MALICIOUS CODE
defences

MALICIOUS CODE - BASICS

- ***MALICIOUS CODE (MALWARE)*** is any code added, changed or removed from a software system in order to intentionally cause harm or subvert the intended function of the system
- The problems with malware is steadily increasing due to a number of trends:

- the increased *networking*

- the rising system *complexity*

- system configurations are constantly *changing*

MALICIOUS CODE - DEFENCE PRINCIPLES

There are four main approaches that the host can take to protect itself:

1. **Analyze** the code and reject it - if it may cause harm (pre-check and stop)
2. **Rewrite** the code before executing it - so that it can do no harm. (pre-check and fix)
3. **Monitor** the code execution and stop it - before it does harm. (supervise and stop)
4. **Audit** the code during execution - and recover if it did harm. (check result and recover)

MALICIOUS CODE - DEFENCE PRINCIPLES

Some details and examples:

1. **Analyze** the code and reject it - if it may cause harm (pre-check and stop)
   - scanning for a known virus (and rejecting)
   - dataflow analysis (to detect novel malicious code)
   - analysis to find vulnerabilities (e.g. buffer limitations)

2. **Rewrite** the code before executing it - so that it can do no harm. (pre-check and fix)
   - insert extra code to perform dynamic checks, e.g checking array indices (Java compiler)

MALICIOUS CODE - DEFENCE PRINCIPLES

(cont'd)

3. **Monitor** the code execution and stop it - before it does harm. (supervise and stop)
   - using ***reference monitors (RM)*** is the traditional approach
   - is often done in hardware and included in the OS
   - an on-line RM is JVM interpreter that monitors the execution of applets

4. **Audit** the code during execution - and recover if it did harm. (check result and recover)
   - recovery is only possible if the damage can be properly assessed.
   - requires use of secure auditing tools (logging).

MALICIOUS CODE - TODAY’S DEFENCES

Traditionally, the security policy was enforced using the computer hardware and standard OS mechanisms. Such mechanisms are not easy to expand.

Present defences against malicious code are:

• ***scanning for “malicious” signatures***
- used by anti-virus scanners
- easy to implement
- easy to circumvent by making small changes in signature
- only works for previously known malware

• ***code signing*** *(cryptographic signing)*
- ensures transmission integrity, i.e. that nobody has changed the code during the transmission.
- only means just that. Does no imply that the code is safe, robust or secure. You have to *trust the sender*

MALICIOUS CODE - TOMORROW’S DEFENCES

Promising new defences against malicious code are:

• ***software-based reference monitors***
- present methods to ensure memory safety, i.e. that all memory accesses are correct
- basic idea is to rewrite binary code so that it checks and validates all memory accesses and all control transfers.
- Available tools/methods are:
  SFI = Software-Based Fault Isolation
  IRM = In-line Reference Monitor

MALICIOUS CODE - TOMORROW'S DEFENCES

- ***type-safe languages***
  - ensure that operations are only applied to the appropriate type, i.e. preventing unauthorized code from applying the wrong operations to the wrong values.
  - allows specification of new abstract types that could enforce application-specific access policies
- ***proof-carrying code (PCC)***
  - untrusted code is required to come with an explicit machine-checkable proof that the code is secure (wrt to a specific security policy.)