CS 392/681 - Computer Security

Module 16 – Vulnerability Analysis
Homework 5 due tonight
Homework 6 posted
Read Chapter 23
Some definitions

- **Computer system**
  - A computer system consists of software, hardware, policies and people

- **Vulnerability**
  - Failure or weakness in a computer system that leads to violations of its policies, procedures and integrity

- **Exploit**
  - Use of vulnerability to violate the policies, procedures

- **Attacker**
  - One who attempts to exploit a vulnerability
Two types of Vulnerability Testing

- **Formal Verification:**
  - In formal verifications preconditions give the constrains of the state of the system when the program is run, and the postconditions state the effect of running the program.
  - Formal verification can theoretically prove the absence of vulnerabilities.

- **Penetration Testing:**
  - In penetration testing preconditions describe the state of the system in which hypothesized security flaw can be exploited and the postconditions are the result of testing.
  - If the postconditions are inconsistent with the security policies the hypothesis is correct- in other words a vulnerability exists.
  - Cannot prove the absence of vulnerabilities but can only prove their existence.
Why Penetration Analysis?

- If formal verification can prove existence of vulnerabilities why not use it instead of penetration testing?
- Formal verification is applied to software components.
- Formal verifications do not capture flaws in the policies, procedures and people!
- Penetration analysis attempts to capture flaws in the whole computer system.
- However, penetration analysis is not a substitute to rigorous design and implementation of computer systems.
Penetration Testing (PenTest)

- Also known as
  - Penetration analysis (PentAn)
  - Tiger team attack
  - Read team [Blue team]

- Ultimate goal of the test is to violate a site’s security policies

- However, usefulness of the study comes from degree of penetration, documentation and conclusions of the study not from the success or failure of the attempted penetration
  - For example: successful compromise of user data is not equivalent to privilege escalation in a computer system
Layered Model for Penetration Testing

1. External attacker with no knowledge
   - Testers know the target exists
   - Have enough information to identify it
   - No access, they have to figure this out through social engineering
   - Cannot get information about security of the system
   - Usually not part of penetration testing

2. External attacker with access to the system
   - Attackers have access to the system
   - Either use an access mechanism to login
   - Usual attacks: guessing passwords, look for unprotected accounts (guest)
   - Attack network servers
   - Usually exploits implementation flaws
Layered model contd.

3. Internal attacker with access to system
   - Testers have an authorized account on the system
   - Tests involve gaining unauthorized privileges
   - Exploits design, operational and implementation flaws
Flaw Hypothesis Methodology

- Penetration testing methodology springs from flaw hypothesis methodology
- Developed by System Development Cooperation and provides a framework for penetration studies
  - Information gathering
  - Flaw hypothesis
  - Flaw testing
  - Flaw generalization
  - Flaw elimination
Information Gathering

- Design and implementation of the system is analyzed
  - Design specifications and manuals
  - Operating system
  - Services
  - Users
- Fingerprinting: Identifying system parameters
  - Active fingerprinting
  - Passive fingerprinting
- Testers should identify policies that govern the system
Flaw Hypothesis

- Testers use the knowledge gathered in previous step
- Identify flaws in similar systems
  - Prior records of testing
  - CERT documents
  - Newsgroups
  - Faulty configurations and defaults
- Hypothesize flaws in current system
Flaw Testing

- Prioritize the flaw hypothesis
  - Priority is a function of goals of the test
    - External access to system
    - Privilege escalation
    - Internal access

- Demonstrate the flaw either through analysis or proof of concept
  - Proof of concept code segments can be found on the Internet, Newsgroups etc.
Flaw Generalization

- Generalize the flaws by categorization
  - Buffer overflows
  - Setuid flaws
  - Etc.

- Different category flaws alone may be benign but a combined flaw can be devastating

- See Secure Programming lectures for examples

- Analyze the system for similar flaws or flaws of same family
Flaw Elimination

- Vulnerability analysis, testing usually does not fix flaws
- Recommendations are made to development team based on flaws and exploits
Goals of VA

- Develop ability to specify, design and implement information systems without vulnerabilities
- Develop techniques to analyze information systems for vulnerabilities
- Develop ability to address any vulnerabilities introduced during the operation of the information system
- Develop techniques to detect attempted exploits
VA Frameworks

- Frameworks are abstractions developed by classifiers to achieve certain goals.
- Frameworks present different perspectives.
- Frameworks we will cover
  - Research Into Secure Operating System (RISOS)
  - Protection Analysis Model
  - NRL Taxonomy
  - Aslam’s Model
Goal: help information systems managers understand security issues in OS and help estimate the effort required to enhance system security

Flaw Classes
1. Incomplete parameter validation
2. Inconsistent parameter validation
3. Implicit sharing of privileged data
4. Asynchronous validation
5. Inadequate authentication
6. Violable prohibition
7. Exploitable logic error
RISOS Classes

- Incomplete parameter validation
  - Parameters are not checked properly before use
  - Check parameters for type, value, access permissions and their presence or absence
  - E.g: buffer overflows

- Inconsistent parameter validation
  - Data validation is done on inconsistent data format
  - E.g: delimiters of one routine may be data to another

- Implicit sharing of privileged data
  - Failure to isolate processes and users properly
  - E.g: password guessing on TENEX

- Asynchronous validation
  - Race conditions
RISOS Classes

- Inadequate authentication
  - Failure to properly authenticating users
  - E.g: guest accounts, Trojan Horses

- Violable prohibition
  - Failure to handle bounds conditions properly
  - E.g: failure in array bound checks, pointer arithmetic

- Exploitable logic error
  - Flaws that don’t fall into any of the above classes
Protection Analysis Model

- Goal: attempted to break the operating system into smaller manageable pieces to reduce the required expertise in dealing with operating systems
Protection Analysis Model

- Flaw classes
  - Improper protection domain initialization and enforcement
    - Improper choice of initial protection domain
    - Improper isolation of implementation detail
    - Improper change
    - Improper naming
    - Improper deallocation or deletion
  - Improper validation
  - Improper synchronization
    - Improper indivisibility
    - Improper sequencing
  - Improper choice of operand or operation
Protection Analysis Model

- Improper protection domain initialization and enforcement
  - Flaws raising from initialization of the system or programs and enforcement of security requirements
  - Improper choice of initial protection domain
    - Bad umask values
  - Improper isolation of implementation details
    - Failure to provide proper abstractions
  - Improper change
    - Change in property bindings. E.g: access-open calls
  - Improper naming
    - Collisions in names of objects in a system
    - E.g: naming a user file ‘ls’
  - Improper deallocation
    - Inadequate garbage collection E.g: failure to free memory
Protection Analysis Model

- Improper validation
  - Improper validation of parameters and data
  - E.g: bounds checking– buffer overflows

- Improper synchronization
  - Occurs when designers fail to synchronize events properly
  - Improper indivisibility
    - Failure to provide atomic operations
  - Improper sequencing
    - Inadequate logic sequences in events
    - E.g: flaws in some one-time password schemes

- Improper choice of operand or operation
  - Inappropriate or erroneous calls to functions
  - E.g: calling rand() without seeding it!
The NRL Taxonomy

- Goal: a taxonomy to help designers and operators of the system enforce security.
- Three questions to answer...
  - How did the flaw enter the system?
  - When did it enter the system?
  - Where in the system is it manifest?
- Built three different sets of flaw classes to answer the above questions
  - Flaw by Genesis
  - Flaw by Time of Introduction
  - Flaw by Location
NRL: Flaws by Genesis

- Malicious
  - Trojan Horse
    - Non-replicating
    - Replicating
  - Trapdoor
  - Logic/Time Bomb

- Non-Malicious
  - Covert Channel
    - Storage
    - Timing
  - Other
NRL: Flaws by Time of Introduction

- Development
  - Requirements
  - Source Code
  - Object Code
- Maintenance
- Operation
NRL: Flaws by Location

- **Software**
  - Operating System
    - System initialization
    - Memory management
    - Process management
    - Device management
    - File management
    - Authentication
    - Other/unknown
  - Support
    - Privileged utilities
    - Unprivileged utilities
  - Application
- **Hardware**
  - Faulty APIs
Aslam’s Model

- Goal: Group security flaws in Unix by similar faults without ambiguity to store in databases

- Flaw Classes
  - Coding faults
    - Synchronization errors
    - Condition validation errors
  - Emergence faults
    - Configuration errors
    - Environment faults
Gupta’s and Gligor’s Theory

- A formal analysis technique to detect vulnerabilities arising from failure to perform adequate checks

- Hypothesis
  - Of Penetration Patterns
    “System flaws that cause a large class of penetration patterns can be identified in system source code as incorrect/absent condition checks or integrated flaws that violate the intentions of the system designers”
  - Of Penetration-Resistant Systems
    “A system is largely resistant to penetration if it adheres to a specific set of design properties”

- Some properties chosen to formalize the model
Properties chosen in the model

- **System isolation and tamperproofness**
  - User must not be able to tamper with the system
  - System interaction must be through well-defined abstractions

- **System noncircumventability**
  - System checks all references to objects
  - User cannot circumvent any system checks

- **Consistency of global objects of the system**
  - With respect to timing and storage objects are consistent

- **Elimination of undesirable system and user dependencies**
Further readings

- Chapter 23
- A Taxonomy of Security Faults in the Unix Operating System, Tamir Aslam
- Vulnerabilities Analysis, Matt Bishop
- Use of A Taxonomy of Security Faults, Taimur Aslam, Ivan Krsul, Eugene Spafford