Programs and Programming (I)

Jun Li
lijun@cs.uoregon.edu

Learning Objectives

- <u>Programming oversights</u>
  - Including the understanding of buffer overflows, incomplete mediation, and time-of-check to time-of-use errors
- <u>Malicious code</u>
  - Viruses, Trojan Horses, Worms
- Countermeasures against program threats

Program Security at the Heart of Computer Security

- Recall: A **computing system** is a collection of hardware, software, data, and users
- Software, i.e., programs, can be the operating system, device drivers, networking code, database management system, or any other applications
- Our focus in this chapter: *The writing of programs*.

Program Security Assessment

A program is “secure” if

- it takes too long to break?
- it has run for a long period without failures? or
- if it has no potential faults in meeting security requirements?

One approach to judging quality in security has been **fixing faults.**

Fixing Program Faults

- Software with many faults early on is likely to have many others later
- Faults lead to failures
- Early practice: penetrate and patch
  - Tiger team
  - Can a program withstand attacks?
  - Could create false impression if no faults found
  - Patch may introduce new faults and performance penalty

Software Security *is* Hard

No “silver bullet”:

- Security often conflicts with usefulness and performance
- Easy to test “should do” of a program, but hard to test “shouldn’t do”
  - Sheer size and complexity of the latter
- Programming techniques evolve faster than security techniques

Unexpected Behavior

- Program security flaw: inappropriate program behavior caused by a program fault/vulnerability
- Vulnerability/fault -> flaws/unexpected behavior -> failures/harms
  - A vulnerability usually leads to a class of flaws
- Flaws have two categories: inadvertent human errors vs. malicious, intentionally introduced flaws
  - The former is more numerous than the latter
  - The former can be exploited by attackers

Nonmalicious Program Errors

Human make mistakes, especially the following three classic error types:

- Incomplete mediation
- Time-of-check to time-of-use errors
- Buffer overflows

Incomplete Mediation

```
http://www.things.com/order.asp?cus
tID=101&part=555A&qy=20&price=10@sh
ip=boat@shipcost=5&total=205
```

```
http://www.things.com/order.asp?cus
tID=101&part=555A&qy=20&price=1
@ship=boat@shipcost=5&total=25
```

Time-of-check to Time-of-use Error

- TOCTTOU
- Also known as serialization or synchronization flaw
- Attackers can exploit the delay: What was checked is no longer valid when an object is accessed

my_file	change byte 4 to "A"

A work ticket: Data Structure for File Access.

your_file	delete file

Modified Data: while the mediator has copied the work ticket and is doing the checking

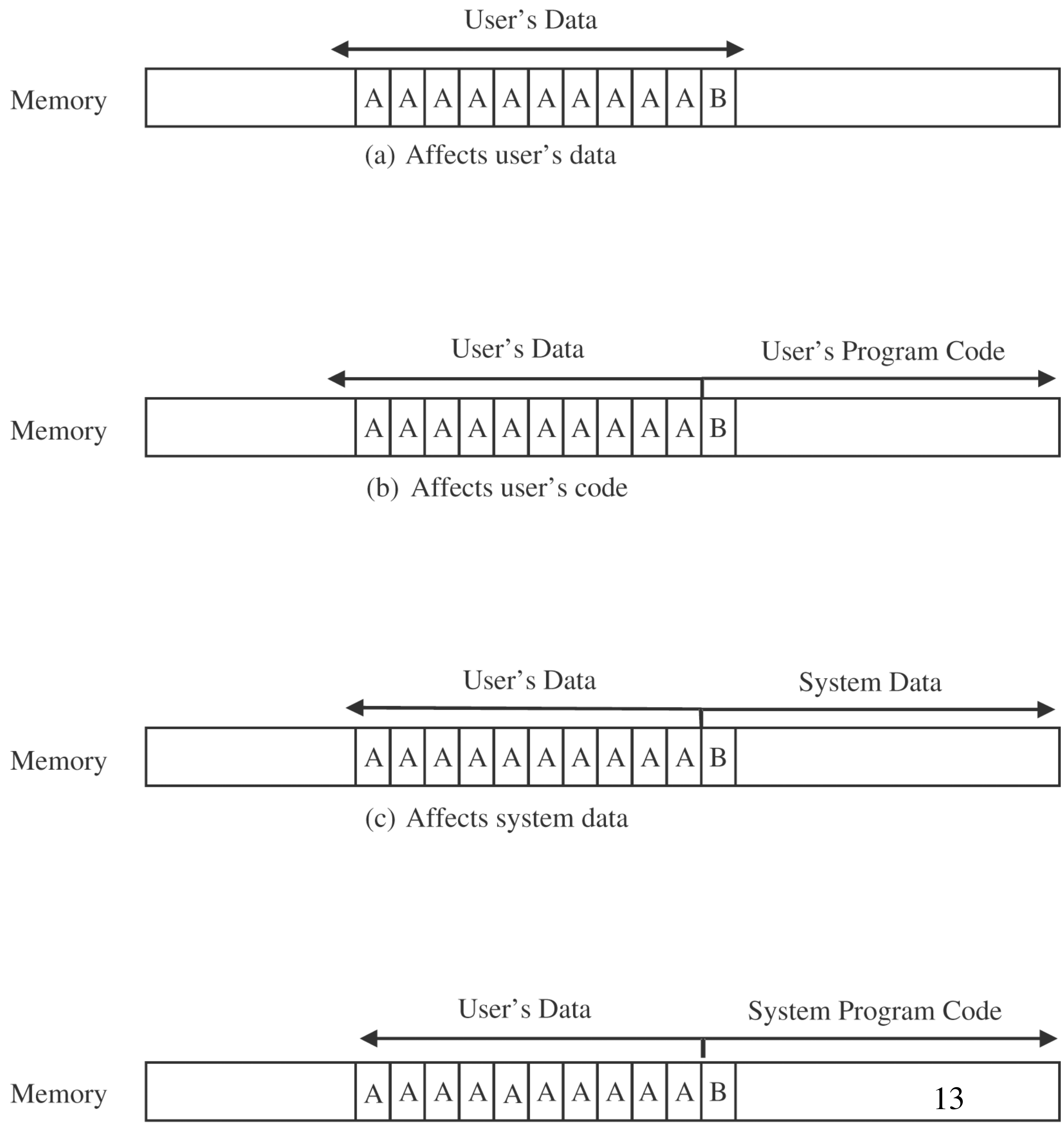
Buffer Overflows--Turning a minor annoyance to a major attack vector

- A buffer is a space in memory to hold data
- Every buffer has a finite capacity
- In many program languages, the programmer must declare the buffer size
  - But in some, no need to predefine it
- Compiler: can help in some cases, but not all

```
char sample[10];
sample[10] = 'B';
sample[i] = 'B';
```

```
for (i=0; i<=9;
     i++)
sample[i]='A';

sample[10]='B'
```



(a) Affects user's data

(b) Affects user's code

(c) Affects system data

(d) Affects system code

Security Implications

- Attack can plan instruction codes toward the overflowed area to execute malicious functions
  - System code space
  - Stack space
  - Parameter space

Stack Space

- Transferring control to a sub-procedure uses a stack
  - parameters, return address, old stack pointer, local values are pushed onto a stack
- Attacker can change the old stack pointer, or the return address
  - Thus redirecting execution to attacker's code

Parameter Space

- Example: `http://www.somesite.com/subpage/userinput.asp?parm1=(808)555-1212&parm2=2009Jan17`
- What if one enters an extremely long telephone number?
  - Crash?
  - Or more dangerous consequence?

Reflections: Are These Three Classic Error Types Easy to Avoid?

- Buffer overflows
- Incomplete mediation
- Time-of-check to time-of-use errors

•

Malicious Code

- Malicious code runs under the user's authority
  - If the user starts the malicious code
  - But usually without user's explicit permission or knowledge
- Malicious code has been known for a long time
  - Virus behavior reference dates back to 1970's.
- What's new?
  - Types, amount, appearing speed of new exploits
  - More pervasive

Questions on Malicious Code

- How can malicious code take control of a system?
- How can it lodge in a system?
- How does it spread?
- How to detect it?
- How to stop it?
- How to prevent it?

Malicious Code Types

Based on the behavior pattern of malicious code:

- **Virus**: A program that can replicate itself and pass on malicious code to other nonmalicious programs (host program) by modifying them
  - Transient: The virus runs when its host program executes, and terminates when the host program ends.
  - Resident: The virus locates itself in memory, and remains active even after the host program ends.

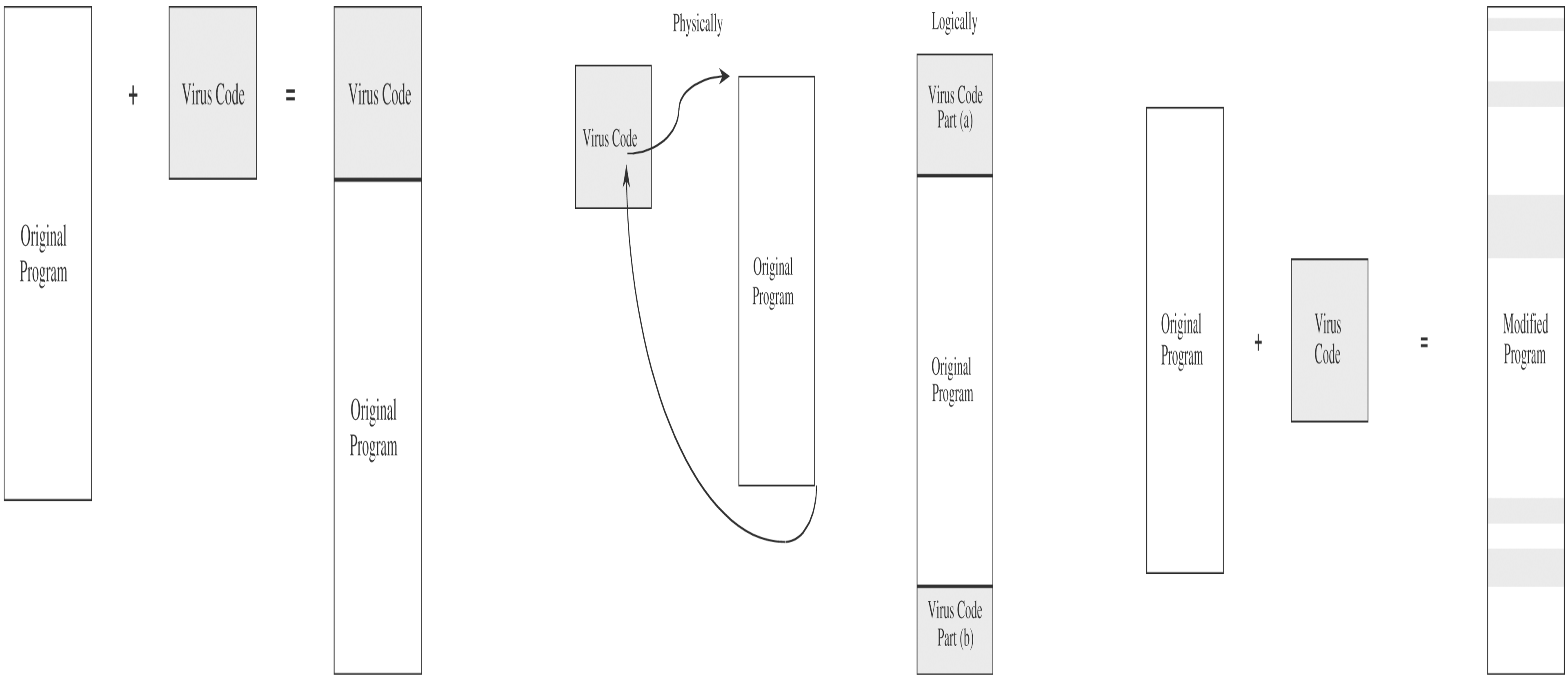
Malicious Code Types (cont'd)

- **Trojan horse**: primary effect + nonobvious malicious effect
- **Logic bomb**: goes off when a specified condition is met
  - Time bomb
- **Trapdoor/backdoor**: a program's nonobvious access point
- **Worm**: program that self-spreads in network
- **Rabbit**: a virus/worm that self-replicates endlessly

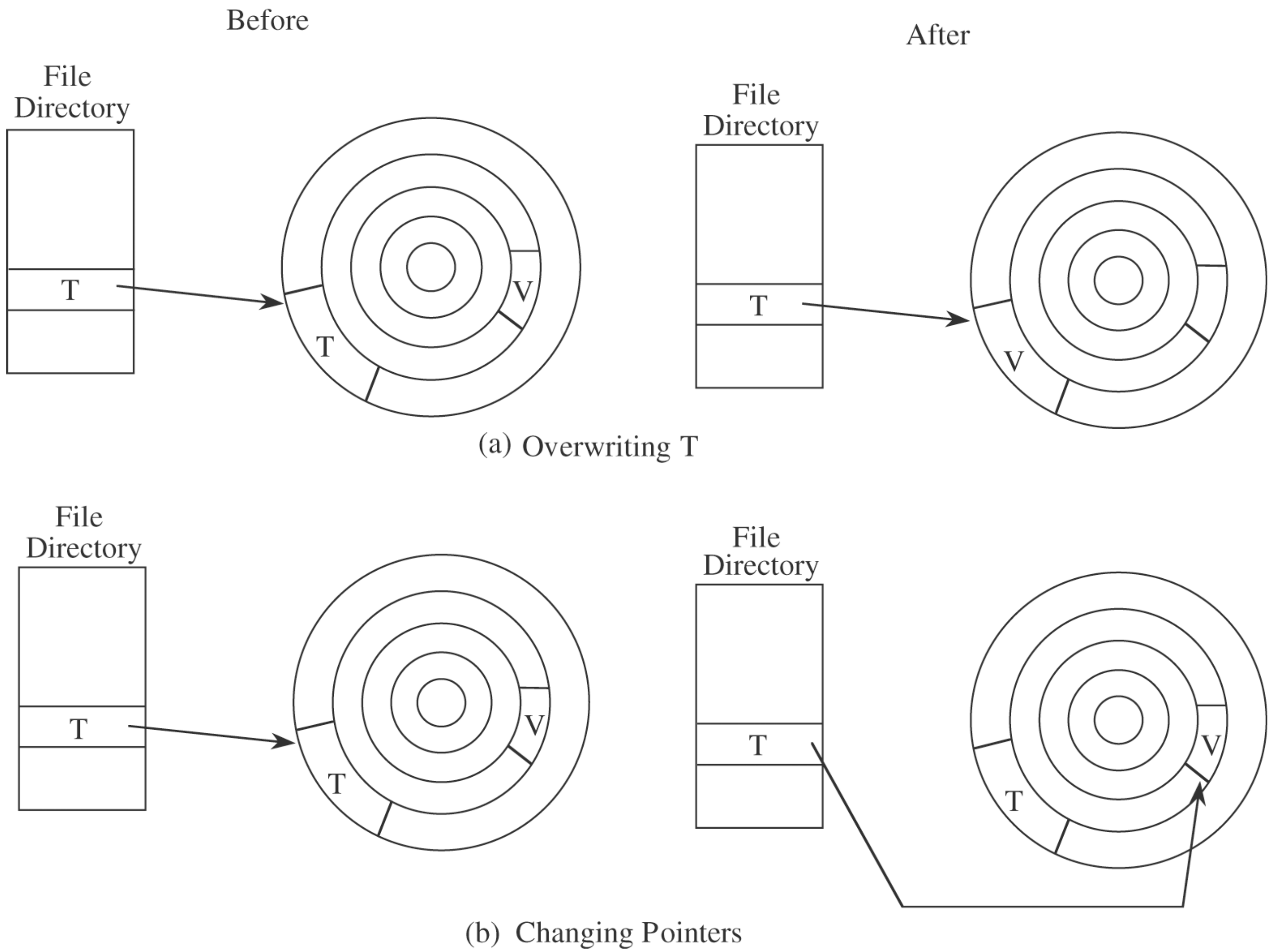
Notation Note

- Sometimes we use “virus” to represent all malicious code

How Viruses Attach

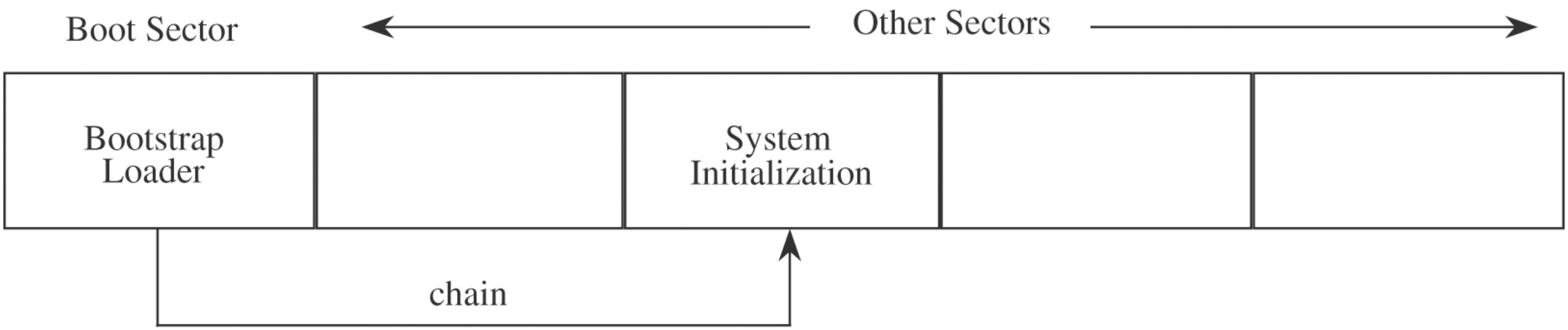


How Viruses Gain Control

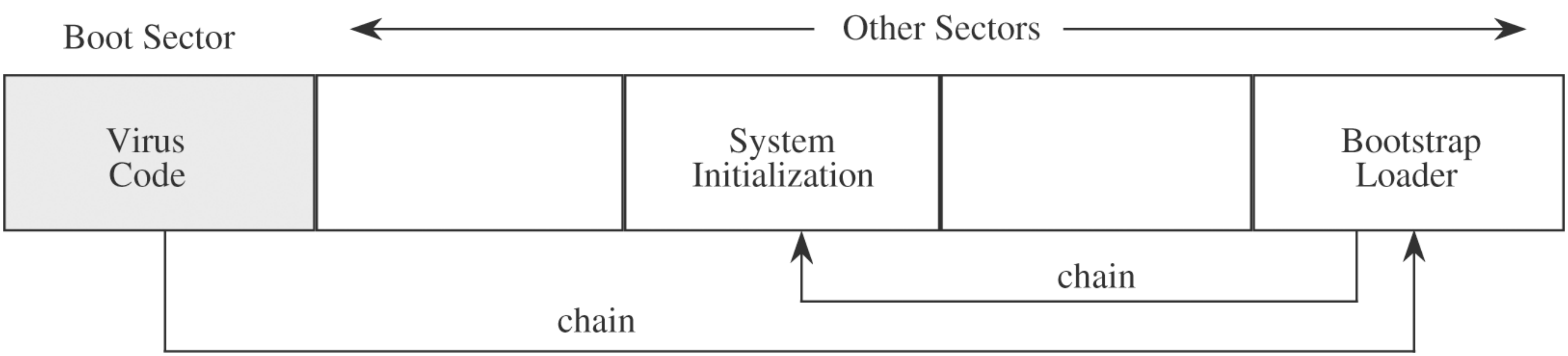


Homes for Viruses

- Applications
  - E-mail attachments
  - “Macros” of word processors and spreadsheets
  - Libraries
- Memory
  - “Terminate and stay resident” (TSR) routines
  - OS’s table of programs to run
    - Windows registry includes programs to run at startup
- Boot Sector



(a) Before infection



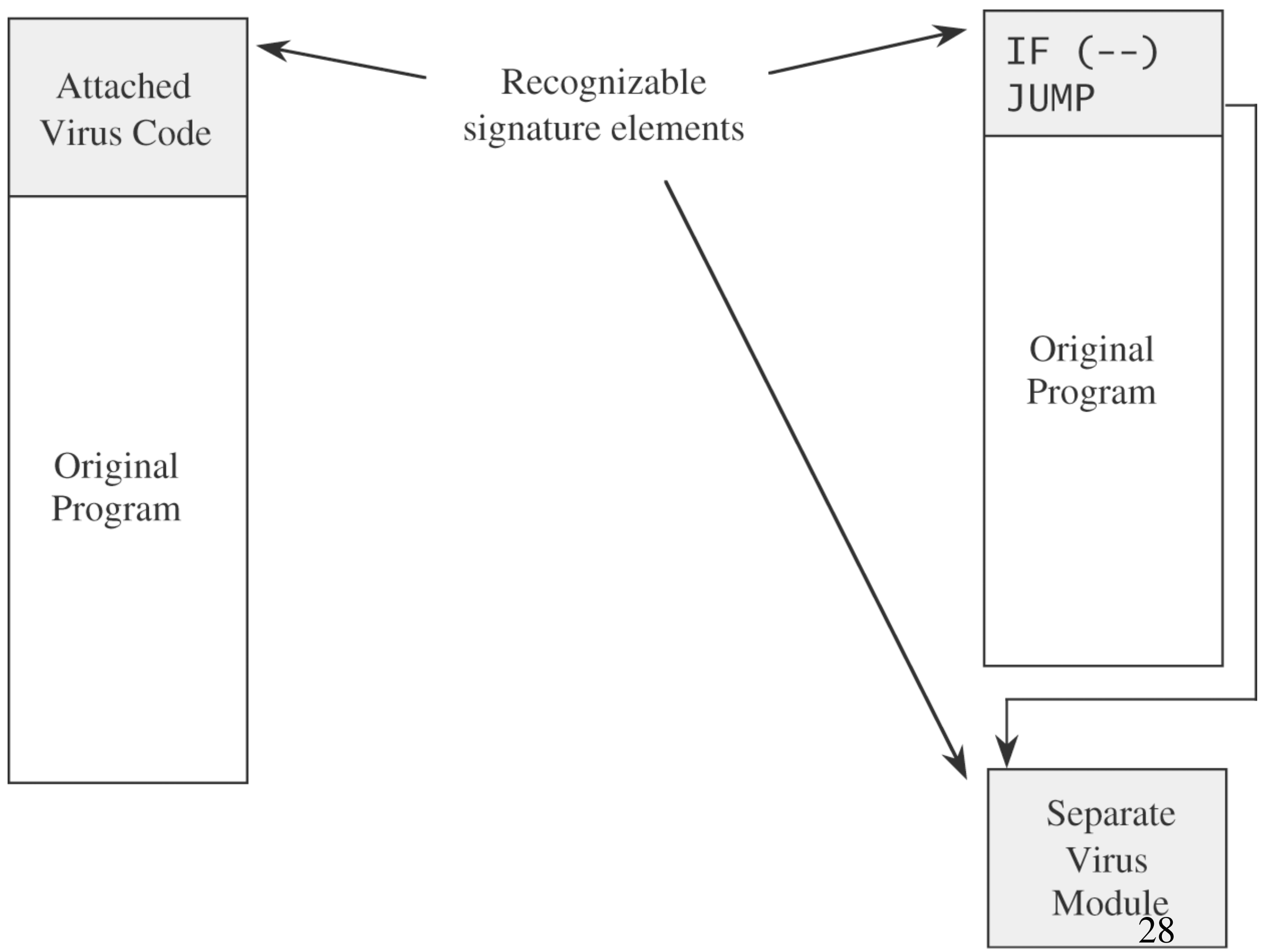
(b) After infection

Virus Detection

- Virus code must be stored somewhere, and must be in memory to execute
- Virus scanner searches memory and disk, monitors execution, and watches for **virus signatures**
  - If a virus is found, block the virus, inform the user, and remove the virus

Virus Signatures (1)

- Storage patterns



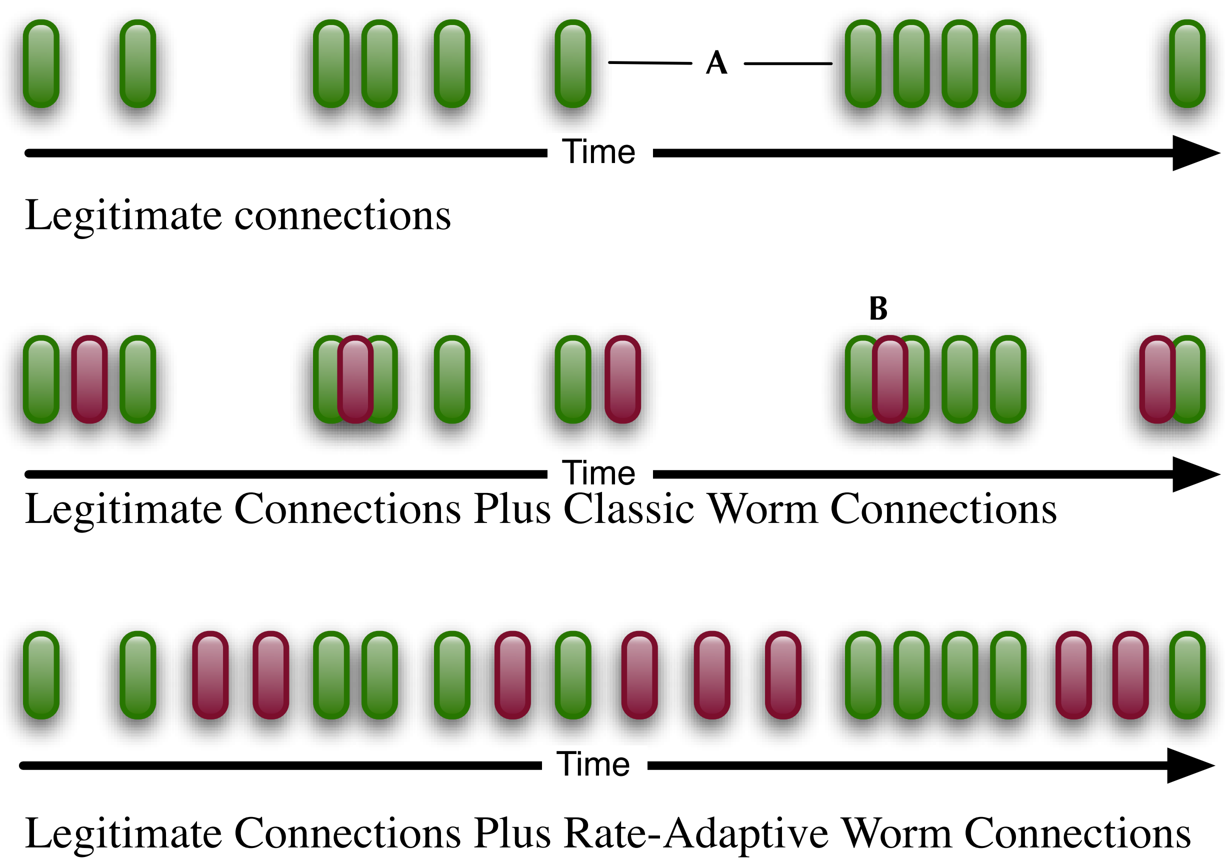
Virus Signatures

- Execution patterns

Virus Effect	How It is Caused
Attach to executable program	• Modify file directory • Write to executable program file
Attach to data or control file	• Modify directory • Rewrite data • Append to data • Append data to self
Remain in memory	• Intercept interrupt by modifying interrupt handler address table • Load self in non-transient memory area
Infect disks	• Intercept interrupt • Intercept operating system call • Modify system file • Modify ordinary executable program
Conceal self	• Intercept system calls that would reveal self and falsify result • Classify self as "hidden" file
Spread infection	• Infect boot sector • Infect system program • Infect ordinary program • Infect data ordinary program reads to control its execution
Prevent deactivation	• Activate before deactivating program and block deactivation • Store copy to reinfect after deactivation

Virus Signatures

- Transmission patterns



Polymorphic Viruses

- Virus signature example: Begins with string 47F0F00E08, and has string 00113FFF at word 12.
- Polymorphic
  - insert *no-ops* instructions
  - Randomly reposition all parts of itself
  - Randomly change all fixed data
  - Encrypted using different keys