International Association
of Software Architects

Application Security Principles

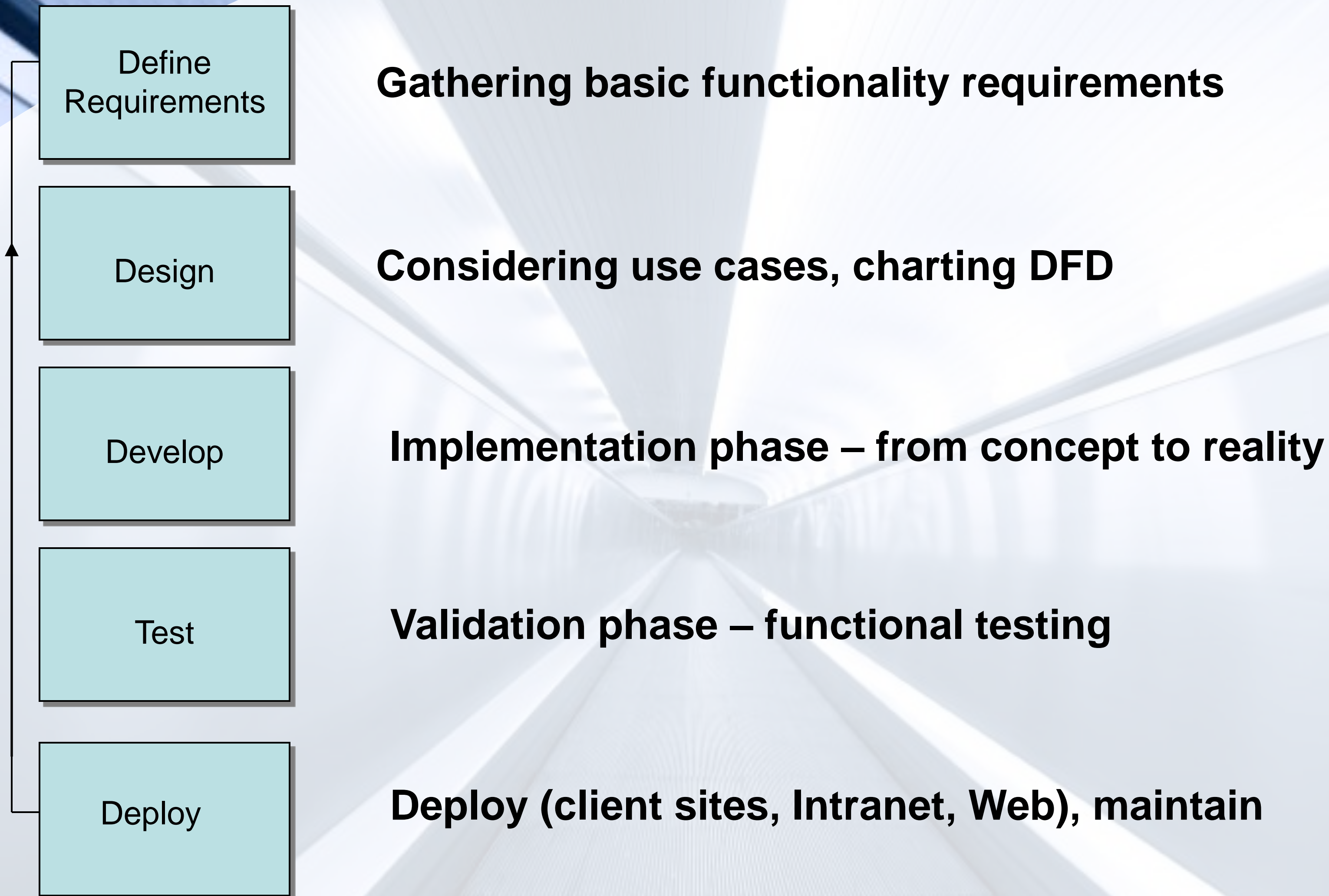
throughout the Software Development Lifecycle

Stephen Evans

Secure Application Services APAC

symantec™

Software Development Lifecycle



Gathering basic functionality requirements

Considering use cases, charting DFD

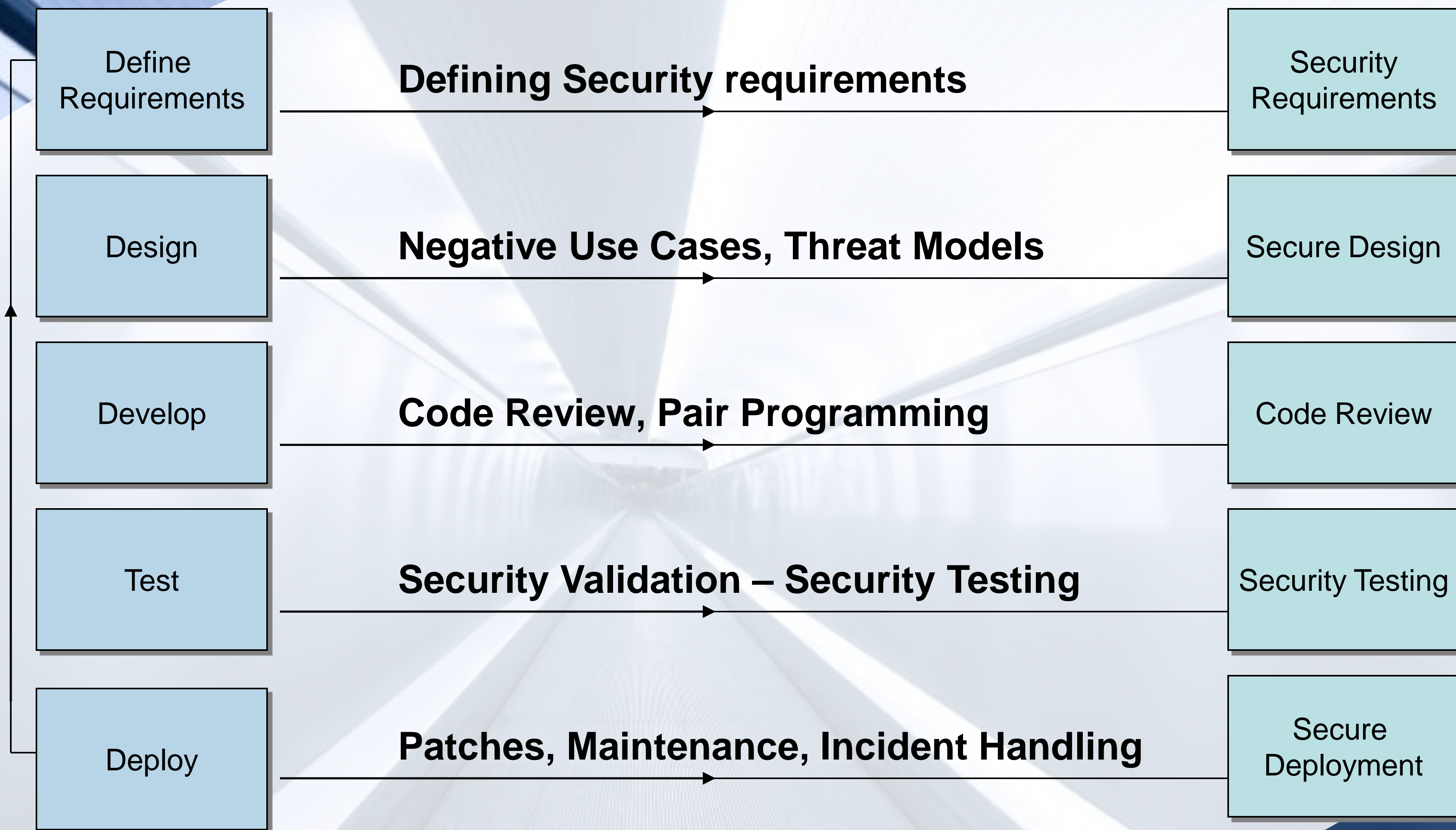
Implementation phase – from concept to reality

Validation phase – functional testing

Deploy (client sites, Intranet, Web), maintain

International Association of Software Architects

Software Development Lifecycle



Software Development Lifecycle

Conceptual phase:

- Bugs and faults captured in this stage –
  never existed

Natal phase:

- Bugs and faults captured in this stage –
  Incur implementation costs, possibly roll back design

[Proactive Security]

Existential Phase: **[Reactive Security]**

- Bugs and faults captured in this stage –
  Incident response, patches, patch management

Security Requirements

Secure Design

Code Review

Security Testing

Secure Deployment

Security Requirements

- Security Risk Profiling
- Security Requirements

Risk: Defined

Risk = Threat x Vulnerability x Cost

- Threat – frequency of potentially adverse events
- Vulnerability – likelihood of success of a particular threat against an organization
- Cost – total cost of the impact of a particular threat experienced by a vulnerable target

Security Risk Profile: defined

- An exercise to determine the risk rating associated with an application and its development
- Takes place in the beginning of the SDLC
- Output geared toward both Project Managers & Security Personel
- Output specifies security tasks to be carried out during the SDLC
- “How risky is this application?” questionnaire/discussion

Creating a risk measurement rating

- Each application is evaluated and receives a risk rating
- The risk measurement is based on several questions (10-15) tailored to each organization
- Each question will receive a risk score based on a scale (such as, low/med/high or 1-5)
- The total risk score gives a general risk rating of the application
- Compare risk ratings across an organization to effectively allocate security resources

Security Risk Profiles

- Application: Estimated calendar time for project completion? Size of the total project team? How many different physical locations will the application be deployed?
- Development: How many developers will be used? Update of an existing application or a brand new application? Application architecture design created internally or by a partner?
- Access & User: Will this application use credential access? Who will have access to this application externally? Will this application be available externally on the Internet?
- Application Processing: What information is processed by the application? What is the highest data classification of the data used by the application? Is data converted by the application? If so, what is the difficulty?
- Reputational Risk: Regulatory - What is the project's visability to regulators? Media - What is the project's visibility to the media? Public Relations - What is the project's visibility to the customers?

Risk Profile Analysis: Categorize Risk

- 75% and above: High Risk
  - High likelihood of application/data compromise and reputational damage
  - First applications addressed in corporate security budget
- 50% - 75%: Moderate Risk
  - Good possibility of application/data compromise and reputational damage
- Below 50%: Lower Risk
  - Low/medium chance of application/data compromise and reputational damage

Risk Profile Analysis: Security Reviews and the SDLC

Using the application's risk rating, the Project Manager can plan the appropriate security reviews during the SDLC.

	High Risk	Med. Risk	Low Risk
Application Risk Assess.	X	X	X
Security Requirements Review	X	X	
Threat Modeling	X	(optional)	
Security Design Review	X	X	
Security Code Review	X	(optional)	
Security Testing	X	X	X

Functional vs. Security Requirements

Functional Requirements

- Determine application functionality
- Based on traditional use cases
- Define logical constraints
- .. Are modeled after a “lawful” user

Security Requirements

- Constrain application functionality
- Are modeled after **AB**use cases
- Focus on the “else” rather than the “if”
- .. Are modeled after a “chaotic” user

Use Cases

Given the classic ATM use cases:

- Customer inserts card
- Customer enters PIN
- Customer asks for money
- Customer gets money
- Customer leaves

ABUse Cases

Customer inserts card

- Inserts a piece of plastic; inserts a forged card; attempts to bypass card

Customer enters PIN

- Does not enter PIN; enters longer PIN; brute forces PIN

Customer asks for money

- Asks for $10000; tries to overdraft; tries other operation

Customer gets money

- No communication to mainframe; hatch could be jammed

Customer leaves

- Attempts repeated transactions; customer attempts to break ATM; customer attempts to take ATM home

Establishing Security Requirements

- Gather up all valid use cases
- For each use case consider:
  - What am I assuming? Implicitly or explicitly?
  - What constraints have I placed on the user?
  - What could possibly go wrong?
- Allow yourself to go off on tangents
- Don’t get too specific as to attacks
  - Threat Modeling, next, will give you time for that
- Consider all possibilities
  - Most may be discarded, but consider all

Secure Design

- Threat Modeling
- Security Design Review

Threat Modeling

- Threat modeling analyzes theoretical risks and “attack vectors”
- Attack vectors define:
  - Direction: avenue of attack
  - Quantity: severity of attack
- Direction scoped by the STRIDE methodology

Defining Attack Vectors : STRIDE

- Popular Methodology used by Microsoft
- Defines common attack vector classes

Vector Class	Examples
Spoofing	Session Hijacking, MiM attacks
Tampering	Input malformation, cookie poisoning
Repudiation	Rogue clients, Transaction disavowal
Info. disclosure	Privacy leaks, overly descr. errors
Denial of Service	Broken exception handling
Esc. of Privilege	Broken access control

- Not all classes are necessarily applicable in your app

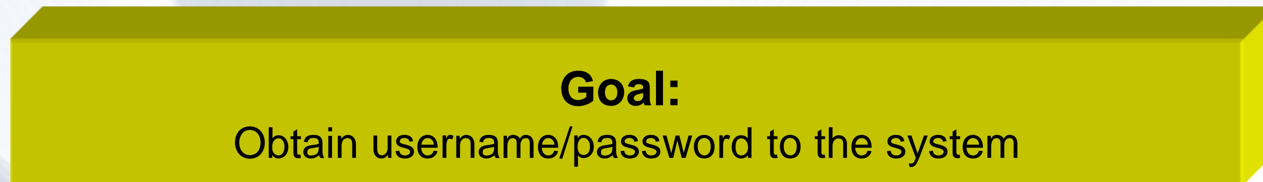
Quantifying Attack Vectors : DREAD

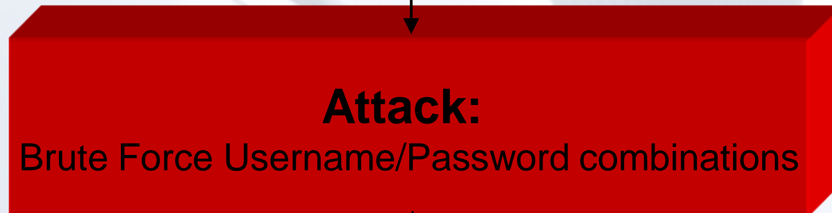
- Another Popular Methodology used by Microsoft
- Defines a metric to assign values to vectors

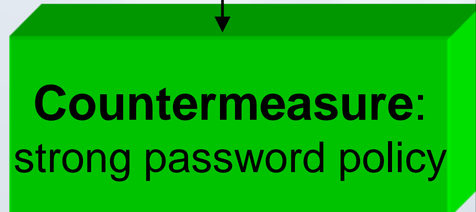
Vector Class	Examples
Damage potential	Impact of successful exploitation
Reproducibility	Special settings, or mitigating circumstances
Exploitability	Likelihood of successful exploitation
Affected users	%-age and class of users affected
Discoverability	Likelihood of uncovering vulny

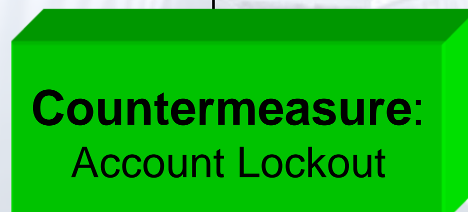
- Metric may be used to prioritize attacks

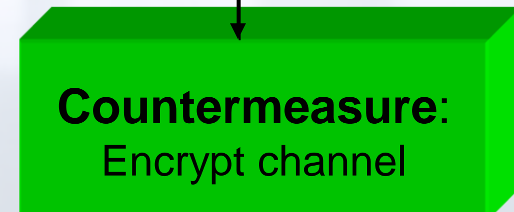
Defining Attack Vectors: Attack Trees

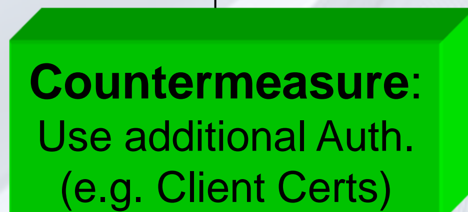












Security Design Review: Areas of Analysis

- Areas of Analysis contain many topics to be reviewed and analyzed
- Topics follow company standard framework for secure design
- Follow security industry best practices if no company standards or defined security policies/guidelines
- Results in list of recommendations to current design

Areas of Analysis

- Authentication
- Authorization & Access Control
- Data integrity
- Error and exception handling
- Monitoring and logging
- Cryptography and encryption
- Database security
- Privacy, confidentiality and segmentation
- Web security
- Product Security

International Association of Software Architects

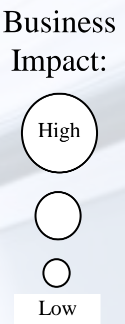
Design Review: Business Impact Matrix

Business Impact Matrix — Summary of Findings

LEVEL OF EFFORT TO REMEDIATE: LOW / HIGH

DEGREE OF RISK/EASE OF EXPLOIT: HIGH / LOW

	LOW effort	HIGH effort
HIGH risk	**Red flag: fix immediately**	**Red flag: plan to remediate**
	Data validation errors; Sensitive info manipulation and discovery; Training and awareness; Authentication errors; Authorization errors; Cross-site scripting; Dead code	Sec dev processes; Logging; Authorization features; Authentication features
LOW risk	Poor password handling; Bad error messages	Security metrics; User and admin app colocation; DB/App controls and segmentation
	Fix at client's discretion	**Bear risk**, or fix at client's discretion



Each finding's x-y position in the Business Impact Matrix indicates the relative risk and likelihood of exploit (vertical axis) and the effort required to remediate (horizontal axis). The circle diameter signifies the overall impact on your business and brand value.

Code Review

- Approaches
- Tools

What is Code Review?

- Security bugs may stem from many reasons:
  - Improper use of language API calls (for example, Strcpy)
  - Incorrect framework/class utilization (as in, Java/.Net)
  - Design bugs or use-cases that were not considered
- Project source is carefully scrutinized, looking for:
  - Coding errors
  - Dangerous API calls
  - Implementation faults
  - Design-Level and Logic security problems

- Challenges:
  - Optimally, code review will obtain 100% coverage
  - Practically, this is almost never achieved
  - Methodologies have been devised to max efficiency

Approaches: Low Hanging Fruit

- Scan code for potential buffer overflows
  - Insecure copy operations: strcpy/strcat
  - Improper formatting: sprintf
  - Insecure input methods: scanf, read, recv
- Find the "easy" bugs
  - Variable format strings: *printf(var); /* C/C++ */
  - malloc()/free() pairings /* C/C++ */
  - Improperly escaped input /* SQL injection, Null Bytes */
  - Insecure system calls /* all languages */
  - Command injection /* all languages, system/exec */
  - Directory Traversals /* all languages, file input */

Approaches: API Handling

- Focus on specific API calls:
  - Object Creation (Win32 CreateXXX, fopen…)
  - System escapes (exec, CreateProcess, system())
  - Dangerous APIs (str* functions, JNI, Unmanaged code)
  - Third Party/other component API calls
- Validate all API return codes
  - Make sure API calls are assigned as an lvalue

Approaches: Danger Zones

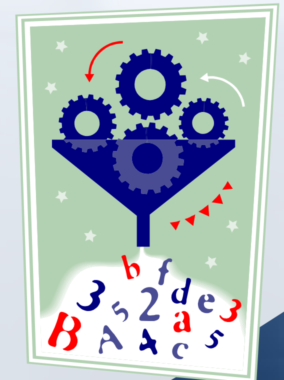
- Extra care is given to sensitive segments:
  - Authentication logic
  - Authorization logic
  - Cryptography-oriented code
  - Integer Arithmetic
  - Input handling
  - Exception Handling
  - Multi-Threaded code
- At the expense of static segments:
  - Functions with no input
  - Constant code paths (with no flow control)

Approaches: Pair Programming & Review

- A programmer is likely to overlook his own faults
- Reviewer is a different person, validating:
  - Design was properly implemented
  - Patterns were followed
  - Code was correctly annotated and commented
  - Assumptions made in code documented and validated

Approaches: Manual Tools

- Text processing utilities are especially useful:
  - grep: clever regular expressions to find:
    - Dangerous APIs (e.g. “egrep ‘str(cpy|cat)’”)
    - Format string Bugs (e.g “grep ‘printf’ | grep –v \” “)
  - find: quickly find header files, or external resources
- Use IDE “find in files”
  - find variable/function definitions



Approaches: Automated Tools

Automatic tools may often be used

- Secure Software's (now Fortify) RATS
  - Rough Auditing Tool for Security scans C, C++, Perl, PHP and Python source code
- FortifySoftware's Fortify Source Code Analysis Suite
- Microsoft's FxCop
  - Analyzes .NET managed assemblies
- Can jumpstart code review with penetration test results

Security Testing

- Quality Assurance vs. Security Testing
- Code Review and Security Testing
- Penetration Testing
- Tools

Security Testing vs. QA

- QA testing tests Functional Requirements
  - Making sure functionality is “as documented”
  - Bugs are defined as intended functionality that differs from the actual functionality
  - Program does not do **LESS** than it is supposed to
- Security Testing tests Security Requirements
  - Making sure the program does not exceed its design
  - Faults are defined as actual functionality that differs from the intended functionality
  - Program does not do **MORE** than it is supposed to

Code Review vs. Security Testing

- Security Testing complements Code Review
  - Code Review is a WHITE BOX approach
    - Useful only when the source code is available
  - Code Review may be severely limited by size
    - Automated CR may yield false positives/negatives
    - Manual CR is extremely time consuming
- Security Testing is a BLACK or GRAY BOX approach
    - Always possible, even on closed source
    - Insider knowledge helps, but not a prerequisite
- Security Testing can often be automated

Security Test Cases

- Incorporate “abuse cases” conceived during design
- Focus on boundary conditions:
  - Large (or obviously invalid) input
  - Border-range integers
  - Metacharacters (anything non-alphanumeric)
  - Timeouts
  - Limited resource availability
  - High system load

Penetration Testing

- Vulnerability classification schemes
  - OWASP Top 10, WASC, and Fortify donation to OWASP
- Vulnerability scanning vs. penetration testing
- Black hat hacking vs. pentesting as white hat testing and application testing
- Non-intrusive vs. intrusive
- What pentesting can't do:
  - Conclude that an application or system is safe
- What pentesting can do:
  - Find design & implementation security issues
  - Verify that configuration & hardening is done properly

Testing Tools

- Virtualization Software
  - VMWare, Virtual PC
- Unit Testing software
  - JUnit, NUnit, C++Test, …
- HTTP Tools
  - Browser Plugins
  - Application Proxies (Paros, WebScarab, BurpSuite)
- Input “Fuzzers”
  - Automated Tools (SPI Dynamics, etc.)
  - Custom Scripts (Perl, Python, etc.)

More Testing Tools

- Low Level Tools
  - Ethereal/Wireshark
  - TCPDump/TCPReplay
  - Netcat, Nmap, SNORT
- Vulnerability Scanners
  - Nikto
  - Wikto
  - Nessus
  - Metasploit

Secure Deployment

- Configuration
- Deployment
- Maintenance

Configuration

- Should be simple and easy to understand
- Configuration files should be properly protected
  - An application's management program should have read and write access to the configuration files
  - The application should have read access to the configuration files
  - Other users and groups should be denied any access to the configuration files
- Initialization files, if any, should be protected by the file system and stored where only authorized administrators can access them
- Access privileges should be limited by default until configured otherwise

Deployment: Environment Hardening

- It is critical to harden the operating system where the application will reside
- Strip out all unnecessary functionality
- Create standardized host builds
- Harden the overall operating environment as well such as:
  - Router configurations, firewalls, etc.
  - Physical security

Deployment: Security Documentation

- Document and map security requirements to installed features, modules, etc.
- Document all security features fully:
  - The security aspects and configurations of the application
  - All configuration settings that have security implications
  - The security ramifications of enabling any supported feature
  - Areas where privacy compliance is important

Deployment: For Administrators

- Make sure they have the tools to secure it
- Make sure that they have access to the information needed to make configuration decisions
- Split administrative tasks among different administrative roles
- Use admin or root privilege as little as possible, both on the system and within the application
- Enforce best security practices on the administrative account

Maintenance: Logging / Auditing

- Log to a local log file or system log file
- Also log to a central logging server (hackers modify local log files to cover their tracks)
- Ensure synchronized time across log machines for log file correlation & for forensic readiness
- Logging Configuration:
  - Regulate the amount of information that is logged
  - Log critical information under normal operation
  - Reconfigure to log extensively when troubleshooting

Maintenance: Logging / Auditing

- Logging Sources: The documentation should identify:
  - What software modules produce log messages
  - To what dirs and files they write these log messages
- Log Messages and their meaning: The vendor docs should provide a table with the following info accompanying the message ID:
  - The event statement
  - A brief explanation of the event
  - The severity level of the event
  - Recommended action, particularly for higher severity events

Maintenance: Monitoring / Administration

- Provide information that allows the admin to:
  - Detect suspicious activity
  - Corroborate and correlate suspicious activity
  - Demonstrate accountability for that activity
- Remote Management: If allowed, follow strict security practices:
  - Authenticate the client (account and machine) using multi-factor authentication controls
  - Allow access only to legitimate accounts/machines
  - Use an encrypted channel for all communication on the management interface

Maintenance: Patches & Updates

- Updates and patches delivered electronically should be transmitted and executed over secure channels and have technical controls to ensure their integrity (e.g., digital signatures).
- The procedures for procuring and applying software updates and patches should be clearly documented
- A penetration test is recommended after each major software upgrade

Maintenance: Incident Response

- Develop or recommend procedures for reporting known/suspected vulnerabilities and providing workarounds, recommendations and mitigating strategies
- CERT provides excellent resources, documentation, and training to establish a computer security incident response team, or CSIRT (http://www.cert.org/csirts/)

Summary: Conclusions

- Security is a property, not a feature. It's very difficult to bolt it in on at the end of the SDLC
- Security features add complexity, which increases cost and project duration
- Secure application principles can be added bit-by-bit (i.e. CLASP)
- The goal is a baked-in security process from beginning to end that becomes repeatable and measurable.

References

- http://seclists.org
- OWASP (www.owasp.org)
- Books
- Blogs

International Association
of Software Architects

Thank You

symantec™