Integrating Tools Into the SDLC

SecAppDev 2007

The problem

Too many organizations have either:

- Failed to try software security tools at all
- Tried tools, but became overwhelmed
  - Tools relegated to “shelfware”
  - Never got past “pilot study”

This is a loss for all parties involved!

What caused the failures?

Possible reasons include

- Simple lack of awareness
- Tried to use tools too late in the lifecycle
- Expected more from tool technologies than they can deliver
- Poor integration into the build process
- Cost
- Excessive learning curve

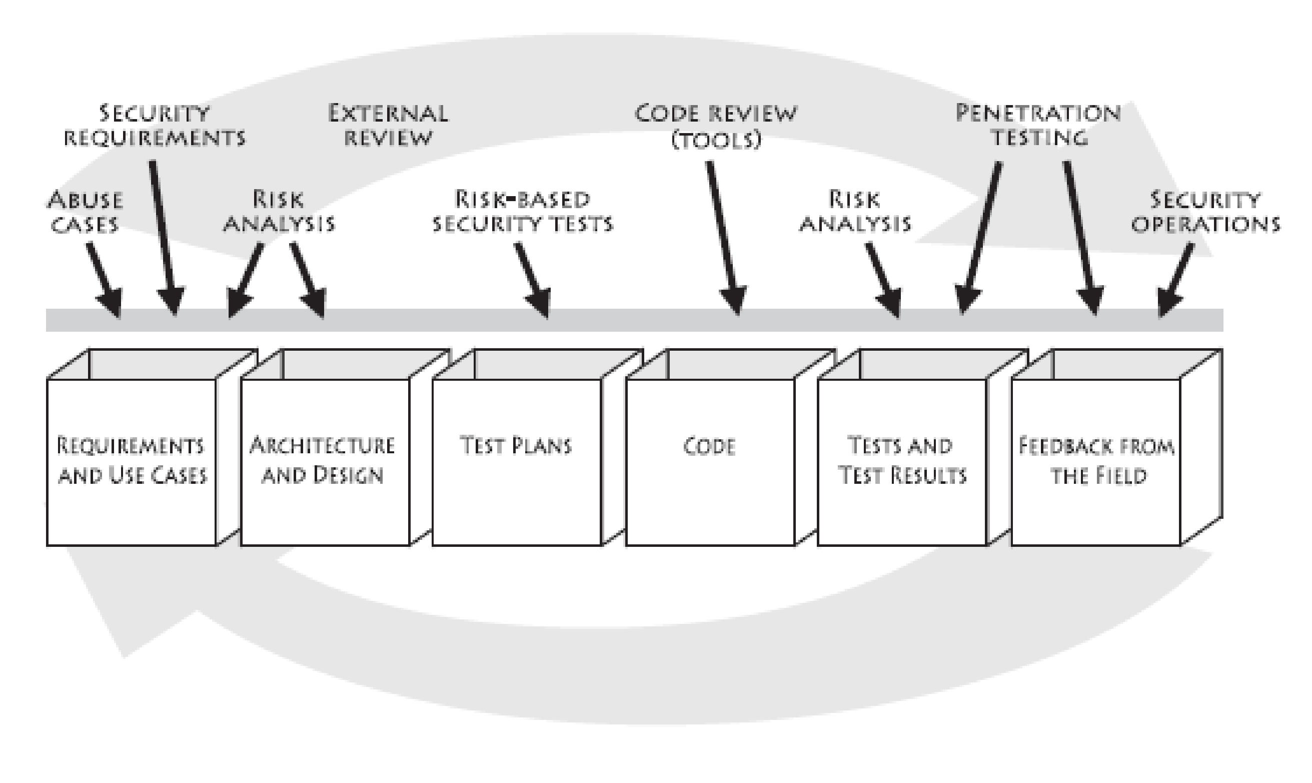
Let’s avoid those pitfalls

We’ll take a balanced view of the tools and how best to use them

We’ll also look at what tools cannot do for us

Process

Start by considering your process



Uneven distribution

In terms of “touchpoint” processes, the available tools are not spread evenly

- Most common tools are useful for testing
- Newer tools useful during code development
- Not so much available for “early” stages

Now, what is possible

Two *general* categories are available today

- IT security tools
- Software security tools

Hint: Consider too their origins in CIO and dev organizations

Infosec tools

Categories include

- Network port scanners
- Vulnerability scanners
- Application scanners
- Web application proxies
- Network sniffers

(For a great list, see http://sectools.org/)

Software security tools

Categories include

- Static code analysis tools
- Testing tools
  - Fuzzers
  - Interposition tools
  - System monitors
  - Process analyzers
  - Etc.

Utilization

- Let’s consider the applicability of each to our purposes
  - How best to apply the tool
  - What pitfalls to avoid
  - How to interpret the results

Network and vul scanners

- Usage: determine open and potentially vulnerable network services
  - Mainstay of “penetration testers”
  - Useful for verifying deployment environment
  - Validating on-going maintenance
  - Rarely directly valuable to developers
- Examples
  - Nmap, nessus, Metasploit, ISS, Core Impact, Retina

Application vul scanners –1

Category of black box test tools that attempts additional "application level" vul probes

- E.g., SQL injection, buffer overflows, cookie manipulation, Javascript tampering
- Increasing in popularity among pen testers
- Useful at verifying (web) app is not vulnerable to the most common attacks
- Moderately useful to developers

Application vul scanners –2

- Challenge is inverting finding into actionable dev guidance
- Danger in over reliance!
- Test coverage is very low (10-20% code is not uncommon)
  - Example: `if (mystate==FOO) {`
    ```
          printf(userstr);}
    ```
- Too often used in uninformed testing

Application vul scanners –3

Examples

- Watchfire’s Appscan, SPI Dynamics’ WebInspect, Nikto

Web app proxies –1

Interposition tools between browser and web app

- Exposes entire web session, data, scripts, etc., to the tester
- Ideal for verifying boundary conditions, script over reliance, etc.
- Another mainstay of pen testers

Web app proxies –2

  - Developers should also use these!
  - Useful for verifying web code, variables, cookies, etc.

Examples

  - Paros proxy, WebScarab

Network sniffers

- Essential tool for accurately capturing network traffic
  - Eavesdrops on network data
  - Encrypted protocols can be problematic
  - Lowest level tool to verify network communications
- Examples
  - Wireshark (formerly Ethereal), Kismet, Tcpdump, Cain and Abel

Fuzzers –1

Growing field of app testing that involves sending malformed data to/from app

- Tools, frameworks, and APIs are popping up
- "One size fits all" approach is highly problematic
  - Informed fuzzing vs. uninformed fuzzing
- Still early adoption among pen testers (arguably)
- Dev knowledge is necessary to get most of it

Fuzzers –2

  - Fuzzing can and should be done from unit to entire app tests
  - QA test team needs to acquire and learn

Examples

  - OWASP’s JBroFuzz, PEACH, SPI Fuzzer

“*At Microsoft, about 20 to 25 percent of security bugs are found through fuzzing a product before it is shipped*”

Interposition and monitors

- Conceptually similar to web app proxies and network sniffers, but work with stand-alone or client-server apps
  - Enables tester to watch and manipulate all system interaction
    - Sys calls, file i/o, registry keys
- Examples
  - Holodeck, filemon, regmon, AppVerif

Static code analysis

Peer (manual) review vs. automated

- Each has pros and cons
- Many organizations already do peer review
- Don’t lose sight of the benefits when adopting tools for automated review
- The value of mentoring is enormous

Static code analyzers –1

Review source code for common coding bugs

- A bit of history
  - 1999: First examples appear from research projects
    - E.g., ITS4, RATS, Flawfinder
    - Tokenize input streams and perform rudimentary signature analysis
    - Accurate at finding *strcpy()* and the like, but lacking context to really be useful

Static code analyzers –2

- 2001: “2nd generation” tools arrive
  - E.g., Fortify, Ounce Labs, Coverity
  - Parse and build abstract syntax tree for analysis
  - Enables execution flow, data flow, etc., traces
  - Significant leap forward, but much work remains
  - Hundreds of common bugs in several languages
  - Management tools for overseeing, measuring, and policy enforcement
  - Integration into popular IDEs
- Still, many are shelfware

Static code analyzers –3

- Biggest mistake is to dump entire src tree into tool and expect miracles
  - Increasingly being done by IT security
- Unreasonable expectation
- Consider instead
  - Give coders access to tool
  - Incorporate into nightly build process
  - Take many small steps instead of one big one

Static code analyzers –4

- *Then* do large scale analysis at project completion
- Possibly using more than one tool set

Selecting a static analyzer –1

Considerations abound

- Cost
  - Per seat
  - How many do you need?
- Infrastructure needed
- Language/technology support
- Knowledge base

Selecting a static analyzer –2

Management features

- Capabilities vary tremendously
    - Metrics, trending, visualization
    - Per project, team, person…
    - Policy centralization (next slide)
- What works best in your dev process and organizational culture?

Selecting a static analyzer –3

Policy centralization

- Most of the tools enable central policies
    - E.g., overriding a buffer overrun requires 2-person sign-off
- Consider these features carefully
    - Technical features and cultural impact to your org

Selecting a static analyzer –4

Extensibility

- All the commercial tools enable the user to custom build rules
  - Allows localization of rules that matter to *you*
  - Ensure the rule builder suits your needs
- What sort of learning curve will be required to get the most out of the tool?

Selecting a static analyzer –5

Consider a “bake-off”

- The vendors hate (but expect) this
- Start with a src tree you’ve already analyzed
  - And you know where the problems are
- Invite vendors to prove their tools on this code base
- Compare and contrast

Static analysis of binaries

- Tools and services just beginning to emerge
  - Many pros and cons
  - Src analysis nearly always preferable
  - Sometimes you don’t have src
  - Consider 3rd party code
- Examples
  - Veracode, AspectSecurity

Getting the most out of them –1

Regardless of the tools you choose, you should get the most of your investment

- Vendor-based tool training for key personnel
- Internal/external forums for sharing tips and pitfalls
  - Talk with others who have similar experiences
  - Be cautious about what you say in public
- Tech support from vendor

Getting the most out of them –2

- Test scenario development
  - Especially if your QA testers or IT security use the tools
  - Assist them in developing realistic test scenarios
  - Prioritize level of effort in descending risk priority order
    - This presumes you're doing risk analysis!

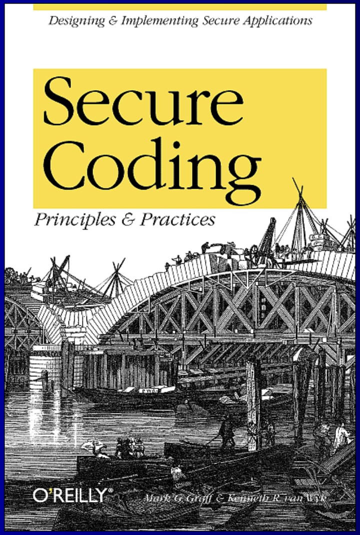
References

Some useful additional reading

- “The Security Development Lifecycle”, Michael Howard and Steve Lipner
- OWASP (http://www.owasp.org)
  - Webgoat, Webscarab, JBroFuzz, in particular
- Insecure.org’s “Top 100” list (http://sectools.org/)
- Fuzz testing tools and techniques http://www.hacksafe.com.au/blog/2006/08/21/fuzz-te
- System Internals (now owned by Microsoft) (http://www.sysinternals.com)

Kenneth R. van Wyk
KRvW Associates, LLC

Ken@KRvW.com
http://www.KRvW.com


Designing & Implementing Secure Applications
Secure Coding
Principles & Practices
O'REILLY
Mark G. Graff & Kenneth R. van Wyk