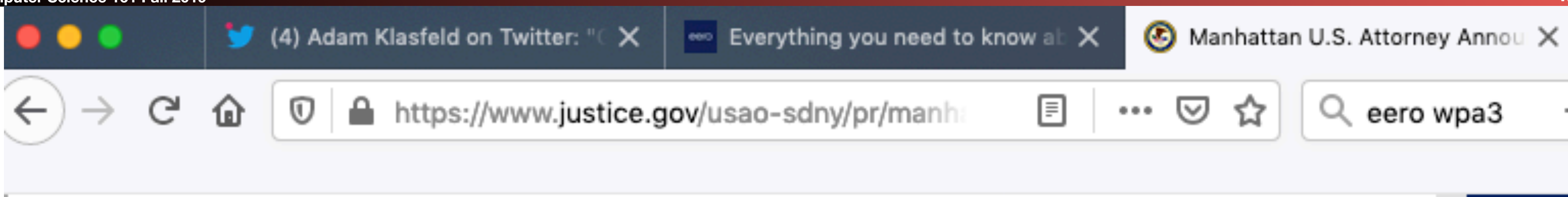
Malcode Continued

News of The Day...



Manhattan U.S. Attorney Announces Arrest Of United States Citizen For Assisting North Korea In Evading Sanctions

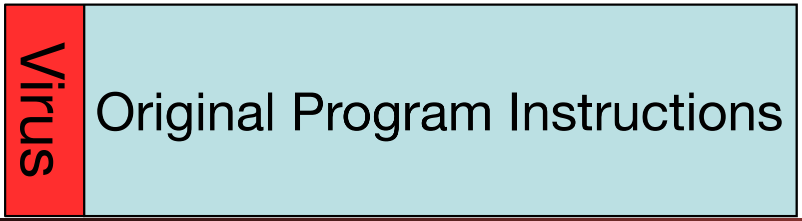
Geoffrey S. Berman, the United States Attorney for the Southern District of New York, John C. Demers, the Assistant Attorney General for National Security, John Brown, Assistant Director of the Federal Bureau of Investigation ("FBI") Counterintelligence Division, and William F. Sweeney Jr., the Assistant Director-in-Charge of the New York Field Office of the FBI, announced today the unsealing of a criminal complaint charging VIRGIL GRIFFITH, a United States citizen, with violating the International Emergency Economic Powers Act ("IEEPA") by traveling to the Democratic People's Republic of Korea ("DPRK" or "North Korea") in order deliver a presentation and technical advice on using cryptocurrency and blockchain technology to evade sanctions. GRIFFITH was arrested at Los Angeles International Airport yesterday and will be presented in federal court in Los Angeles on Monday, December 2.

Virus Writer / AV Arms Race

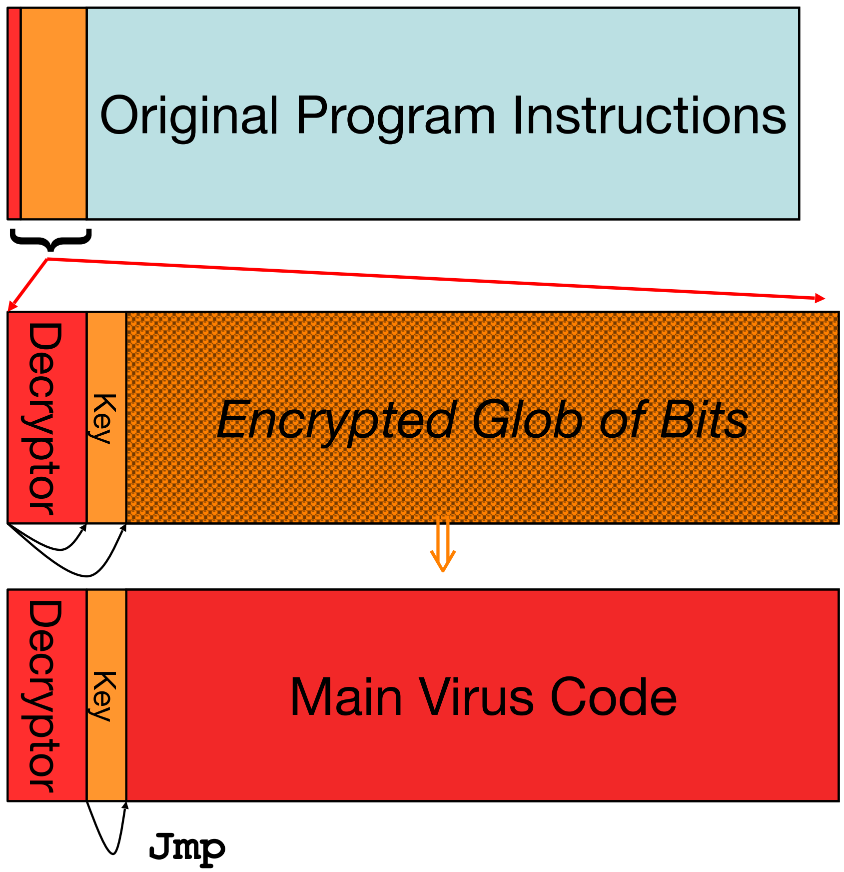
- If you are a virus writer and your beautiful new creations don't get very far because each time you write one, the AV companies quickly push out a signature for it ….
  - …. What are you going to do?
- Need to keep changing your viruses …
  - … or at least changing their appearance!
- How can you mechanize 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

- We've already seen technology for creating a representation of data apparently completely unrelated to the original: encryption!
- 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 (like an IV)
  - Note: weak (but simple/fast) crypto algorithm works fine
    - No need for truly strong encryption, just obfuscation
- When injected code runs, it decrypts itself to obtain the original functionality



Instead of this …

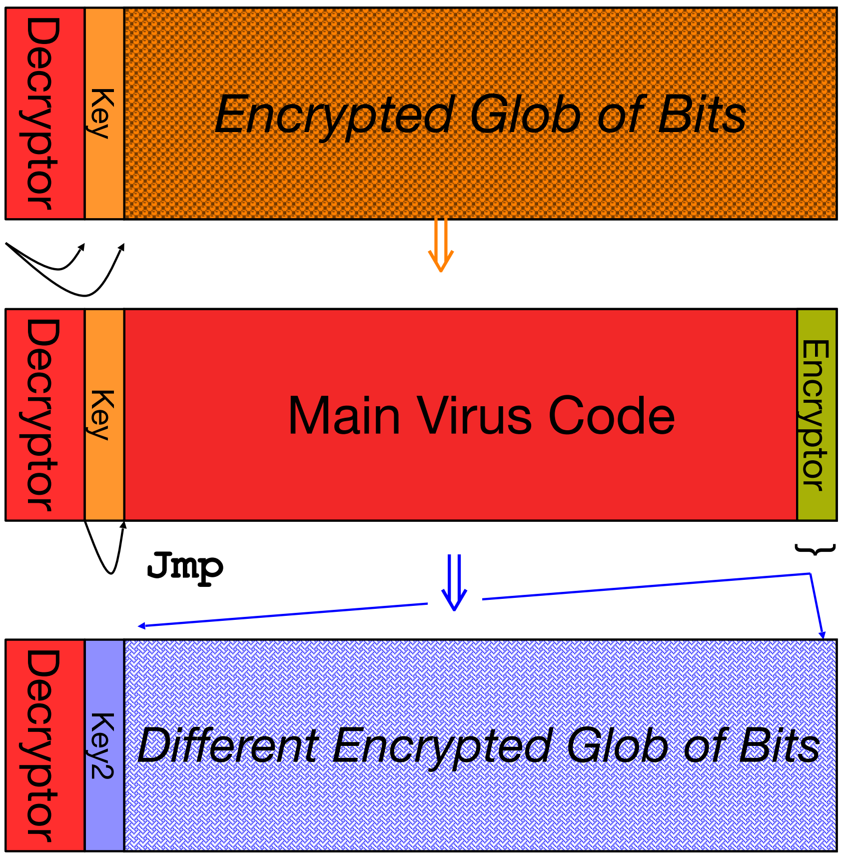


Virus has *this* initial structure

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

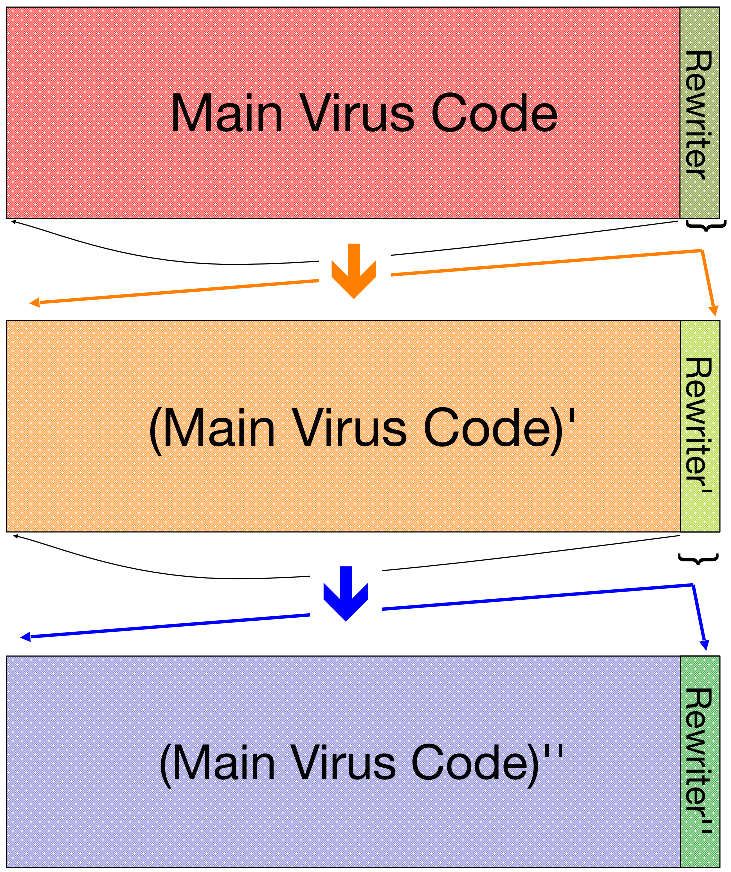
Arms Race: Polymorphic Code

- 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!

Metamorphic Propagation



When ready to propagate, virus invokes a randomized *rewriter* to construct new but semantically equivalent code (*<u>including the rewriter</u>*)

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

Malcode Wars and the Halting Problem...

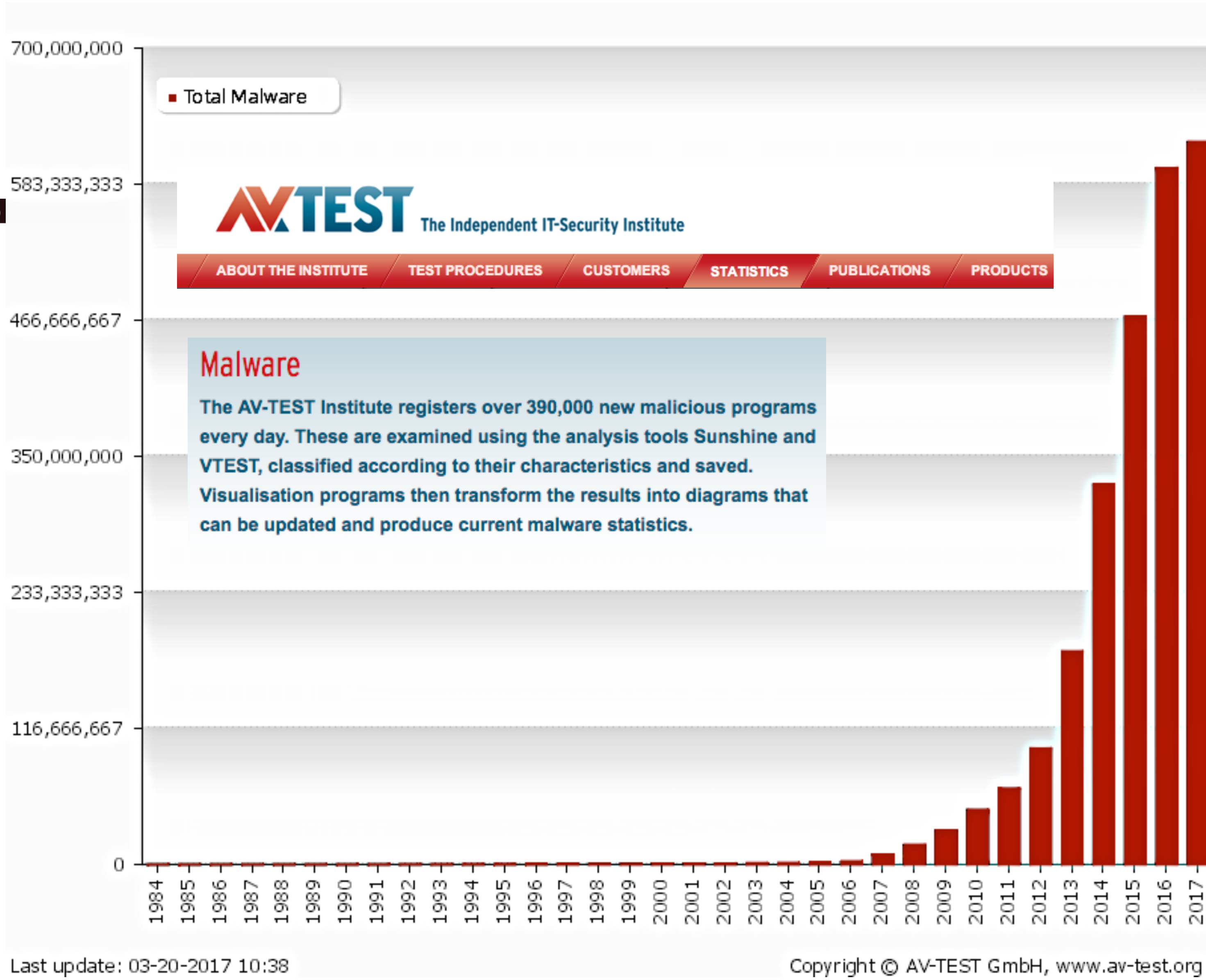
- Cyberwars are not won by solving the halting problem...
  Cyberwars are won by making some other poor sod solve the halting problem!!!
  - In the limit, it is ***undecidable*** to know "is this code bad?"
- Modern focus is instead "is this code ***new?***"
  - Use a secure cryptographic hash (so sha-256 not md5)
  - Check hash with central repository:
    If ***not*** seen before, treat binary as inherently more suspicious
- Creates a bind for attackers:
  - Don't make your code *morphic:
    Known bad signature detectors find it
  - Make your code *morphic:
    It always appears as new and therefore ***inherently*** suspicious

Creating binds is very powerful...

- You have a detector D for some bad behavior...
  - So bad-guys come up with a way of avoiding the detector D
- So come up with a detection strategy for ***avoiding detector D***
  - So to avoid ***this*** detector, the attacker ***must not*** try to avoid D
- When you can do it, it is very powerful!

How Much Malware Is Out There?

- A final consideration re 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 huge malware populations exist is in the vendors' own interest)



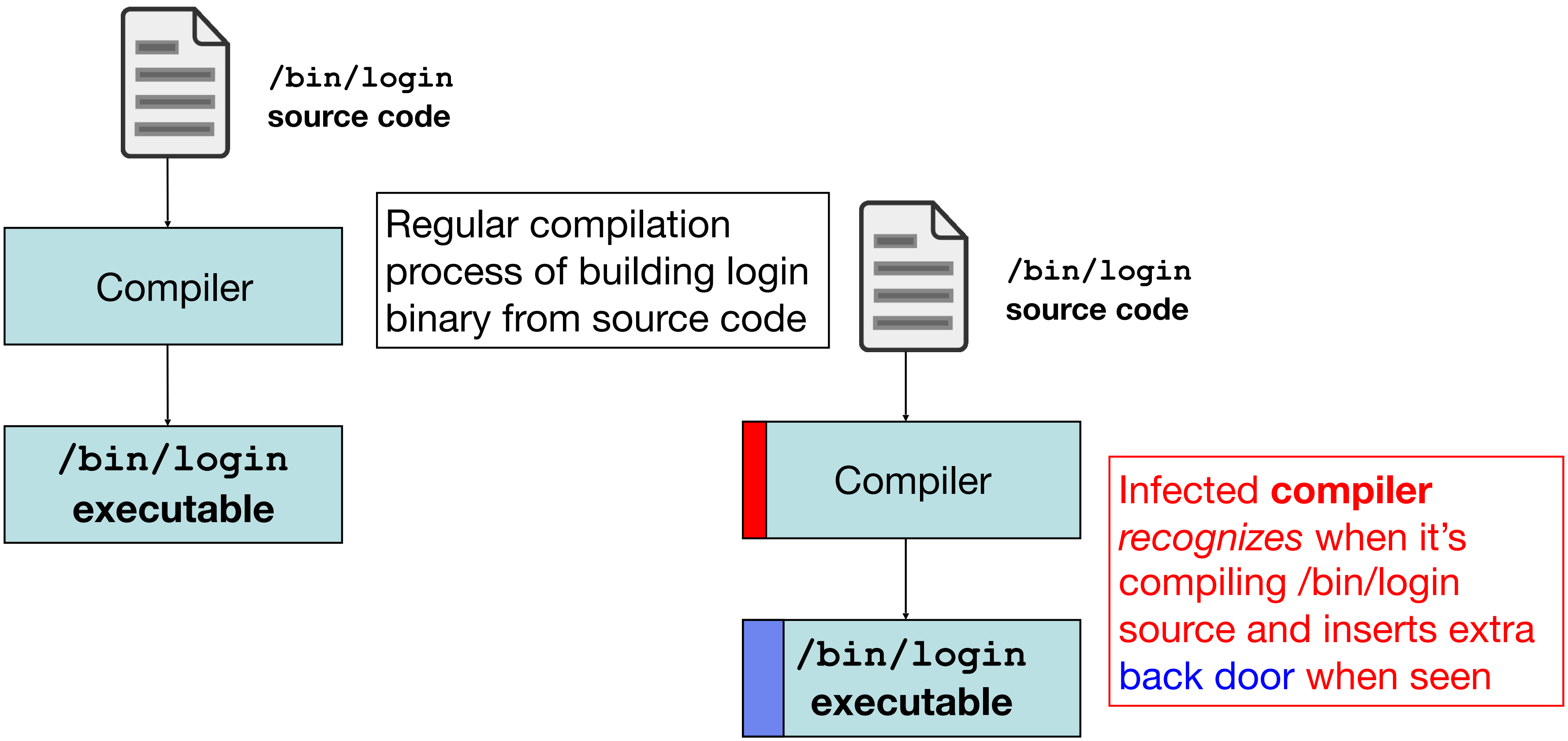
700,000,000
583,333,333
466,666,667
350,000,000
233,333,333
116,666,667
0
Total Malware
AV-TEST The Independent IT-Security Institute
ABOUT THE INSTITUTE
TEST PROCEDURES
CUSTOMERS
STATISTICS
PUBLICATIONS
PRODUCTS
Malware
The AV-TEST Institute registers over 390,000 new malicious programs every day. These are examined using the analysis tools Sunshine and VTEST, classified according to their characteristics and saved. Visualisation programs then transform the results into diagrams that can be updated and produce current malware statistics.
1984 1985 1986 1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017
Last update: 03-20-2017 10:38
Copyright © AV-TEST GmbH, www.av-test.org

Infection Cleanup

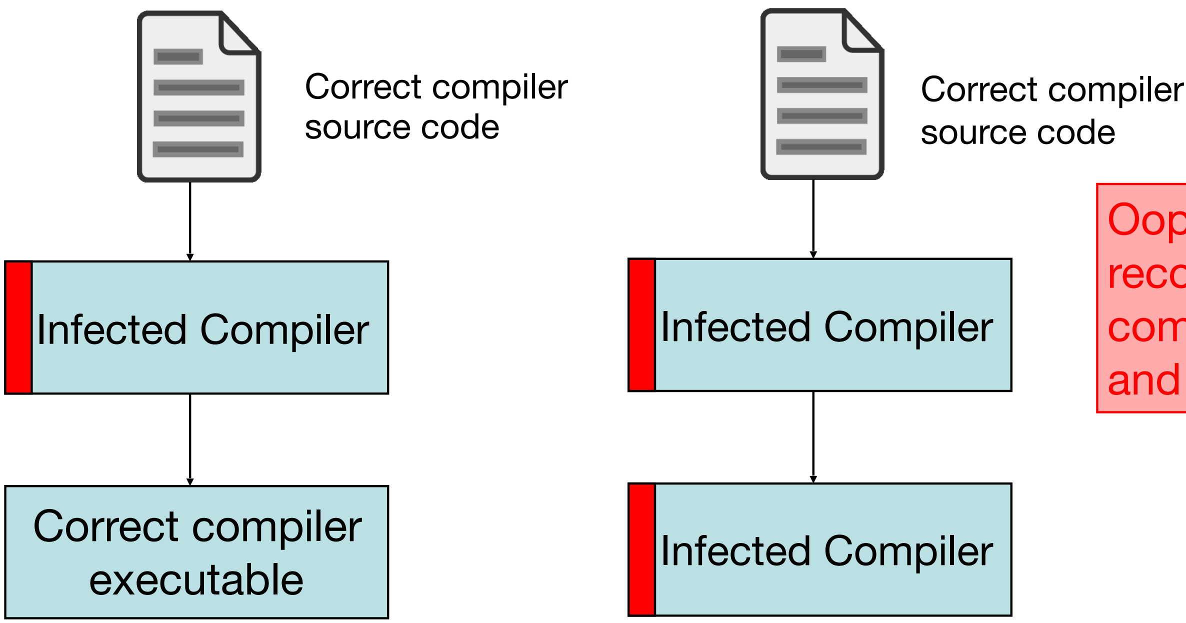
- Once malware detected on a system, how do we get rid of it?
- May require restoring/repairing many files
  - This is part of what AV companies sell: per-specimen disinfection procedures
- What about if malware executed with adminstrator privileges?
  - "Game over man, Game Over!"
  - “Dust off and 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 back door when seen

No probl irst step,
rebuild th ompiler so
it's unin cte



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

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

More On "Rootkits"

- If you control the operating system...
  - You can hide extremely well
- EG, your malcode is on disk...
  - So it will persist across reboots
- But if you try to ***read the disk***...
  - The operating system just says "Uhh, this doesn't exist!"

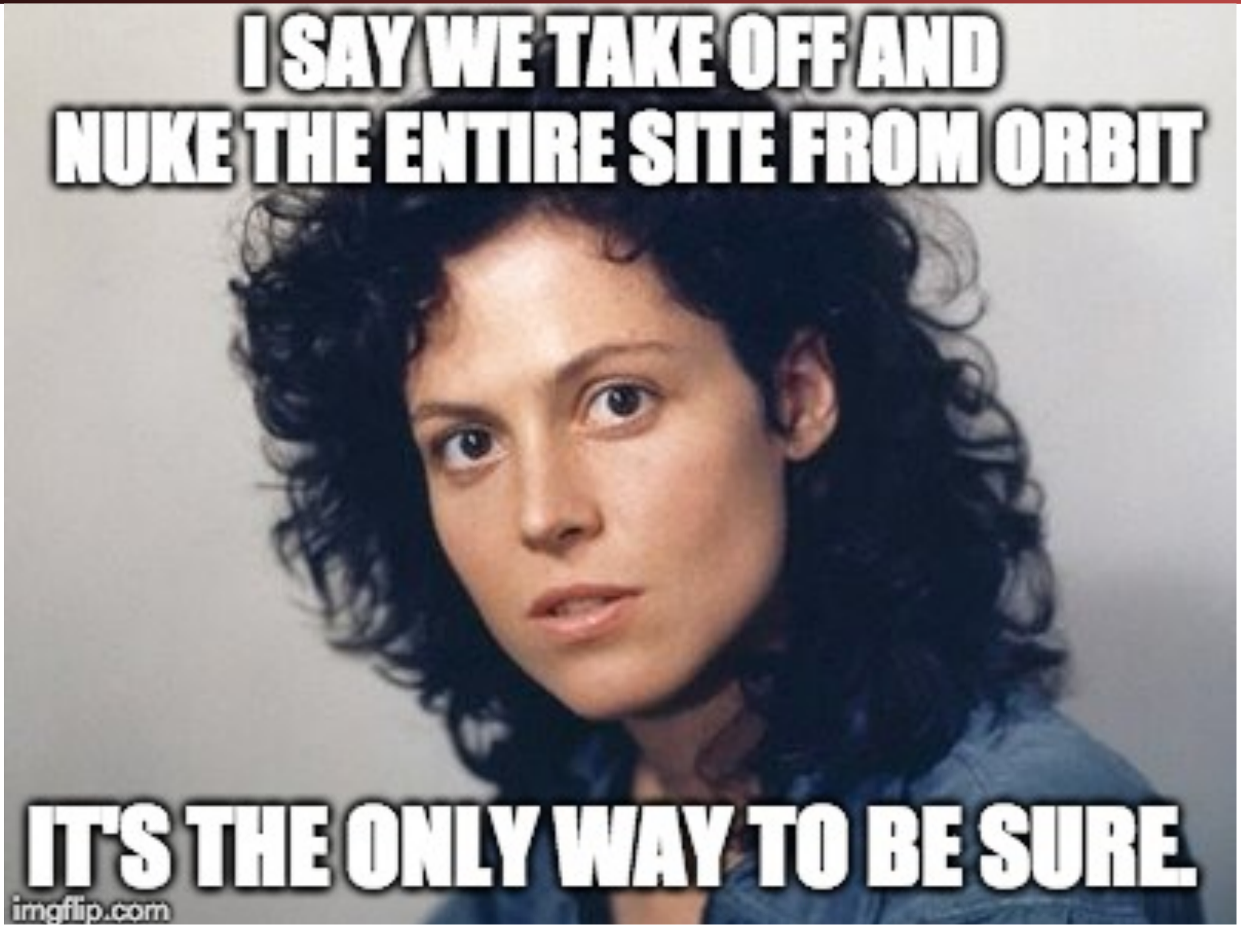
Even More Places To Hide!

- In the BIOS/EFI Firmware!
  - So you corrupt the BIOS which corrupts the OS...
  - Really hard to find:
    Defense, ***only*** run cryptographically signed BIOS code as part of the Trusted Base
- In the disk controller firmware!
  - So the master boot record, when read on boot up corrupts the OS...
  - But when you try to read the MBR later... It is just "normal"
  - Again, defense is ***signed code:*** The Firmware will only load a signed operating system
    - Make sure the disk itself is ***not trusted!***

Robust Rootkit Detection: Detect the act of hiding...

- Do an "in-system" scan of the disk...
  - Record it to a USB drive
- Reboot the system with trusted media
  - So a known good operating system
- Do the same scan!
  - If the scans are different, you found the rootkit!
- For windows, you can also do a "high/low scan" on the Registry:
  - Forces the bad guy to understand the registry as well as Mark Russinovich (the guy behind Sysinternals who's company Microsoft bought because he understood the Registry better than Microsoft's own employees!)
- Forces a bind on the attacker:
  - Hide and persist? You can be detected
  - Hide but don't persist? You can't survive reboots!

Which Means ***Proper*** Malcode Cleanup...



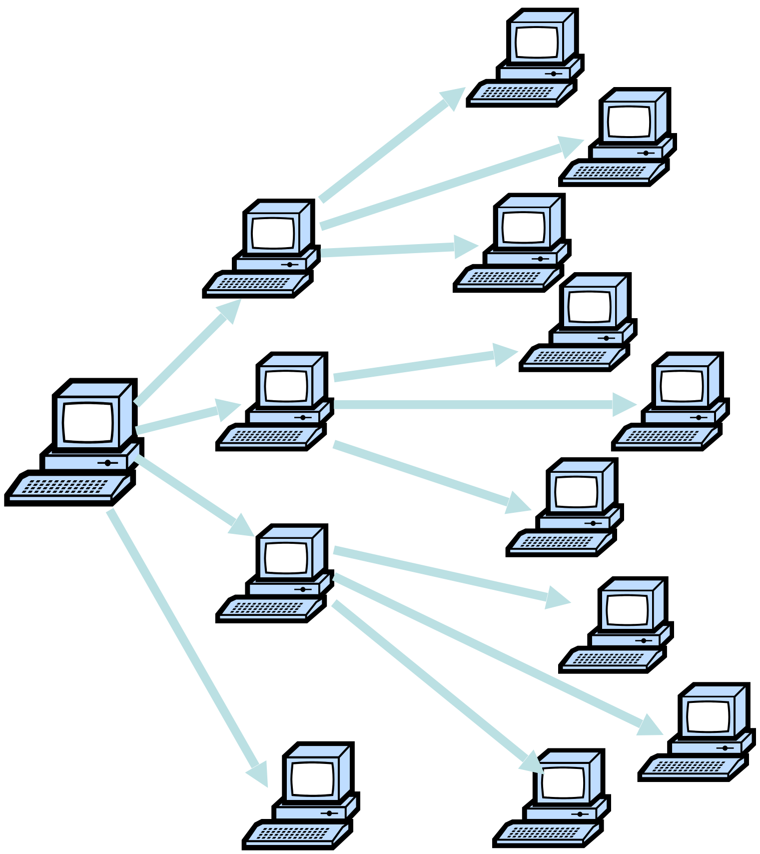
Large-Scale Malware

- 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

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.

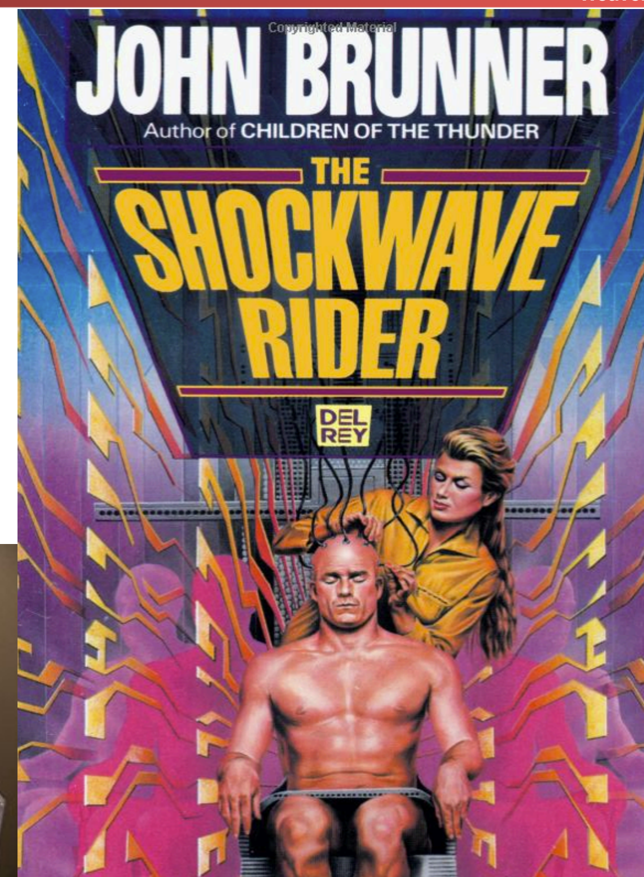


Worms

- 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?
    - One common approach: random scanning of 32-bit IP address space
      - Generate pseudo-random 32-bit number; try connecting to it; if successful, try infecting it; repeat
    - But for example "search worms" use Google results to find victims
  - How does worm get code to automatically run?
    - One common approach: buffer overflow ⇒ code injection
    - But for example a web worm might propagate using XSS

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



Arrival of Internet Worms, con't

- 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

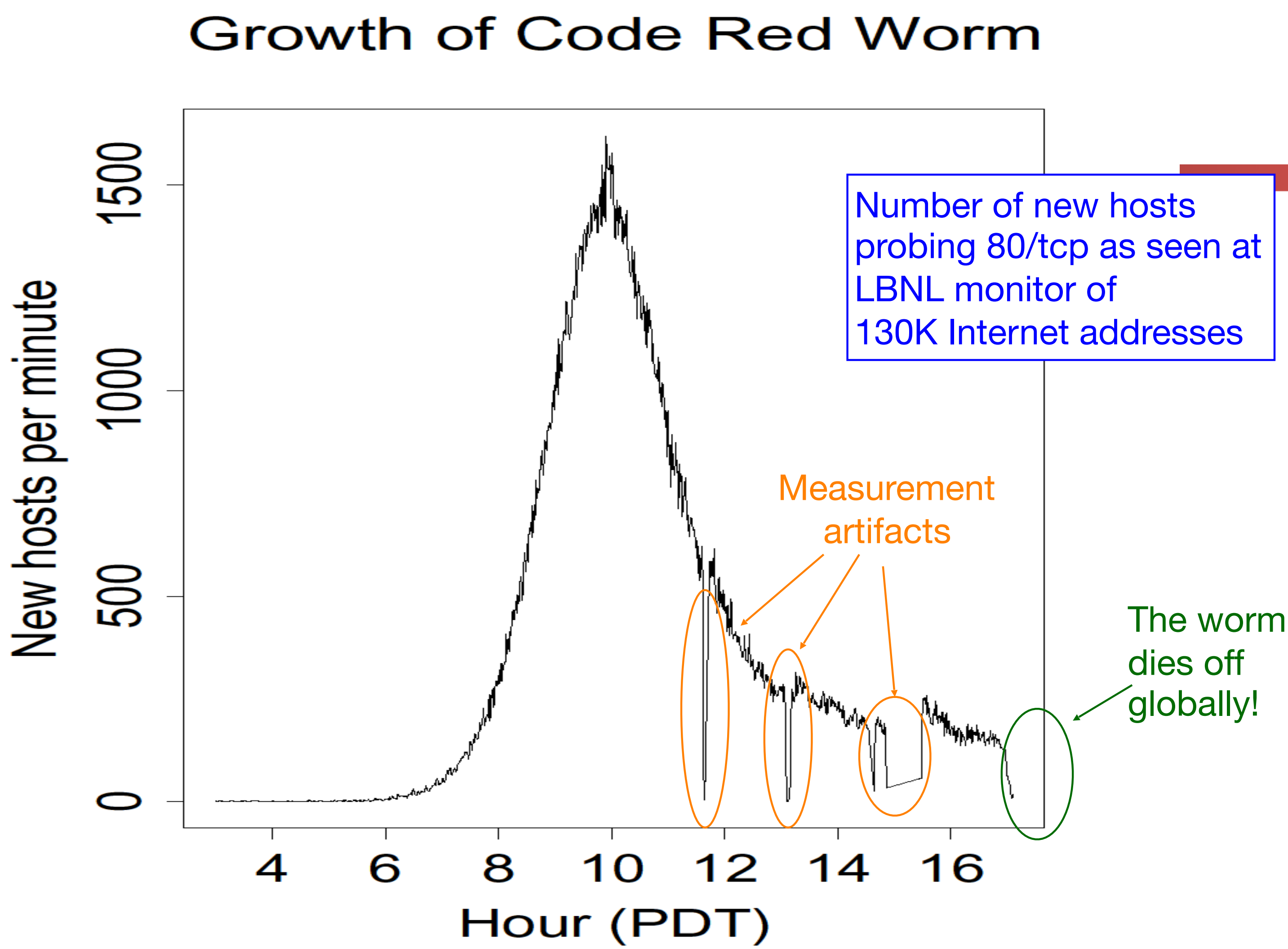
Code Red of Jul 13 2001, con't

- 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
New hosts per minute
0
500
1000
1500
4
6
8
10
12
14
16
Hour (PDT)
Number of new hosts probing 80/tcp as seen at LBNL monitor of 130K Internet addresses
Measurement artifacts
The worm dies off globally!

Nick's Reaction to Code Red

- Come on, we are computer people…
  - What do we do that EVER takes 13 hours?!?!?
- How to speed up
  - Preseed to skip the initial ramp-up
  - Scan faster (100x/second rather than 10x)
  - Scan smarter
    - Self-coordinated scanning techniques with shutoff strategies
  - Validated in ***simulation!***
- The “Warhol Worm” concept…
  - Implications that any worm defense needs to be automatic

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 250 Internet addresses per unit time, and 2% of Internet addresses run a vulnerable (maybe already infected) server ⇒ $\beta = 5$
    - For scanning worms, larger (= denser) vulnerable pop. ⇒ higher β ⇒ faster worm!

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

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

Computing How An Epidemic Progresses

- In continuous time:

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

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

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

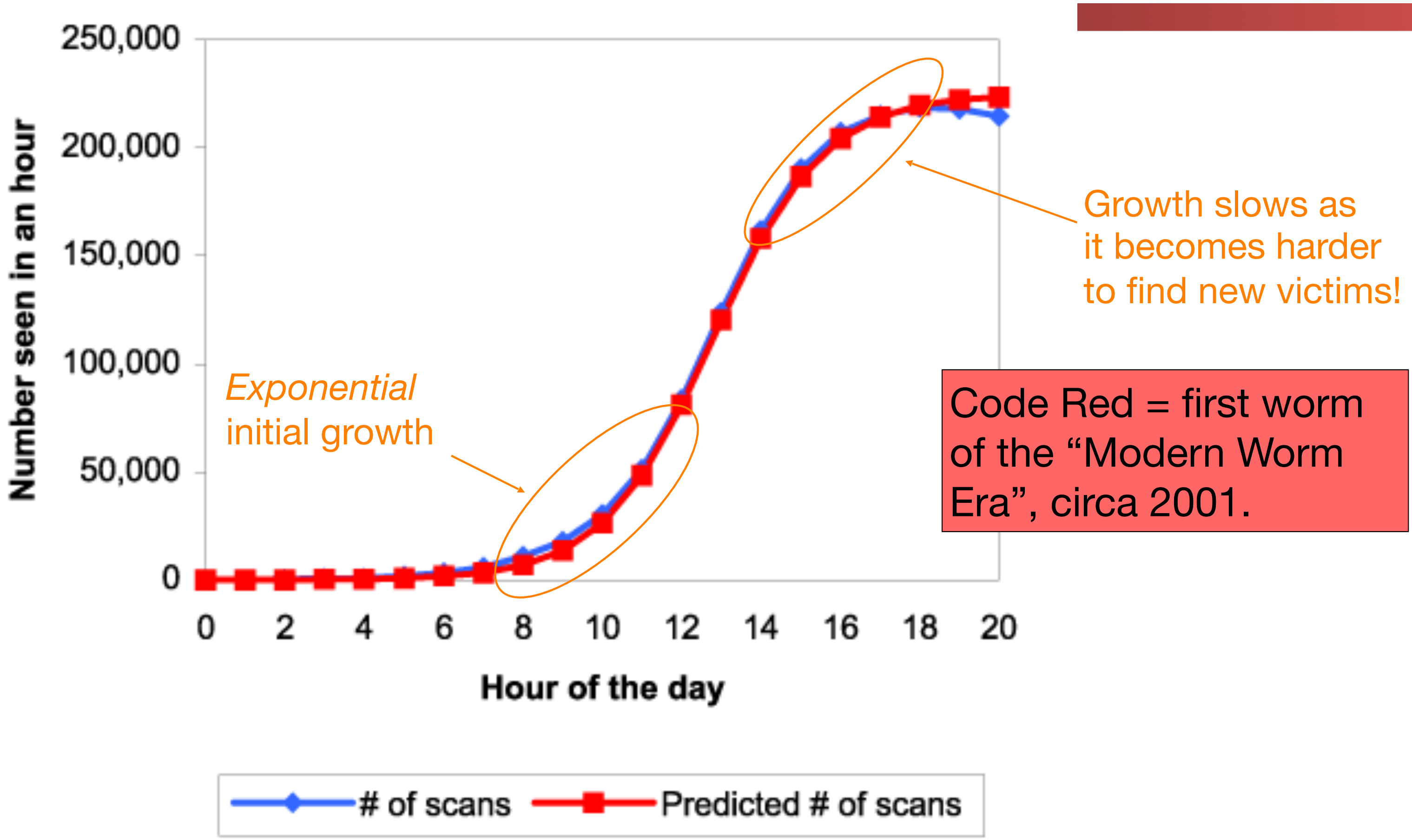
Proportion of contacts expected to succeed ($\frac{S}{N}$)

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

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

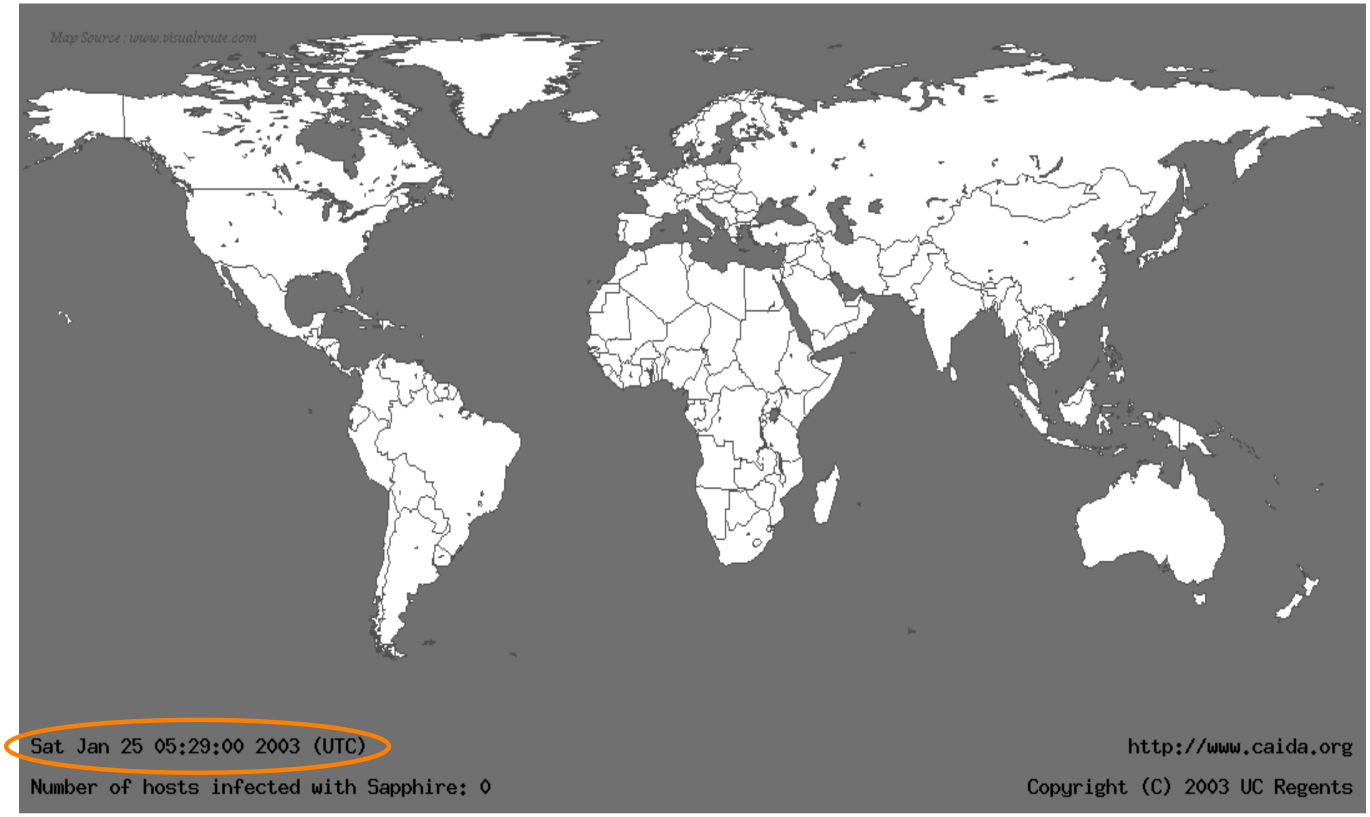
Fraction infected grows as a *logistic*

Fitting the Model to "Code Red"

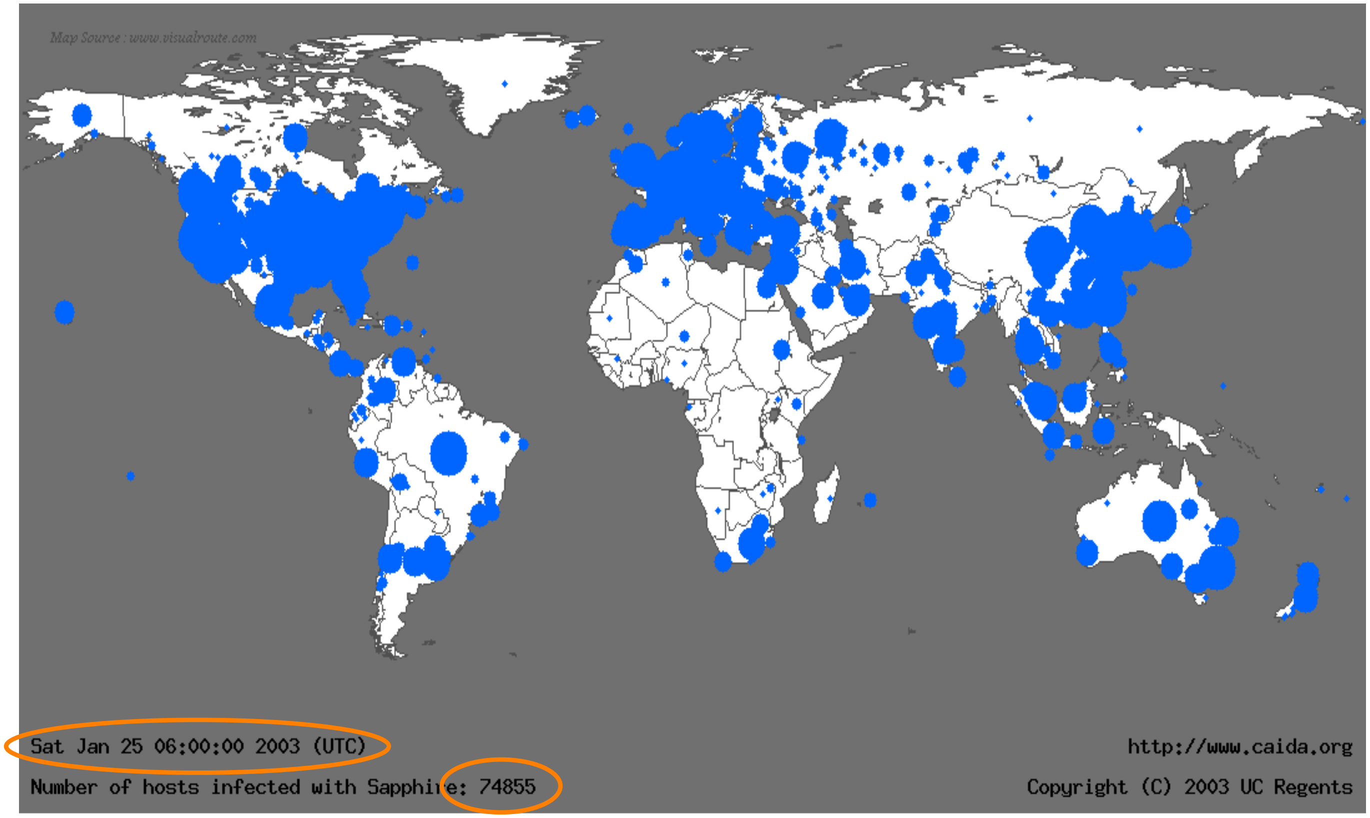


Code Red = first worm of the "Modern Worm Era", circa 2001.

Life Just Before Slammer



Life 10 Minutes After Slammer

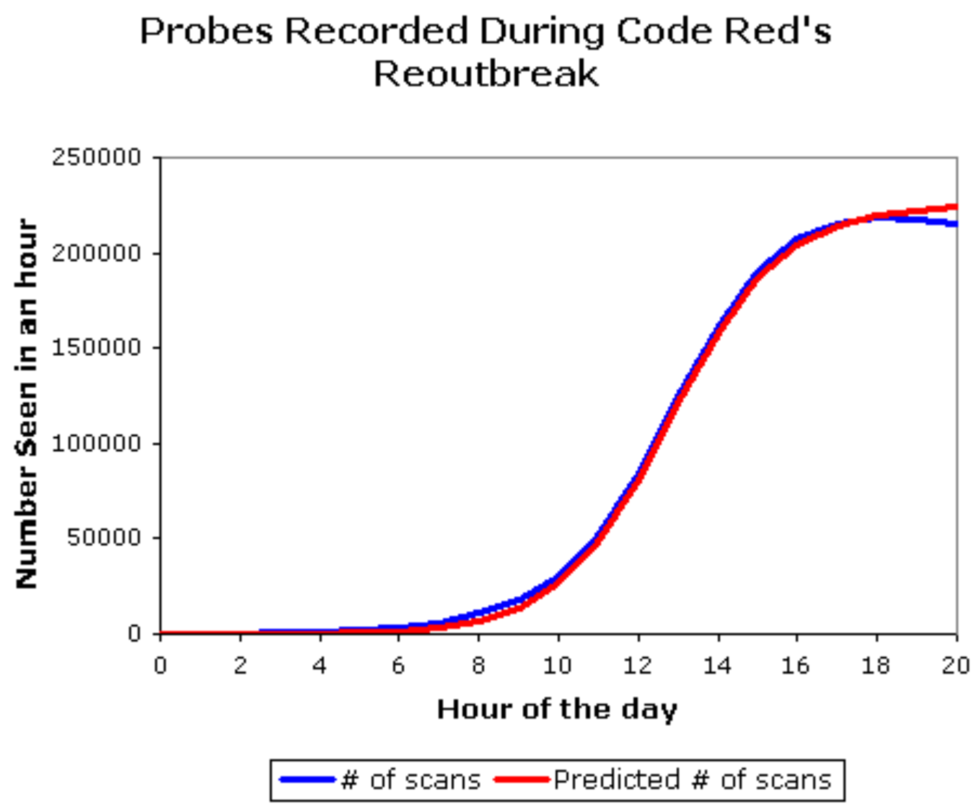


Going Fast: Slammer

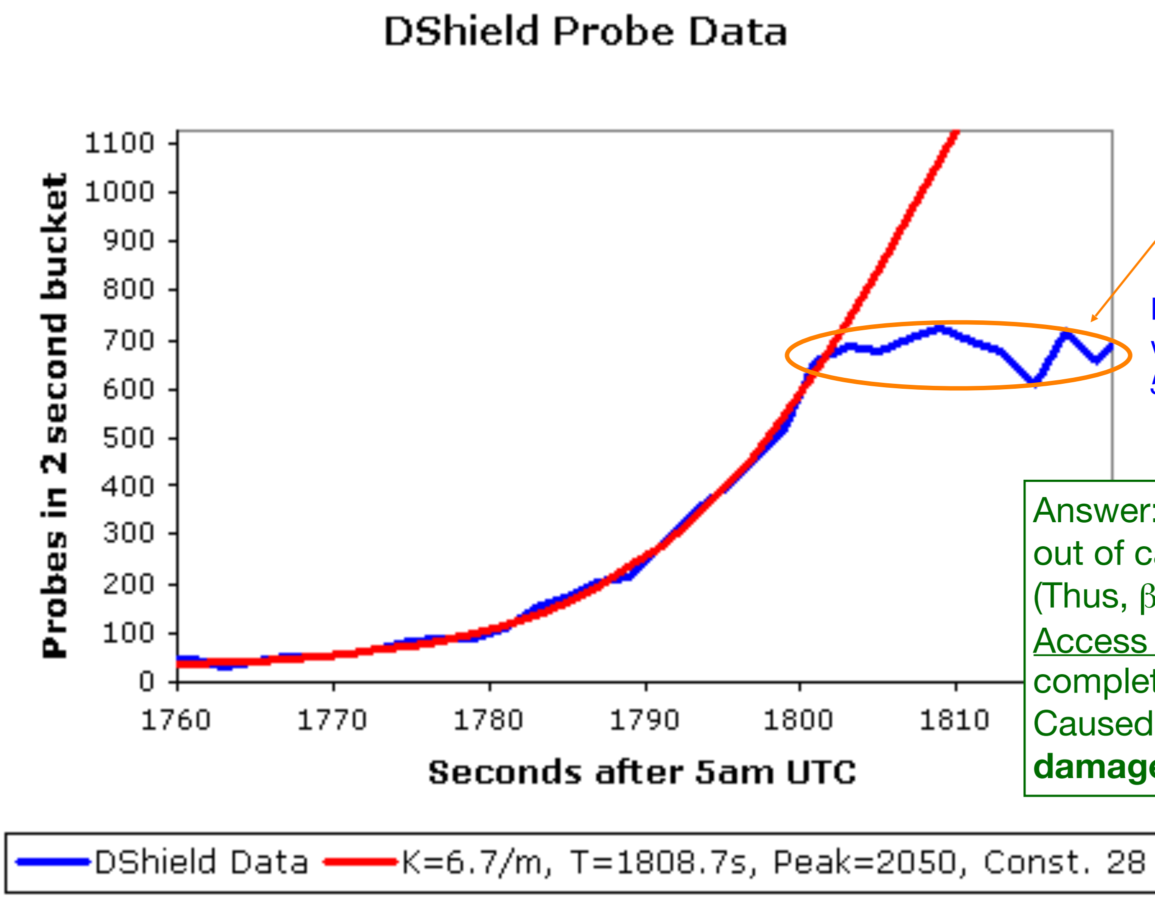
- Slammer exploited connectionless UDP service, rather than connection-oriented TCP
- Entire worm fit in a single packet!
- ⇒ When scanning, worm could "fire and forget"

  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/sec*

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

Witty...

- A worm like Slammer but with a twist...
  - Targeted network intrusion detection sensors!
  - Released ~36 hours after vulnerability disclosure and patch availability!
- Payload wasn't just spreading, however...
  - ```
    while true {
      for i := range(20000){
        send self to random target;
      }
      select random disk (0-7)
      if disk exists {
         select random block, erase it;
      }}
    ```
    ```

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 scanning
- 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

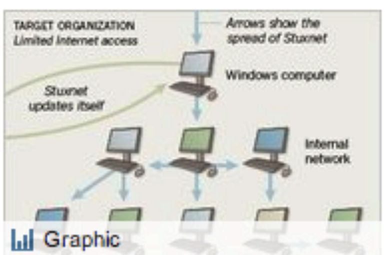
This article is by ***William J. Broad, John Markoff*** *and* ***David E. Sanger.***

Enlarge This Image

Nicholas Roberts for The New York Times

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

Multimedia



Graphic

How Stuxnet Spreads

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

The "Toddler" Attack Payload...

- Stuxnet was very carefully engineered...
  - Designed to only go off under ***very specific*** circumstances
- But industrial control systems are inherently vulnerable
  - They consist of sensors and actuators
  - And safety is a ***global*** property
- Generic Boom:
  - At zero hour, the payload sees that it is on control system:
    map the sensors and actuators, see which ones are low speed vs high speed
  - T+30 minutes: Start replaying sensor data, switch actuators in low-speed system
  - T+60 minutes: Switch all actuators at high speed...
- This ***has been done***:
  A presumably Russian test attack on the Ukranian power grid! ("CrashOverride" attack)

And NotPetya...

- NotPetya was a worm deliberately launched by Russia against Ukraine
  - Initial spread: A corrupted update to MeDoc Ukranian Tax Software
  - Then spread within an institution using "Eternal Blue" (Windows vulnerability) and "Mimikatz"
    - Mimikatz is way ***way more*** powerful:
Takes advantage of windows transitive authorization...
    - IF you are running on the admin's machine, you can take over the domain controller
    - IF you are running on the domain controller, you can take over ***every computer***!!!
- Then wiped machines as fake ransomware
  - Give a veneer of deniability...
  - Shut down Mersk and many other global companies!

And Overall Taxonomy of Spread

- Scanning
  - Look for targets
  - Can be bandwidth limited
- "Target Lists"
  - Pregenerated (Hitlist)
  - On-the-host (Topological)
  - Query a third party server that lists servers (Metaserver)
- Passive
  - Wait for a contact: Infect with the counter-response
- More detailed taxonomy here:
  - http://www.icir.org/vern/papers/taxonomy.pdf