Outline for January 15, 2003

Reading: Text, §23.3–23.4

Discussion Problem

A recurring question in computer security is how the discoverer of a vulnerability in a program or computer system should report it to the responsible party (in this context, the vendor of the program or system). The SANS Organization has proposed the following, which they are calling the Fisher Plan. This description is from SANS NewsBites, Vol. 4, Num. 50 (Dec. 12, 2002):

> There would be established a reporting center for new vulnerabilities, either inside the government or outside (that's one of the things that needs to be decided) along with reporting guidelines that required sufficient data to ensure the person doing the reporting has found something real. All reports will be recorded and immediately passed both to the vendor (which may have already received it from the person who found it) and multiple government or government-funded centers of excellence such as CERT/CC and the research group of the National Infrastructure Protection Center.
>
> The centers of excellence would identify how critical the vulnerability might be and would set a priority for correcting the problem. (The scale is yet to be determined but can be modeled after SANS Critical Vulnerability Analysis rating scale or the CERT/CC rating scale or both).
>
> Government officials will monitor the vendor's progress and exert appropriate high-level pressure on the vendors for rapid response of important vulnerabilities.
>
> When a method of eliminating the vulnerability is found, it will be published by the vendor and at the same time, the person or organization that found the vulnerability will be awarded both public recognition and a sum of money which may come from the government or may be provided by the SANS Institute. (Financial remuneration is controversial; your feedback would be appreciated.)

What are the good points about this plan? What are its drawbacks?

Outline for the Day

1. Vulnerability Models
   a. PA model
   b. RISOS
   c. NRL
   d. Aslam
2. RISOS
   a. Goal: Aid managers, others in understanding security issues in OSes, and work required to make them more secure
   b. Incomplete parameter validation - failing to check that a parameter used as an array index is in the range of the array;
   c. Inconsistent parameter validation - if a routine allowing shared access to files accepts blanks in a file name, but no other file manipulation routine (such as a routine to revoke shared access) will accept them;
   d. Implicit sharing of privileged/confidential data - sending information by modulating the load average of the system;
   e. Asynchronous validation/Inadequate serialization - checking a file for access permission and opening it non-atomically, thereby allowing another process to change the binding of the name to the data between the check and the open;
   f. Inadequate identification/authentication/authorization - running a system program identified only by name, and having a different program with the same name executed;
   g. Violable prohibition/limit - being able to manipulate data outside one's protection domain; and
   h. Exploitable logic error - preventing a program from opening a critical file, causing the program to execute an error routine that gives the user unauthorized rights.
3. PA Model (Neumann's organization)
   a. Goal: develop techniques to search for vulnerabilites that less experienced people could use

   b. Improper protection (initialization and enforcement)
      i. improper choice of initial protection domain - "incorrect initial assignment of security or integrity level at system initialization or generation; a security critical function manipulating critical data directly accessible to the user";
      ii. improper isolation of implementation detail - allowing users to bypass operating system controls and write to absolute input/output addresses; direct manipulation of a "hidden" data structure such as a directory file being written to as if it were a regular file; drawing inferences from paging activity
      iii. improper change - the "time-of-check to time-of-use" flaw; changing a parameter unexpectedly;
      iv. improper naming - allowing two different objects to have the same name, resulting in confusion over which is referenced;
      v. improper deallocation or deletion - leaving old data in memory deallocated by one process and reallocated to another process, enabling the second process to access the information used by the first; failing to end a session properly
   c. Improper validation - not checking critical conditions and parameters, leading to a process' addressing memory not in its memory space by referencing through an out-of-bounds pointer value; allowing type clashes; overflows
   d. Improper synchronization
      i. improper indivisibility - interrupting atomic operations (*e.g.* locking); cache inconsistency
      ii. improper sequencing - allowing actions in an incorrect order (*e.g.* reading during writing)
   e. Improper choice of operand or operation - using unfair scheduling algorithms that block certain processes or users from running; using the wrong function or wrong arguments.
   f. Analysis procedure
      i. Collect descriptions of protection patterns
      ii. Convert to raw error patterns
      iii. Abstract into system-independent components
      iv. Determine which features in the OS code are relevant, and abstract relevant contexts of those features
      v. Compare the combinations of the relevant features in the OS with generic error patterns

4. NRL
   a. Goal: Find out how vulnerabilities enter the system, when they enter the system, and where they are
   b. Axis 1: inadvertent (RISOS classes) vs. intentional (malicious/nonmalicious)
   c. Axis 2: time of introduction (development, maintenance, operation)
   d. Axis 3: location (hardware, software: OS, support utilities, applications)
5. Aslam
   a. Goal: Treat vulnerabilities as faults
   b. Coding faults: introduced during software development
      i. Synchronization errors
      ii. Validation errors
   c. Emergent faults: introduced by incorrect initialization, use, or application
      i. Configuration errors
      ii. Environment faults
   d. Introduced decision procedure to classify vulnerabilities in exactly one category
6. Common Implementation Vulnerabilities
   a. Unknown interaction with other system components (DNS entry with bad names, assuming finger port is finger and not chargen)
   b. Overflow (year 2000, *lpr* overwriting flaw, *sendmail* large integer flaw, *su* buffer overflow)
   c. Race conditions (*xterm* flaw, *ps* flaw)
   d. Environment variables (*vi* one-upsmanship, *loadmodule*)
   e. Not resetting privileges (Purdue Games incident)