Engineering Secure Software

RISK-DRIVEN TEST PLANNING

Risk Management

- Beyond assessment
  - Assess: Enumerate, Prioritize, Discuss
  - Manage: Act on those discussions
- Mitigate risk
  - Every risk has a mitigation
  - Plan, plan, plan
  - Know the limitations of your solution
- Track risk
  - Effective mitigations?
  - Increased p(exploit)?
  - Increased asset value?

Top-Down Test Planning

- Start with the broad analysis of the domain
  - Goals
  - Assets
  - Top-down analysis (“forest-level”)
- Goals → Risks → Indicators → Tests
- Vulnerability-focused
  - Instead of exploit-focused
  - Too much functionality
  - Move on when the vulnerability is found
  - Valued assets are given a priority

Goals → Risks

- Goals
  - Overall objectives of the system
    - Business-focused objectives → revenue streams
    - User-focused objectives → branding
  - Constraints on the development
    e.g. release dates
  - Availability concerns
  - A product has a ***finite*** number of goals
- High-Level Risks
  - Directly map to 1+ objectives
  - Influenced by both p(vuln) & assets
  - A product has a ***finite*** number of high-level risks

Risks → Indicators

- How will we know that a high-level risk became a problem?
  - A measurable outcome of the system
  - What is the poor behavior of the system?
  - What are the potential underlying causes?
- E.g. downtime, asset exposure
- Indicators are potentially infinite

  …but three will get you very far

Indicators → Tests

- Given an indicator, how do we ensure that the indicator is avoided or satisfied?
  - Test for it!
  - Key: *specific expectations*
- Tests are even more infinite
- Might require more design & architecture work to execute this step

e.g. BlogReader Goals

- Goals:
  - (user) Provide pretty-looking formatting of user's blogs
  - (business) Make money via advertisements
  - Constraints: web-based configuration, mobile app, 6-month release cycles
  - Availability: 99.9% uptime
    (8.76 hours downtime/year)
- Assets
  - User subscription information (e.g. blog feeds)
  - Personal data (e.g. emails)
  - Social graph

e.g. BlogReader Risks →.. → Tests

- High-Level Risk: social graph disclosure
  - Indicator: APIs allow unauthorized access to social graph
  - Test: direct access to user friends should be denied
  - Test: votes logged are anonymized or digested
- High-Level Risk: availability is compromised
  - User-focused: users are unable to reach their feeds
  - Business-focused: customers move to a different tool
  - Indicators: high processor loads, full hard drives, downtime
  - Tests: stress tests for networking, disk activity, and crashes

When do I do what?

- At the requirements phase:
  - Goals
  - Risks
- At a high-level design phase (i.e. architecture)
  - Indicators
  - Some tests
- At a low-level design phase (incl. maintenance)
  - More Tests
- All the time
  - Track
  - “Bubble up” new risks from new test ideas

Bottom-Up Security Test Planning

- Step 1: Write down a lot of tests
  - Document it in short form
  - Doesn't have to be complete – just seeds for now
- Step 2: Group those tests into various categories
  - By assets
  - By functionality
  - By CIA consequences
  - By what your team requires to run the test
  - etc.
- Step 3: Revise the categories as a group
  - Missing groups?
  - Missing tests in a group?
- Step 4: Add more tests to each category

Benefits and Drawbacks

- Top-down security test planning
  - Benefit: tied to specific goals
  - Drawback: incomplete within the categories
    - “Just to check it off the list” syndrome
    - Miss out on planning for really creative tests
- Bottom-up security test planning
  - Benefit: gives you freedom to write your best tests immediately
  - Drawbacks: easy to miss stuff
    - Entire goals/categories/assets can get missed
    - Without proper grouping it disintegrates into your “bag of tricks”
    - Requires security expertise in the first place

12-Minute Test Plans

- Bottom-up test planning activity for today
- Make a GoogleDoc called “Test Planning”
  - Everyone at your table will be editing this doc simultaneously
    - So give everyone access
    - Make some whitespace
- Instructor will give you a well-known software system
  - **5 minutes**: individually, everyone write down as many security tests as you can think of
  - **7 minutes**: as a team, edit them together, categorize them
- We will do several of these, as time allows