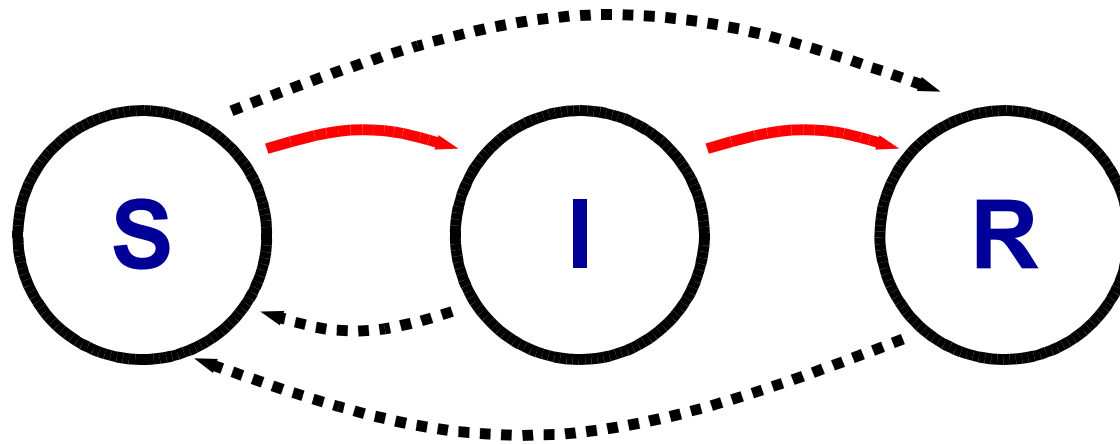


# A finite state model of individual to study the spreading



## Model states

- **susceptible** = Host that may be infected
- **Infected** = Infected host
- **Recovered** = Host that cannot be infected

## Typical transition sequences (red arrows)

The host runs the software that is vulnerable (**potential**).

The worm has exploited the vuln and successfully attacked the node (**infected**).

The infection is detected and the system reconfigured (**recovered**).

# A set of diff equations

## Classic epidemiology

- [Kermack and McKendrick, 1927]
- All the nodes follows the red paths in the automata (P to I, I to R)

$$\frac{ds}{dt} = -\beta si$$

$$\frac{di}{dt} = \beta si - \gamma i$$

$$\frac{dr}{dt} = \gamma i$$

**s = potentially infected**

**i = infected**

**r = recovered**

**Beta = infection rate**

**Gamma = recovery rate**

**Gamma may be neglected  
in the case of worms  
because the time to spread  
is very litte**



## Kermack and McKendrick model

---

- $\beta$  is a function of
  - The function to generate the IP addresses
  - The number of the systems affected by the vulns
  - It increases with the virulence
  - The model assumes that a node can infected any other node = complete connection and no defence
- $\gamma$  should not be neglected anytime
  - The spreading is rather slow
  - There are some automatic components to detect and remove the infected nodes



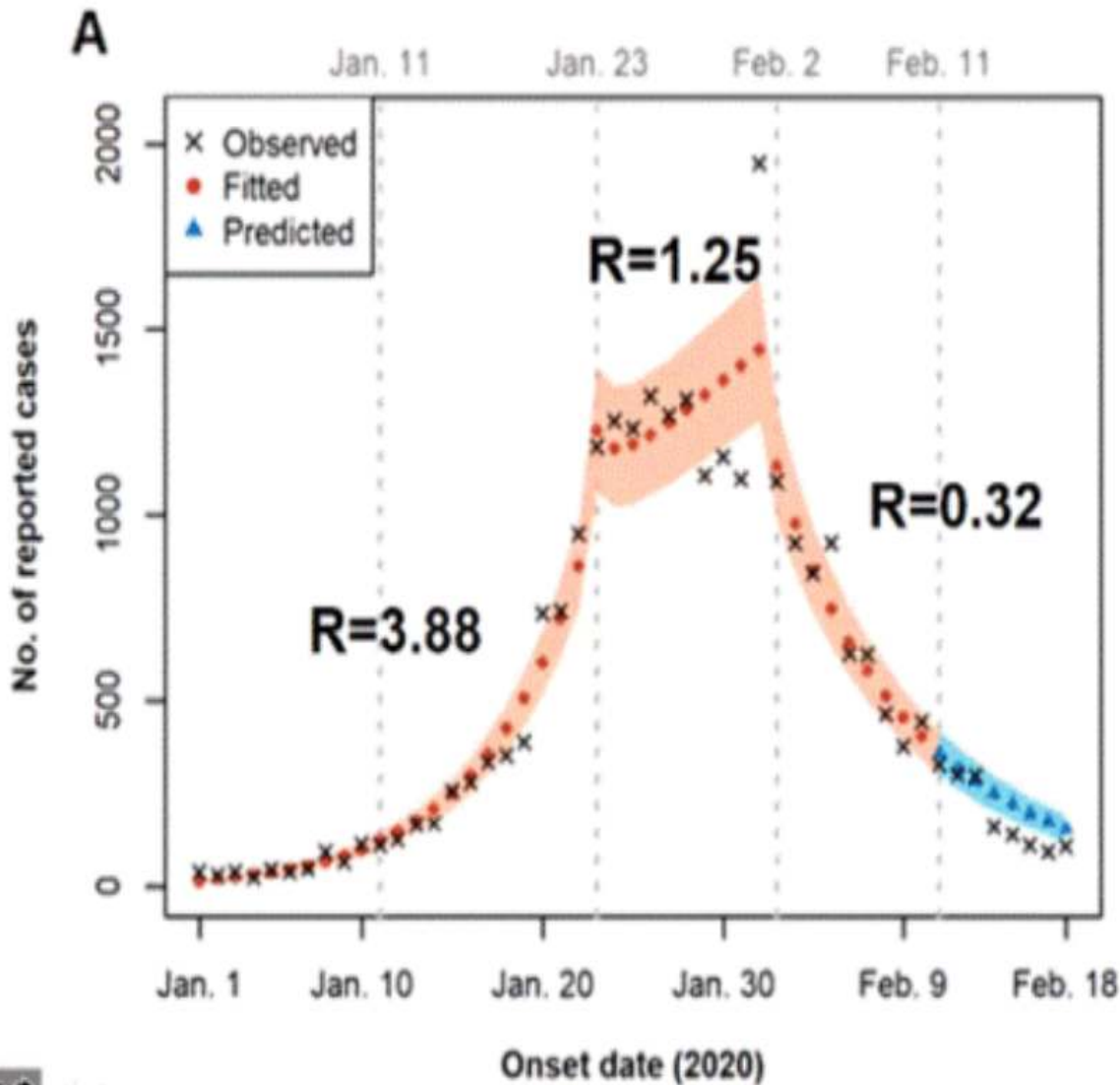
# Epidemiological threshold

---

$R_0 = \beta \sigma / \gamma$  (a TV star for more than a year :- ( )

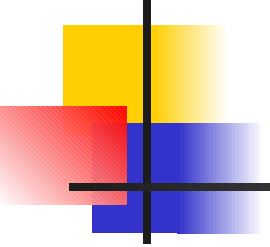
- $\sigma$  = percentage of nodes that may be infected
- It is the average number of nodes infected by an infected node
- If  $R_0 > 1$  the worm spreads, otherwise it will be defeated

# The naughty R (Covid-19)



i

Effective reproductive number  $R_t$



# Solution of the system of diff equations

---

An exact solution has been computed

$$S(t) = S(0)e^{-\xi(t)}$$

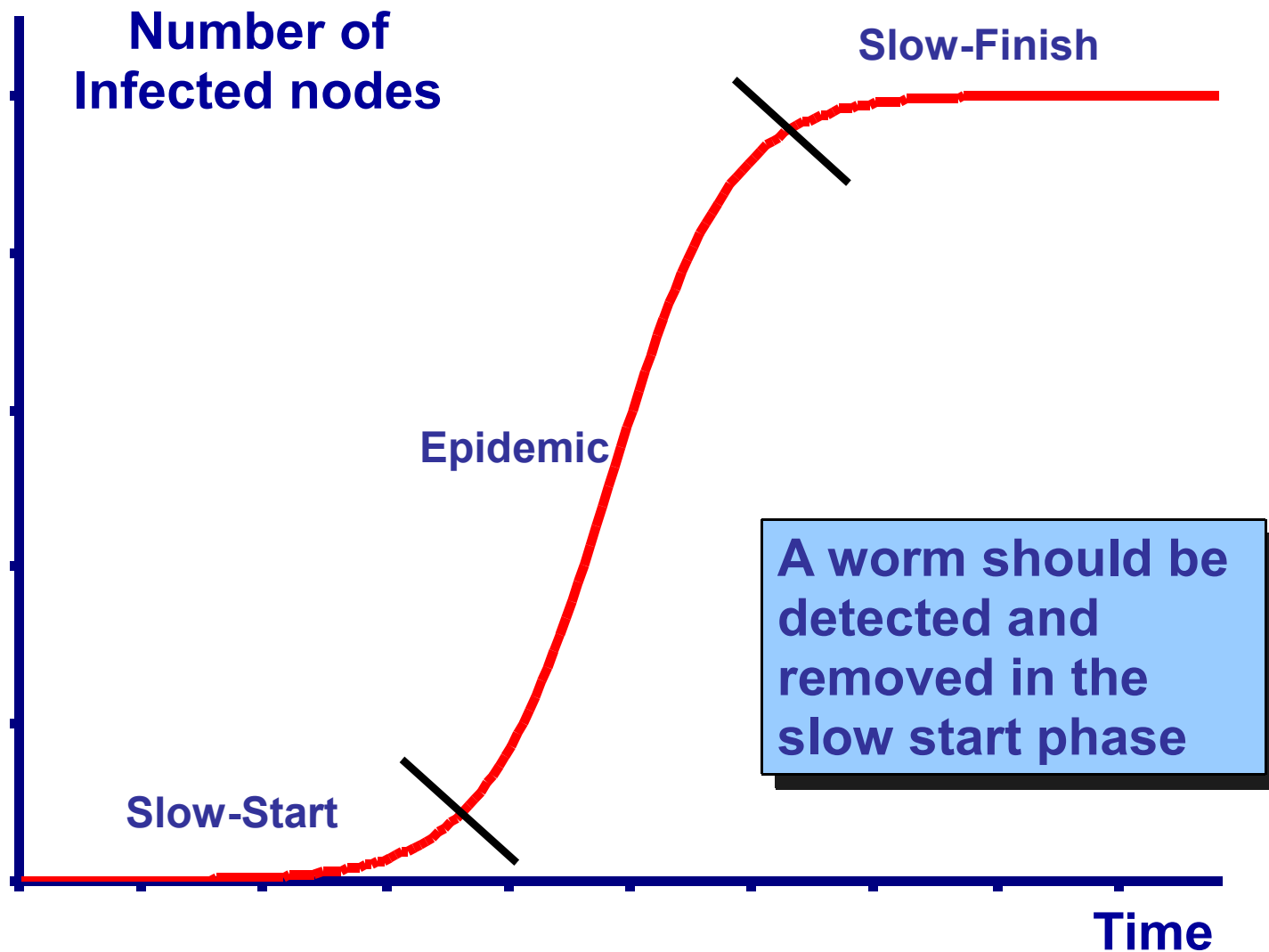
$$I(t) = N - S(t) - R(t)$$

$$R(t) = R(0) + N \frac{\gamma}{\beta} \xi(t)$$

$$\xi(t) = \frac{\beta \int_0^t I(t^*) dt^*}{N}$$



# Solution = logistic function





# A model that consider patching

---

$$dS(t)/dt = -\beta S(t)I(t) - dP(t)/dt$$

$$dR(t)/dt = \gamma I(t)$$

$$dP(t)/dt = \mu S(t)I(t) \quad \leftarrow \text{patched}$$

$$dI(t)/dt = +\beta S(t)I(t)$$

$$S(t) + I(t) + R(t) + P(t) = N$$

There are two reasons why a node is no longer susceptible

1. It has been infected
2. It has been patched

The number of patched nodes increases with numbers of susceptible and of infected ones



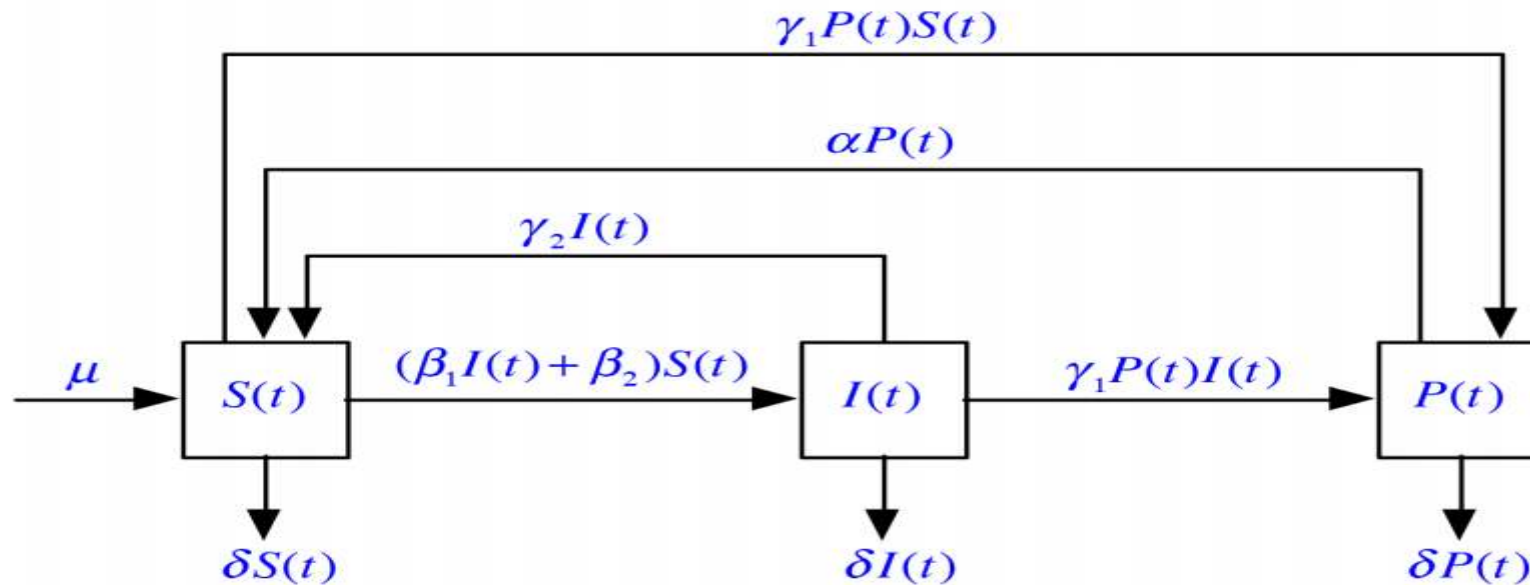


# A more complex model

---

- (H1) The nodes outside the network are all susceptible.
- (H2) The nodes outside the network are connected to the network at constant rate  $\mu > 0$ .
- (H3) Every node in the network is disconnected from the network with constant probability per unit time  $\delta > 0$ . Clearly, we have  $\frac{\mu}{\delta} = N$ .
- (H4) Due to connections with infected nodes, at time  $t$  every susceptible node in the network gets infected with probability per unit time  $\beta_1 I(t)$ , where  $\beta_1 > 0$  is a constant. This hypothesis captures the distributed nature of virus propagation.
- (H5) Due to existence of infected removable storage media, every susceptible node in the network gets infected with constant probability per unit time  $\beta_2 > 0$ .
- (H6) Due to connections with patched nodes, at time  $t$  every susceptible or infected node in the network acquires the newest patch with probability per unit time  $\gamma_1 P(t)$ , where  $\gamma_1 > 0$  is a constant. This hypothesis captures the distributed nature of patch dissemination.
- (H7) Due to system reinstallation, every infected node in the internet becomes susceptible with constant probability per unit time  $\gamma_2 > 0$ .
- (H8) Due to patch invalidation, every patched node in the network becomes susceptible with constant probability per unit time  $\alpha > 0$ .

# A more complex model - II



$$\begin{cases} \frac{dS(t)}{dt} = \mu - \beta_1 S(t)I(t) - \beta_2 S(t) - \gamma_1 S(t)P(t) + \gamma_2 I(t) + \alpha P(t) - \delta S(t), \\ \frac{dI(t)}{dt} = \beta_1 S(t)I(t) + \beta_2 S(t) - \gamma_1 I(t)P(t) - \gamma_2 I(t) - \delta I(t), \\ \frac{dP(t)}{dt} = \gamma_1 S(t)P(t) + \gamma_1 I(t)P(t) - \alpha P(t) - \delta P(t), \end{cases}$$



# Further interesting models

---

- Let suppose that there is a partial connection among nodes (scale free, small world, ...)
- Initially some nodes are infected
- We would like to know
  - How the connection structure influences the spreading and the parameter  $R_0$
  - How patching (=vaccination) influences the spreading
  - Alternative vaccination strategies
- Alternative topologies may be be considered to discover how they influence the spreading

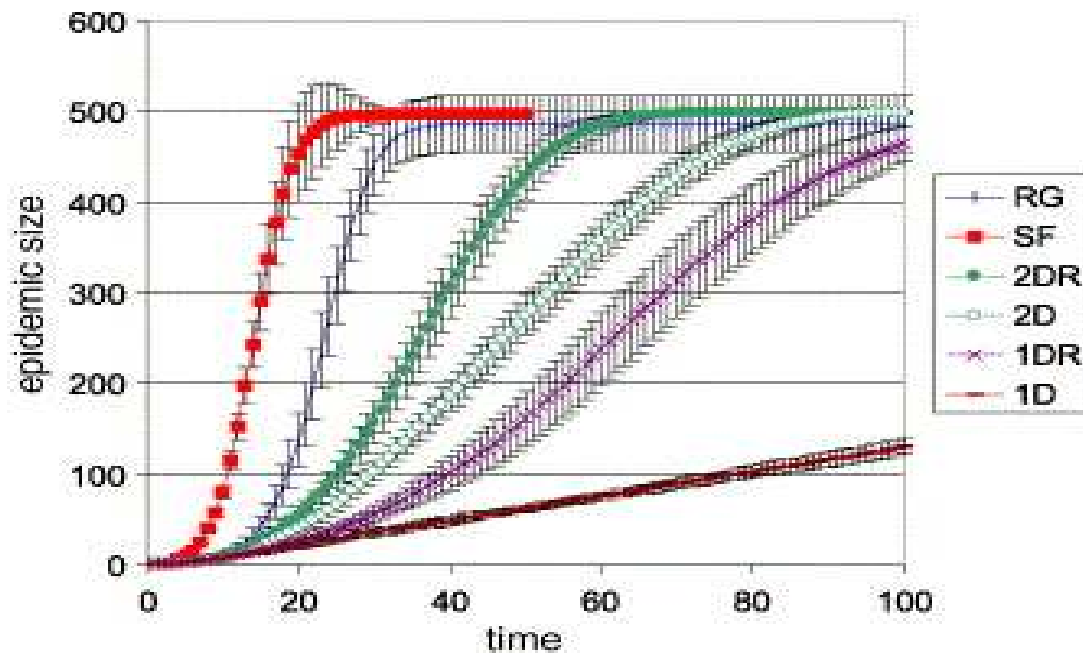


# Scale free

---

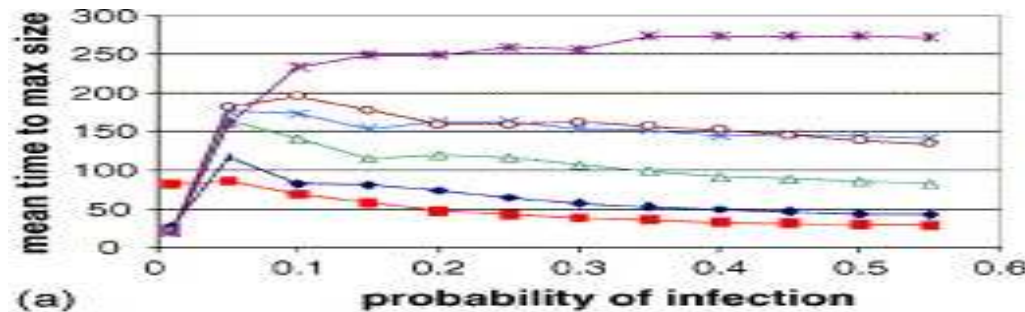
- Scale free
  - When a connection is created, nodes with a larger number of connections are preferred
  - The rich becomes richer
  - A few hubs with a huge number of connections and a huge number of nodes with few connections
- Very robust with respect to random node attacks, highly fragile with respect to targeted attacks

# Interconnection Topology

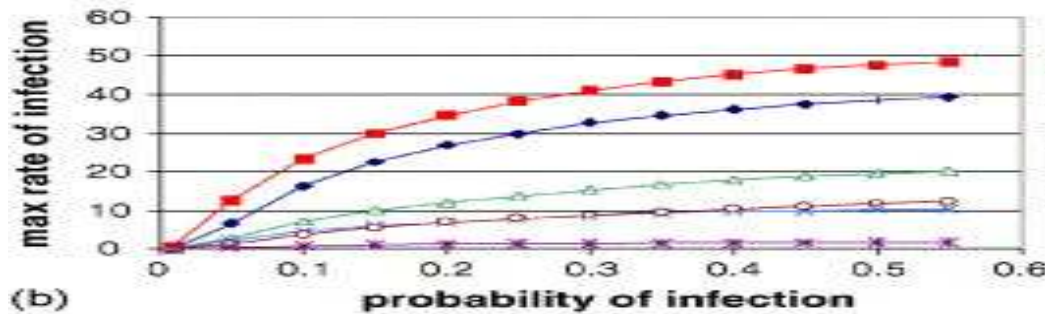


RG=random, SF=scale free, 2D= two dimensions lattice,  
1D= one dimension lattice 2DR= two dimensions lattice rewired ,  
1DR= one dimension rewired

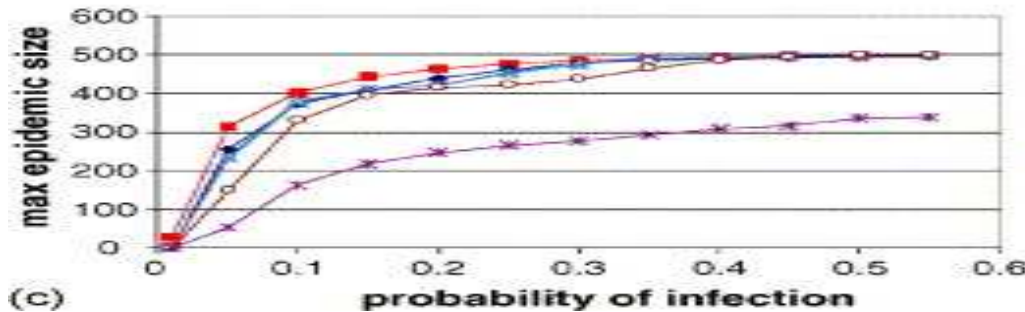
# Other interesting values



Average time to max infected



Max infection rate



Number of infected



# Computing a worm $\beta$

---

$$\beta = \frac{C}{N} \times \frac{\alpha}{\tau}$$

*Alpha*

*Tau*

**C = 1** (a random machine is selected)

**C = N** (an infected machine is always selected)

**N = 2<sup>32</sup>** (size of IP address)

**Alpha** = number of nodes tested in parallel

**Tau** = average time for testing a machine



# Code red

---

***Tau* = 19 seconds**

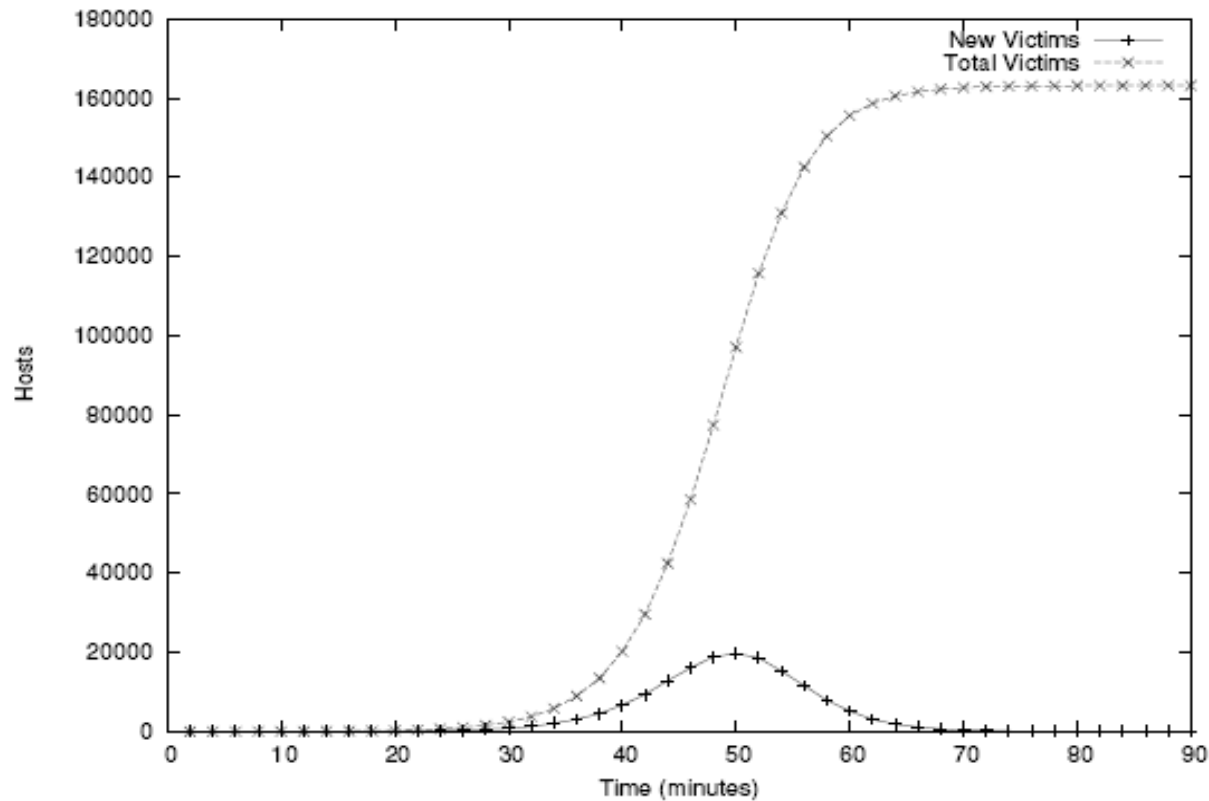
***Alpha* = 100**

$$\beta = \frac{1}{2^{32}} \times \frac{100}{19} = 1.23 \times 10^{-9}$$

Good approximation

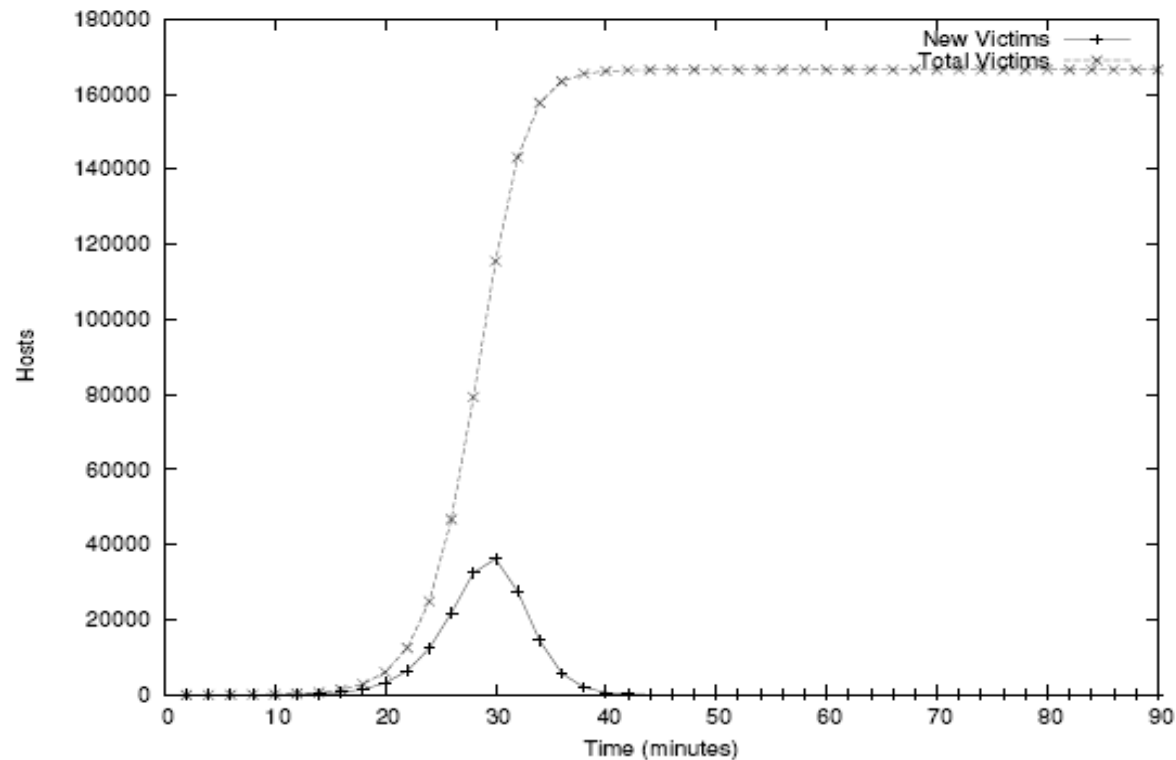


# Spreading - I



10 parallel threads and conflicts on nodes to be infected are neglected

# Spreading - II



Optimization of the time out to detect that no node exists with the random IP address that has been generated

# Spreading - III

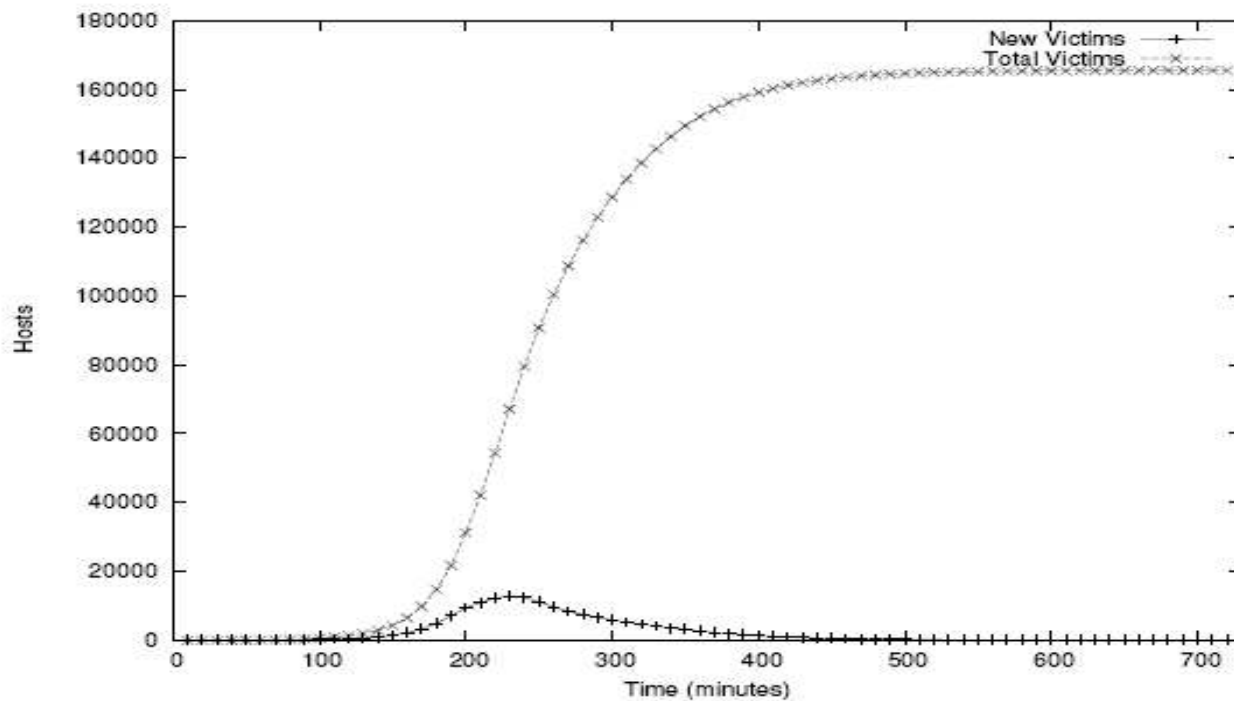
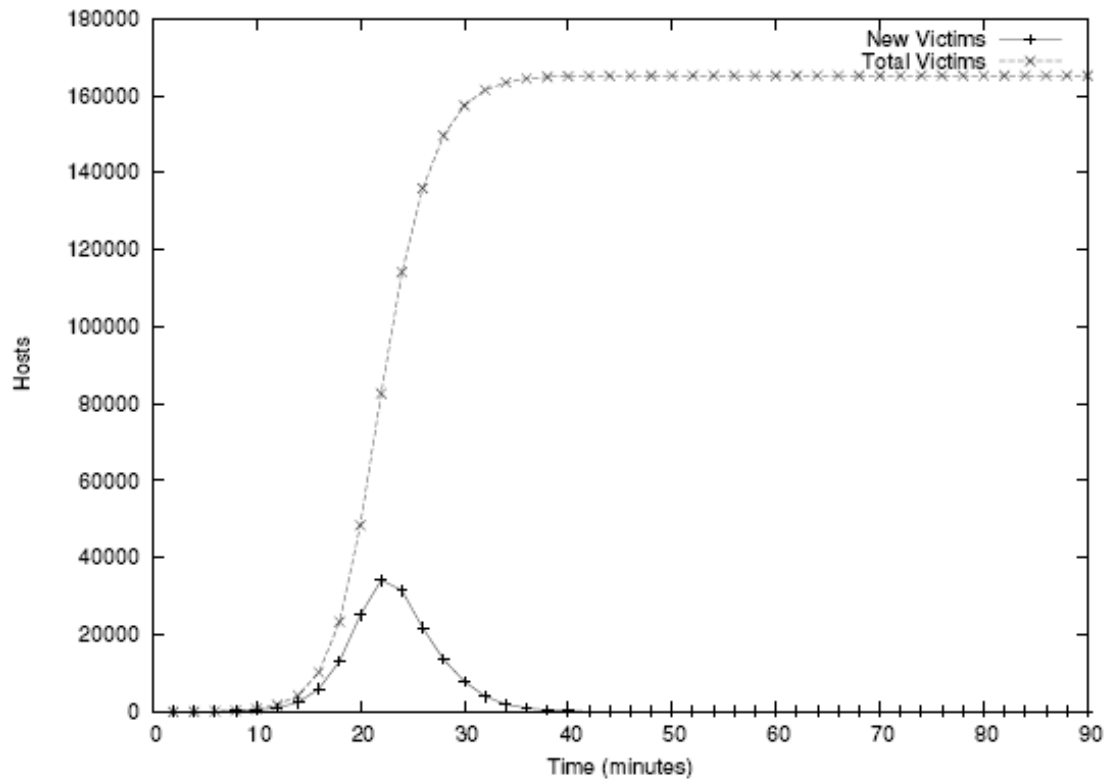


Figure 7: Local preference propagation

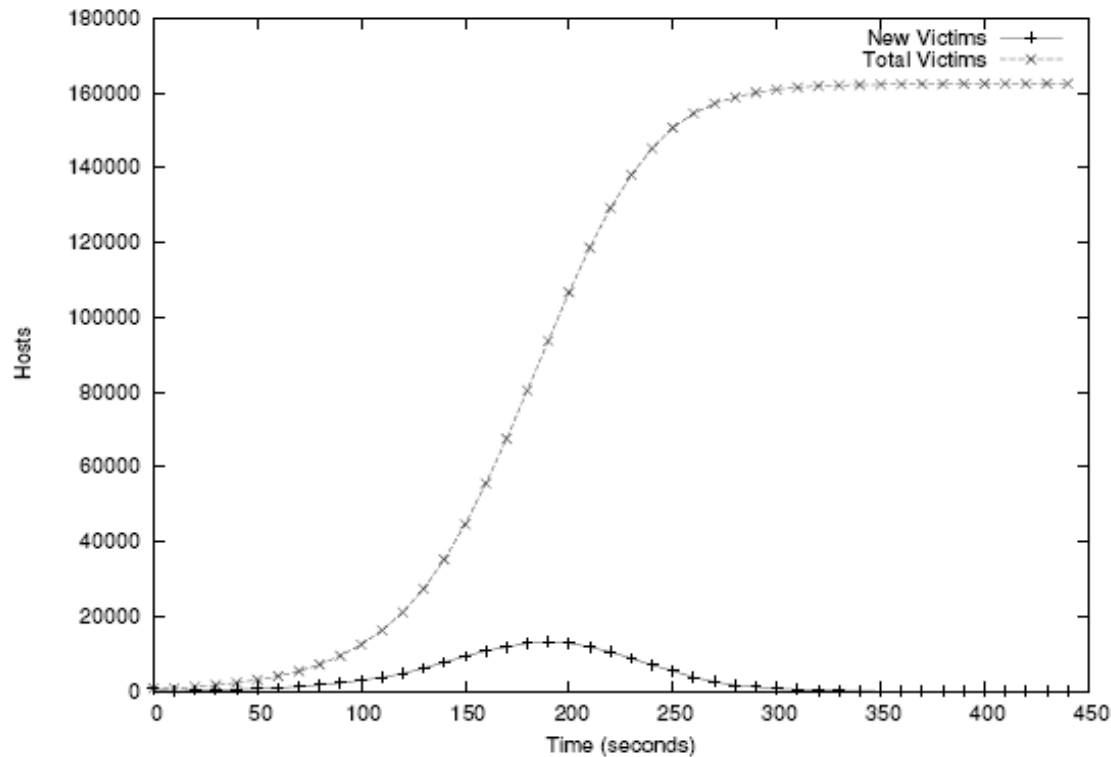
Local bias in the generation

# Spreading - IV



Local bias + multithreading + short timeouts

# Spreading - V



- + prescan to find better subspaces to generate IP addresses
- + with a large number of susceptible nodes
- + Infected nodes are remembered and neglected
- + multithread

# Local vs global

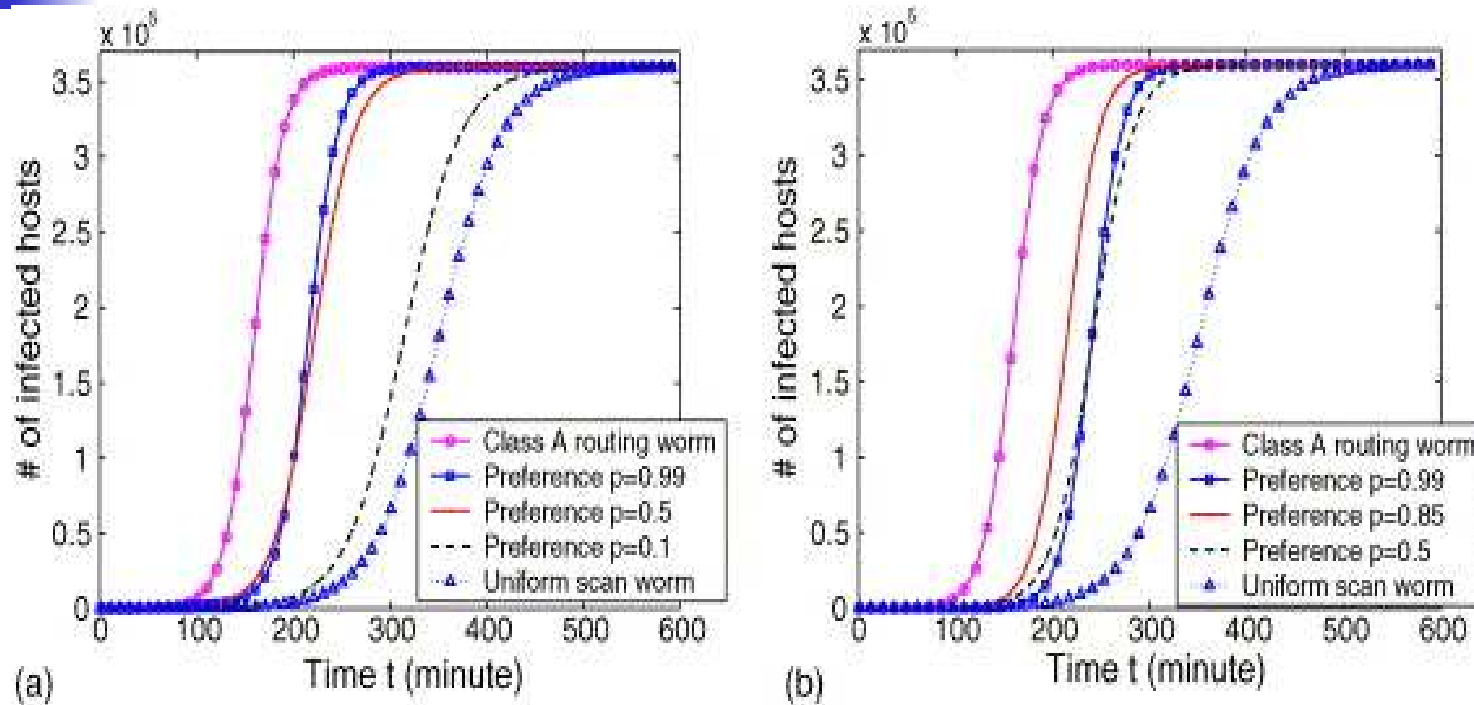
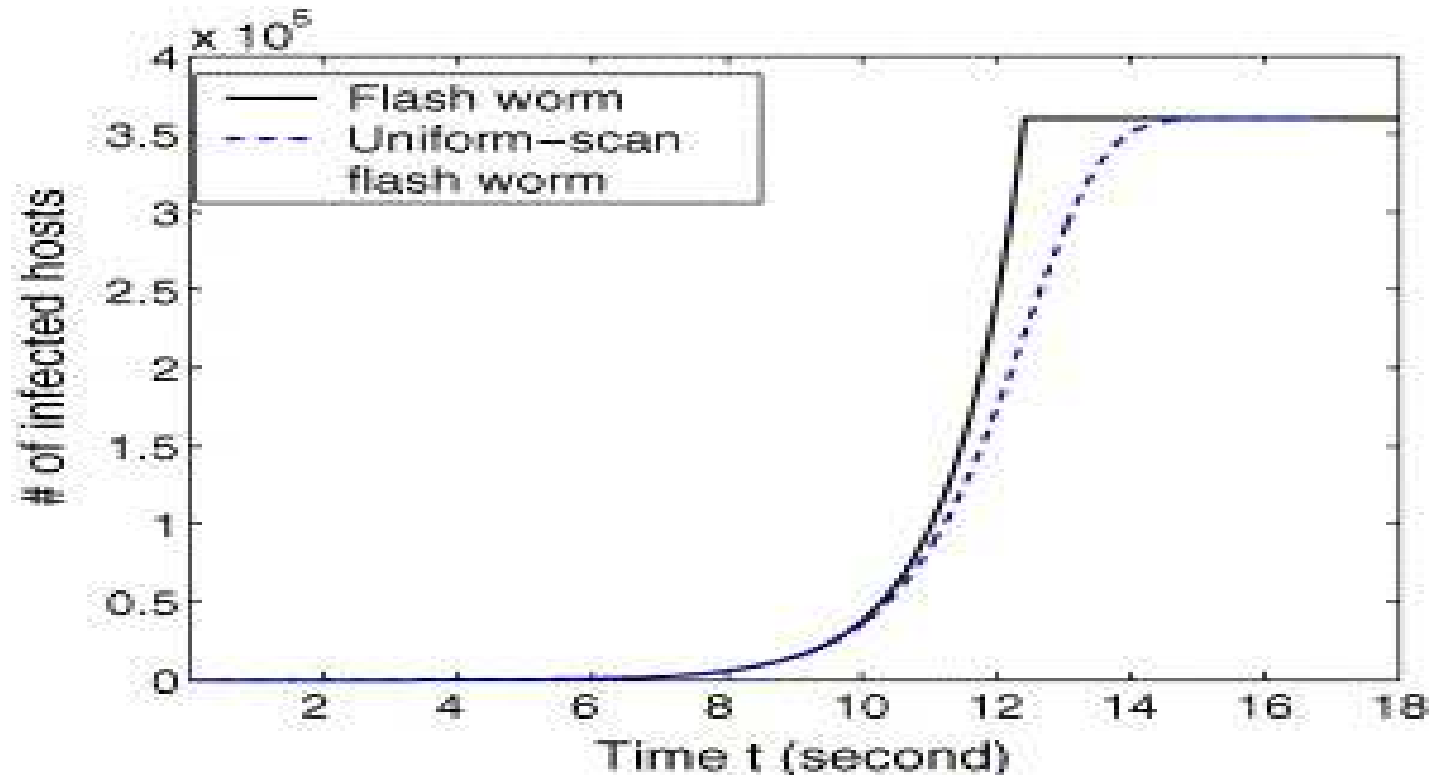


Fig. 5. Comparison of Code Red, a /8 routing worm, a local preference worm with different preference probabilities  $p$ .  
(a) Local preference scan on “/8” network level (K=256, m=116).  
(b) Local preference scan on “/16” network level (K=65,536, m=29,696).

# Extreme optimization



The time scale has changed



# Which address space?

---

- Some worms consider IP addresses
  - ⇒ Any node can infect any other nodes
  - ⇒ The addresses that are generated depend upon the adopted function and not upon the interconnection
  - ⇒ Highly effective but high error rate
- Some worms consider logical addresses, ie email addresses
  - ⇒ A node can infect only nodes it already knows
  - ⇒ The interconnection structure that has to be considered is the logical one





# Trojan horse

---

- A program that has a different goal from the expected one
- Its main goal is to implement a backdoor to enable illegal accesses to the system (persistence in the MITRE Att&ck Matrix)
- Governmental to acquire information and defeat encryption
- Malware



# Trojan horse defence (wikipedia)

---

- This defense (SODDI, some other dude did it) typically involves defendant denial of responsibility for
  - (i) the presence of cyber contraband on the defendant's computer system;
  - (ii) commission of a cybercrime via the defendant's computer, on the basis that a malware or on some other perpetrator using such malware, was responsible for the offence in question.
- A modified use of the defense involves a defendant charged with a non-cyber crime admitting that whilst technically speaking the defendant may be responsible for the commission of the offence, he or she lacked the necessary criminal intent or knowledge on account of malware involvement.



# “Reflections on Trusting Trust”

---

## Ken Thompson’s 1983 Turing Award lecture

1. Added a backdoor-opening Trojan to login program
2. Anyone looking at source code would see this, so changed the compiler to add backdoor at compile-time
3. Anyone looking at compiler source code would see this, so changed the compiler to recognize when it’s compiling a new compiler and to insert Trojan into it

“ The moral is obvious. You can’t trust code you did not totally create yourself. (Especially code from companies that employ people like me).”



# Rootkits

---

**Rootkit** is a set of trojan system binaries

Main characteristic: stealthiness

- Create a hidden directory
  - /dev/.lib, /usr/src/.poop and similar
  - Often use invisible characters in directory name (why?)
- Install hacked binaries for system programs such as netstat, ps, ls, du, login

Can't detect attacker's processes, files or network connections by running standard UNIX commands!

- Modified binaries have same checksum as originals
  - What should be used instead of checksum?



# Function Hooking

---

Rootkit may “re-route” a legitimate system function to the address of malicious code

## Pointer hooking

- Modify the pointer in OS’s Global Offset Table, where function addresses are stored

## “Detour” or “inline” hooking

- Insert a jump in first few bytes of a legitimate function
- This requires subverting memory protection

Modifications may be detectable by a clever rootkit detector



# Kernel Rootkits

---

Get loaded into OS kernel as an external module

- For example, via compromised device driver or a badly implemented “digital rights” module (e.g., Sony XCP)

Replace addresses in system call table, idt

If kernel modules disabled, directly patch kernel memory through /dev/kmem (SuckKIT rootkit)

Inject malicious code into a running process via `PTRACE_ATTACH` and `PTRACE_DETACH`

- Security and antivirus software are often the first injection targets



# Mebroot (Windows)

---

- Replaces the host's Master Boot Record (MBR)
  - First physical sector of the hard drive
  - Launches before Windows loads
- No registry changes, very little hooking
- Stores data in physical sectors, not files
  - Invisible through the normal OS interface
- Uses its own version of network driver API to send and receive packets
  - Invisible to “personal firewall” in Windows
- Used in Torpig botnet



# Detecting Rootkit's Presence

---

## Sad way to find out

- Run out of physical disk space because of sniffer logs
- Logs are invisible because du and ls have been hacked

## Manual confirmation

- Reinstall clean ps and see what processes are running

## Automatic detection

- Rootkit does not alter the data structures normally used by netstat, ps, ls, du, ifconfig
- Host-based intrusion detection can find rootkit files
  - ...assuming an updated version of rootkit did not disable the intrusion detection system!



# Remote Administration Tools



---

## Legitimate tools are often abused

- Citrix MetaFrame, WinVNC, PC Anywhere
  - Complete remote control over the machine
  - Easily found by port scan (e.g., port 1494 – Citrix)
- Bad installations, crackable password authentication
  - “The Art of Intrusion” – hijacking remote admin tools to break into a cash transfer company, a bank’s IBM AS/400 server

## Semi-legitimate tools

- Back Orifice, NetBus
- Rootkit-like behavior: hide themselves, log keystrokes
- Considered malicious by anti-virus software



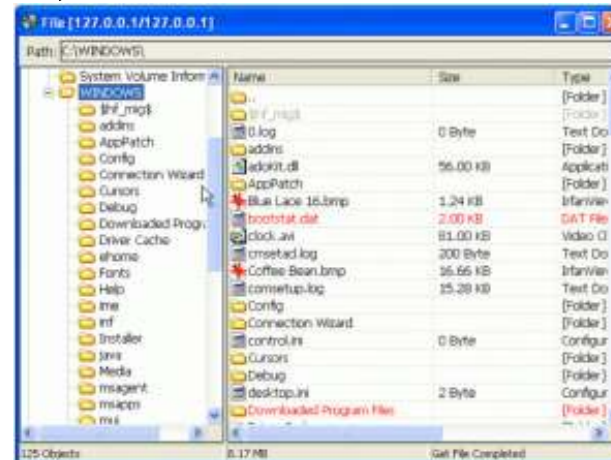
# Communicating Via Backdoors

---

- All sorts of standard and non-standard tunnels
- SSH daemons on a high port
  - Communication encrypted  $\Rightarrow$  hard to recognize for a network-based intrusion detector
  - Hide SSH activity from the host by patching netstat
- UDP listeners
- Passively sniffing the network for master's commands

# RAT Capabilities

- “Dropper” program installs RAT DLL, launches it as persistent Windows service, deletes itself
- RAT notifies specified C&C server, waits for instructions
- Attacker at C&C server has full control of the infected machine, can view files, desktop, manipulate registry, launch command shell





# Hybrid

---

- Most malware current integrates all the previous behavior
- Software with an opportunistic approach to spread to other nodes
  - Usb
  - Share
  - Mail
  - Attack
  - .....



# Autonomous Hybrid

---

- They can transmit themselves to other nodes without exploiting the node resources
- Even if the node does not exchange email, it can
  - Transmit email from the node
  - Hide in the mail

# Attack Analysis: the takeaways



---

- Complex vs elementary attacks
- The attack pyramid (target vs mass attacks)
- Attack surface
- Attack graph strengths and weakness
- Countermeasures as graph cuts
- Automated Attack
- Worms and spreading
- MITRE ATT&CK matrix