CSc 466/566

Computer Security

11 : Malware II — Worms

Version: 2019/10/07 13:35:49

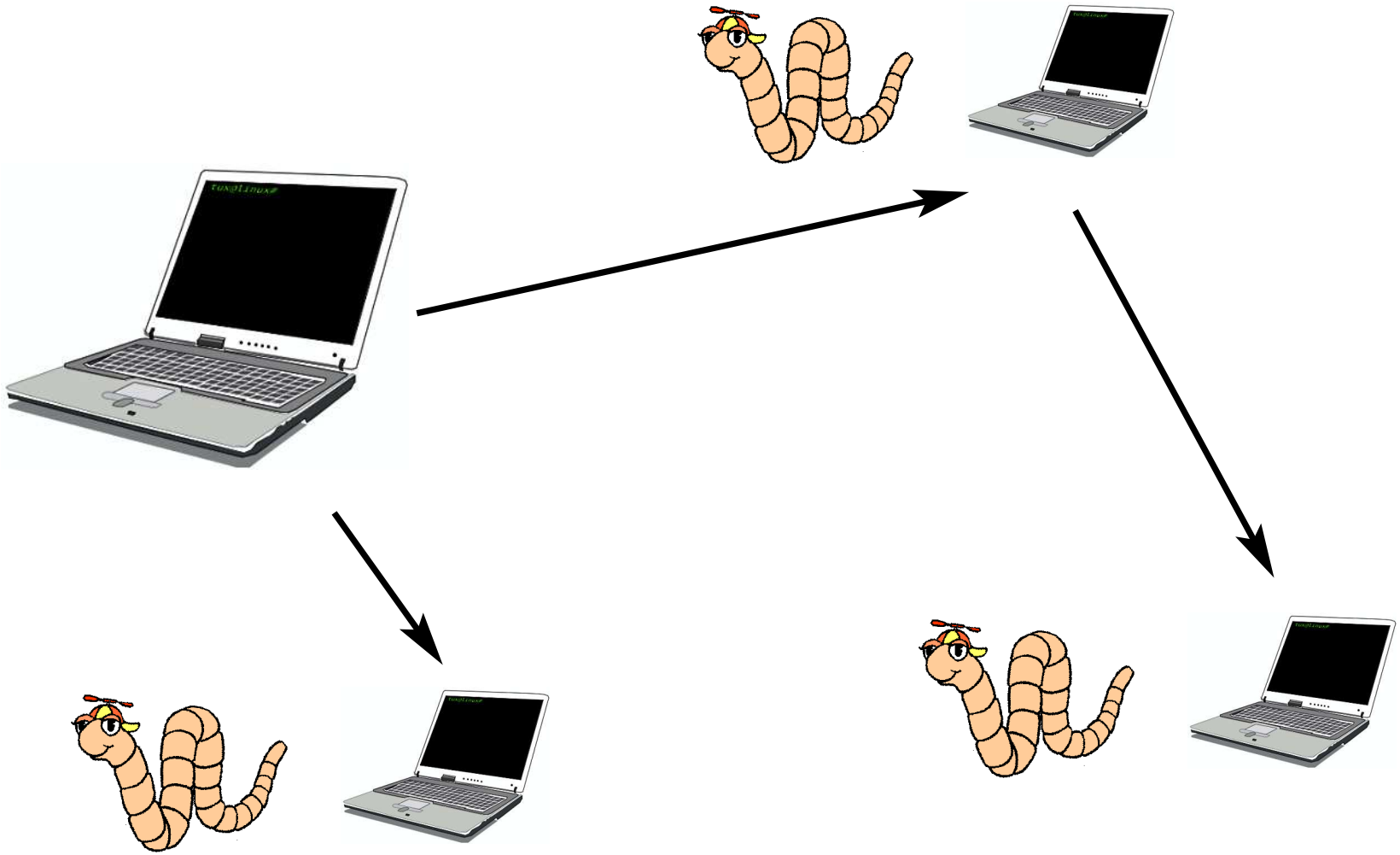
Department of Computer Science
University of Arizona

collberg@gmail.com

Christian Collberg

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

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

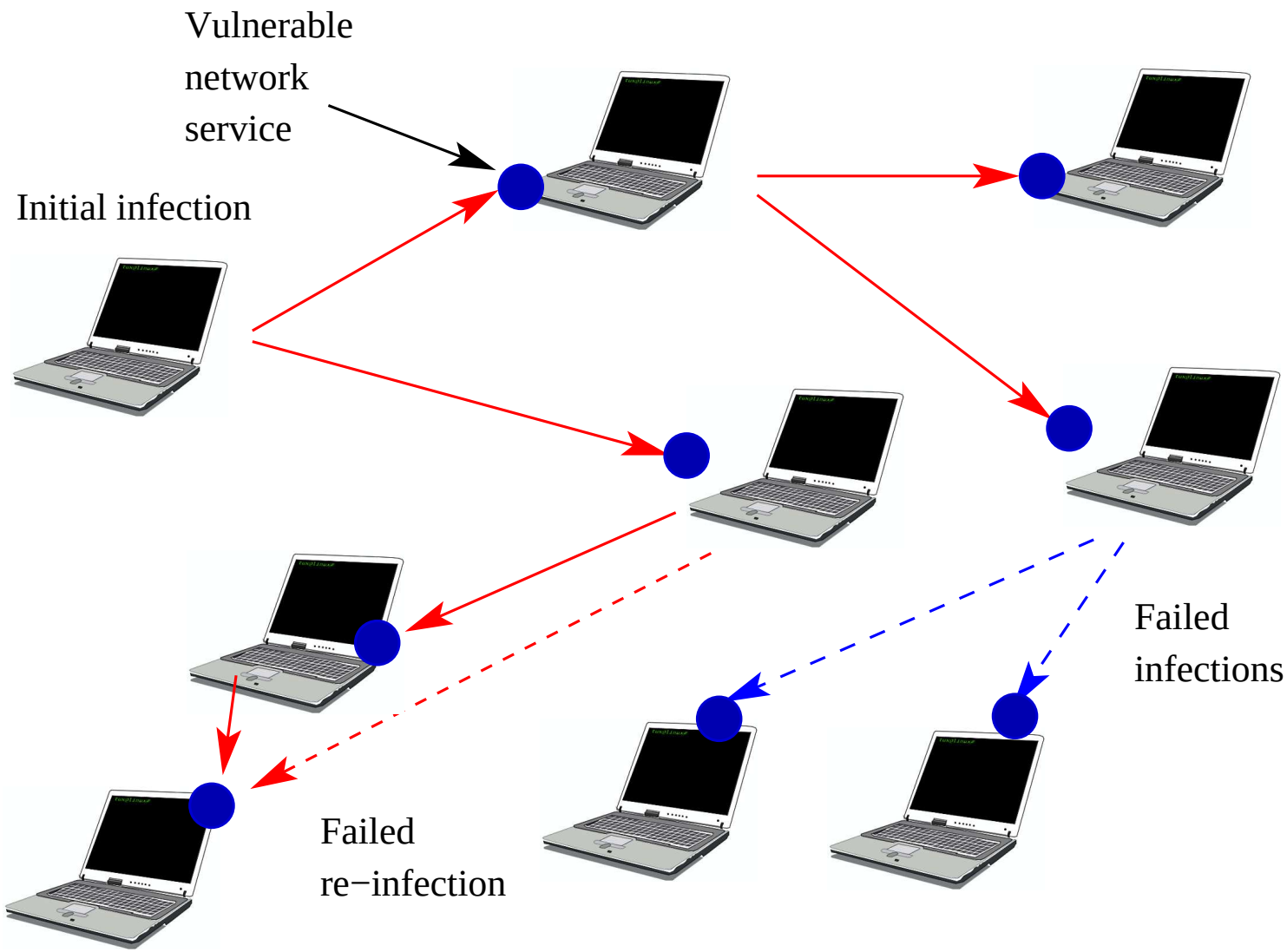
Initial infection

Vulnerable network service

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

4. Ensure survival on reboot
5. Execute payload

Worm Propagation



Infection Animation

- Witty worm:

  https://www.caida.org/research/security/witty/animations/world_big-witty_2h.gif

- Nyxem Email Virus:

  https://www.caida.org/research/security/blackworm/animations/nyxem-hosts-both-02.gif

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.

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

Exercise

1. $\beta = 0, I(t) = 1, S(t) = 4 \Rightarrow I(t+1) =$

Exercise

1. $\beta = 0, I(t) = 1, S(t) = 4 \Rightarrow I(t+1) = 1$
2. $\beta = 1, I(t) = 1, S(t) = 4 \Rightarrow I(t+1) =$

Exercise

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. $\beta = 0.5, I(t) = 1, S(t) = 4 \Rightarrow I(t+1) =$

Exercise

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...

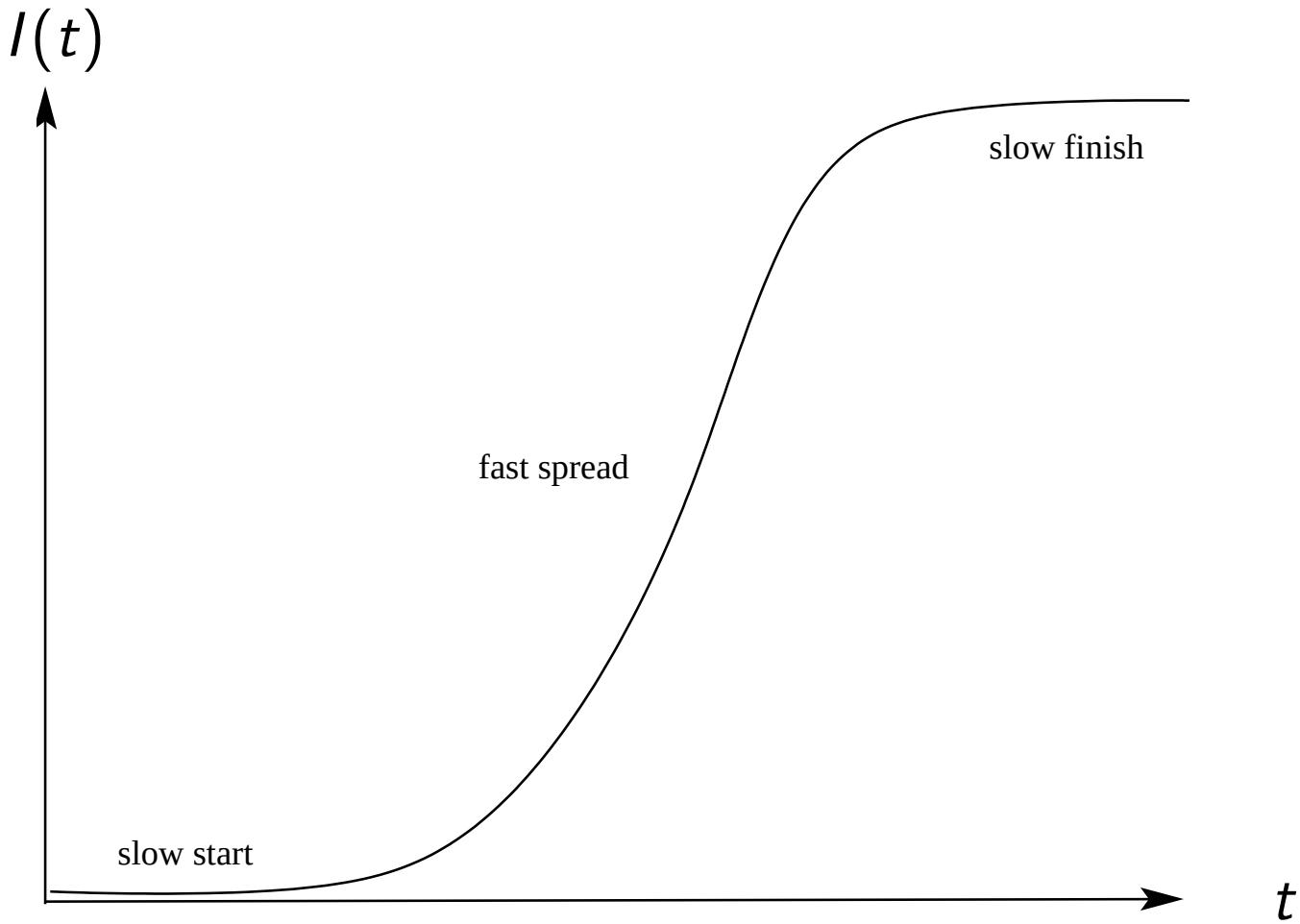
$$
\begin{aligned}
I(0) &= 1 \\
S(0) &= N - 1 \\
I(t+1) &= I(t) + \beta \cdot I(t) \cdot S(t) \\
S(t+1) &= N - I(t+1)
\end{aligned}
$$

- 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

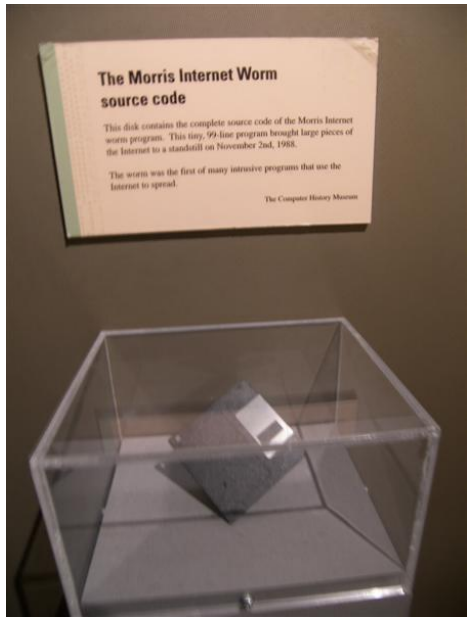
- Attacked BSD Unix derivative systems on the internet on 2 November 1988.
- Specifically targeted SUNs and VAXes.
- Computer Virus TV News Report 1988:

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

Robert Morris

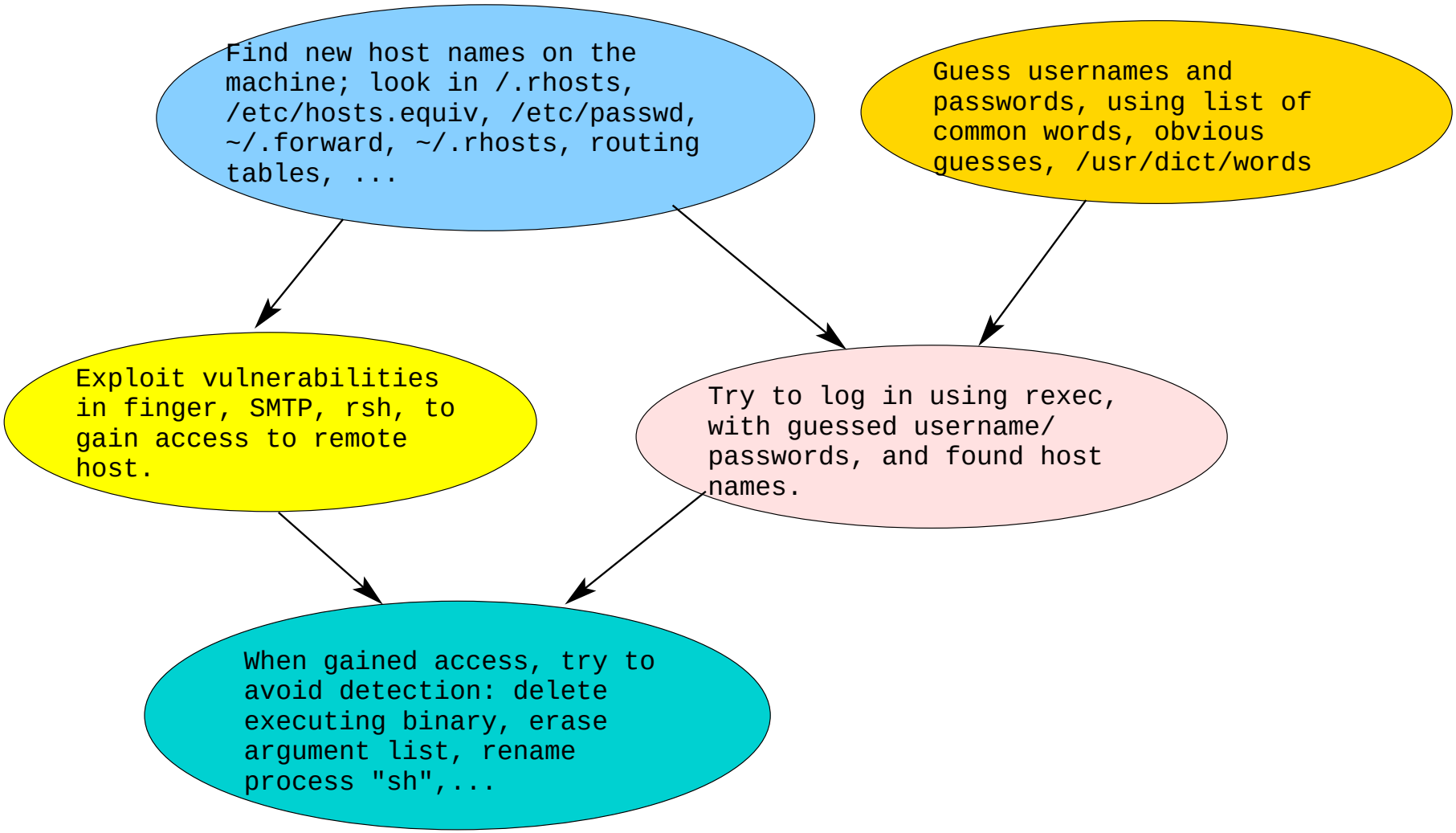
- Then: Cornell graduate student.
- Convicted under the Computer Fraud and Abuse Act: 3 years probation, 400 hours community service, $10,050.
- Now: Tenured Professor at MIT.
- Son of Robert Morris, coauthor of UNIX, former chief scientist at the National Computer Security Center.

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.

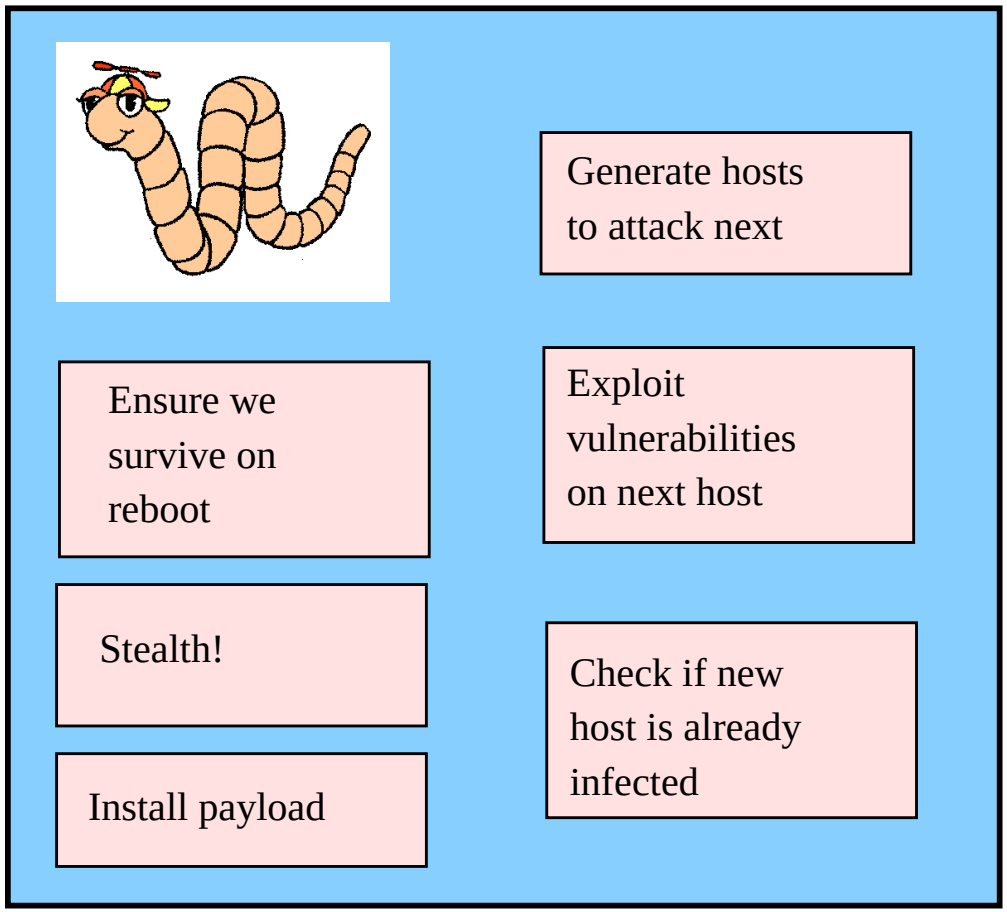
Morris Worm Overview



What needs to be done?

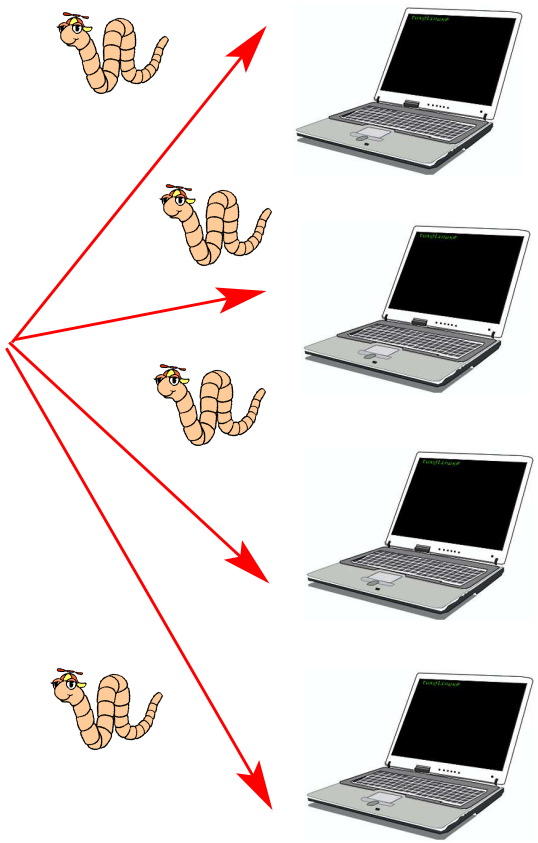
1. Find vulnerabilities to exploit.
2. Write code to
    1. generate machines to attack;
    2. exploit vulnerability;
    3. check if host is already infected;
    4. install/execute the payload;
    5. make the worm survive reboots.
3. Launch the worm on initial victims.

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



3. Find initial hosts to infect.

4. Launch!



Simple Scanning Strategies

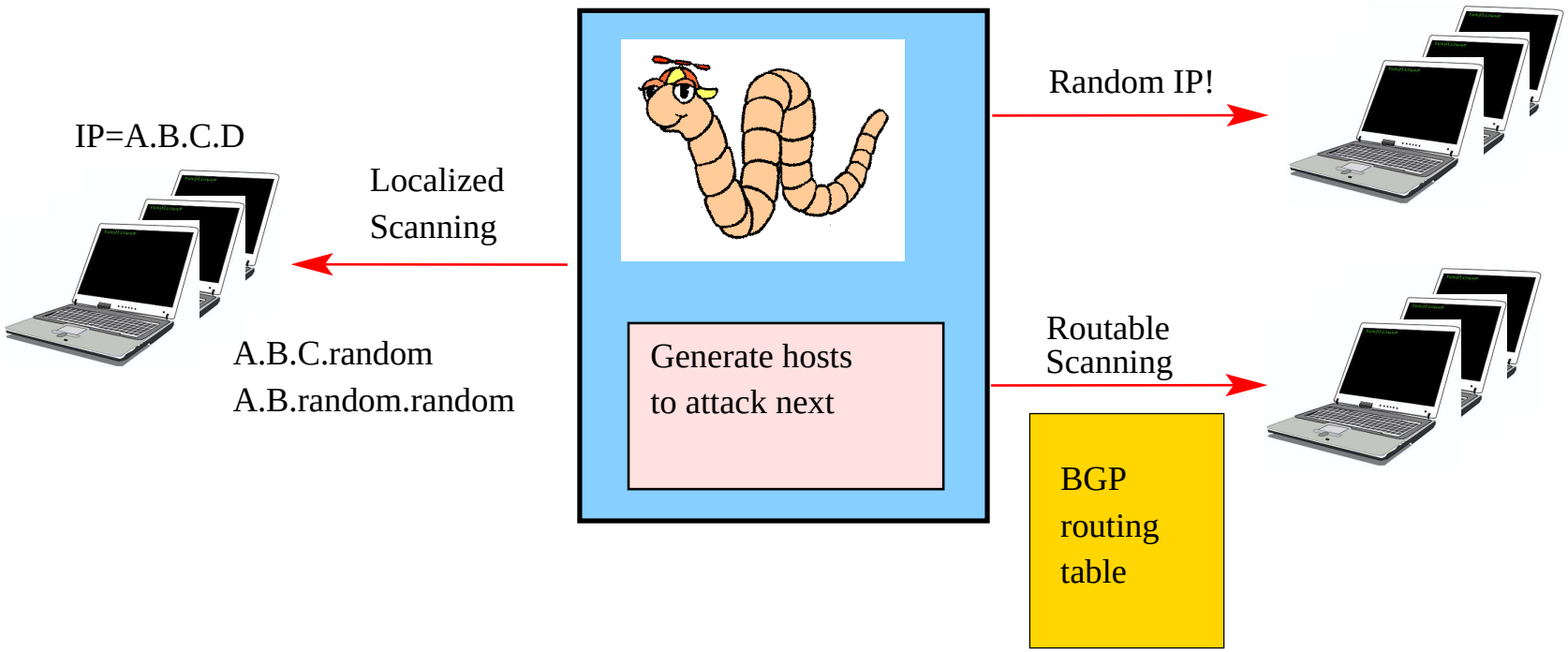
- Random scanning: choose target IP addresses at random.
- Routable scanning: select targets only in the routable address space by using the information provided by BGP routing table. $\Rightarrow$ no need to probe unassigned IP addresses.

Simple Scanning Strategies. . .

- Localized scanning preferentially searches for vulnerable hosts in the "local" address space. The *Code Red II* worm 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.

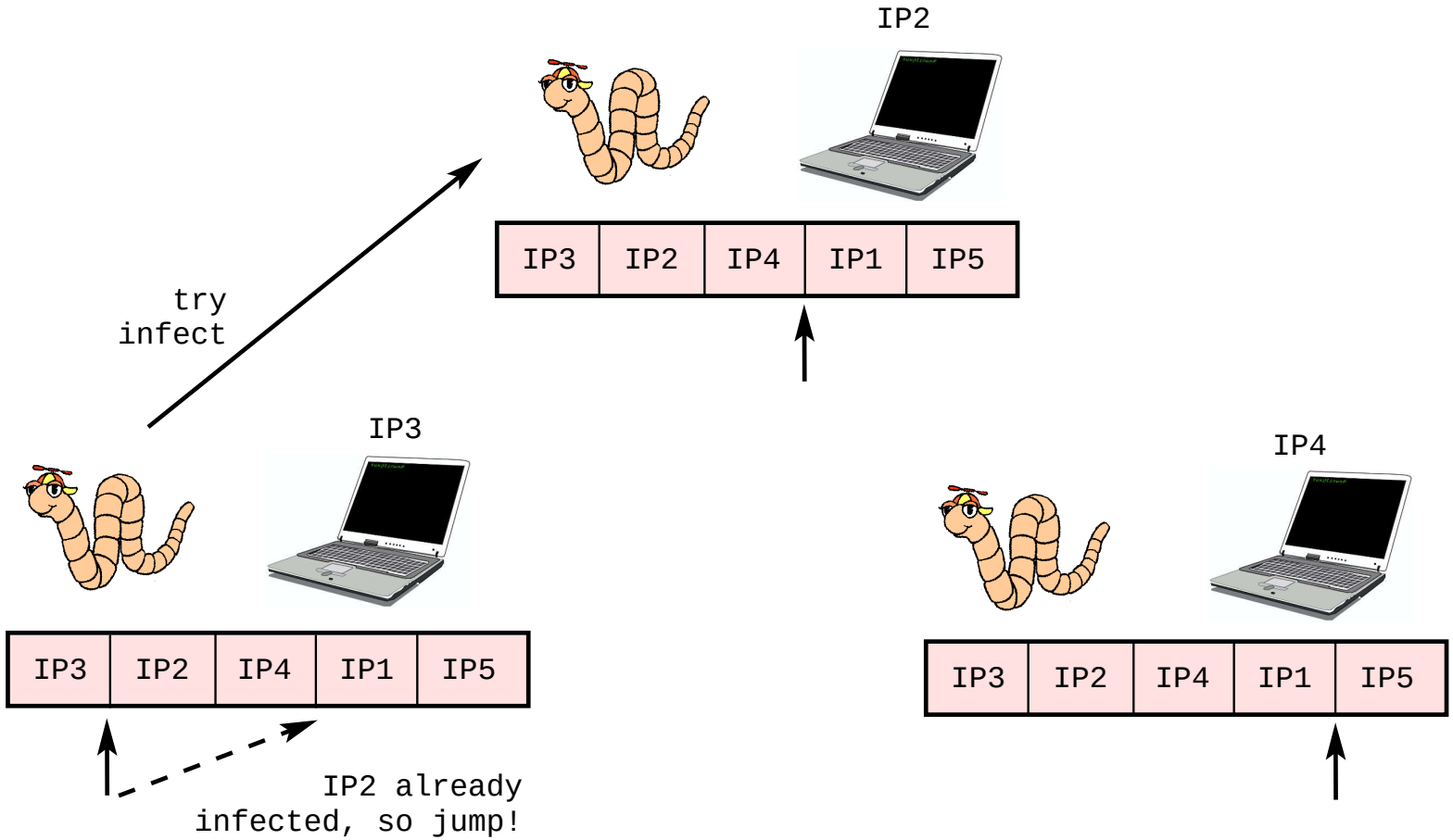
Simple Scanning Strategies. . .



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...



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

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

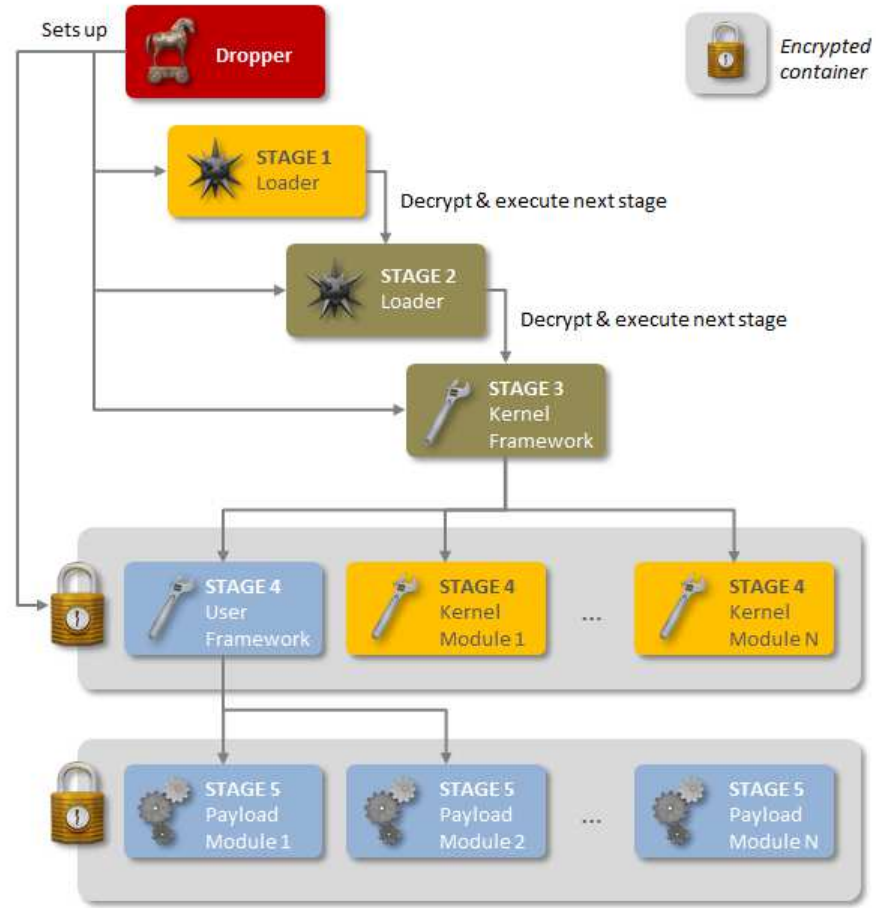


Figure 1. Regin's five stages

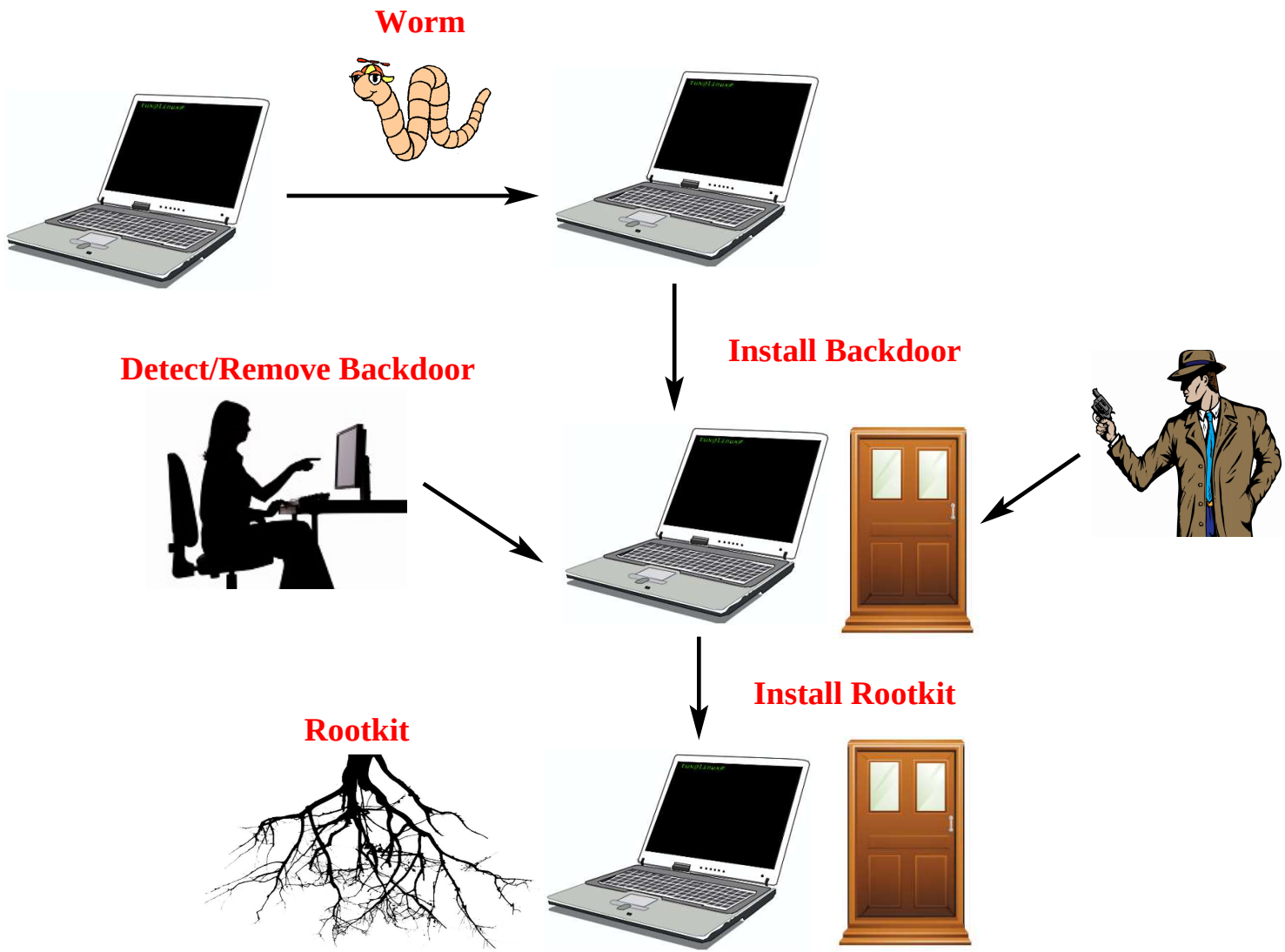
https://www.youtube.com/watch?v=6wMS1aybCZ8

Exercise

- Get in groups with a few friends.
- Go to https://cve.mitre.org/cve/search_cve_list.html. This is a list of known vulnerabilities.
- Search for "something interesting". Record the CVE.
- Tell the class about the vulnerability you found!

Outline

Rootkits


Worm
Detect/Remove Backdoor
Install Backdoor
Install Rootkit
Rootkit

Rootkits

- Rootkits try to make other malware on a machine undetectable to malware scanners.
- They don't typically perform any malicious activity themselves.
- *Rootkit exploitation explained by Kevin Rose:*

  https://www.youtube.com/watch?v=8qFmKV8CTSQ

Rootkit Types: User-Mode

- A User Mode Rootkit runs with the privileges of a regular user.
- Here's the code for /bin/ls:

```
void ls () {
   for(all files f)
     printf("%s\n", f);
}
```

- The rootkit changes the code of /bin/ls to hide the presence of certain files:

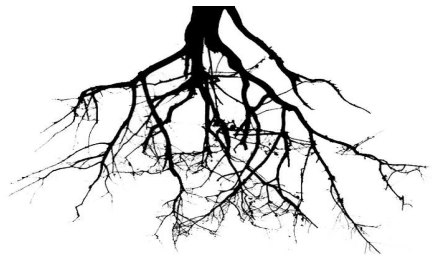
```
void ls () {
   for(all files f)
     if (f != "malware.exe")
        printf("%s\n", f);
}
```

Detecting User-Mode Rootkits: tripwire

Worm

Install Rootkit
Install Backdoor

Rootkit



file	hash
/bin/ls	0x42fa5
/etc/passwd	0x13dc4
...	...

file	hash
/bin/ls	0x013f5
/etc/passwd	0x13dc4
...	...

Cryptographic Hash Functions

- cryptographic hash function (CHF) are one way hash functions:
  - Given message M, it's easy to compute $y \leftarrow h(M)$;
  - But, given a value y it's hard to find an M such that $y = h(M)$.
- CHFs are checksums or compression functions: they take an arbitrary block of data and generate a unique, short, fixed-size, bitstring.

Common Hash Functions

- MD5: 128-bit hash
- SHA-1: 160-bit hash
- SHA-256: 256-bit hash

```
> echo "hello" | shasum
f572d396fae9206628714fb2ce00f72e94f2258f  -
> echo "hella" | shasum
1519ca327399f9d699afb0f8a3b7e1ea9d1edd0c  -
> echo "can't believe it's not butter!"|shasum
34e780e19b07b003b7cf1babba8ef7399b7f81dd  -
```

Detecting User-Mode Rootkits: tripwire...

1. When initializing a system, compute hashes of all important files:

```
> shasum /etc/passwd > okfiles
```

   You can store the hashes in secure storage, such as on a CDROM.

2. Detect tampering by re-computing the hashes and comparing against the original values.

```
> shasum --check okfiles
/etc/passwd: OK
```

- http://sourceforge.net/projects/tripwire.

Rootkit Types: Kernel-Mode

- A Kernel-Mode Rootkit runs with the highest operating system privileges.
- The rootkits replaces parts of the OS kernel or device drivers.
- Function hooking: replace OS functions with new versions that hide the presence of malware.

Rootkit Types: Kernel-Mode...

- Here's the system call getdents:

```
Name
    getdents - get directory entries
Synopsis
    int getdents(unsigned int fd,
                 struct linux_dirent *dirp,
                 unsigned int count);
```

- The rootkit changes the code to hide the presence of certain files:

```
int getdents(unsigned int fd, struct
    linux_dirent *dirp, unsigned int count){
    for(all files in fd)
        if (f != "malware.exe")
            add f to dirp
}
```

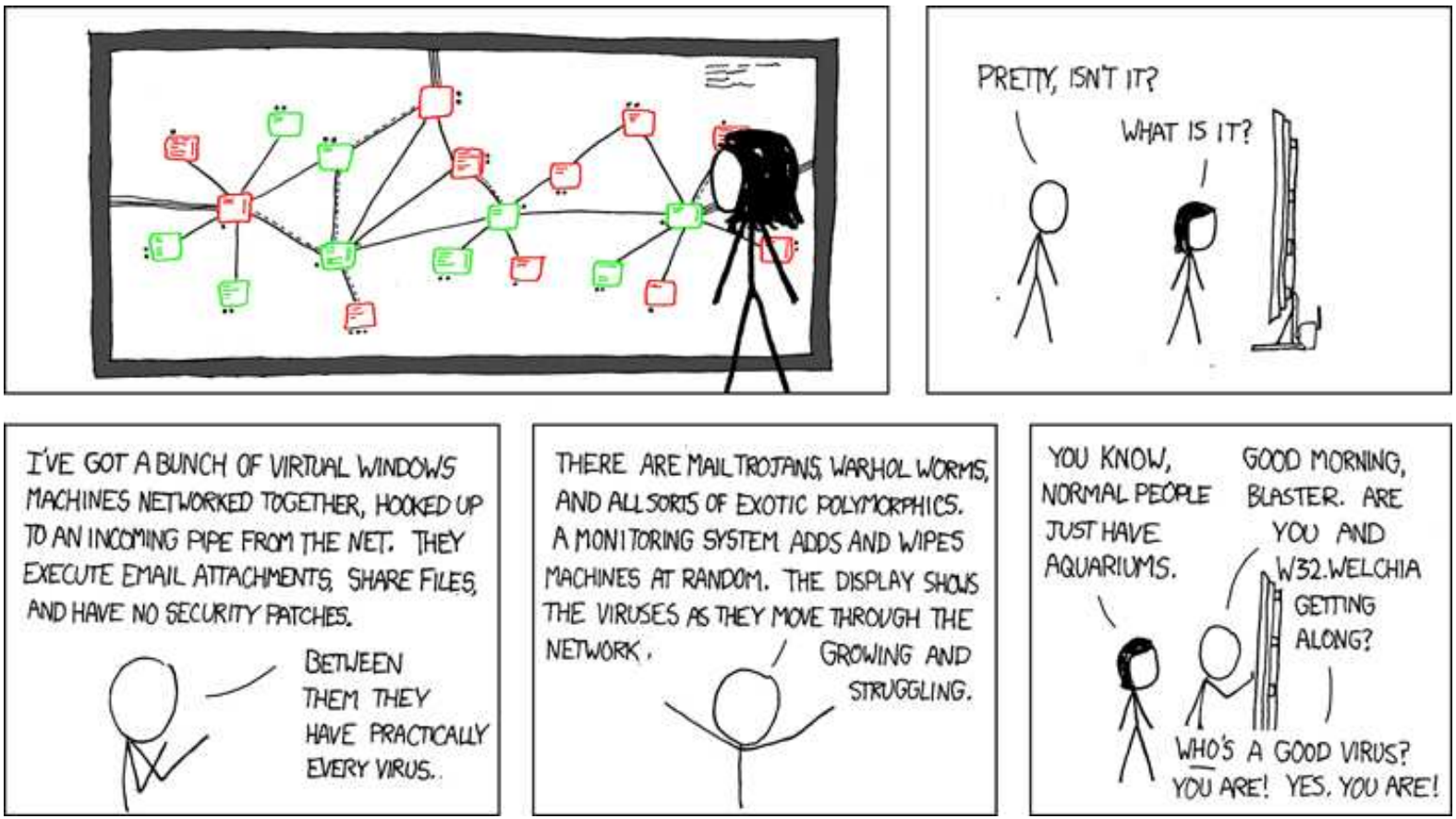
Rootkit Types: User-Mode

- The advantage of the Kernel-Mode attack is that any user-mode program that lists the content of a directory will exclude the malware files.
- This includes `/bin/ls`, which now doesn't need to be changed:

```
void ls () {
   for(all files f)
     /* Won't show malware.exe: */
     printf("%s\n", f);
}
```

Outline

XKCD — Network



https://xkcd.com/350

Readings and References

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