CS-4920: Lecture 5 Developing Secure Software

- Today's Outcomes
  - Discuss the connection between defects and security
  - Identify several types of defects
  - Discuss the cost/schedule ramifications of defect reduction
  - State several benefits of managing defects throughout the SDLC
  - Discuss approaches to integrating secure development practices into Scrum

1

Software defects and security

- Many defects are security problems
- All security problems are defects
  - During requirements, design, implementation, …
- To do security well, it must be "built in" to the software development life cycle (SDLC)
  - As opposed to being "bolted on" later

2

Major Points

- Defective software is seldom secure
- Defective software is preventable
- Reducing defects is less costly than responding to released vulnerabilities

3

Defective software is seldom secure

- Experienced developers still inject a lot of defects
  - During requirements, design, implementation, test
  - 1 defect per every 7-10 lines of new/changed code!
  - Even with 99% removal, that's 1-1.5 defects per kLOC
  - Jones' study on released software showed typically 1-7 defects per new/changed kLOC

4

Relationship of defects and security problems

- Nearly all vulnerabilities are caused by known defect types
- Vast majority are are on top 10/25 lists updated by SANS, OWASP, and others
  - http://www.sans.org/top25-software-errors/
  - https://www.owasp.org/index.php/Top_10_2013-Top_10
  - See also http://cve.mitre.org/

5

Types of defects

- Sophisticated
  - Inadequate authentication
  - Invalid authorization
  - Incorrect use of cryptography
  - Failure to protect data
  - Failure to partition applications
- Simple / most common
  - Declaration errors
  - Logic errors
  - Loop control errors
  - Failure to validate input
  - Interface specification errors
  - Configuration errors
  - Failure to understand simple security issues

Which of the basic security principles apply here?

6

Is defective software unavoidable due to its complexity?

- Numerous studies have shown that certain development processes drastically reduce defects.
  - Researchers and practitioners concur that this applies to security defects as well.
  - Security defects have their own character: generally not caught by use case (normal use) or basic functional requirements

7

How much does it cost?

- Very little and sometimes less
- How?
  - Processes make meeting schedule more likely
    - Schedule error reduced to 6%
  - Less time spent on repair / early repair is cheaper
    - 4% of total time
  - 78% resulting increase in productivity
  - Very good correlations between fewer defects and less schedule error [Jones]

8

Post-release costs also reduced

- Producers
  - Creating, testing, and releasing patches
  - Bad press
  - Customer dissatisfaction
  - Legal action
- Consumers
  - Testing and deployment
  - Downtime

9

Two-pronged process approach

- Defects managed throughout SDLC
- Address security throughout SDLC

10

Managing defects throughout the SDLC

- Managing = removal and measurement
- Multiple defect removal points needed
  - Less propagation of defects
  - Easier to trace to root cause
  - Early warnings of process variance
  - More chances to catch an early defect
  - Common points:
    - Requirements, architectural analysis, design verification, design review
    - Code review, static code analysis
    - Unit test, penetration test, system test

11

Address security throughout SDLC

- Understand common causes of vulnerabilities
  - Buffer overflows
  - SQL injection
  - Race conditions
  - Cross-site scripting
    - Unfortunate name, essentially unquoted HTML or script
    - Types: local, reflected, stored
- But, is that enough?
  - One company has a 700-page document describing common causes…

12

Augment understanding with best practices

- Consider buffer overflow – what is the problem?
  - Overwrite stack (common exploit)
  - May happen inside library call
- General principles to avoid the problem
  - Design: validate all input (use of custom types, etc.)
  - Verification: State machine (UML state chart)
  - Coding: checklists, static analysis (identify unsafe calls)
  - Testing: Fuzz test
- Apply throughout the SDLC
- These go back to many of our earlier principles
  - Risk analysis, defense in depth, application partitioning (least common mechanism?), least privilege

13

Scrum security challenges

- Product backlog doesn't work well for security because it is not generally visible and within the client's expertise
  - Proposal: addition of "security backlog" managed by "Security Master": document risk, apply testing to new/selected features
    - Avoid disrupting sprint flow
    - Recognize and manage risk from the beginning
    - Training recognized as key by many sources

14

Process commonalities

- Scrum/agile have commonalities with older processes such as TSP
  - Plan for security from the beginning
  - The self-directed team takes responsibility and requires decision making ability
  - Multiple defect removal points minimize cost
  - Training in security issues is critical
  - Adapt based on what's working and what's not

15

References

- Azham, Z., I. Ghani, and N. Ithnin. *Security Backlog in Scrum Security Practices*, 5th Malaysian Conference in Software Engineering (MySec), IEEE, pp. 414-417, 2011.
- Schoonover, Glenn. *Enhancing Customer Security: Built-in versus Bolt-on*, The DoD Software Tech News, v8 i2, July 2005.
- Jones, Capers. *Software Assessments, Benchmarks, and Best Practices*. Reading, MA: Addison-Wesley, 2000.

16