CIS 551 / TCOM 401

Computer and Network Security

Spring 2006

Lecture 21

Outline for Today (and Next Time)

- Containing worms and viruses
- Detecting viruses and worms
- Intrusion detection in general
- Defenses against viruses/worms/general intruders
  - Tools for determining system vulnerability

- Research Paper: "Automated Worm Fingerprinting"
  - Singh, Estan, Varghese, and Savage
  - (may not cover until next time)

Infection rate over time

- Change in infection rate is expressed as:

$$\frac{dI}{dt} = \underbrace{I(t)}_{\text{\# of infected hosts}} * \underbrace{\beta}_{\text{rate of contact}} * \underbrace{s(t)}_{\text{likelihood that contacted hosts is susceptible}}$$

# of infected hosts — rate of contact — likelihood that contacted hosts is susceptible

Rewrite to obtain:

$$\frac{di}{dt} = \beta * i(t) * (1-i(t))$$

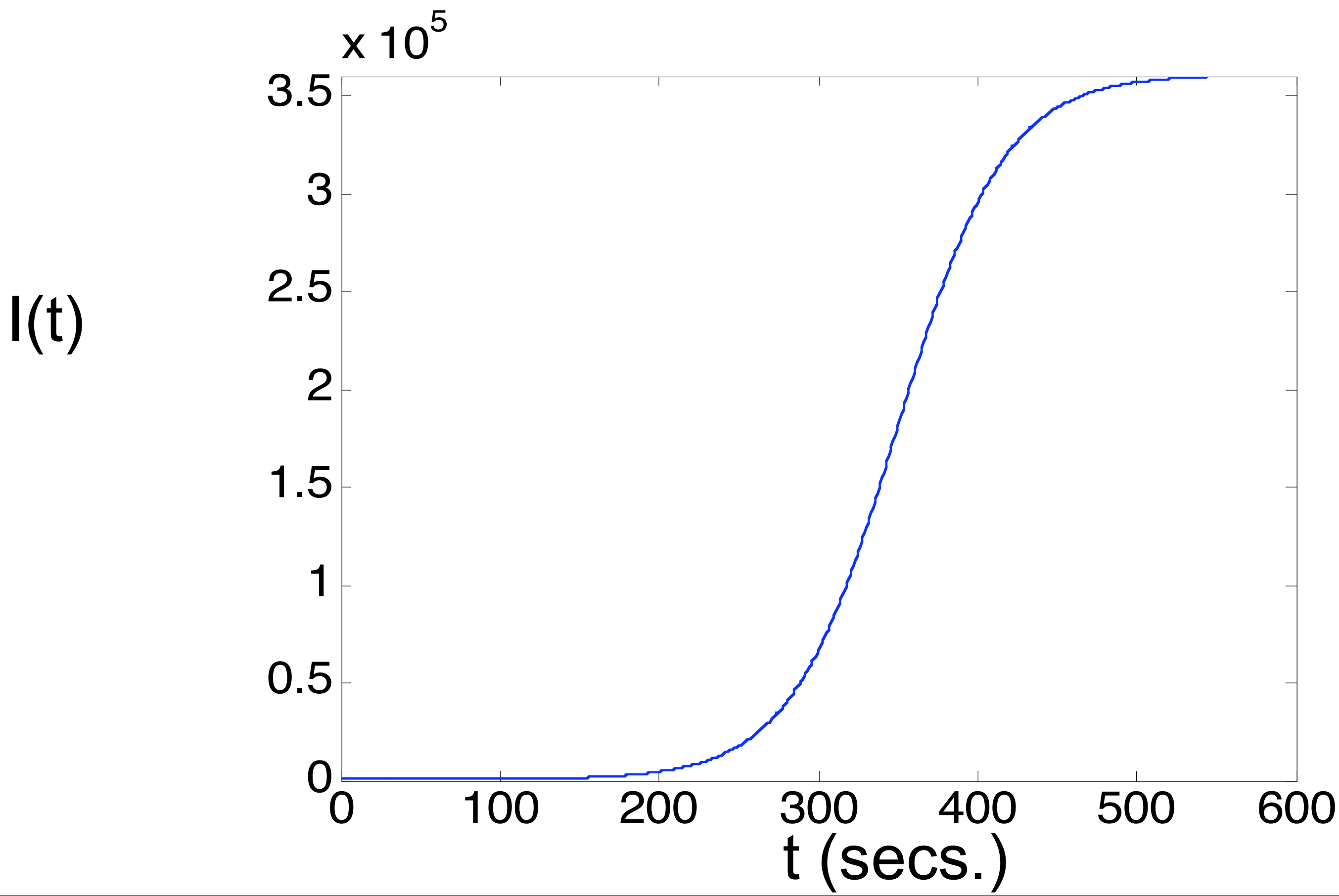
Integrate to get this closed form:

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

T = integration constant

Exponential growth, tapers off

- Example curve of I(t)   (which is i(t) * N)
- Here, N = 3.5×10^5 (β affects steepness of slope)



What can be done?

- Reduce the number of infected hosts
  - **Treatment**, reduce I(t) while I(t) is still small
  - e.g. shut down/repair infected hosts
- Reduce the contact rate
  - **Containment**, reduce ß while I(t) is still small
  - e.g. filter traffic

}Reactive

- Reduce the number of susceptible hosts
  - **Prevention**, reduce S(0)
  - e.g. use type-safe languages

}Proactive

Containment

- Reduce contact rate β

- **Oblivious defense**
  - Consume limited worm resources
  - Throttle traffic to slow spread
  - Possibly important capability, but worm still spreads…

- **Targeted defense**
  - Detect and block worm

Design Space

- Design Issues for Reactive Defense
  [Moore et al 03]

- Any reactive defense is defined by:
  - ***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

- Savage et al. evaluate the requirements for these parameters to build **any** effective system for worm propagation.

Methodology

- **Moore et al., "Internet Quarantine:…" paper**

- **Simulate spread of worm across Internet topology:**
  – infected hosts *attempt* to spread at a fixed rate (probes/sec)
  – target selection is uniformly random over IPv4 space

- **Simulation of defense:**
  – system detects infection within reaction time
  – subset of network nodes employ a containment strategy

- **Evaluation metric:**
  – % of vulnerable hosts infected in 24 hours
  – 100 runs of each set of parameters (95^{th} percentile taken)
    - Systems must plan for reasonable situations, **not** the average case

- **Source data:**
  – vulnerable hosts: 359,000 IP addresses of CodeRed v2 *victims*
  – Internet topology: AS routing topology derived from RouteViews

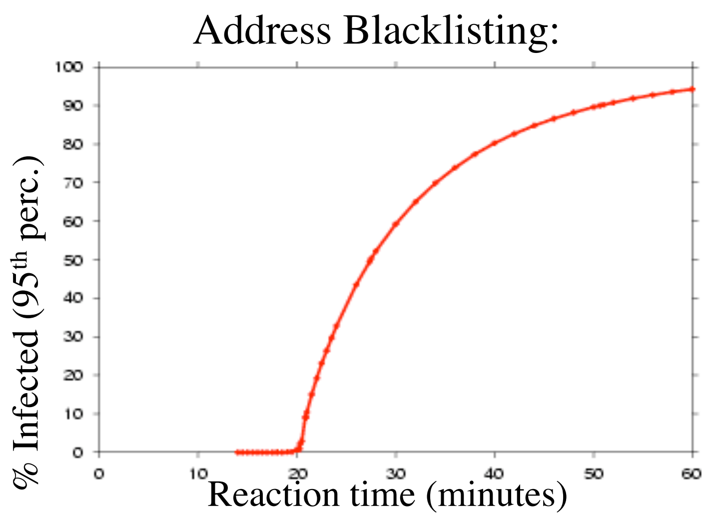
Initial Approach: Universal Deployment

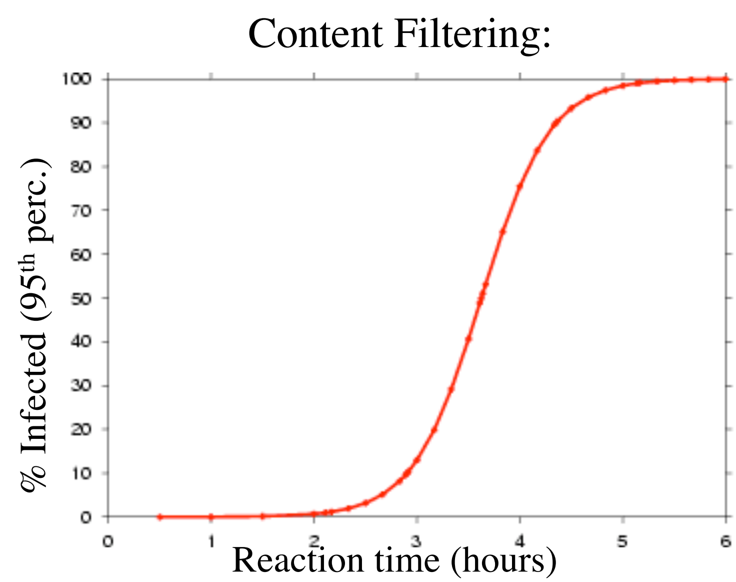
- Assume **every host** employs the containment strategy

- Two containment strategies they tested:
  - ***Address blacklisting:***
    - block traffic from malicious source IP addresses
    - reaction time is relative to each infected host
  - ***Content filtering:***
    - block traffic based on signature of content
    - reaction time is from first infection

- How quickly does each strategy need to react?
- How sensitive is reaction time to worm probe rate?

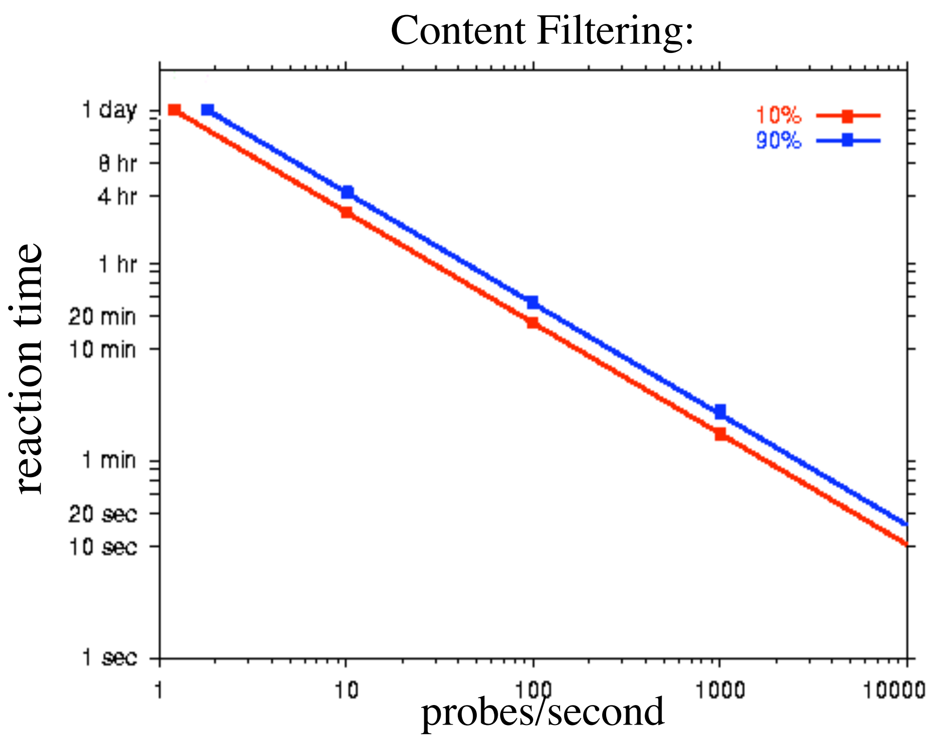
Reaction times?





- To contain worms to 10% of vulnerable hosts after 24 hours of spreading at 10 probes/sec (CodeRed):
  - Address blacklisting: reaction time must be < 25 minutes.
  - Content filtering: reaction time must be < 3 hours

Probe rate vs. Reaction Time



- Reaction times must be fast when probe rates get high:
  - 10 probes/sec: reaction time must be < 3 hours
  - 1000 probes/sec: reaction time must be < 2 minutes

Limited Network Deployment

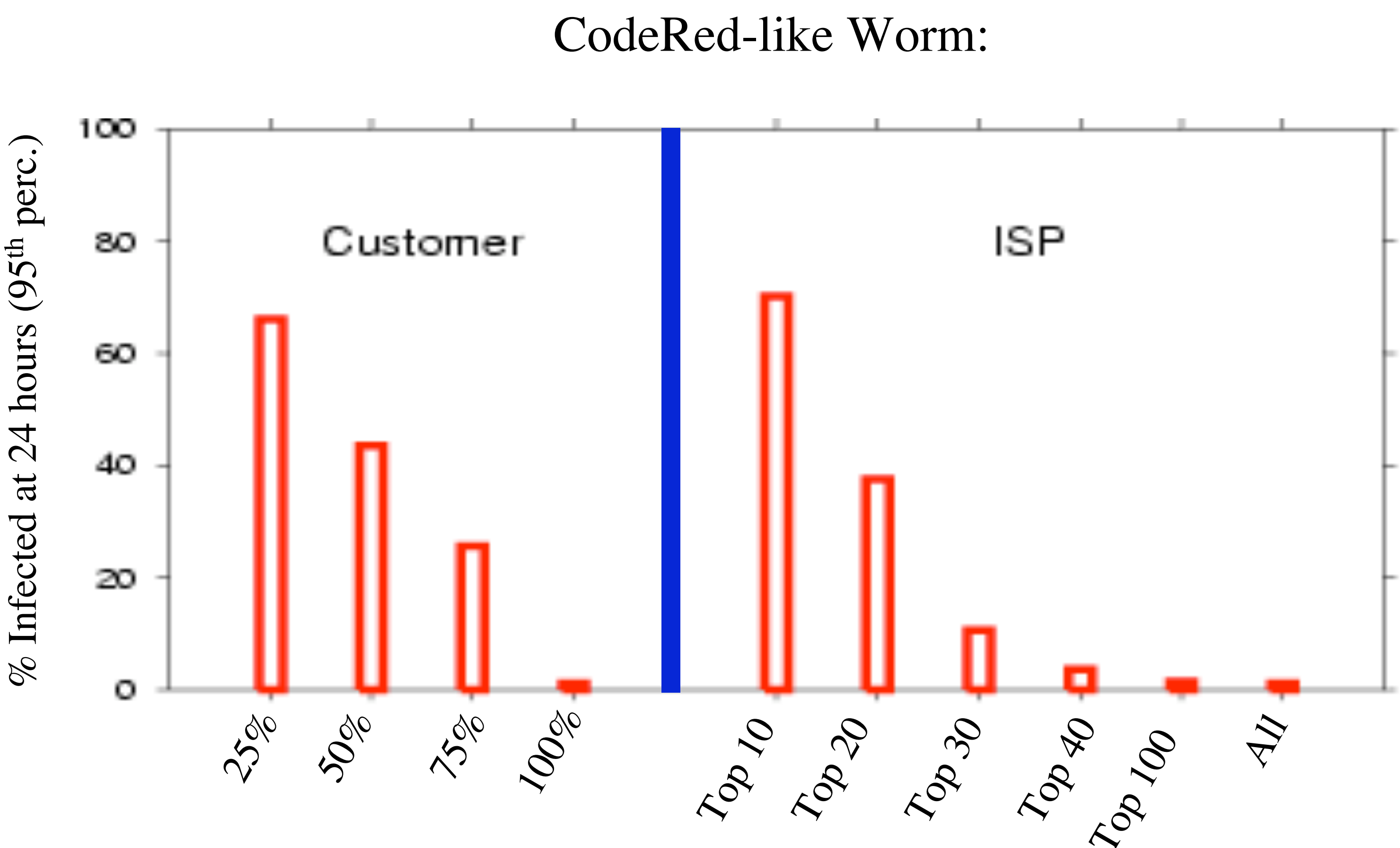
- Depending on <u>every</u> **host** to implement containment is not feasible:
  - installation and administration costs
  - system communication overhead

- A more realistic scenario is <u>limited</u> deployment in the **network**:
  - Customer Network: firewall-like inbound filtering of traffic
  - ISP Network: traffic through border routers of large transit ISPs

- How effective are the deployment scenarios?
- How sensitive is reaction time to worm probe rate under limited network deployment?

Deployment Scenario Effectiveness?

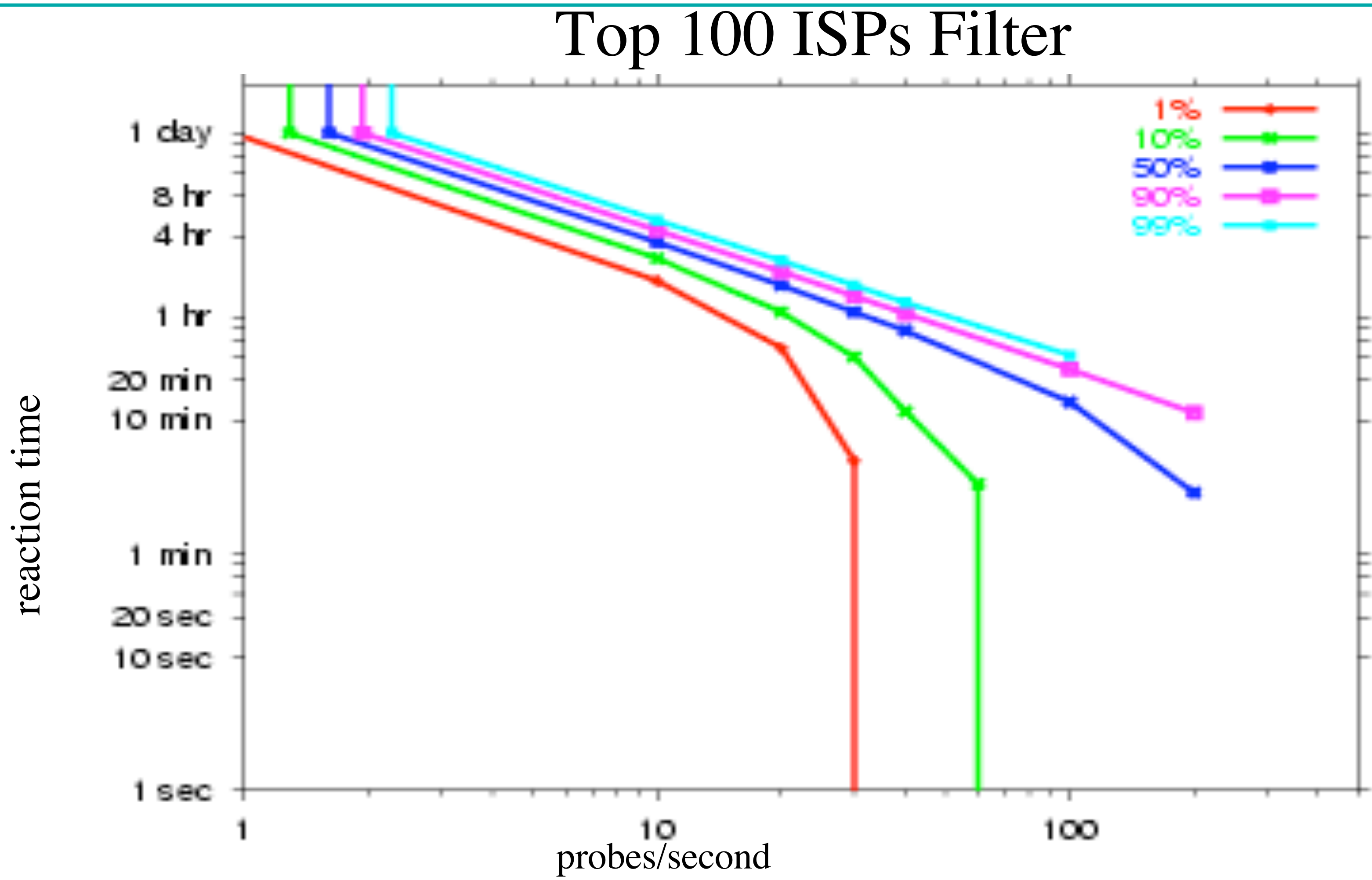
Reaction time = 2 hours



Content filtering firewalls at edge of customer nets.

Content filtering at exchange points in major ISPs.

Reaction Time vs. Probe Rate (II)



- Above 60 probes/sec, containment to 10% hosts within 24 hours is impossible even with *instantaneous* reaction.

Summary: Reactive Defense

- Reaction time:
  - required reaction times are a couple minutes or less (far less for bandwidth-limited scanners)
- Containment strategy:
  - content filtering is more effective than address blacklisting
- Deployment scenarios:
  - need nearly all customer networks to provide containment
  - need at least top 40 ISPs provide containment

Virus & Worm Signatures

- Viruses & worms can't be completely invisible:
  - Code must be stored somewhere
  - They must do something (e.g. propagate) when they run
- Fragments of the virus/worm code itself
  - Strings "kindly check the attached LOVELETTER"
- Effects on the computing environment
  - Changes to the Windows registry
- Propagation Behavior
  - Copying/modifying system files.
- (More generally, any attack will have some observable effect…)

Example Signatures

- The "Code Red" worm can be identified on victim machines by the presence of the following string in IIS log files:

/default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNN%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

- Slammer Worm can be identified by packets of size > 100 bytes sent to port 1434.

Detecting Viruses & Worms

- How do you even know there's a problem?
  - System administrators/users notice unusual behavior
  - Automatic intrusion/anomaly detection
  - Internet "telescopes"
  - Lure in an attack: honeypots / honeynets

- Techniques:
  - Integrity checks
  - Heuristic detection
  - Signature checking
  - System auditing

Virus Scanners

- Search the system for virus signatures
  - Main memory
  - All files in file system
  - Should also check boot sector
- Where to scan?
  - At each host (e.g. Norton Antivirus)
  - At the firewall
  - At the mail server
- When to scan?
  - On access (when a program is run)
  - On demand (at user’s request, or scheduled)
  - When e-mail is received?
  - Before web content is displayed?
- How to scan?
  - Potentially large database of signatures
  - Need to match against all software on the system => Use Merkle Hash trees

Virus/Worm Scanning

- Pros
  - Effectively detects *known* viruses/worms before they can cause harm
  - Few false alarms
- Cons
  - Can detect only viruses/worms with known signatures
  - Performance penalty (due to scanning)
  - Signature set must be kept up to date
  - Virus/worm writers can easily change signatures
- ==> Generate signatures automatically
  - Automated Worm Fingerprinting (more in a bit)

Software Integrity Checks

- Compute a hash or checksum of executable files
  - Merkle Hash trees
  - Assumes the software to be virus free!
  - Store the hash information for later verification
- Verify new hash vs. saved one during scan
  - Also used for ensuring that software is not corrupted/modified when shipped over the network.

- Pros:
  - Can detect corruption of executables too
  - Reliable
  - Doesn’t require virus signatures

- Cons:
  - False positives (i.e. recompilation)
  - Can’t use it on documents (they change too often)
  - Not supported by most vendors

Heuristic Detection

- Collection of ad hoc rules that identifies virus behavior or virus-like programs
  - Modification of system executables
  - Modification of “template documents” like normal.doc
  - Self-modifying and self-referential code
  - Atypical or abnormal behavior
- Pros
  - Perhaps able to detect unknown viruses/worms
  - Can build tools to look for these features
- Cons
  - Heuristics are expensive and hard to develop.
  - Too may false positives?

Detecting Attacks

- Attacks (against computer systems) usually consist of several stages:
  - Finding software vulnerabilities
  - Exploiting them
  - Hiding/cleaning up the exploit

- Attackers care about finding vulnerabilities:
  - What machines are available?
  - What OS / version / patch level are the machines running?
  - What additional software is running?
  - What is the network topology?

- Attackers care about not getting caught:
  - How detectible will the attack be?
  - How can the attacker cover her tracks?

- Programs can automate the process of finding/exploiting vulnerabilities.
  - Same tools that sys. admins. use to audit their systems…
  - A worm is just an automatic vulnerability finder/exploiter…

Attacker Reconnaissance

- Network Scanning
  - Existence of machines at IP addresses
  - Attempt to determine network topology
  - ping, tracert
- Port scanners
  - Try to detect what processes are running on which ports, which ports are open to connections.
  - Typical machine on the internet gets 10-20 port scans per day!
  - Can be used to find hit lists for flash worms
- Web services
  - Use a browser to search for CGI scripts, Javascript, etc.

Determining OS information

- Gives a lot of information that can help an attacker carry out exploits
  - Exact version of OS code can be correlated with vulnerability databases
- Sadly, often simple to obtain this information:
  - Just try telnet

```
playground~> telnet hpux.u-aizu.ac.jp
Trying 163.143.103.12 ...
Connected to hpux.u-aizu.ac.jp.
Escape character is '^]'.
HP-UX hpux B.10.01 A 9000/715 (ttyp2)

login:
```

Determining OS

- Or ftp:

```
$ ftp ftp.netscape.com 21
Connected to ftp.gftp.netscape.com.
220-36
220 ftpnscp.newaol.com FTP server (SunOS 5.8) ready.
Name (ftp.netscape.com:stevez):
331 Password required for stevez.
Password:
530 Login incorrect.
ftp: Login failed.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> system
215 UNIX Type: L8 Version: SUNOS
ftp>
```

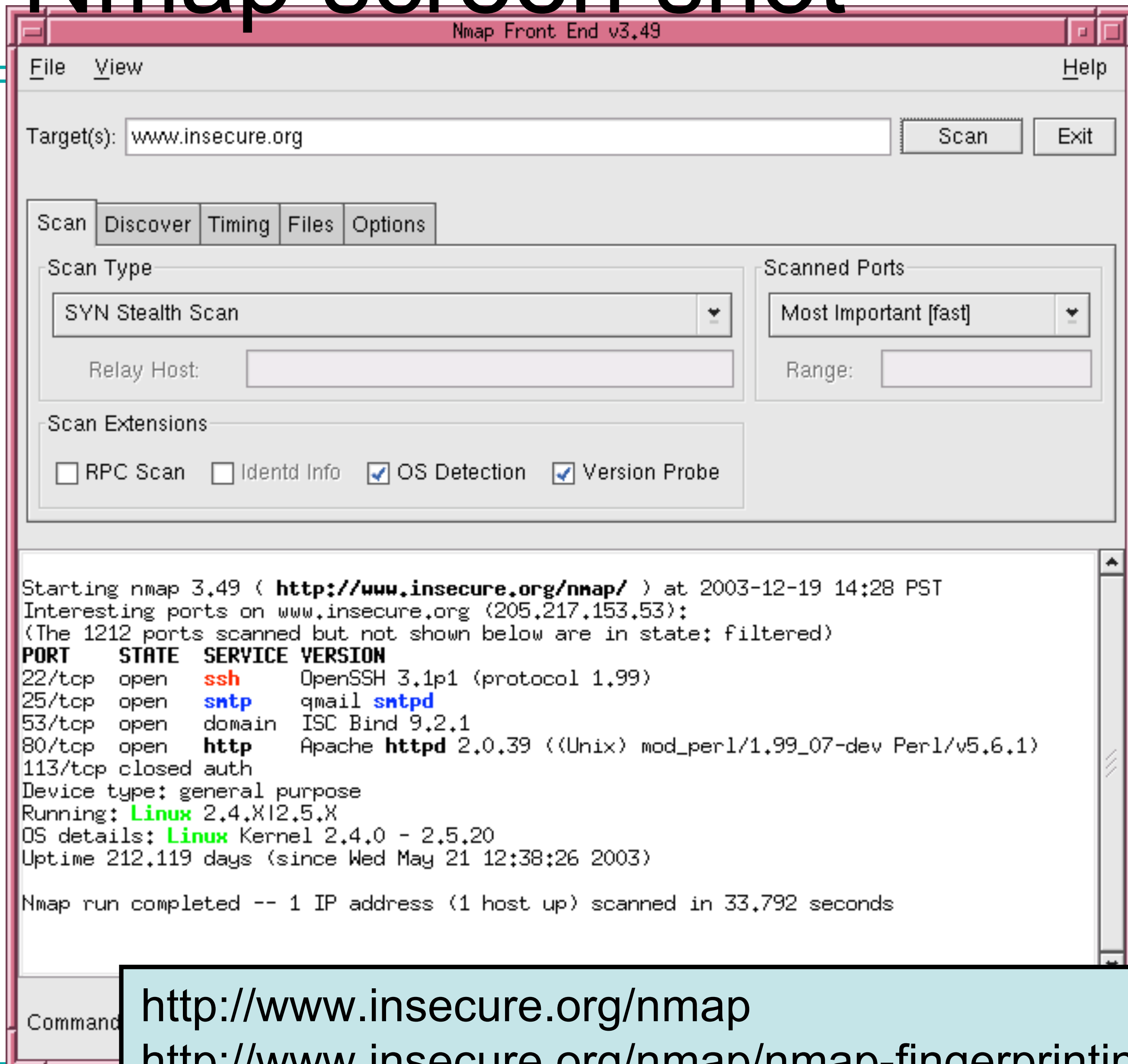
Determining OS

- Exploit different implementations of protocols
  – Different OS's have different behavior in some cases
- Consider TCP protocol, there are many flags and options, and some unspecified behavior
  – Reply to bogus FIN request for TCP port (should not reply, but some OS's do)
  – Handling of invalid flags in TCP packets (some OS's keep the invalid flags set in reply)
  – Initial values for RWS, pattern in random sequence numbers, etc.
  – Can narrow down the possible OS based on the combination of implementation features
- Tools can automate this process

Auditing: Remote auditing tools

- Several utilities available to “attack” or gather information about services/daemons on a system.
  - SATAN (early 1990’s):
    Security Administrator Tool for Analyzing Networks
  - SAINT - Based on SATAN utility
  - SARA - Also based on SATAN
  - Nessus - Open source vulnerability scanner
    - http://www.nessus.org
  - Nmap
- Commercial:
  - ISS scanner
  - Cybercop

Nmap screen shot



http://www.insecure.org/nmap
http://www.insecure.org/nmap/nmap-fingerprinting-article.html

Kinds of Auditing done

- Nessus web pages:
  - Backdoors
  - CGI abuses
  - Denial of Service
  - Finger abuses
  - Firewalls
  - FTP
  - Gain a shell remotely
  - Gain root remotely
  - Netware
  - NIS
  - Port scanners
  - Remote file access
  - RPC
  - Settings
  - SMTP problems
  - SNMP
  - Useless services
  - Windows
  - Windows : User management

- Doing this kind of auditing by hand is complex and error prone
- These tools aren't fool proof or complete.