Malware

INGI2347: COMPUTER SYSTEM SECURITY (Spring 2016)

Marco Canini

UCL
Université catholique de Louvain

Lecture slides adapted from UC Berkeley CS161 by Vern Paxson
Reproduced with permission

The Problem of Malware

- Malware = malicious code that runs on a victim's system
- How does it manage to run?
  - Attacks a network-accessible vulnerable service
  - Vulnerable client connects to remote system that sends over an attack (a *driveby*)
  - *Social engineering:* trick user into running/installing
  - "Autorun" functionality (esp. from plugging in USB device)
  - Slipped into a system component (at manufacture; compromise of software provider; substituted via MITM)
  - Attacker with local access downloads/runs it directly
    - Might include using a "local root" exploit for privileged access

What Can Malware Do?

- Pretty much *anything*
  - Payload generally decoupled from how manages to run
  - Only subject to permissions under which it runs
- Examples:
  - Brag or exhort or extort (pop up a message/display)
  - Trash files (just to be nasty)
  - Damage hardware (!)
  - Launch external activity (spam, *click fraud*, DoS)
  - Steal information (*exfiltrate*)
  - Keylogging; screen / audio / camera capture
  - Encrypt files (*ransomware*)
- Possibly delayed until condition occurs
  - "time bomb" / "logic bomb"

Types of Malware

- Viruses – propagates with help of other programs
- Worms – self-contained programs
- Trojan horses – pretends to do one thing; does another
- Backdoors – secret entry point into a system
- Rootkit – hides the presence of other malware
- Spyware – sends personal information to third party
- Adware – shows Ads

Malware That Automatically Propagates

- Virus = code that propagates (**replicates**) across systems by arranging to have itself *eventually executed,* creating a new additional instance
  - Generally infects by altering stored code
- Worm = code that self-propagates/replicates across systems by arranging to have itself *immediately executed* (creating new addl. instance)
  - Generally infects by altering running code
  - No user intervention required
- (Note: line between these isn't always so crisp; plus some malware incorporates both styles)

Virus/Worm Writer's Goals

- Hard to detect
- Hard to destroy or deactivate
- Spreads infection widely/quickly
- Can reinfect a host
- Easy to create
- Machine/OS independent

The Problem of Viruses

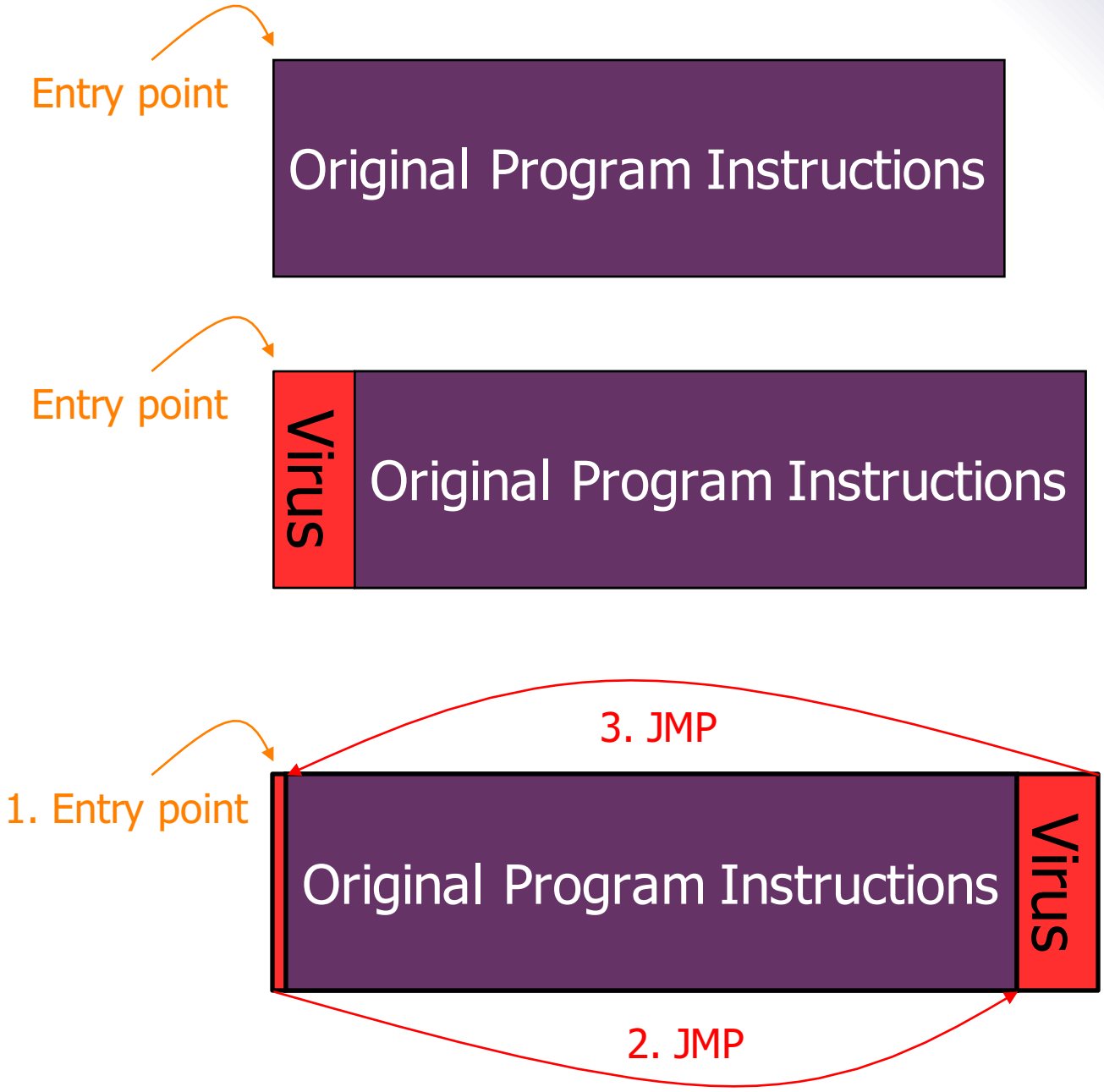
- Opportunistic = code will eventually execute
  - Generally due to user action
    - Running an app, booting their system, opening an attachment
- Separate notions: how it *propagates* vs. what else it does when executed (*payload*)
- General infection strategy:
  find some code lying around, alter it to include the virus
- Have been around for decades …
  - … resulting **arms race** has heavily influenced evolution of modern malware

Propagation

- When virus runs, it looks for an opportunity to infect additional systems
- One approach: look for USB-attached thumb drive, alter any executables it holds to include the virus
  - Strategy: when drive later attached to another system & altered executable runs, it locates and infects executables on new system's hard drive

autorun is handy here!

- Or: when user sends email w/ attachment, virus alters attachment to add a copy of itself
  - Works for attachment types that include programmability
  - E.g., Word documents (macros), PDFs (Javascript)
  - Virus can also send out such email proactively, using user's address book + enticing subject ("I Love You")



Original program instructions can be:

- Application the user runs
- Run-time library / routines resident in memory
- Disk blocks used to boot OS
- Autorun file on USB device
- …

Other variants are possible; whatever manages to get the virus code executed

Detecting Viruses

- Signature-based detection
  - Look for bytes corresponding to injected virus code
  - High utility due to replicating nature
    - If you capture a virus V on one system, by its nature the virus will be trying to infect *many other systems*
    - Can protect those other systems by installing recognizer for V
- Drove development of multi-billion $$ AV industry (AV = "antivirus")
  - So many endemic viruses that detecting well-known ones becomes a "*checklist item*" for security audits
- Using signature-based detection also has de facto utility for (glib) marketing
  - Companies compete on number of signatures …
    - … rather than their quality (harder for customer to assess)

virustotal

VirusTotal is a free service that **analyzes suspicious files and URLs** and facilitates the quick detection of viruses, worms, trojans, and all kinds of malware.

SHA256: 71d1723d1269abef2b78d6c46390452058c047bc44949bad8f493446f947c8bc

File name: qvodsetupls27.exe

Detection ratio: 41 / 46

Analysis date: 2013-04-11 11:56:27 UTC (3 days, 10 hours ago)

More details

Analysis | Additional information | Comments | Votes

Antivirus	Result	Update
Agnitum	Trojan.DR.AgentIAmUdZaEHJGw	20130410
AhnLab-V3	Dropper/Win32.Agent	20130410
AntiVir	DR/MicroJoiner.Gen	20130411
Antiy-AVL	-	20130411
Avast	Win32:Microjoin-CD [Trj]	20130411
AVG	Dropper.Tiny.I	20130411
BitDefender	Trojan.Crypt.CG	20130411

Virus Writer / AV Arms Race

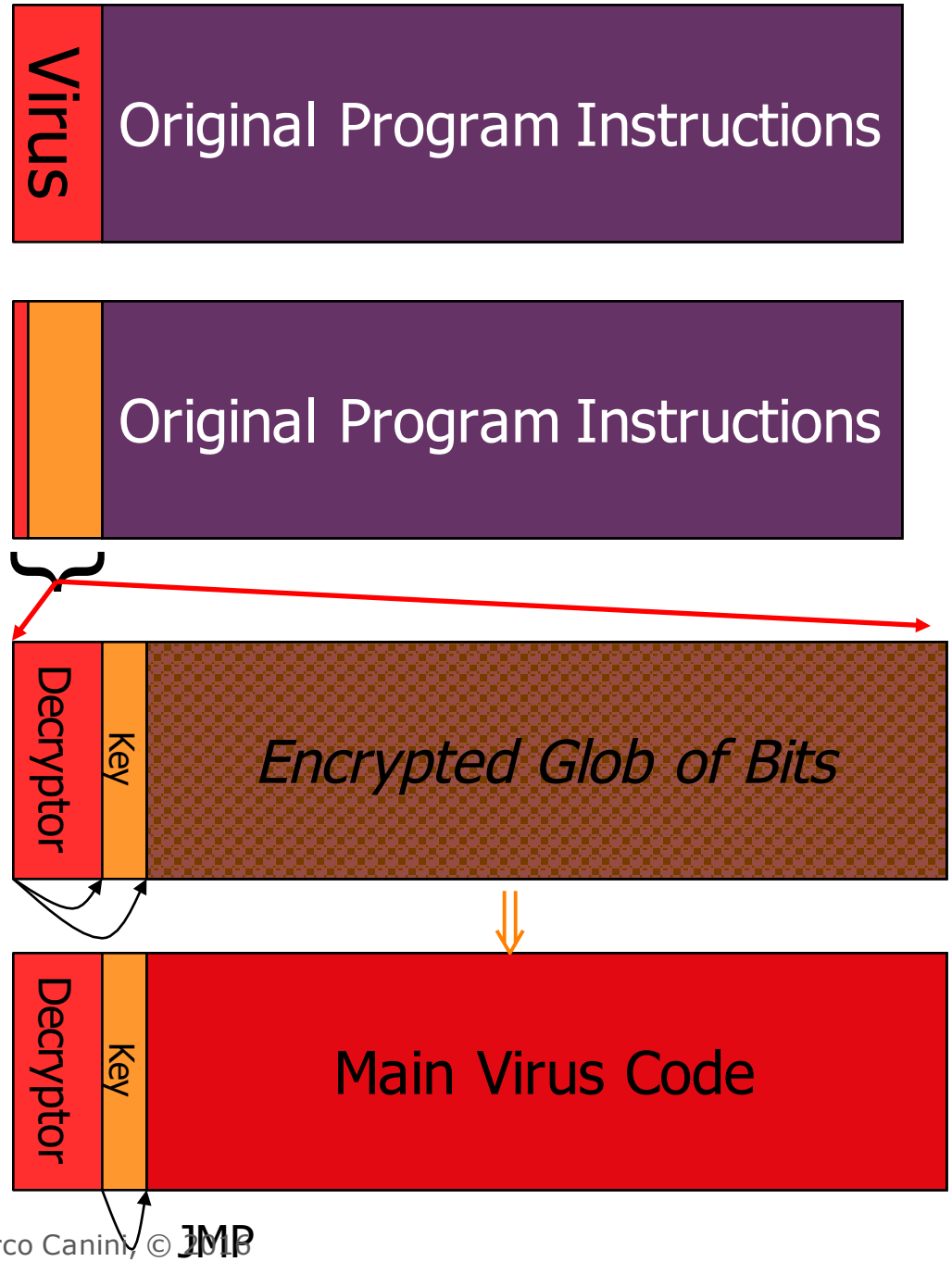
- If you are a virus writer and your beautiful new creations don't get very far because each time you write one, the AV companies quickly push out a signature for it ….
  - …. *What are you going to do?*
- Need to keep changing your viruses …
  - … or at least changing their appearance!
- How can you mechanize the creation of new instances of your viruses …
  - … so that whenever your virus propagates, what it injects as a copy of itself looks different?

Polymorphic Code

- Idea: every time your virus propagates, it inserts a newly encrypted copy of itself
  - Clearly, encryption needs to vary
    - Either by using a different key each time
    - Or by including some random initial padding
  - Note: weak (but simple/fast) crypto algorithm works fine
    - No need for truly strong encryption, just obfuscation

encryption:
transforms a plaintext into a ciphertext that is unintelligible for non-authorized parties

- When injected code runs, it decrypts itself to obtain the original functionality



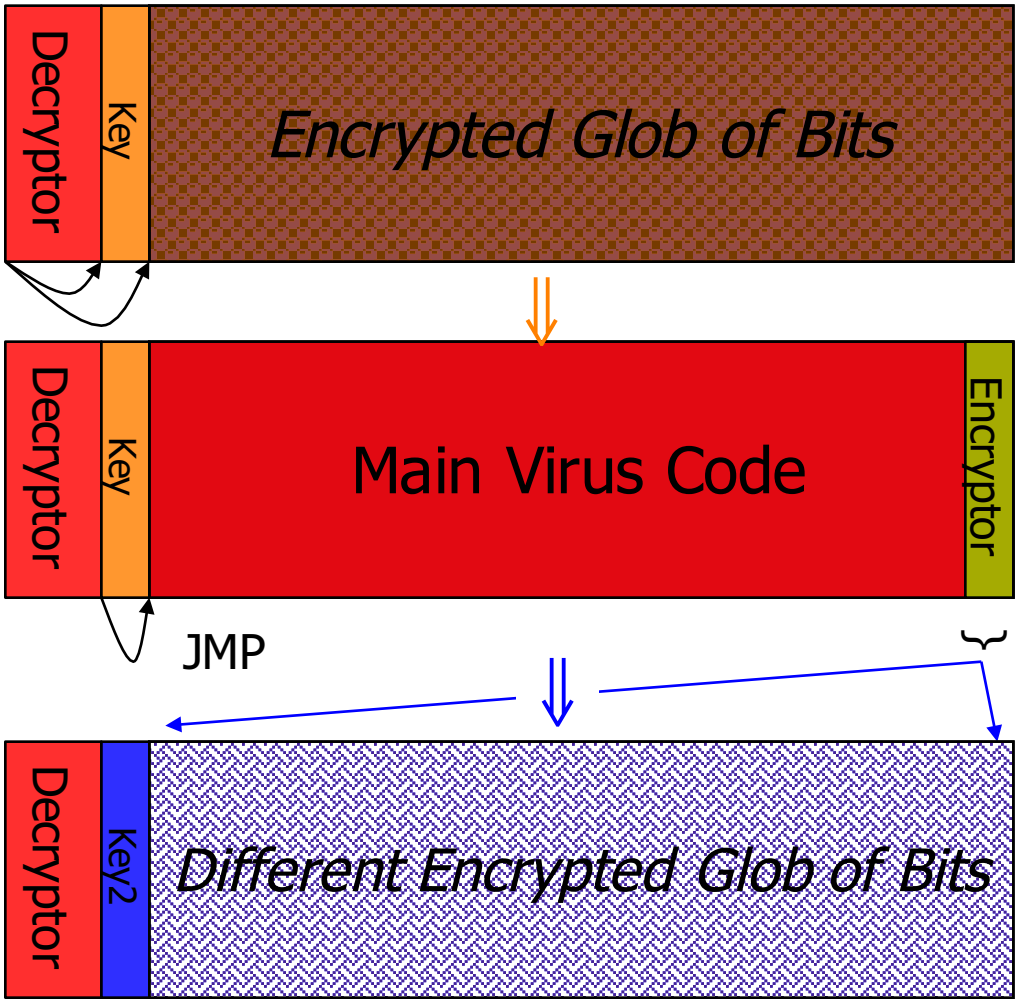
Instead of this …

Virus has *this* initial structure

When executed, decryptor applies key to decrypt the glob …

… and jumps to the decrypted code once stored in memory

Polymorphic Propagation



Once running, virus uses an *encryptor* with a new key to propagate

New virus instance bears little resemblance to original

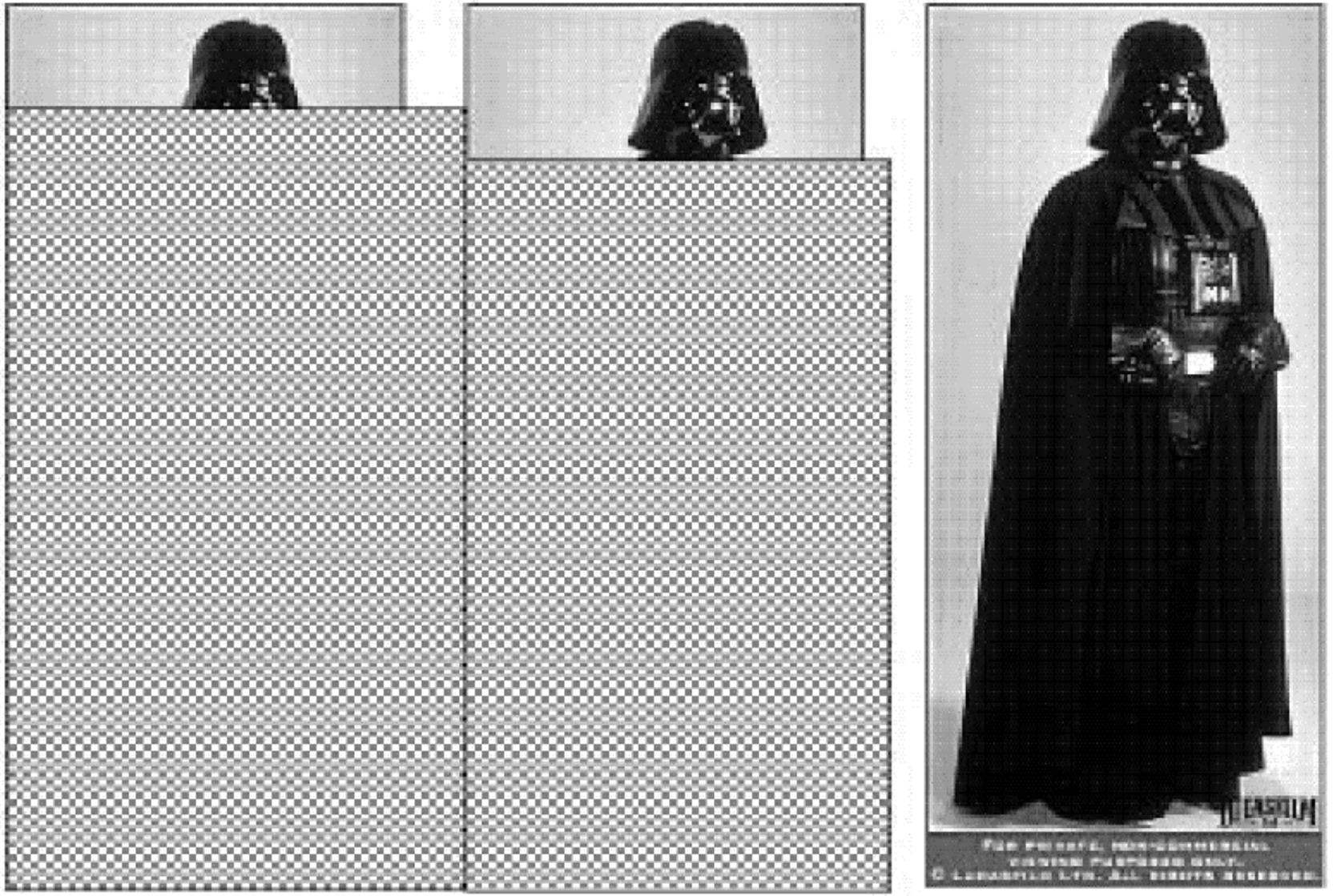
Arms Race: Polymorphic Code

- Given polymorphism, how might we then detect viruses?
- Idea #1: use narrow sig. that targets decryptor
  - Issues?
    - Less code to match against ⇒ more false positives
    - Virus writer spreads decryptor across existing code
- Idea #2: execute (or statically analyze) suspect code to see if it decrypts!
  - Issues?
    - Legitimate "*packers*" perform similar operations (decompression)
    - How long do you let the new code execute?
      - If decryptor only acts after lengthy legit execution, difficult to spot
- Virus-writer countermeasures?

Metamorphic Code

- Idea: every time the virus propagates, generate *semantically* different version of it!
  - Different semantics only at immediate level of execution; higher-level semantics remain same
- How could you do this?
- Include with the virus a code rewriter:
  - Inspects its own code, generates random variant, e.g.:
    - Renumber registers
    - Change order of conditional code
    - Reorder operations not dependent on one another
    - Replace one low-level algorithm with another
    - Remove some do-nothing padding and replace with different do-nothing padding ("chaff")
      - Can be very complex, legit code … if it's never called!

Polymorphic Code In Action



Hunting for Metamorphic, Szor & Ferrie, Symantec Corp., Virus Bulletin Conference, 2001

Metamorphic Code In Action

Hunting for Metamorphic, Szor & Ferrie, Symantec Corp., Virus Bulletin Conference, 2001

Detecting Metamorphic Viruses?

- Need to analyze execution behavior
  - Shift from syntax (*appearance* of instructions) to semantics (*effect* of instructions)
- Two stages: (1) AV company analyzes new virus to find behavioral signature; (2) AV software on end systems analyze suspect code to test for match to signature
- What countermeasures will the virus writer take?
  - Delay analysis by taking a long time to manifest behavior
    - Long time = await particular condition, or even simply clock time
  - Detect that execution occurs in an analyzed environment and if so behave differently
    - E.g., test whether running inside a debugger, or in a Virtual Machine
- Counter-countermeasure?
  - AV analysis looks for these tactics and skips over them
- Note: attacker has edge as *AV products supply an **oracle***

How Much Malware Is Out There?

- A final consideration regarding polymorphism and metamorphism:
  - Presence can lead to mis-counting a single virus outbreak as instead reflecting 1,000s of *seemingly different* viruses

- Thus take care in interpreting vendor statistics on malcode varieties
  - (Also note: public perception that many varieties exist is *in the vendors' own interest*)

Infection Cleanup

- Once malware detected on a system, how do we get rid of it?
- May require restoring/repairing many files
  - This is part of what AV companies sell: per-specimen disinfection procedures
- What about if malware executed with adminstrator privileges?

  "*nuke the entire site from orbit. It's the only way to be sure*"
  - Aliens

  - i.e., rebuild system from **original media + data backups**
- Malware may include a **rootkit**: *kernel patches* to hide its presence (its existence on disk, processes)

Infection Cleanup, con't

- If we have complete source code for system, we could rebuild from that instead, couldn't we?
- No!
- Suppose forensic analysis shows that virus introduced a backdoor in `/bin/login` executable
  - (Note: this threat isn't specific to viruses; applies to any malware)
- Cleanup procedure: rebuild `/bin/login` from source …

/bin/login
source code

Compiler

/bin/login
executable

Regular compilation process of building login binary from source code

/bin/login
source code

Compiler

/bin/login
executable

Infected compiler *recognizes* when it's compiling /bin/login source and inserts extra backdoor when seen

Correct compiler source code

No problem: first step, rebuild the compiler so it's uninfected

Infected Compiler

Correct compiler executable

Correct compiler source code

Oops - infected compiler recognizes when it's compiling its own source and inserts the infection!

Infected Compiler

Infected Compiler

No amount of careful source-code scrutiny can prevent this problem.
And if the *hardware* has a back door …

Reflections on Trusting Trust
Turing-Award Lecture, Ken Thompson, 1983

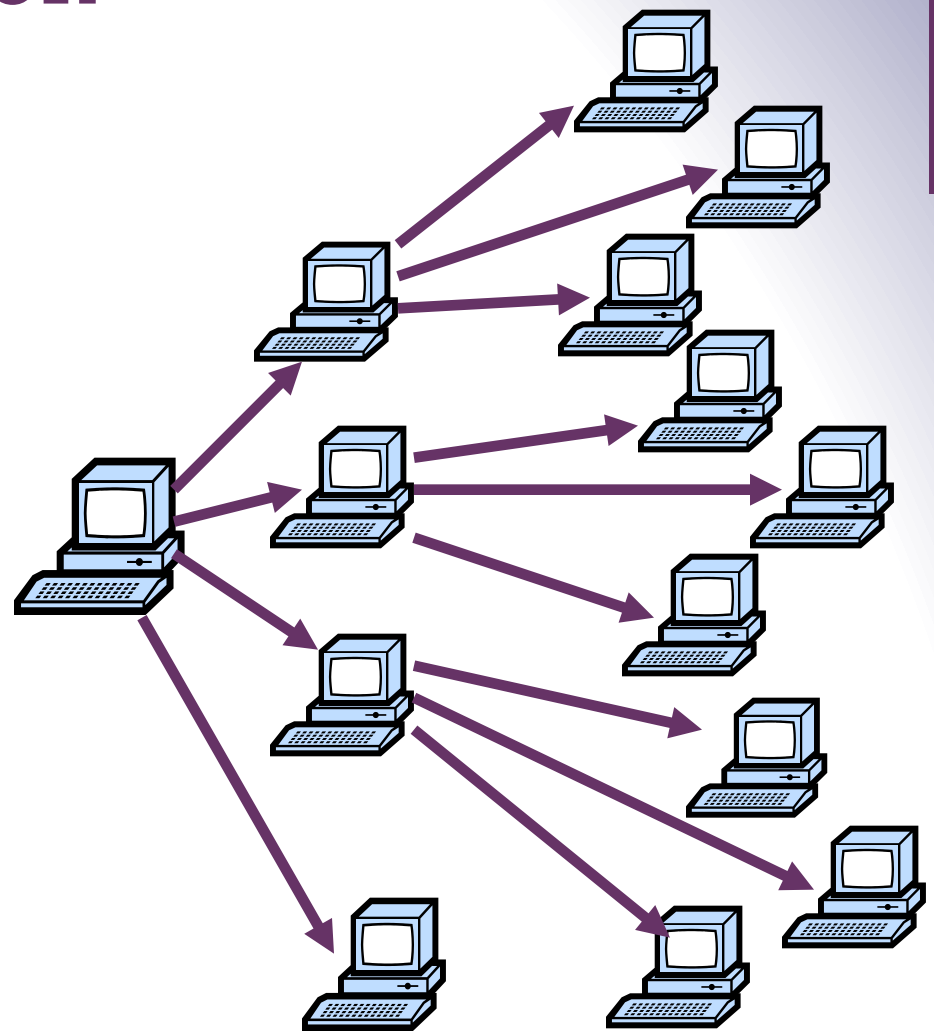
Large-Scale Malware

- Worm = code that self-propagates/replicates across systems by arranging to have itself immediately executed
  - Generally infects by altering running code
  - No user intervention required

Rapid Propagation

Worms can potentially spread quickly because they parallelize the process of propagating/ replicating.

Same holds for viruses, but they often spread more slowly since require some sort of user action to trigger each propagation.



Large-Scale Malware

- Worm = code that self-propagates/replicates across systems by arranging to have itself immediately executed
  - Generally infects by altering running code
  - No user intervention required
- Propagation includes notions of *targeting* & *exploit*
  - How does the worm **find** new prospective victims?
  - How does worm get code to **automatically run**?

- Botnet = set of compromised machines ("bots") under a common *command-and-control* (**C&C**)
  - Attacker might use a worm to get the bots, or other techniques; orthogonal to bot's use in botnet

The Arrival of Internet Worms

- Worms date to Nov 2, 1988 - the *Morris Worm*
- ***Way*** ahead of its time
- Employed whole suite of tricks to infect systems …
  - *Multiple* buffer overflows
  - Guessable passwords
  - "Debug" configuration option that provided shell access
  - Common user accounts across multiple machines
- … and of tricks to find victims
  - Scan local subnet
  - Machines listed in system's network config
  - Look through user files for mention of remote hosts

Love Letter Worm, aka «ILOVEYOU»

- On 5th May 2000, arrived as email with a Visual Basic Script attachment
- Exploited Window's extension hiding to display a fake "txt" extension for the original file iloveyou.txt.vbs
- Propagates by email to all addresses in the address book
- Also propagates through IRC
- Modifies IE's home page
- Replaces several different kinds of files with copies of itself

ILOVEYOU, con't

American parliament science committee:

- "In one day's time, roughly 47 million people received the e-mail worldwide and the virus looked for love in all the wrong places in over 10 million computers. [...] Insurance giant Lloyd's of London has estimated the virus will cost over $15 billion in damages and lost productivity"

Modern Era of Internet Worms

- Modern Era began Jul 13, 2001 with release of initial version of Code Red
- Exploited known buffer overflow in Microsoft IIS Web servers
  - *On by default* in many systems
  - Vulnerability & fix announced previous month
- Payload part 1: web site defacement
  - **`HELLO! Welcome to http://www.worm.com! Hacked By Chinese!`**
  - Only done if language setting = English

+

Code Red's exploit


```
/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090%u6858%ucbd3%u78
01%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%
u9090%u9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u00
78%u0000%u00=a
```

- More: http://www.caida.org/analysis/security/code-red/

Code Red of Jul 13 2001, con't

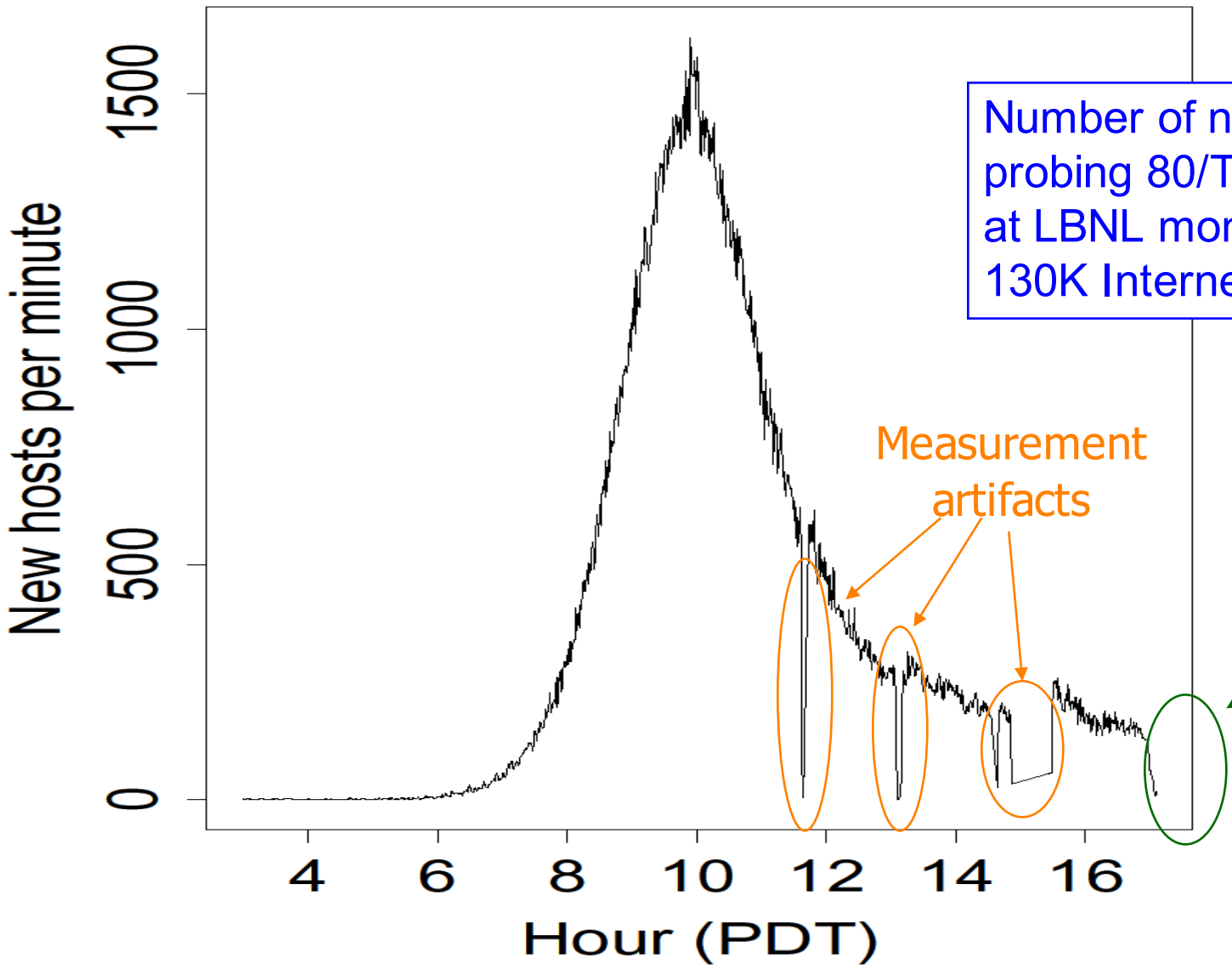
- Payload part 2: check day-of-the-month and …
  - … 1st through 20th of each month: spread
  - … 20th through end of each month: attack
    - Flooding attack against 198.137.240.91 …
    - … i.e., *www.whitehouse.gov*
- Spread: via *random scanning* of 32-bit IP address space
  - Generate pseudo-random 32-bit number; try connecting to it; if successful, try infecting it; repeat
  - Very common (but not fundamental) worm technique
- Each instance used same random number seed
  - How well does the worm spread?

Linear growth rate

Code Red, con't

- Revision released July 19, 2001.
- White House responds to threat of flooding attack by changing the address of *www.whitehouse.gov*
- Causes Code Red to die for date ≥ 20th of the month due to failure of TCP connection to establish.
  - Author didn't carefully test their code - buggy!
- But: this time random number generator correctly seeded. Bingo!

Growth of Code Red Worm



Modeling Worm Spread

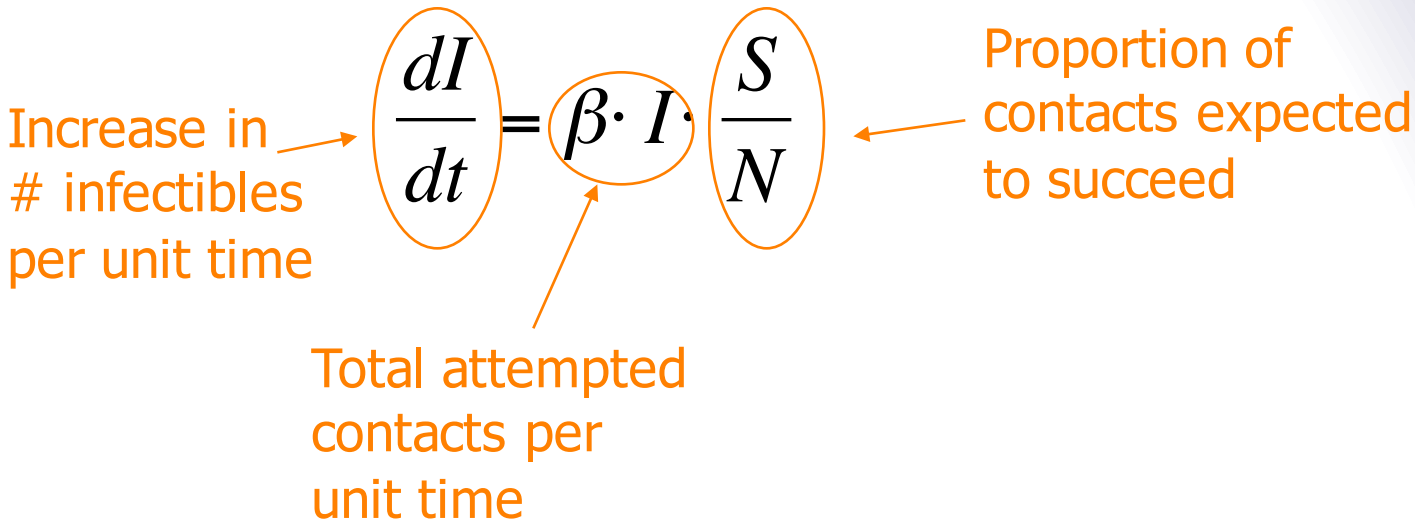
- Worm-spread often well described as *infectious epidemic*
  - Classic SI model: homogeneous random contacts
    - SI = Susceptible-Infectible
- Model parameters:
  - N: population size
  - S(t): susceptible hosts at time t.
  - I(t): infected hosts at time t.
  - β: *contact rate*
    - How many population members each infected host communicates with per unit time
    - E.g., if each infected host scans 10 Internet addresses per unit time, and 2% of Internet addresses run a vulnerable server $\Rightarrow \beta = 0.2$

$$N = S(t) + I(t)$$
$$S(0) = S_0 \; I(0) = I_0$$

- Normalized versions reflecting relative proportion of infected/susceptible hosts
  - $s(t) = S(t)/N \quad i(t) = I(t)/N \quad s(t) + i(t) = 1$

Computing How An Epidemic Progresses

- In continuous time:

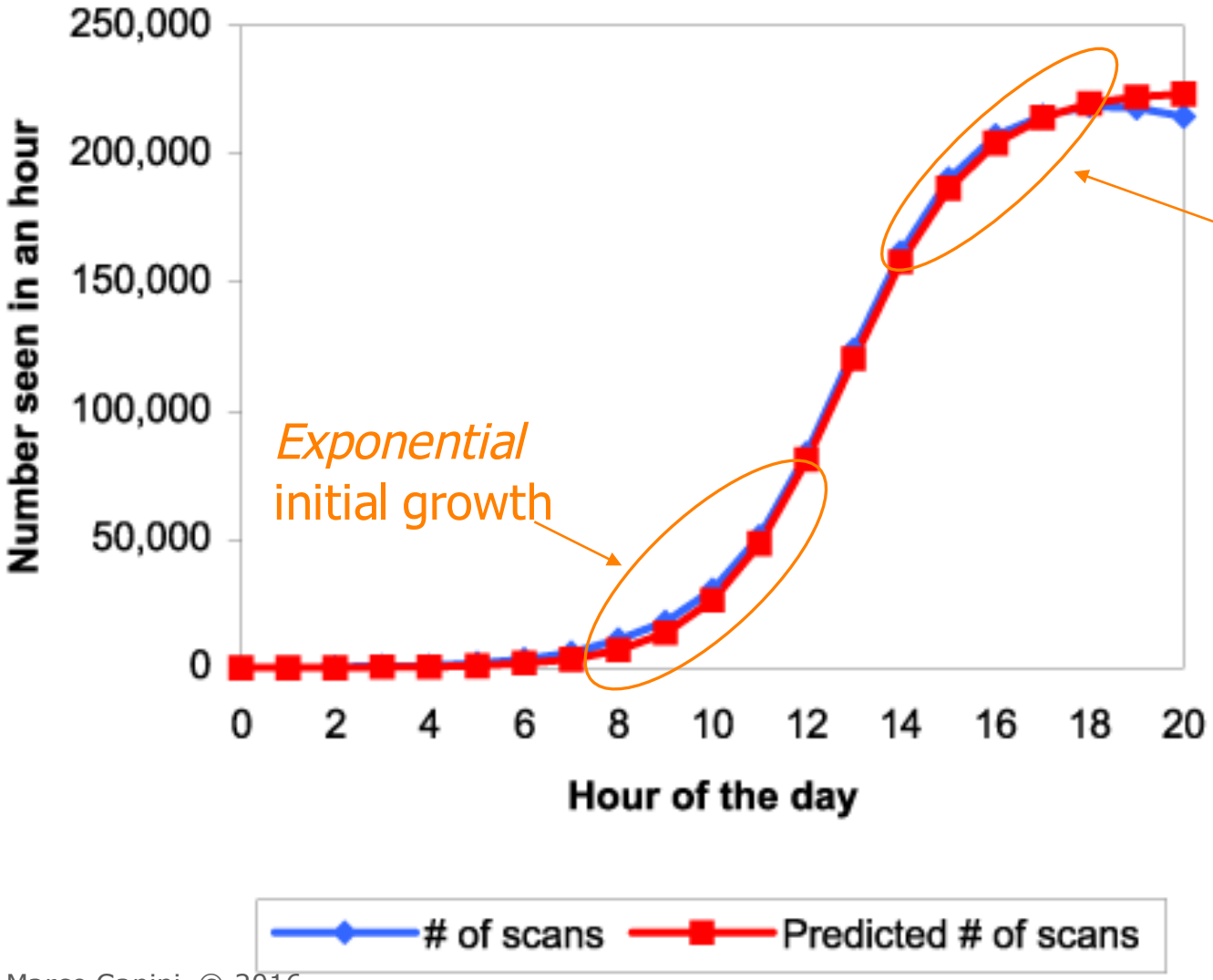


- Rewriting by using i(t) = I(t)/N, S = N - I:

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

Fraction infected grows as a *logistic*

Fitting the Model to Code Red



Spread of Code Red, con't

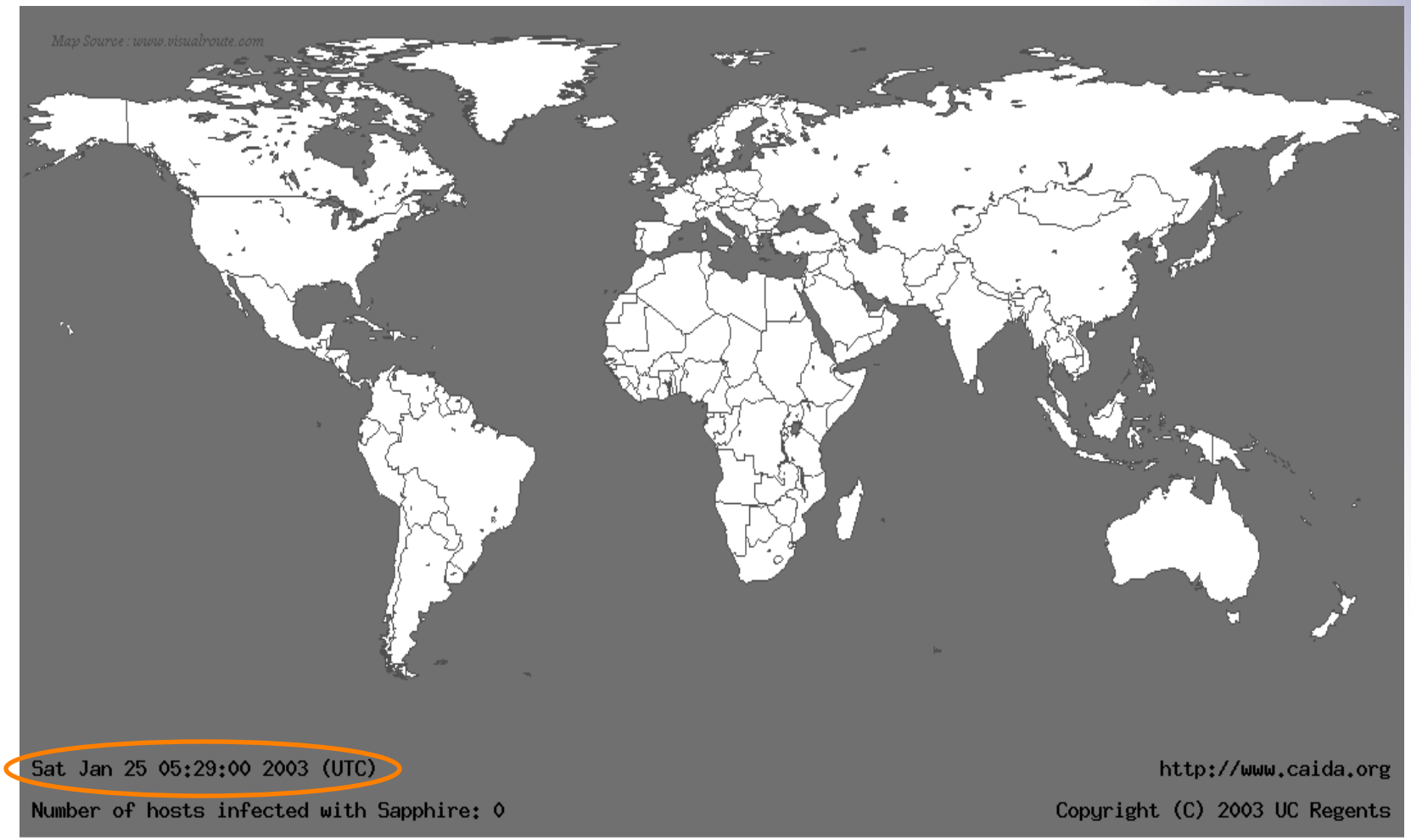
- Recall that # of new infections scales with contact rate β

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

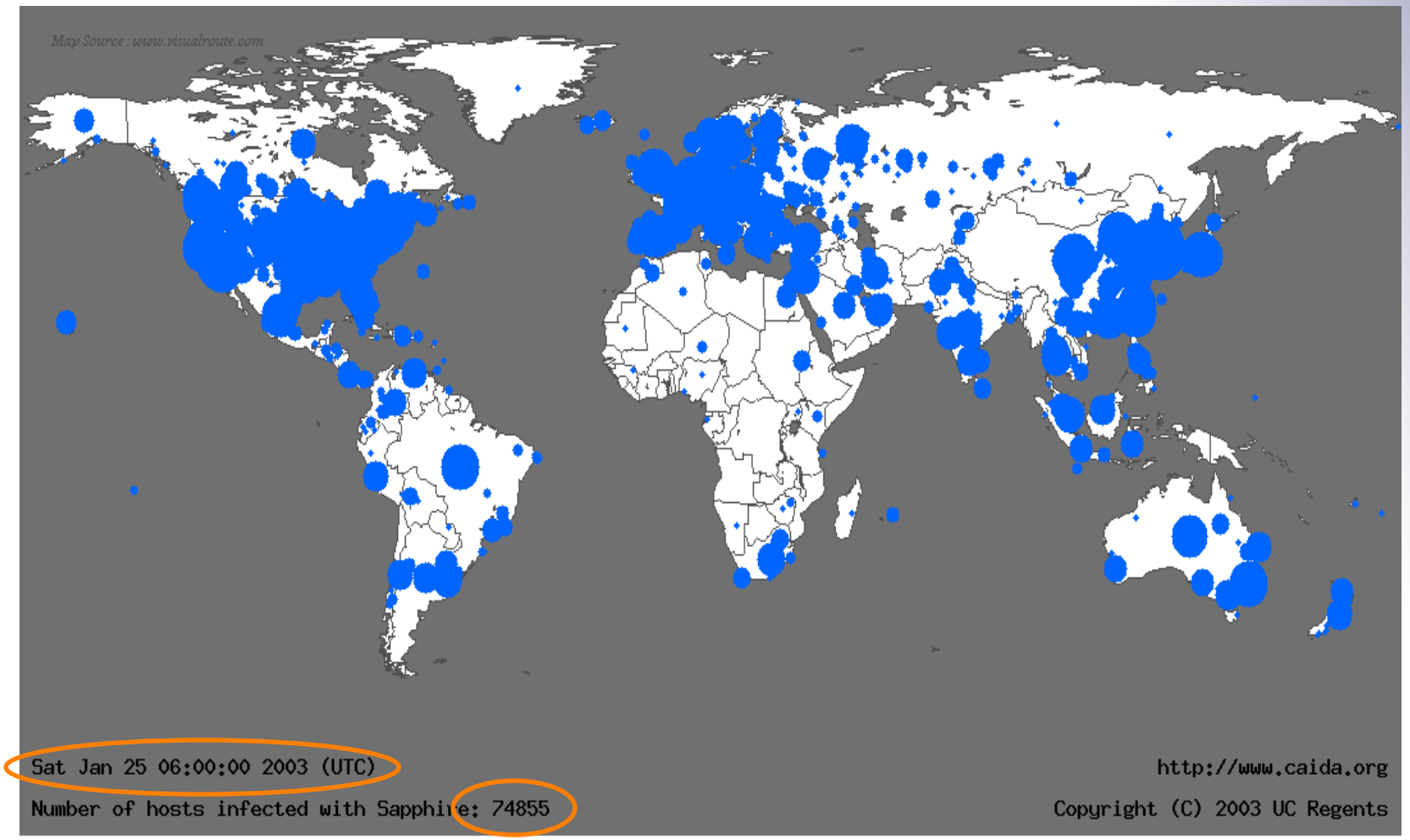
- For a scanning worm, β *increases* with N
  - Larger populations infected more quickly!
    - More likely that a given scan finds a population member
- Large-scale monitoring finds 360K systems infected with Code Red on July 19
  - Worm got them in 13 hours
- That night (⇒ 20th), worm dies due to DoS bug
- Worm actually managed to restart itself Aug. 1
  - … and each successive month for years to come!

Emergent behavior

Life Just Before Slammer



Life Just After Slammer



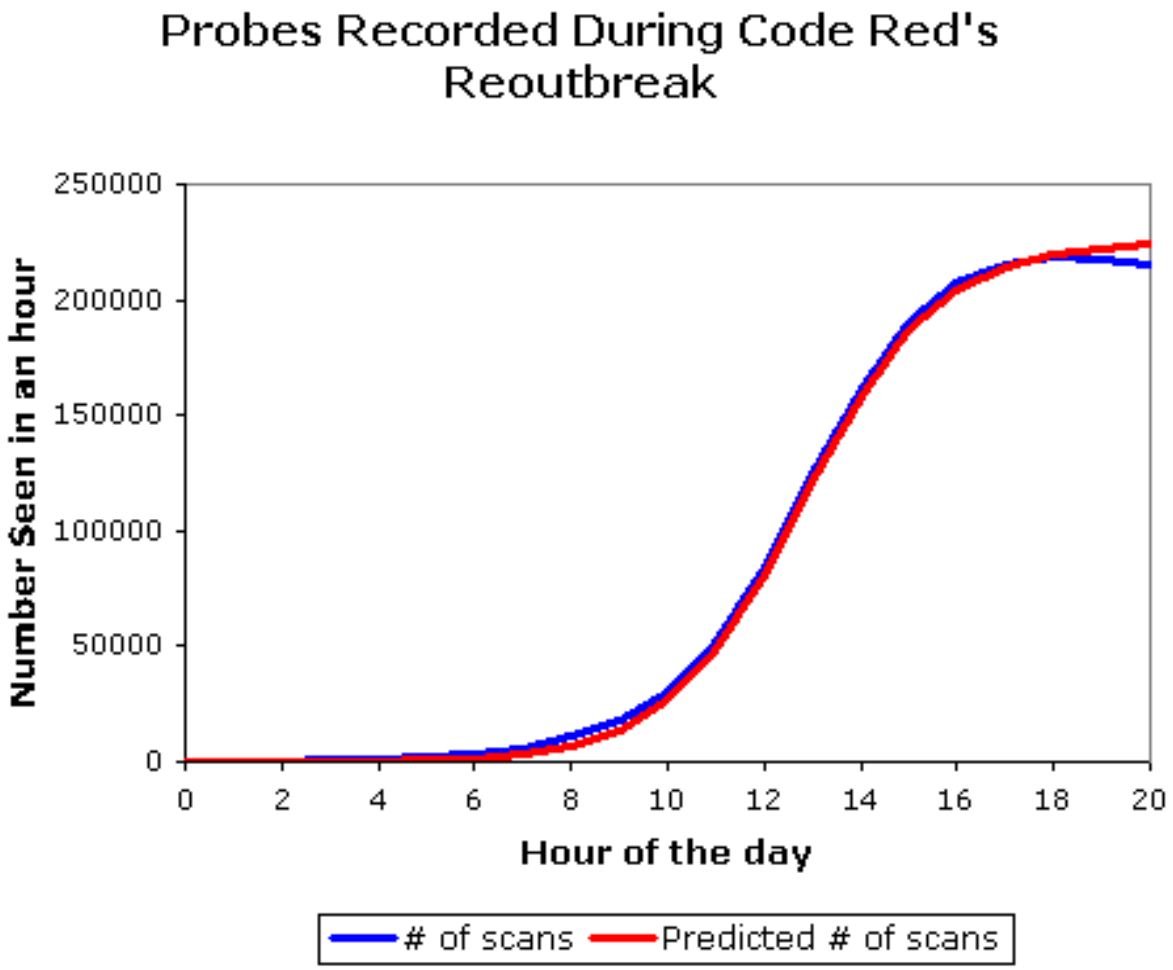
Going Fast: Slammer

- Slammer exploited connectionless UDP service, rather than connection-oriented TCP
- *Entire worm* fits in a single packet!

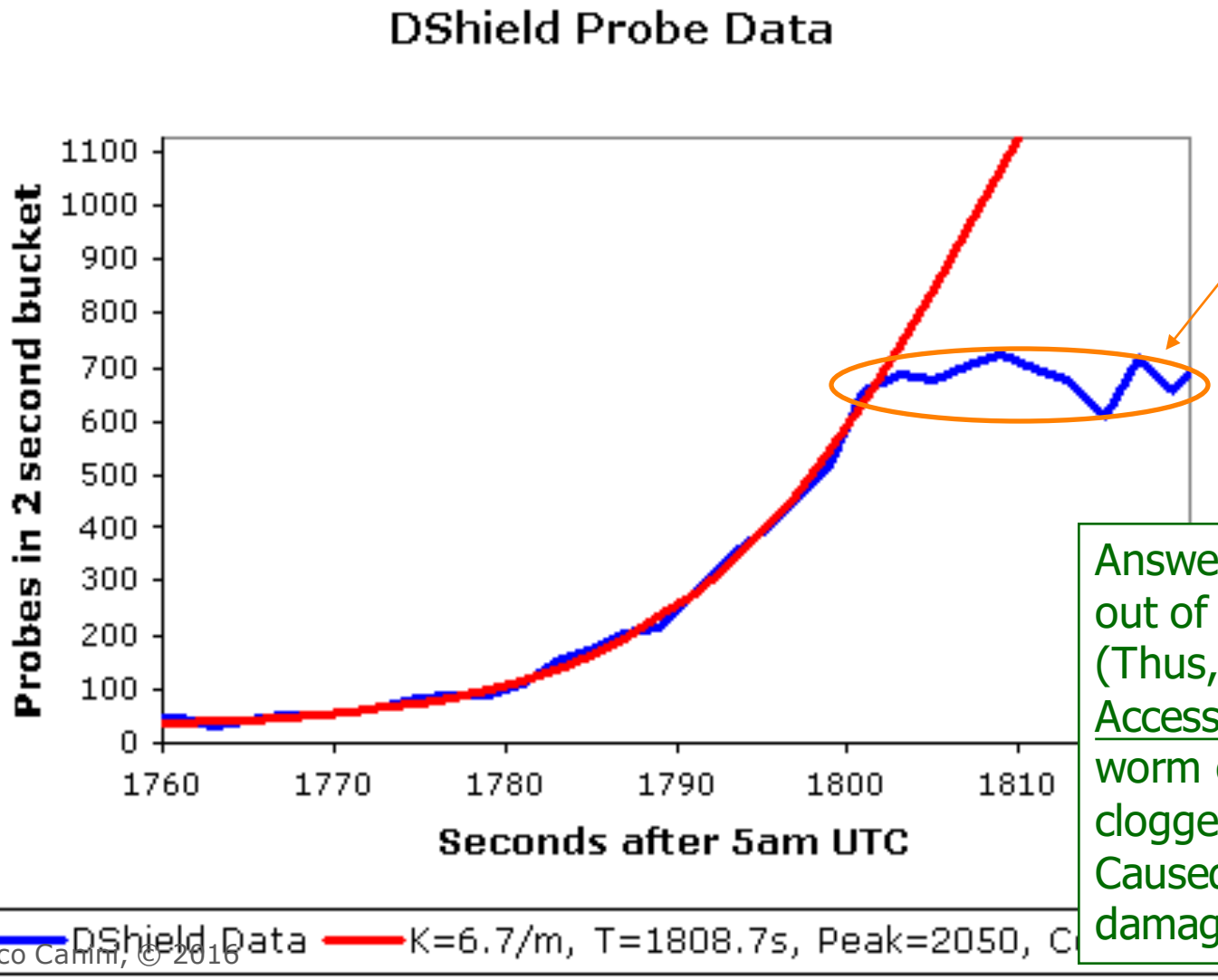
⇒ When scanning, worm could "fire and forget"
Stateless!

- Worm infected 75,000+ hosts in *<< 10 minutes*
- At its peak, **doubled every 8.5 seconds**

The Usual Logistic Growth



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/s*

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

Stuxnet

- Discovered July 2010. (Released: Mar 2010?)
- Multi-mode spreading:
  - Initially spreads via USB (virus-like)
  - Once inside a network, quickly spreads internally using Windows RPC
- Kill switch: programmed to die June 24, 2012
- Targeted *SCADA* systems
  - Used for industrial control systems, like manufacturing, power plants
- Symantec: infections geographically clustered
  - Iran: 59%; Indonesia: 18%; India: 8%

Stuxnet, con't

- Used four *Zero Days*
  - Unprecedented expense on the part of the author
- "Rootkit" for hiding infection based on installing Windows drivers with valid digital signatures
  - Attacker stole private keys for certificates from two companies in Taiwan
- Payload: do nothing …
  - … unless attached to particular models of frequency converter drives operating at 807-1210Hz
  - … like those made in Iran (and Finland) …
  - … and used to operate centrifuges for producing *enriched uranium for nuclear weapons*

Stuxnet, con't

- Payload: do nothing …
  - … unless attached to particular models of frequency converter drives operating at 807-1210Hz
  - … like those made in Iran (and Finland) …
  - … and used to operate centrifuges for producing *enriched uranium for nuclear weapons*
- For these, worm would slowly increase drive frequency to 1410Hz …
  - … enough to cause centrifuge to **fly apart** …
  - … while sending out fake readings from control system indicating everything was okay …
- … and then drop it back to normal range

Israel Tests on Worm Called Crucial in Iran Nuclear Delay

By WILLIAM J. BROAD, JOHN MARKOFF and DAVID E. SANGER
Published: January 15, 2011

*This article is by **William J. Broad**, **John Markoff** and **David E. Sanger**.*

Enlarge This Image

Nicholas Roberts for The New York Times

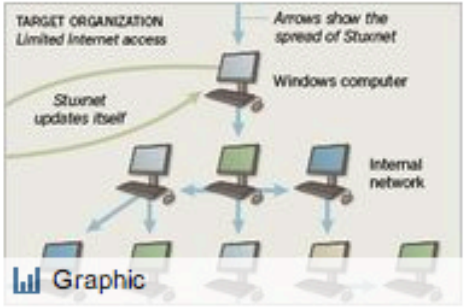
Ralph Langner, an independent computer security expert, solved Stuxnet.

The Dimona complex in the Negev desert is famous as the heavily guarded heart of Israel's never-acknowledged nuclear arms program, where neat rows of factories make atomic fuel for the arsenal.

Over the past two years, according to intelligence and military experts familiar with its operations, Dimona has taken on a new, equally secret role — as a critical testing ground in a joint American and Israeli effort to undermine Iran's efforts to make a bomb of its own.

Behind Dimona's barbed wire, the experts say, Israel has spun nuclear centrifuges virtually identical to Iran's at Natanz, where Iranian scientists are struggling to enrich uranium. They say Dimona tested the effectiveness of the Stuxnet computer worm, a destructive program that appears to have wiped out roughly a fifth of Iran's nuclear

Multimedia



How Stuxnet Spreads

Worm Take-Aways

- Potentially enormous reach/damage
  - ⇒ *Weapon*
- Hard to get right
- Emergent behavior / surprising dynamics
- Remanence: worms stick around
  - E.g. Slammer still seen in 2013!
- *Propagation faster than human response*

Any questions?

Stay tuned

+

Next time you will learn about

Cryptography