COMP4670 Lecture 4: Program Security

Terminology

FAULT: incorrect step, command process, or data definition in a computer program, lead by an *error*. It is the cause of a problem and an inside view of the system, as seen by the eyesn of devs.
FAILURE: The *effect* of the faults, a departure from the system's required behaviour. It is an outside view, the problem a user sees.

Secure Programs

When do we say a program or software is *secure*?

- characteristics of software that contribute to its overall security
- quanitity and types of faults for evidence of a product's quality

Penetrate and Patch

- searching for faults and repairing them by analysts
- patch efforts are largely useless, making the system less secure rather than more secure because they frequently introduce new faults
  - narrow focus on the fault iteself and not on its context
  - fault often has non-obvious side effects in places other than the immediate area
  - fault could not be fixed properly because system functionality or performance would suffer as a consequence

Unexpected Behaviour (Program Security Flaw)

- compare software requirements with the behaviour
- it is based on *flaws* and *vulnerabilities*
  - *flaw* could be a fault or a failure
  - *vulnerability* usually describes a class of flaws

Types of Flaws

Intentional: malicious or non-malicious
Inadvertent:

1. validation error (incomplete or inconsistent): permission checks
2. domain error: controlled access to data
3. serialization and aliasing: program flow order
4. inadequate identification and authentication: basis for authorization
5. boundary condition violation: failure on first or last case
6. other exploitable logic errors

Classic Errors

Buffer Overflows:

- trying to pour two litres of water into a one-litre pitcher: some water is going to spill and make a mess

- user's program data can overflow into system data, user's program code, and system program code

Incomplete Mediation:

- occurs when the application accepts incorrect data from user. ex: `somesite.com/sub/userinput.asp?parm1=(909)555-6969&`
- user can send arbitrary values to the server

Time-of-Check to Time-of-Use Errors:

- serialization or synchronization flaw
- caused by ineffective access control
- informally: when items checked for validity are no longer valid when the item is accessed. Something could happen inbetween the time-of-check and time-of-use

Viruses and Other Malicious Code

- *unanticipated* or *undesired* effects in programs or program parts, caused by an agent intent on damage
- a program that can replicate itself and pass on malicious code to other non-malicious programs by modifying them

Virus:

- **Transient**:
  - has a life that depends on the life of its host
  - virus runs when its attached program executes and terminates when its attached program ends
- **Resident**:
  - locates itself in memory; can remain active or be activated as a stand-alone program, even after its attached program ends

Trojan Horse: has a second, nonobvious malicious effect
Logic Bomb: class of malicious code that detonates when a specified condition occurs
Time Bomb: a logic bomb whose trigger is a time or date
Trapdoor or Backdoor: feature in a program by which someone can access the program other than by the obvious, direct call, perhaps with special privileges
Worm:

- program that spread copies of itself through a network
- primary difference between a worm and a virus is that a worm operates through networks, and a virus can spread through any medium
- spreads copies of itself as a stand-alone program, whereas the virus spreads copies of itself as a program that attaches to or embeds in other programs

Rabbit: virus or worm that self-replicates without bound, with the intention of exhausting some computing resource
Appended Viruses:

- virus code could be a program on the distribution medium
- when program executed, virus could install itself on permanent storage medium
- newer way: as an attachment to an email message, activated when opened

Viruses that surround a program: viruses that run the original program but have control before and after its execution, example:

- attach itself to the program that contructs the listing of files on the disk
- after listing program has generated the listing, the virus could eliminate its entry from the listing and falsify space counts

Integrated Viruses and Replacements:

- virus replaces some of its target, integrating itself into the original code of the target
- virus writer has to know the exact structure of the original program to know where to insert which pieces of the virus

Document Viruses: one popular virus type is called the document virus

- implemented within a formatted document, such as a written document, a database, spreadsheet
- these document types are highly structured files that contain both data (words or numbers) and commands (such as formulas, formatting controls, links)

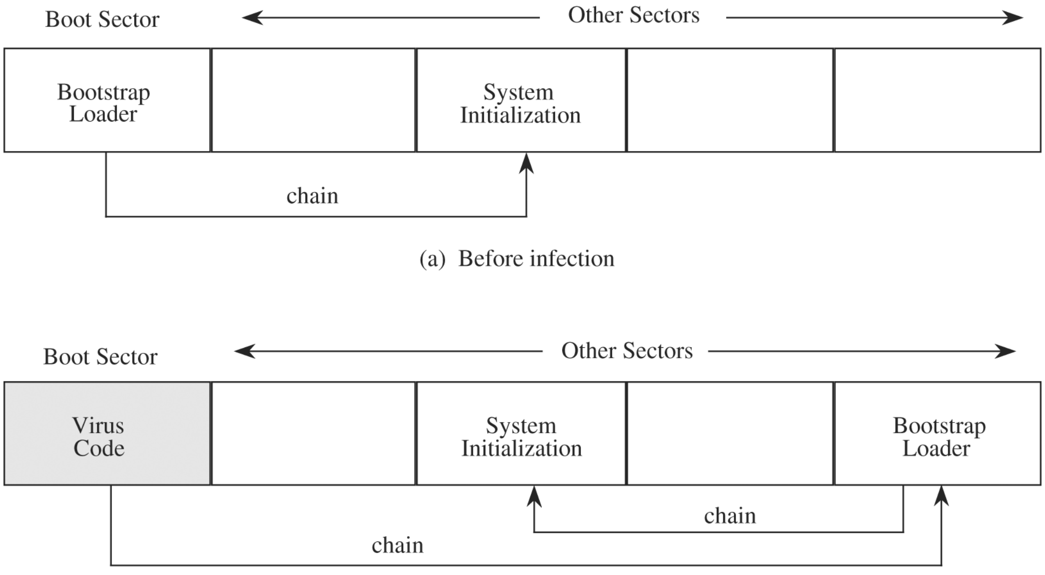
- commands are part of a rich programming language, including macros, variables and procedures, file accesses, system calls
- writer of a document virus uses any of the features of the programming language to perform malicious actions

How Viruses Gain Control

The virus either has to seem to be a valid program T, saying effectively, “I am T”, or the virus has to push T out of the way, becoming a substitute for T, saying effectively, “Call me instead of T”

Where Viruses Reside

One-Time Execution: the majority of viruses today execute only once, spreading their infection and causing their effect in that one execution. A virus often arrives as an email attachment of a document virus
Boot Sector Viruses:

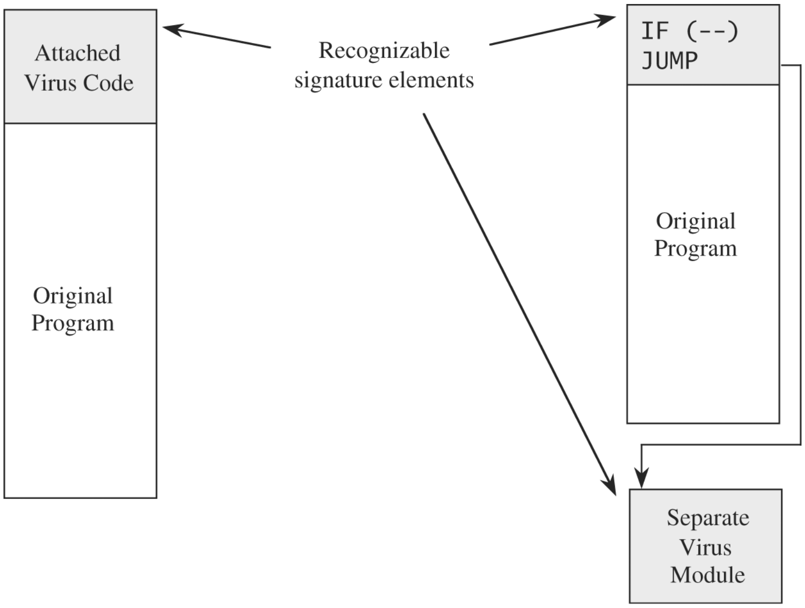


(a) Before infection

(b) After infection

Memory-Resident Virues: resident code is activated many times while the machine is running. Each time the resident code runs, the virus does too.
Macros: With macros, a user can record a series of commands and repeat them with one invocation. Programs can also provide a startup macro that is executed every time the application is executed
Libraries: used by many programs, thus the code in them will have a broad effect. They are often shared among users and transmitted from one user to another, a practice that spreads infection
Virus Signature: Every virus must be stored somewhere, and exectues in a particular way
Virus Scanner: Uses signature to detect and possible remove viruses

Storage Patterns:



Techniques for Building a Safe Electronic Community

- use only commercial software acquired from reliable, well-established vendors
- test all new software on an isolated computer and look for unexpected behaviour
- open attachments only when you know them to be safe
- make recoverable system image and store it safely. Keep image write-protected during reboot
- make and retain backup copies of executable system files and important data files
- use virus detectors regularly and update them daily. Keep detector's signature file up to date

Trapdoors

- undocumented entry point to a module
- devs insert trapdoors during code development
  - to test the module
  - to provide hooks by which to connect future modifications or enhancements
  - to allow access if the module should fail in the future
  - to allow a programmer access to a program once it is placed in production
- useful trapdoors (for audit purposes)
  - must be documented
  - access to them must be strongly controlled
  - must be designed and used with full understanding of the potential consequences

Controls Against Program Threats

Types of Controls

1. Developmental
   - Specify, design, implement, and test the system. Review the system at various stages. Document, manage, and maintain the system
2. Operating System
3. Administrative

Software Development

1. Modulartiy
   - create a design or code in small, self-contained units called components or modules
   - **high cohesion**: elements of a component have a logical and functional reason for being there
   - **low coupling**: loosely coupled components are free from unwitting interference from other components
2. Encapsulation
   - easier to trade a problem
   - easier to maintain the system
   - easier to see where vulnerabilities may lie
3. Information Hiding
   - each component hides its precise implementation or some other design decision from the others (black box)
   - when a change is needed, overall design can remain intact while only necessary changes are made to particular components
4. Security Practices
   - **mutual suspicion**: calling program cannot trust its called sub-procedures to be correct. A called sub-procedure cannot trust its calling program to be correct. Each protects its interface data so the other only has limited access
   - **confinement**: a confined program is strictly limited in what system resources it can access. If a program is not trustworthy, the data it can access are strictly limited
   - **genetic diversity**: it's risky having many components of a system come from one source
5. Other techniques
   - **hazard analysis**: developing hazard lists, as well as procedures for exploring "what if" scenarios to trigger consideration of non-obvious hazards
   - **testing** black-box testing, white-box testing
   - **testing types** unit, integration, function, performance, acceptance, regression, and penetration testing