - Loaded thanks to autorun.exe on the CD
  - Intercepted read requests for its music files
  - If anyone but Sony's music player is accessing them, then garble the data
  - Hid itself from the user (to avoid deletion)

- How it messed up
  - Morally: violated trust
  - Technically: Hid ***all files*** that started with "sys"
  - Seriously?: The uninstaller did not check the integrity of the code it downloaded, and would not delete it afterwords.

STUXNET

June 2010

- Virus in that it initially spread by infected USB stick
  - Once inside a network, it acted as a worm, spreading quickly
- Exploited **four** zero-day exploits
  - Zero-day: Known to only the attacker until the attack
  - Typically, one zero-day is enough to profit
  - Four was unprecedented
    - Immense cost and sophistication on behalf of the attacker
- Rootkit: installed *signed* device drivers
  - Thereby avoiding user alert when installing
  - Signed with certificates stolen from two Taiwanese CAs

STUXNET: PAYLOAD

STUXNET: PAYLOAD

- Do nothing

STUXNET: PAYLOAD

- Do nothing
- Unless attached to particular models of frequency converter drives that operate at 807-1210Hz

STUXNET: PAYLOAD

- Do nothing
- Unless attached to particular models of frequency converter drives that operate at 807-1210Hz
    - You know, like those in Iran and Finland

STUXNET: PAYLOAD

- Do nothing
- Unless attached to particular models of frequency converter drives that operate at 807-1210Hz
    - You know, like those in Iran and Finland
    - .. those ones that are used to operate centrifuges

STUXNET: PAYLOAD

- Do nothing
- Unless attached to particular models of frequency converter drives that operate at 807-1210Hz
  - You know, like those in Iran and Finland
  - .. those ones that are used to operate centrifuges
  - .. for producing enriched uranium for nuclear weapons

STUXNET: PAYLOAD

- Do nothing
- Unless attached to particular models of frequency converter drives that operate at 807-1210Hz
  - You know, like those in Iran and Finland
  - .. those ones that are used to operate centrifuges
  - .. for producing enriched uranium for nuclear weapons
- In which case, slowly increase the freq to 1410Hz

STUXNET: PAYLOAD

- Do nothing

- Unless attached to particular models of frequency converter drives that operate at 807-1210Hz
  - You know, like those in Iran and Finland
  - .. those ones that are used to operate centrifuges
  - .. for producing enriched uranium for nuclear weapons

- In which case, slowly increase the freq to 1410Hz
  - You know, enough to break the centrifuge

STUXNET: PAYLOAD

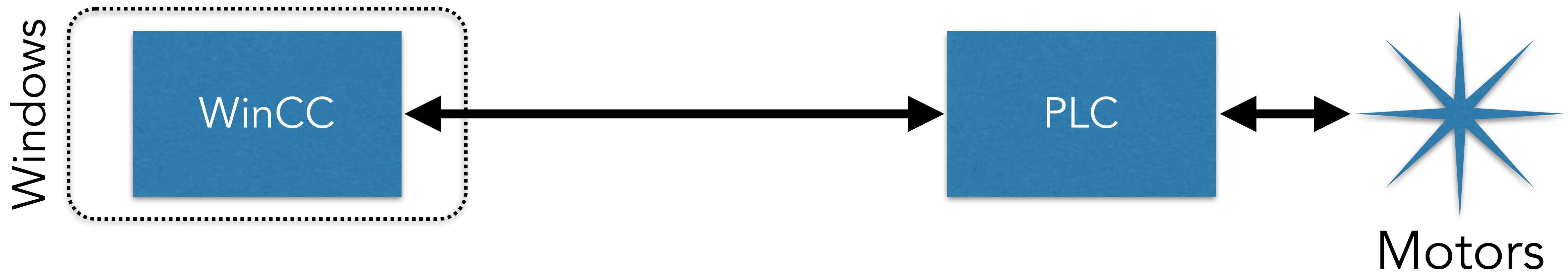
- Do nothing
- Unless attached to particular models of frequency converter drives that operate at 807-1210Hz
  - You know, like those in Iran and Finland
  - .. those ones that are used to operate centrifuges
  - .. for producing enriched uranium for nuclear weapons
- In which case, slowly increase the freq to 1410Hz
  - You know, enough to break the centrifuge
  - .. all the while sending “looks good to me” readings to the user

STUXNET: PAYLOAD

- Do nothing
- Unless attached to particular models of frequency converter drives that operate at 807-1210Hz
    - You know, like those in Iran and Finland
    - .. those ones that are used to operate centrifuges
    - .. for producing enriched uranium for nuclear weapons
- In which case, slowly increase the freq to 1410Hz
    - You know, enough to break the centrifuge
    - .. all the while sending “looks good to me” readings to the user
    - .. then drop back to normal range

STUXNET: PAYLOAD

- Targets industrial control systems by overwriting programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



- In reality, it sped up and slowed down the motors
- Result: Destroy (or at least decrease the productivity of) nuclear centrifuges

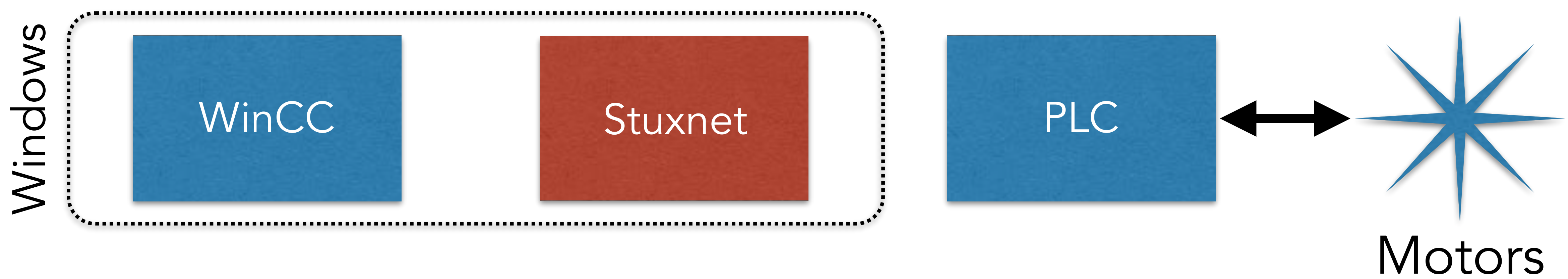
STUXNET: PAYLOAD

- Targets industrial control systems by overwriting programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator

- In reality, it sped up and slowed down the motors
- Result: Destroy (or at least decrease the productivity of) nuclear centrifuges

STUXNET: PAYLOAD

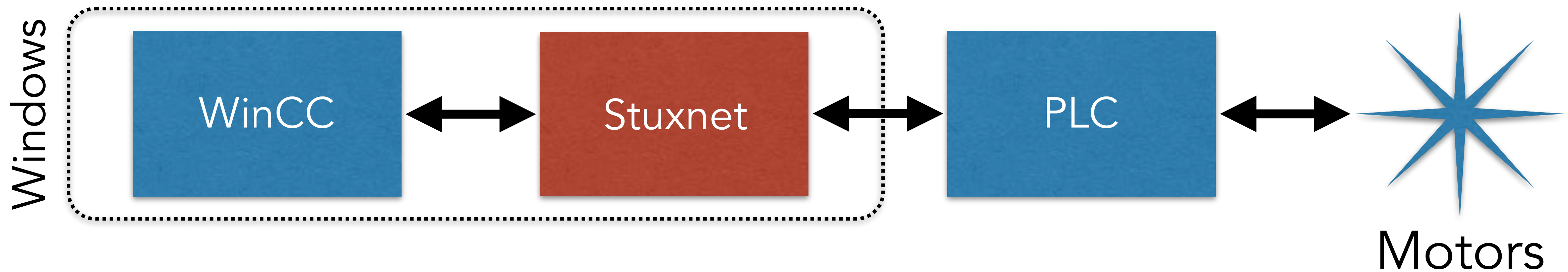
- Targets industrial control systems by overwriting programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



- In reality, it sped up and slowed down the motors
- Result: Destroy (or at least decrease the productivity of) nuclear centrifuges

STUXNET: PAYLOAD

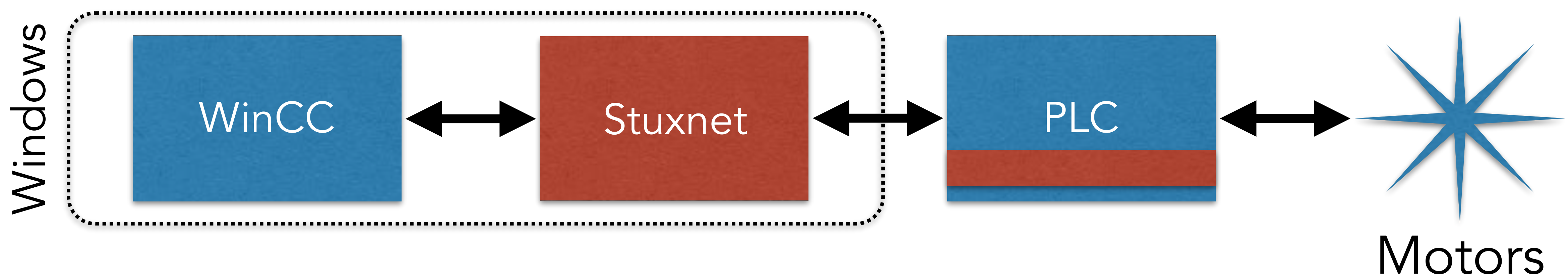
- Targets industrial control systems by overwriting programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



- In reality, it sped up and slowed down the motors
- Result: Destroy (or at least decrease the productivity of) nuclear centrifuges

STUXNET: PAYLOAD

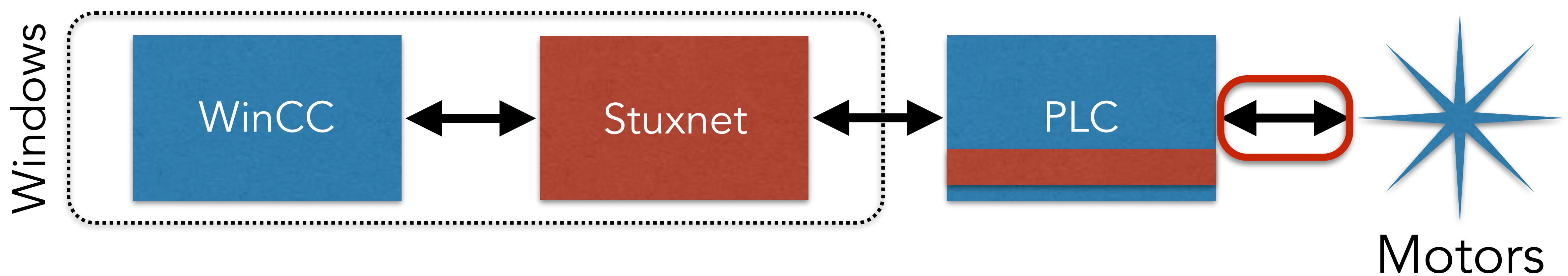
- Targets industrial control systems by overwriting programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



- In reality, it sped up and slowed down the motors
- Result: Destroy (or at least decrease the productivity of) nuclear centrifuges

STUXNET: PAYLOAD

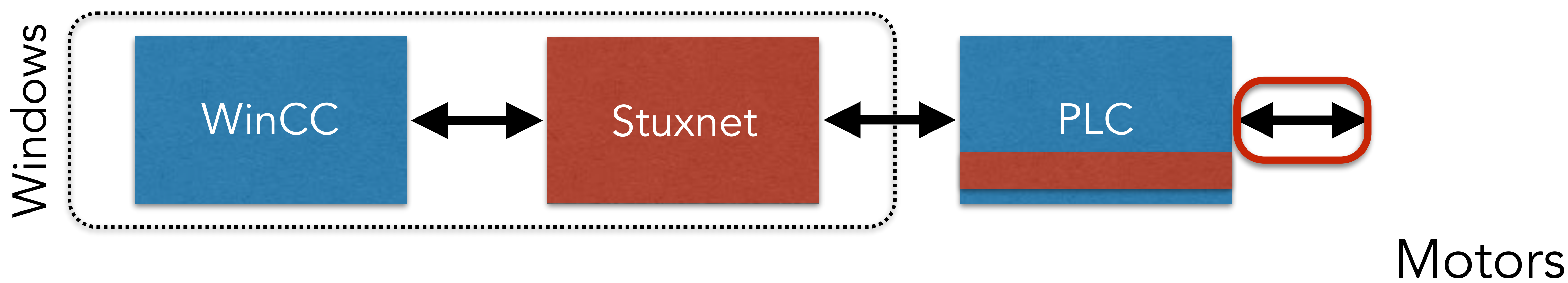
- Targets industrial control systems by overwriting programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



- In reality, it sped up and slowed down the motors
- Result: Destroy (or at least decrease the productivity of) nuclear centrifuges

STUXNET: PAYLOAD

- Targets industrial control systems by overwriting programmable logic boards
- Man-in-the-middle between Windows and Siemens control systems; looked like it was working properly to the operator



- In reality, it sped up and slowed down the motors
- Result: Destroy (or at least decrease the productivity of) nuclear centrifuges

STUXNET FALLOUT

- Iran denied they had been hit by Stuxnet
- Then claimed they were, but had contained it
- Understood now that it took out 1k of Iran’s 5k centrifuges
- Security experts believe the U.S. did it (possibly along with Israel) due to its sophistication and cost
- **Legitimized cyber warfare**

VIRUSES: SUMMARY

- Technological arms race between those who wish to detect and those who wish to evade detection
- Started off innocuously, capable by only a few very clever people
- But viruses have become commoditized; any scriptkiddy can launch one (creation remains hard)
- No longer purely of academic interest
  - Economic pursuits (zero-day markets)
  - Cyber warfare

OTHER WORK

- Detecting malware in the Android app store
- Lots of drive-by-download work
- Malware distribution networks: use enterprise-wide network traces to detect malware downloads
- Side-channel defenses: Measure, e.g., power consumption of benign vs. malicious code
- Metamorphic arms race

10/10 Malware

- **Hunting For Metamorphic**, Péter Ször, Peter Ferrie
- **The Ghost In The Browser Analysis of Web-based Malware**, Niels Provos, Dean McNamee, Panayiotis Mavrommatis, Ke Wang, Nagendra Modadugu

- Dissecting Android Malware: Characterization and Evolution, Yajin Zhou, Xuxian Jiang
- Hey, you, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets, Yajin Zhou, Zhi Wang, Wu Zhou, Xuxian Jiang
- All Your iFrames Point to Us, Niels Provos, Panayiotis Mavrommatis, Moheeb Abu Rajab, Fabian Monrose
- Android Permissions Demystified, Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, David Wagner
- Prudent Practices for Designing Malware Experiments: Status Quo and Outlook, Christian Rossow, Christian J. Dietrich, Chris Grier, Christian Kreibich, Vern Paxson, Norbert Pohlmann, Herbert Bos, Maarten van Steen
- Detection and Analysis of Drive-by-Download Attacks and Malicious JavaScript Code, Marco Cova, Christopher Kruegel, Giovanni Vigna
- Towards Automatic Generation of Vulnerability-Based Signatures, David Brumley, James Newsome, Dawn Song, Hao Wang, Somesh Jha
- Nazca: Detecting Malware Distribution in Large-Scale Networks, Luca Invernizzi, Stanislav Miskovic, Ruben Torres, Sabyasachi Saha, Sung-Ju Lee, Marco Mellia, Christopher Kruegel, Giovanni Vigna
- WattsUpDoc: Power Side Channels to Nonintrusively Discover Untargeted Malware on Embedded Medical Devices, Shane S. Clark, Benjamin Ransford, Amir Rahmati, Shane Guineau, Jacob Sorber, Kevin Fu, Wenyuan Xu
- Sony's DRM Rootkit: The Real Story, Bruce Schneier
- Lessons from the Sony CD DRM Episode, J. Alex Halderman, Edward W. Felten

DETECTING METAMORPHIC VIRUSES

New idea: side-channel metamorphic detection

- Measure some effect of the system
  - Power consumption
  - Sounds during computation

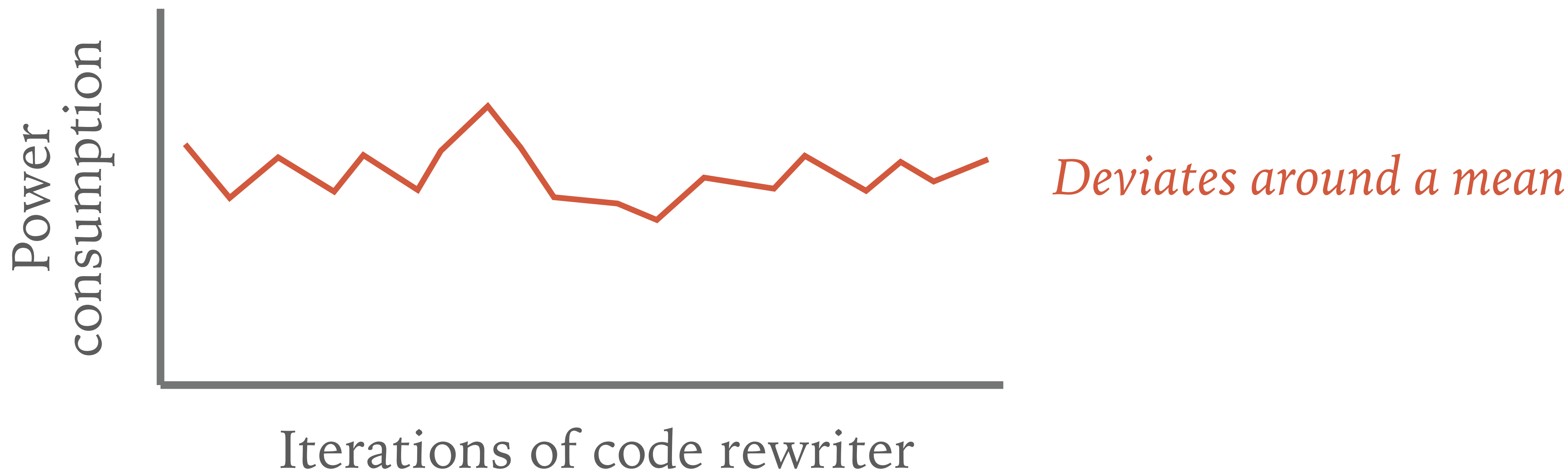
- Compare that to normal operation
  - Raise alarm when different

DETECTING METAMORPHIC VIRUSES

New idea: side-channel metamorphic detection

- Measure some effect of the system
  - Power consumption
  - Sounds during computation

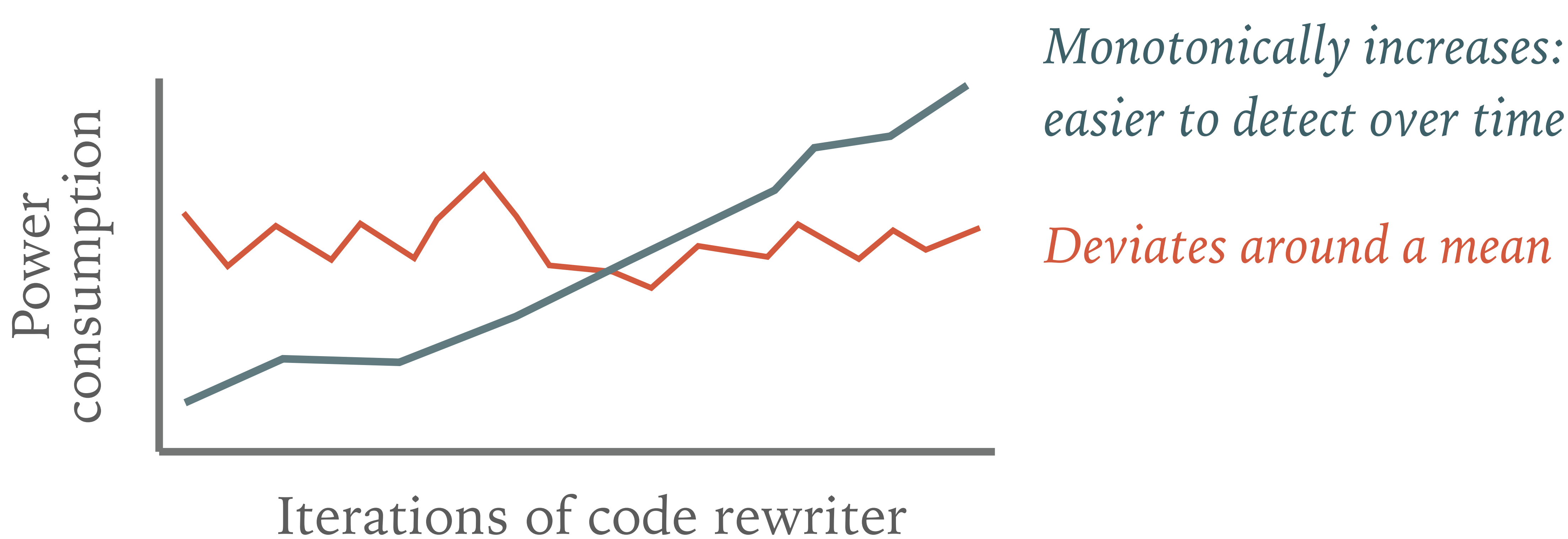
- Compare that to normal operation
  - Raise alarm when different



DETECTING METAMORPHIC VIRUSES

New idea: side-channel metamorphic detection

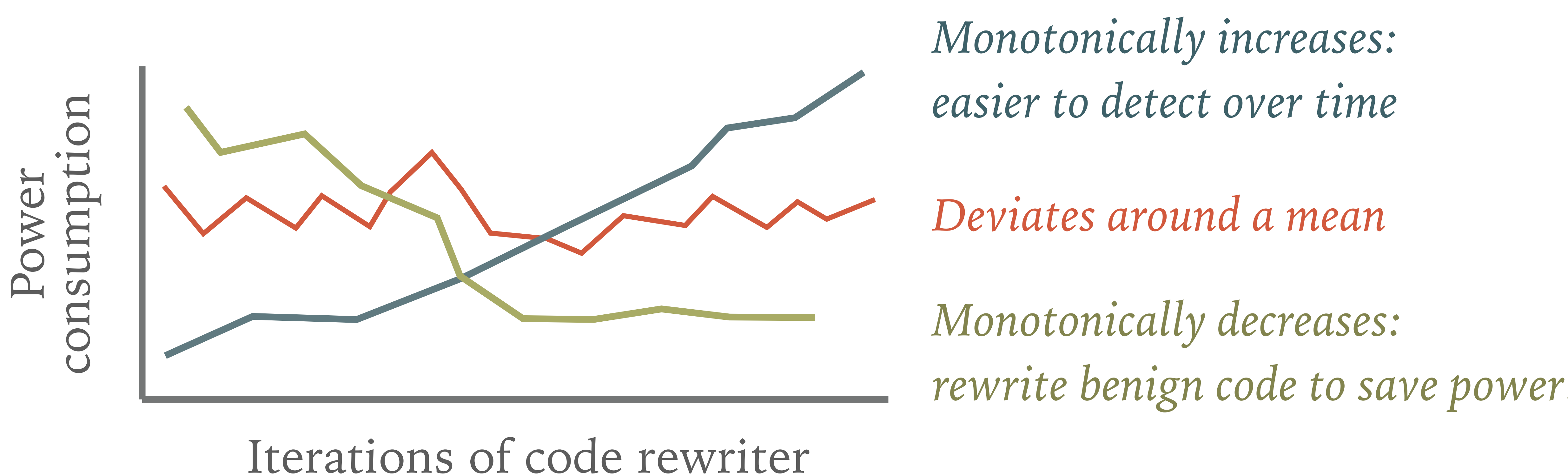
- Measure some effect of the system
  - Power consumption
  - Sounds during computation
- Compare that to normal operation
  - Raise alarm when different



DETECTING METAMORPHIC VIRUSES

New idea: side-channel metamorphic detection

- Measure some effect of the system
  - Power consumption
  - Sounds during computation
- Compare that to normal operation
  - Raise alarm when different



TODAY'S PAPERS

How to Own the Internet in Your Spare Time

Stuart Staniford* — Silicon Defense — stuart@silicondefense.com
Vern Paxson† — ICSI Center for Internet Research — vern@icir.org
Nicholas Weaver‡ — UC Berkeley — nweaver@cs.berkeley.edu

Abstract

The ability of attackers to rapidly gain control of vast numbers of Internet hosts poses an immense risk to the overall security of the Internet. Once subverted, these hosts can not only be used to launch massive denial of service floods, but also to steal or corrupt great quantities of sensitive information, and confuse and disrupt use of the network in more subtle ways.

We present an analysis of the magnitude of the threat. We begin with a mathematical model derived from empirical data of the spread of Code Red I in July, 2001. We discuss techniques subsequently employed for achieving greater virulence by Code Red II and Nimda. In this context, we develop and evaluate several new, highly virulent possible techniques: hit-list scanning (which creates a *Warhol* worm), permutation scanning (which enables self-coordinating scanning), and use of Internet-sized hit-lists (which creates a *flash* worm).

We then turn to the to the threat of *surreptitious* worms that spread more slowly but in a much harder to detect "contagion" fashion. We demonstrate that such a worm today could arguably subvert upwards of 10,000,000 Internet hosts. We also consider robust mechanisms by which attackers can control and update deployed worms.

In conclusion, we argue for the pressing need to develop a "Center for Disease Control" analog for virus- and worm-based threats to national cybersecurity, and sketch some of the components that would go into such a Center.

1 Introduction

If you can control a million hosts on the Internet, you can do enormous damage. First, you can launch distributed denial of service (DDOS) attacks so immensely diffuse that mitigating them is well beyond the state-of-the-art for DDOS traceback and protection technologies. Such attacks could readily bring down e-commerce sites, news outlets, command and coordination infrastructure, specific routers, or the root name servers.

Second, you can access any sensitive information present on any of those million machines—passwords, credit card numbers, address books, archived email, patterns of user activity, illicit content—even blindly searching for a "needle in a haystack," i.e., information that might be on a computer somewhere in the Internet, for which you trawl using a set of content keywords.

Third, not only can you access this information, but you can sow confusion and disruption by corrupting the information, or sending out false or confidential information directly from a user's desktop.

In short, if you could control a million Internet hosts, the potential damage is truly immense: on a scale where such an attack could play a significant role in warfare between nations or in the service of terrorism.

Unfortunately it is reasonable for an attacker to gain control of a million Internet hosts, or perhaps even ten million. The highway to such control lies in the exploitation of *worms*: programs that self-propagate across the Internet by exploiting security flaws in widely-used services.[1] Internet-scale worms are not a new phenomenon [Sp89, ER89], but the severity of their threat has rapidly grown with (i) the increasing degree to which the In-

*Research supported by DARPA via contract N66001-00-C-8045
†Also with the Lawrence Berkeley National Laboratory, University of California, Berkeley.
‡Additional support from Xilinx, ST Microsystems, and the California MICRO program

[1] We distinguish between the worms discussed in this paper—active worms—and viruses (or email worms) in that the latter require some sort of user action to abet their propagation. As such, they tend to propagate more slowly. From an attacker's perspective, they also suffer from the presence of a large anti-virus industry that actively seeks to identify and control their spread.

Inside the Slammer Worm

David Moore — Cooperative Association for Internet Data Analysis and University of California, San Diego
Vern Paxson — International Computer Science Institute and Lawrence Berkeley National Laboratory
Stefan Savage — University of California, San Diego
Colleen Shannon — Cooperative Association for Internet Data Analysis
Stuart Staniford — Silicon Defense
Nicholas Weaver — Silicon Defense and University of California, Berkeley

The Slammer worm spread so quickly that human response was ineffective. In January 2003, it packed a benign payload, but its disruptive capacity was surprising. Why was it so effective and what new challenges do this new breed of worm pose?

Slammer (sometimes called Sapphire) was the fastest computer worm in history. As it began spreading throughout the Internet, the worm infected more than 90 percent of vulnerable hosts within 10 minutes, causing significant disruption to financial, transportation, and government institutions and precluding any human-based response. In this article, we describe how it achieved its rapid growth, dissect portions of the worm to study some of its flaws, and look at our defensive effectiveness against it and its successors.

Slammer began to infect hosts slightly before 05:30 UTC on Saturday, 25 January 2003, by exploiting a buffer-overflow vulnerability in computers on the Internet running Microsoft's SQL Server or Microsoft SQL Server Desktop Engine (MSDE) 2000. David Litchfield of Next Generation Security Software discovered this underlying indexing service weakness in July 2002; Microsoft released a patch for the vulnerability before the vulnerability was publicly disclosed (www.microsoft.com/security/slammer.asp). Exploiting this vulnerability, the worm infected at least 75,000 hosts, perhaps considerably more, and caused network outages and unforeseen consequences such as canceled airline flights, interference with elections, and ATM failures (see Figure 1).

Slammer's most novel feature is its propagation speed. In approximately three minutes, the worm achieved its full scanning rate (more than 55 million scans per second), after which the growth rate slowed because significant portions of the network had insufficient bandwidth to accommodate more growth.

Although Stuart Staniford, Vern Paxson, and Nicholas Weaver had predicted rapid-propagation worms on theoretical grounds,[1] Slammer provided the first real-world demonstration of a high-speed worm's capabilities. By comparison, Slammer was two orders of magnitude faster than the Code Red worm, which infected more than 359,000 hosts on 19 July 2001,[2] and had a leisurely 37 minutes of population doubling time.

While Slammer had no malicious payload, it caused considerable harm by overloading networks and disabling database servers. Many sites lost connectivity as local copies of the worm saturated their access bandwidths. Although most backbone providers appeared to remain stable throughout the epidemic, there were several reports of Internet backbone disruption. For a single snapshot of the activity, see www.digitaloffense.net/worms/mssql_udp_worm/internet_health.jpg. Additionally, Tim Griffin of AT&T Research, has plotted Internet routing data (an overall view of Internet routing behavior) that shows a substantial perturbation in network connectivity resulting from Slammer's spread (www.research.att.com/~griffin/bgp_monitor/sql_worm.html). If the worm had carried a malicious payload, attacked a more widespread vulnerability, or targeted a more popular service, its effects would likely have been far more severe.

For more on how we tracked Slammer, see the "Network telescope operations" sidebar.

How Slammer chooses its victims

The worm's spreading strategy uses random scanning—it randomly selects IP addresses, eventually finding and infecting all susceptible hosts. Random-scanning worms initially spread exponentially, but their rapid new-host

WORM PROPAGATION

CONTROLLING MILLIONS OF HOSTS: WHY?

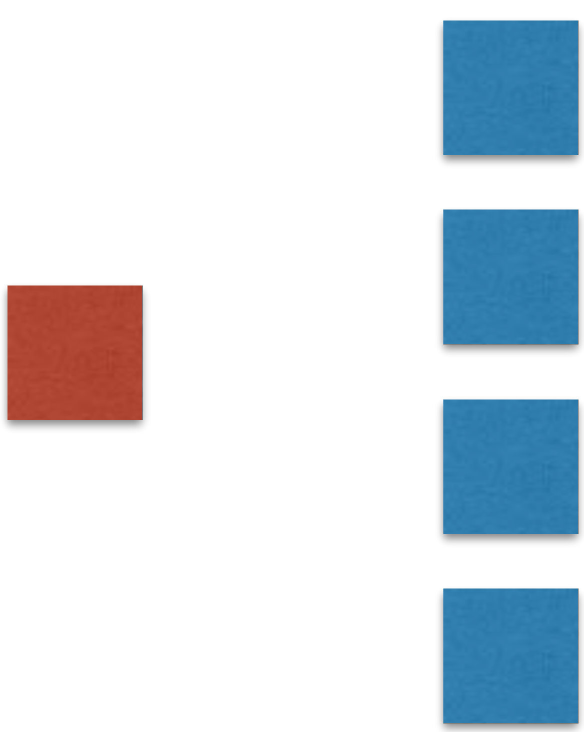
- Distributed Denial of Service (DDoS)
  - Generate network traffic from many sources..
  - .. to a single destination
  - .. with the intention of overloading their network
    - Consume too many resources for legitimate users to also use
- Steal sensitive information from millions of others
  - Even a small fraction of unprotected people ⇒ $
- Confuse and disrupt

CONTROLLING MILLIONS OF HOSTS: HOW?

- Worm: *self*-propagates by arranging to have itself *immediately* executed
  - At which point it creates a new, additional instance of itself
- Typically infects by altering *running* code
  - No user intervention required
- Like viruses, propagation and payload are orthogonal

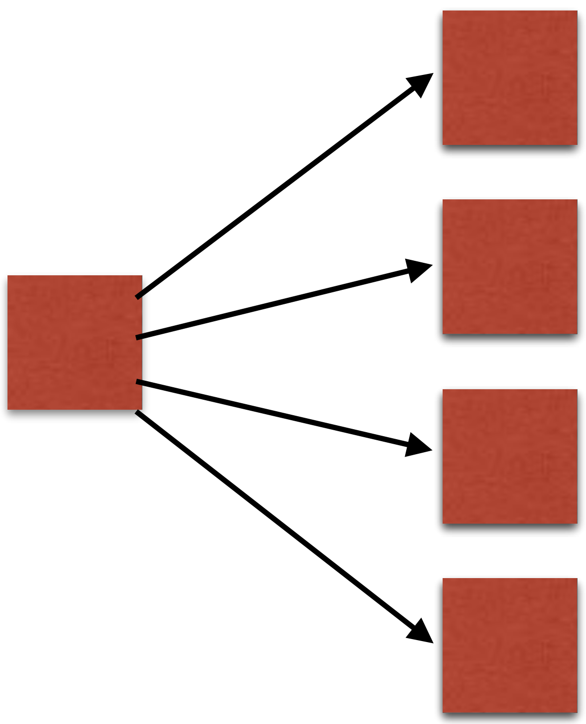
SELF PROPAGATION

- The goal is to spread as quickly as possible
- The key is ***parallelization***
    - Ditto for viruses, but they require human interaction to trigger each propagation



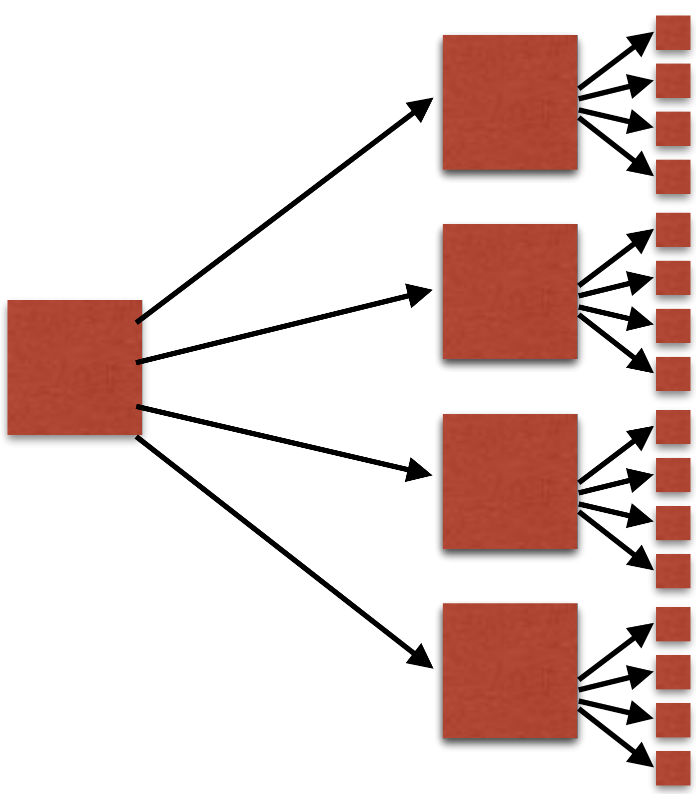
SELF PROPAGATION

- The goal is to spread as quickly as possible
- The key is ***parallelization***
  - Ditto for viruses, but they require human interaction to trigger each propagation



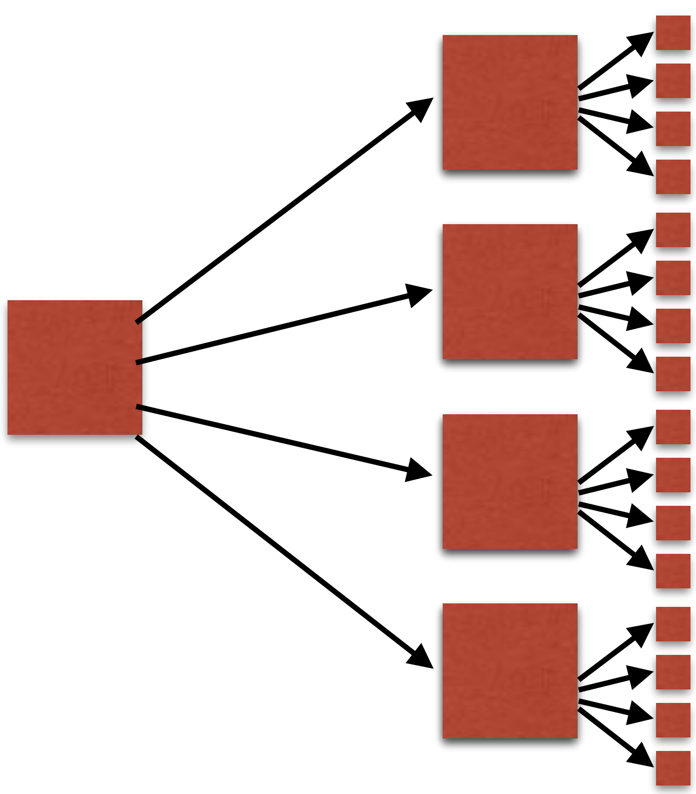
SELF PROPAGATION

- The goal is to spread as quickly as possible
- The key is ***parallelization***
  - Ditto for viruses, but they require human interaction to trigger each propagation



SELF PROPAGATION

- The goal is to spread as quickly as possible
- The key is ***parallelization***
  - Ditto for viruses, but they require human interaction to trigger each propagation



Propagation

(1) Targeting: how does the worm **find new prospective victims**?

(2) Exploit: how does the worm get code to **automatically run**?

WORMS: A BRIEF HISTORY

First arrival

1988 | 1999 | 2000 | 2001 | 2002 | 2003

- Morris worm
  - Propagated across machines (too aggressively, thanks to a bug)
  - One way it propagated was a **buffer overflow attack** against a vulnerable version of fingerd on VAXes
    - Sent a special string to the finger daemon, which caused it to execute code that created a new worm copy
    - Didn’t check OS: caused Suns running BSD to crash
  - End result: $10-100M in damages, probation, community service

WORMS: A BRIEF HISTORY

First arrival

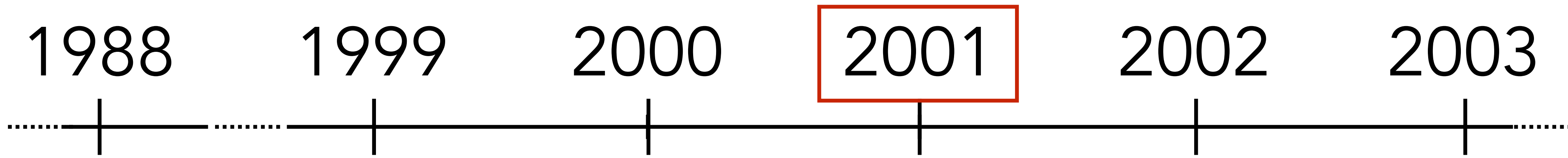
1988 | 1999 | 2000 | 2001 | 2002 | 2003

- Morris worm
  - Propagated across machines (too aggressively, thanks to a bug)
  - One way it propagated was a **buffer overflow attack** against a vulnerable version of fingerd on VAXes
    - Sent a special string to the finger daemon, which caused it to execute code that created a new worm copy
    - Didn't check OS: caused Suns running BSD to crash
  - End result: $10-100M in damages, probation, community service

Robert Morris is now a professor at MIT

WORMS: A BRIEF HISTORY

Introduction of "modern day" worms



- CodeRed
  - Propagation: Exploited an overflow in the MS-IIS server
  - Payload 1: website defacement
    - HELLO! Welcome to http://www.worm.com
      Hacked By Chinese!
  - Payload 2: time bomb
    - Day of month 1-20: **Spread**
    - Day of month 20+: **Attack** (flood 198.137.240.91 = whitehouse.gov)
  - 300,000 machines infected in 14 hours

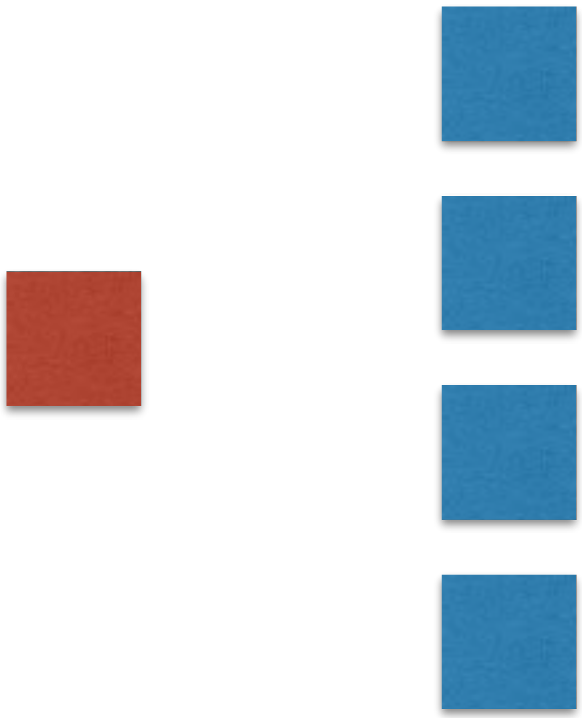
WORMS: A BRIEF HISTORY

Introduction of “modern day” worms

1988 — 1999 — 2000 — **[2001]** — 2002 — 2003

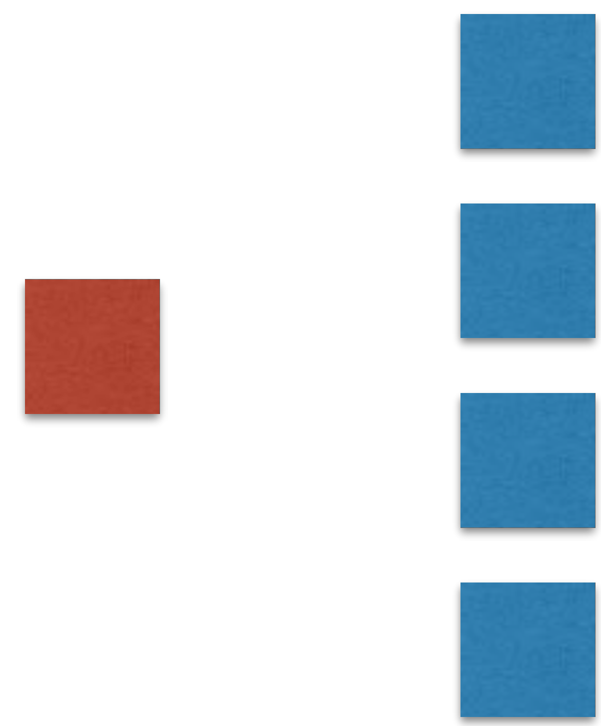
- CodeRed
  - Propagation: Exploited an overflow in the MS-IIS server
  - Payload 1: website defacement
    - HELLO! Welcome to http://www.worm.com
      Hacked By Chinese!
  - Payload 2: time bomb
    - Day of month 1-20: **Spread**
    - Day of month 20+: **Attack** (flood 198.137.240.91 = whitehouse.gov)
  - 300,000 machines infected in 14 hours

WORM PROPAGATION: PROBLEM



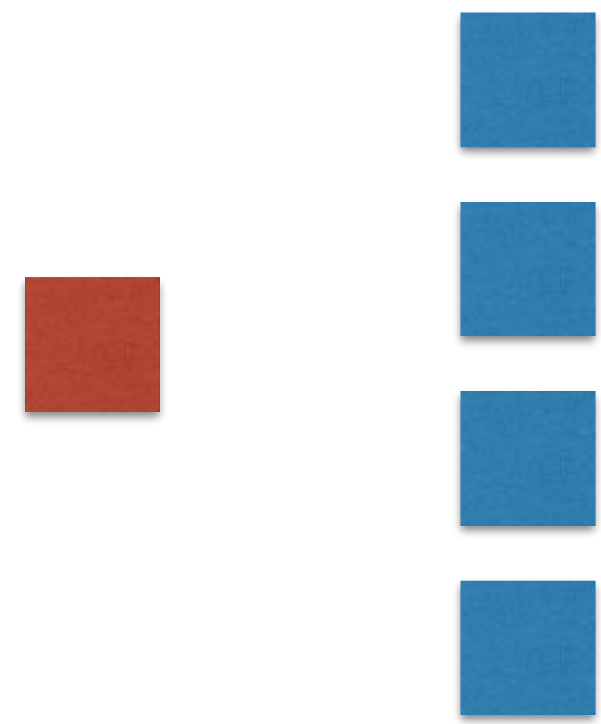
WORM PROPAGATION: PROBLEM

- Goal: spread your virus as widely and as quickly as possible



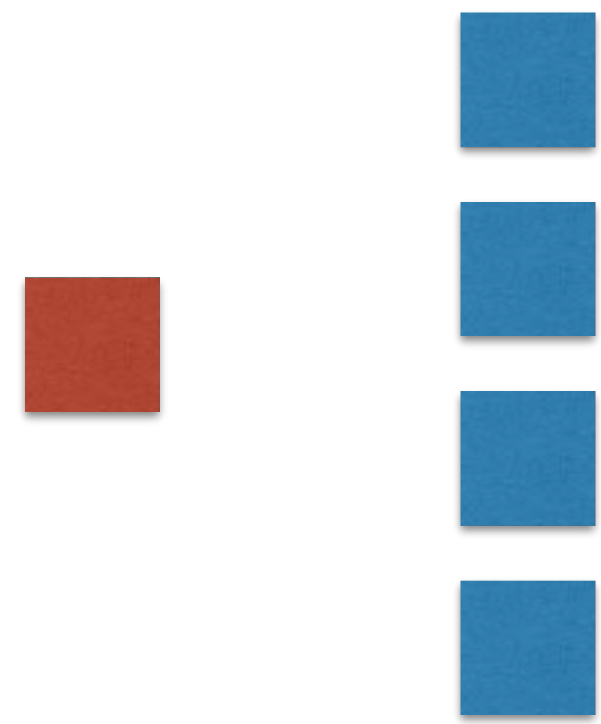
WORM PROPAGATION: PROBLEM

- Goal: spread your virus as widely and as quickly as possible
- Mechanisms:
  - <u>attack(IP address)</u>
    tries to connect to and exploit MS-IIS (if it happens to be running on that IP address)



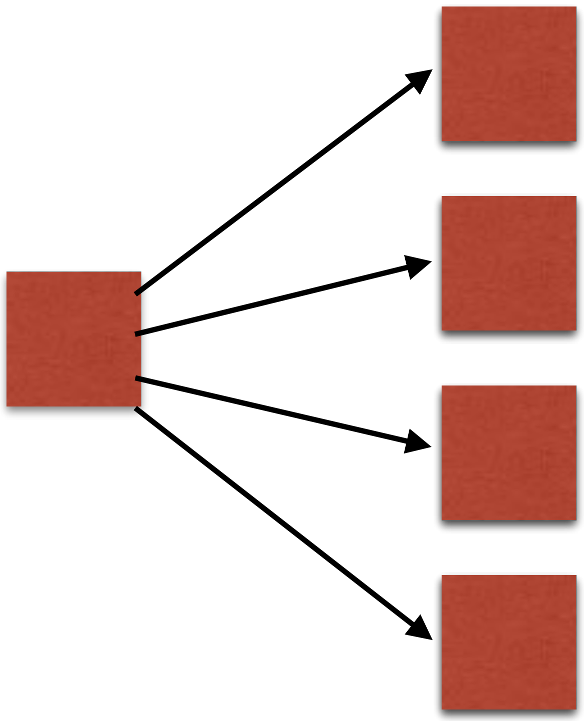
WORM PROPAGATION: PROBLEM

- Goal: spread your virus as widely and as quickly as possible
- Mechanisms:
  - <u>attack(IP address)</u>
    tries to connect to and exploit MS-IIS (if it happens to be running on that IP address)
  - <u>spread(code, state, IP addr)</u>
    copies over the code (plus whatever extra state) to the IP address, and executes it



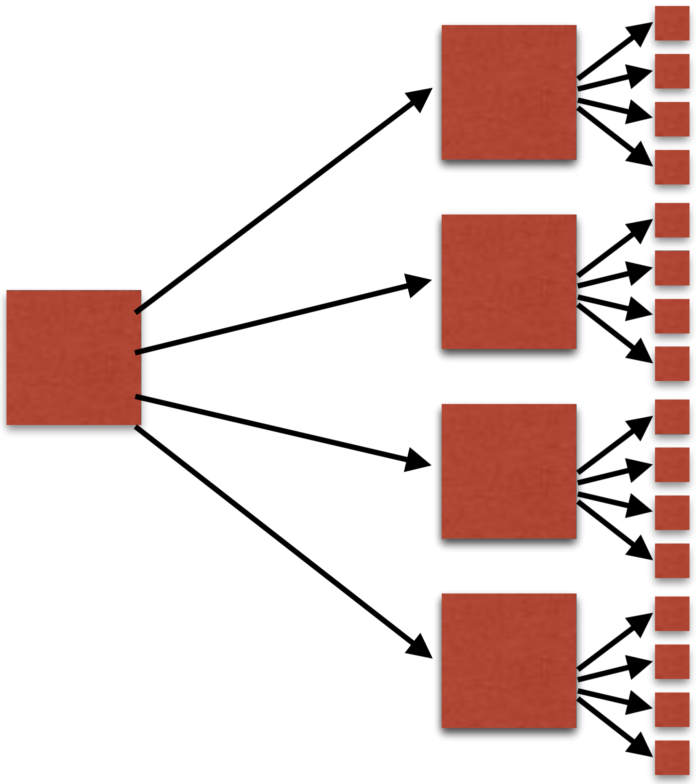
WORM PROPAGATION: PROBLEM

- Goal: spread your virus as widely and as quickly as possible
- Mechanisms:
  - attack(IP address)
    tries to connect to and exploit MS-IIS (if it happens to be running on that IP address)
  - spread(code, state, IP addr)
    copies over the code (plus whatever extra state) to the IP address, and executes it



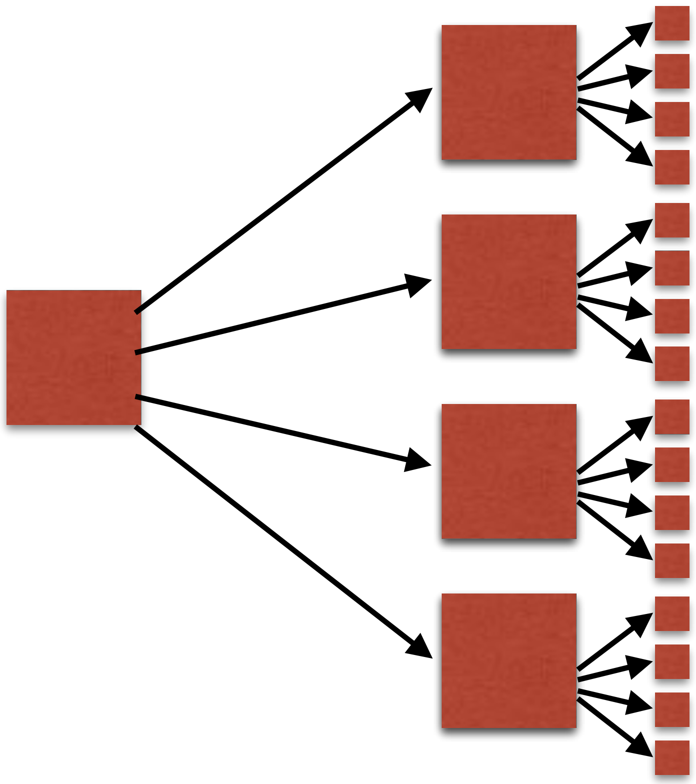
WORM PROPAGATION: PROBLEM

- Goal: spread your virus as widely and as quickly as possible
- Mechanisms:
  - attack(IP address)
    tries to connect to and exploit MS-IIS (if it happens to be running on that IP address)
  - spread(code, state, IP addr)
    copies over the code (plus whatever extra state) to the IP address, and executes it



WORM PROPAGATION: PROBLEM

- Goal: spread your virus as widely and as quickly as possible
- Mechanisms:
  - attack(IP address)
    tries to connect to and exploit MS-IIS (if it happens to be running on that IP address)
  - spread(code, state, IP addr)
    copies over the code (plus whatever extra state) to the IP address, and executes it
- Question: What IP addresses do you choose? What do your "children" choose?



CODERED'S PROPAGATION: V.1

- Spread by randomly scanning the entire 32-bit IP address space
  - Pick a pseudorandom 32-bit number = IP addr
  - Try attack(IP addr)
  - Repeat
- This is a very common *but not fundamental* worm technique
- Each instance of the worm used the *same random number seed*

CODERED'S PROPAGATION: V.1

- Spread by randomly scanning the entire 32-bit IP address space
  - Pick a pseudorandom 32-bit number = IP addr
  - Try attack(IP addr)
  - Repeat
- This is a very common *but not fundamental* worm technique
- Each instance of the worm used the *same random number seed*

Linear

CODERED'S PROPAGATION: V.1

- Spread by randomly scanning the entire 32-bit IP address space
  - Pick a pseudorandom 32-bit number = IP addr
  - Try attack(IP addr)
  - Repeat

- This is a very common *but not fundamental* worm technique

- Each instance of the worm used the *same random number seed*

What would the growth over time be? *Linear*

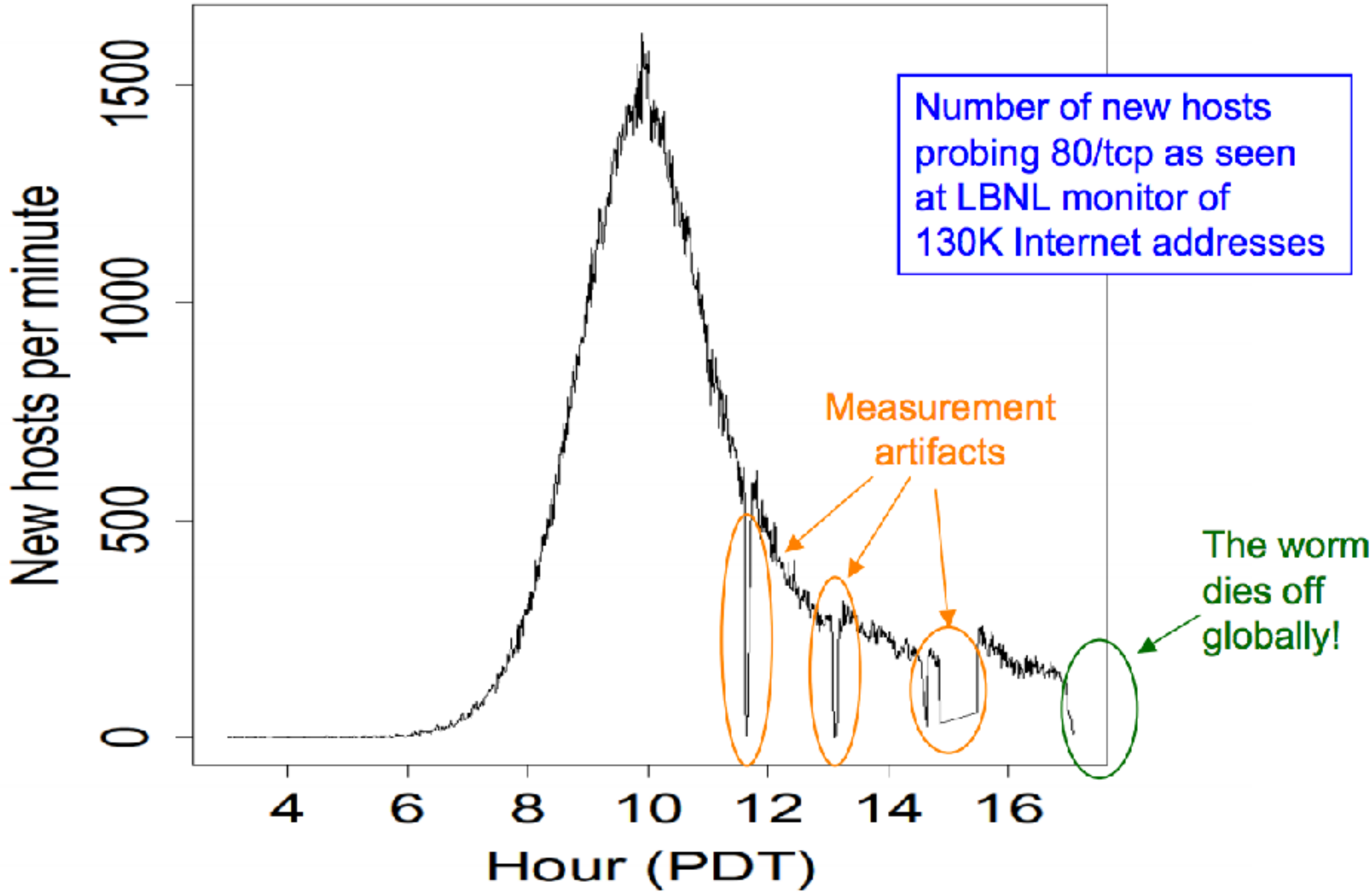
CODERED'S PROPAGATION: V.2

- Revision released one week later (July 19, 2001)

- Whitehouse.gov **changed** its IP address
  - This caused CodeRed to **die** for date ≥20th of the month
  - .. Author didn't test the code: it was ***buggy***!

- But with this revision, the random number generator was **seeded** properly!

CODERED'S GROWTH



MODELING WORM SPREAD

- Worm spread is well described as *infectious epidemic*
  - Classic "SI" model (Susceptible-Infectible)

- Model parameters:
  - N: Population size
  - S(t): # Susceptible hosts at time t
  - I(t): # Infected hosts at time t
  - β: contact rate
    - How many population members each infected host communicates with per unit time
    - E.g., if each infected host scans 10 IP addresses per unit time, and 2% of all IP addresses run a vulnerable server, then β = 0.2

$$N = S(t) + I(t)$$
$$S(0) = I(0) = N/2$$

$$i(t) = I(t) / N = \text{fraction of hosts infected}$$

COMPUTING HOW AN EPIDEMIC PROGRESSES

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

COMPUTING HOW AN EPIDEMIC PROGRESSES

Increase in
infectibles
per unit time

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

COMPUTING HOW AN EPIDEMIC PROGRESSES

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

Increase in # infectibles per unit time → $\frac{dI}{dt}$

Total attempted contacts per unit time → $\beta \cdot I$

COMPUTING HOW AN EPIDEMIC PROGRESSES

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

Increase in # infectibles per unit time → $\frac{dI}{dt}$

Total attempted contacts per unit time → $\beta \cdot I$

Fraction of contacts expected to succeed → $\frac{S}{N}$

COMPUTING HOW AN EPIDEMIC PROGRESSES

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

Increase in # infectibles per unit time ($\frac{dI}{dt}$)

Total attempted contacts per unit time ($\beta \cdot I$)

Fraction of contacts expected to succeed ($\frac{S}{N}$)

Rewriting using $i(t) = \mathbf{I}(t) / N$ and $S = N - \mathbf{I}$:

COMPUTING HOW AN EPIDEMIC PROGRESSES

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

- $\frac{dI}{dt}$: Increase in # infectibles per unit time
- $\beta \cdot I$: Total attempted contacts per unit time
- $\frac{S}{N}$: Fraction of contacts expected to succeed

Rewriting using i(t) = $\mathbf{I}$(t) / N and S = N - $\mathbf{I}$:

$$\frac{di}{dt} = \beta \cdot i \cdot (1-i)$$

COMPUTING HOW AN EPIDEMIC PROGRESSES

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

Increase in # infectibles per unit time

Total attempted contacts per unit time

Fraction of contacts expected to succeed

Rewriting using $i(t) = \mathbf{I}(t) / N$ and $S = N - \mathbf{I}$:

$$\frac{di}{dt} = \beta \cdot i \cdot (1-i) \quad \Rightarrow \quad i(t) = \frac{e^{\beta t}}{1+e^{\beta t}}$$

COMPUTING HOW AN EPIDEMIC PROGRESSES

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

Increase in # infectibles per unit time → $\frac{dI}{dt}$

Total attempted contacts per unit time → $\beta \cdot I$

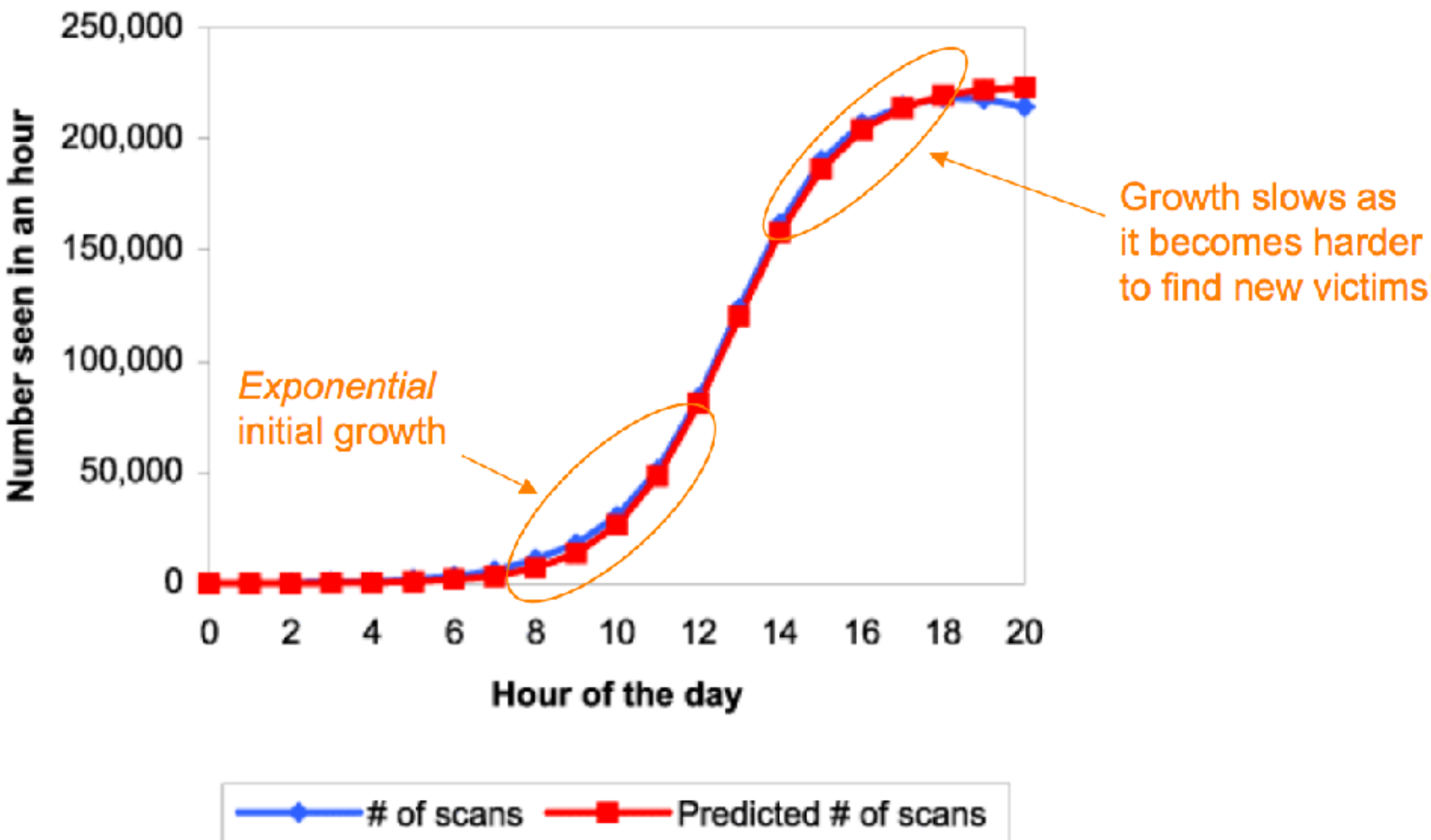
Fraction of contacts expected to succeed → $\frac{S}{N}$

Rewriting using i(t) = **I**(t) / N and S = N - **I**:

$$\frac{di}{dt} = \beta \cdot i \cdot (1-i) \quad \Rightarrow \quad i(t) = \frac{e^{\beta t}}{1+e^{\beta t}}$$

Fraction infected grows as a *logistic*

FITTING THE MODEL TO CODERED



Credit: Vern Paxson's CS 161 at Berkeley

CODERED SPREAD

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

CODERED SPREAD

- Note that # of new infections scales with contact rate β

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

CODERED SPREAD

- Note that # of new infections scales with contact rate β
- For a scanning worm, β increases with N
  - Larger populations are infected more quickly!
  - More likely that a given scan finds a population member

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

CODERED SPREAD

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

- Note that # of new infections scales with contact rate β
- For a scanning worm, β increases with N
  - Larger populations are infected more quickly!
  - More likely that a given scan finds a population member
- Large-scale monitoring found 360K systems were infected with Code Red on July 19
  - Within 13 hours

CODERED SPREAD

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

- Note that # of new infections scales with contact rate β
- For a scanning worm, β increases with N
  - Larger populations are infected more quickly!
  - More likely that a given scan finds a population member
- Large-scale monitoring found 360K systems were infected with Code Red on July 19
  - Within 13 hours
- That night (the 20th) the worm died due to bug

CODERED SPREAD

$$\frac{dI}{dt} = \beta \cdot I \cdot \frac{S}{N}$$

- Note that # of new infections scales with contact rate β
- For a scanning worm, β increases with N
  - Larger populations are infected more quickly!
  - More likely that a given scan finds a population member
- Large-scale monitoring found 360K systems were infected with Code Red on July 19
  - Within 13 hours
- That night (the 20th) the worm died due to bug
- Successfully managed to restart itself Aug 1
  - … and each successive month for years to come

BETTER WORMS

Key challenge:

Coordinating action across distributed hosts

This is a distributed systems problem!

BETTER WORMS

BETTER WORMS

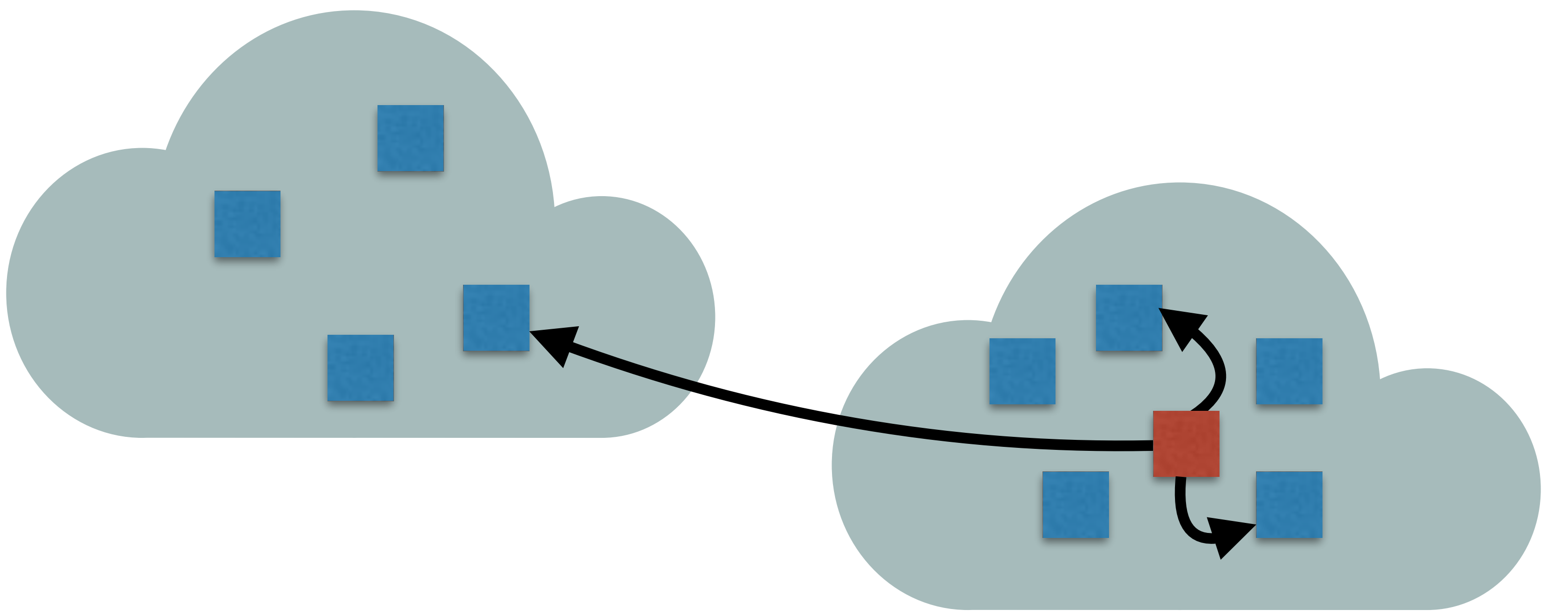
Localized scanning:

Preferentially hosts with similar IP addresses

BETTER WORMS

Localized scanning:

Preferentially hosts with similar IP addresses



Pr = 3/8	Random IP address in same /16
Pr = 4/8	Random IP address in same /8
Pr = 1/8	Random IP address

BETTER WORMS

BETTER WORMS

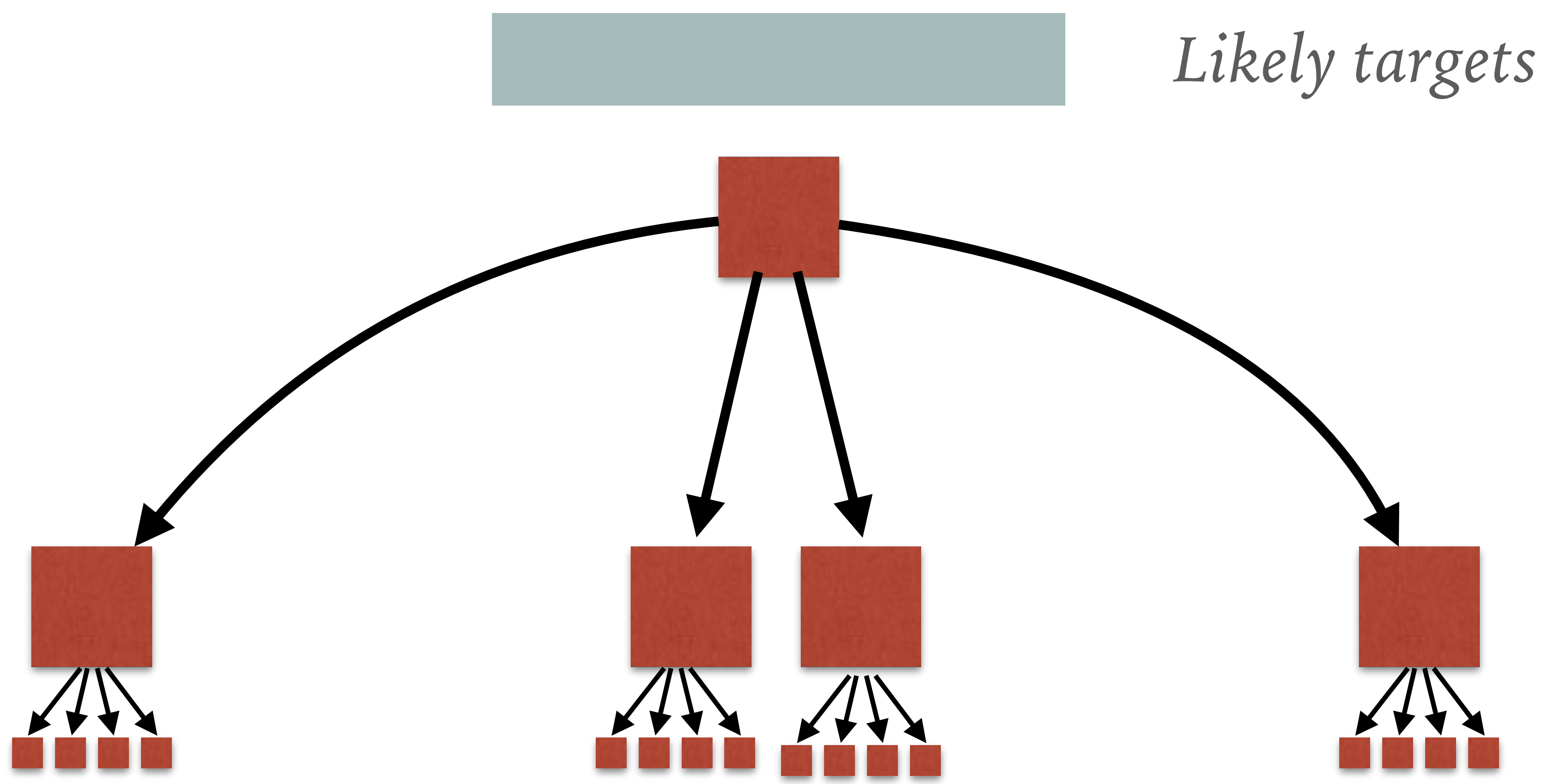
Hit-list scanning

Start with a list of likely-successful targets

BETTER WORMS

Hit-list scanning

Start with a list of likely-successful targets



Quickly build an initial set of bots to “get off the ground”

BETTER WORMS

BETTER WORMS

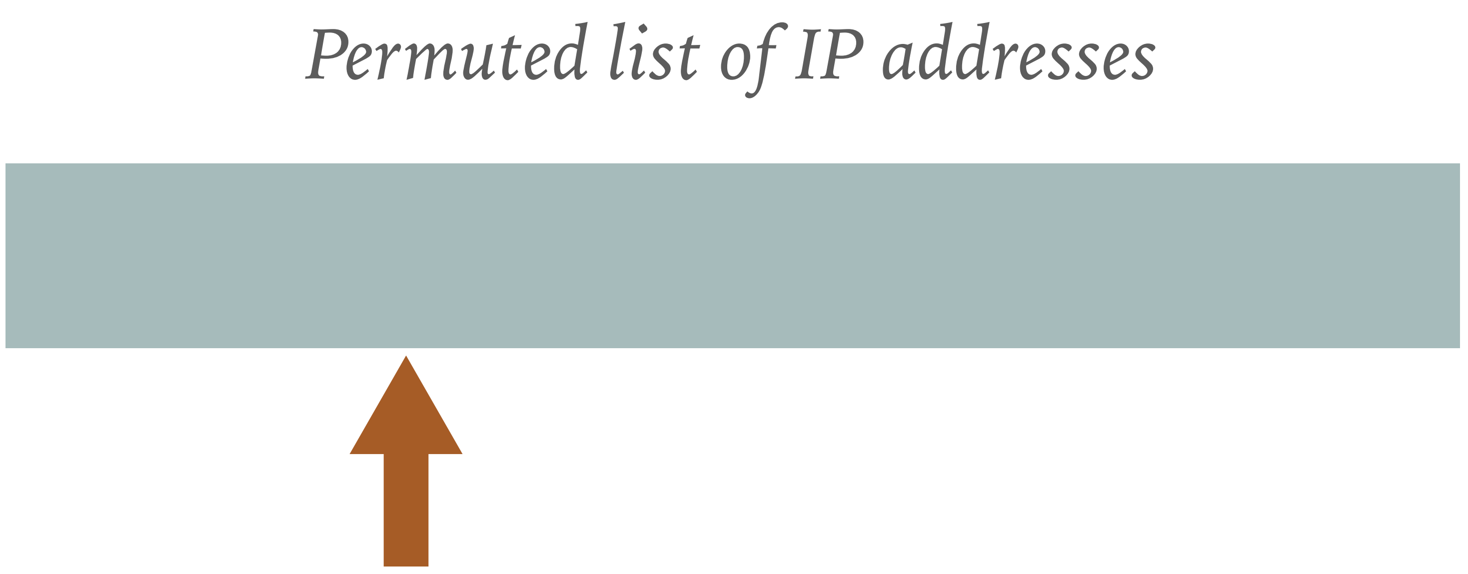
Permutation scanning

Break up the work deterministically

BETTER WORMS

Permutation scanning

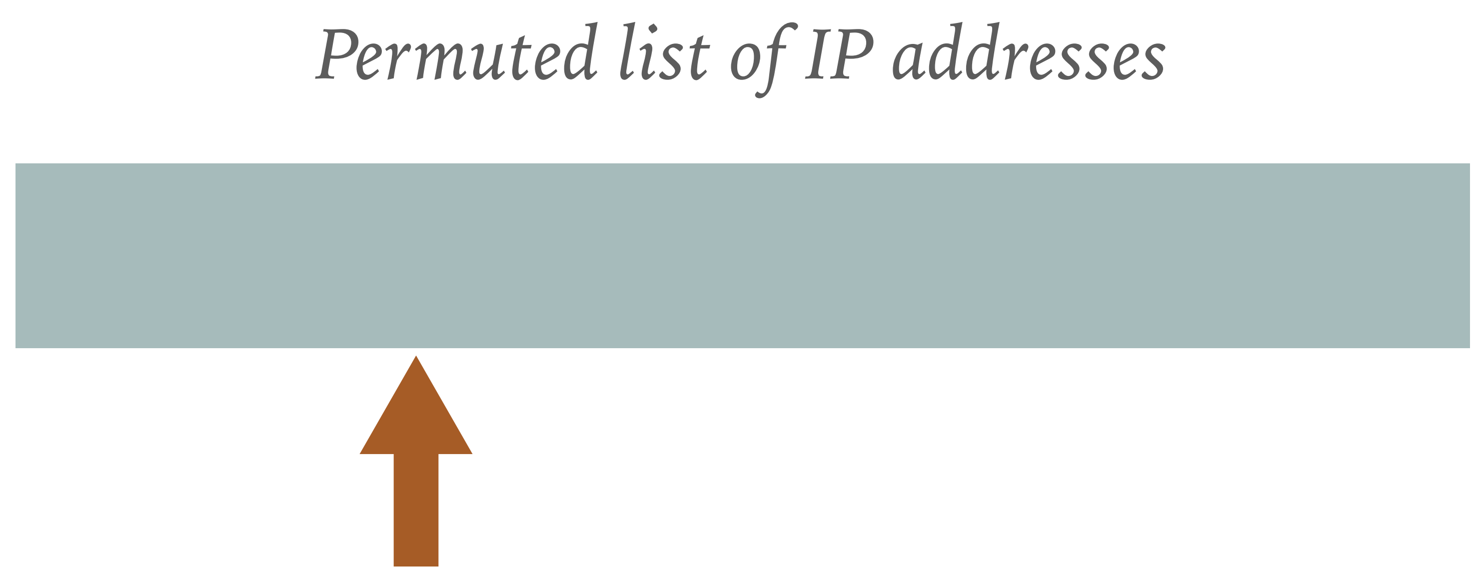
Break up the work deterministically



BETTER WORMS

Permutation scanning

Break up the work deterministically



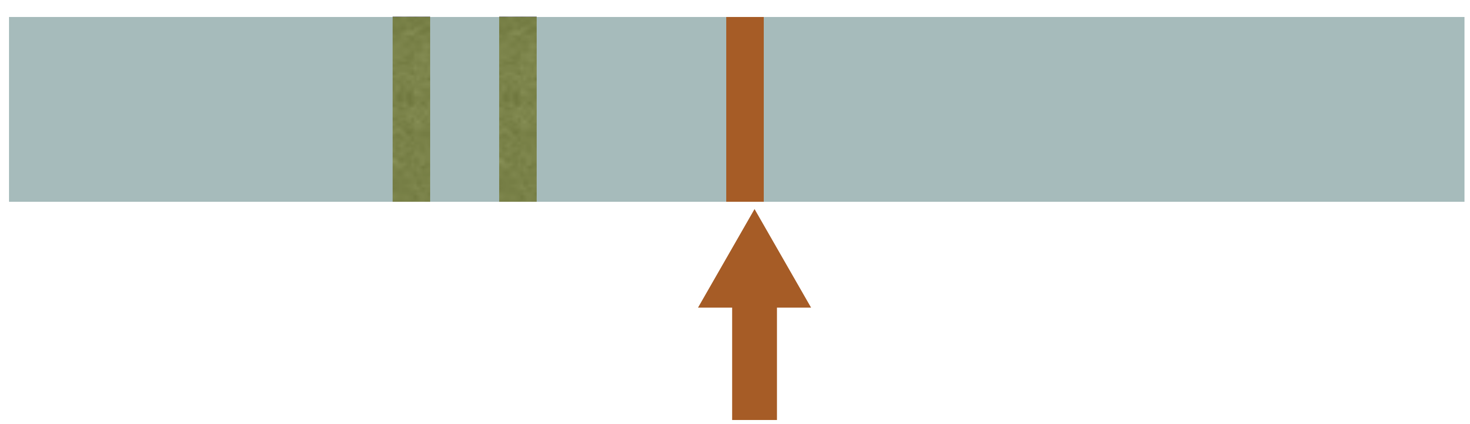
Start at a random seed; iteratively attack

BETTER WORMS

Permutation scanning

Break up the work deterministically

Permuted list of IP addresses



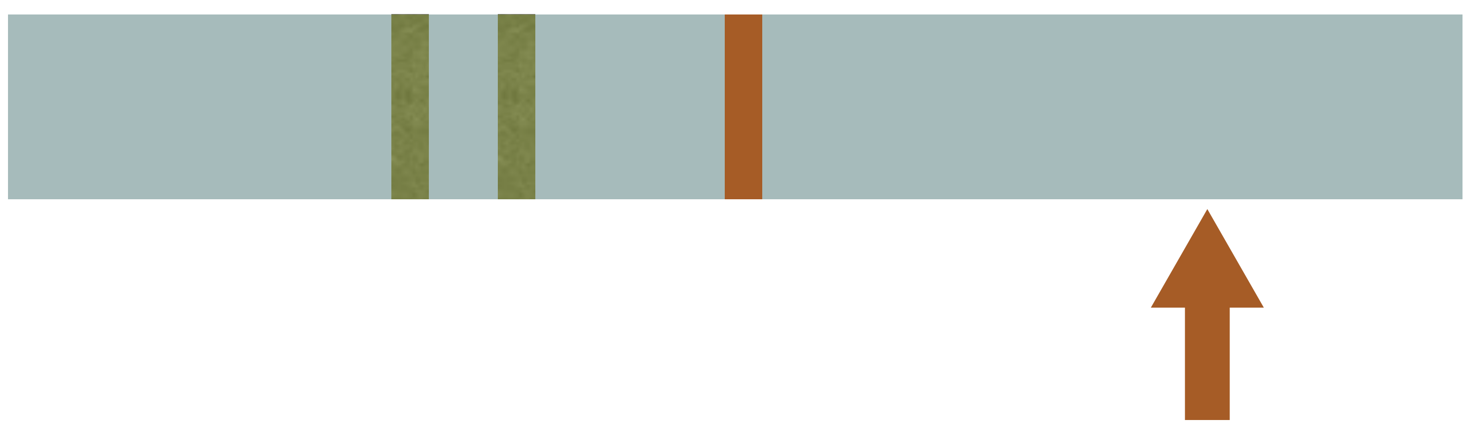
Start at a random seed; iteratively attack

BETTER WORMS

Permutation scanning

Break up the work deterministically

Permuted list of IP addresses



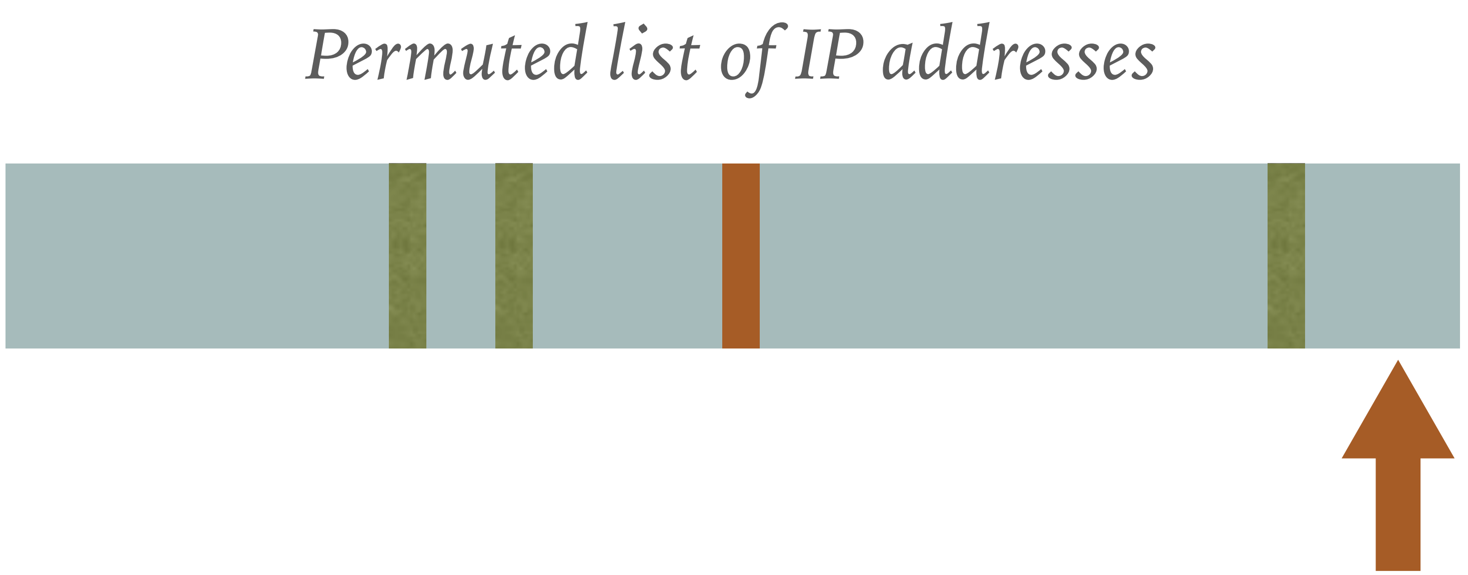
Start at a random seed; iteratively attack

When you reach an already-attacked node, re-seed

BETTER WORMS

Permutation scanning

Break up the work deterministically

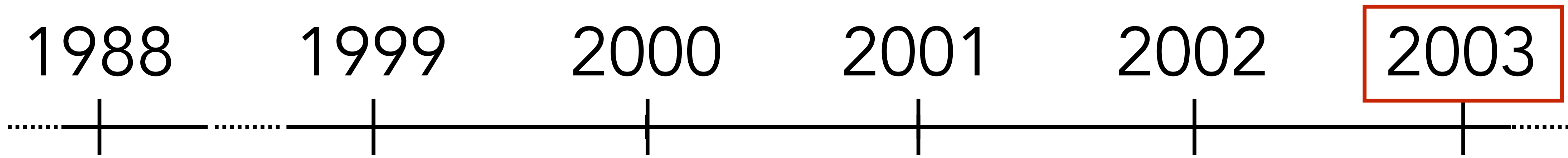


Start at a random seed; iteratively attack

When you reach an already-attacked node, re-seed

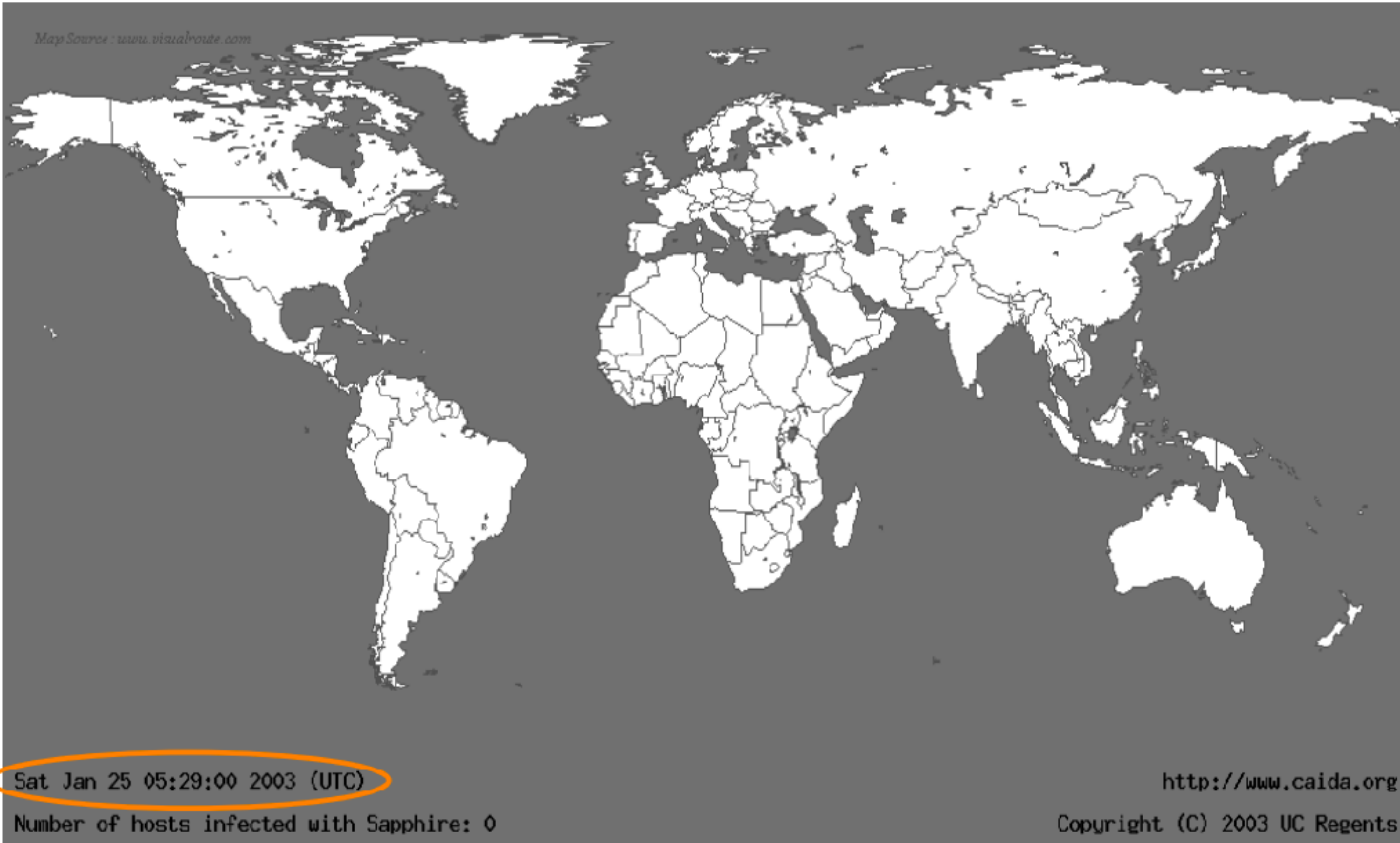
WORMS: A BRIEF HISTORY

The harm can be substantial



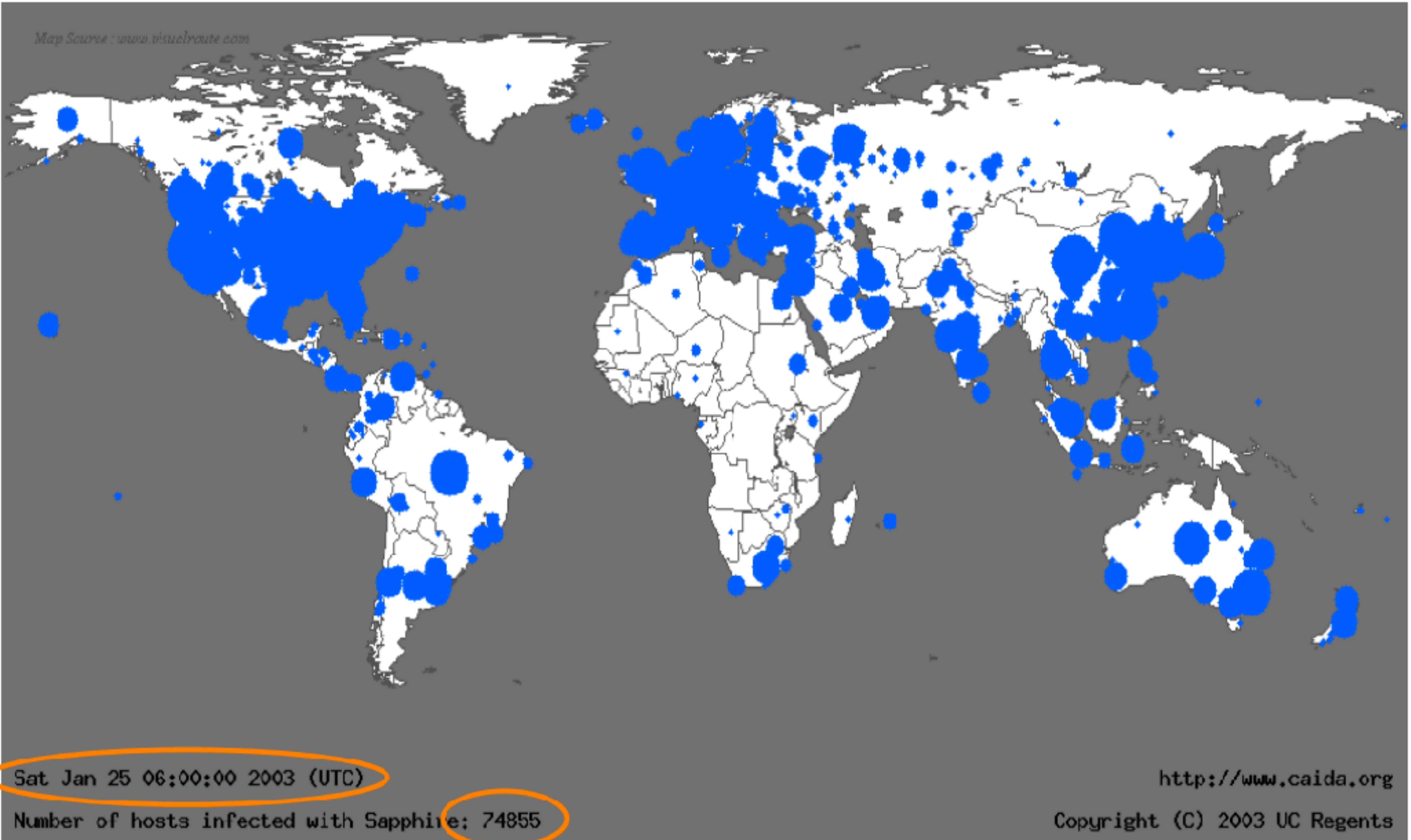
- SQL Slammer
  - Exploited an overflow in the MS-SQL server
  - 75,000 machines infected in 10 *minutes*

LIFE BEFORE SLAMMER



Credit: Vern Paxson's CS 161 at Berkeley

LIFE JUST AFTER SLAMMER

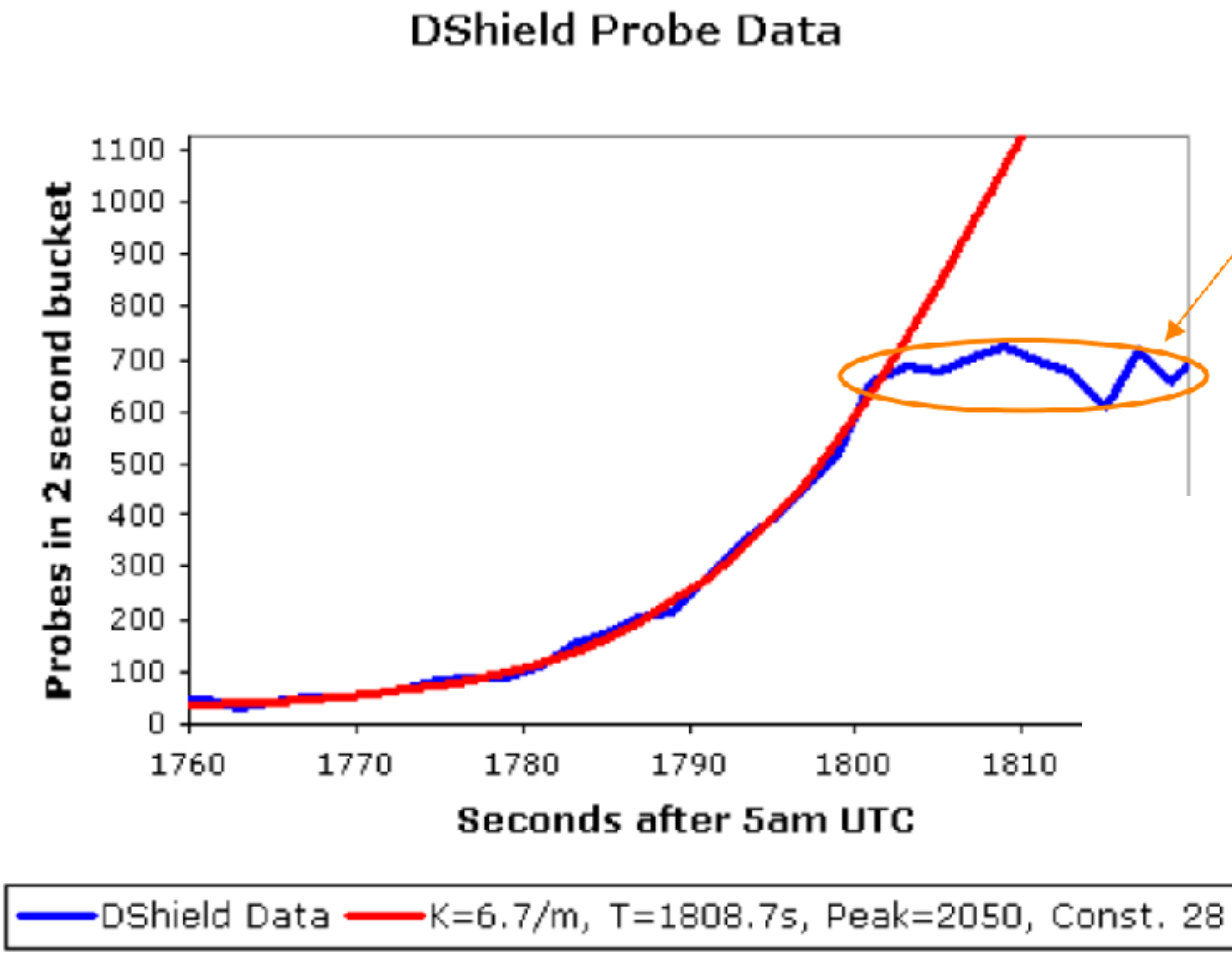


Credit: Vern Paxson's CS 161 at Berkeley

SLAMMER PROPAGATION

- Slammer exploited connectionless UDP service rather than connection-oriented TCP
- Entire worm fit in a single packet!
- When scanning, the worm could “fire and forget”
  - Stateless!
- Infected 75k+ hosts in < 10 minutes
- At its peak, doubled ever <u>8.5 seconds</u>

SLAMMER'S GROWTH

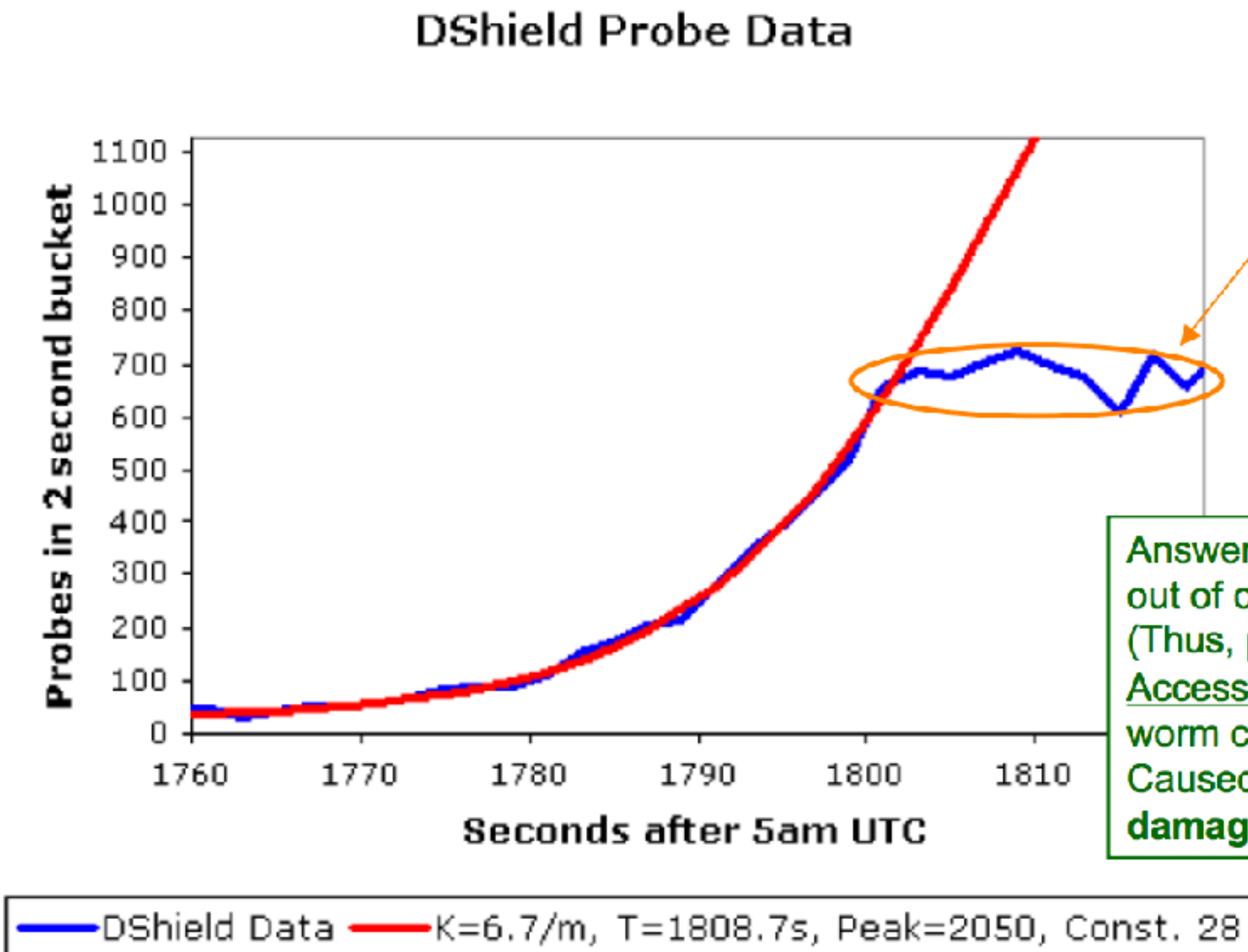


What could have caused growth to deviate from the model?

Hint: at this point the worm is generating *55,000,000 scans/sec*

Credit: Vern Paxson's CS 161 at Berkeley

SLAMMER'S GROWTH



What could have caused growth to deviate from the model?

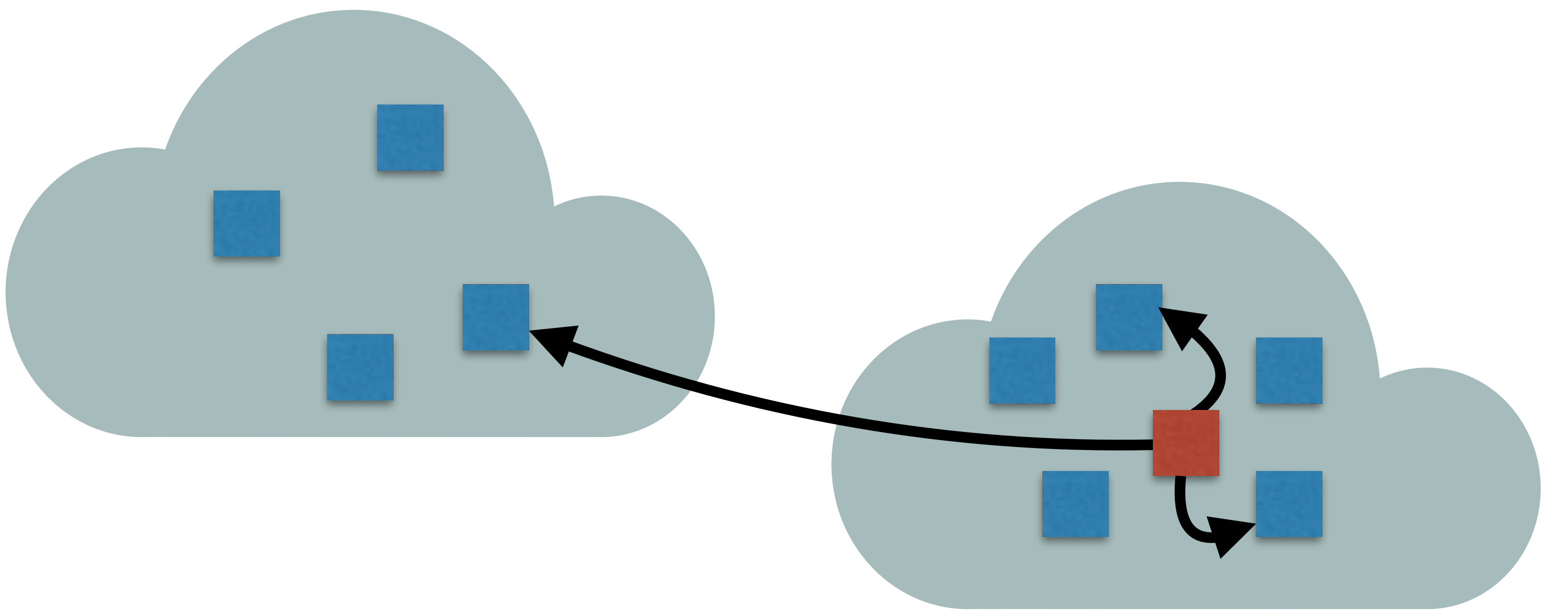
Hint: at this point the worm is generating *55,000,000 scans/sec*

Answer: the Internet ran out of carrying capacity! (Thus, β decreased.) Access links used by worm completely clogged. Caused **major collateral damage**.

Credit: Vern Paxson's CS 161 at Berkeley

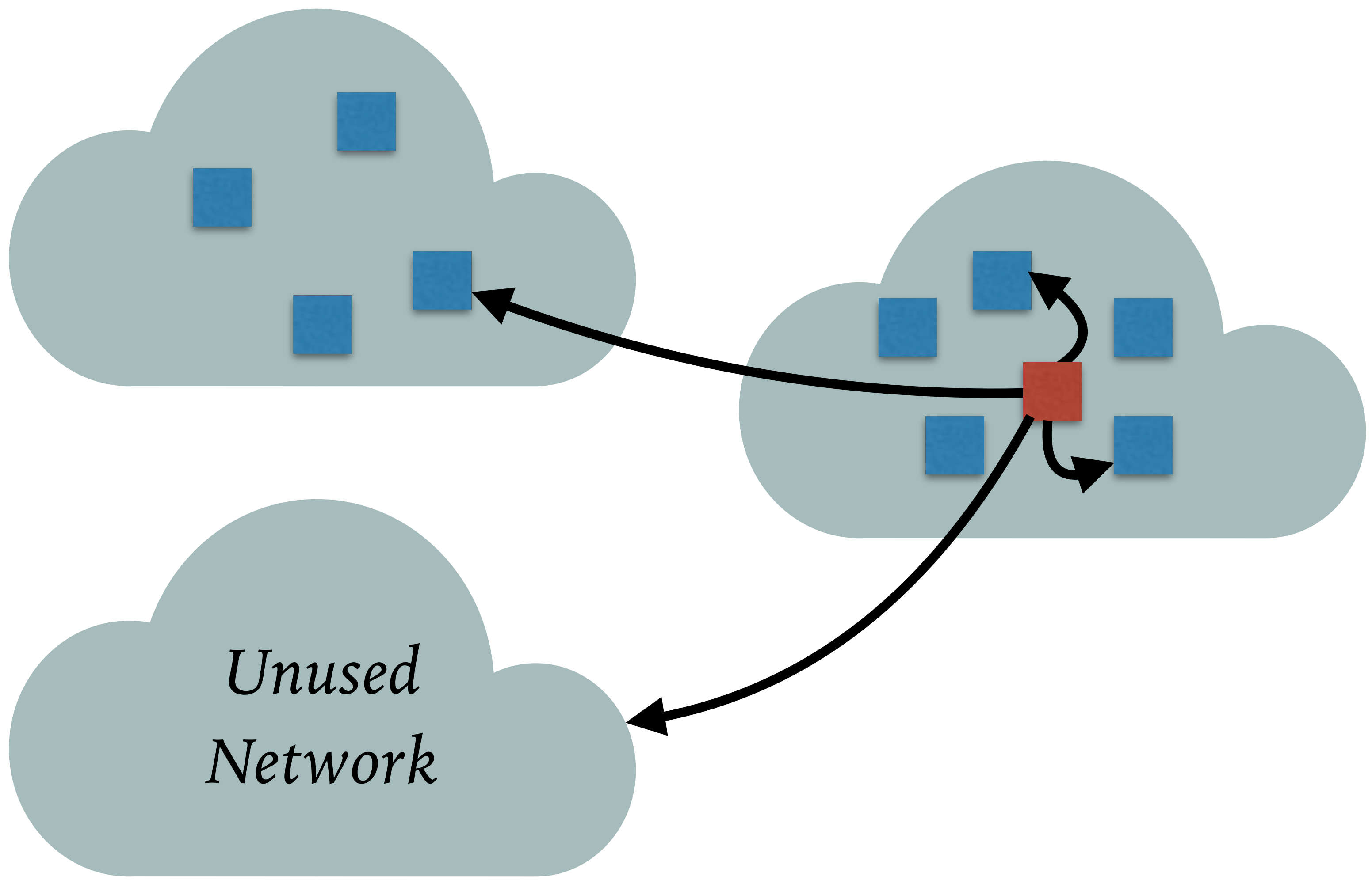
HOW CAN YOU MEASURE WORM ACTIVITY AT SCALE?

Idea: Exploit the fact that worms indiscriminately scan



HOW CAN YOU MEASURE WORM ACTIVITY AT SCALE?

Idea: Exploit the fact that worms indiscriminately scan



No-one *should* be hitting this network

INTERNET BACKGROUND RADIATION

Traffic sent to unused IP addresses

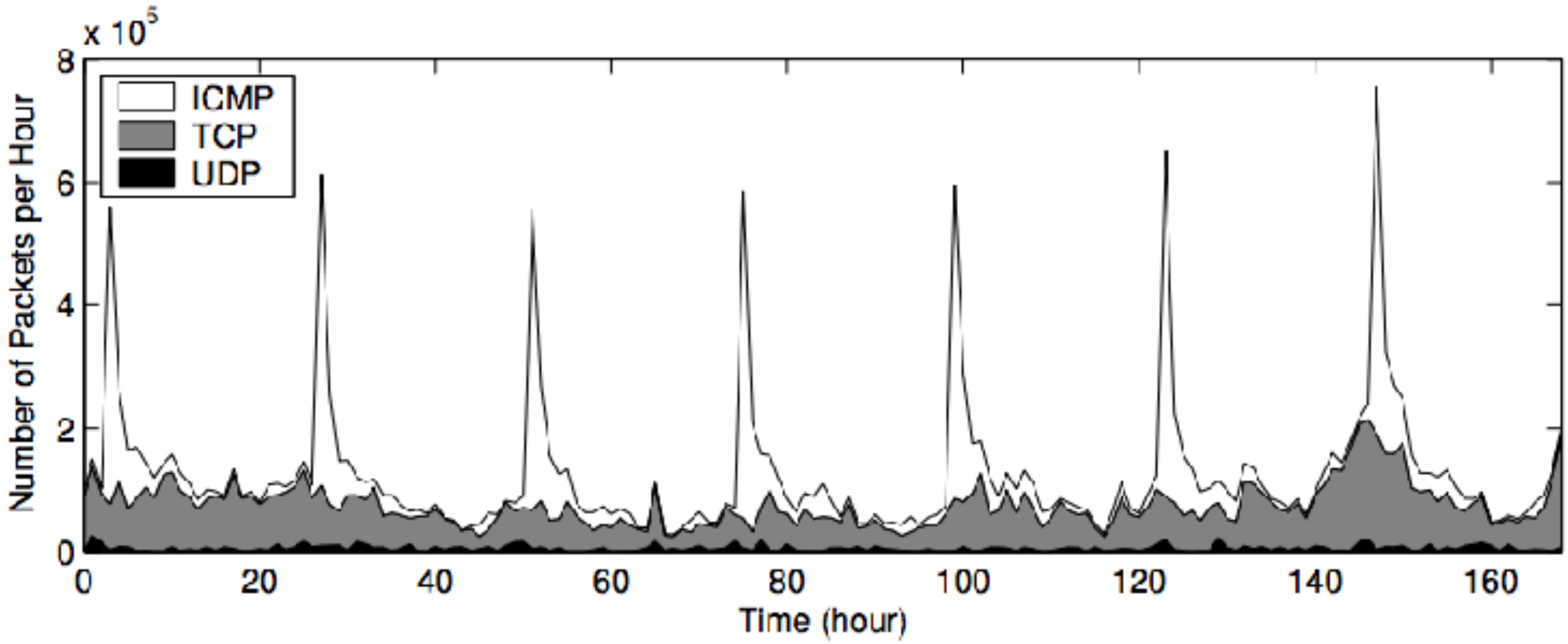


Figure 6: Number of background radiation packets per hour seen at LBL

WHAT CAN WE LEARN FROM BACKGROUND RADIATION?

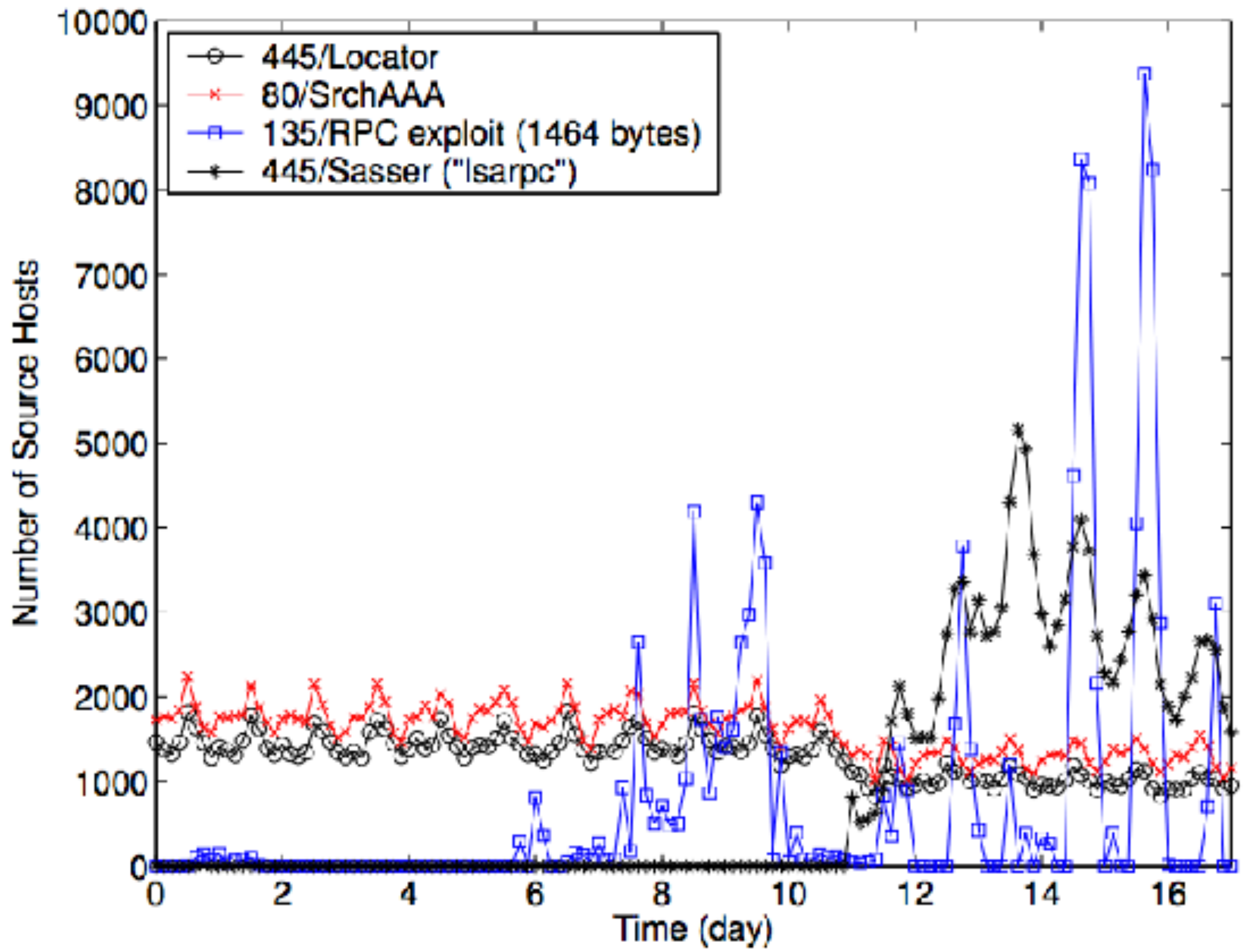


Figure 11: The Big Exploits (Apr 20 to May 7, 2004), as observed on 5 /C networks at LBL. The source hosts are counted every three hours.