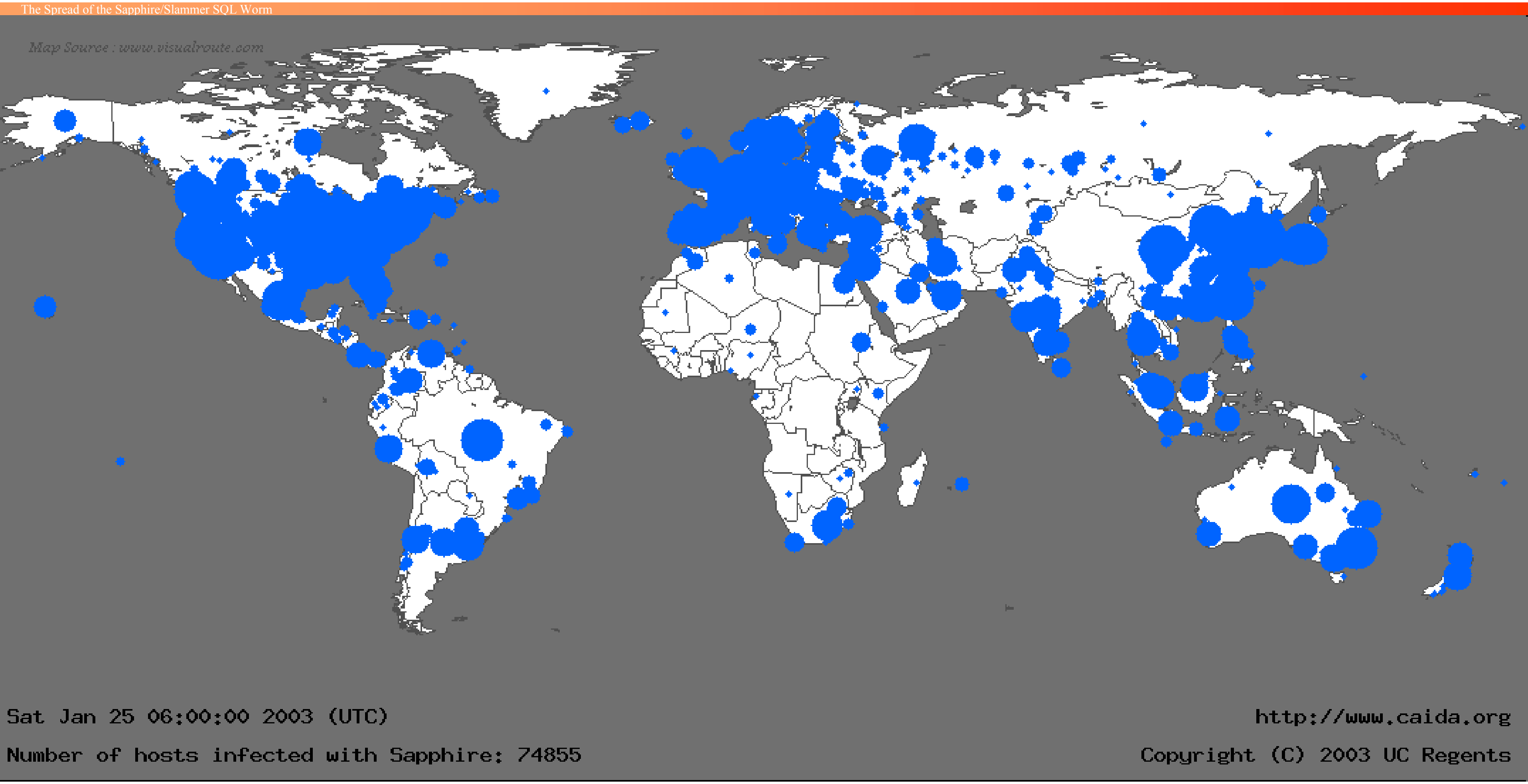
The Spread of the Sapphire/Slammer SQL Worm

David Moore	Vern Paxson	Stefan Savage	Colleen Shannon	Stuart Staniford	Nicholas Weaver
CAIDA & UCSD CSE	ICIR & LBNL	UCSD CSE	CAIDA	Silicon Defense	Silicon Defense & UCB EECS

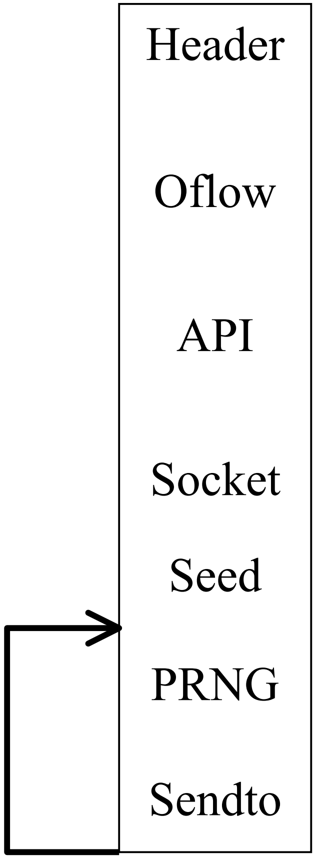
http://www.caida.org/analysis/security/sapphire/
http://www.silicondefense.com/sapphire/
http://www.cs.berkeley.edu/~nweaver/sapphire/

The Spread of Sapphire



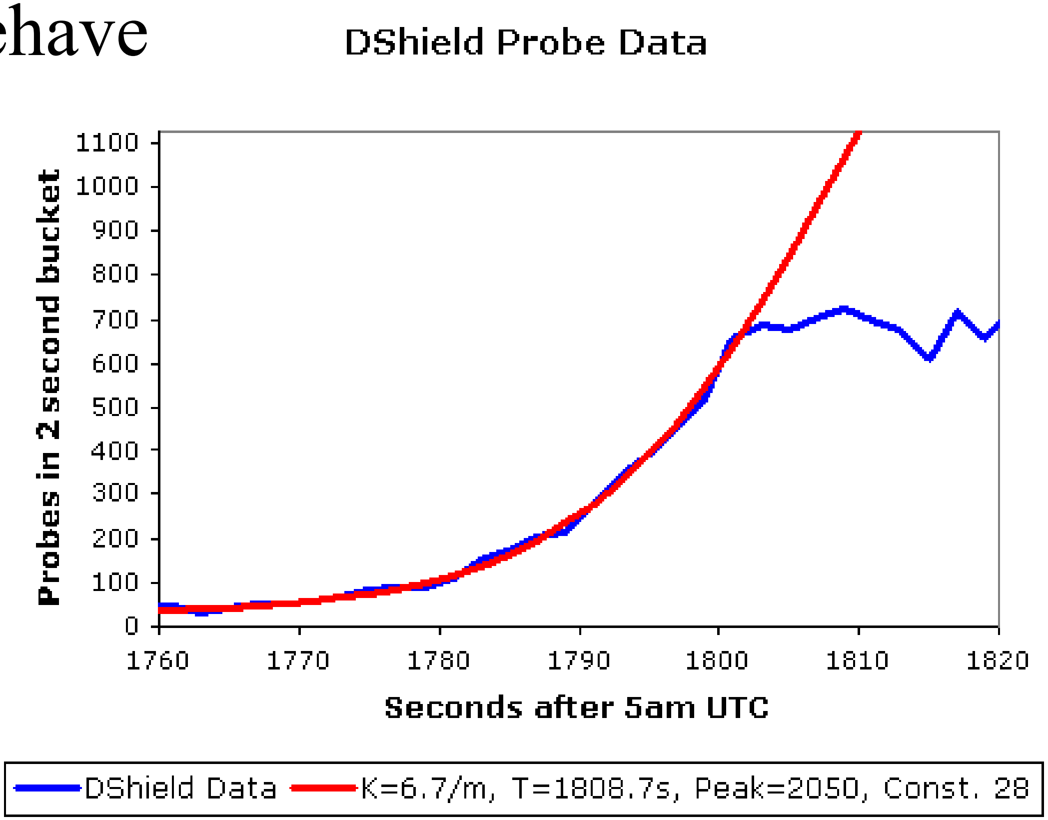
What Was Sapphire

- Sapphire was a single packet UDP worm
  - Cleanup from buffer overflow
  - Get API pointers
    - Code borrowed from published exploit
  - Create socket & packet
  - Seed PRNG with `getTickCount()`
  - While 1
    - Increment PRNG
      - Mildly buggy
    - Send packet to PRNG address
- 404 bytes total
- Worldwide Spread in 10 minutes



Sapphire is a Scanning Worm

- First ~40 seconds behave like classic scanning worm
  - Doubling time of ~8.5 seconds
- Matches Random -Constant-Spread (RCS) model
  - No sign of hitlisting or other acceleration

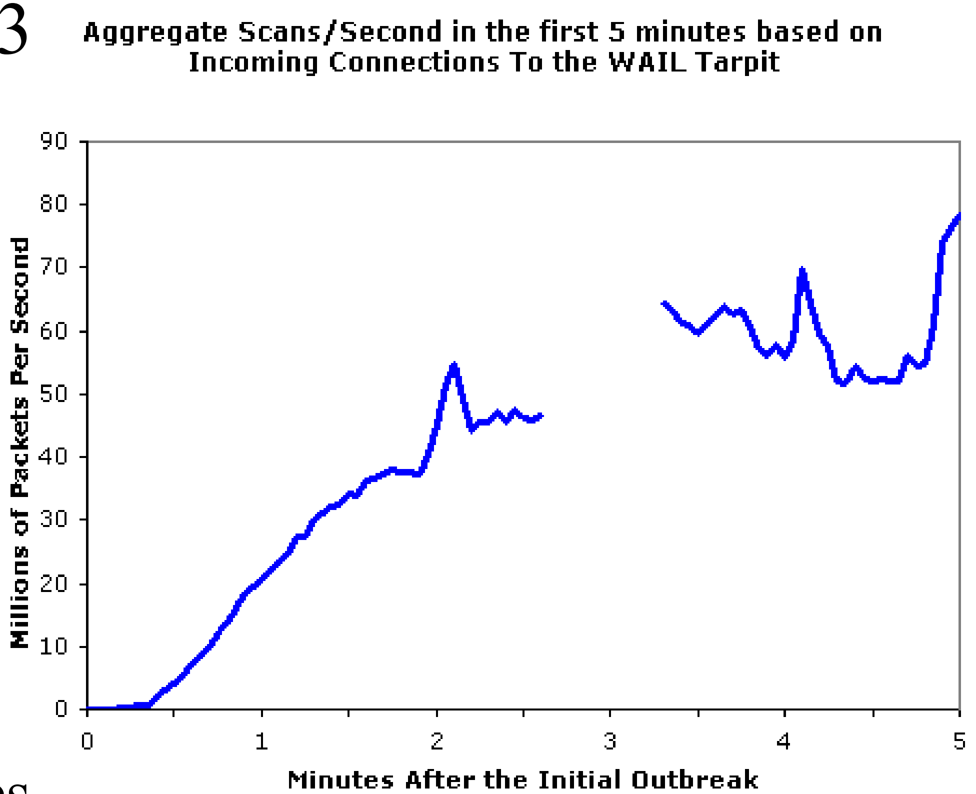


Why Was Sapphire Fast: A Bandwidth-Limited Scanner

- Code Red's scanner is latency-limited
  - In many threads: send SYN to random address, wait for response or timeout
  - Code Red → ~6 scans/second,
    - population doubles about every 40 minutes
- Every Sapphire copy sent infectious packets at maximum rate
  - 1 Mb upload bandwidth → 280 scans/second
  - 100 Mb upload bandwidth → 28,000 scans/second

How Fast:

- Full scanning rate in ~3 minutes
  - >55 Million IPs/s
- Scanning rate scans the net in less than 10 minutes
- Local saturations occur in <1 minute
  - When Sapphire deviates from RCS model

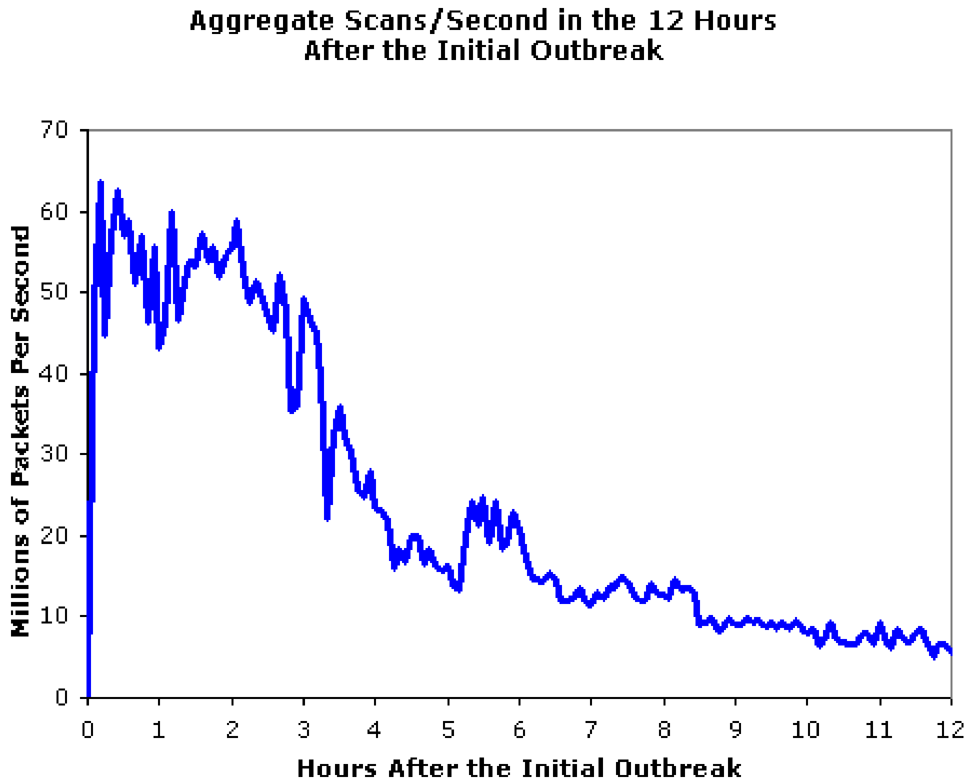


Is This Speed an Isolated Case?

- Any single packet UDP worm, unless deliberately limited or broken, will scan like Sapphire
- Any reasonably small TCP worm can scan like Sapphire
  - Needs to construct SYNs at line rate, receive ACKs in a separate thread
- Three Rhetorical Questions
  - How to construct a bandwidth-limited TCP scanner?
  - How to respond to upstream congestion when transmitting infection attempt and worm body?
  - What happens when there is public sample code?

What Worked: The Backbones and the Net

- Backbones generally kept up
  - Several windows kept seeing packets at full rate
- Substantial response began in 2-3 hours
  - On a Friday night in the US
  - Response only stopped side-effects, not Sapphire



What Didn’t: Firewalls and Firewall Policies

- Sapphire showed a minor aversion to scanning the local net
  - Many copies will never scan the local network due to PRNG bugs
    - Unlike Code Red II and Nimda who’s scanning was designed to exploit firewalls
  - Those that do will likely require several minutes
- How to estimate firewall coverage:
  - Machines infected in the first few minutes:
    - Through the firewall
  - Machines infected later:
    - Internal infections/Infections from allowed addresses

What Didn't:
Local Net, Switches, Routers

- Some edge devices failed due to load
  - Temporary disruptions in our dataset
  - Several UCB switches needed resetting after infected machines were removed
- Many sites connectivity disrupted by outgoing traffic
  - Often with only a few infected machines
  - Need to deploy fairness/bandwidth capping
- Some critical systems are not well isolated from the Internet
  - Bellevue WA 911 system, BofA ATM system

Thoughts on the Future

- Nastier worms will happen
- Smaller populations are now vulnerable
  - ~20,000 machines can support viable fast scanning worms
- We need more/wider network telescopes
  - We would like a distributed /8
    - Composed of many smaller windows
- Automatic defenses are essential
  - Future worm can be exceedingly fast
    - Speed is not limited to single UDP packet worms
  - We had 30 seconds to 1 minute to do something

Conclusions:

- Sapphire was the first fast worm
  - ~10 minutes to spread worldwide
  - Completely outpaces human response
- Sapphire revealed weaknesses in our infrastructure
  - Local networks and connections susceptible to internal DoS
  - Too many permissive firewalls
- Some good things
  - The Internet survives
    - Mitigation happens very quickly

Backup Slide: When Was Machine X Infected?

- We can only see when a machine probes our address ranges, not when it is infected
- Many worms will never probe our telescopes due to PRNG errors
  - So we will never know these addresses
- Small Network Telescopes:
  - Scanning rate per worm drops quickly as the outbound links are saturated
  - We only have a few /16s worth of addresses and smaller telescopes are far less sensitive to discerning individual events

Backup Slide: Implications of the PRNG

- Seeded with `getTickCount()`, 3 bugs
- Worst behavior in the lower significant bits
  - Due to enddianness, buggyness occurs in the upper octets of the address
    - **`0xAB.CD.EF.01`** → **`01.EF.CD.AB`**
  - Lots of short cycles
    - Probability of choosing a cycle is proportional to the cycle size
- Any given worm will only scan a subset of the net
  - Based on the initial random seed
  - Many worms will not probe our monitors

Backup Slide: The PRNG and Our Estimates

- PRNG consists of many cycles
  - Probability of being in a particular cycle is proportional to the cycle size, so actual scans are well distributed
- Skews our estimate somewhat
  - Can estimate the total scanning rate
  - Can estimate the fraction of the internet infected
    - Based on scanning rate and coverage of the Internet
  - Can't estimate the total infection or when machines are infected
    - Many copies will not probe our telescopes
    - Those that can may not probe them immediately

Backup Slide: How Many Bugs in the PRNG?

- Intended PRNG, **`mod`** 2^{32}
  - `addr' = (addr*214013 + 2531011)`
- Actual PRNG, **`mod`** 2^{32}
  - `addr' = (addr *214013 + (-2531012` **`xor`** `EBX))`
- Bugs (did not limit spread much):
  - used **`OR`** instead of **`XOR`** to clear the value of `EBX`
    - EBX values include `0x77f8313c`, `0x77e89b18`, `0x77ea094c`
  - Forgot to "add 1" in the 2s complement negation
  - Used **`ADD`** instead of **`SUB`**
- Good:
  - Well seeded using `getTickCount()`