Lecture 13

Vulnerability

Wholeness Statement

A vulnerability is a failure of security policies, procedures, and controls that could allow a subject to violate security policy. Penetration tests are designed to find and exploit vulnerabilities. A vulnerability is a weakness. Similarly all problems are caused by weakness due to violation of natural law. Violation of natural law causes stress. Stress in an individual's life is the cause of sickness and suffering, and the build-up of stress among all the individuals in society is the cause of crime, violence, conflict, and war. Thus spontaneous alliance with natural law would solve all such problems.

Overview

- 1. What is a vulnerability?
- 2. Penetration studies
- 3. Flaw Hypothesis Methodology
- 4. Vulnerability examples (including buffer overflow)
- 5. Vulnerability classification schemes

Definitions

- Vulnerability, security flaw: failure of security policies, procedures, and controls that allow a subject to commit an action that violates the security policy
- – Subject is called an attacker
- – Using the failure to violate the policy is exploiting the vulnerability or breaking in
- Comparison of Formal Verification and Penetration Testing

Formal Verification

- Mathematically verifying that a system satisfies certain constraints
- *Preconditions* state assumptions about the system
- *Postconditions* are result of applying system operations to preconditions, inputs
- Required: postconditions satisfy constraints

Formal Specifications

```
/*@ requires x >= 0 && y > 0
  @ ensures  0 <= \result && \result < y
  @  && (\exists int d;;x == d*y + \result);
  @*/
int math_mod(int x, int y) { ... }
```

Penetration Testing

- Testing to verify that a system satisfies certain constraints
- Hypothesis stating system characteristics, environment, and state relevant to vulnerability
- Result is compromised system state
- Apply tests to try to move system from state in hypothesis to compromised system state

Notes

- Penetration testing is a *testing* technique, not a verification technique
  - It can prove the *presence* of vulnerabilities, but not the *absence* of vulnerabilities
- For formal verification to prove absence, proof and preconditions must include *all* external factors
  - Realistically, formal verification proves absence of flaws within a particular program, design, or environment and not the absence of flaws in a computer system (think incorrect configurations, etc.)

Comparisons

Formal Verification	Penetration Testing
Preconditions	System characteristics, environment, and state
Program	Program or system
Postconditions	System state

Penetration Testing

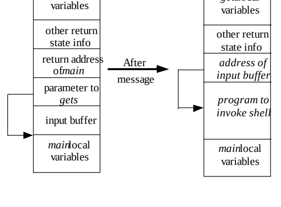
- When you do penetration testing make sure you have been authorized
- And make sure to let those who need to know about any vulnerabilities so corrective action can be taken

Example Flaw: *fingerd*

- Exploited by Internet Worm of 1988
  - Recurs in many places, even now
- *finger* client sends request for information to server *fingerd* (*finger* daemon)
  - Request is name of at most 512 chars
  - What happens if you send more?

Buffer Overflow

- Extra chars overwrite rest of stack, as shown
- Can make those chars change return address to point to beginning of buffer
- If buffer contains small program to spawn shell, attacker gets shell on target system



Main Point

1. A buffer overflow attack causes data to flow outside the boundaries of the memory reserved for it. Here breaking boundaries is bad. However, breaking boundaries to our thinking is good because it allows us to have access to the field of all possibilities. TM gives us access to this field.

Penetration Studies

- Authorized test for evaluating the strengths and effectiveness of all security controls on system
  - Also called *tiger team attack* or *red team attack*
  - Goal: violate site security policy
  - Not a replacement for careful design, implementation, and structured testing
  - Tests system *in toto*, once it is in place
    - Includes procedural, operational controls as well as technological ones

Goals

- Attempt to violate specific constraints in security and/or integrity policy
  - Implies metric for determining success
  - Must be well-defined
- Example: subsystem designed to allow owner to require others to give password before accessing file (i.e., password protect files)
  - Goal: test this control
  - Metric: did testers get access either without a password or by gaining unauthorized access to a password?

Goals

- Find some number of vulnerabilities, or vulnerabilities within a period of time
  - If vulnerabilities categorized and studied, can draw conclusions about care taken in design, implementation, and operation
  - Otherwise, list helpful in closing holes but not more
- Example: vendor gets confidential documents, 30 days later publishes them on web
  - Goal: obtain access to such a file; you have 30 days
  - Alternate goal: gain access to files; no time limit (a Trojan horse would give access for over 30 days)

Layering of Tests

1. External attacker with no knowledge of system
   - Locate system, learn enough to be able to access it
2. External attacker with access to system
   - Can log in, or access network servers
   - Often try to expand level of access
3. Internal attacker with access to system
   - Testers are authorized users with restricted accounts (like ordinary users)
   - Typical goal is to gain unauthorized privileges or information

Layering of Tests (con't)

- Studies conducted from attacker's point of view
- Environment is that in which attacker would function
- If information about a particular layer irrelevant, layer can be skipped
  - Example: penetration testing during design, development skips layer 1
  - Example: penetration test on system with guest account usually skips layer 2

Methodology

- Usefulness of penetration study comes from documentation, conclusions
  - Indicates whether flaws are endemic or not
  - It does not come from success or failure of attempted penetration
- Degree of penetration's success also a factor
  - In some situations, obtaining access to unprivileged account may be less successful than obtaining access to privileged account

Steps of Flaw Hypothesis Methodology

1. Information gathering
2. Flaw hypothesis
3. Flaw testing (do no harm!)
4. Flaw generalization
5. Flaw elimination (*maybe*)

Flaw Hypothesis Methodology

1. Information gathering
   - Become familiar with system's functioning
   - Read manuals and specifications! (principle of open design)
   - Devise model of system and/or components
   - Review management of system looking for potential people problems (are policies being followed)
   - Look at how system manages privileged users
2. Flaw hypothesis
   - Draw on knowledge to hypothesize vulnerabilities
   - Examine policies and procedures
     - Try omitting steps in procedures
   - Examine implementations
     - If manual indicates a maximum length of some field, try a longer length
   - Result is list of possible flaws

Flaw Hypothesis Methodology (cont.)

3. Flaw testing (do no harm!)
   - Assign priorities and test out hypotheses (e.g. focus on external attack rather than inside job such as external access protocols and programs)
   - Make sure system is backed up in case some test goes too far.
   - Avoid actually exploiting the flaw unless management doesn't believe the flaw exists.
   - As with any test, it must be as simple as possible and must be repeatable
4. Flaw generalization
   - Generalize vulnerability to find others like it
   - For example, if an account with a default password is found, generalize to two things:
     1. Users poorly educated about password management
     2. There may be other accounts with default passwords

Flaw Hypothesis Methodology (cont.)

5. Flaw elimination (*maybe*)
   - Usually not included since testers are not best people to fix these flaws
     - Designers and implementers are
   - Requires understanding of context, details of flaw including environment, etc.
   - If flaw discovered during development of the system, it should be corrected then
   - If discovered after system is in production, need to block paths to flaw until patch can be created.

Main Point

2. The Flaw Hypothesis Methodology has five steps to systemize the finding of vulnerabilities. The steps of progress only require two steps: rest and activity. The steps of the Flaw Hypothesis Methodology are substeps of activity.

Michigan Terminal System

- General-purpose OS running on IBM 360, 370 systems
- Class exercise: gain access to terminal control structures
  - Had approval and support of center staff
  - Began with authorized account (level 3)

Step 1a: Information Gathering

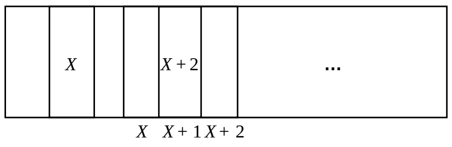
- Learn details of system's control flow and supervisor
  - When program ran, memory split into segments
  - 0-4: supervisor, system programs, system state
    - Protected by hardware mechanisms
  - 5: system work area, process-specific information including privilege level
    - Process should not be able to alter this
  - 6 on: user process information
    - Process can alter these
- Focus on segment 5

Step 1b: Information Gathering

- Segment 5 protected by virtual memory protection system
  - System mode: process can access, alter data in segment 5, and issue calls to supervisor
  - User mode: segment 5 not present in process address space (and so can't be modified)
- Run in user mode when user code being executed
- User code issues system call, which in turn issues supervisor call

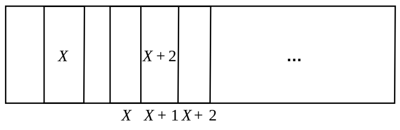
How to Make a Supervisor Call

- System code checks parameters to ensure supervisor accesses authorized locations only
  - Parameters passed as list of addresses (*X*, *X*+1, *X*+2) constructed in user segment
  - Address of list (*X*) passed via register



Step 2: Flaw Hypothesis

- Consider switch from user to system mode
  - System mode requires supervisor privileges
- Found: a parameter could point to another element in parameter list
  - Below: address in location *X*+1 is that of parameter at *X*+2
  - Means: system or supervisor procedure could alter parameter's address *after* checking validity of old address



Step 3a: Flaw Testing

- Find a system routine that:
  - Used this calling convention;
  - Took at least 2 parameters and altered 1
  - Could be made to change parameter to any value (such as an address in segment 5)
- Chose line input routine
  - Returns line number, length of line, line read
- Setup:
  - Set address for storing line number to be address of line length

Step 3b: Flaw Testing

- System routine validated all parameter addresses
  - All were indeed in user segment
- Supervisor read input line
  - Line length set to value to be written into segment 5
- Line number stored in parameter list
  - Line number was set to be address in segment 5
- When line read, line length written into location address of which was in parameter list
  - So it overwrote value in segment 5

Step 4: Flaw Generalization

- Could not overwrite anything in segments 0-4
  - Protected by hardware
- Testers realized that privilege level in segment 5 controlled ability to issue supervisor calls (as opposed to system calls)
  - And one such call turned off hardware protection for segments 0-4 …
- Effect: this flaw allowed attackers to alter anything in memory, thereby completely controlling computer

Penetration of a Corporate Computer System

Penetration Test

- Goal: determine whether corporate security measures were effective in keeping external attackers from accessing system
- Testers focused on policies and procedures
  - Both technical and non-technical

Step 1a: Information Gathering

- Searched Internet
  - Got names of employees, officials
  - Got telephone number of local branch, and from them got copy of annual report
- Constructed much of the company's organization from this data
  - Including list of some projects on which individuals were working

Step 1b: Get Telephone Directory

- Corporate directory would give more needed information about structure
  - Tester impersonated new employee
    - Learned two numbers needed to have something delivered off-site: employee number of person requesting shipment, and employee's Cost Center number
  - Testers called secretary of executive they knew most about
    - One impersonated an employee, got executive's employee number
    - Another impersonated auditor, got Cost Center number
  - Had corporate directory sent to off-site "subcontractor"

Step 2: Flaw Hypothesis

- Controls blocking people giving passwords away not fully communicated to new employees
  - Testers impersonated secretary of senior executive
    - Called appropriate office
    - Claimed senior executive upset he had not been given names of employees hired that week
    - Got the names

Step 3: Flaw Testing

- Testers called newly hired people
  - Claimed to be with computer center
  - Provided “Computer Security Awareness Briefing” over phone
  - During this, learned:
    - Types of comnputer systems used
    - Employees’ numbers, logins, and passwords
- Called computer center to get modem numbers
  - These bypassed corporate firewalls
- Success

Step 4: Flaw Generalization

- Security education not valued in company

Debate

- How valid are these tests?
  - Not a substitute for good, thorough specification, rigorous design, careful and correct implementation, meticulous testing
  - Very valuable *a posteriori* testing technique
    - Ideally unnecessary, but in practice very necessary
- Finds errors introduced due to interactions with users, environment
  - Especially errors from incorrect maintenance and operation
  - Examines system, site through eyes of attacker

Problems

- Flaw Hypothesis Methodology depends on caliber of testers to hypothesize and generalize flaws
- Flaw Hypothesis Methodology does not provide a way to examine system systematically
  - Flaw taxonomies help here (look for flaws found in other systems)

Flaw Taxonomy

- In 1992, Landwehr, Bull, McDermott, and Choi developed a taxonomy to help designers and operators of systems enforce security.
- This can also be viewed as a checklist for penetration testers.
- See \\10.10.10.124\Demos\CS466\FlawTaxonomy.pdf,
- Skim over the list and slides that follow:

1. How did flaw enter system?

Intentional
- Malicious
  - Trojan horse
  - Trapdoor
  - Logic/time bomb
- Non-malicious
  - Covert channel

Inadvertent
- Validation error
- ...

2. When did flaw enter system?

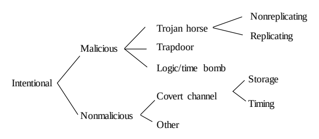
- During development
- During maintenance
- During operation (modification by viruses)

3. Where in the system does the flaw occur?

Software
- Operating system
- Support (compilers, etc.)
- Application (user programs)

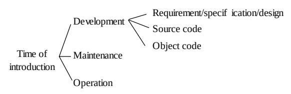
Hardware

Genesis of Flaws



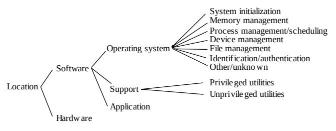
- Inadvertent (unintentional) flaws classified using RISOS categories; not shown above
  - If most inadvertent, better design/coding reviews needed
  - If most intentional, need to hire more trustworthy developers and do more security-related testing

Time of Flaws



- Development phase: all activities up to release of initial version of software
- Maintenance phase: all activities leading to changes in software performed under configuration control
- Operation phase: all activities involving patching and not under configuration control

Location of Flaw



- Focus effort on locations where most flaws occur, or where most serious flaws occur

Key Points

- Given large numbers of non-secure systems in use now, unrealistic to expect less vulnerable systems to replace them
- Penetration studies are effective tests of systems provided the test goals are known and tests are structured well
- Vulnerability classification schemes aid in flaw generalization and hypothesis

Using Google to help with penetration testing

- If classified information from the site you are penetration testing is found on Google, this means that somebody penetrated the site!!
- So one trick that penetration testers can use is to search Google for classified information of the company.

- See the book
  - Google Hacking for Penetration Testers

Main Point

3. Fuzz testing (providing random input data) is a technique to discover buffer overflows. Techniques produce effectiveness in action. TM is a technique that provides the deep rest and contact with pure consciousness that increases the effectiveness of all other techniques.

CONNECTING THE PARTS OF KNOWLEDGE WITH THE WHOLENESS OF KNOWLEDGE

1. Let's bang on the system and see what happens.
2. Let's employ the Flaw Hypothesis Methodology and see what happens.

3. <u>Transcendental Consciousness</u> is the field of all possibilities.
4. <u>Wholeness moving within itself</u>: in Unity Consciousness one has total knowledge of the object, namely it is a manifestation of one's own Self.