Secure Software Development

Penetrate and Patch

- Usual approach to software development
  - Develop product as quickly as possible
  - Release it without adequate testing
  - Patch the code as flaws are discovered
- In security, this is "penetrate and patch"
  - A **bad** approach to software development
  - An **even worse** approach to secure software!

Why Penetrate and Patch?

- First to market advantage
  - First to market likely to become market leader
  - Market leader has huge advantage in software
  - Users find it safer to "follow the leader"
  - Boss won't complain if your system has a flaw, as long as everybody else has same flaw…
  - User can ask more people for support, etc.
- Sometimes called "network economics"

Why Penetrate and Patch?

- Secure software development is hard
  - Costly and time consuming development
  - Costly and time consuming testing
  - Cheaper to let customers do the work!
- No serious economic disincentive
  - Even if software flaw causes major losses, the software vendor is not liable
  - Is any other product sold this way?
  - Would it matter if vendors were legally liable?

Penetrate and Patch Fallacy

- **Fallacy:** If you keep patching software, eventually it will be secure
- Why is this a fallacy?
- Empirical evidence to the contrary
- Patches often add new flaws
- Software is a moving target: new versions, features, changing environment, new uses,…

Open vs Closed Source

- Open source software
  - The source code is available to user
  - For example, Linux
- Closed source
  - The source code is not available to user
  - For example, Windows
- What are the security implications?

Open Source Security

- Claimed advantages of open source is
  - **More eyeballs:** more people looking at the code should imply fewer flaws
  - A variant on Kerchoffs Principle
- Is this valid?
  - How many "eyeballs" looking for security flaws?
  - How many "eyeballs" focused on boring parts?
  - How many "eyeballs" belong to security experts?
  - Attackers can also look for flaws!
  - Evil coder might be able to insert a flaw

Open Source Security

- Open source example: wu-ftp
  - About 8,000 lines of code
  - A security-critical application
  - Was deployed and widely used
  - After 10 years, serious security flaws discovered!
- More generally, open source software has done little to reduce security flaws
- Why?
  - Open source follows penetrate and patch model!

Closed Source Security

- Claimed advantage of closed source
  - Security flaws not as visible to attacker
  - This is a form of "security by obscurity"
- Is this valid?
  - Many exploits do not require source code
  - Possible to analyze closed source code…
  - …though it is a lot of work!
  - Is "security by obscurity" real security?

Open vs Closed Source

- Advocates of open source often cite the **Microsoft fallacy** which states
  1. Microsoft makes bad software
  2. Microsoft software is closed source
  3. Therefore all closed source software is bad
- Why is this a fallacy?
  - Not logically correct
  - More relevant is the fact that Microsoft follows the penetrate and patch model

Open vs Closed Source

- No obvious security advantage to either open or closed source
- More significant than open vs closed source is software development practices
- Both open and closed source follow the "penetrate and patch" model

Open vs Closed Source

- If there is no security difference, why is Microsoft software attacked so often?
  - Microsoft is a big target!
  - Attacker wants most "bang for the buck"
- Few exploits against Mac OS X
  - **Not** because OS X is inherently more secure
  - An OS X attack would do less damage
  - Would bring less "glory" to attacker
- Next, we consider the theoretical differences
  - See this paper

Security and Testing

- *Can be shown that probability of a security failure after* t *units of testing is about*

  E = K/t *where* K *is a constant*

- *This approximation holds over large range of* t
- *Then the "mean time between failures" is*

  MTBF = t/K

- *The good news: security improves with testing*
- *The bad news: security only improves* ***linearly*** *with testing!*

Security and Testing

- The "mean time between failures" is approximately

  $$\text{MTBF} = t/K$$

- To have 1,000,000 hours between security failures, must test 1,000,000 hours!
- Suppose **open source** project has $\text{MTBF} = t/K$
- If flaws in **closed source** are twice as hard to find, do we then have $\text{MTBF} = 2t/K$?
  - No! Testing not as effective $\text{MTBF} = 2(t/2)/K = t/K$
- The same result for open and closed source!

Security and Testing

- Closed source advocates might argue
  - Closed source has "open source" alpha testing, where flaws found at (higher) open source rate
  - Followed by closed source beta testing and use, giving attackers the (lower) closed source rate
  - Does this give closed source an advantage?
- Alpha testing is minor part of total testing
  - Recall, first to market advantage
  - Products rushed to market
- Probably no real advantage for closed source

Security and Testing

- No security difference between open and closed source?
- Provided that flaws are found "linearly"
- Is this valid?
  - Empirical results show security improves linearly with testing
  - Conventional wisdom is that this is the case for large and complex software systems

Security and Testing

- The fundamental problem
  - Good guys must find (almost) all flaws
  - Bad guy only needs 1 (exploitable) flaw
- Software reliability far more difficult in security than elsewhere
- How much more difficult?
  - See the next slide…

Security Testing: Do the Math

- Recall that MTBF = t/K
- Suppose 10^6 security flaws in some software
  - Say, Windows XP
- Suppose each bug has MTBF of 10^9 hours
- Expect to find 1 bug for every 10^3 hours testing
- Good guys spend 10^7 hours testing: **find 10^4 bugs**
  - Good guys have found 1% of all the bugs
- Trudy spends 10^3 hours of testing: **finds 1 bug**
- Chance good guys found Trudy's bug is only **1%** !!!

Software Development

- General software development model
  - Specify
  - Design
  - Implement
  - Test
  - Review
  - Document
  - Manage
  - Maintain

Secure Software Development

- Goal: move away from "penetrate and patch"
- Penetrate and patch will always exist
  - But if more care taken in development, then fewer and less severe flaws to patch
- Secure software development not easy
- Much more time and effort required thru entire development process
- Today, little economic incentive for this!

Secure Software Development

- We briefly discuss the following
  - Design
  - Hazard analysis
  - Peer review
  - Testing
  - Configuration management
  - Postmortem for mistakes

Design

- Careful initial design
- Try to avoid high-level errors
  - Such errors may be impossible to correct later
  - Certainly costly to correct these errors later
- Verify assumptions, protocols, etc.
- Usually informal approach is used
- Formal methods
  - Possible to rigorously **prove** design is correct
  - In practice, only works in simple cases

Hazard Analysis

- Hazard analysis (or threat modeling)
  - Develop hazard list
  - List of what ifs
  - Schneier's "attack tree"
- Many formal approaches
  - Hazard and operability studies (HAZOP)
  - Failure modes and effective analysis (FMEA)
  - Fault tree analysis (FTA)

Peer Review

- Three levels of peer review
  - Review (informal)
  - Walk-through (semi-formal)
  - Inspection (formal)
- Each level of review is important
- Much evidence that peer review is effective
- Although programmers might not like it!

Levels of Testing

- Module testing   test each small section of code
- Component testing   test combinations of a few modules
- Unit testing   combine several components for testing
- Integration testing   put everything together and test

Types of Testing

- Function testing   verify that system functions as it is supposed to
- Performance testing   other requirements such as speed, resource use, etc.
- Acceptance testing   customer involved
- Installation testing   test at install time
- Regression testing   test after any change

Other Testing Issues

- Active fault detection
  - Don't wait for system to fail
  - Actively try to make it fail attackers will!
- Fault injection
  - Insert faults into the process
  - Even if no obvious way for such a fault to occur
- Bug injection
  - Insert bugs into code
  - See how many of injected bugs are found
  - Can use this to estimate number of bugs
  - Assumes injected bugs similar to unknown bugs

Testing Case History

- In one system with 184,000 lines of code
- Flaws found
  - 17.3% inspecting system design
  - 19.1% inspecting component design
  - 15.1% code inspection
  - 29.4% integration testing
  - 16.6% system and regression testing
- Conclusion: must do many kinds of testing
  - Overlapping testing is necessary
  - Provides a form of "defense in depth"

Security Testing: The Bottom Line

- ***Security testing*** is far more demanding than non-security testing
- Non-security testing does system do what it is supposed to?
- Security testing does system do what it is supposed to ***and nothing more***?
- Usually impossible to do exhaustive testing
- How much testing is enough?

Security Testing: The Bottom Line

- How much testing is enough?
- Recall MTBF = t/K
- Seems to imply testing is nearly hopeless!
- But there is some hope…
  - If we eliminate an entire class of flaws then statistical model breaks down
  - For example, if a single test (or a few tests) find all buffer overflows

Configuration Issues

- Types of changes
  - Minor changes   maintain daily functioning
  - Adaptive changes   modifications
  - Perfective changes   improvements
  - Preventive changes   no loss of performance
- Any change can introduce new flaws!

Postmortem

- After fixing any security flaw…
- Carefully analyze the flaw
- To learn from a mistake
  - Mistake must be analyzed and understood
  - Must make effort to avoid repeating mistake
- In security, **always** learn more when things go wrong than when they go right
- Postmortem may be the most under-used tool in all of security engineering!

Software Security

- First to market advantage
  - Also known as "network economics"
  - Security suffers as a result
  - Little economic incentive for secure software!
- **Penetrate and patch**
  - Fix code as security flaws are found
  - Fix can result in worse problems
  - Mostly done **after** code delivered
- Proper development can reduce flaws
  - But costly and time-consuming

Software and Security

- Even with best development practices, security flaws will still exist
- Absolute security is (almost) never possible
- So, it is not surprising that absolute software security is impossible
- The goal is to minimize and manage risks of software flaws
- Do not expect dramatic improvements in consumer software security anytime soon!