

# UMSSIA

**DAY IV: WHERE THE WILD THINGS...**

# MALWARE

- ... Or **malicious software**, is any code with “intentional”, undesirable side effects.
- ... Has been known under various guises since the early 1970s.
- The term “virus” originates in Cohen’s 1984 Ph.D. Thesis.
- ... Has had a lot of press due to “spyware” and fast-spreading worms.

# MALWARE TAXONOMY

- A **virus** is **propagating malware** that requires user "action" to propagate
- A **Trojan**<sup>1</sup> is a "legitimate" program with "additional functionality"
- **Spyware** is the new name for a Trojan that steals personal information.
- A **worm** is self-propagating malware
- A **logic bomb** or **time bomb** is malware that triggers under certain conditions
- A **trapdoor** or **backdoor** is a "hole" left by a virus, trojan, or worm...

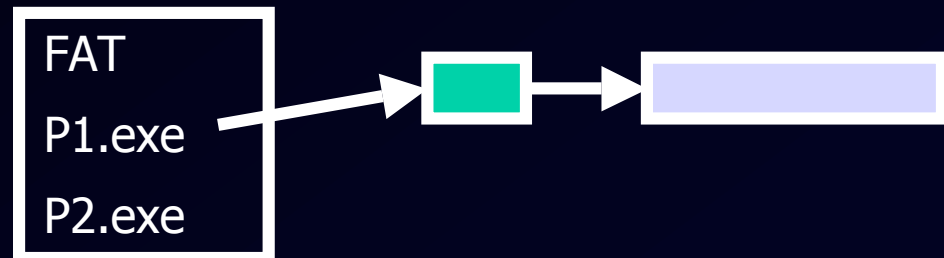
<sup>1</sup> as in horse, or, in Modern Parlance, rabbit.

# VIRUSES

- ... Are typically attached to an executable **host**
  - Jump to end, jump back to main
  - Insert before main



- ... Or intercept or replace "host" program



- Typical host programs include executables, shared libraries, antivirus code...

# MACRO VIRUSES

- **It is now common for "data" formats to include executable content:**
  - MS Office
  - HTML: Javascript, ActiveX
  - PDF, postscript...
- **A macro virus "includes" itself in executable content:**
  - Copy to "startup" macros, write on save
  - Send as email attachment from mail client
- **Recent variants exploit coding errors (e.g. buffer overflows) in the rendering software.**

# BOOT SECTOR

- **PCs have a “bootloading” process:**
  - **Initialize BIOS**
  - **Read “boot sector” from disk, execute**
  - **Loads OS into RAM, starts.**
- **A boot sector virus overwrites the boot sector with malware.**
- **This guarantees the “return” of malware.**
- **Typically they will install a backdoor (maybe)**
- **An old method of propagation: copy to the boot sector of other disks (floppy, CD, USB stick, etc...)**

# ANTIVIRUS STRATEGIES

**Two common strategies of antivirus software:**

- **Look for “integrity failures”**
  - **False positives: Files can be legitimately changed**
  - **False negatives: Infect the integrity checker...**
- **Look for “virus signatures”**
  - **False Negatives: only finds known viruses**
  - **Other trouble: requires “finding” infected hosts and analyzing virus actions**
  - **First generation checkers scanned the end and beginning of each file for a unique sequence of instructions per signature.**

# ANTI<sup>2</sup>-VIRUS STRATEGIES

**Viruses adapted several strategies to hide from antivirus software:**

- **“Bad sectors”**: hide virus code on disk
- **Host integration**: Virus code is “integrated” with the host executable. This makes scanning more expensive... but not impossible.



- **Polymorphism**: “Encrypt” the virus code, and use a small loop to decrypt it.
- **Terminate-Stay-Resident (TSR)**: Hide the virus code in memory, and write it back to disk on interrupts.

# ANTI<sup>3</sup>-VIRUS STRATEGIES

- **Generic decryption** is a technique employed by most “current” AV products.
- Generic decryption finds viruses as follows:
  1. Load the executable in an emulator
  2. Step through looking for a “virus signature” (Code in memory, sequence of system calls, etc.)
- This approach has several problems:
  - Efficiency.
  - Anti-debuggers!
  - The Halting Problem.

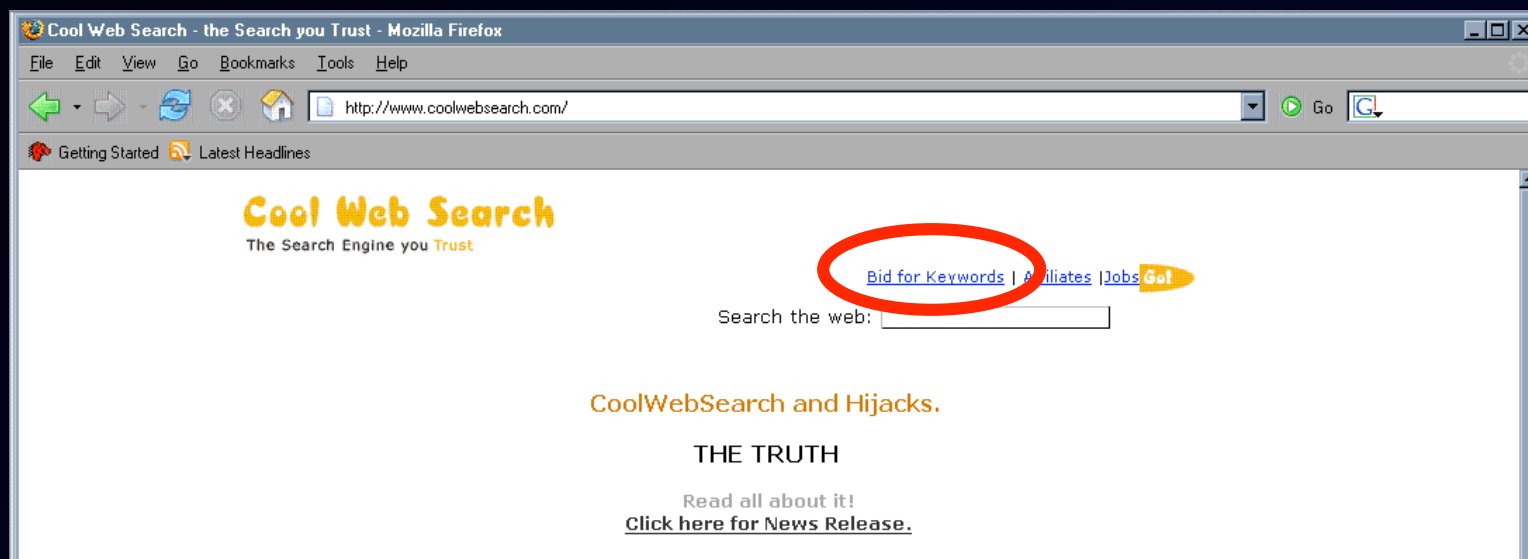
# SPYWARE

... is a form of trojan horse that monitors user data.

e.g. browsing history, web searches, emails

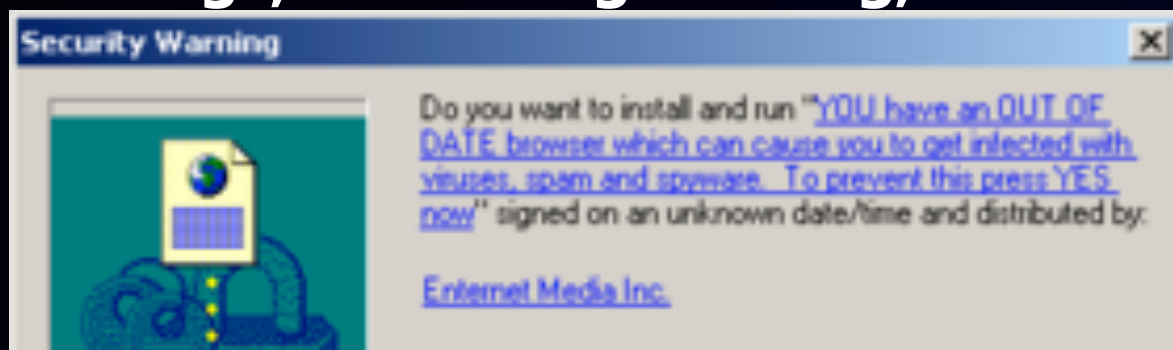
e.g. passwords, bank accounts, credit card numbers

Motives for 2<sup>nd</sup> list are clear. What about the first?



# SPYWARE

... Is typically not propagating, but is installed as a trojan horse. Typical vectors include freeware "piggybacking", social engineering, and IE.

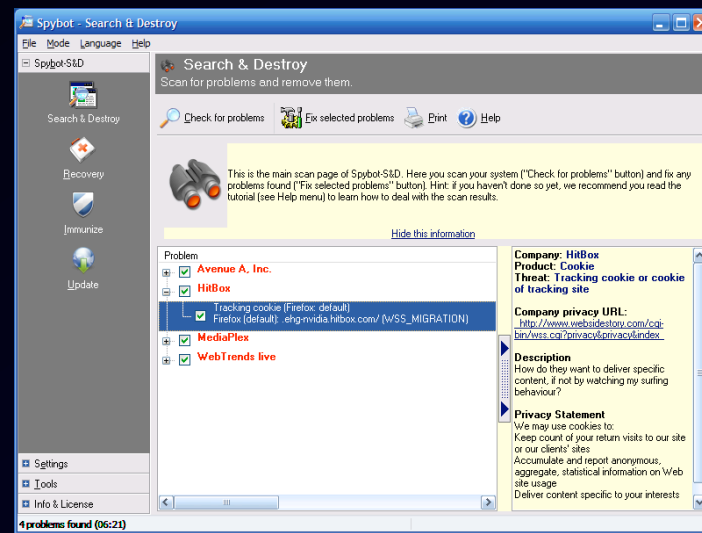
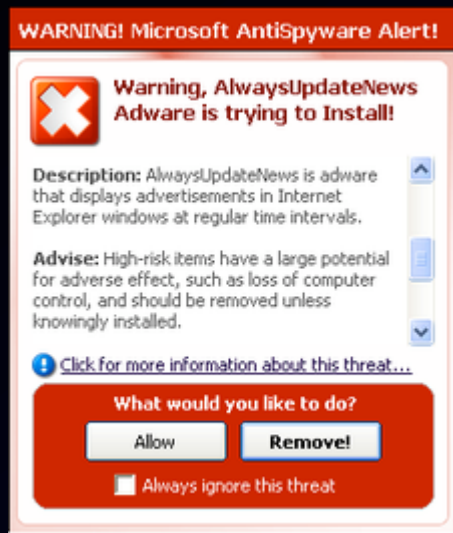


... often employs sophisticated techniques to prevent removal and detection:

- incorporation into OS binaries
- "rootkit" techniques to hide processes
- hiding in file attributes

# ANTI-SPYWARE

Standard programs include SpyBot SD, HijackThis, Windows Defender:



Techniques are related to Virus scanning: check for known signatures, URLs, integrity checking.

# WORMS

- ... are programs that self-propagate through the network. They typically spread through “exploits” in common services
- ... Can spread very quickly, by a variety of mechanisms.
- ... have been in the news mainly because of this DoS-like effect. Some High-Profile worms:
  - Morris Worm, 1988
  - Code Red I/II/ Nimda, 2001
  - Slammer, 2003

# EXAMPLE: MORRIS WORM

- **Cornell grad student Robert Morris Jr.**
- **Released 2 Nov 1988**
- **Spread via several methods:**
  - **fingerd buffer overflow,**
  - **password cracking & .rhosts files**
  - **sendmail bug**
- **Short exploit code for fingerd, sendmail downloaded complete worm after infection.**
- **Stealth:**
  - **Used one-time password to authenticate infected machines**
  - **Ocassionally changed process name, uid**

# THE MORRIS WORM

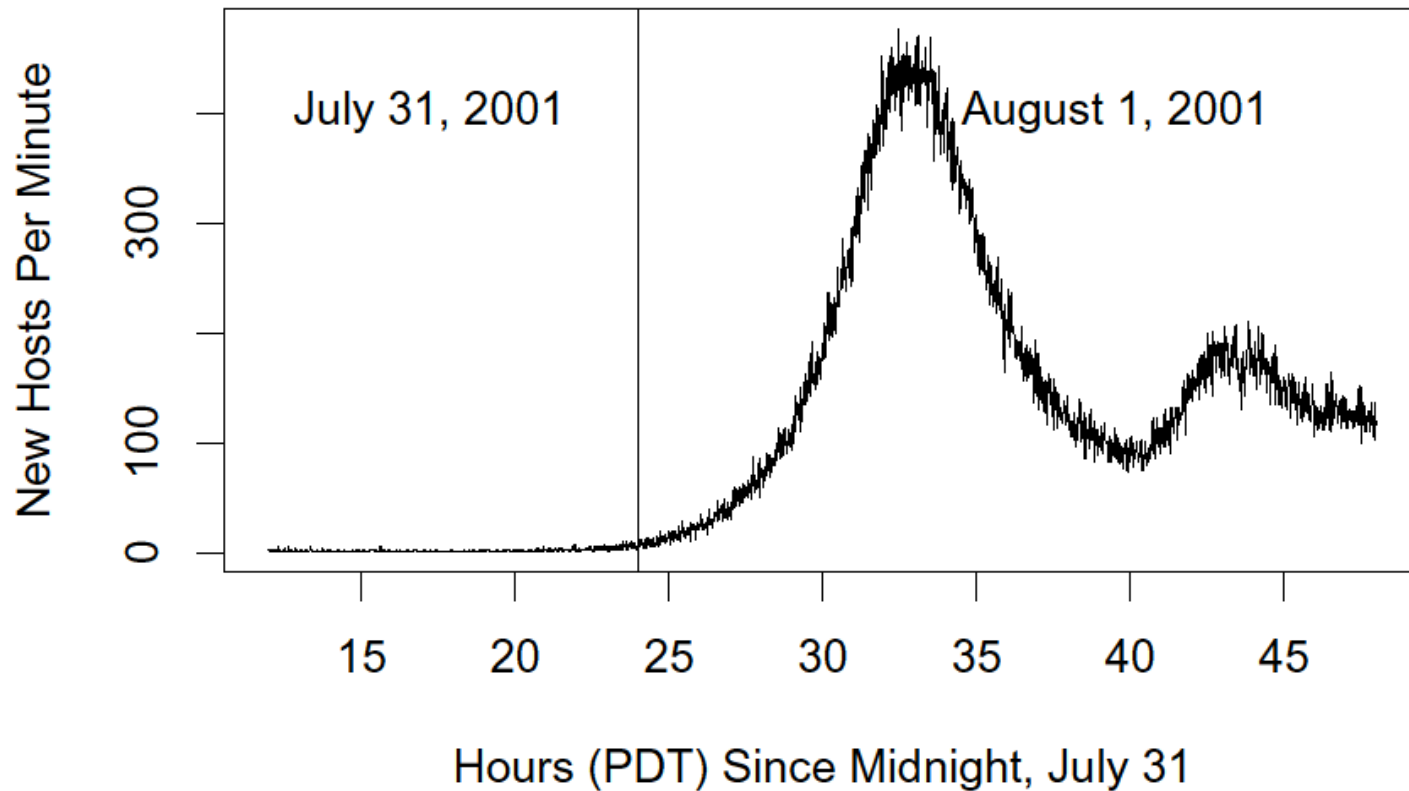
- ... Eventually spread to  $\sim 10\%$  of ARPANET hosts. The traffic it generated temporarily shut down the ARPANET.
- ... Had a "bug": it was meant to infect only one machine, then terminate. Instead, it looped forever.
- ... Prompted formation of the US CERT.

# CODE RED

- **Code red was a random scanning worm, initially released July 13, 2001. It exploited a known (patch available) bug in IIS Web servers.**
- **On the 1st through 20th of each month, it spread. On the 20th through the end of each month: DDoS on [www.whitehouse.gov](http://www.whitehouse.gov)**
- **Its "payload" was a web site defacement:  
"HELLO!  
Welcome to  
<http://www.worm.com> !  
Hacked by Chinese!"**
- **Code Red found new hosts via random scanning of IPv4 address space. The original failed to properly seed the RNG, resulting in slower growth.**
- **Rereleased with correct RNG behavior July 19.**
- **New Bug: DDoS tool broken, dies on 20<sup>th</sup> each month**

# CODE RED PROPAGATION

Return of Code Red Worm

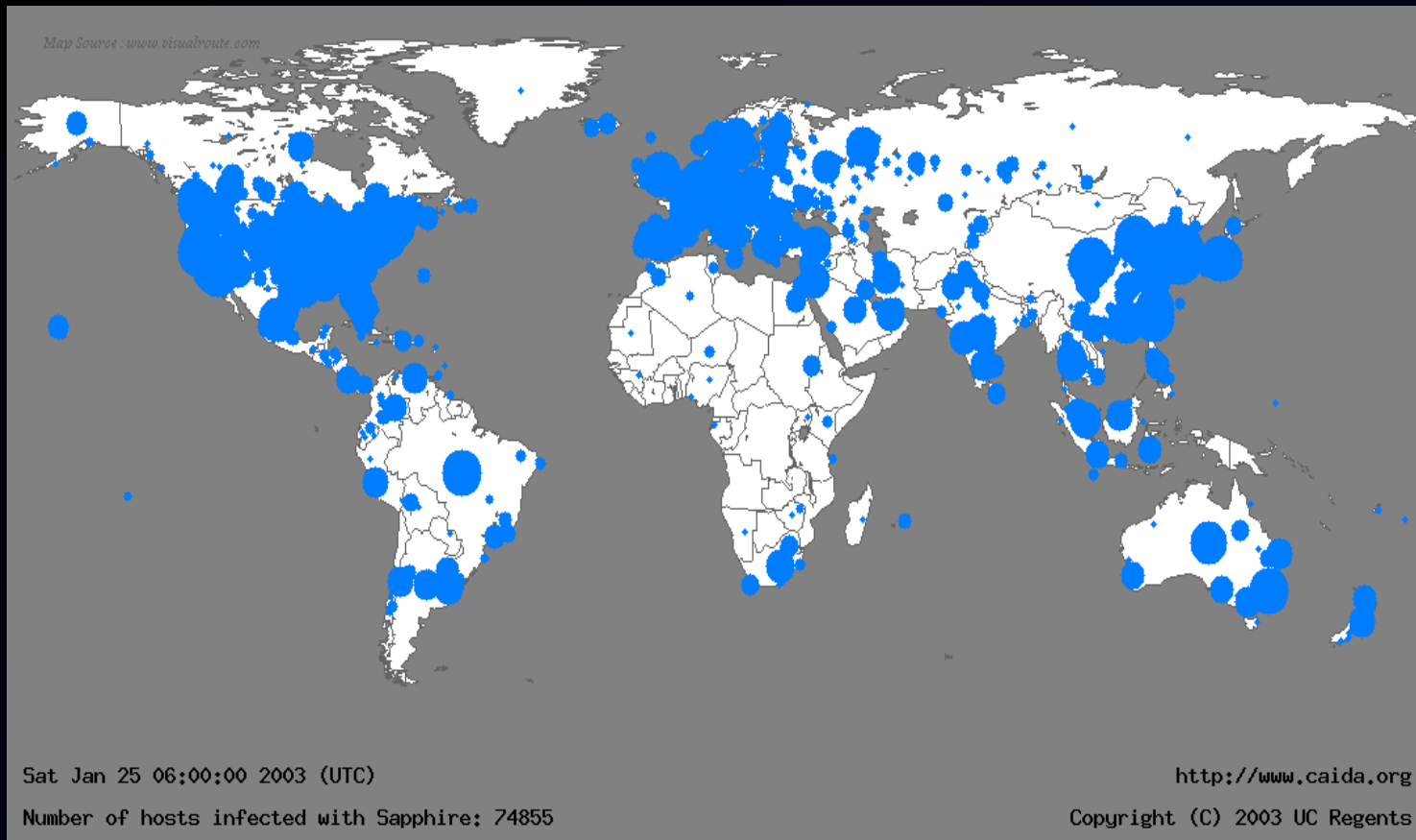


[Paxson et al.]

# MORE RAPID MALWARE

- **Code Red II: August 4, 2001**
  - Kills Code Red, installs root backdoor
  - Programmed to die Oct. 1
- **Nimda: September 18, 2001**
  - Multimodal transmission: IIS exploit, IE exploit, Email virus, open shares, Code Red II backdoor
- **The three worms form an “ecosystem”:**
  - Code Red II “wipes out” CRI, then dies...
  - Code Red I comes back due to bad clocks...
  - CRII is revived by Nimda...
- **Combined, Code Red I,II, and Nimda infected about 1M servers/clients...**

# SLAMMER: BEFORE AND AFTER



[Paxson et al.]

# SLAMMER

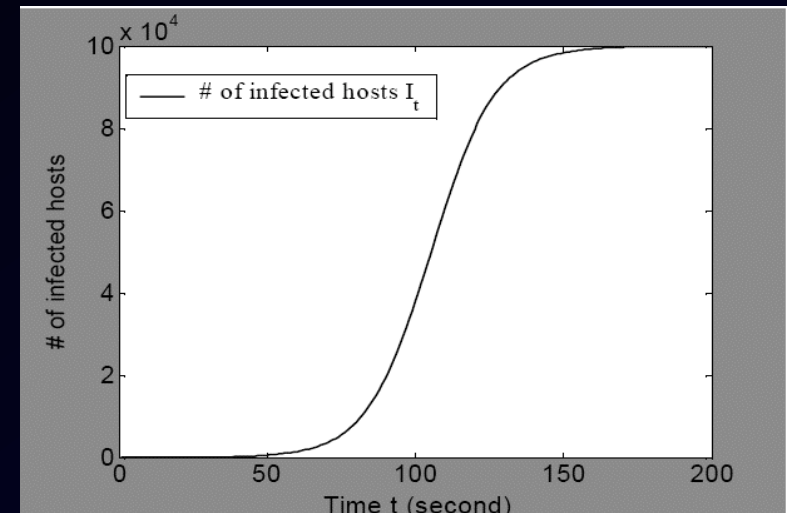
- ... is another “rapid malware” Worm, first released on Jan 25, 2003
- ... Exploits a buffer overflow in MS SQL Server. The payload fits in a single packet, that is delivered over UDP.
- ... Thus its infection rate is limited only by bandwidth.
- ... Infected 75K hosts in 10 minutes.
- Like Code Red, Slammer had a broken RNG, and only scanned about 1/8-1/4 of the available address space.

# WORM PROPAGATION

- **Random scanning worms** find new hosts by picking a random IP, (on subnet, routable...)
- **Permutation worms**: synchronize their scanning efforts by breaking IP addresses into chunks.
- **Meta-Server worms** use a server to search for vulnerable hosts.
- **Hit List worms** find vulnerable hosts before launch. This boosts the worm's "initial population," and avoids "scanning behavior"
- **Topological worms** use existing host relationships to find new victims.
- **Contagion worms** propagate inside existing host communications.

# WORM PROPAGATION SPEED

- **Let**
  - $i(t)$  = % infected hosts,
  - $\beta$  = "contact rate"
  - Then  $di/dt = \beta i(1-i)$
- **So  $i(t) = e^{\beta(t-T)} / (1 + e^{\beta(t-T)})$**



**This predicts faster growth than observed. Why?**

# WORM DEFENSES

- **If we want to thwart worms, we have two choices:**
  1. **Detect and “heal” infected hosts at a high rate**
  2. **Slow the infection rate to a level that allows (1)**
- **Detection:**
  - **Use “network telescope” or “honeynet” to find unusual activity (backscatter, connection attempts, outgoing connections...) and extract a signature.**
  - **Use “network sensors” to find widely dispersed packets.**
- **Containment:**
  - **Tarpits, Scan Suppressors, Clever TCP/IP stacks**

# DENIAL OF SERVICE

... is an attack on network availability. The goal of a DoS attack is to prevent a computer from accessing the network.

DoS attacks fall into two broad categories:

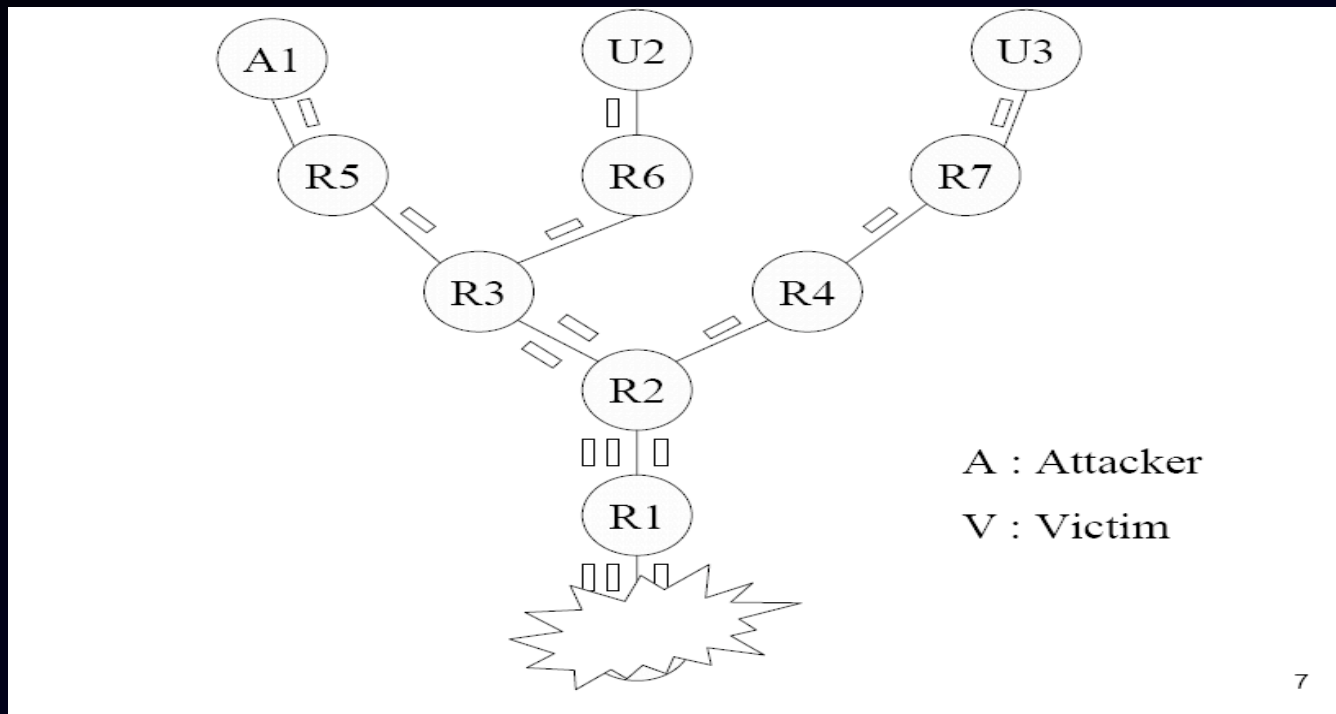
**Distributed Denial of Service (DDoS)** is a “brute force” attack.

**Protocol-based attacks** attempt to deny service with as few packets as possible.

# DDOS

... is the "Brute force" approach to DoS:

1. Get control of lots of boxes (e.g. with a worm and backdoor)
2. Point them all at [www.victim.com](http://www.victim.com)
3. Go offline, disavow all knowledge, etc...



# BOTNETS

- A **botnet** is a group of compromised systems with “remote control” software installed on them.
- This software typically supports:
  - Upgrades
  - Authentication, to prevent “stealing” of zombies
  - Arbitrary program payloads
- Botnets are used as an attack base for various activities:
  - DDoS attacks
  - Spam forwarding
  - Launching pad for new exploits/worms
  - ...
- The HoneyNet project observed bot nets with over 80K zombies in 2001.
- Dutch Police discovered a 1.5M node botnet in 2005.

# **FILTERING**

**Typical filtering options include:**

- **only allow packets from known hosts**
- **Check for reverse path: only accept packets from X if there is an outgoing connection to X**
- **Ingress/egress filtering**
  - **Packets in must have outside source / inside destination**
  - **Packets out must have inside source / outside destination**
- **Rate limiting**
  - **Limit rate of ICMP packets and/or SYN packets**

# PUZZLES

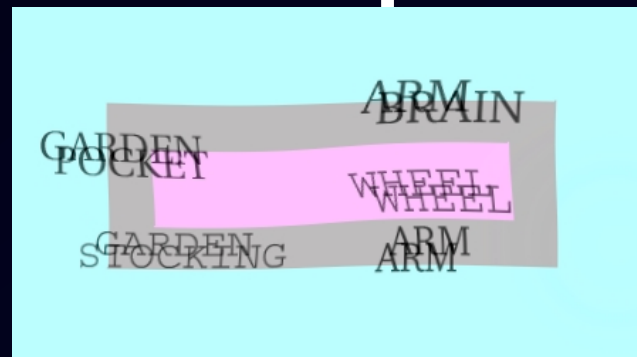
... are problems that take work to solve but are easy to check.

... can be used to prevent some DoS attacks.

**Example: (RSA client puzzle protocol)**

- Normally, S accepts any connection request
- During high load, S responds with a puzzle
- S allows connections only for clients that solve the puzzle within some regular TCP timeout period

**Application level example: CAPTCHA...**

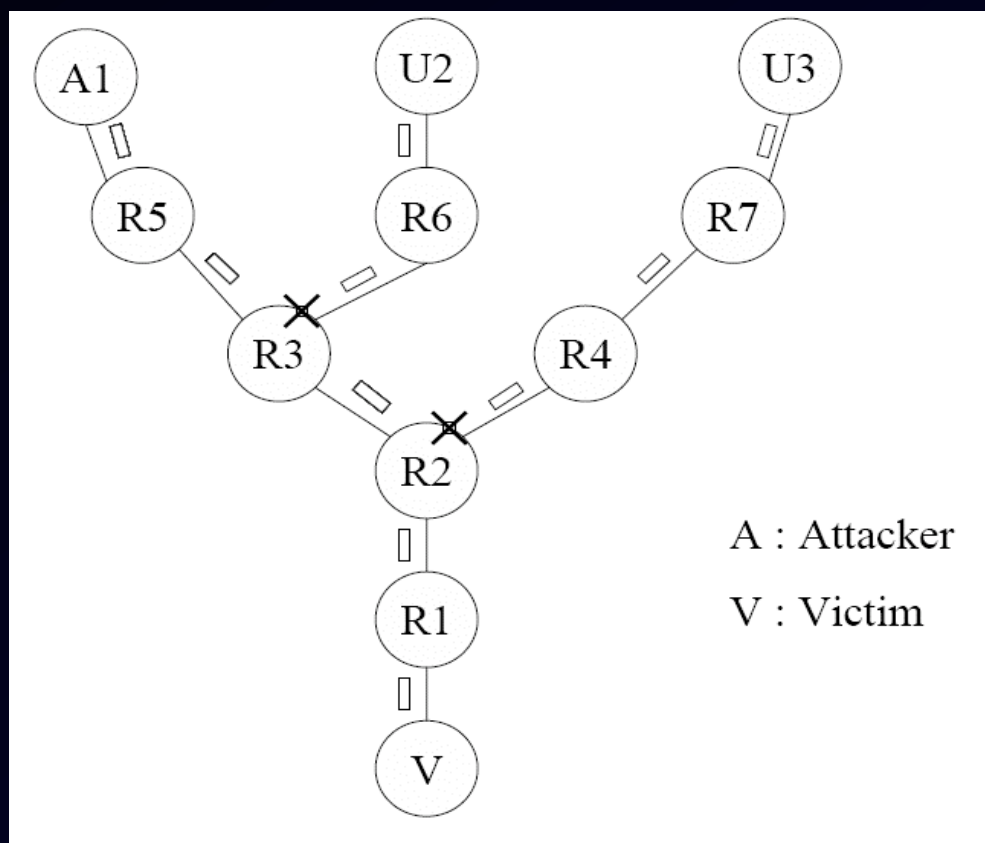


# TRACEBACK

... is the process of trying to identify the path from the Victim to the agents of a DDoS attack.

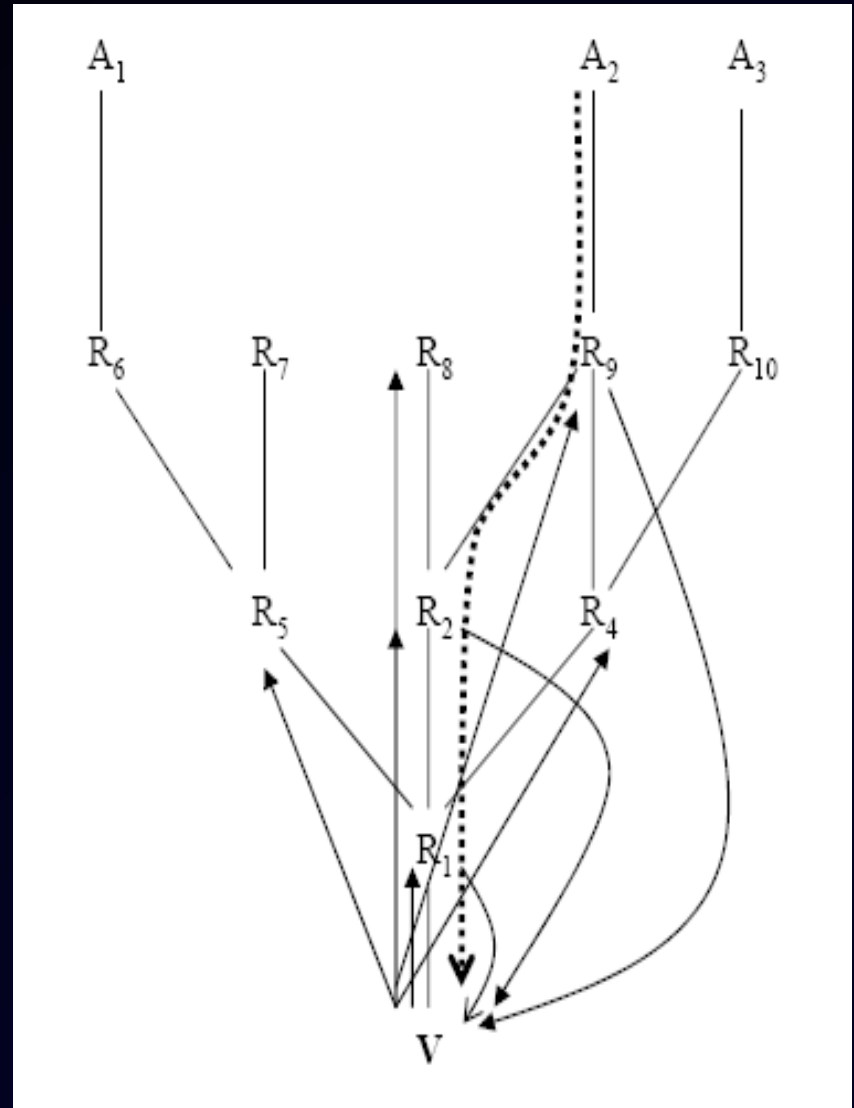
**Typical assumptions:**

- **Most routers are uncompromised**
- **Each attacker sends many packets**
- **The route from each attacker to the victim is stable**



# LINK TESTING

- **Start from the victim and test upstream links: If disabling the link slows the attack, it is on the path.**
- **Recursively repeat until all sources are located**
- **The attack must remain active until the trace is completed**



# LOGGING

- **If we can change the routers, traceback is an easier problem.**
- **In the logging approach, each router keeps a log of packets it has seen.**
- **When an attack packet P arrives, we use the same recursive procedure:**
  - **Ask each router R along the edge of the path if it has seen packet P**
  - **If yes, add R to the path.**

**Main problem: lots of packets!**

# PACKET MARKING

A **packet marking** traceback scheme stores path information in **packets** rather than routers.

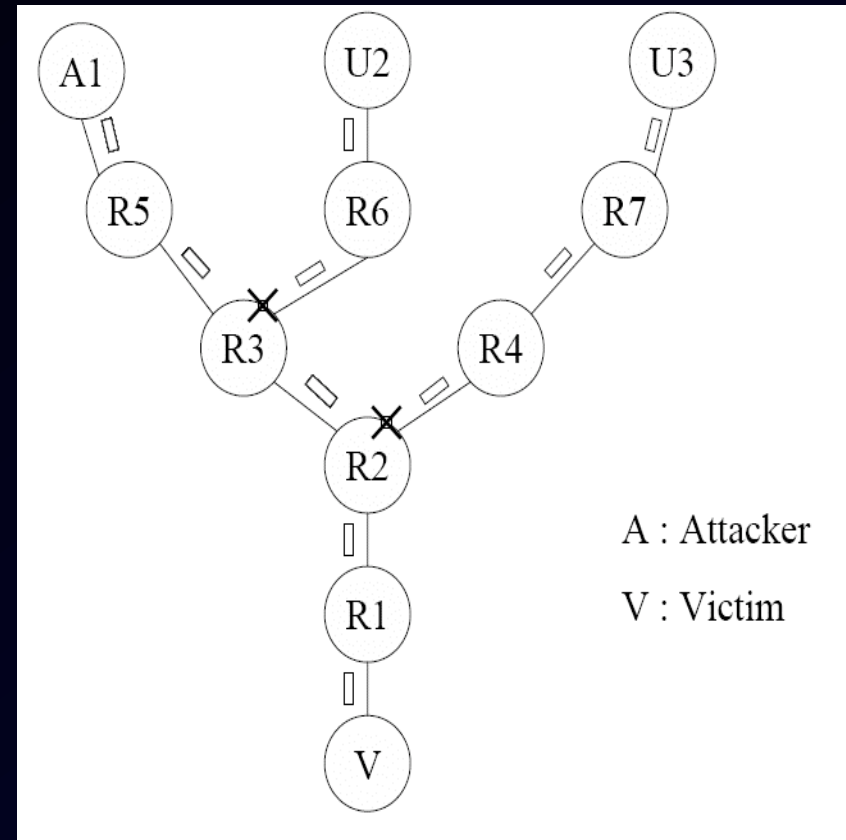
**Victims use this info to reconstruct the path.**

**The simplest idea is to store the whole path:**

- Each router adds its name to each packet
- The victim reads the path from an attack packet

# PROBABILISTIC MARKING

- A DDoS node sends many packets along the path: we don't need to see the full path in every packet.
- In **probabilistic marking** schemes, each router writes part of the path in some of the packets
- This takes less space per packet
- Given "enough" attack packets, all of the path will show up



# NODE SAMPLING

**In the node sampling approach, each router writes its own address into each packet with probability  $p$**

**A Router at distance  $d$  has probability  $p(1-p)^d$  of showing up in a marked packet**

**To infer path order at depth  $d$  we need about  
 $1/(p^2(1-p)^{d-1})^2$  packets**

**There is no easy way to separate multiple paths.**

# EDGE SAMPLING

- **Since it is hard to find the edges with node sampling, why not sample edges? In each packet we will keep:**
  - **One edge in the path: start and end IP addresses**
  - **Distance: number of hops since edge stored**
- **Marking procedure for router R:**
  - Pick random  $r \in [0,1]$ .**
  - if ( $r < p$ ) then**
    - write R into **start** field**
    - write 0 into distance field**
  - else**
    - if distance = 0 write R into **end** field**
    - increment distance field**

# EXAMPLE

## 1. Packet received

- $R_1$  receives packet from source or router
- Packet contains space for start, end, distance

## 2. Begin writing edge

- $R_1$  chooses to write start of edge
- Sets distance to 0

## 3. Finish writing edge

- $R_2$  chooses not to overwrite edge
- Distance is 0: write end of edge, increment distance to 1



## 4. Increment distance

- $R_3$  chooses not to overwrite edge
- Distance  $>$  0: Increment distance to 2

# PATH RECONSTRUCTION

- **Extract information from attack packets**
- **Build graph rooted at victim**
  - Each (start,end,distance) tuple provides an edge
  - Eliminate edges with inconsistent distance
- **# packets needed to reconstruct the path:**

$$E(X) < \ln(d) / [p(1-p)^{d-1}]$$

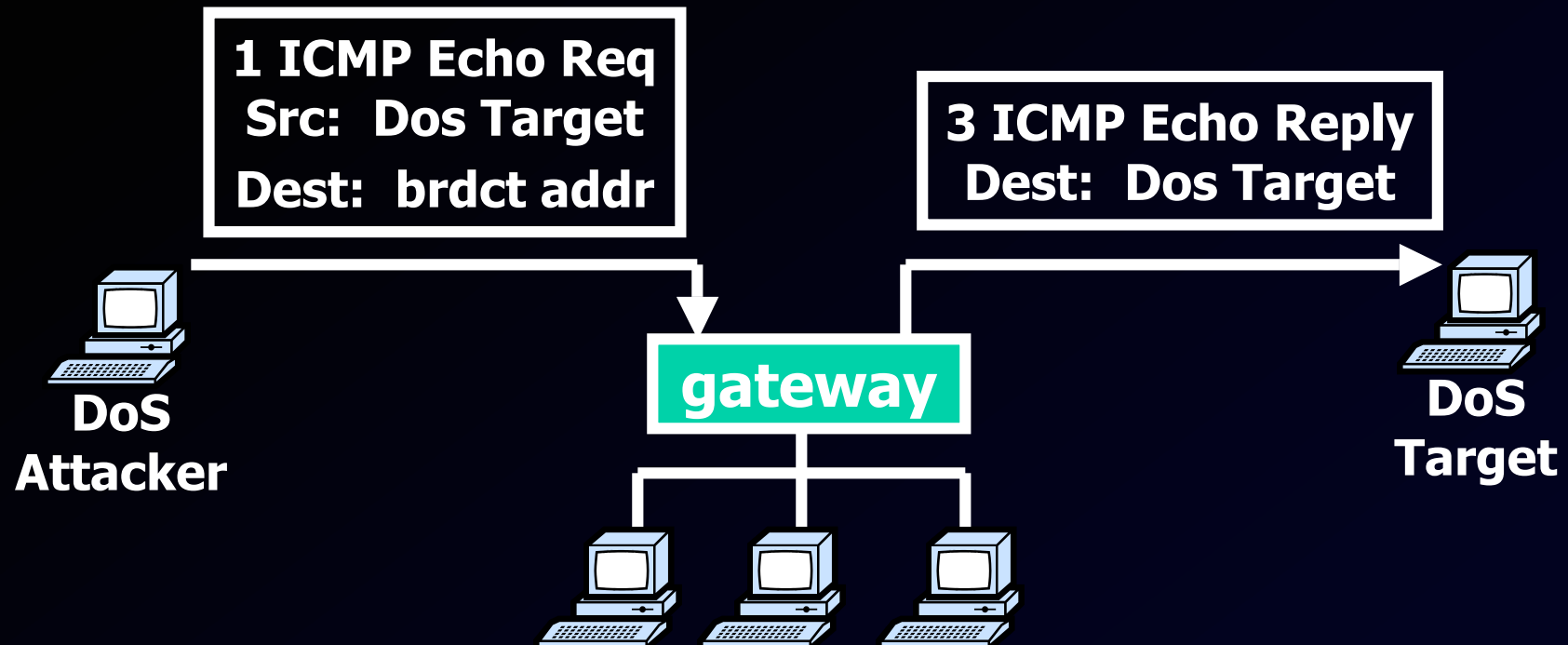
where  $p$  = marking probability,  $d$  = length of path

(Optimal when  $p = 1/d$ )

# WIRELESS EXAMPLE

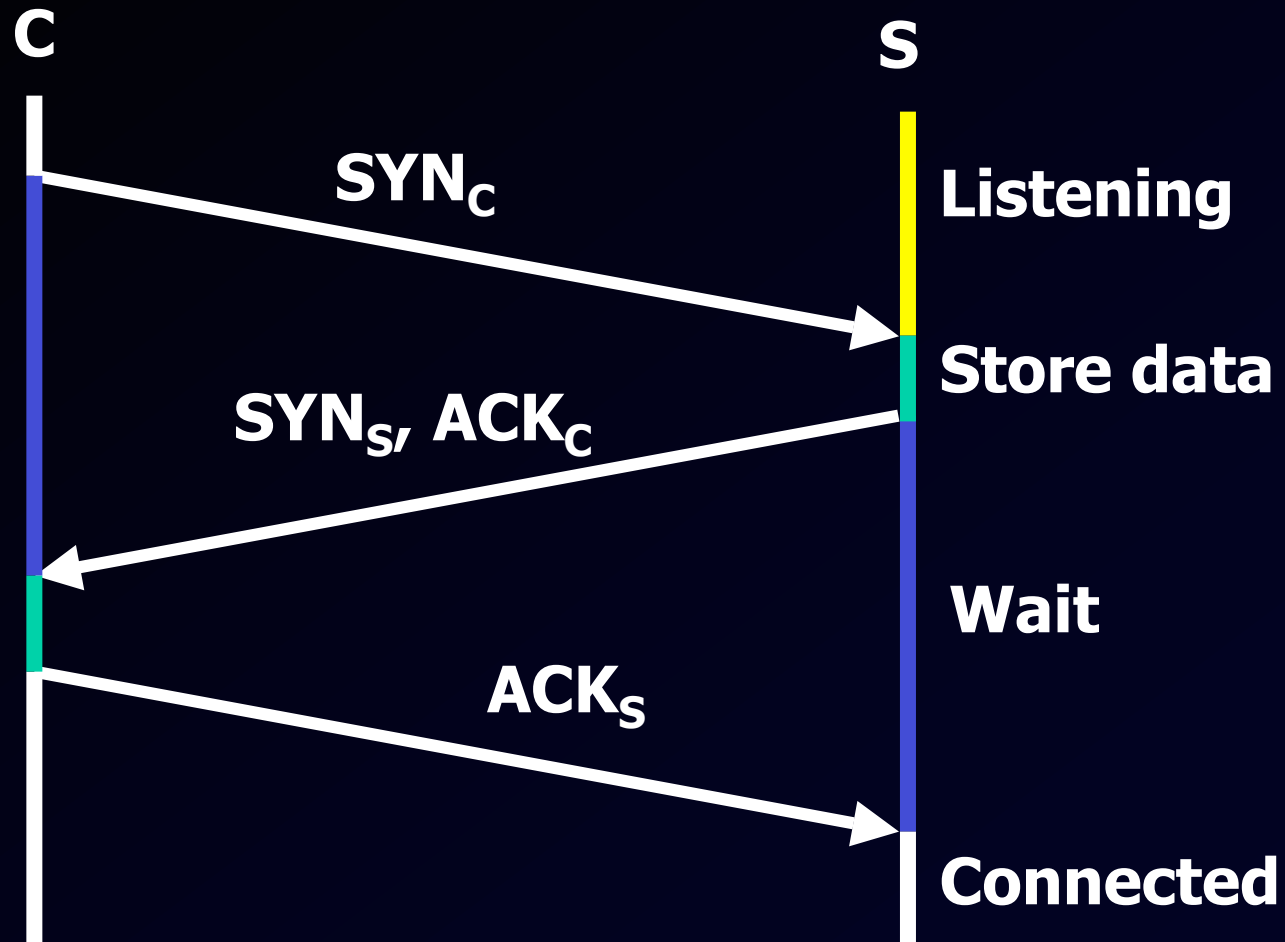
- **Cell phones have two requirements:**
  - Calls should be placed and billed correctly
  - Voice data should be delivered quickly
- **Designed with two channels**
  - Control channel, slow and very reliable
  - Data channel, high bandwidth & lossy
- **“Text messages” (SMS): don’t tolerate loss.**
  - So use the Control Channel!
- **Txt2web interfaces: send from a computer!**
- **DoS: sending 165 SMS messages/second can wipe out Manhattan’s cellular network.**

# SMURF ATTACK

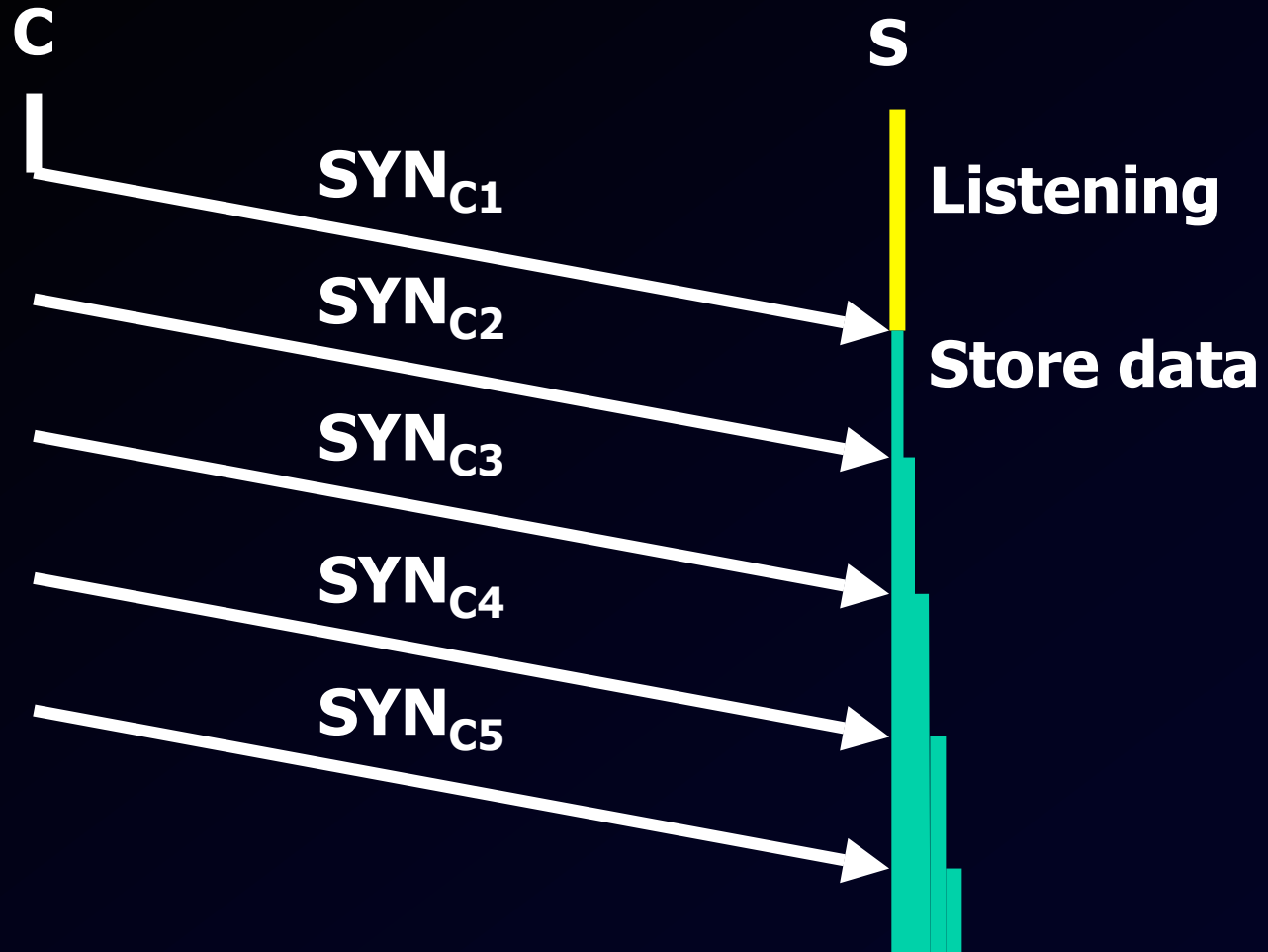


- **Send ping request to broadcast addr**
- **Every host on target network generates a ping reply (ICMP Echo Reply) to victim**
- **Ping replies overload victim**

# TCP HANDSHAKE

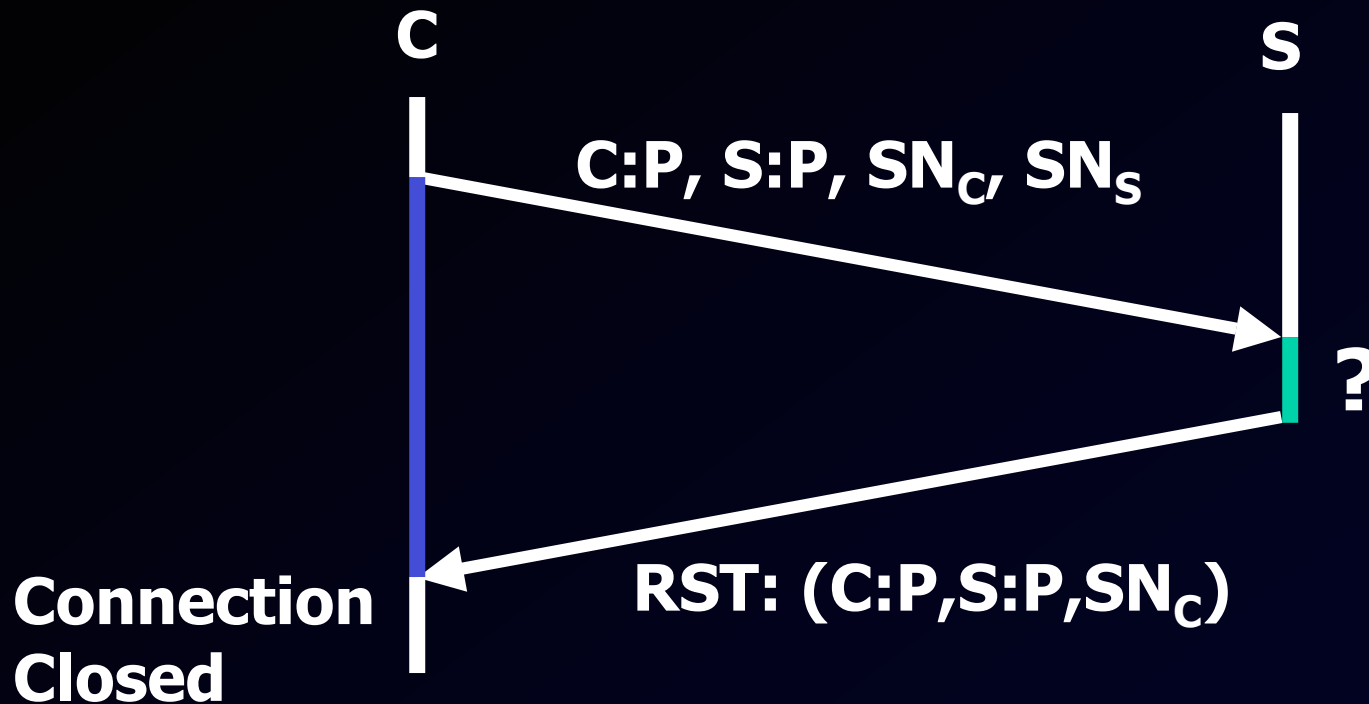


# SYN FLOODING



# TCP RESET

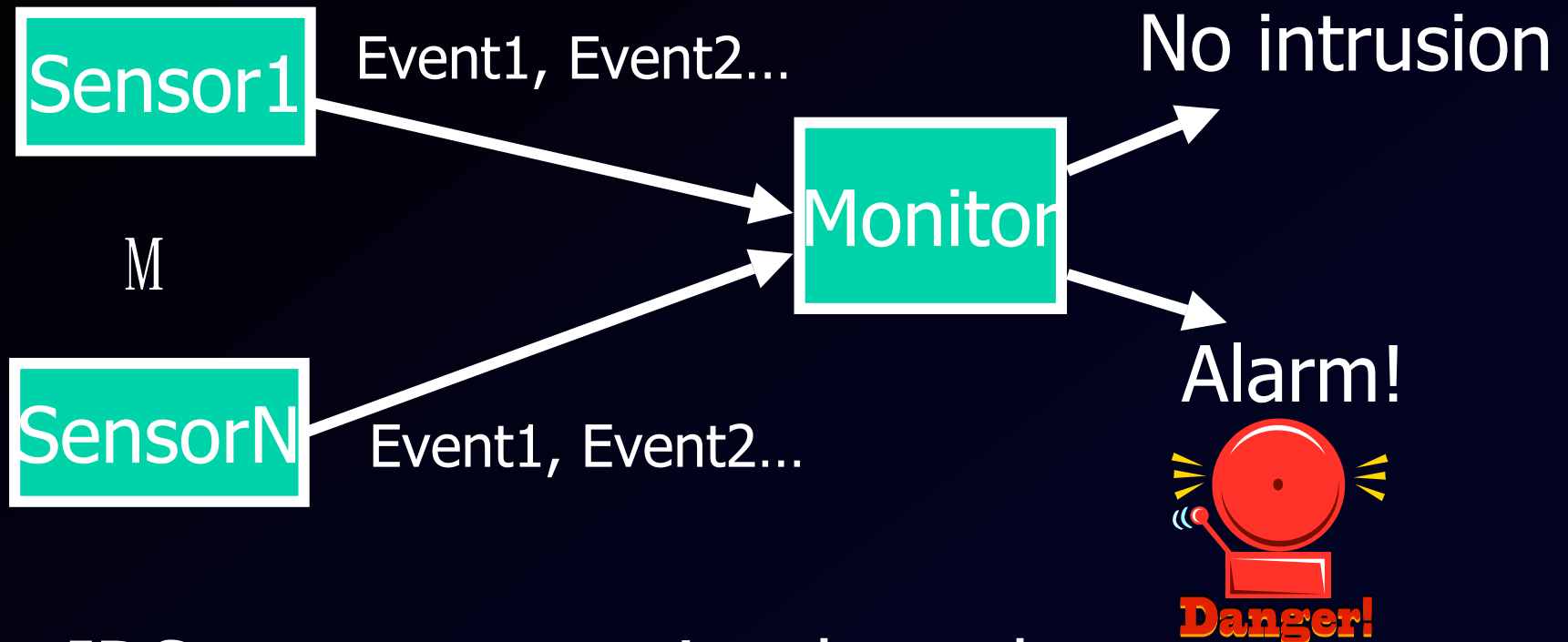
**A TCP Connection can die in three ways:  
Timeout, FIN, and RST.**



**C will accept a RST within 65K of the last SN.**

**Predictable ISNs compound this problem.**

# INTRUSION DETECTION



Host IDS: sensors monitor host data

Network IDS: sensor monitors network

Distributed IDS: combine host/network monitors

# ERRORS

- A **False Positive** error occurs when a non-intrusion causes an alarm, e.g.
  - Squirrel chews through sensor cable
  - Printer driver scans subnet for printer
  - User mistypes password 3x
- A **False Negative** error occurs when an intrusion does not result in an alarm.
  - “Stealth scans”, “Mimic attack”
- False Positive Rate (FPR) =  
**#False Positives / #Normal Events**
- False Negative Rate (FNR) =  
**#False Negatives / #Intrusions**

# BASE RATE FALLACY

- **Suppose the U of M has**
  - **10M network flows each day**
  - **100 flows are genuine intrusions**
- **Suppose we use an IDS with a False positive rate of 0.1%. Then:**
  - **There are 10K false alarms.**
  - **So 99.9% of alarms are false positives**
- **We would need a FPR of just 0.001% to make 50% of alarms real.**

# **SIGNATURE MATCHING IDS**

- **The “misuse detection” problem is to find behavior that “looks like” an intrusion.**
- **The basic form of such systems is:**
  - 1. Collect information about known attack methods and types, such as SrcAddr, SrcPort, DstAddr, DstPort, Protocol, Packet contents**
  - 2. Create and look for signatures: a Code red packet, a port scan, etc...**

# SIGNATURE DETECTION

- **Language to specify intrusion patterns**
  - **4-tuple/protocol values for potential intrusion**
    - Eg External host -> file server (port 110, 135, ...)
    - Eg Internal workstation -> external P2P host
  - **Packet contents: Could be single or multiple packets (stream reconstruction)**
  - **Possibly model of protocol/app finite state machine**
- **Lots of state in pattern matching engine**
- **Example rule:**
  - **alert tcp any any -> myip 21 (content:"site exec"; content:"%"; msg:"site exec buffer overflow attempt";)**

# SIGNATURE MATCHING

- **Advantages:**
  - very low false positive rate
  - Automated extraction is possible
- **Disadvantages**
  - Only detects already known attacks
  - Simple changes to an attack can defeat detection, for example:
    - scan even ports, then odd ports.
    - "rm -rf /\*" -> "rm -rf ../../../../../../../../../\*"

# ANOMALY DETECTION

**1. Try to identify what is normal, e.g. "normal" command sequence or common 4-tuple/protocol, session length, intervals, etc...**

**2. Look for major deviations (outliers), e.g. unusual target port, source addr, or port sequence (scan)**

**Sometimes (but not always): Apply AI/Machine Learning to "learn" what is normal.**

**Advantage: more robust to "altered" attacks.**

# ANOMALOUS PROBLEMS

- **High false positive rate**
- **Attacks might not be obvious until too late**
- **Attacks can hide in “normal” traffic**
- **Require training on “known good” data**
- **Problems when what’s “normal” changes**
  - **Eg, flash crowds, new users, new applications...**

# HOST-BASED IDS

**... monitor a single host to detect intrusions.**

**Typical “sensors” include:**

- Disk & memory contents**
- User input and commands**
- System calls**

**... Typically equate “intrusion” with privilege escalation, eg remote ! local ! root.**

**... Are vulnerable to various attacks, e.g.:  
replace HIDS process/binaries, “Mimic”  
attack, context problems, subvirtion.**

# EXAMPLE: TRIPWIRE

- A “standard attack” might go like this:
  - Gain user access to system
  - Gain root access
  - Replace system binaries to set up backdoor
  - Use backdoor for future activities
- **Tripwire** is a simple anomaly-based HIDS that monitors system binaries:
  - Compute hash of key system binaries
  - Compare current hash to earlier stored hash
  - Report problem if hash is different
  - Store reference hash codes on read-only medium
- Example attack: “rootkit” replaces binaries with new versions **with the same hash.**

# EXAMPLE: IDES

**IDES was a HIDS developed in the 1980s by Dorothy Denning, SRI. It worked as follows:**

- **For each user, store daily count of certain activities**
  - **E.g., Fraction of hours spent reading email**
- **Maintain a list of counts for several days**
- **Report an anomaly if the current count is outside the weighted norm**

**The main vulnerability of IDES was the ability to “train” the system to accept intrusive behaviour**

**IDES proposed a solution: include signatures of known intrusive acts and raise an alarm if the profile matches the attack profile.**

# EXAMPLE: SYSCALL HIDS

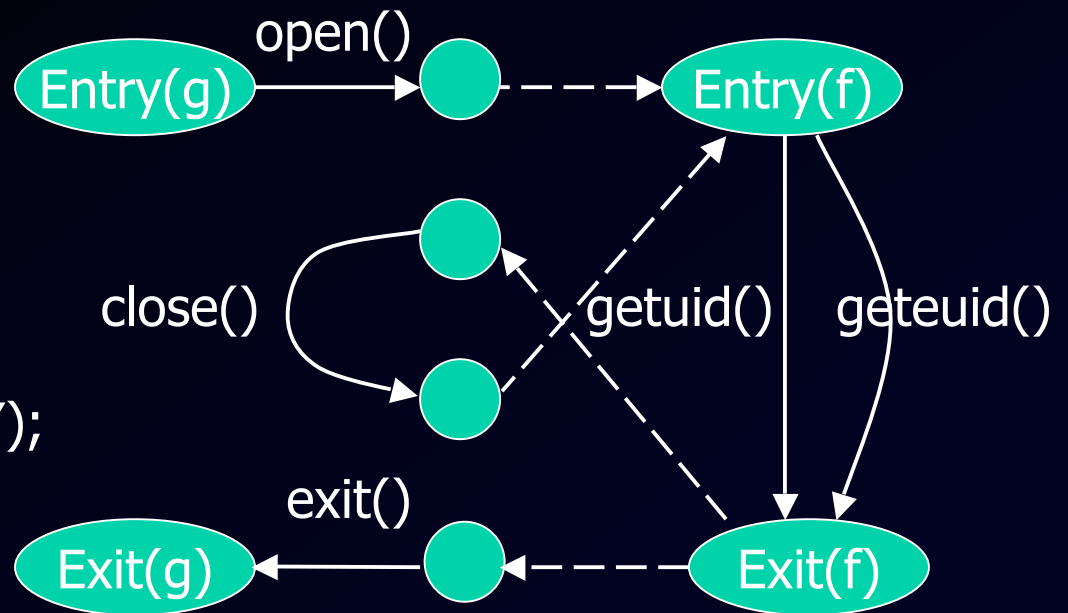
- **In the 1990s**
- **Build traces during normal run of program**
  - **Example program behavior (sys calls)**  
open read write open mmap write fchmod close
  - **Store all 4-call sequences in a file:**  
open read write open  
read write open mmap  
write open mmap write  
open mmap write fchmod  
mmap write fchmod close
- **Use an “Order 4 Markov Model” to compute the probability of a new syscall sequence...**
  - **Compute # of mismatches to get mismatch rate**
  - **For example, report an anomaly if you see:**  
open read read open mmap write fchmod close
- **Vulnerability: “Mimic attack”**

# EXAMPLE: WD01

- **The current trend is toward using formal models of correct behavior**
- **For example** [Wagner, Dean IEEE S&P '01]
  - **Build automaton of expected system calls (automatically, from source code)**
  - **Monitor system calls from each program**
  - **Catch violation**
- **Results so far: lots better than not using source code!**

# EXAMPLE

```
f(int x) {  
  x ? getuid() : geteuid();  
  x++  
}  
g() {  
  fd = open("foo", O_RDONLY);  
  f(0); close(fd); f(1);  
  exit(0);  
}
```



# NETWORK IDS

- **A network-based IDS monitors network traffic for intrusions, e.g.:**
  - **DoS**
  - **Exploiting bugs in protocol, application, OS**
  - **Install worm, virus, bot, spyware...**
- **Challenges for NIDS:**  
**fragmentation, data volume, “low and slow” attacks...**

# EXAMPLE

- A typical attack is “port scanning” to find out what services a host is running.
- For example, the Nmap tool:  
(<http://www.insecure.org/nmap/>)
  - Determines OS/hostname/device type via “service fingerprinting” (eg, IRIX listens on TCP port 1)
  - Determines what service is listening on a port and can determine application name, version
  - Operates in optional obfuscation mode
- Afterwards, attackers can exploit known vulnerabilities in the OS/applications found via Nmap.

# EXAMPLE: SNORT

- ... is a signature-based, portable, opensource NIDS with over 2 million downloads, and 100K active users.
- ... scans a tcpdump log, and finds connections matching attack signatures.
- ... uses regular expressions that match known attacks.
- **Example sig:**
  - alert tcp any any -> [a.b.0.0/16,c.d.e.0/24] 80  
( msg:"WEB-ATTACKS conf/httpd.conf attempt"; nocase; sid:1373;  
flow:to\_server,established;  
content:"conf/httpd.conf"; [...] )

# EXAMPLE: BRO

- ... is a high-speed "policy-based" NIDS. It uses scripts that monitor connections, and check for conformance to "expected" behavior based on protocol
- ... logs, for all TCP connections (via SYN/FIN/RST packets): start time, duration, service, addresses, sizes, etc. It also identifies the application based on these packets.
- ... supports many standard applications: DNS, FTP, HTTP, SMTP, NTP, ...
- ... on Telnet and Rlogin generates the following events:
  - login\_successful, login\_failure, activating\_encryption, login\_confused, login\_input\_line, login\_output\_line
- In its first five months of operation, Bro found 120 UCB break-ins (60 root compromises)

# DISTRIBUTED IDS

- **Idea: combine data from many sensors**
  - HIDS data from many hosts
  - NIDS data from multiple segments
- **Positives: can detect “stealthy” scans, possibly lower false alarm rate, etc.**
- **Negatives: much more data to scan, exchange, etc.**

# HONEYPOTS & TARPITS

A **honeypot** is a closely monitored network decoy or decoy(s). Honey pots:

- Distract adversaries from more valuable machines on a network (?)
- Provide early warning about new attack and exploitation trends
- Frequently consist of multiple VMs that use up the “empty” IP address space on a network.

A **tarpit** is essentially a slow honeypot that accepts connections, but prevents resets.

# INTRUSION PREVENTION

- **Idea: respond to detected intrusion**
  - **Reset connection**
  - **Block host(s) at firewall**
  - **Etc...**
- **Big problem: self-DoS.**
  - **Attacker makes it look like potentially legitimate connections are intrusions**
  - **Hosts get blocked at firewall**

# ATTACKS ON THE IDS

- **Witty worm: buffer overflow in commercial IDS systems**
- **DoS**
  - **On NIDS: crash system or overwhelm monitor**
  - **Using NIDS: see IPS**
- **Subterfuge:**
  - **Retransmits, NULs in packets, TTLs, checksums, etc..**
  - **“rob<DEL><BS><BS>ot”**