Lecture 19
November 7, 2025

ECS 235A, Computer and Information Security

Administration

- Homework 3, extra credit 3 due date extended to November 10, 2025

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

Standards

- Descriptive databases used to identify vulnerabilities and weaknesses
- Examples:
  - Common Vulnerabilities and Exposures (CVE)
  - Common Weaknesses and Exposures (CWE)
  - NIST National Vulnerability Database (NVD)

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=4ab2ab03d4351914ee53248dc5aef4a8c88ff8b9
- 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

CVSS

- Common Vulnerability Scoring System (CVSS)
  - Version 4.0
  - Managed by FIRST (Forum for Internet Response Security Teams)
- Scores
  - CVSS-B: Base metrics — **this measures severity, not risk**
  - CVSS-BE: Base and environmental metrics
  - CVSS-BT: Base and threat metrics
  - CVSS-BTE: Base, threat, and environmental metrics

CVSS-B

- Exploitability metrics
  - Attack Vector (AV):
    - Network (N): vulnerable system is on the network, set of attackers is anywhere on the Internet
    - Adjacent (A): vulnerable system is on the network, set of attackers limited to logically adjacent topology
    - Local (L): vulnerable system is not on the network, attacker's path via r/w/x capabilities
    - Physical (P): attacker must physically manipulate the vulnerable system
  - Attack Complexity (AC):
    - Low (L): attacker must take no measurable action to exploit vulnerability
    - High (H): success depends on evading or circumventing security techniques such as ASLR or gathering target-specific secrets

CVSS-B

- More exploitability metrics
  - Attack Requirements (AT):
    - None (N): attack does not depend on deployment, execution condition conditions
    - Present (P): attack depends on specific deployment, execution condition conditions such as a race condition or being able to inject code into a network connection
  - Privileges Required (PR):
    - None (N): attacker need not authenticate
    - Low (L): attacker needs ordinary user privileges
    - High (H): attacker needs significant privileges (eg., administrative, *root*)
  - User Interaction (UI):
    - None (N): no user interaction other than that of the attacker
    - Passive (P): limited interaction by targeted user with vulnerable system,
    - Active (A): Targeted user must perform specific, conscious interaction with vulnerable system

CVSS-B

- Impact metrics
  - Confidentiality (VC)
    - None (N): no loss of confidentiality within vulnerable system
    - Low (L): access to some confidential information, but attacker does not control what information and it doesn't cause direct, serious loss to vulnerable system
    - High (H): total loss of confidentiality, so attacker can see everything; or, the attacker can only see some information, but that information poses a direct, serious loss
  - Confidentiality impact to the subsequent system (SC)
    - As above, but to a system attacked from the first
  - Integrity (VI): as with VC
  - Integrity impact to the subsequent system (SI): as with SC
  - Availability (VA): as with VC
  - Availability impact to the subsequent system (SA): as with SC

CVSS-B

- Exploit maturity E:
  - Unreported (U): no knowledge of public proof-of-concept exploits, of reported attempts to exploit, of public solutions to ameliorate it
  - Proof-of-concept (P): public proof-of-concept exploits, no knowledge of reported attempts to exploit, of public solutions to ameliorate it
  - Attacked (A): attacks have been reported, or tools to simplify exploiting vulnerability are available
  - Not defined (X): no threat intelligence available; treated as A when calculating score

Environmental Metrics

- Confidentiality Requirements (CR)
  - Low (L): Loss of confidentiality has a limited bad effect on organization, associated people
  - Medium (M): Loss of confidentiality has a serious bad effect on organization, associated people
  - High (H): Loss of confidentiality has a catastrophic bad effect on organization, associated people
- Integrity Requirements (IR): see CR
- Availability Requirements AR): see CR

Other Metrics

- Modified *base metric*: adds Not Defined (X) as default
  - For subsequent systems (SC, SI, SA), lowest value is Negligible (N), not None
  - Also for subsequent systems integrity, highest severity level is Safety (S)
- Supplemental metrics
  - Safety (S)
  - Automatable (AU)
  - Provider Urgency (U)
  - Recovery (R)
  - Value Density (V)
  - Vulnerability Response Effort (RE)

Example: CVSS Vector

`CVSS:4.0/AV:N/AC:L/AT:N/PR:H/UI:N/`

Attackers can attack over the Internet (AV:N), and complexity of attack is low (AC:L); the attack does not depend on conditions on the vulnerable systems (AT:N) but requires administrative/*root* privileges (PR:H); it does not require anyone (except the attacker) to do anything

`VC:L/VI:N/VA:N/`

There is some loss of confidentiality, but not of integrity or availability

`SC:N/SI:N/SA:N`

There is no loss of confidentiality, integrity, or availability to any downstream system

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.

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

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

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

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.)