CSc 466/566

Computer Security

21 : Malware

Version: 2014/11/25 14:57:39

Department of Computer Science
University of Arizona

collberg@gmail.com

Christian Collberg

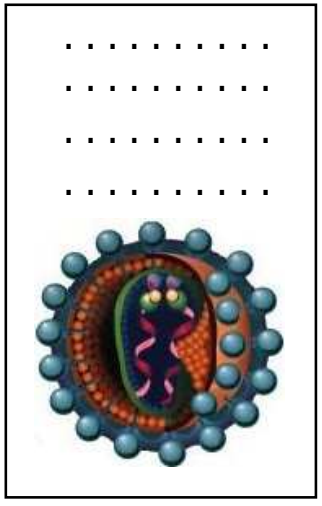
Outline

Logic Bomb

```
if (event)
  do_bad_stuff()
```

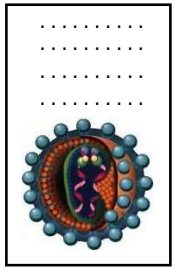
```
main(){
   play_game();
   send_spam();
}
```

report.doc



Backdoor

report.doc



Introduction

- A backdoor is a hidden command inserted by the developer.
- A logic bomb is an action that will happen when a certain condition is true.
- A virus
  1. is self-replicating;
  2. attaches itself to other files;
  3. requires user assistance to replicate.
- A trojan horse is a program that appears to perform a useful task, but, in addition, performs a malicous task.
- A worm self-propagates fully working versions of itself to other machines.

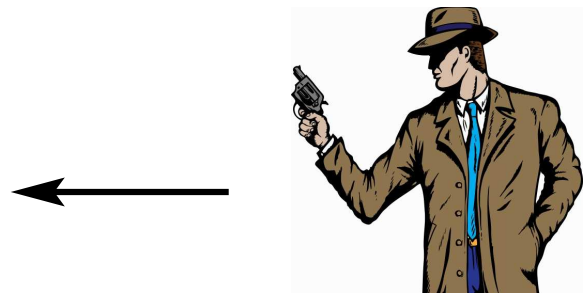
Outline

Insider Attacks

- A backdoor is a hidden command inserted by the developer.
- Backdoors can be inserted
  1. for debugging
  2. to bypass security checks in an emergency
- War Games:
  - Trailer: http://www.youtube.com/watch?v=tAcEzhQ7oqA

Backdoors

Backdoor



Malicious Backdoors

- A programmer may insert a malicious backdoor to gain access later.
- A deliberate vulnerability (buffer overflow, etc.) can be inserted, allowing the programmer easy access.

Logic Bombs

Logic Bomb

```
if (event)
  do_bad_stuff()
```

Logic Bombs

- A programmer inserts an action that will happen when a certain condition is true:

```
if (Bob is not on the payroll anymore)
    crash_system();
```

- A logic bomb can be combined with a backdoor that allows the programmer to disable it:

```
boolean disabled=false;

if (today=="April 1" && not disabled)
    delete_all_backups();

void backdoor() {
    disabled = true;
}
```

Fannie Mae Logic Bomb

Source: http://www.wired.com/threatlevel/2009/01/fannie/
Unix engineer Rajendrasinh Babubha Makwana, 35, was indicted Tuesday in federal court in Maryland on a single count of computer sabotage for allegedly writing and planting the malicious code on Oct. 24, the day he was fired from his job. The malware had been set to detonate at 9:00 a.m. on Jan. 31, but was instead discovered by another engineer five days after it was planted, according to court records.
On the afternoon of Oct. 24, he was told he was being fired because of a scripting error hed made earlier in the month, but he was allowed to work through the end of the day.

Fannie Mae Logic Bomb...

Five days later, another Unix engineer at the data center discovered the malicious code hidden inside a legitimate script that ran automatically every morning at 9:00 a.m. Had it not been found, the FBI says the code would have executed a series of other scripts designed to block the companys monitoring system, disable access to the server on which it was running, then systematically wipe out all 4,000 Fannie Mae servers, overwriting all their data with zeroes. "This would also destroy the backup software of the servers making the restoration of data more difficult because new operating systems would have to be installed on all servers before any restoration could begin," wrote Nye.

Fannie Mae Logic Bomb...

As a final measure, the logic bomb would have powered off the servers.
The trigger code was hidden at the end of the legitimate program, separated by a page of blank lines. Logs showed that Makwana had logged onto the server on which the logic bomb was created in his final hours on the job.

Fannie Mae Logic Bomb...

Source:

http://www.thetechherald.com/articles/Fannie-Mae-logic-bomb-creator-found-guilty/11557

Facts in the case prove that Fannie Mae had strong logging processes. The initial affidavit says Makwana was singled out as the person who wrote the malicious script because logs revealed his username was the last to access the system where the logic bomb was located. In addition, he was the last to access the malicious file itself, and IP address assignment was used to show he did all of this from his company laptop.

Defending Against Insider Attacks

- No single points of failure — administrators shouldn't be allowed to access important systems alone.

Defending Against Insider Attacks

- No single points of failure — administrators shouldn't be allowed to access important systems alone.
- Code walk-throughs — can the programmer explain the logic bomb?

Defending Against Insider Attacks

- No single points of failure — administrators shouldn't be allowed to access important systems alone.
- Code walk-throughs — can the programmer explain the logic bomb?
- Use software engineering tools.

Defending Against Insider Attacks

- No single points of failure — administrators shouldn't be allowed to access important systems alone.
- Code walk-throughs — can the programmer explain the logic bomb?
- Use software engineering tools.
- Use least privilege principle — no one should have more privileges than they need to do their job.

Defending Against Insider Attacks

- No single points of failure — administrators shouldn't be allowed to access important systems alone.
- Code walk-throughs — can the programmer explain the logic bomb?
- Use software engineering tools.
- Use least privilege principle — no one should have more privileges than they need to do their job.
- Physically secure systems.

Defending Against Insider Attacks

- No single points of failure — administrators shouldn't be allowed to access important systems alone.
- Code walk-throughs — can the programmer explain the logic bomb?
- Use software engineering tools.
- Use least privilege principle — no one should have more privileges than they need to do their job.
- Physically secure systems.
- Monitor employee behavior — watch out for disgruntled administrators.

Defending Against Insider Attacks

- No single points of failure — administrators shouldn't be allowed to access important systems alone.
- Code walk-throughs — can the programmer explain the logic bomb?
- Use software engineering tools.
- Use least privilege principle — no one should have more privileges than they need to do their job.
- Physically secure systems.
- Monitor employee behavior — watch out for disgruntled administrators.
- Limit new software installations.

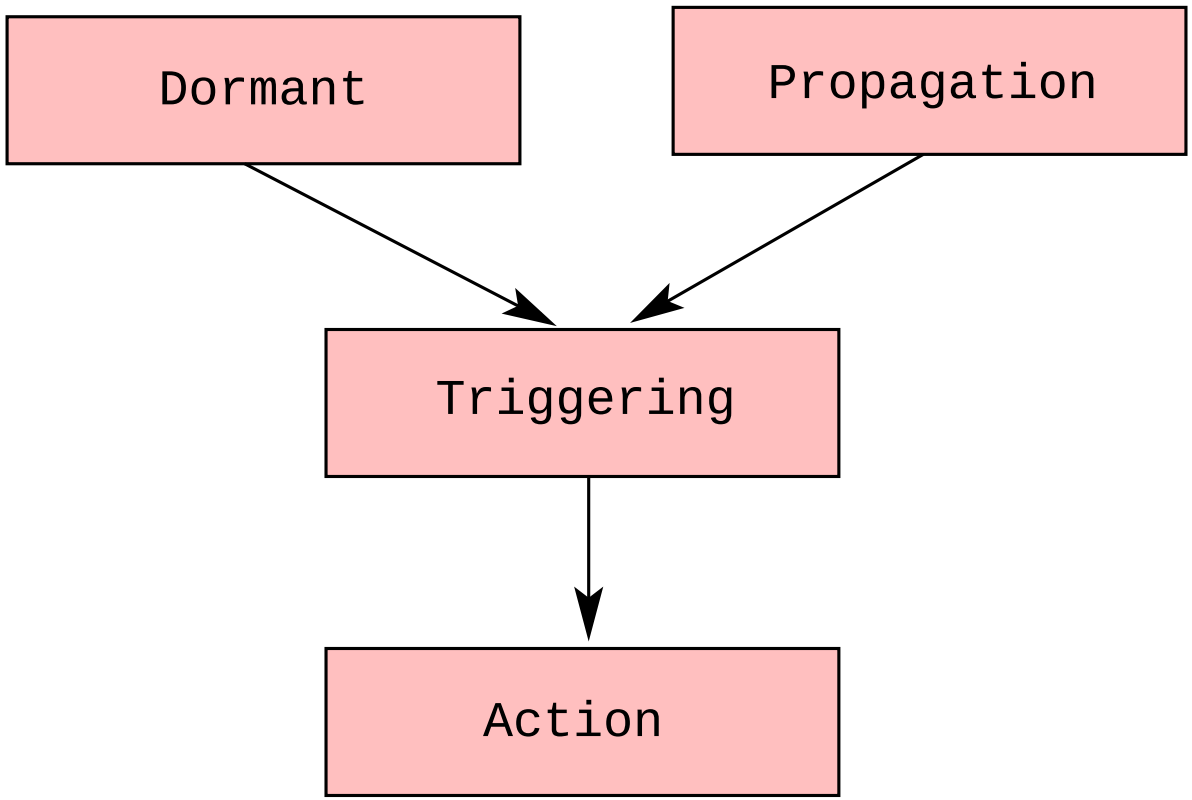
Outline

Computer Viruses

Computer Viruses

- Viruses
  1. are self-replicating;
  2. attach themselves to other files;
  3. require user assistance to replicate.

Computer Viruses: Phases



Computer Viruses: Phases...

- Dormant — lay low, avoid detection.
- Propagation — infect new files and systems.
- Triggering — decide to move to action phase
- Action — execute malicious actions, the payload.

Virus Types

- Program/File virus:

Virus Types

- Program/File virus:
  - Attaches to: program object code.

Virus Types

- Program/File virus:
  - Attaches to: program object code.
  - Run when: program executes.

Virus Types

- Program/File virus:
  - Attaches to: program object code.
  - Run when: program executes.
  - Propagates by: program sharing.

Virus Types

- Program/File virus:
  - Attaches to: program object code.
  - Run when: program executes.
  - Propagates by: program sharing.
- Doocument/Macro virus:

Virus Types

- Program/File virus:
  - Attaches to: program object code.
  - Run when: program executes.
  - Propagates by: program sharing.
- Doocument/Macro virus:
  - Attaches to: document (.doc,.pdf,. . .).

Virus Types

- Program/File virus:
  - Attaches to: program object code.
  - Run when: program executes.
  - Propagates by: program sharing.
- Doocument/Macro virus:
  - Attaches to: document (.doc,.pdf,. . .).
  - Run when: document is opened.

Virus Types

- Program/File virus:
  - Attaches to: program object code.
  - Run when: program executes.
  - Propagates by: program sharing.
- Doocument/Macro virus:
  - Attaches to: document (.doc,.pdf,. . .).
  - Run when: document is opened.
  - Propagates by: emailing documents.

Virus Types

- Program/File virus:
  - Attaches to: program object code.
  - Run when: program executes.
  - Propagates by: program sharing.
- Doocument/Macro virus:
  - Attaches to: document (.doc,.pdf,. . .).
  - Run when: document is opened.
  - Propagates by: emailing documents.
- Boot sector virus:

Virus Types

- Program/File virus:
  - Attaches to: program object code.
  - Run when: program executes.
  - Propagates by: program sharing.
- Doocument/Macro virus:
  - Attaches to: document (.doc,.pdf,. . .).
  - Run when: document is opened.
  - Propagates by: emailing documents.
- Boot sector virus:
  - Attaches to: hard drive boot sector.

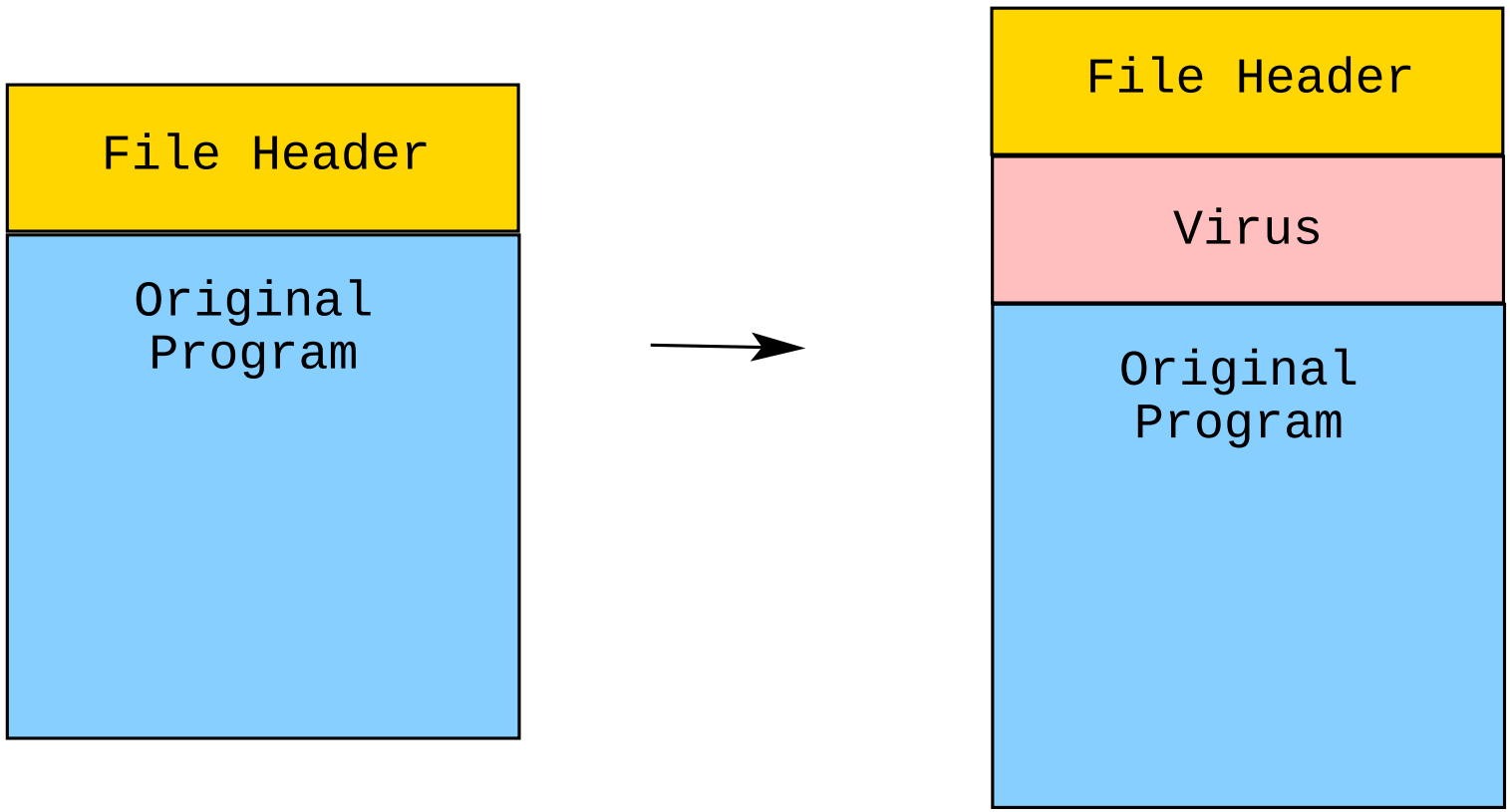
Virus Types

- Program/File virus:
  - Attaches to: program object code.
  - Run when: program executes.
  - Propagates by: program sharing.
- Doocument/Macro virus:
  - Attaches to: document (.doc,.pdf,. . .).
  - Run when: document is opened.
  - Propagates by: emailing documents.
- Boot sector virus:
  - Attaches to: hard drive boot sector.
  - Run when: computer boots.

Virus Types

- Program/File virus:
  - Attaches to: program object code.
  - Run when: program executes.
  - Propagates by: program sharing.
- Doocument/Macro virus:
  - Attaches to: document (.doc,.pdf,. . .).
  - Run when: document is opened.
  - Propagates by: emailing documents.
- Boot sector virus:
  - Attaches to: hard drive boot sector.
  - Run when: computer boots.
  - Propagates by: sharing floppy disks.

Computer Viruses: Propagation



Example: Jerusalem

- Target: DOS executables.
- Trigger: Friday the 13th.
- Payload: Deletes files.
- Propagation: Memory resident, infects executed programs.
- http://www.youtube.com/watch?v=u3k-8kJ54sg

Example: Melissa

- Target: MS Word Macro virus.
- Trigger: User opens document.
- Payload/Propagation: Emails infected documents to 50 contacts.
- http://www.youtube.com/watch?v=hu-rhz0gExg

Example: Elk Cloner

- Target: Apple II boot sector.
- Payload: Prints poem every 50th time the program is rebooted.

Elk Cloner: The program with a personality
It will get on all your disks
It will infiltrate your chips
Yes, it's Cloner!
It will stick to you like glue
It will modify RAM too
Send in the Cloner!

Example: Elk Cloner...

Source: http://en.wikipedia.org/wiki/Elk_Cloner
Elk Cloner was created in 1981 by Rich Skrenta, a 15-year-old high school student. Skrenta was already distrusted by his friends because, in sharing computer games and software, he would often alter the floppy disks to shut down or display taunting on-screen messages. Because his friends no longer trusted his disks, Skrenta thought of methods to alter floppy disks without physically touching them.
During a winter break [] Skrenta discovered how to launch the messages automatically on his Apple II computer. He developed what is now known as a boot sector virus, and began circulating it in early 1982 among high school friends and a local computer club.

Example: Sality

- Target: Windows executable files.
- Propagation: Infects other local executable files.
- Obscures its entry point.
- Downloads and executes other malware.
- Creates peer-to-peer botnet.
- Disables security software.
- Injects itself into running processes to make sure it remains on the computer.

Virus Defenses

- Signatures: Regular expressions over the virus code used to detect if files have been infected.
- Checking can be done
  1. periodically over the entire filesystem;
  2. whenever a new file is downloaded.

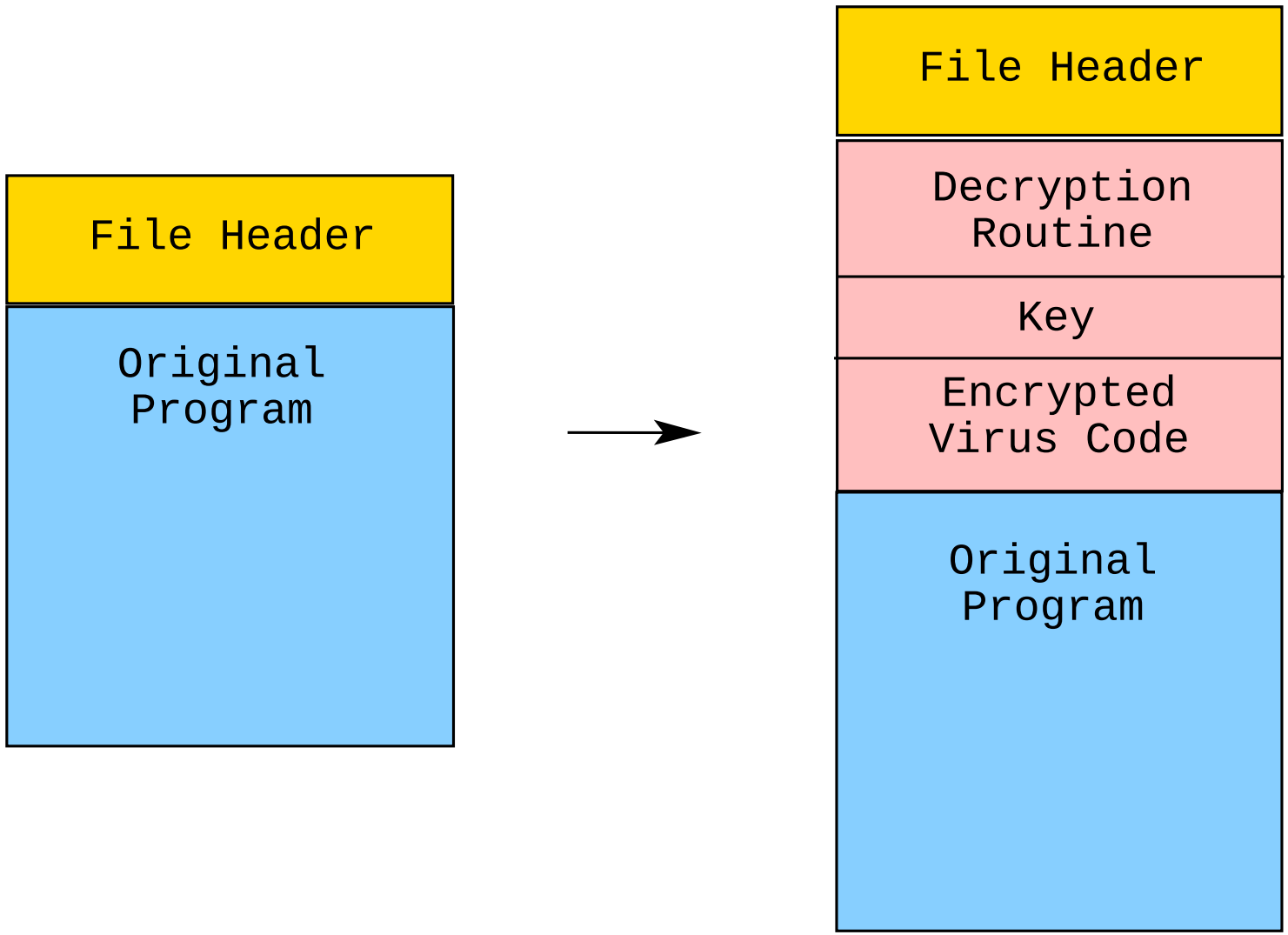
Virus Countermeasures

- Viruses need to protect themselves against detection.
- This means hiding any distringuishing features, making it hard to construct signatures.
- By encrypting its payload, the virus hides its distinguishing features.
- Encryption is often no more than xor with a constant.

Virus Countermeasures: Encryption

- By encrypting its payload, the virus hides its distinguishing features.
- The decryption routine itself, however, can be used to create a signature!

Computer Countermeasures: Encryption...



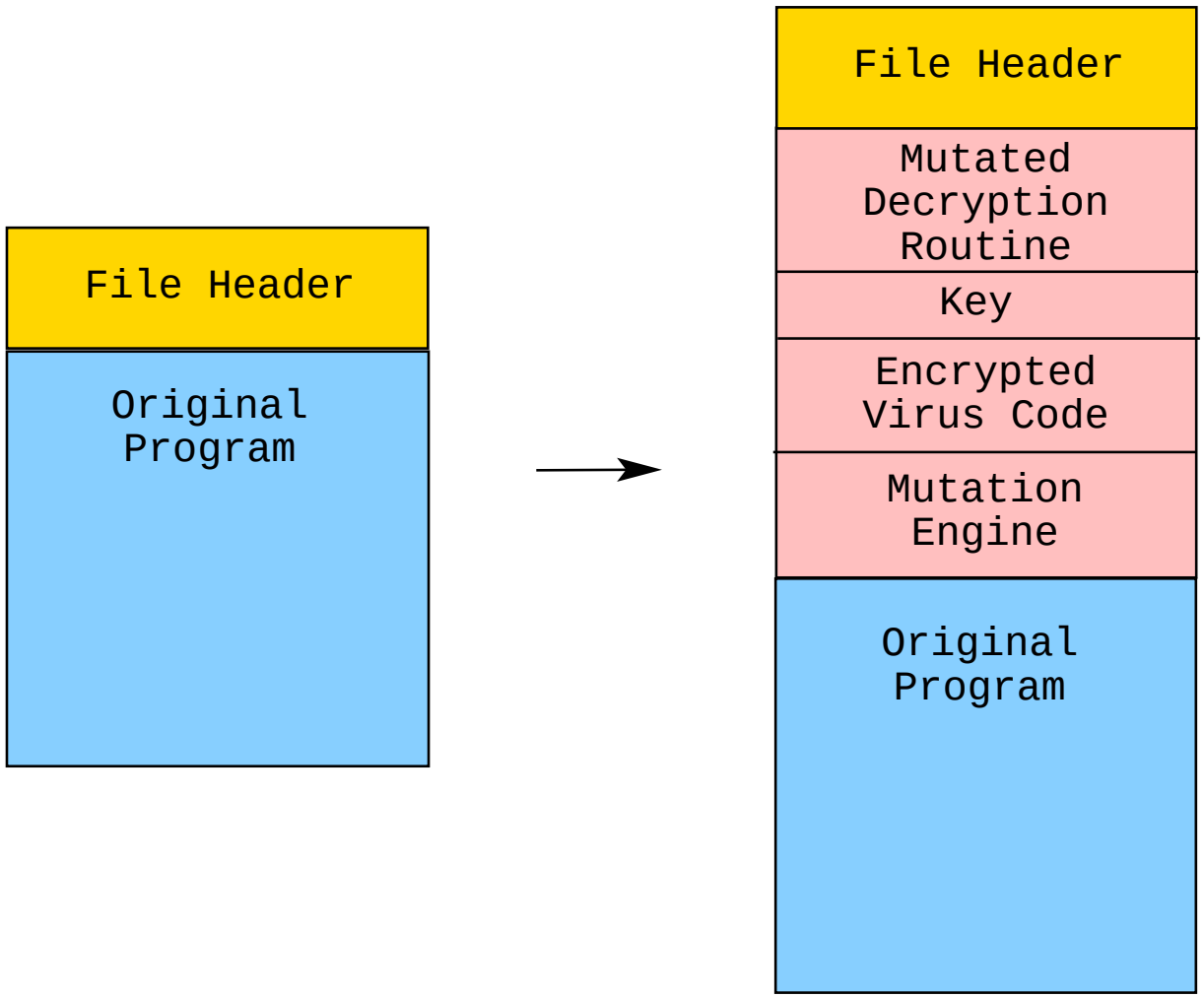
Virus Countermeasures: Polymorphism

- Each variant is encrypted with a different key.

Virus Countermeasures: Metamorphism

- To prevent easy creation of signatures for the decryption routine, metamorphic viruses will mutate the decryptor, for each infection.
- The virus contains a mutation engine which can modify the decryption code while maintaining its semantics.

Computer Countermeasures: Metamorphism...



Virus Countermeasures: Metamorphism...

- To counter metamorphism, virus detectors can run the virus in an emulator.
- The emulator gathers a trace of the execution.
- A virus signature is then constructed over the trace.
- This makes it easier to ignore garbage instructions the mutation engine may have inserted.

Outline

Trojan Horses

```
main(){
   play_game();
   send_spam();
}
```

Trojan Horses

- A trojan horse is a program that appears to perform a useful task, but, in addition, performs a malicous task.
- Example:
  - Useful task: A better `ls`.
  - Malicious task: Exfiltrate company secrets.

Trojan Horses: The AIDS Trojan

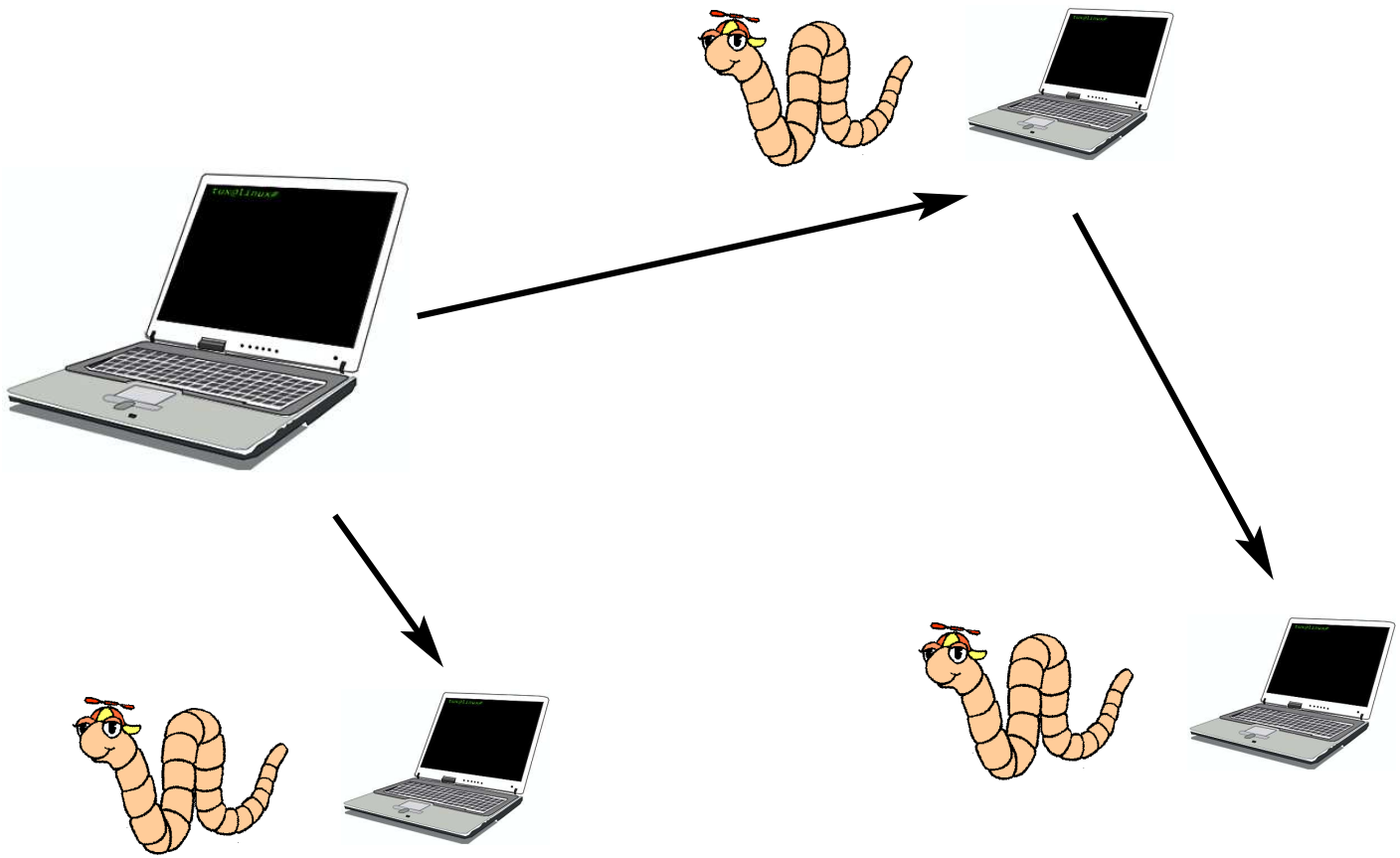
- When the boot count reaches 90, AIDS encrypts the names of all files.
- The user is asked to *renew the license*.
- To recover the files the user needs to send $189 to a P.O. box in Panama.

Trojan Horses: The AIDS Trojan

Source: http://en.wikipedia.org/wiki/AIDS_(trojan_horse)
The AIDS software end user license agreement:
If you install [this] on a microcomputer... then under terms of this license you agree to pay PC Cyborg Corporation in full for the cost of leasing these programs... In the case of your breach of this license agreement, PC Cyborg reserves the right to take legal action necessary to recover any outstanding debts payable to PC Cyborg Corporation and to use program mechanisms to ensure termination of your use... These program mechanisms will adversely affect other program applications... You are hereby advised of the most serious consequences of your failure to abide by the terms of this license agreement; your conscience may haunt you for the rest of your life... and your [PC] will stop functioning normally... You are strictly prohibited from sharing [this product] with others...

Outline

Worms



Worms

- A computer virus:
  - adds itself to other programs;
  - cannot run independently;
  - needs help from a human to spread.
- A worm propagates fully working versions of itself to other machines without
  - attaching itself to other programs;
  - human assistance.
- Worms carry malicious payloads, such as
  - installing backdoors;
  - deleting files, ...

Worm tasks

- Worm tasks:
  1. infect a victim machine by exploiting a vulnerability (buffer overflow) in a network service exported by the machine;

Worm tasks

- Worm tasks:
  1. infect a victim machine by exploiting a vulnerability (buffer overflow) in a network service exported by the machine;
  2. spread by infecting other computers reachable from the victim machine;

Worm tasks

- Worm tasks:
  1. infect a victim machine by exploiting a vulnerability (buffer overflow) in a network service exported by the machine;
  2. spread by infecting other computers reachable from the victim machine;
  3. ensure survival when the victim machine is rebooted.

Worm Propagation

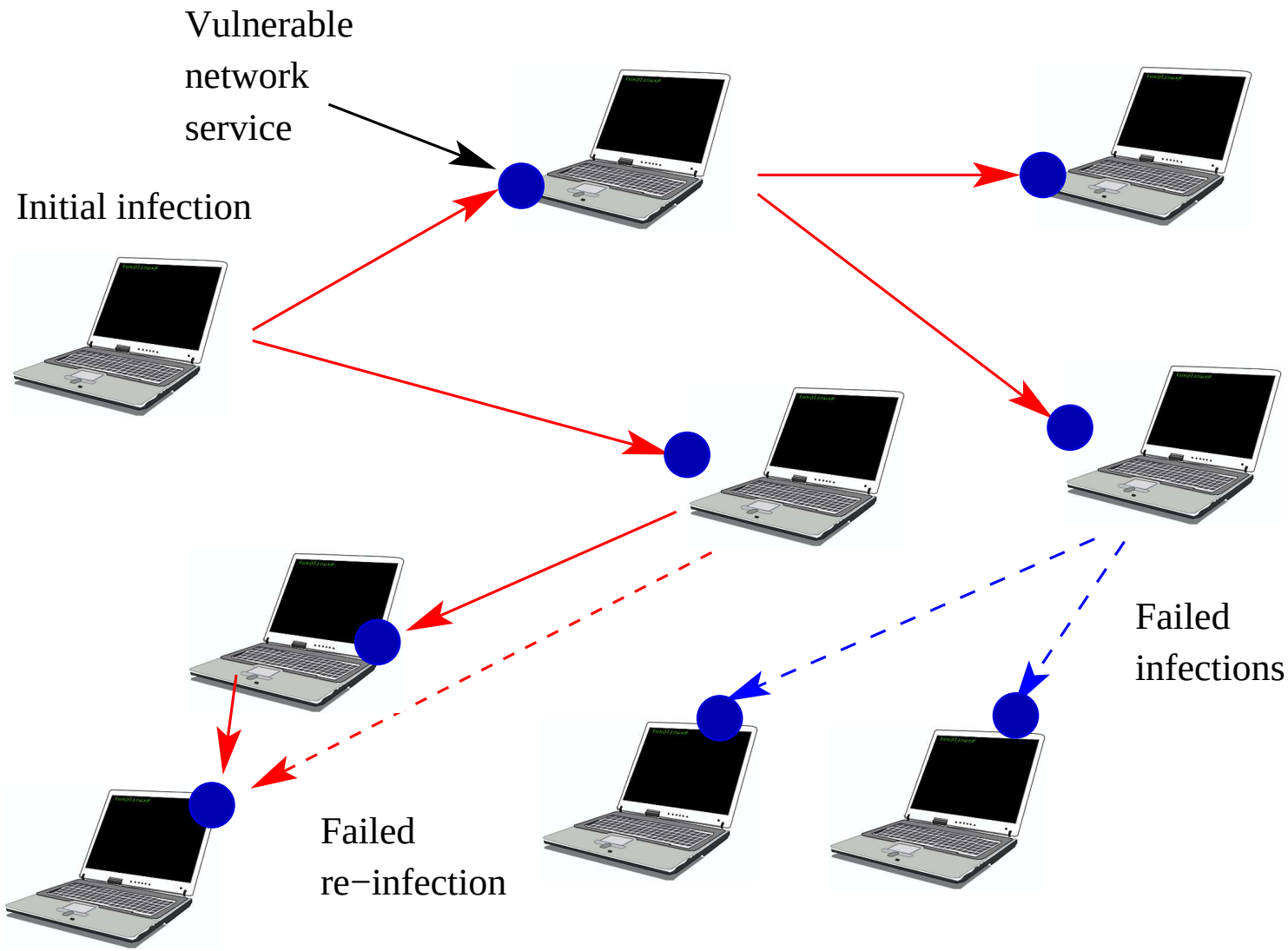
Initial infection

1. Find next target host
2. Find vulnerability
3. Propagate

Vulnerable network service

4. Ensure survival on reboot
5. Execute payload

Worm Propagation



Worm Propagation Rate

- N — total number of vulnerable hosts.
- $I(t)$ — number of infected hosts at time t.
- $S(t)$ — number of susceptible hosts at time t. A host is susceptible if it's vulnerable but not infected yet.
- β — infection rate, constant describing the speed of propagation.

$$
\begin{aligned}
I(0) &= 1 \\
S(0) &= N - 1 \\
I(t+1) &= I(t) + \beta \cdot I(t) \cdot S(t) \\
S(t+1) &= N - I(t+1)
\end{aligned}
$$

Worm Propagation Rate

- $I(t)$ — number of infected hosts at time t.
- $S(t)$ — number of susceptible hosts at time t.
- β — infection rate

$$I(t+1) \quad = \quad I(t) + \beta \cdot I(t) \cdot S(t)$$

- Example:
  1. $\beta = 0, I(t) = 1, S(t) = 4 \Rightarrow I(t+1) = 1.$
  2. $\beta = 1, I(t) = 1, S(t) = 4 \Rightarrow I(t+1) = 1 + 4 = 5.$
  3. $\beta = 0.5, I(t) = 1, S(t) = 4 \Rightarrow I(t+1) = 1 + 2 = 3.$

Worm Propagation Rate...

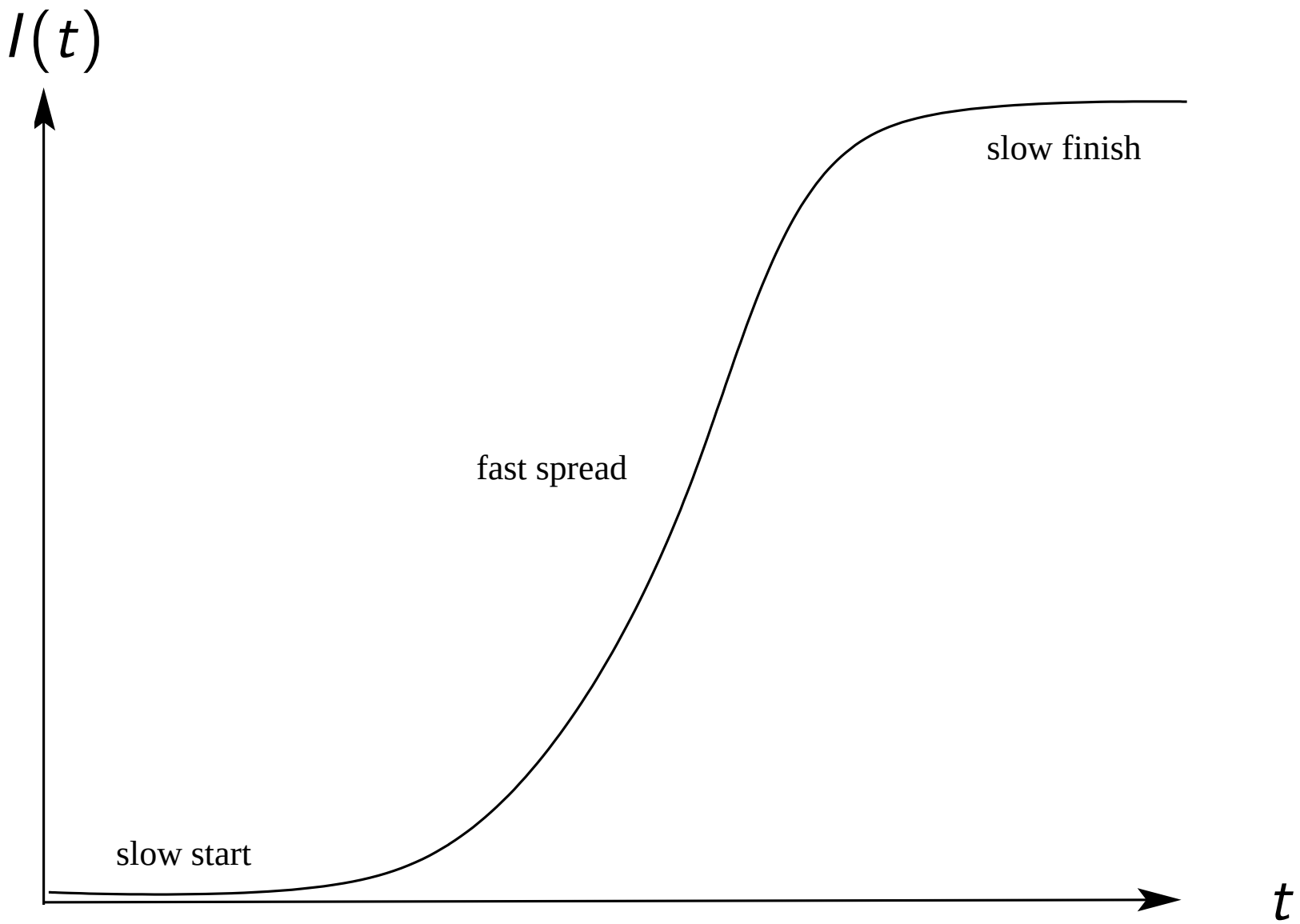
$$
\begin{aligned}
I(0) &= 1 \\
S(0) &= N - 1 \\
I(t+1) &= I(t) + \beta \cdot I(t) \cdot S(t) \\
S(t+1) &= N - I(t+1)
\end{aligned}
$$

- The number of new infections is $I(t+1) - I(t)$:

$$
\begin{aligned}
I(t+1) - I(t) &= (I(t) + \beta \cdot I(t) \cdot S(t)) - I(t) \\
&= \beta \cdot I(t) \cdot S(t)
\end{aligned}
$$

- The number of new infections is proportional to the current number of infected hosts and to the number of susceptible hosts.

Three Phases of Worm Propagation



The Morris Worm

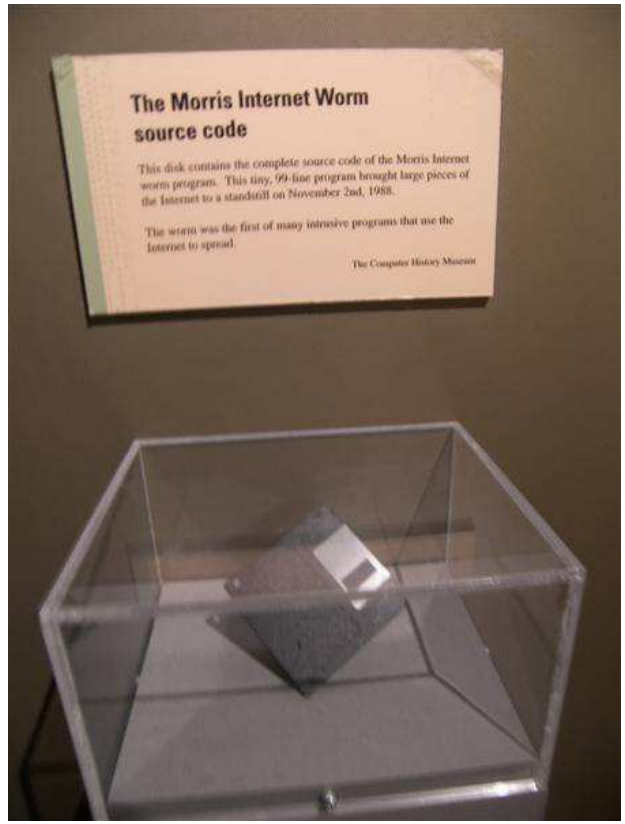
- Attacked BSD Unix derivative systems on the internet on 2 November 1988.
- Specifically targeted SUNs and VAXes.
- Computer Virus TV News Report 1988:

  https://www.youtube.com/watch?v=fj8S6Hd-5bk

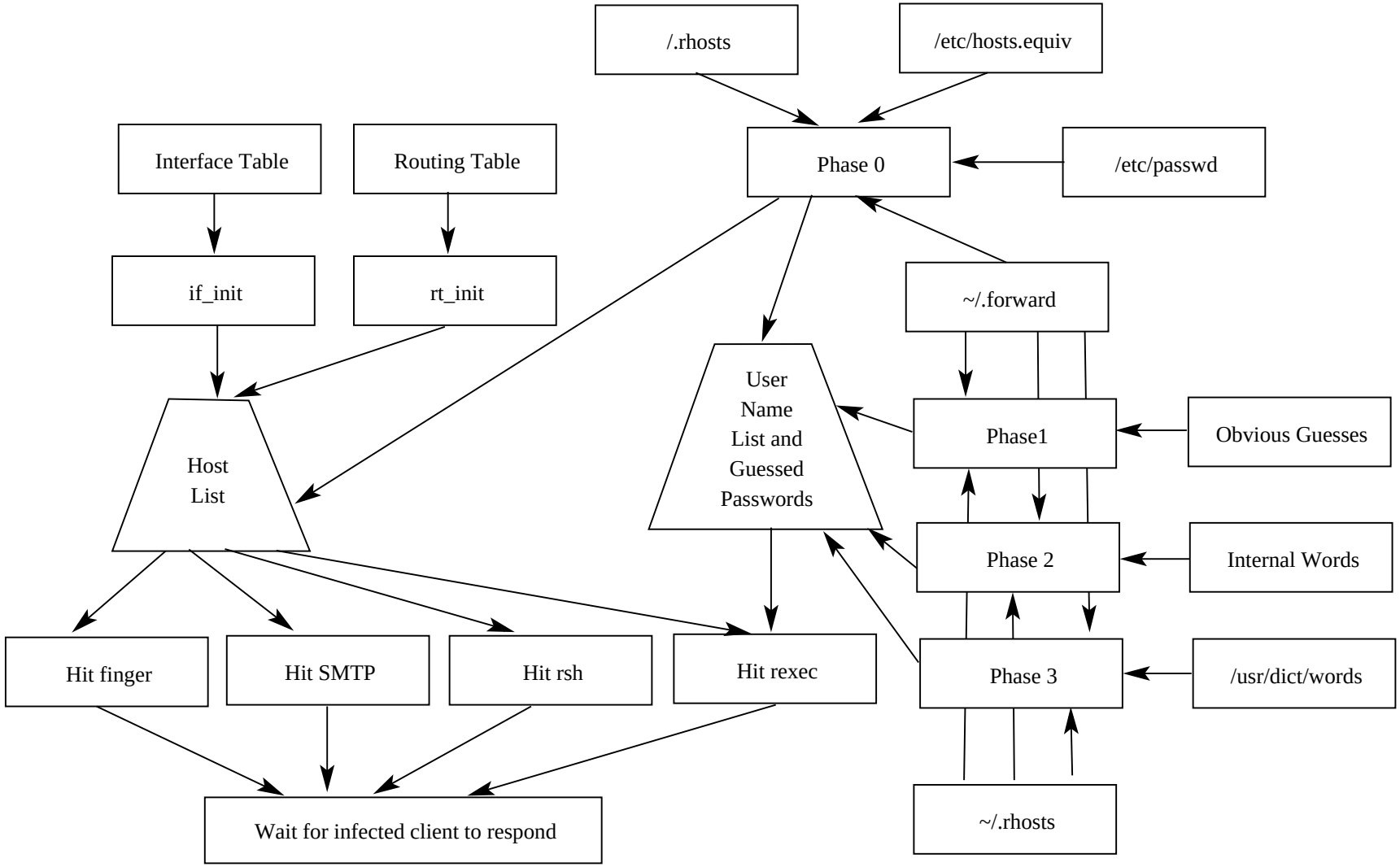
Robert Morris

- Cornell graduate student.
- Convicted under the Computer Fraud and Abuse Act: 3 years probation, 400 hours community service, $10,050.
- Tenured Professor at MIT.
- Son of Robert Morris, coauthor of UNIX, former chief scientist at the National Computer Security Center.

The Worm



- Disk containing the source code for the Morris Worm held at the Boston Museum of Science.
- Source: http://en.wikipedia.org/wiki/Morris_worm.


/.rhosts
/etc/hosts.equiv
Interface Table
Routing Table
Phase 0
/etc/passwd
if_init
rt_init
~/.forward
User
Name
List and
Guessed
Passwords
Host
List
Phase1
Obvious Guesses
Phase 2
Internal Words
Hit finger
Hit SMTP
Hit rsh
Hit rexec
Phase 3
/usr/dict/words
~/.rhosts
Wait for infected client to respond

Which Vulnerabilities to Exploit?

- SUN: `sendmail`
- VAX: `finger`
- Remote execution system using `rsh` or `rexec`.

Exploting weak passwords

- Accounts with obvious passwords
- Accounts with passwords in a 432 word dictionary
- Accounts with passwords in `/usr/dict/words`
- Accounts which trusted other machines via the `.rhosts` mechanism

Which machines to spread to?

- SUNs and VAXes
- Machines in `/etc/hosts.equiv`
- Machines in `.rhosts`
- Machines in `.forward` files
- Network gateways from routing tables
- Machines at the far end of point-to-point interfaces
- Machines at randomly guessed addresses on networks of first hop gateways

What the Worm did not do. . .

- Did not
  - gain or attack root
  - destroy or attempt to destroy data
  - differentiate among networks

`sendmail` Exploit

- Worm exploited the "debug" function
- Debug includes the ability to send a mail message with a program as the recipient
- The recieving program runs with input from the body of the message

`fingerd` Exploit

- `fingerd` used `gets` for input
- `gets` takes input to a buffer without bounds checking
- buffer overflow allows for the creation of a fake stack frame, causing code to execute when the function returns

`rexec` Exploit

- `rexec` requires username and plaintext password that are passed over the network.
- The worm used pairs of usernames and passwords that it found to be correct on the local host.
- `/etc/passwd` facilitated this search.

`rsh` Exploit

- `/etc/hosts.equiv` contains a list of trusted hosts
- `.rhosts` contains a list of trusted hosts on a per-user basis
- `rsh` trusts the machine rather than any property of the user

How to Evade Detection?

- Covering tracks:
  - Erased argument list.
  - Deleted executing binary.
  - Used resource limit functions to prevent core dump.

Evading Detection...

- Camouflage:
  - Compiled under the name `sh`
  - Forked every 3 minutes
    - parent exits, child continues
  - Obscured constant strings by xor'ing each character with the constant 81

Code Red

- Code Red (original)
  - Released 7/13/2001
  - Exploited a buffer overflow in Microsoft Internet Information Server (Web Server)
  - Launched 99 threads to attack random IP addresses
  - 100th thread defaced the web server itself
- Problems with Code Red
  - Random number generator had a fixed seed
  - All copies of worm, on all infected hosts, attacked the same sequence of random IP addresses
  - Linear Spread

Code Red I

- Released 7/19/2001
- Fixed the random number generator seed problem
- Attacked `www.whitehouse.gov`
- Spread very rapidly
- Compromised all vulnerable MS IIS servers on the net
- Random Constant Spread (RCS) model

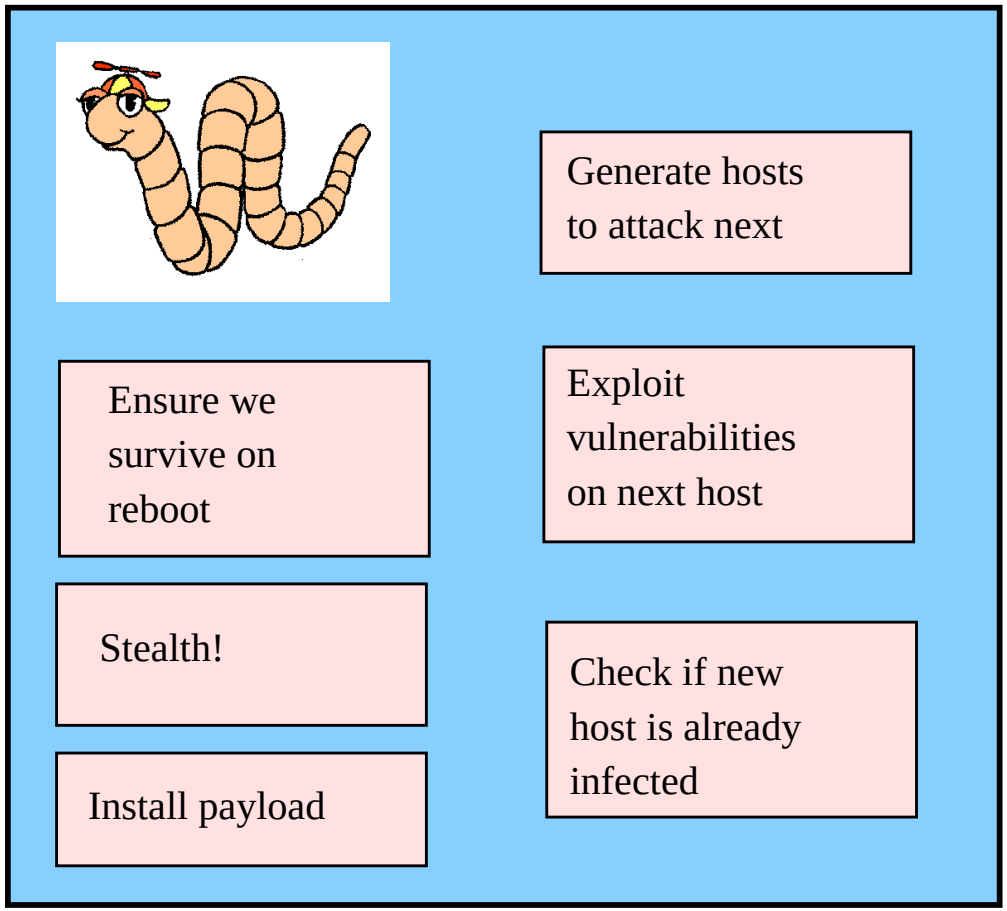
Code Red II

- Released 8/4/2001
- Unrelated to Code Red I
- A comment in the worm identified it as "Code Red II"
- Attacked MS IIS on Windows 2000
- Caused MS IIS on Windows NT to crash
- Installed root-access back door
- Introduced Localized Scanning strategy

What needs to be done?

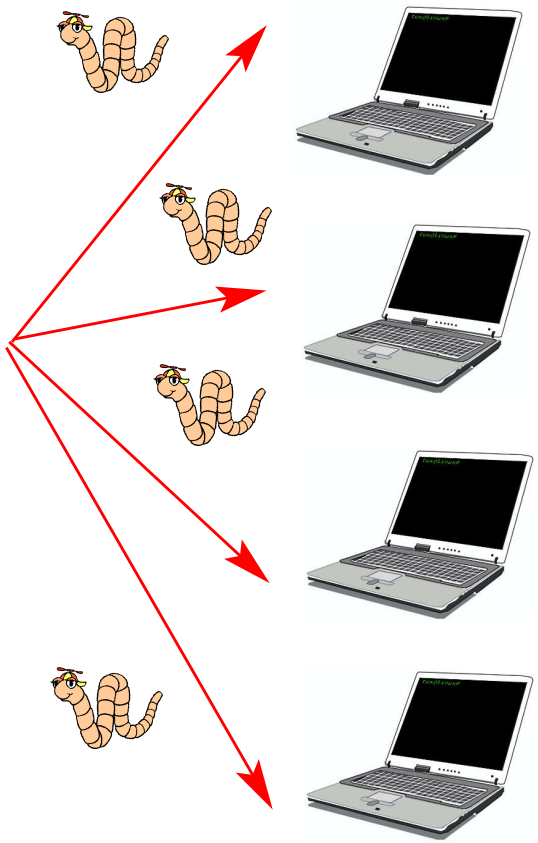
1. Find vulnerabilities to exploit.
2. Write code to
    1. generate machines to attack;
    2. exploit vulnerability;
    3. check if host is already infected;
    4. install/execute the payload;
    5. make the worm survive reboots.
3. Launch the worm on initial victims.

1. Find/buy vulnerabilities.
2. Code up the worm:



3. Find initial hosts to infect.

4. Launch!



Simple Scanning Strategies

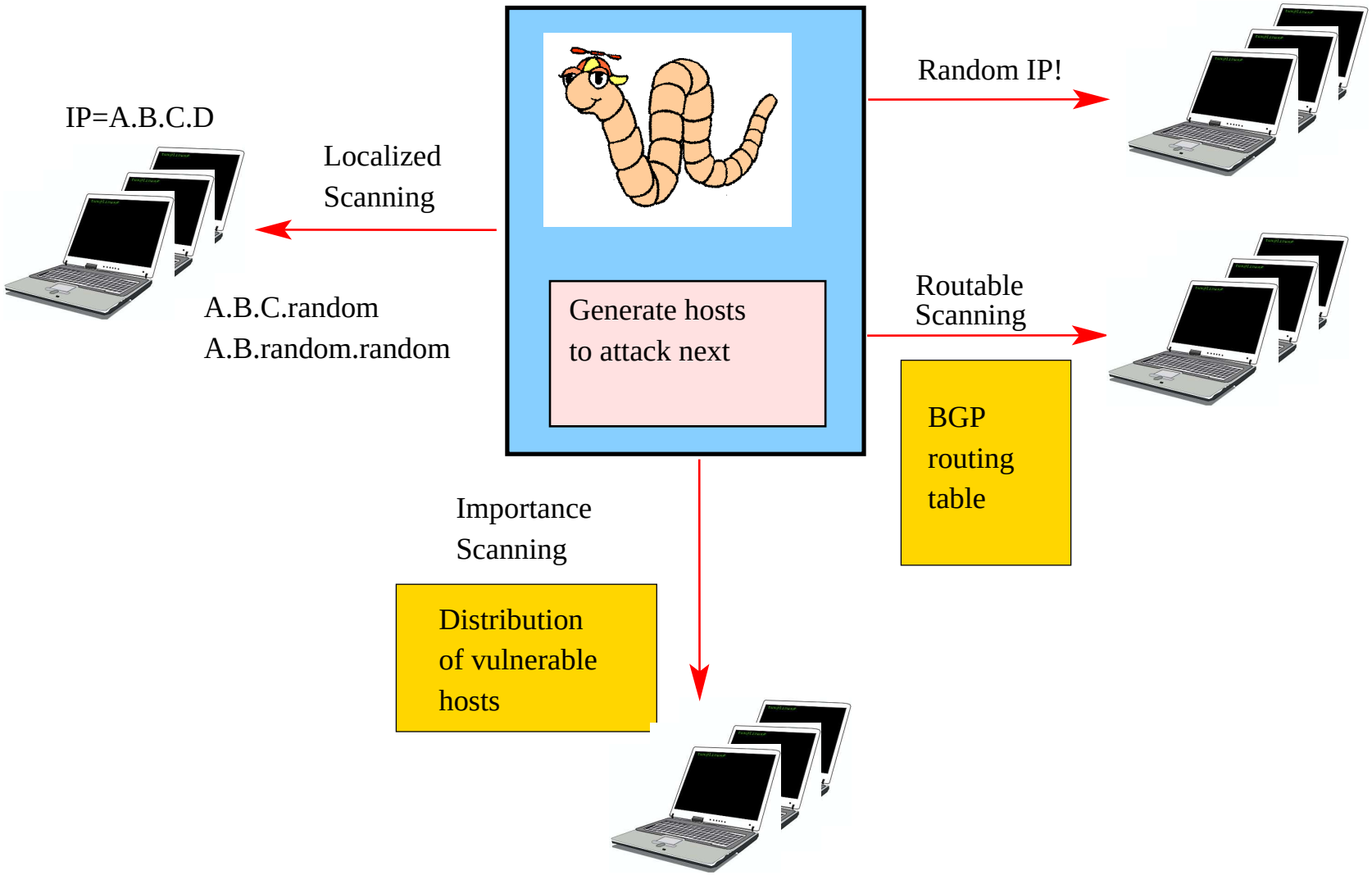
- Random scanning: choose target IP addresses at random.
- Routable scanning: select targets only in the routable address space by using the information provided by BGP routing table. $\Rightarrow$ no need to probe unassigned IP addresses.
- Importance scanning focus the scan on the most relevant parts of the address space. Needs a distribution of hosts which are vulnerable.

Simple Scanning Strategies. . .

- Localized scanning preferentially searches for vulnerable hosts in the "local" address space. Code Red II selects target IP addresses by:
  1. 50% of the time, choose an address with the same first byte;
  2. 37.5% of the time, choode an address with the same first two bytes,
  3. 12.5% of the time, chose a random address.

  Why? I node N has a vulnerability, it's close neighbors are likely to have the same vulnerability.

Simple Scanning Strategies. . .



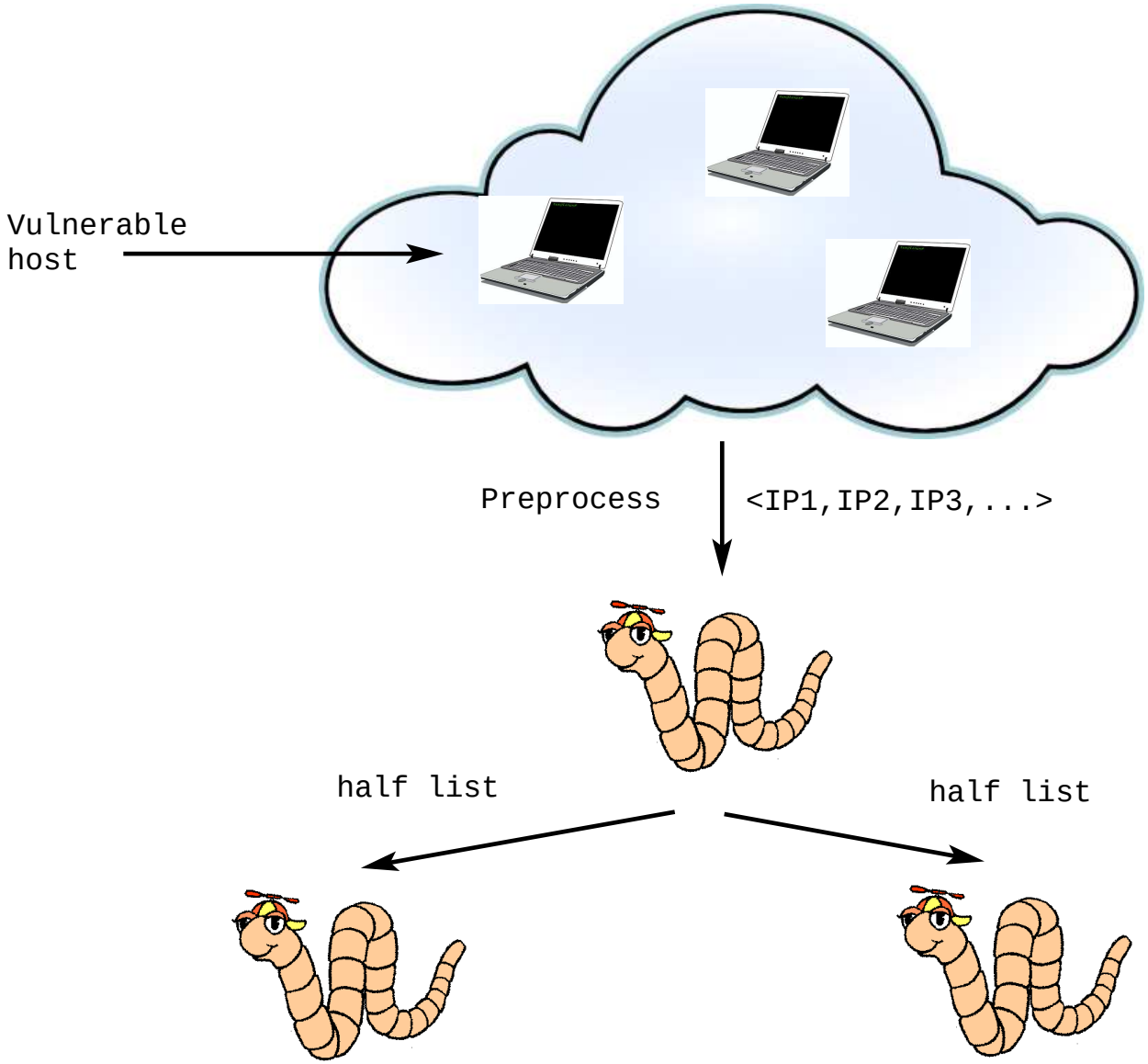
Advanced Scanning Strategies

- Hit-List Scanning
- Permutation Scanning
- Warhol Worms (Hit-List + Permutation Scanning)
- Topological Worms
- Flash Worms
- Stealth Worms

Hit-List Scanning

- Worms need to "get off the ground" quickly
- Do preparatory work before releasing the worm:
  - Collect IP addresses of vulnerable servers
  - Create a hit-list with them
- The worm starts with the full hit-list
- Partition the hit-list in half each time a host is infected
- Divide-and-Conquer approach
- Once hit-list is exhausted, do random attacks

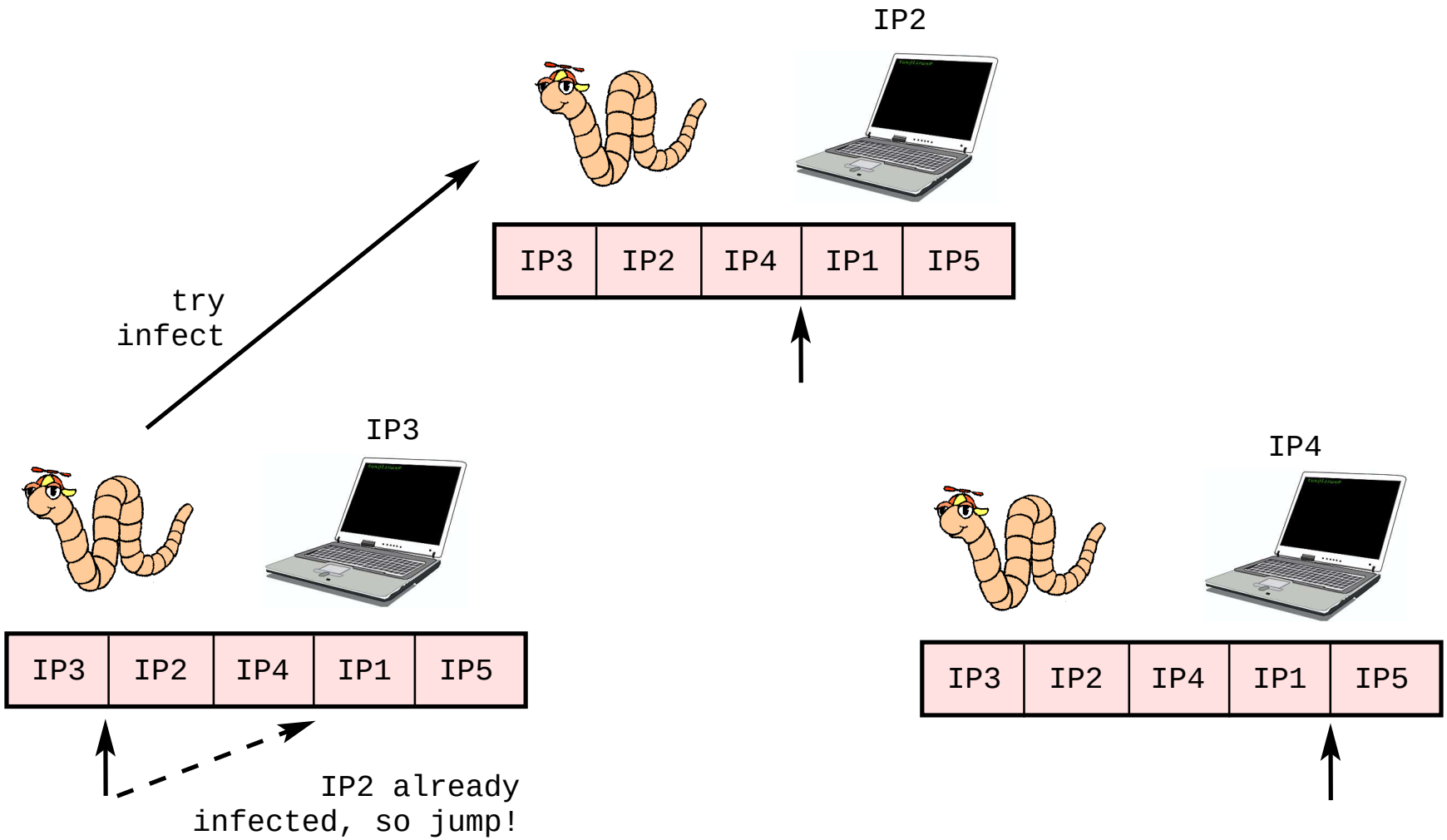
Hit-List Scanning. . .



Permutation Scanning

- Would like to avoid attacking already-attacked hosts
- But, we can't tell ahead of time which hosts have already been attacked
- However, we can predict what other worms are doing
- How to stay out of the way of other worms:
  - All worms start with the same random permutation of addresses
  - Each worm starts at a different spot in the list
  - If you find an already-compromised host, then jump to a new random spot in the list

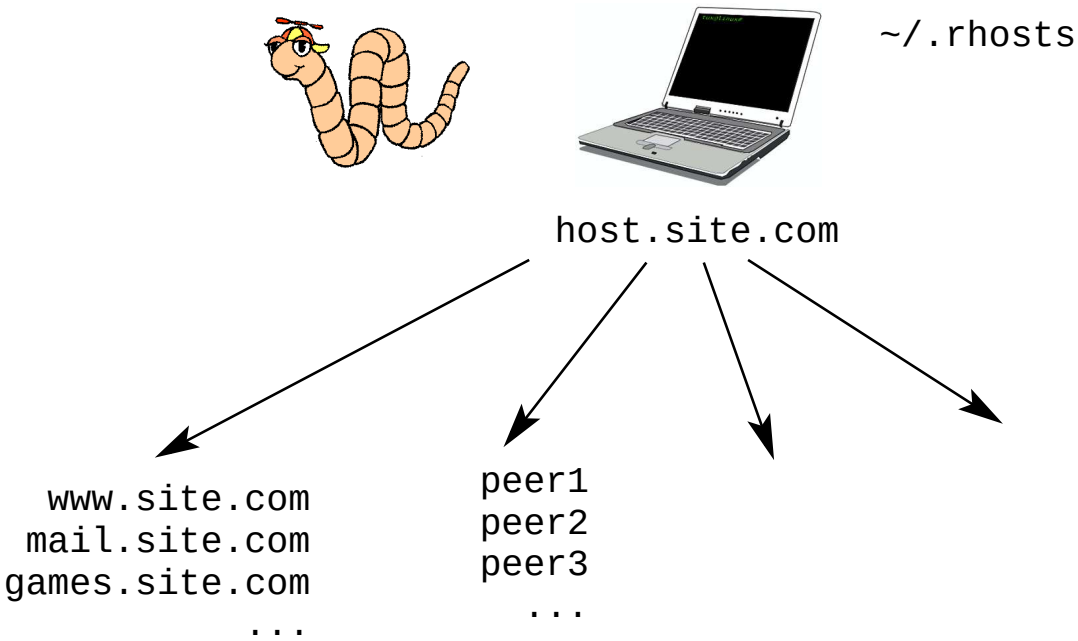
Permutation Scanning...



Topological Scanning

- Alternative to Hit-List
- Use information on the compromised host to find more targets
- Examples:
  - List of peers on peer-sharing systems
  - URLs on web servers
  - `www.yahoo.com` $\longrightarrow$ `mail.yahoo.com`, `games.yahoo.com`, ...

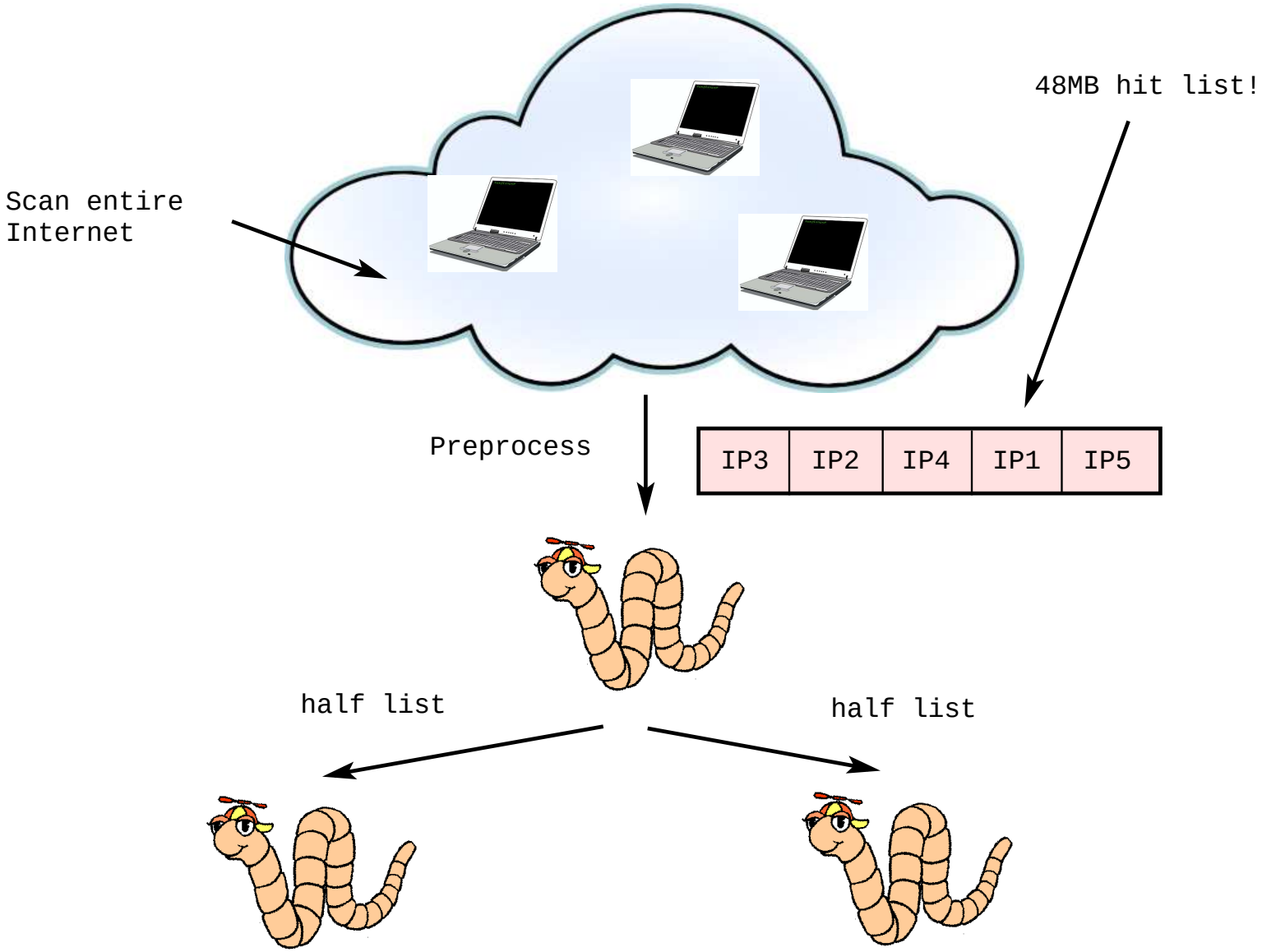
Topological Scanning. . .



Flash Worms

- Infect most vulnerable servers within 10s of seconds
- Works like hit-lists
- Scan the entire Internet for vulnerable machines prior to launching attack
- Scanning could be done in 2 hours with an OC-12 connection
- 12.6 million web servers = 48 MB list
- Divide and Conquer

Flash Worms. . .


48MB hit list!
Scan entire Internet
Preprocess
IP3
IP2
IP4
IP1
IP5
half list
half list

Warhol Worms

- Warhol Worms: Everyone has their 15 minutes of fame
- Extermely fast spreading worm.
- Combines Hit-List and Permutation Scanning
- Attacks most vulnerable hosts on the net within 15-60 minutes

Stealth Worms

- Spread slowly rather than quickly
- Infect without being detected
- Use unsuspecting clients to spread the attack

Detecting Worms

- Establish a Cyber Center for Disease Control (CDC)
  - Identify Outbreaks
  - Analyze Pathogens
  - Fight Infections
  - Anticipate new vectors
  - Devise detectors for new vectors
  - Resist future attacks

Outline

Stuxnet

https://www.youtube.com/watch?v=CS01Hmjv1pQ

Zero-Day Attack

- A zero-day attack or threat is an attack that exploits a previously unknown vulnerability in a computer application or operating system.
- It is called a *zero-day* because the programmer has had zero days to fix the flaw (in other words, a patch is not available).
- Once a patch is available, it is no longer a zero-day exploit.
- It is common for individuals or companies who discover zero-day attacks to sell them to government agencies for use in cyberwarfare.

http://en.wikipedia.org/wiki/Zero-day_attack

Buy the Book!

Stuxnet is embarrassing, not amazing

Whoever developed the code was probably in a hurry and decided using more advanced hiding techniques wasn't worth the development/testing cost. For future efforts, I'd like to suggest the authors invest in a few copies of Christian Collberg's book. It's excellent and could have bought them a few more months of obscurity.

http://rdist.root.org/2011/01/17/stuxnet-is-embarrassing-not-amazing

Regin

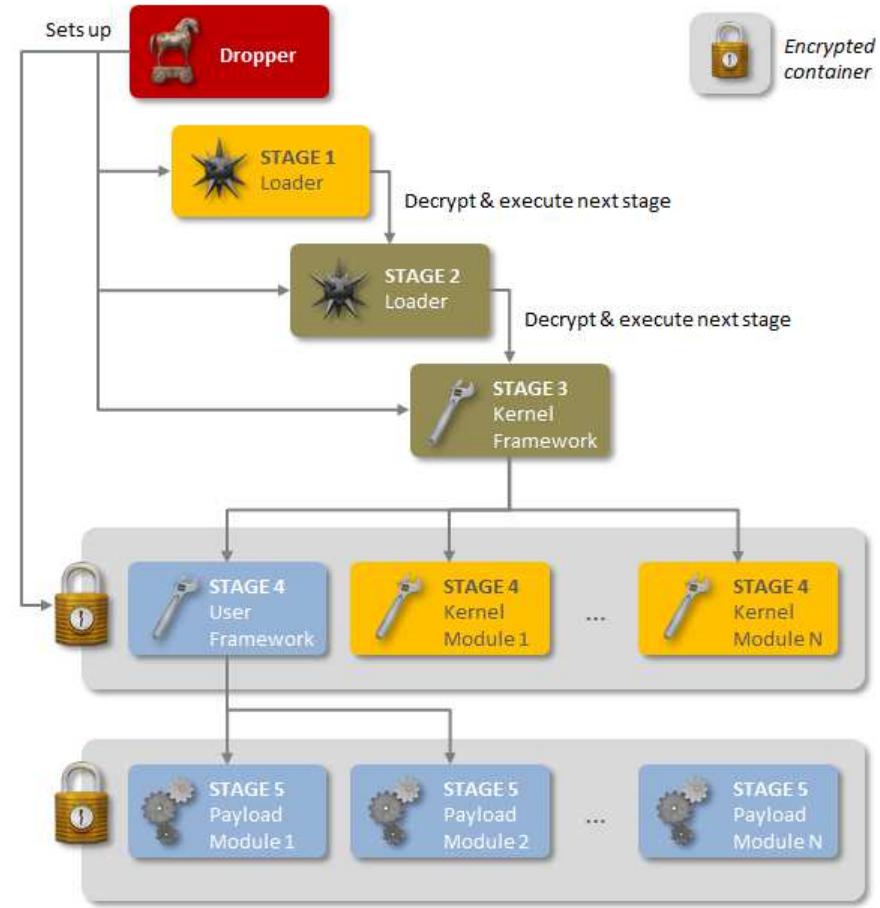


Figure 1. Regin's five stages

https://www.youtube.com/watch?v=6wMS1aybCZ8

Outline

Quiz

- What is a virus?
- What is a Trojan?
- What is a worm?
- What is a backdoor?
- What is a logic bomb?
- What is an encryption virus?
- What is a polymorphic virus?
- What is a metamorphic virus?
- What are important modules in a worm?

Readings and References

- Chapter 4 in *Introduction to Computer Security*, by Goodrich and Tamassia.

Exam Problem

- You have discovered a new virus that embeds itself in a host program P like on the next slide.

P

$$K = \langle K_0, K_1, K_2, K_3, K_4, K_5 \rangle$$

.....

$$C^{obf} = \begin{array}{l} C_0 \oplus K_0,\ C_1 \oplus K_1,\ C_2 \oplus K_2,\ C_3 \oplus K_3,\ C_4 \oplus K_4,\ C_5 \oplus K_5, \\ C_6 \oplus K_0,\ C_7 \oplus K_1,\ C_8 \oplus K_2,\ C_9 \oplus K_3, C_{10} \oplus K_4, C_{11} \oplus K_5, \\ C_{12} \oplus K_0, C_{13} \oplus K_1, C_{14} \oplus K_2, C_{15} \oplus K_3, C_{16} \oplus K_4, C_{17} \oplus K_5, \\ \quad \ldots \qquad \ldots \qquad \ldots \qquad \ldots \qquad \ldots \qquad \ldots \end{array}$$

```
void run_virus() {
    extract K from P;
    extract C^obf from P;
    for(i=0; i<6·n; i++)
        C_i ← C_i^obf ⊕ K_{i mod 6};
    jump to the beginning of the virus code C_0;
}
```

Exam Problem...

- C^{obf} is an obfuscated version of the virus body C,
- K is a random secret key.
- `run_virus()` is a function that loads, de-obfuscates, and executes the virus body.
- All are inserted at random positions in the host program P.

Exam Problem...

- You have learned the following facts about the virus:
  1. The virus body C is a multiple of 6 bytes: $C = \langle C_0, C_1, C_2, C_3, \ldots, C_{6 \cdot n-1} \rangle$
  2. K is 6 bytes long, and different for each host program.
  3. C^{obf} is obtained by XOR-ing the cleartext virus body C with K, where K is repeated the necessary number of times.

Exam Problem...

- Assume that you have obtained the cleartext body C of the virus and a set of programs that you suspect are infected.
- You want to write a virus scanner that returns `true` if C occurs in host program P (consisting of bytes $\langle P_0, P_1, P_2, \ldots \rangle$) and `false` otherwise:

```
boolean detect(
    C = ⟨C_0, C_1, C_2, ...⟩,
    P = ⟨P_0, P_1, P_2, ...⟩) {
    ...
}
```

Exam Problem...

- You can't look for a signature of `run_virus()` in P (because it's too similar to code that occurs in normal programs).
- You can't examine `run_virus()` for the location of C^{obf} (because you don't have tools that are precise enough to analyze the `run_virus()`'s binary code).
- You can't execute P to look for the behavior of `run_virus()` or the virus body (because it would take too much time).
- Your algorithm should be fast, i.e. polynomial-time in the length of P.