Vulnerability Analysis

Introduction

- Definitions
- Formal verification vs penetration testing
- Methodology
- Vulnerability Databases

Verification and Testing

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*

Vulnerabilities

- Bugs at the implementation level
- Flaws at the design level
  - The hardest defect category to handle
  - The most prevalent and critical

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

Secure Testing vs. Standard Testing

- Standard software testing focus on software failure
- Secure software testing adds an intelligent adversary
- Security risk management

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.)

Penetration Testing

Penetration Studies

- 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?
- Find some number of vulnerabilities, or vulnerabilities within a period of time

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

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

Flaw Hypothesis Methodology

Flaw Hypothesis Methodology

1. Information gathering
   - Become familiar with system's functioning
2. Flaw hypothesis
   - Draw on knowledge to hypothesize vulnerabilities
3. Flaw testing
   - Test them out
4. Flaw generalization
   - Generalize vulnerability to find others like it
5. (*maybe*) Flaw elimination
   - Testers eliminate the flaw (usually *not* included)

Information Gathering

- Devise model of system and/or components
  - Look for discrepancies in components
  - Consider interfaces among components
- Need to know system well (or learn quickly!)
  - Design documents, manuals help
    - Unclear specifications often misinterpreted, or interpreted differently by different people
  - Look at how system manages privileged users

Flaw Hypothesizing

- Examine policies, procedures
  - May be inconsistencies to exploit
  - May be consistent, but inconsistent with design or implementation
  - May not be followed
- Examine implementations
  - Use models of vulnerabilities to help locate potential problems
  - Use manuals; try exceeding limits and restrictions; try omitting steps in procedures

Flaw Hypothesizing (*con't*)

- Identify structures, mechanisms controlling system
  - These are what attackers will use
  - Environment in which they work, and were built, may have introduced errors
- Throughout, draw on knowledge of other systems with similarities
  - Which means they may have similar vulnerabilities
- Result is list of possible flaws

Flaw Testing

- Figure out order to test potential flaws
  - Priority is function of goals
    - Example: to find major design or implementation problems, focus on potential system critical flaws
    - Example: to find vulnerability to outside attackers, focus on external access protocols and programs
- Figure out how to test potential flaws
  - Best way: demonstrate from the analysis
    - Common when flaw arises from faulty spec, design, or operation
  - Otherwise, must try to exploit it

Flaw Testing (*con't*)

- Design test to be least intrusive as possible
  - Must understand exactly why flaw might arise
- Procedure
  - Back up system
  - Verify system configured to allow exploit
    - Take notes of requirements for detecting flaw
  - Verify existence of flaw
    - May or may not require exploiting the flaw
    - Make test as simple as possible, but success must be convincing
  - Must be able to repeat test successfully

Flaw Generalization

- As tests succeed, classes of flaws emerge
  - Example: programs read input into buffer on stack, leading to buffer overflow attack; others copy command line arguments into buffer on stack ⇒ these are vulnerable too
- Sometimes two different flaws may combine for devastating attack
  - Example: flaw 1 gives external attacker access to unprivileged account on system; second flaw allows any user on that system to gain full privileges ⇒ any external attacker can get full privileges

Flaw Elimination

- Usually not included as testers are not best folks to fix this
  - Designers and implementers are
- Requires understanding of context, details of flaw including environment, and possibly exploit
  - Design flaw uncovered during development can be corrected and parts of implementation redone
    - Don't need to know how exploit works
  - Design flaw uncovered at production site may not be corrected fast enough to prevent exploitation
    - So need to know how exploit works

Vulnerability Databases

Standards

- Descriptive databases used to identify vulnerabilities and weaknesses
- Examples:
  - Common Vulnerabilities and Exposures (CVE)
  - Common Weaknesses Enumeration (CWE)

CVE

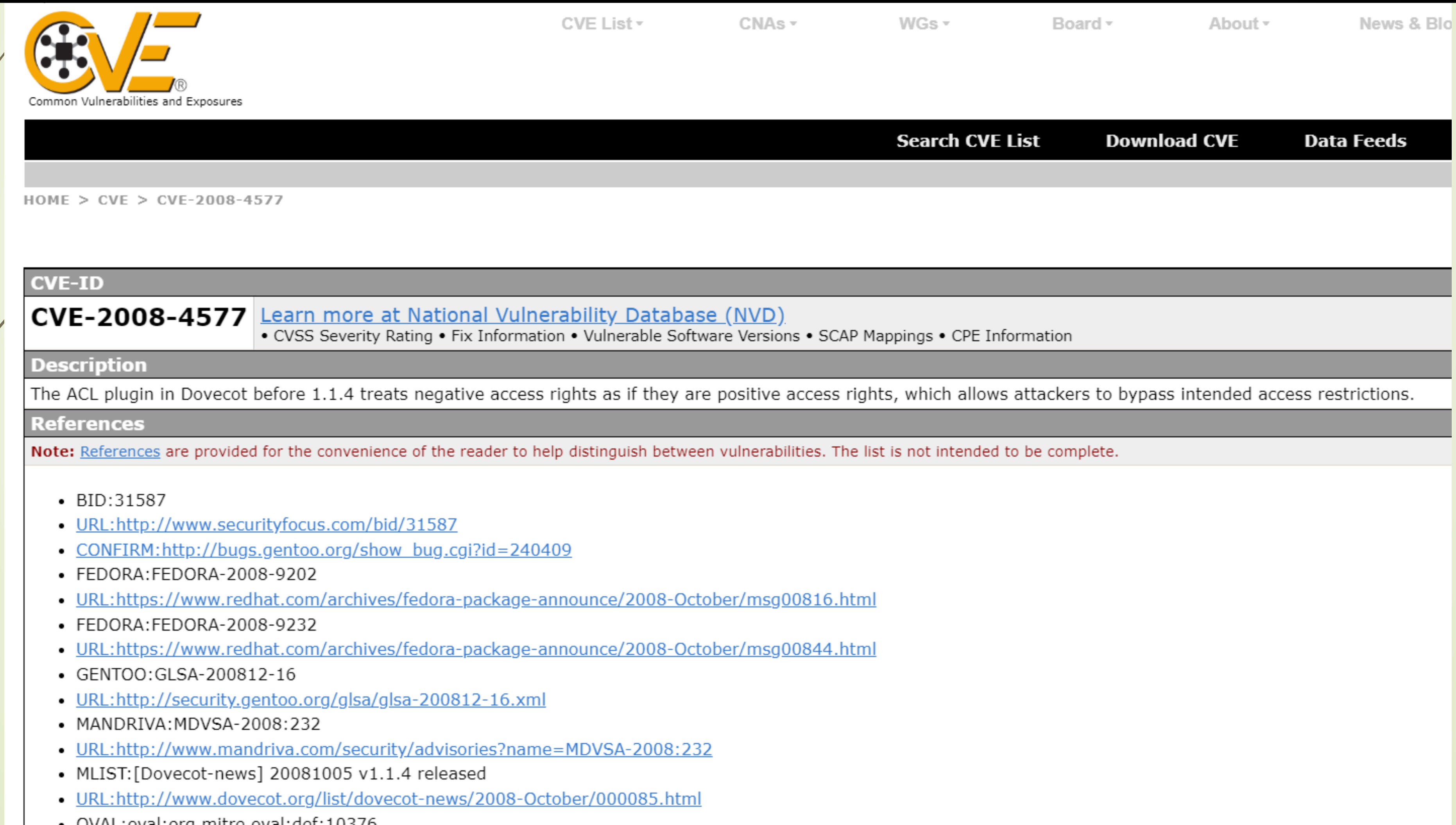
- Goal: create a standard identification catalogue for vulnerabilities
  - So different vendors can identify vulnerabilities by one common identifier
  - Created at MITRE Corp.
- Governance
  - CVE Board provides input on nature of specific vulnerabilities, determines whether 2 reported vulnerabilities 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*

CVE Sample



CVE-ID

CVE-2008-4577 [Learn more at National Vulnerability Database (NVD)](#)
• CVSS Severity Rating • Fix Information • Vulnerable Software Versions • SCAP Mappings • CPE Information

Description

The ACL plugin in Dovecot before 1.1.4 treats negative access rights as if they are positive access rights, which allows attackers to bypass intended access restrictions.

References

Note: References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.

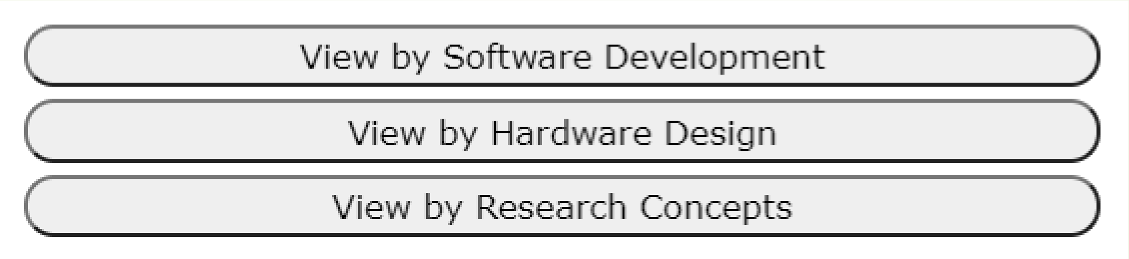
- BID:31587
- URL:http://www.securityfocus.com/bid/31587
- CONFIRM:http://bugs.gentoo.org/show_bug.cgi?id=240409
- FEDORA:FEDORA-2008-9202
- URL:https://www.redhat.com/archives/fedora-package-announce/2008-October/msg00816.html
- FEDORA:FEDORA-2008-9232
- URL:https://www.redhat.com/archives/fedora-package-announce/2008-October/msg00844.html
- GENTOO:GLSA-200812-16
- URL:http://security.gentoo.org/glsa/glsa-200812-16.xml
- MANDRIVA:MDVSA-2008:232
- URL:http://www.mandriva.com/security/advisories?name=MDVSA-2008:232
- MLIST:[Dovecot-news] 20081005 v1.1.4 released
- URL:http://www.dovecot.org/list/dovecot-news/2008-October/000085.html
- OVAL:oval:org.mitre.oval:def:10376

CVE Use

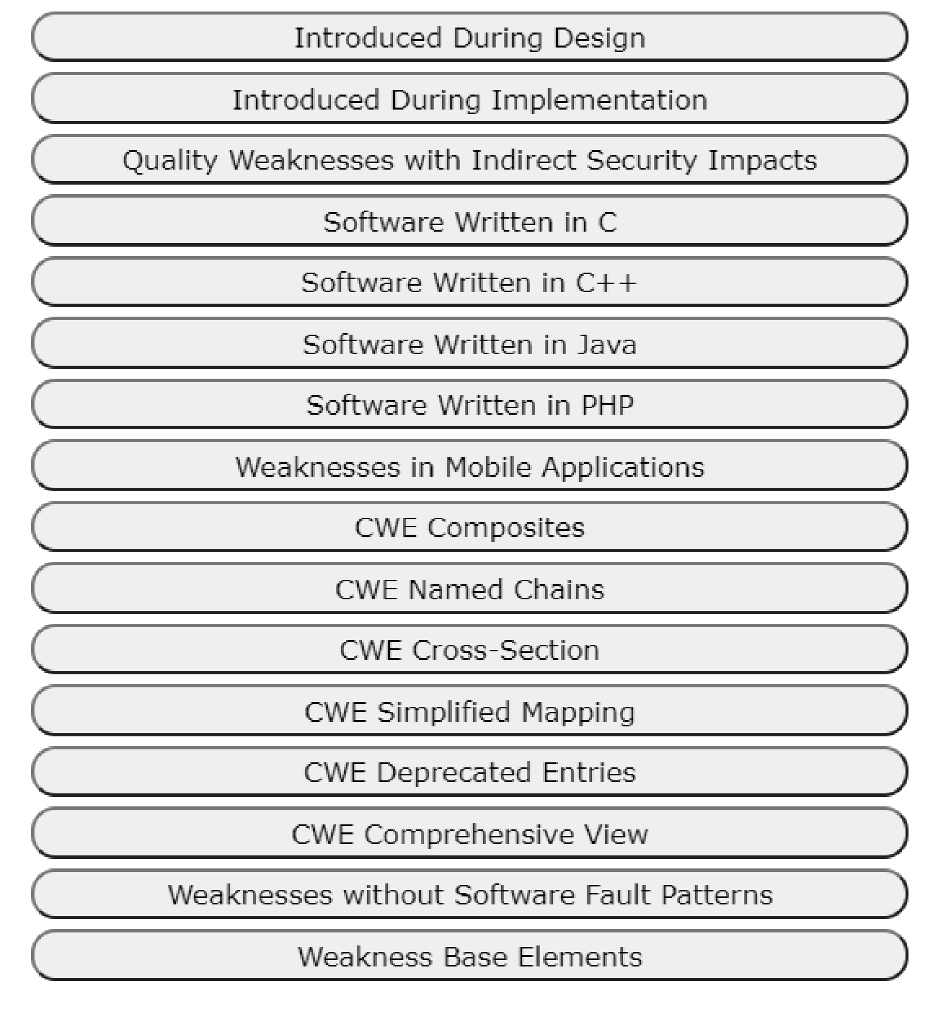
- CVE database begun in 1999
  - Contains some vulnerabilities from before 1999
- Used by many organizations
  - Security vendors such as Symantec, Trend Micro, Tripwire
  - Software and system vendors such as Apple, Juniper Networks, Red Hat, IBM
  - Other groups such as CERT/CC, U.S. NIST

CWE

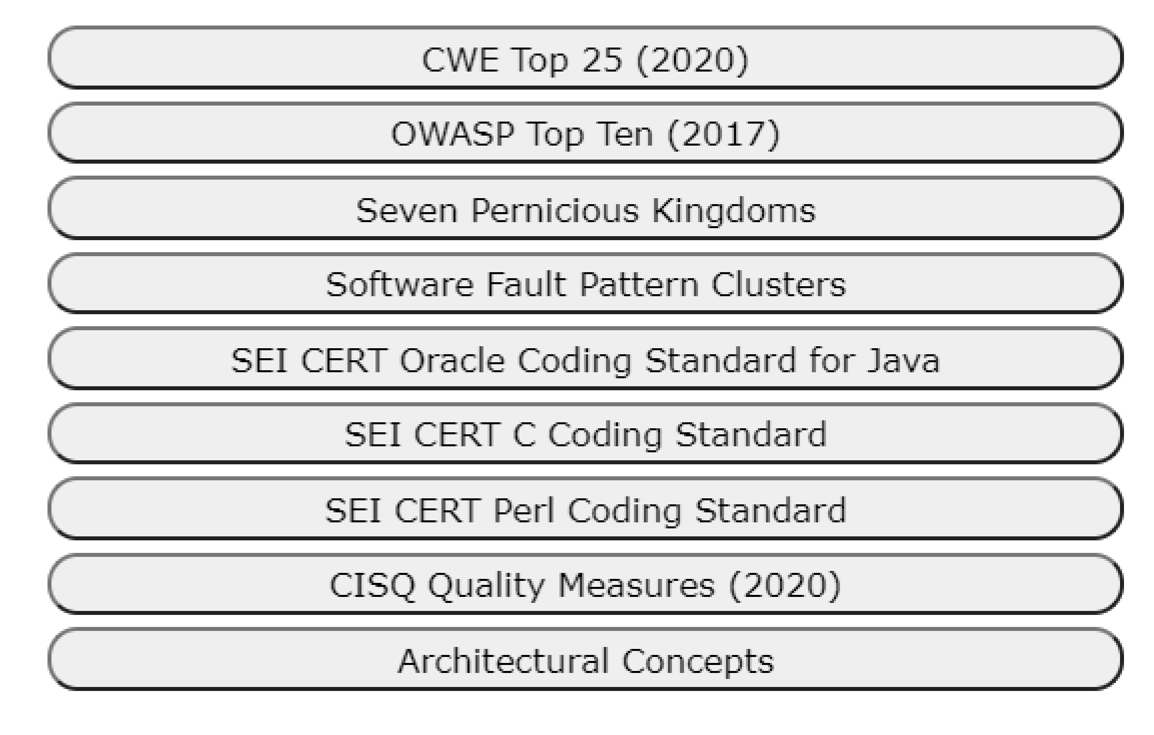
- Common Weakness Enumeration is a list of software and hardware weaknesses types
- Ongoing work to capture the specific effects, behaviors, exploit mechanisms, and implementation details
- Hierarchical representations
  - Frequently used or encountered in software development
  - Frequently used or encountered in hardware design
  - Facilitates research into weakness types and organizes items by behaviors using multiple levels of abstraction



Helpful Views



External Mappings


CWE Top 25 (2020)
OWASP Top Ten (2017)
Seven Pernicious Kingdoms
Software Fault Pattern Clusters
SEI CERT Oracle Coding Standard for Java
SEI CERT C Coding Standard
SEI CERT Perl Coding Standard
CISQ Quality Measures (2020)
Architectural Concepts

Theory of Penetration

Gupta and Gilgor's Theory of Penetration

- Goal: detect previously undetected flaws
- Based on two hypotheses:
  - Implies an appropriate set of design and implementation principles would prevent vulnerabilities
- Idea: formulate principles consistent with these hypotheses and check system for inconsistencies

Penetration Resistance

- ***Theorem***: Let the system be in a state that is penetration-resistant to an attack exploiting a failure to check conditions. Then if a state transition function is applied to the current state, the resulting state will also be penetration-resistant to an attack exploiting a failure to check conditions.
- Can it be generalized?

Key Points

Key Points

- Formal verification
- Theory of penetration
- Penetration testing
- Three layers
- Five steps of flaw hypothesis methodology
- CWE and CVE