Lecture 16
November 1, 2023
Noisy vs. Noiseless

• Noiseless: covert channel uses resource available only to sender, receiver

• Noisy: covert channel uses resource available to others as well as to sender, receiver
  • Idea is that others can contribute extraneous information that receiver must filter out to “read” sender’s communication
Defending Against Covert Channels

• Add lots of noise
  • The idea is to prevent the receiver from being able to pick up the signal the sender is sending

• Make the events regular
  • Similar to adding noise, this hides the signal in the regularity
Vulnerability Classification

• Describe flaws from differing perspectives
  • Exploit-oriented
  • Hardware, software, interface-oriented

• Goals vary; common ones are:
  • Specify, design, implement computer system without vulnerabilities
  • Analyze computer system to detect vulnerabilities
  • Address any vulnerabilities introduced during system operation
  • Detect attempted exploitations of vulnerabilities
Example Flaws

• Use these to compare classification schemes
• First one: race condition (*xterm*)
• Second one: buffer overflow on stack leading to execution of injected code (*fingerd*)
• Both are very well known, and fixes available!
  • And should be installed everywhere ...
Flaw #1: xterm

- *xterm* emulates terminal under X11 window system
  - Must run as *root* user on UNIX systems
    - No longer universally true; reason irrelevant here
- Log feature: user can log all input, output to file
  - User names file
  - If file does not exist, *xterm* creates it, makes owner the user
  - If file exists, *xterm* checks user can write to it, and if so opens file to append log to it
Check that user can write to file requires special system call
  - Because root can append to any file, check in open will always succeed

```
Check that user can write to file “/usr/tom/X”
if (access(“/usr/tom/X”, W_OK) == 0){
    Open “/usr/tom/X” to append log entries
    if ((fd = open(“/usr/tom/X”, O_WRONLY|O_APPEND))< 0){
        /* handle error: cannot open file */
    }
}
```
Problem

- Binding of file name “/usr/tom/X” to file object can change between first and second lines
  - left is at *access*; right is at *open*
  - Note file opened is *not* file checked
Flaw #2: fingerd

• Exploited by Internet Worm of 1988
  • Recurs in many places, even now

• `finger` client send 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
Frameworks

• Goals dictate structure of classification scheme
  • Guide development of attack tool ⇒ focus is on steps needed to exploit vulnerability
  • Aid software development process ⇒ focus is on design and programming errors causing vulnerabilities

• Following schemes classify vulnerability as n-tuple, each element of n-tuple being classes into which vulnerability falls
  • Some have 1 axis; others have multiple axes
Research Into Secure Operating Systems (RISOS)

- Goal: aid computer, system managers in understanding security issues in OSes, and help determine how much effort required to enhance system security
- Attempted to develop methodologies and software for detecting some problems, and techniques for avoiding and ameliorating other problems
- Examined Multics, TENEX, TOPS-10, GECOS, OS/MVT, SDS-940, EXEC-8
Classification Scheme

- Incomplete parameter validation
- Inconsistent parameter validation
- Implicit sharing of privileged/confidential data
- Asynchronous validation/inadequate serialization
- Inadequate identification/authentication/authorization
- Violable prohibition/limit
- Exploitable logic error
Incomplete Parameter Validation

- Parameter not checked before use
- Example: emulating integer division in kernel (RISC chip involved)
  - Caller provided addresses for quotient, remainder
  - Quotient address checked to be sure it was in user’s protection domain
  - Remainder address *not* checked
    - Set remainder address to address of process’ level of privilege
    - Compute 25/5 and you have level 0 (kernel) privileges
- Check for type, format, range of values, access rights, presence (or absence)
Inconsistent Parameter Validation

• Each routine checks parameter is in proper format for that routine but the routines require different formats

• Example: each database record 1 line, colons separating fields
  • One program accepts colons, newlines as part of data within fields
  • Another program reads them as field and record separators
  • This allows bogus records to be entered
Legacy of RISOS

• First funded project examining vulnerabilities

• Valuable insight into nature of flaws
  • Security is a function of site requirements and threats
  • Small number of fundamental flaws recurring in many contexts
  • OS security not critical factor in design of OSes

• Spurred additional research efforts into detection, repair of vulnerabilities
Program Analysis (PA)

• Goal: develop techniques to find vulnerabilities
• Tried to break problem into smaller, more manageable pieces
• Developed general strategy, applied it to several OSes
  • Found previously unknown vulnerabilities
Classification Scheme

• 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
Improper Choice of Initial Protection Domain

• Initial incorrect assignment of privileges, security and integrity classes
• Example: on boot, protection mode of file containing identifiers of all users can be altered by any user
  • Under most policies, should not be allowed
Improper Isolation of Implementation Detail

- Mapping an abstraction into an implementation in such a way that the abstraction can be bypassed
- Example: virtual machines modulate length of time CPU is used by each to send bits to each other
- Example: Having raw disk accessible to system as ordinary file, enabling users to bypass file system abstraction and write directly to raw disk blocks
Improper Change

• Data is inconsistent over a period of time
• Example: xterm flaw
  • Meaning of “/usr/tom/X” changes between access and open
• Example: parameter is validated, then accessed; but parameter is changed between validation and access
  • Burroughs B6700 allowed allowed allowed this
Improper Naming

• Multiple objects with same name
• Example: Trojan horse
  • `loadmodule` attack discussed earlier; “bin” could be a directory or a program
• Example: multiple hosts with same IP address
  • Messages may be erroneously routed
Improper Deallocation or Deletion

• Failing to clear memory or disk blocks (or other storage) after it is freed for use by others

• Example: program that contains passwords that a user typed dumps core
  • Passwords plainly visible in core dump
Improper Validation

• Inadequate checking of bounds, type, or other attributes or values
• Example: *fingerd’s* failure to check input length
Improper Indivisibility

• Interrupting operations that should be uninterruptable
  • Often: “interrupting atomic operations”

• Example: *mkdir* flaw (UNIX Version 7)
  • Created directories by executing privileged operation to create file node of type directory, then changed ownership to user
  • On loaded system, could change binding of name of directory to be that of password file after directory created but before change of ownership
  • Attacker can change administrator’s password
Improper Sequencing

• Required order of operations not enforced
• Example: one-time password scheme
  • System runs multiple copies of its server
  • Two users try to access same account
    • Server 1 reads password from file
    • Server 2 reads password from file
    • Both validate typed password, allow user to log in
    • Server 1 writes new password to file
    • Server 2 writes new password to file
  • Should have every read to file followed by a write, and vice versa; not two reads or two writes to file in a row
Improper Choice of Operand or Operation

• Calling inappropriate or erroneous instructions

• Example: cryptographic key generation software calling pseudorandom number generators that produce predictable sequences of numbers
Legacy

• First to explore automatic detection of security flaws in programs and systems

• Methods developed but not widely used
  • Parts of procedure could not be automated
  • Complexity
  • Procedures for obtaining system-independent patterns describing flaws not complete
NRL Taxonomy

• Goals:
  • Determine how flaws entered system
  • Determine when flaws entered system
  • Determine where flaws are manifested in system

• 3 different schemes used:
  • Genesis of flaws
  • Time of flaws
  • Location of flaws
Genesis of Flaws

- Intentional
  - Malicious
    - Trojan horse
    - Trapdoor
    - Logic bomb
  - Nonmalicious
    - Covert channel
    - Other

- Nonreplicating
- Replicating
- Storage channel
- Timing channel

- 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

<table>
<thead>
<tr>
<th>Location</th>
<th>Software</th>
<th>Application</th>
<th>Support</th>
<th>Hardware</th>
<th>Operating system</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>System initialization</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Memory management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Process management/scheduling</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Device management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>File management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Identification/authentication</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Other/unknown</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Privileged utilities</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Unprivileged utilities</td>
</tr>
</tbody>
</table>
Legacy

• Analyzed 50 flaws
• Concluded that, with a large enough sample size, an analyst could study relationships between pairs of classes
  • This would help developers focus on most likely places, times, and causes of flaws
• Focused on social processes as well as technical details
  • But much information required for classification not available for the 50 flaws
Aslam’s Model

• Goal: treat vulnerabilities as faults and develop scheme based on fault trees
• Focuses specifically on UNIX flaws
• Classifications unique and unambiguous
  • Organized as a binary tree, with a question at each node. Answer determines branch you take
  • Leaf node gives you classification
• Suited for organizing flaws in a database
Top Level

• Coding faults: introduced during software development
  • Example: *fingerd*’s failure to check length of input string before storing it in buffer

• Emergent faults: result from incorrect initialization, use, or application
  • Example: allowing message transfer agent to forward mail to arbitrary file on system (it performs according to specification, but results create a vulnerability)
Coding Faults

• Synchronization errors: improper serialization of operations, timing window between two operations creates flaw
  • Example: *xterm* flaw

• Condition validation errors: bounds not checked, access rights ignored, input not validated, authentication and identification fails
  • Example: *fingerd* flaw
Emergent Faults

- Configuration errors: program installed incorrectly
  - Example: `tftp` daemon installed so it can access any file; then anyone can copy any file

- Environmental faults: faults introduced by environment
  - Example: on some UNIX systems, any shell with “-” as first char of name is interactive, so find a setuid shell script, create a link to name “-gotcha”, run it, and you have a privileged interactive shell
Legacy

- Tied security flaws to software faults
- Introduced a precise classification scheme
  - Each vulnerability belongs to exactly 1 class of security flaws
  - Decision procedure well-defined, unambiguous
Comparison and Analysis

• Point of view
  • If multiple processes involved in exploiting the flaw, how does that affect classification?
    • $xterm$, $fingerd$ flaws depend on interaction of two processes ($xterm$ and process to switch file objects; $fingerd$ and its client)

• Levels of abstraction
  • How does flaw appear at different levels?
    • Levels are abstract, design, implementation, etc.
xterm and PA Classification

• Implementation level
  • *xterm*: improper change
  • attacker’s program: improper deallocation or deletion
  • operating system: improper indivisibility
xterm and PA Classification

- Consider higher level of abstraction, where directory is simply an object
  - create, delete files maps to writing; read file status, open file maps to reading
  - operating system: improper sequencing
    - During read, a write occurs, violating Bernstein conditions

- Consider even higher level of abstraction
  - attacker’s process: improper choice of initial protection domain
    - Should not be able to write to directory containing log file
    - Semantics of UNIX users require this at lower levels
**xterm** and RISOS Classification

- Implementation level
  - **xterm:** asynchronous validation/inadequate serialization
  - attacker’s process: exploitable logic error and violable prohibition/limit
  - operating system: inconsistent parameter validation
xterm and RISOS Classification

- Consider higher level of abstraction, where directory is simply an object (as before)
  - all: asynchronous validation/inadequate serialization

- Consider even higher level of abstraction
  - attacker’s process: inadequate identification/authentication/authorization
    - Directory with log file not protected adequately
    - Semantics of UNIX require this at lower levels
xterm and NRL Classification

• Time, location unambiguous
  • Time: during development
  • Location: Support:privileged utilities

• Genesis: ambiguous
  • If intentional:
    • Lowest level: inadvertent flaw of serialization/aliasing
  • If unintentional:
    • Lowest level: nonmalicious: other
  • At higher levels, parallels that of RISOS
**xterm** and Aslam’s Classification

- **Implementation level**
  - attacker’s process: object installed with incorrect permissions
    - attacker’s process can delete file
  - **xterm**: access rights validation error
    - **xterm** doesn’t properly validate file at time of access
  - operating system: improper or inadequate serialization error
    - deletion, creation should not have been interspersed with access, open
  - Note: in absence of explicit decision procedure, all could go into class race condition
The Point

- The schemes lead to ambiguity
  - Different researchers may classify the same vulnerability differently for the same classification scheme

- Not true for Aslam’s, but that misses connections between different classifications
  - \textit{xterm} is race condition as well as others; Aslam does not show this
fingerd and PA Classification

• Implementation level
  • *fingerd*: improper validation
  • attacker’s process: improper choice of operand or operation
  • operating system: improper isolation of implementation detail
fingerd and PA Classification

• Consider higher level of abstraction, where storage space of return address is object
  • operating system: improper change
  • fingerd: improper validation
    • Because it doesn’t validate the type of instructions to be executed, mistaking data for valid ones

• Consider even higher level of abstraction, where security-related value in memory is changing and data executed that should not be executable
  • operating system: improper choice of initial protection domain
fingerd and RISOS Classification

• Implementation level
  • *fingerd*: incomplete parameter validation
  • attacker’s process: violable prohibition/limit
  • operating system: inadequate identification/authentication/authorization
fingertd and RISOS Classification

• Consider higher level of abstraction, where storage space of return address is object
  • operating system: asynchronous validation/inadequate serialization
  • fingertd: inadequate identification/authentication/authorization

• Consider even higher level of abstraction, where security-related value in memory is changing and data executed that should not be executable
  • operating system: inadequate identification/authentication/authorization
fingerd and NRL Classification

• Time, location unambiguous
  • Time: during development
  • Location: support: privileged utilities

• Genesis: ambiguous
  • Known to be inadvertent flaw
  • Parallels that of RISOS
fingerd and Aslam Classification

• Implementation level
  • *fingerd*: boundary condition error
  • attacker’s process: boundary condition error
    • operating system: environmental fault
      • If decision procedure not present, could also have been access rights validation errors
Standards

• Descriptive databases used to identify vulnerabilities and weaknesses

• Examples:
  • Common Vulnerabilities and Exposures (CVE)
  • Common Weaknesses and Exposures (CWE)
CVE

- Goal: create a standard identification catalogue for vulnerabilities
  - So different vendors can identify vulnerabilities by one common identifier
  - Created at MITRE Corp.

- Governance
  - CVE Board provides input on nature of specific vulnerabilities, determines whether 2 reported vulnerabilities overlap, and provides general direction and very high-level management
  - Numbering Authorities assign CVE numbers within a distinct scope, such as for a particular vendor

- CVE Numbers: CVE-year-number
  - Number begins at 1 each year, and is at least 4 digits
Structure of Entry

Main fields:

• CVE-ID: CVE identifier

• Description: what is the vulnerability

• References: vendor and CERT security advisories

• Date Entry Created: year month day as a string of 8 digits
Example: Buffer Overflow in GNU C Library

CVE-ID: CVE-2016-3706
Description: Stack-based buffer overflow in the getaddrinfo function in sysdeps/posix/getaddrinfo.c in the GNU C Library (aka glibc or libc6) allows remote attackers to cause a denial of service (crash) via vectors involving hostent conversion. NOTE: this vulnerability exists because of an incomplete fix for CVE-2013-4458
References:
• CONFIRM:https://sourceware.org/bugzilla/show_bug.cgi?id=20010
• CONFIRM:https://sourceware.org/git/gitweb.cgi?p=glibc.git;h=4ab2ab03d4351914ee53248dc5aeef4a8c88ff8b9
• CONFIRM:http://www-01.ibm.com/support/docview.wss?uid=swg21995039
• CONFIRM:https://source.android.com/security/bulletin/2017-12-01
• SUSE:openSUSE-SU-2016:1527
• URL:http://lists.opensuse.org/opensuse-updates/2016-06/msg00030.html
• SUSE:openSUSE-SU-2016:1779
• URL:http://lists.opensuse.org/opensuse-updates/2016-07/msg00039.html
• BID:88440
• URL:http://www.securityfocus.com/bid/88440
• BID:102073
• URL:http://www.securityfocus.com/bid/102073
Assigning CNA: N/A
Date Entry Created: 20160330
CVE Use

• CVE database begun in 1999
  • Contains some vulnerabilities from before 1999
• Currently over 82,000 entries
• Used by over 150 organizations
  • Security vendors such as Symantec, Trend Micro, Tripwire
  • Software and system vendors such as Apple, Juniper Networks, Red Hat, IBM
  • Other groups such as CERT/CC, U.S. NIST, and internationally
CWE

- Database listing weaknesses underlying CVE vulnerabilities
  - Developed by CVE list developers, with help from NIST, vulnerabilities research community

- Organized as a list
  - Can also be viewed as a graph as some weaknesses are refinements of others
  - Not a tree as some nodes have multiple parents
Types of Entries

• **Category entry**: identifies set of entries with a characteristic of the current entry
• **Chain entry**: sequence of distinct weaknesses that can be linked together within software
  • One weakness can create necessary conditions to enable another weakness to be exploited
• **Compound element composite entry**: multiple weaknesses that must be present to enable an exploit
• **View entry**: view of the CWE database for particular weakness or set of weaknesses.
• **Weakness variant entry**: weakness described in terms of a particular technology or language
• **Weakness base entry**: more abstract description of weakness than a weakness variant entry, but in sufficient detail to lead to specific methods of detection and remediation
• **Weakness class**: describes weakness independently of any specific language or technology.
Examples

• CWE-631, Resource-Specific Weaknesses (a view entry)
  • Child: CWE-632, Weaknesses that Affect Files or Directories
  • Child: CWE-633, Weaknesses that Affect Memory
  • Child: CWE-634, Weaknesses that Affect System Processes

• CWE-680, Integer Overflow to Buffer Overflow (a chain entry)
  • Begins with integer overflow (CWE-190)
  • Leads to failure to restrict some operations to bounds of buffer (CWE-119)

• CWE-61, UNIX Symbolic Link (Symlink) Following (a composite entry)
  • Requires 5 weaknesses to be present before it can be exploited
  • CWE-362, CWE-340, CWE-216, CWE-386, CWE-732
Abstraction Level of Weaknesses

• Goal is to avoid problem of different classifications depending on the layer of abstraction

• Levels:
  • **Class**: weakness at an abstract level, independent of any programming language or environment
  • **Base**: weakness at an abstract level, with enough detail to enable development of methods of detection, prevention, remediation
  • **Variant**: weakness at a low level, usually tied to specific technology, system, programming language

• Useful demarcation of vulnerabilities related to design, implementation, or both