Forensics, Malware, and Penetration Testing

Introduction to Malware Part 2

Mihai Ordean
University of Birmingham

m.ordean@cs.bham.ac.uk

Ref: Computer Security by Bishop; Introduction to Malware, UT Dallas

Malware detection

1. There is no generic technique that can detect all malicious logic.

2. Defence must focus on individual aspects of the malware.

   Defence in depth is often required!

Defence “strategies”

- Detect alterations
- Distinguish between data and instructions
- Limit resource sharing

Antivirus Programs

- Look for specific sequences of bytes in files, and compare them against known “signatures” (i.e. hash values specific for that sequence of bytes).
  - If found, warn user and/or disinfect file
- Antivirus programs must look for known signatures

- Cannot deal with malware not yet analyzed
  - It’s difficult to determine whether a generic program is a virus or not.

Tools

- Process monitoring, guardians and watchdogs
  - Intercept system requests (e.g. open files)
  - Can determine if file has changed
  - Can decide if access is to be allowed
  - Redefine system (or library) calls
    - e.g. Kaspersy AV had a “network service” that preprocessed network traffic to detect email viruses

Distinguish between data and instructions

- Malicious logic is both:
  - a virus has code written in a program's **data** section, but uses it to execute instructions.

- **data** and **instructions** should be treated as separate types, and require a certifying authority to approve conversion between them.

Linux capabilities

- UNIX implementations distinguish two categories of processes:
  - privileged processes (UID is 0) which bypass all kernel permission checks
  - unprivileged processes (UID is nonzero) which are subject to full permission checking

Linux capabilities

- Divides the privileges traditionally associated with superuser into distinct units i.e. **capabilities**:
  - Applied per thread
  - Divide the power of superuser into pieces, such that if a program that has one or more capabilities is compromised, its power to do damage to the system would be less than the same program running with root privilege.

Linux capabilities

- Kernel must check whether the thread has the required capability

- The kernel must provide system calls allowing a thread's capability sets to be changed and retrieved

- The filesystem must support attaching capabilities to an executable file

Limit sharing, code interaction

Running malicious code in an isolated environment is acceptable as long as any potential compromise can be contained to that environment.

Sandboxing

- A "virtual machine" that has ability to restrict rights
  - Modify program by inserting instructions to cause traps when violation of security policy
  - Replace dynamic load libraries with instrumented routines

Example: Sandboxing and race conditions

- Race conditions can occur when successive system calls operate on object e.g.:
  - Multiple calls identify object by name
  - Attack: bind/rebind name to different object between calls to prevent object tracking
- Fix: sandbox with instrumented calls
  - Unique identifier (e.g. filename) saved on first call
  - On second call, identifier is compared to that of first call. *If they are different signal a potential attack…*

Inhibit code sharing between programs

- Integrity policies should have built in separation
- E.g.,: SELinux
  - Defines contexts for users and processes
    - context <- (username, role, domain)
  - Policies are used to describe the circumstances under which a process is allowed into a certain domain
  - Processes are launched into an explicitly specified context (user, role and domain), but SELinux will deny the transition if it is not approved by the policy.

Conclusion

- Malware is only getting bigger and better
- There are many types of malware and cataloguing their behaviour is difficult
- Mitigation techniques exist, but few are used in practice