How Chicken Little sees the Internet...

Why Chicken Little is a naïve optimist

- Imagine the following species:
  - Poor genetic diversity; heavily inbred
  - Lives in “hot zone”; thriving ecosystem of infectious pathogens
  - Instantaneous transmission of disease
  - Immune response 10-1M times slower
  - Poor hygiene practices
  
  **What would its long-term prognosis be?**

  - What if diseases were designed…
    - Trivial to create a new disease
    - Highly profitable to do so

Threat transformation

- **Traditional threats**
  - Attacker manually targets high-value system/resource
  - Defender increases cost to compromise high-value systems
  - Biggest threat: insider attacker

- **Modern threats**
  - Attacker uses automation to target all systems at once (can filter later)
  - Defender must defend all systems at once
  - Biggest threats: software vulnerabilities & naïve users

Large-scale technical enablers

- **Unrestricted connectivity**
  - Large-scale adoption of IP model for networks & apps

- **Software homogeneity & user naïveté**
  - Single bug = mass vulnerability in millions of hosts
  - Trusting users (“ok”) = mass vulnerability in millions of hosts

- Few meaningful defenses
- Effective anonymity (minimal risk)
Driving Economic Forces

- No longer just for fun, but for profit
  - SPAM forwarding (MyDoom.A, backdoor, SoBig), Credit Card theft (Korgo), DDoS extortion, etc...
  - Symbiotic relationship: worms, bots, SPAM, etc
  - Fluid third-party exchange market (millions of hosts for sale)
    - Going rate for SPAM proxying 3-10 cents/host/week
    - Seems small, but 25k botnet gets you $40k-130k/yr
    - Generalized search capabilities are next

- “Virtuous” economic cycle
  - The bad guys have large incentive to get better

Today’s focus: Outbreaks

- Outbreaks?
  - Acute epidemics of infectious malcode designed to actively spread from host to host over the network
  - E.g. Worms, viruses (for me: pedantic distinctions)

- Why epidemics?
  - Epidemic spreading is the fastest method for large-scale network compromise

- Why fast?
  - Slow infections allow much more time for detection, analysis, etc (traditional methods may cope)


- First ~1min behaves like classic random scanning worm
  - Doubling time of ~8.5 seconds
  - CodeRed doubled every 40mins

- >1min worm starts to saturate access bandwidth
  - Some hosts issue >20,000 scans per second
  - Self-interfering (no congestion control)

- Peaks at ~3min
  - >55million IP scans/sec

- 90% of Internet scanned in <10mins
  - Infected ~100k hosts (conservative)

Was Slammer really fast?

- Yes, it was orders of magnitude faster than CR
- No, it was poorly written and unsophisticated

- Who cares? It is literally an academic point
  - The current debate is whether one can get < 500ms
- Bottom line: way faster than people!
How to think about worms

- Reasonably well described as infectious epidemics
  - Simplest model: Homogeneous random contacts
- Classic SI model
  - $N$: population size
  - $S(t)$: susceptible hosts at time $t$
  - $I(t)$: infected hosts at time $t$
  - $\beta$: contact rate
  - $i(t)$: $I(t)/N$, $s(t)$: $S(t)/N$
  - $\frac{di}{dt} = \beta \frac{IS}{N}$
  - $\frac{dS}{dt} = -\beta \frac{IS}{N}$
  - $i(t) = \frac{e^{\beta(\tau-t)}}{1 + e^{\beta(\tau-t)}}$

What’s important?

- There are lots of improvements to the model…
  - Chen et al, *Modeling the Spread of Active Worms*, Infocom 2003 (discrete time)
  - ... but the bottom line is the same. We care about two things:
    - How likely is it that a given infection attempt is successful?
      - Target selection (random, biased, hitlist, topological, ...)
      - Vulnerability distribution (e.g. density – $S(0)/N$)
    - How frequently are infections attempted?
      - $\beta$: Contact rate

What can be done?

- Reduce the number of susceptible hosts
  - *Prevention*, reduce $S(t)$ while $I(t)$ is still small
    (ideally reduce $S(0)$)
- Reduce the contact rate
  - *Containment*, reduce $\beta$ while $I(t)$ is still small

Prevention: Software Quality

- **Goal**: eliminate vulnerability
  - Static/dynamic testing (e.g. Cowan, Wagner, Engler, etc)
  - Software process, code review, etc.
  - Active research community
  - Taken seriously in industry
    - Security code review *alone* for Windows Server 2003 ~ $200M
  - Traditional problems: soundness, completeness, usability
  - Practical problems: scale and cost
Prevention: Hygiene Enforcement

- **Goal**: keep susceptible hosts off network

- Only let hosts connect to network if they are “well cared for”
  - Recently patched, up-to-date anti-virus, etc…
  - Automated version of what they do by hand at NSF

- Cisco Network Admission Control (NAC)

Containment

- **Reduce contact rate**

- **Slow down**
  - Throttle connection rate to slow spread
    - Twycross & Williamson, *Implementing and Testing a Virus Throttle*, USENIX Sec ’03
  - Important capability, but worm still spreads…

- **Quarantine**
  - Detect and block worm

Defense requirements

- We can define reactive defenses in terms of:
  - **Reaction time** – how long to detect, propagate information, and activate response
  - **Containment strategy** – how malicious behavior is identified and stopped
  - **Deployment scenario** - who participates in the system

- Given these, what are the engineering requirements for any effective defense?

Defense requirements summary

- **Reaction time**
  - Required reaction times are a couple minutes or less for CR-style worms (seconds for worms like Slammer)

- **Containment strategy**
  - Content filtering is far more effective than address blacklisting for a given reaction speed

- **Deployment scenarios**
  - Need nearly all customer networks to provide containment
  - Need at least top 40 ISPs provide containment; top 100 ideal

- Is this possible? Let’s see…
Outbreak Detection/Monitoring

- Two classes of detection
  - **Scan detection**: detect that host is infected by infection attempts
  - **Signature inference**: automatically identify content signature for exploit (sharable)

- Two classes of monitors
  - Ex-situ: “canary in the coal mine”
    - Network Telescopes
    - HoneyNets/Honeypots
  - In-situ: real activity as it happens

Telescopes + Active Responders

- Problem: Telescopes are passive, can’t respond to TCP handshake
  - Is a SYN from a host infected by CodeRed or Welchia? Dunno.
  - What does the worm payload look like? Dunno.

- Solution: proxy responder
  - Stateless: TCP SYNACK (Internet Motion Sensor), per-protocol responders (iSink)
  - Stateful: Honeyd
  - Can differentiate and fingerprint payload
    - False positives generally low since no regular traffic

Network Telescopes

- Infected host scans for other vulnerable hosts by randomly generating IP addresses
- Network Telescope: monitor large range of unused IP addresses – will receive scans from infected host
- Very scalable. UCSD monitors 17M+ addresses

HoneyNets

- Problem: don’t know what worm/virus would do? No code ever executes after all.
- Solution: redirect scans to real “infectable” hosts (honeypots)
  - Individual hosts or VM-based: Collapsar, HoneyStat, Symantec
  - Can reduce false positives/negatives with host-analysis (e.g. TaintCheck, Vigilante, Minos) and behavioral/procedural signatures

- Challenges
  - Scalability
  - Liability (honeywall)
  - Isolation (2000 IP addr -> 40 physical machines)
  - Detection (VMWare detection code in the wild)
Overall limitations of telescope, honeynet, etc monitoring

- **Depends** on worms scanning it
  - What if they don’t scan that range (smart bias)
  - What if they propagate via e-mail, IM?
- Inherent tradeoff between liability exposure and detectability
  - Honeypot detection software exists
- It doesn’t necessary reflect what’s happening on your network (can’t count on it for local protection)

- Hence, we’re always interested in native detection as well

Scan Detection

- Idea: detect worm’s infection attempts
  - In the small: ZoneAlarm, but how to do in the network?
- Indirect scan detection
  - Whyte et al, *DNS-based Detection of Scanning Worms in an Enterprise Network*, NDSS ’05
- Direct scan detection
  - Weaver et al, *Very Fast Containment of Scanning Worms*, USENIX Sec ’04
    - Threshold Random Walk – bias source based on connection success rate (Jung et al); use approximate state for fast hardware implementation
    - Can support multi-Gigabit implementation, detect scan within 10 attempts
    - Few false positives: Gnutella (finding accessing), Windows File Sharing (benign scanning)

Signature inference

- Challenge: need to automatically **learn** a content “signature” for each new worm – potentially in less than a second!

- **Singh et al**, *Automated Worm Fingerprinting*, OSDI ’04
- **Kim et al**, *Autograph: Toward Automated, Distributed Worm Signature Detection*, USENIX Sec ‘04

Approach

- Monitor network and look for strings common to traffic with worm-like behavior
- Signatures can then be used for content filtering
Content sifting

- Assume there exists some (relatively) unique invariant bitstring W across all instances of a particular worm (true today, not tomorrow...)
- Two consequences:
  - **Content Prevalence:** W will be more common in traffic than other bitstrings of the same length
  - **Address Dispersion:** the set of packets containing W will address a disproportionate number of distinct sources and destinations
- Content sifting: find W’s with high content prevalence and high address dispersion and drop that traffic

The basic algorithm

Detector in network
Prevalence Table

1

Address Dispersion Table
Sources
Destinations

1 (A)
1 (B)
1 (C)

1

Detector in network
Prevalence Table

1

Address Dispersion Table
Sources
Destinations

1 (A)
1 (B)
1 (C)

1
The basic algorithm

Challenges

- **Computation**
  - To support a 1Gbps line rate we have 12us to process each packet
    - Dominated by memory references; state expensive
    - Content sifting requires looking at *every* byte in a packet

- **State**
  - On a fully-loaded 1Gbps link a naïve implementation can easily consume 100MB/sec for tables

Kim et al’s solution: Autograph

- Pre-filter flows for those that exhibit scanning behavior (i.e. low TCP connection ratio)
  - HUGE reduction in input, fewer prevalent substrings
  - Don’t need to track dispersion at all
  - Fewer possibilities of false positives

- However, only works with TCP scanning worms
  - Not UDP (Slammer), e-mail viruses (MyDoom), IM-based worms (Bizex), P2P (Benjamin)

Which substrings to index?

- **Approach 1: Index all substrings**
  - Way too many substrings → too much computation → too much state

- **Approach 2: Index whole packet**
  - Very fast but trivially evadable (e.g., Witty, Email Viruses)

- **Approach 3: Index all contiguous substrings of a fixed length ‘S’**
  - Can capture all signatures of length ‘S’ and larger
  - \[ A B C D E F G H I J K \]

How to represent substrings?

- **Store hash** instead of literal to reduce state
- **Incremental hash** to reduce computation
- **Rabin fingerprint** is one such efficient incremental hash function [Rabin81, Manber94]
  - One multiplication, addition and mask per byte

  \[ \begin{array}{cccccccccccc}
  P1 & R & A & N & D & A & B & C & D & O & M \\
  \end{array} \]
  - **Fingerprint = 11000000**

  \[ \begin{array}{cccccccccccc}
  P2 & R & A & B & C & D & A & N & D & O & M \\
  \end{array} \]
  - **Fingerprint = 11000000**

How to subsample?

- **Approach 1: sample packets**
  - If we chose 1 in N, detection will be slowed by N

- **Approach 2: sample at particular byte offsets**
  - Susceptible to simple evasion attacks
  - No guarantee that we will sample same sub-string in every packet

- **Approach 3: sample based on the hash of the substring**

Value sampling [Manber ’94]

- Sample hash if last ‘N’ bits of the hash are equal to the value ‘V’
  - The number of bits ‘N’ can be dynamically set
  - The value ‘V’ can be randomized for resiliency

  \[ \begin{array}{cccccccccccc}
  \text{Fingerprint: 10000000} \end{array} \]

- **\( P_{\text{track}} \) → Probability of selecting at least one substring of length ‘S’ in a ‘L’ byte invariant**
  - For 1/64 sampling (last 6 bits equal to 0), and 40 byte substrings
  - **\( P_{\text{track}} = 99.64\% \) for a 400 byte invariant**
**Content sifting summary**

- Index fixed-length substrings using incremental hashes
- Subsample hashes as function of hash value
- Multi-stage filters to filter out uncommon strings
- Scalable bitmaps to tell if number of distinct addresses per hash crosses threshold

- **Now** its fast enough to implement

---

**Sasser**

---

**Kibvu**

- Slower spread (1.5 packets/minute inbound)
- Consequently, slower detection (42mins to dispersion of 30)
- Response time is wrong metric
False Negatives

- Easy to prove presence, impossible to prove absence

- **Live evaluation**: over 8 months detected every worm outbreak reported on popular security mailing lists

- **Offline evaluation**: several traffic traces run against both Earlybird and Snort IDS (w/all worm-related signatures)
  - Worms not detected by Snort, but detected by Earlybird
  - The converse never true

False Positives

- **Common protocol headers**
  - Mainly HTTP and SMTP headers
  - Distributed (P2P) system protocol headers

- **Procedural whitelist**
  - Small number of popular protocols

- **Non-worm epidemic Activity**
  - SPAM
  - BitTorrent

Summary

- Internet-connected hosts are highly vulnerable to worm outbreaks
  - Millions of hosts can be "taken" before anyone realizes
  - If only 10,000 hosts are targeted, no one may notice

- Prevention is a critical element, but there will always be outbreaks

- Containment requires fully automated response (dp)

- Scaling issues favor network-based defenses

- Different detection strategies, monitoring approaches
  - Very active research community

- Content sifting: automatically sift bad traffic from good