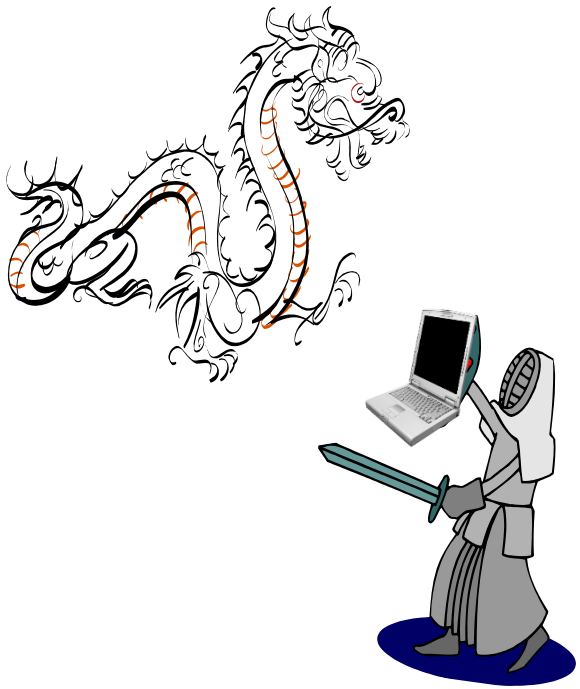


Vulnerabilities and Attacks

A Rose by Any Other Name

- Sometimes vulnerabilities and attacks are undifferentiable, but that's mostly a naming issue. E.g. Sql Injection is a vulnerability, and its also a type of attack.
- A buffer overflow vulnerability can be exploited by several different attack methods.

Twenty Vulnerabilities

1. SQL Injection
2. Cross-site scripting
3. Buffer Overflows
4. Format String Issues
5. Integer Overflows
6. Command Injections
7. Failing to handle errors properly
8. Failing to protect network traffic
9. Using magic URL's and hidden form fields
10. Improper use of SSL/TLS
11. Weak password systems
12. Failing to store and protect data securely
13. Information leakage
14. Improper file access
15. Trusting network name resolution
16. Race conditions
17. Unauthenticated key exchange.
18. Not cryptographically strong random nos.
19. Poor usability
20. Cross-site request forgery

Or Are They Design/Coding Errors

1. Failure to neutralize special characters
2. Failure to control system state
3. Failing to handle errors properly
4. Failing to protect network traffic
5. Depending on obscurity for security
6. Weak cryptography
7. Weak authentication
8. Weak authorization model
9. Failing to store and protect data securely
10. Improper trust in unreliable data/components
11. Failure to handle race conditions securely

Understanding a Vulnerability

- The Attack – how does it happen
- The Risk – how bad can it get
- Discovery – how would you (or an attacker) find it
- Remediation – how do you prevent it
- Avoidance – how should you develop

What is Risk?

- Risk = Probability (exploit) x Exploit Cost
- Probability of exploit is not that obvious
- Combination of:
  - Discoverability (D)
  - Reproducibility (R)
  - Exploitability (E)
- Cost is combination of
  - Damage potential (D) and
  - Number of affected users (A).
- DREAD Model – how you evaluate risk

Risk Example

SQL Injection that reveals customer data.

1 = lowest, 5 = highest

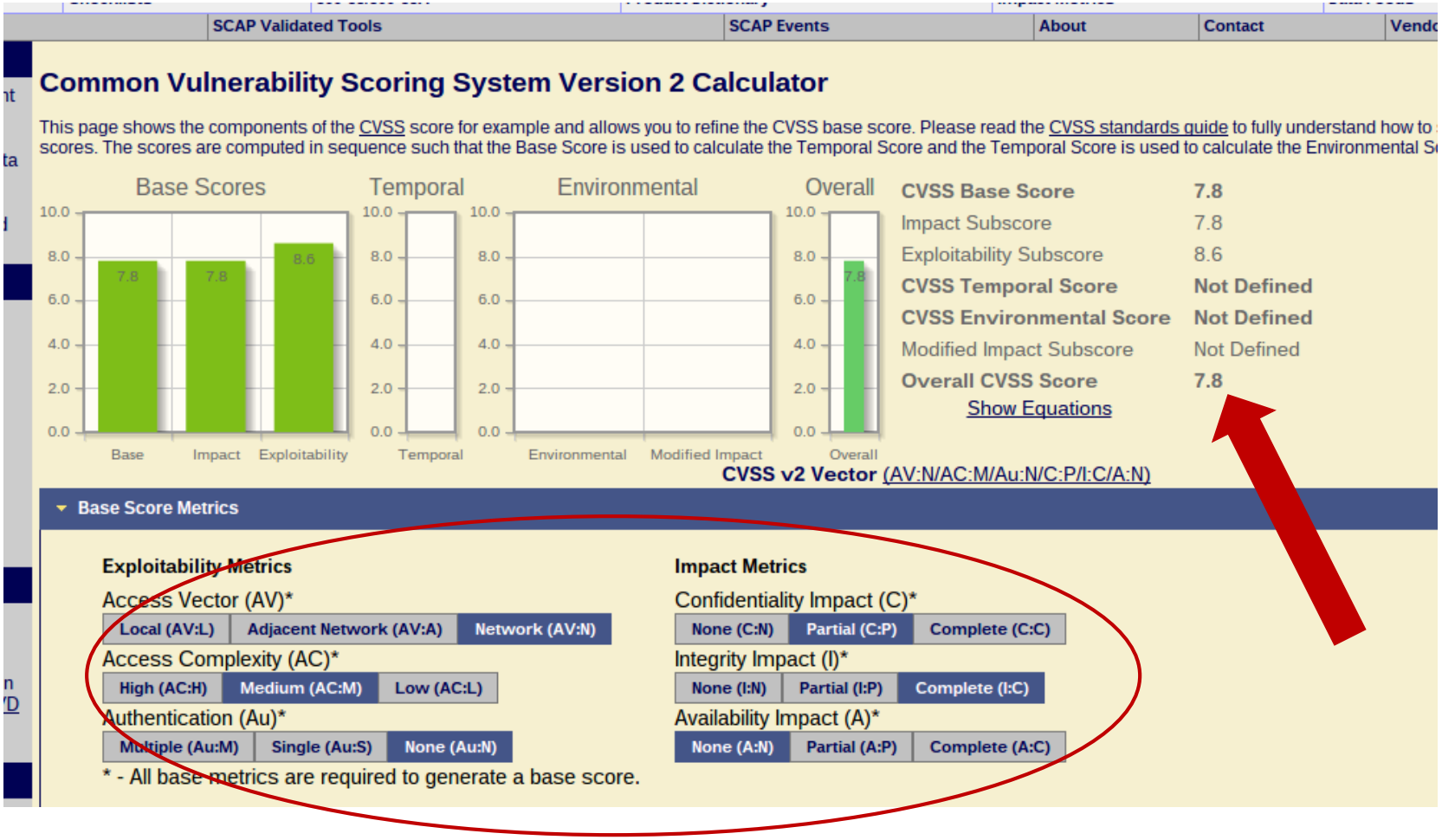
D = 3, R=5, E=3, A=4, D=5

Risk = (3+5+3+4+5)/5 = 4.0/5

The Common Vulnerability Scoring System

- CVSS
- A methodology for scoring specific vulnerabilities
- http://nvd.nist.gov/cvss.cfm?calculator&version=2
- Exploitability metrics
  - Access Vector = Local, Adjacent Network, Network
  - Access Complexity = High, Medium, Low
  - Authentication = Multiple, Single, None
- Impact metrics
  - Confidentiality Impact = None, Partial, Complete
  - Integrity Impact = None, Partial, Complete
  - Availability = None, Partial, Complete

The Common Vulnerability Scoring System



SQL Injection
The Attack

- It is possible for an attacker to inject SQL code into an SQL request
- For example, instead of entering a valid user name, someone enters

```
XXX' or '1'='1
```

- And the result is that they are logged in
- Internally, an SQL request to the database has been corrupted. For example:

```
select userid from users where id='$username'
```

- becomes

```
select userid from users where id='XXX' or '1'='1'
```

SQL Injection
The Risk

- Discoverability – low to high
- Reproducibility – typically high
- Exploitability – medium to high
- Affected Users – medium to high
- Damage Potential – typically high

SQL Injection
The Discovery

- By trying injection strings in application inputs
- Identify entry points that might lead to directly database access. Why directly??
- Typically not difficult to identify a weakness, but it may be more difficult to find the exploit

SQL Injection
The Remediation

- Use prepared statements to allow the database engine to use its knowledge of the database to protect itself
- Sanitize inputs
  - Whitelisting
  - Blacklist

SQL Injection Avoidance

- Design phase
  - Require prepared statements for database access
  - Centralize access to the database
  - Require that all inputs be whitelist sanitized
  - Centralize input sanitization
  - Treat database access and input sanitization modules as high risk functionality
  - Write security tests for at-risk dataflows
- Implementation
  - Follow the secure coding guide
  - Use code reviews and static code analyzers
- Testing
  - Execute all security tests
  - Use dynamic analyzers to find potential vulnerabilities

Threats

- The list of threats includes:
  - All the different ways of exploiting
  - Every different vulnerability
- It's just too large to manage
- The STRIDE model categorizes threats
  - Spoofing
  - Tampering
  - Repudiation
  - Information Disclosure
  - Denial of Service
  - Elevation of Privilege
- Each vulnerability is manifested by one or more of these threats

Where Are We?

- Security Engineering has the following pieces
  - What are the threats to software?
    - SQL Injection and friends
  - How do we remediate the threats?
    - Each is different, but there are similarities
  - How do we find the vulnerabilities?
    - Processes and tools
    - Experience - Hackthissite???
  - How do we develop securely?
    - The Secure Development Lifecycle (SDL)