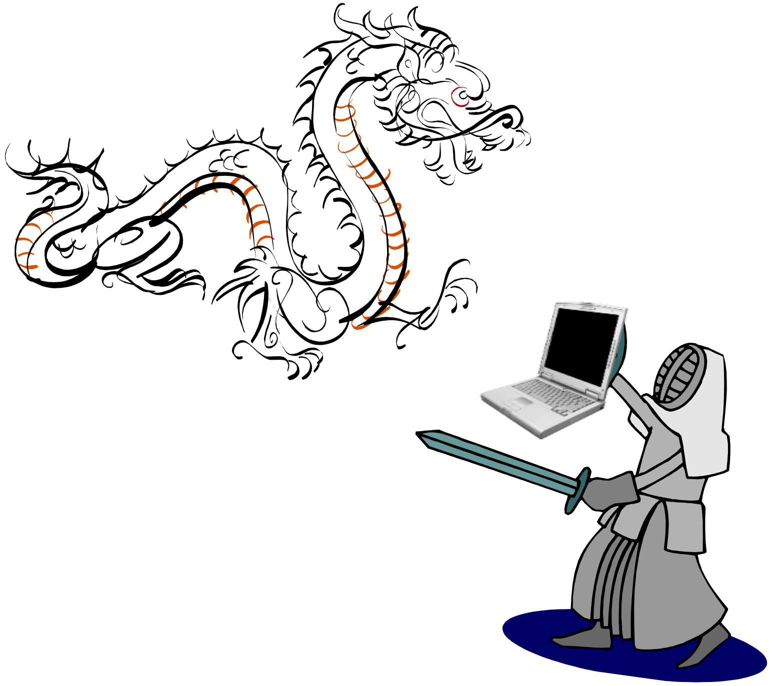


Creating Secure Code - Principles

CSC – Java
Course Goals

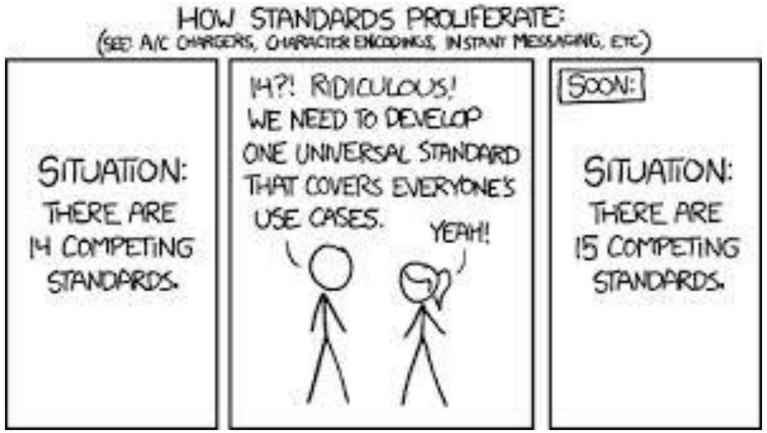
- After taking this course, you will be better able to develop secure Java applications by:
  - Knowing and applying the Principles of Secure Coding
  - Having a better understanding of the causes of common vulnerabilities and the methods for preventing them
  - Being able to recognize opportunities to apply secure coding principles
  - Being able to remediate security vulnerabilities by applying secure coding principles

Image courtesty of: http://tweakyourbiz.com/growth/files/shutterstock-Goal-setting.jpg

- Creating Secure Code - Principles
  - Understanding Secure Coding Principles
  - Common Secure Coding Principles
  - Summary

Understanding Secure Coding Principles

- The common secure coding principles are have been known for more than a decade
- They have changed over time as the understanding of application security has improved
- This list is based on a variety of sources
  - OWASP(http://owasp.org/index.php/Secure_Coding_Principles)
  - CERT (http://securecoding.cert.org)
  - Personal experience

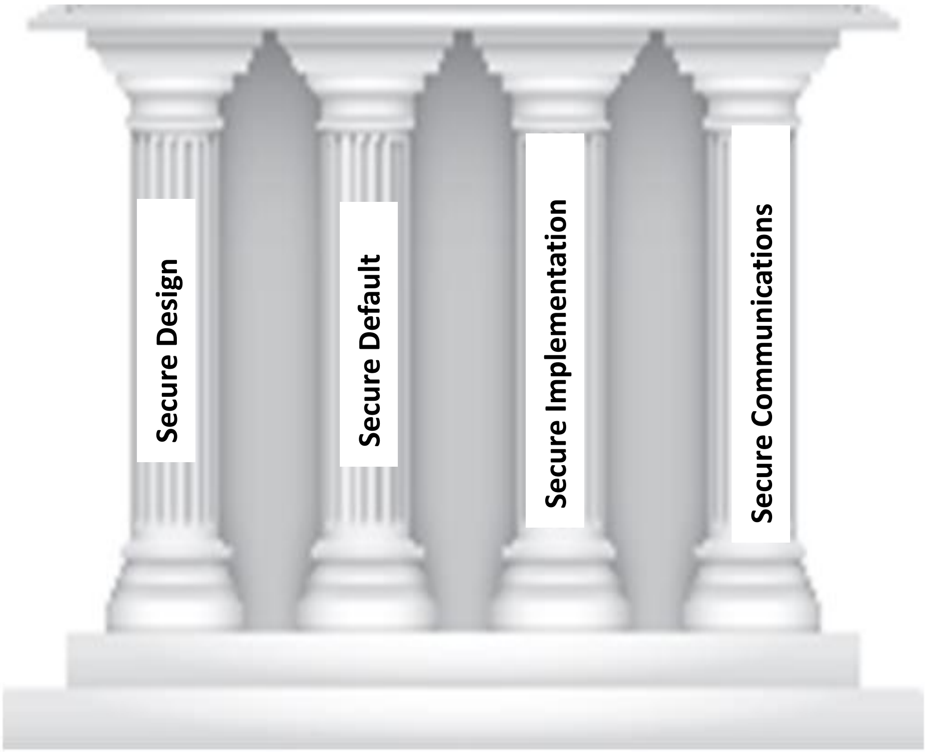


http://blogs.fsfe.org/mk/files/2014/02/xkcd-standards.png

Understanding Secure Coding Principles

- You may remember that the Four Pillars of Software Security
- These are the goals of Security Engineering
- Create an application that is:

  - Secure by Design
  - Secure by Default
  - Secure by Implementation
  - Secure by Communication



Understanding Secure Coding Principles

- The Secure Coding Principles could be described as Laws or Rules that if followed, will lead to the desired outcomes
- Each is described as a security design pattern, but they are less formal in nature than a design pattern

- Creating Secure Code - Principles
  - Understanding Secure Coding Principles
  - Common Secure Coding Principles
  - Summary

Common Secure Coding Principles
The Principles

- Secure By Design
  - Establish Trust Boundaries
  - Don't Reinvent the Wheel
  - Economy of Mechanism
  - Trust Reluctance
  - Open Design
  - Minimize the Attack Surface
  - Secure the Weakest Link
- Secure By Default
  - Use Least Privilege
  - Use Default Deny
  - Fail Securely
- Secure By Communication
  - Secure Trust Relationships
- Secure by Implementation
  - Psychological Acceptability
  - Least Common Mechanism
  - Validate Inputs
  - Secure Data at Rest
  - Prevent Bypass Attacks
  - Audit and Verify
  - Defense in Depth

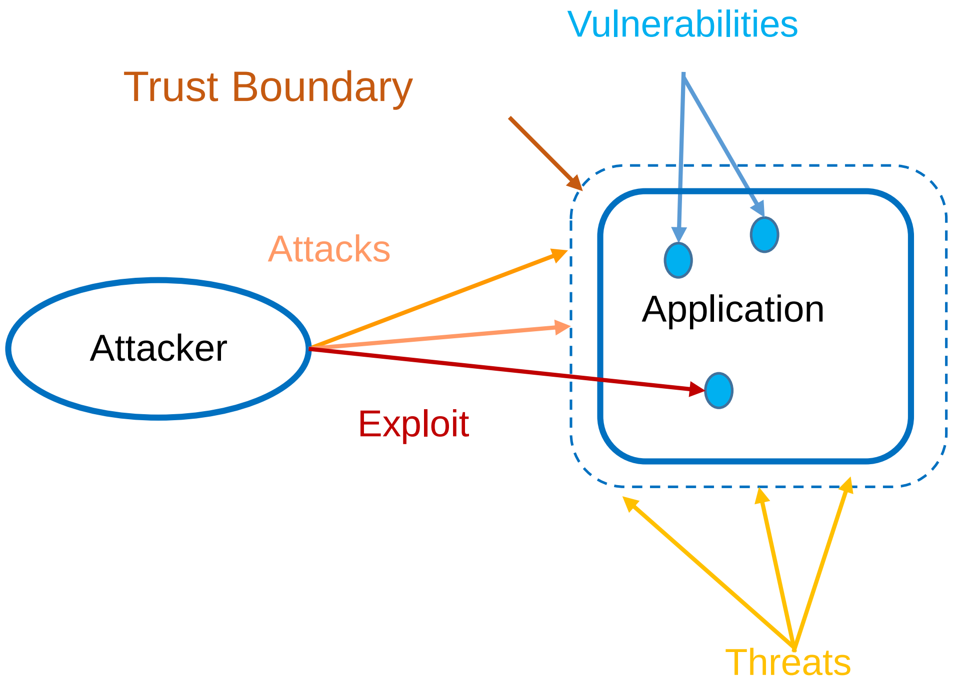
Creating Secure Code - Principles - Java

Secure By Design

- Establish Trust Boundaries
- Don't Reinvent the Wheel
- Economy of Mechanism
- Trust Reluctance
- Open Design
- Minimize the Attack Surface
- Secure the Weakest Link

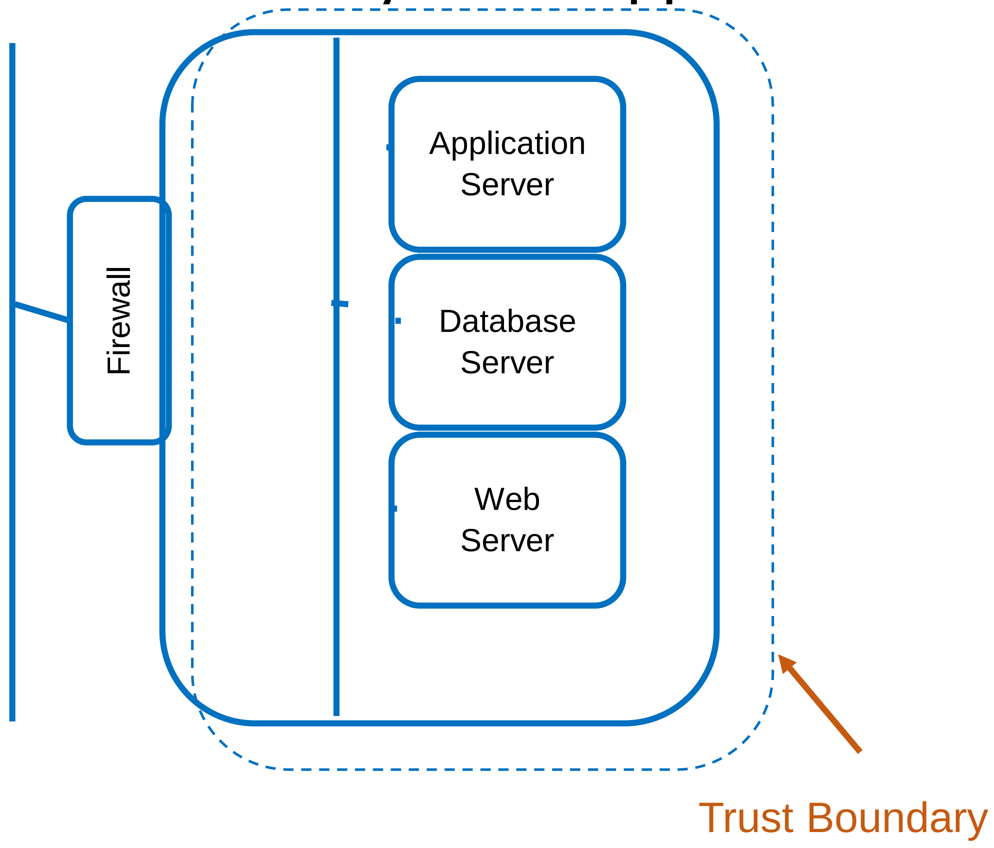
Common Secure Coding Principles
Establish Trust Boundaries - Introduction

- If you recall this diagram, it shows the Trust Boundary around the application
  - The Trust Boundary is an imaginary border that architects create
  - By making everything inside it trustworthy
  - By definition
    - All systems inside are trusted
    - All software inside is trusted
    - All data inside is trusted
  - The Trust Boundary only exists if the system is designed and implemented to create and protect it



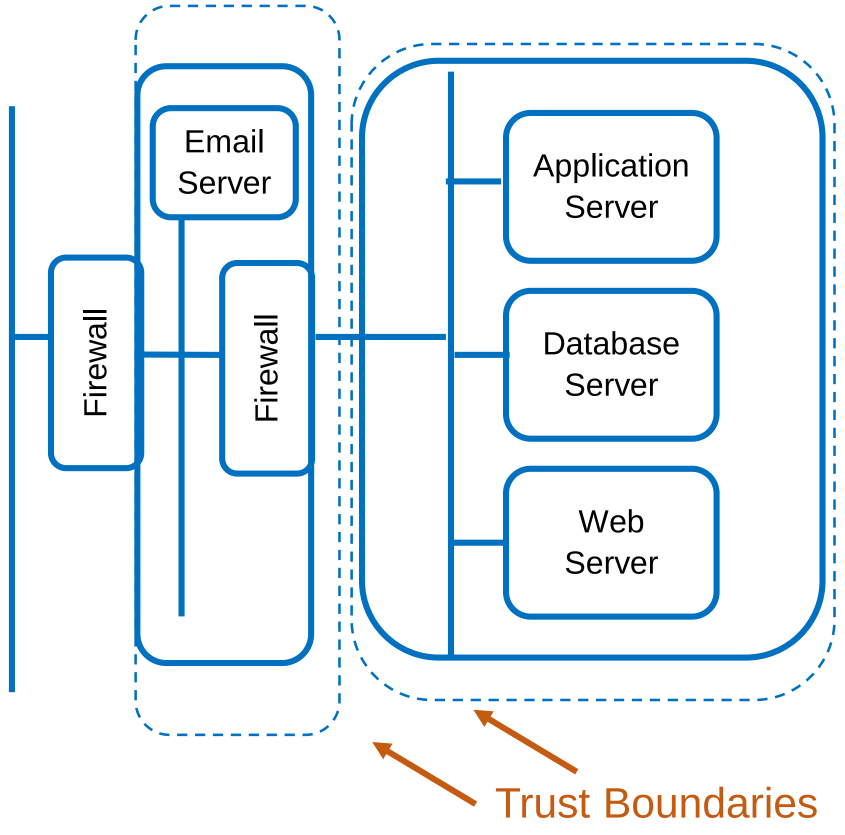
Common Secure Coding Principles
Establish Trust Boundaries - Introduction

- Trust Boundaries serve an important purpose in the SDL by providing a vehicle for answering many questions about the security of an application.
  - In this situation, everything inside of the TB is trusted
  - Someone wants to add an email server inside the TB, do you allow it?
  - Email servers talk to many systems outside of your system, and they accept content that is often questionable
  - If you don't feel that you can make the email server completely secured and trusted, then the answer is no



Common Secure Coding Principles
Establish Trust Boundaries - Introduction

- One solution is to create two Trust Boundaries
  - You have the original TB in which everything is completely trustworthy
  - And a second one that includes the email server and possible some other things that are secured, but not as trustworthy as needed for the Application Trust Boundary
  - Sometimes called a Demilitarized Zone or DMZ



Common Secure Coding Principles
Establish Trust Boundaries - Introduction

- It is not uncommon to have many Trust Boundaries defining different security needs around a large system
  - The level of trustworthiness is what you define it to be
  - Create Trust Boundaries where necessary to help make decisions about security issues

Dumpty said in rather a scornful tone, "it means just what I choose it to mean -- neither more nor less."

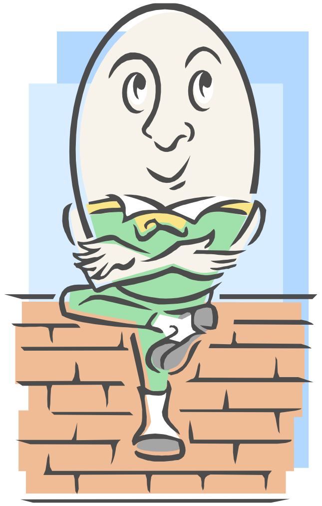
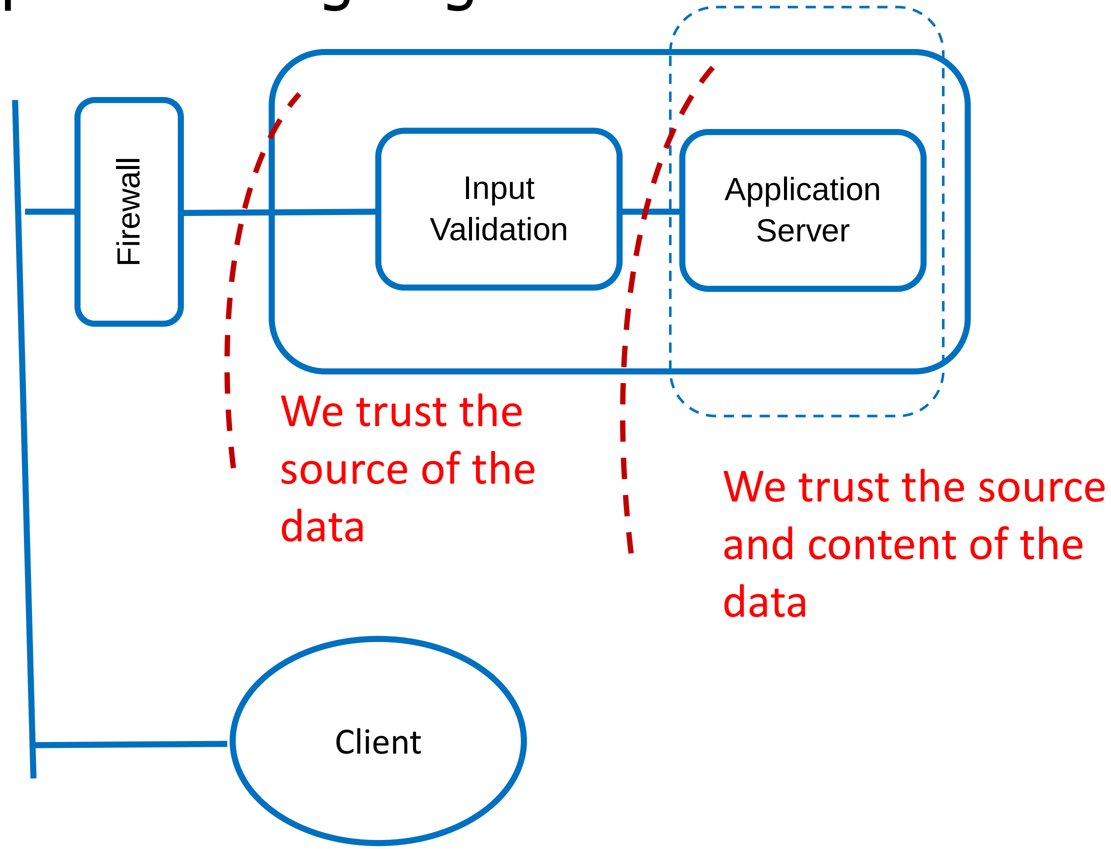


Image courtesy of: http://support.perkins.org/images/content/pagebuilder/humptydumpty.jpg

Common Secure Coding Principles
Establish Trust Boundaries - Introduction

- Trust Boundaries are logical constructs
- For example, your Trust Boundary for input handling might look like this

  - Input Validation obviously takes place on the application server
  - But logically, the data can't be trusted until it is validated



What might the Trust Boundary be for the firewall?

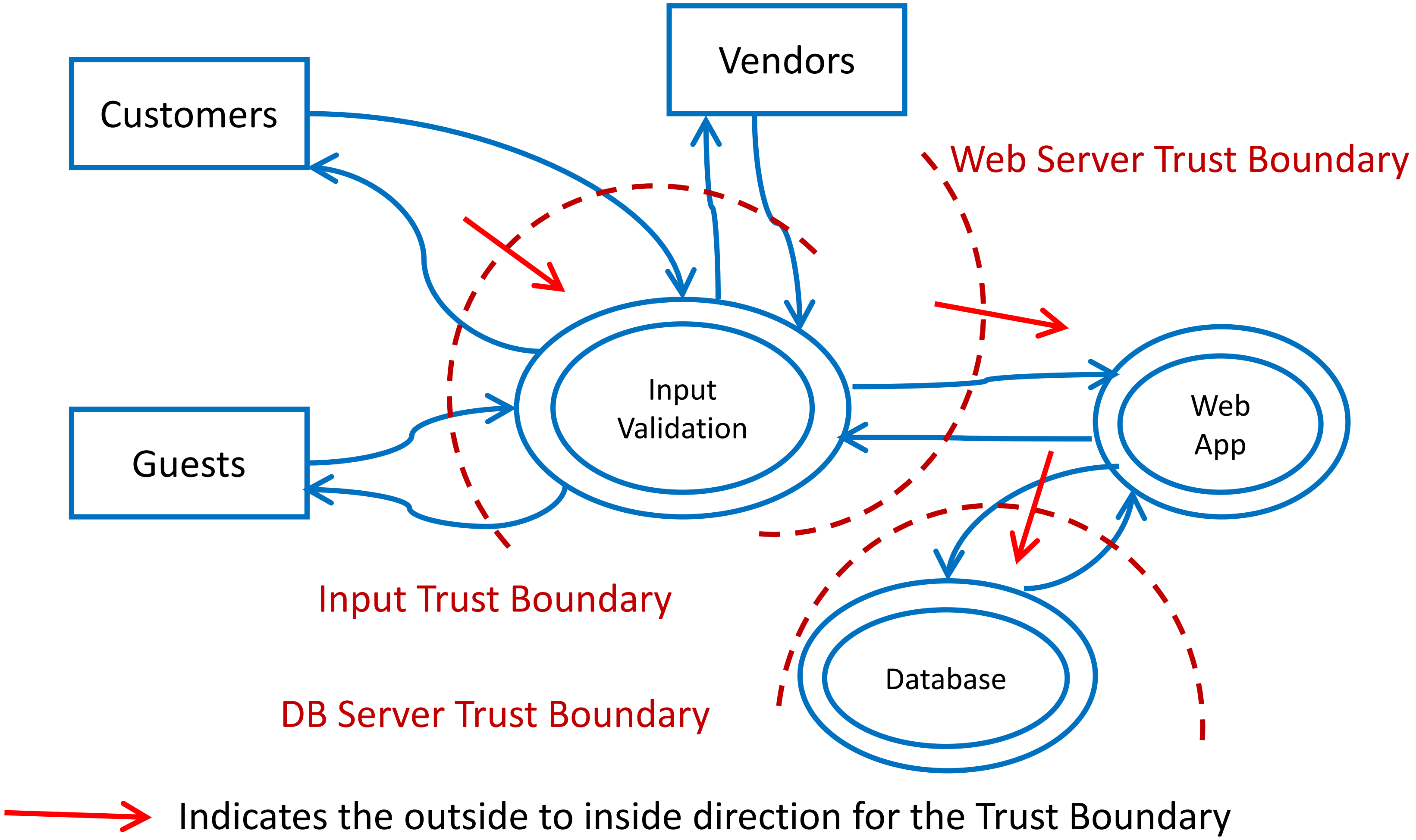
Common Secure Coding Principles
Establish Trust Boundaries – Security Design Pattern

- Alias: None
- Some Forces:
  - There is often confusion over the what levels of security are required in various parts of a system
  - Security requirements get neglected once the Threat Models are created
- Consequences
  - Trust Boundaries are easily defined and understood
  - They work well with Data Flow Diagrams
  - Better understanding of the security needed in system components
  - Improved handling of inter-component trust relationships

Common Secure Coding Principles
Establish Trust Boundaries - Application

- During the design phase, you will use
  - Threat Models, risk analyses and quality gates to determine what level of security is required
    - For systems
    - For applications
    - For data
    - For users
  - Trust Boundaries are a means of describing the security requirements

Common Secure Coding Principles
Establish Trust Boundaries - Application



Common Secure Coding Principles
Establish Trust Boundaries - Application

- From the diagram and the requirements you can define what data is allowed inside a Trust Boundary
- Input Validation Trust Boundary requirements
  - Input accepted from any connection on port 8080
  - Input accepted only from known IP addresses on port 443
  - Vendors are required to use SSL
  - Data is not assumed to be in any form
  - All payment card data must be sent over SSL
  - All credit card data is masked by the firewall

What is another common restriction on the Input Validation Trust Boundary?

Common Secure Coding Principles
Establish Trust Boundaries - Application

- Web App Trust Boundary requirements
  - All inputs must be sanitized for all potential attack vectors
  - Guest accounts have limited access inside
  - Vendor and Customer streams will be separated by a load balancer
  - And so on
- When complete, the Trust Boundaries tells you everything you need to know about the levels of trust that are required
- They help you make design decisions, settle differences and review requirements
- Trust Boundaries are typically created during or after the Threat Modeling process

Creating Secure Code - Principles - Java

Secure By Design

- Establish Trust Boundaries
- Don't Reinvent the Wheel
- Economy of Mechanism
- Trust Reluctance
- Open Design
- Minimize the Attack Surface
- Secure the Weakest Link

Common Secure Coding Principles
Don't Reinvent The Wheel - Introduction

- There is a lot of code available
  - Some of it has been written by talented people
  - Some of it has been thoroughly vetted and tested
  - Some of it is known to be secure or more secure than untest
  - Check the Common Vulnerability Database for current vulnerabilities
    - http://nvd.nist.gov
- When there is good security code available, don't try to create your own
  - Security is difficult to do
  - Mature frameworks can provide features you wouldn't have time to develop

http://images.thezooom.com/uploads/2012/08/Dont-reinvent-the-wheel.jpg

Common Secure Coding Principles
Don't Reinvent The Wheel - Examples

- Cryptography is difficult to implement well
  - If you are creating your own cryptography methods, do you have the numerical expertise to know it is secure?
  - If you are implementing existing algorithms, can you get it perfectly?
  - Anything less than perfect will be vulnerable
  - The amount of time you invest will be far larger than the time required to use an existing library, which is likely free
  - If you intend to seek any compliance certifications, they will require the use of a certified cryptography library
  - FIPS-140 list of certified libraries

Common Secure Coding Principles
Don't Reinvent The Wheel - Examples

- Consider using an Application Framework
  - Spring MVC, Grails, Google Web Toolkit, Spring, Java Server Faces
  - Many have built-in support for
    - An extensive authentication and session model
    - Input validation
    - SQL Injection avoidance
    - XSS avoidance
    - Path traversal avoidance
    - Cross-site Request Forgery protection
  - This can save you time and the code is already well-tested

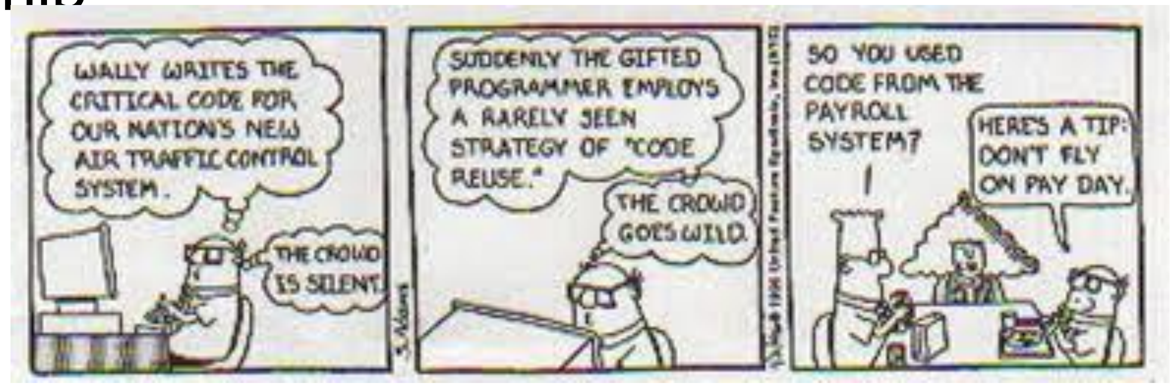
Common Secure Coding Principles
Don't Reinvent The Wheel – Security Design Pattern

- Alias: None
- Forces:
  - There is a temptation among developers to create their own solutions
  - Estimates of time for developing software are notoriously under-stated
  - Proven software is typically faster to implement and more secure
  - Existing software often provides additional facilities
- Consequences:
  - Higher levels of security
  - Access to improved features

What percentage of software projects meet time commitments? Why?

Common Secure Coding Principles
Don't Reinvent The Wheel – Tips

- Parts of a project should be evaluated for their security impact
  - High security impact components (cryptography, secure communications, key management)
  - High project risk
  - High complexity (search engines, databases)
  - Highly common elements (common language library components)
- Look for existing code that is thoroughly tested and known secure
- Think in terms of Total Cost of Ownership



http://www.rohitn.com/images/java/dilbert_codereuse.jpg

Creating Secure Code - Principles - Java

Secure By Design

- Establish Trust Boundaries
- Don't Reinvent the Wheel
- Economy of Mechanism
- Trust Reluctance
- Open Design
- Minimize the Attack Surface
- Secure the Weakest Link

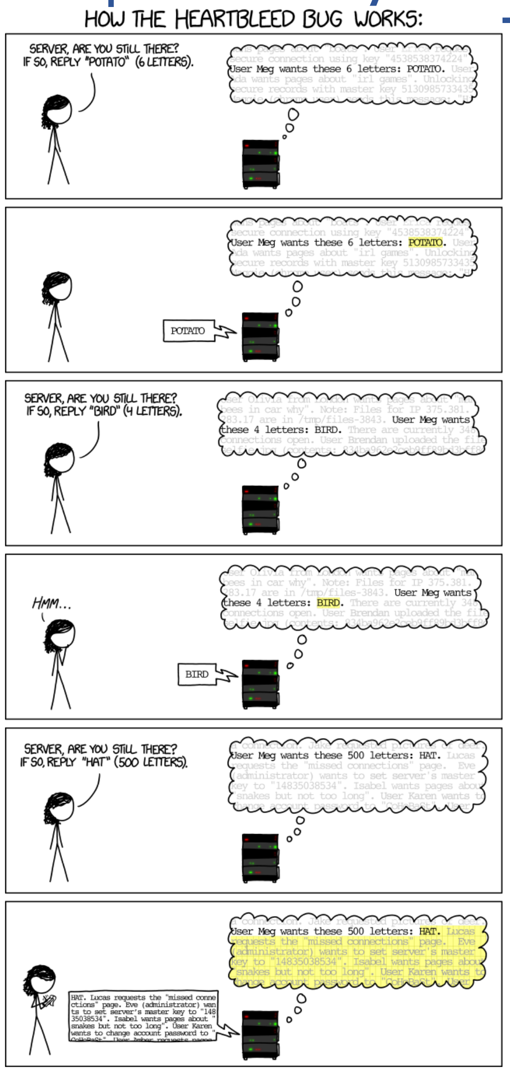
Common Secure Coding Principles
Economy of Mechanism - Introduction

- Principle: Security mechanisms should be as simple as possible
  - Corollary: All code designs should be kept as simple as possible
- The KISS adage, "Keep It Simple Stupid," applies to security
  - Complicated is the enemy of security
    - High complexity leads to more defects
    - Complicated code is more difficult to test and patch
    - Adding security means more code
  - Simple security constructs
    - Are more likely to be defect-free
    - Require less development and test time
  - Don't implement unnecessary security constructs

Common Secure Coding Principles
Economy of Mechanism - Introduction

- Complex and complicated
  - Complexity is an internal quality of a problem or a solution
  - Maybe there is a less complex solution, maybe not
- Complicated is an external quality
  - A complicated implementation of an algorithm has nothing to do with the complexity of the problem
  - Quantum mechanics is complex, but the book about it has complicated explanations
- If you have a complex problem or solution, think about ways to reduce the complexity
- If you have a complicated solution, fix it or start over; what you are doing should be obvious

Common Secure Coding Principles
Economy of Mechanism - Introduction



OR

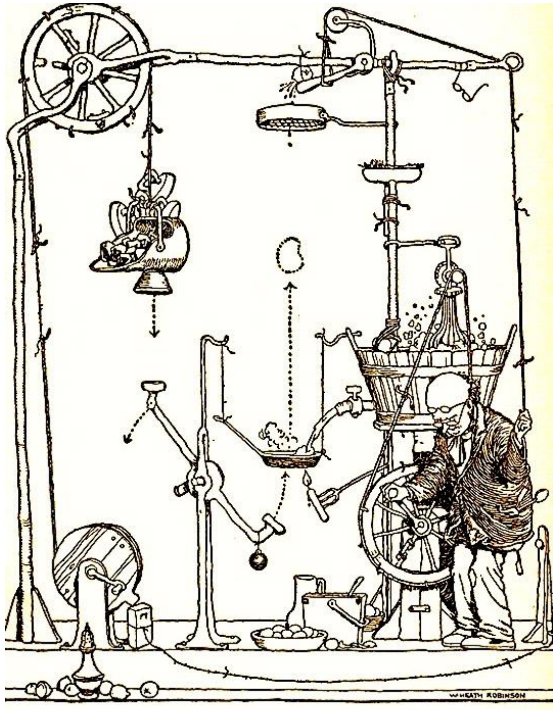
OpenSSL versions 1.0.1 through 1.0.1f had a severe memory handling bug in their implementation of the TLS Heartbeat Extension that could be used to reveal up to 64 kilobytes of the application's memory with every heartbeat.[19][20] By reading the memory of the web server, attackers could access sensitive data, including the server's private key,[21]. This could allow attackers to decode earlier eavesdropped communications if the encryption protocol used does not ensure Perfect Forward Secrecy. Knowledge of the private key could also allow an attacker to mount a man-in-the-middle attack against any future communications. The vulnerability might also reveal unencrypted parts of other users' sensitive requests and responses, including session cookies and passwords, which might allow attackers to hijack the identity of another user of the service.[22]
At its disclosure, some 17% or half a million of the Internet's secure web servers certified by trusted authorities were believed to have been vulnerable to the attack.[23]

Cartoon courtesy of:http://xkcd.com/1354/
Explantion courtesy of: http://en.wikipedia.org/wiki/Heartbleed

Common Secure Coding Principles
Economy of Mechanism – Example

- The IPSEC specification has over a dozen RFC's and has hundreds of pages
- Early implementations had many serious vulnerabilities

- An application has multiple cookies
  - One for authentication
  - One for billing for connect time
  - One for tracking state
  - One for maximum connect time
  - One for idle time
- There are numerous failures of security features due to the complexity of setting and processing the various cookies



The pancake-making machine

http://arnoldzwicky.org/2012/08/07/comic-machines/&ei=dvpLU6HdOuGSyQHJhYHoCA

Common Secure Coding Principles
Economy of Mechanism – Example

- A web application starts as a multi-tenanted system for small numbers of user groups; each group is treated as independent of the others
- Over time, the number of groups and users, and demands for new features make the system so complex that it is unreliable and there are increasing threats to confidentiality
- Solution: virtualization; give each user group their own virtual host and database
- Complexity is reduced; costs are reduced; security improves due to the isolation
- Contrarian view: SalesForce uses a multi-tenanted solution of their own design (Force)
  - Incredibly complex, but, their customer groups tend to be small, so virtualization might actually be more complex

Common Secure Coding Principles
Economy of Mechanism – Example

- A

Config/Doc example

Common Secure Coding Principles
Economy of Mechanism – Security Design Pattern

- Alias: Occam's Razor
- Some Forces:
  - There is a tendency for developers to seek clever solutions
  - Complicated code leads to more and more severe security defects
  - Complicated security mechanisms are more likely to fail
- Consequences
  - Simple security mechanisms are less likely to fail
  - Simple systems has fewer defects

How do you change the culture to one of simplicity?

Common Secure Coding Principles
Economy of Mechanism – Tips

- Evaluate security mechanisms and attempt to simplify
- Seek simple solutions to complex problems
- The later in the process that solutions to security problems are created, the more complicated they will tend to be
- Avoid complex configuration processes for security
  - Users will make mistakes or not bother to implement at all
- Avoid verbose documentation
  - Users won't read it
  - Give them checklists and step-by-step processes with references

Creating Secure Code - Principles - Java

Secure By Design

- Establish Trust Boundaries
- Don't Reinvent the Wheel
- Economy of Mechanism
- Trust Reluctance
- Open Design
- Minimize the Attack Surface
- Secure the Weakest Link

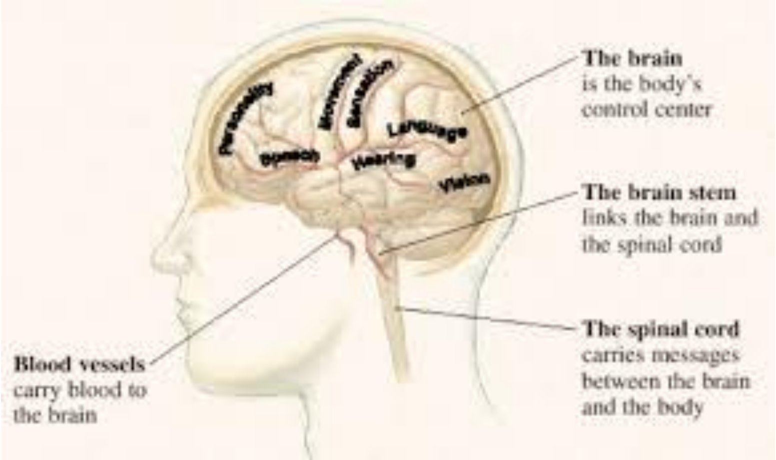
Common Secure Coding Principles
Trust Reluctance - Introduction

- Principle: Privileges should not be granted based on a single condition
  - Corollary: Requiring multiple components to agree before access can be granted to a resource is more secure than requiring only one
- A simple example is the need for two signatures in order for a check to be accepted
- The objective of this principle is to:
  - Prevent the breach of one component leading to the breach of others
    - e.g. Using Forceful Browsing to reach a file should not lead to that file being executed; the authorization of the user should be verified

Images courtesy of:http://www3.nd.edu/~dlindley/handouts/Do%2520Not%2520Trust%2520Me.JPG

Common Secure Coding Principles
Trust Reluctance - Example

- Monolithic software:
  - Is conceptually more complex
  - Has more defects
  - Is harder to maintain
  - Is more expensive to test/deploy/update/patch
- Consider a distributed system using services
  - File mgmt., Mail handling, interface, db access, …
  - Faster to create/build/deploy/patch
  - Easier to secure
  - Easier to update/patch



Images courtesy of: http://stanfordhospital.org/clinicsmedServices/COE/neuro/braintumor/images/brain-map.jpg and http://pubs.usgs.gov/gip/dynamic/graphics/monolith.gif

Common Secure Coding Principles
Trust Reluctance - Introduction

- Classic example of bad compartmentalization is the UNIX privilege model
  - Security-critical operations work on an “all or nothing” basis
  - If you have root privileges, you can do anything you want otherwise you are limited
  - For instance: you can’t bind to ports under 1024 without root access
  - In order to bind to port 25, sendmail needs to run as root
  - This has led to many exploits in the past!
- Improvements would be
  - Allow access to the low-numbered ports to be based on group so that a non-root process could open a port
- The Apache Web Server models uses port 80, but it runs server processes as an unprivileged user

Common Secure Coding Principles
Trust Reluctance - Introduction

- Another Unix example demonstrates the converse
- In order to change to the root user, you need two things:
  - To know the root password
  - And be included in the Wheel group
- Suppose you have an application API for remote management, what could you do to enforce the Trust Reluctance Principle?
-

"This site wants a two-factor authentication. A retina scan and a urine sample."

http://www.cartoonstock.com/newscartoons/cartoonists/lla/lowres/computers-website-retina_scan-urine_sample-authentication-password-llan1069l.jpg

Common Secure Coding Principles
Trust Reluctance - Introduction

- You need to connect to a third-party server and exchange data. How do you establish trust of the server?
- How do you establish trust in the data?

Image courtesy of: http://www.savagechickens.com/images/chickentrust.jpg

Common Secure Coding Principles
Trust Reluctance – Secure Design Pattern

- Alias: Trust Partitioning, Trust Distribution, Separation of Privilege
- Some Forces:
  - –A way to minimize the effect of security breach
  - –A way to increase intra and inter-application security
  - –To increase the flexibility of the application without compromising security
  - –To operate securely in unsecured environment

Image courtesy of: https://shirt.hoboninja.com/images/products/TrustMeDance_002.jpg

Common Secure Coding Principles
Trust Reluctance - Tips

- Consequences:
  - Trust Reluctance provides a deep, independent and reliable defense
  - System parts are independently secure, so they can be flexibly plug into different environments and can easily interoperate with other programs
  - More difficult to program
  - Requires a deeper insight of the system functionality

https://shirt.hoboninja.com/images/products/TrustMeDance_002.jpg

Common Secure Coding Principles
Trust Reluctance - Tips

- Trust must be created, not assumed
  - There should be clear distinctions between privilege levels when users access resources
  - If a user is authenticated, they should also have to have authorization to access a resource
    - Allowing access simply because of authentication is not secure
  - Knowing that a page exists should not be sufficient to gain access
    - Require authentication and/or authorization
  - Assume that any system not under your complete control is completely untrustworthy
- Minimize the set of components to be trusted
  - Personnel, systems, operations
  - Fewer trusted components means fewer secure systems to insure safety

Creating Secure Code - Principles - Java

Secure By Design

- Establish Trust Boundaries
- Don't Reinvent the Wheel
- Economy of Mechanism
- Trust Reluctance
- Open Design
- Minimize the Attack Surface
- Secure the Weakest Link

Common Secure Coding Principles
Open Design- Introduction

- Principle: The security of a component or system should not depend on the secrecy of the design or implementation
  - Kerckhoff's Principle: Crypto-systems should remain secure even when the attacker knows all the internal details (stated in 1883)
- Keys: the secret data that must be protected
- Also known as avoiding "Security by Obscurity"
- It is highly unlikely that any algorithm or method can be kept secret
  - Many people know
  - Attackers can guess and probe the application
  - Your own documentation may reveal the secrets

Common Secure Coding Principles
Open Design- Examples

- Passing Base64-encoded passwords in the URL (AKA Base-64 encryption)
- Hiding your house key under the mat
- Binding your admin web application to a different port
- NT LAN Manager authentication protocol was kept secret; the Samba development team reverse-engineered it and found several bugs

- Client-side hashing was based on a client-side copy of 1024 characters
- The algorithm for seeding the process was easily discovered from the client-side code

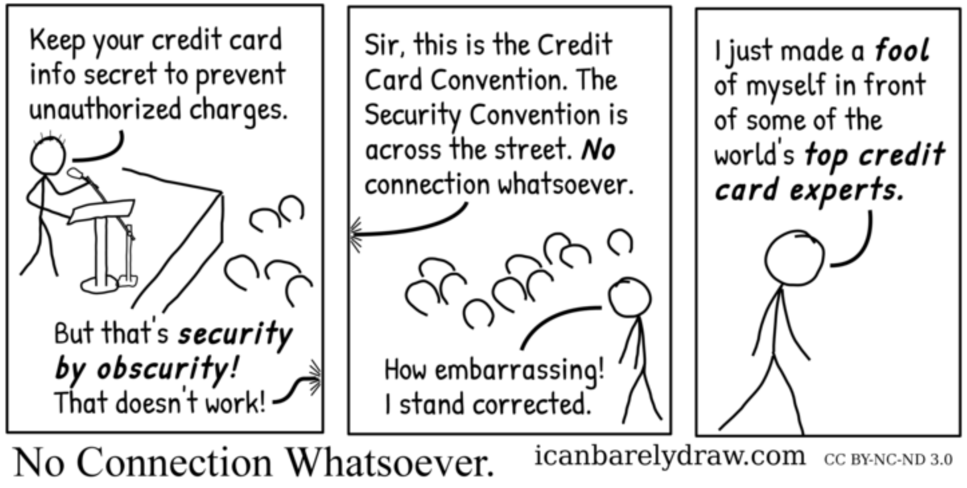


Image courtesy of: http://www.icanbarelydraw.com/comic/2039&ei=9D5NU-_fHOamsQTj_IDYAw

Common Secure Coding Principles
Open Design - Introduction

- Openness applies to algorithms and implementation details, but not to data like encryption keys and passwords (the Keys)
- The more review and testing of security code, the more secure it becomes
- The security of a system should depend on the possession of easy to protect passwords and keys, not on the ignorance of the attacker

- This includes processes
  - The existence of backdoors
  - Default accounts
  - Deployment procedures
  - Backup procedures

Common Secure Coding Principles
Open Design - Example

- Several major security systems were made public in advance of release to insure that they were technically sound
  - RSA's RC4 algorithm was made public ahead of time
  - PGP was released as source code
- The GSM proprietary A5/1 strong crypto-suite was kept secret (1989)
  - It became known through leaks
  - It was cracked and completely reverse engineered by 1994

Common Secure Coding Principles
Open Design – In Defense of Obscurity

- You should not depend on obscurity for protection
- But using obscurity as camouflage is fine
  - Port knocking
  - Changing the name of the Admin user
  - Changing cgi-bin to demos
  - Changing the name of system administration executables
- But making it more difficult, is never the wrong thing to do
- The Cullinan Diamond (3106 carets) was mailed from South Africa to London by regular post, and then carried to Amsterdam in the pocket of a diamond cutter(1905)

Image courtesty of: http://ngm.nationalgeographic.com/wallpaper/img/2010/08/aug10wallpaper-11_1600.jpg

Common Secure Coding Principles
Open Design – In Defense of Obscurity

- A similar strategy is called

Security Through Minority/Rarity/Obsolesence/Unpopularity/Lack of Interest

  - Use methods that are used seldom or are out-of-date, depending on the general lack of knowledge and interest for defense
  - Of limited use in products that attract any interest from potential attackers
- Examples
  - Using an out-of-date and vulnerable PHP version
  - Using a version of SSL vulnerable to the Heartbleed attack
  - Using Microsoft CryptGenRandom based random number generator

Image courtesty of: http://ngm.nationalgeographic.com/wallpaper/img/2010/08/aug10wallpaper-11_1600.jpg

Common Secure Coding Principles
Open Design – Secure Design Pattern

- Alias: None
- Some Forces:
  - Pressures for release deadlines may encourage insecure solutions
  - Fear of publically releasing algorithms or code
  - The need to protect intellectual property
  - An "I don't have to be secure, I just have to be more secure than the other guy" mentality
- Consequences
  - Code gets more testing and scrutiny
  - Developers don't depend on obscurity
  - There is less trepidation over the possible release of informati

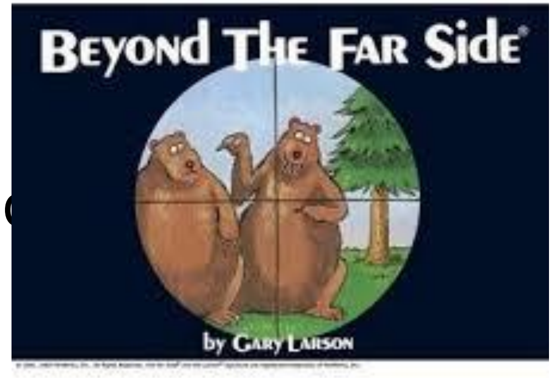


Image courtesy of: http://www.gjsentinel.com/blogs/images/far-side%25202bears(1).jpg

Common Secure Coding Principles
Open Design - Tips

- Design security features as though only the keys are private
  - The point of open design is that secrets usually are not secret
  - If your software has proprietary algorithms, processes and procedures, assume they are public when you design
  - Without proprietary algorithms, you need to be especially careful with keys
- Areas where data is commonly assumed to be safe when it is not
  - Registry keys
  - Hard-coded passwords or encryption keys
  - Data or code in HTML pages
  - Anything stored on a client host
  - Anything sent over an unencrypted communication channel

Common Secure Coding Principles
Open Design - Tips

- Obscure or customized security algorithms are seldom reliable which increases risk
    - Encryption
    - Randomization
    - Session management
- Any security mechanism that depends on no one noticing is doomed to failure
- Keys are safe only if you make them so
  - Inside of your Trust Boundaries
  - Security features designed to protect them even if everyone knows how they work

Creating Secure Code - Principles - Java

Secure By Design

- Establish Trust Boundaries
- Don't Reinvent the Wheel
- Economy of Mechanism
- Trust Reluctance
- Open Design
- Minimize the Attack Surface
- Secure the Weakest Link

Common Secure Coding Principles
Minimize the Attack Surface - Introduction

- Principle: The smaller the attack surface, the safer the application
- The risk to an application is:
  - Size of the Attack Surface x The Probability of a Vulnerability
  - The probability of a vulnerability is not going to be zero
- Attack surface reduction (ASR) requires an acceptance of the idea that code is never completely secure
  - Design or implementation failures
  - Poor configurations
  - Failure to install patches or upgrades
  - New attack types
  - Mistakes

Common Secure Coding Principles
Minimize the Attack Surface – Secure Design Pattern

- Alias: None
- Some Forces:
  - Attack surfaces tend to grow unless deliberately reduced
  - There is considerable pressure to add new entry points in the form of
    - New features and API's
    - Greater connectivity
  - Reducing the attack surface existing code may require significant resource committment
- Consequences
  - Decreased attack surface size decreases the opportunity for attackers
  - Decreased attack surface size simplifies the overall security problem

Common Secure Coding Principles
Minimize the Attack Surface - Tips

- Attack Surface Reduction focuses on
  - Reduce the amount of running code
    - By default, only necessary code should be running
    - This is greatly enhanced by modularizing and isolating software components
    - Consider a distributed architecture
    - Limit access to code when not universally required
  - Minimize open ports and services
    - Only allow services to run that are necessary
    - Close all unnecessary ports
    - Obscure or often unused protocols are often vulnerable
      - finger, whois, Webster,

Common Secure Coding Principles
Minimize the Attack Surface - Tips

- Reduce code paths for anonymous users
  - Anonymous code paths have far larger attack vector sets because anyone can access the paths
  - If anonymous paths are allowed, they should be restricted access and short
  - Anonymous paths are operate in the same code as restricted paths, so the opportunity for privilege escalation is significant
- Reduce the number of entry points
  - Entry points are code locations where untrusted components can access the system
  - Each entry point requires protection (more code)
  - Each entry point is a focus point for an attacker
  - Entry points can be reduces by limiting access to privileged users

Common Secure Coding Principles
Minimize the Attack Surface - Tips

- Attack Points
  - UI Forms and fields
  - HTTP headers
  - Cookies
  - API's and API functions
  - Login/authentication's
  - Interfaces with other systems
  - Database interfaces
  - Admin interfaces

Common Secure Coding Principles
Minimize the Attack Surface - Tips

- The Attack Surface can be modeled with a scanner
  - ZAP Proxy, Burp Proxy, w3af, Arachni
- Prioritize the risk of each attack point
  - Network facing entry points
  - Client-side data and web forms
  - External files
  - Backward compatible interfaces with old code or interfaces
  - Custom API's
  - Security code: encryption, authentication, authorization, session data
- Attempt to reduce the risk to the application by reducing the attack points and the risk at each point

Creating Secure Code - Principles - Java

Secure By Design

- Establish Trust Boundaries
- Don't Reinvent the Wheel
- Economy of Mechanism
- Trust Reluctance
- Open Design
- Minimize the Attack Surface
- Secure the Weakest Link

Common Secure Coding Principles
Secure the Weakest Link - Introduction

- Principle: Attackers will attack the weakest security point in the application
  - Corollary: One valid secure coding outcome is encouraging the attacker to go after someone else
  - Corollary: A chain is as strong as its weakest link

- Adversaries will expend the least amount of effort possible to penetrate a system
  - They will work no harder than necessary
  - If they have to work too hard, they may move on to another target (depending on how attractive the asset is)

Image courtesy of: http://www.businessearth.com/wp-content/uploads/supply_chain_sustainability.jpg

Common Secure Coding Principles
Secure the Weakest Link - Introduction

- Threat Models will lead you to the weakest areas
  - Invest in remediating weakest security defenses
  - There will always be a new weakest link
  - Attackers will go after weakest links simply because they are easy
    - They are looking for a foothold or information to use
- Factor in the four sources of threats
  - Social: People
  - Operational: Processes (and People)
  - Technological: The application and network
  - Environmental: Facilities
- Which threat sources are more likely to be weak?

Common Secure Coding Principles
Secure the Weakest Link - Introduction

- An attacker will look for weaknesses
  - Default settings on the application or servers
  - Backdoors
  - Test servers
  - A wireless network
  - That old backup server no one uses
  - People who are negligent or ill-informed
  - Open ports in the firewall or server

Common Secure Coding Principles
Secure the Weakest Link - Introduction

- HP Printers using Jetdirect firmware include an embedded web server
  - Allows for remote administration of the device
  - Due to an undisclosed design flaw, the server handles passwords in an insecure manner
  - Attacker can gain unauthorized access to the device and also create a denial of service.
- Networked printers, in general, are poorly secured.

Common Secure Coding Principles
Secure the Weakest Link – Security Design Pattern

- Alias: Low Hanging Fruit, Quick Wins
- Some Forces:
  - Development timelines my prohibit a fully secure design
  - The skills required to properly secure applications might not be immediately available
  - An adequate testing environment for new tools and procedures may not be available
  - Inadequate training my lead to weaknesses to operational or social threats

Common Secure Coding Principles
Secure the Weakest Link – Security Design Pattern

- Consequences:
  - Effort is focused where it will do the most good
  - The application, servers and system begin operation with an acceptable, minimum level of protection
  - Applications are not left exposed to trivial attacks and vulnerabilities
  - Basic troubleshooting and auditing trails are enabled

Common Secure Coding Principles
Secure the Weakest Link - Tips

- Use Threat Models to understand your attack surface
- Make sure you don't ignore operational and social threats
- Test your application thoroughly
  - Have a test plan
  - Use Red Teaming and/or third-party audits
  - Test for all of the obvious vulnerabilities
    - SQL Injection
    - XSS
    - Authentication Bypass

Creating Secure Code - Principles - Java

Secure By Default

- Use Least Privilege
- Use Default Deny
- Fail Securely

Secure Coding Principles
Use Least Privilege- Introduction

- Principle: A subject should only be granted only the privileges needed for an operation
  - Corollary: Privileges should be associated with the function being performed, not with the identity
- Least Privilege is a concept that means that at any given application state, the user will operate at the lowest level of access rights possible
- A program should be given only those privileges it needs in order to satisfy its requirements—no more, no less
  - If a program doesn't need an access right, it should not be granted that right
  - Think of it as "need to know" rule
  - Thus if the program is compromised, damage is limited

Secure Coding Principles
Use Least Privilege- Example

- For example,
  - When accessing the database on behalf of an unprivileged user, the database account used will have only the privileges necessary
    - Access to only the required databases and tables
    - Access to only the necessary operational controls
      - CRUD – Create, Read, Update, Delete
  - It is not uncommon for developers to use a single account that works for all user privilege levels
  - In the case of an SQL Injection vulnerability, the attacker could only perform those operations allowed
  - For normal users, only read is likely to be needed

Secure Coding Principles
Use Least Privilege- Example

- Do you run your local desktop as an Administrator user?
  - You are not applying the Least Privilege Principle
  - The damage from executing malicious code will be much greater than if you didn't
- On *nx, running services as root or with setuid permission bit set has the same effect
  - Running the Apache web server as root
  - That gives every executable that Apache runs root privileges on your system
  - If a program runs as root, many of it's normal defects become security defects

Secure Coding Principles
Use Least Privilege- Example

- MS-ISAC 2013-112, Remote execution vulnerability in Microsoft Scripting Runtime Library
  - Allows the remote attacker to run as the logged in user
  - If you are using this library and running as a reduced privilege user the damages will be limited
  - If you are running as Administrator, the damages could be much greater

Secure Coding Principles
Use Least Privilege – Security Design Pattern

- Alias: Principle of Least Privilege (POLA), Minimal privilege
- Some Forces:
  - Most users do not need absolute privilege on their machine
  - Maximum privilege is something the attacker desires
  - It is easy to work as a non-admin user on most systems
  - Developers still code applications to require admin access even when it can be avoided
- Consequences:
  - In case of a breach, damage is limited
  - Cause of breach can be analyzed and fixed
  - Greatly improved overall security

Secure Coding Principles
Use Least Privilege – Tips

- Don’t share code between privilege levels
  - It is difficult to do this well
  - A small error can have serious consequences
- Check privilege level at every entry point, even if you don’t think it’s an entry point
  - If a user can get there by any means, its an entry point
- Laziness/In-a-hurry are the enemies of this principle

Creating Secure Code - Principles - Java

Secure By Default

- Use Least Privilege
- Use Default Deny
- Fail Securely

Secure Coding Principles
Use Default Deny- Introduction

- Principle: Anything not specifically allowed must be denied
  - Corollary: Default configurations should be the most secure setting
- Assume that access is denied, and find conditions that allow access
  - The converse, finding reasons to deny access leads to errors
  - If there is a defect in your system, you will deny access, not grant it
- The access control system should default to no access
  - Unless it is specifically allowed
- In the event of an operation failure, do nothing
  - And restore the system to the state prior to attempting the operation

Secure Coding Principles
Use Default Deny- Introduction

- If you attempt to open a file and the operation fails, report it as an error, don't try to find a different location
- If an attacker uses Forceful Browsing to access a file that controls access to administrative resources
  - Access to the resources should be denied
  - Unless the attacker can provide credentials

Secure Coding Principles
Use Default Deny – Security Design Pattern

- Alias: Fail-Safe Defaults
- Some Forces:
  - Assumptions often lead developers to create defaults that are too liberal
  - User friendliness can lead to errors in judgement with regard to security
- Consequences:
  - Other defects won't result in breaches if the default is to deny access
  - By assuming that access won't be allowed, attention is focused on the criteria for granting access
  - Users often don't change default configurations, or even look at them

Secure Coding Principles
Use Default Deny – Tips

- Trust Boundaries can be used to motivate this effort
  - Consider what criteria are required to move across a Trust Boundary
- All configuration values that define resource access should be designed
  - There should be one value for no access
  - And that should be the default unless changed
  - Users should be aware of the consequences of granting higher levels of access

Creating Secure Code - Principles - Java

Secure By Default

- Use Least Privilege
- Use Default Deny
- Fail Securely

Secure Coding Principles
Fail Securely - Introduction

- Principle: Handle all failures securely and return the system to a proper state

Secure Coding Principles
Fail Securely - Introduction

- If you attempt to open a file and the operation fails, report it as an error, don't try to find a different location
- If an attacker uses Forceful Browsing to access a file that controls access to administrative resources
  - Access to the resources should be denied
  - Unless the attacker can provide credentials

Secure Coding Principles
Fail Securely - Security Design Pattern

- Alias: Fail Safely
- Some Forces:
  - Error messages can disclose information valuable to an attacker
  - Failure can lead to an unhandled state, which can lead to denial of service
  - Unhandled failures can lead to malicious behavior being unnoticed
- Consequences:
  - All error conditions are handled and logged
  - There are no verbose error messages
  - The failure of a component doesn't result in systemic failure
  - A failure does not result in a secure data breach

Secure Coding Principles
Fail Securely - Tips

- Don't assume that an error condition won't occur
  - It's what the attackers want you to assume
  - Errors are like accidents, you don't expect them, but they happen
  - Any code that can throw an exception should be in a Try Block
  - Handle all possible exceptions
  - Use Finally Blocks: leaving files open or exceptions defined after use creates resource leaks and possible system failure
  - Short specific Try Blocks give you more control over the error state

Creating Secure Code - Principles - Java

Secure By Communication

Secure Trust Relationships

Common Secure Coding Principles
Secure Trust Relationships - Introduction

- Principle: Trust in outside or Third Party entities must be established and maintained
  - Corollary: Nothing outside of your Trust Boundaries is safe
  - Corollary: You can't trust others to be secure
  - Corollary: Even the most secure communication has weaknesses
- Most systems of any value will have to do one or more of these
  - Exchange data with a client system
  - Exchange data with a Third Party system

Common Secure Coding Principles
Secure Trust Relationships - Introduction

- The issues in establishing trust are:
  - Authenticating the other endpoint
    - To prevent masquerading
  - Ensuring the security of the communication itself
    - To maintain the confidentiality of the data
  - Preventing tampering to data
    - To maintain the integrity of the data
- Once data enters the Trust Boundary, you must be as certain of its veracity as your application requires
  - Is it acceptable for it to come from an anonymous user?
  - Must it be private?
  - Must it be accurate?

Common Secure Coding Principles
Secure Trust Relationships - Introduction

- Example: You need to exchange information between a web client and your server
  - Authentication is likely a session token created when the client user logs in
  - Secure communication can be performed with SSL/TLS or SSH
  - Anti-tampering can be done by adding checksums to the messages
  - None of these are completely secure if the desktop itself is not secure
    - Proxies and sniffing tools will allow an attacker access to the data between the client and the network
    - The attacker could be a legitimate client user
    - Remember, there are limits to how secure you can make communication through an untrustworthy network, but you can do quite a bit

Common Secure Coding Principles
Secure Trust Relationships - Introduction

- Example: Your system needs to send sensitive data to a third party for processing and then returned
  - Assuming you have done everything possible to guarantee the trustworthiness of the third party
  - You must authenticate that you are sending to the correct endpoint
    - A shared secret
    - Public Key Certificates issues by a trusted Certificate Authority
    - Kerberos, NTLM or integrated authentication
  - Secure communication can be performed using SSH and the certificates
  - Anti-tampering can again be used, possibly over an entire file if that is the send unit

Common Secure Coding Principles
Secure Trust Relationships - Introduction

- Example: You must accept data from a system that only supports FTP, not any secure protocols
  - This data is not terribly sensitive, or this would not be allowed
  - Your primary concern is protecting your system from contamination
  - Unless your standards are low, don't allow the data inside your Trust Boundary
    - Collect the data on an external server (possibly in a DMZ) and process the input data for correctness before importing
    - Authentication may be as little as the FTP account and password which is not terribly secure
  - If possible, require some form of anti-tampering
  - Authentication may be as little as the FTP account and password which is not terribly secure
    - The IP Address can be a form of identification, but it can be spoofed

Common Secure Coding Principles
Secure Trust Relationships - Introduction

- Example: You allow remote access for system management
  - Authentication is critically important; consider two factor authentication
    - Using SMS codes
    - Using hardware authenticators: disconnected or connected tokens
    - Biometrics
    - Use combination of what the user knows, has or inherits
  - All communications tunneled through SSH

Secure Coding Principles
Secure Trust Relationships - Security Design Pattern

- Alias: None
- Some Forces:
  - Nothing outside of the trusted system is trustworthy
  - Even if you trust the other endpoint, you can't trust the communication system or other parties
  - You can't know if the other endpoint is truly secure
  - You need to connect to the outside world
- Consequences:
  - Confidentiality and integrity are maintained in external communications
  - The application is protected from attacks originating outside the Trust Boundary

Secure Coding Principles
Secure Trust Relationships - Tips

- Authentication
  - Use two-factor authentication when required
  - Encrypt all client-side session data; cookies, session tokens
  - Encrypt all authentication credentials in-transit
  - Do not leave authentication information in client memory unless it is encrypted
  - Do not put hard-coded passwords, encryption keys or vital information on the client host
- Communications must be encrypted to defend confidentiality
  - HTTPS uses SSL/TLS for endpoint authentication and encryption
  - Configure SSL/TLS to exclude weak cryptography
  - Use SSH to create an encrypted tunnel for non-HTTP traffic

Secure Coding Principles
Secure Trust Relationships - Tips

- Anti-tampering methods include
  - Hardware-based devices that encrypt/decrypt the message stream
  - Encryption wrapping tunnels all data through an encryption/decryption process
    - There are serious performance limitations with web clients
  - Add tamper-detection checksums to each message

Creating Secure Code - Principles - Java

Secure By Implementation

- Psychological Acceptability
- Least Common Mechanism
- Validate Inputs
- Prevent Bypass Attacks
- Secure Data at Rest
- Audit and Verify
- Defend in Depth

Secure Coding Principles
Psychological Acceptability - Introduction

- Principle: Security mechanisms should not make the resource more difficult to access than if the security mechanism were not present
  - Corollary: In many cases, you will only get a close approximation
- Security and user-friendliness are often contradictory
- Security mechanisms should seek to be as unobtrusive as possible while still meeting security goals

Secure Coding Principles
Psychological Acceptability - Example

- If the configuration for securing a security mechanism is difficult or confusing, users may choose to not enable it
  - In the end, that is not good for anyone
  - Example, a product with five different settings for cookie expiration times that interact
    - Complicated because the development team didn't want to change a outdated method of charging for usage
    - Eventually became a fixed at deployment and locked
- When cookies are turned-off, an application resorts to session ids passed in the URL
  - Just require cookies to be on

Secure Coding Principles
Psychological Acceptability - Example

- Your company mandates two-factor login for google mail
  - Is having to get an authentication code from your cell phone when you login psychologically acceptable?
  - If you only have to do so every 30 days for any given host?
- A college campus decides that usernames should be less "guessable" and replaces them with 8 random characters
  - Psychologically acceptable?
- A system requires that all personal files be passworded, so each access requires a password to be typed
  - Psychologically acceptable?

Secure Coding Principles
Psychological Acceptability - Security Design Pattern

- Alias: None
- Some Forces:
  - Security is going to reduce convenience
  - Security mechanisms that are unusually complex or difficult to use will be evaded or create complaints
  - Security mechanisms can often be disabled
- Consequences:
  - Well-designed security mechanisms
  - Security features are used

Secure Coding Principles
Psychological Acceptability - Tips

- Avoid situations where code is shared between different privilege levels
- This is particularly true where access control for resources is required
  - File systems
  - Databases
  - Underlying operating systems
  - Communication systems

Creating Secure Code - Principles - Java

Secure By Implementation

- Psychological Acceptability
- Least Common Mechanism
- Validate Inputs
- Prevent Bypass Attacks
- Secure Data at Rest
- Audit and Verify
- Defend in Depth

Secure Coding Principles
Least Common Mechanism - Introduction

- Principle: Mechanisms to access resources should not be shared between privilege levels
- The code has to be unnecessarily complex to be secure, and therefore, more likely to have a defect
- Some form of state will have to be shared to control the path of execution, and it will likely be exposed to an attacker
- A small error can result in a significant breach

Secure Coding Principles
Least Common Mechanism - Example

- Both regular users and administrative users need to access files
  - There is a temptation to write the code once and share the code
  - The choice is then to allow all users access as administrator, or to add code to distinguish between the two
  - If the mechanism for deciding the allowed privileges is part of the client request, then it is subject to tampering
    - This could be a very serious vulnerability, so your protection may have to be perfect
  - An access control decision has to made in the software at each operation

Secure Coding Principles
Least Common Mechanism - Introduction

- This can all be avoided by creating two copies of the code
  - Each works for different privilege levels
  - There is no exposure to client-side tampering
  - You still need to authenticate access to pages, but not operations
- You can often avoid these situations by moving some code into a common core, and using a wrapper for each client-facing case

Secure Coding Principles
Least Common Mechanism - Security Design Pattern

- Alias: Fail Safely
- Some Forces:
  - There is a tendency among developers to seek "clever" solutions
  - Security is often weakened by small mistakes
  - Sharing of code raises the risk and the complexity of code
- Consequences:
  - Cleaner, less complex code
  - Simpler interfaces
  - Fewer opportunities for tampering attacks
  - More robust security

Secure Coding Principles
Least Common Mechanism - Tips

- Avoid situations where code is shared between different privilege levels
- This is particularly true where access control for resources is required
  - File systems
  - Databases
  - Underlying operating systems
  - Communication systems

Creating Secure Code - Principles - Java

Secure By Implementation

- Psychological Acceptability
- Least Common Mechanism
- Validate Inputs
- Secure Data at Rest
- Prevent Bypass Attacks
- Audit and Verify
- Defend in Depth

Secure Coding Principles
Validate Inputs - Introduction

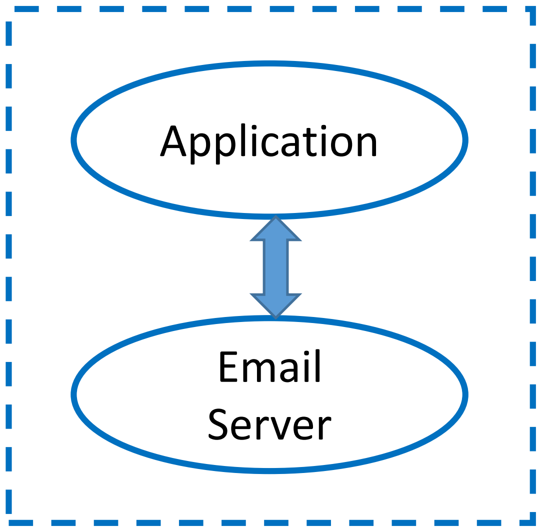
- Principle: All inputs should be treated as untrustworthy
  - Corollary: Anything coming from outside of your application may not agree with your expected input requirements
  - Corollary: Good developers validate their inputs regardless of security considerations
- Inputs that don't meet expectations can lead to program states that are undefined
  - Undefined states are problematical for security because there is no sure way to predict how the application will respond

Secure Coding Principles
Validate Inputs - Introduction

- There are many vulnerabilities that are created or worsened by using unvalidated inputs in code constructs
  - All of the injection vulnerabilities (SQLi, XSS, HTTP Response Splitting, Command Injection, etc)
  - Open Redirect
  - Buffer Overflow, Uncontrolled Format String, …
  - Parameter Tampering
  - …
- In addition, if you allow inputs that are incorrect for any reason into your system, you risk its integrity

Secure Coding Principles
Validate Inputs - Example

- An application is accepting input from an email server that is running on another system inside your Trust Boundary
  - What could happen?
    - An insider with access to the email server uses that platform to send persistent XSS data into the application
    - The format for the incoming mail on the application changes, but the patches to the email server don't get done, and the incoming data crashes the application
    - The email server happens to have an open backdoor vulnerability that is exploited; from there, the attacker can misappropriate the service that accepts email input to perform command injection



Trust Boundary

Secure Coding Principles
Validate Inputs - Introduction

- This

```
if(!isset($cert)) $cert=$_GET['cert'];      // $cert contains unfiltered user data.
...
...
function x509_open($cert) {
       global $cert_in_dir, $openssl;
       $lines = array();
       exec("$openssl x509 -in $cert_in_dir$cert -subject -issuer   -dates
                                                   -serial -fingerprint -noout ...
```

Here the $cert variable can be crafted to contain a piped UNIX command which will be executed on the server.

The solution is to pass this variable to a sanitization method before using it in the code

Secure Coding Principles
Validate Inputs - Security Design Pattern

- Alias: Data Validation
- Some Forces:
  - Input validation is time-consuming and difficult to maintain
  - Attackers often turn to input as the primary means of attacking or gaining information about an application
- Consequences:
  - The application functions flawlessly in the presence of incorrect or malicious inputs
  - Accidental input mistakes are discovered and properly handled
  - Attackers find it more difficult to attack the application

Secure Coding Principles
Validate Inputs - Tips

- Always test the big three:
  - Correct type (null/non-null, text/integer/float, scalar/array/object, …)
  - Proper length (minimum-to-maximum)
  - Acceptable content
    - Use Whitelisting if possible
    - Use regular expressions to test; they are the best descriptions
- Remember that Trust Boundaries are not smooth
  - An application needs to test all input, regardless of the origin
  - Input from supposedly trusted systems is subject to errors, corruption and manipulation

Secure Coding Principles
Validate Inputs - Tips

- Modularized input validation leads to:
  - Better and more robust code
  - Consistent application; any mistake can lead to an exploit
  - Easier updating and patching
  - A clear view of the purpose of the code
- Client-side validation is not effective
  - Common tools, like web proxies can be used to intercept messages after client validation and modify the values
  - All validation must be done on the server
  - Validation performed on the client for user-friendliness must be repeated

Creating Secure Code - Principles - Java

Secure By Implementation

- Psychological Acceptability
- Least Common Mechanism
- Validate Inputs
- Secure Data at Rest
- Prevent Bypass Attacks
- Audit and Verify
- Defend in Depth

Secure Coding Principles
Secure Data at Rest - Introduction

- Principle: Data at rest must be protected to meet security requirements
- Data stored as part of an application has needs for:
  - Security, which determines if access granted or not granted for a data set
  - Privacy, determines what the access means to a given user; what form is the data in what operations are allowed
- Example: physicians in a Los Angles hospital have full access to patient records; but not to extract information of prurient interest
- As you can see, this can be a difficult problem

Secure Coding Principles
Secure Data at Rest - Example

- An application has information that can be divided into give different types of access: admin, agent, support, user, guest
  - Each has access to some tables/fields and not others (except admin)
  - How can that be implemented
- Solution:
  - Create a role for each type, and either create a matrix of roles and resources with permissions for Create, Read, Update and Delete for each.
  - It is likely that opinion on the roles will change, so make your system configurable
  - Obviously, the Administrator will have rights over the role privileges

Secure Coding Principles
Secure Data at Rest - Example

Resource	Admin	Agent	Support	User	Guest
User Accounts	CRUD	RU	R	R	-
User Permission	CRUD	-	-	-	-
Sales Data	CRUD	RU	RU	R	-
Forum Data	CRUD	CRU	RU	CRU	R
Products	CRUD	CRU	RU	CRU	R
Orders	CRUD	CRUD	CRU	CRU	CR

- This table can be used to determine to privileges for each user and operation

Secure Coding Principles
Secure Data at Rest - Example

- An application manages data for an online sales organization
  - Sales consultants need to access the personal information of the customer on the phone or chat including credit card information to complete purchases
  - But, those consultants should not be able to see credit card numbers
- Solution:
  - The sales consultants have access to the payment card data
  - But they see the credit card masked: **** **** **** 9999
  - They can press a button to make the purchase, but not see the PCN
  - Since CV codes cannot be stored, they will have to ask the customer
  - The server logs operations to create an audit trail; it also cannot contain the PCN

Secure Coding Principles
Secure Data at Rest - Example

- A thick client handles authentication for an application by fetching the user name and password on the first attempt
  - If additional attempts are required, they can be handled without communicating with the server
  - The problem is that the username and password are stored on the client
  - They are vulnerable to client-side attacks
    - Memory dumps
    - Dynamic debuggers
- Solution:
  - Encrypt the credentials while stored in memory
  - When authentication is complete, overwrite all memory locations used
  - The encryption keys also have to be protected

Secure Coding Principles
Secure Data at Rest - Example

- A thick client caches database information locally to improve performance
  - The client user has access to everything if they operate with Administrative privileges
  - So the data in the files is unprotected unless it us encrypted
- Solution
  - Send the data that requires privacy in an encrypted form
  - Use a database that can be controlled by the application
  - If decrypted locally, overwrite in memory immediately

Secure Coding Principles
Secure Data at Rest - Example

- An Oracle database can encrypt data column-wise
  - The key for the encryption must be stored somewhere
  - So Oracle stored those keys in the database; in the clear
  - That is not terribly secure, so for $, they would sell you the TDE Oracle Wallet
  - Which would store another key used to encrypt the keys in the DB
  - But the keys in the wallet were encrypted with another key which was stored in the clear in memory
  - CVE-2006-0270
- Solution
  - Public key encryption could be used to end the endless encryption

Secure Coding Principles
Secure Data at Rest - Security Design Pattern

- Alias: None
- Some Forces:
  - Data must be stored and accessed to make an application valuable
  - There many variations of security and privacy that affect data
  - Data at rest is often at risk simply by existing
- Consequences:
  - Data in memory will be protected from malicious access
  - Data in files or the database will be protected from under-privileged access
  - Data with privacy restrictions will be protected from under-privileged viewing

Secure Coding Principles
Secure Data at Rest - Tips

- Create a security system that can define that access controls for every operation and every role
- There are three kinds of data
  - Data that is private to everyone except the owner
    - User passwords
    - Cookie contents
    - Encryption keys
    - Critical data like PCN, SSN or as defined by the requirements
  - This data should be encrypted at rest
  - This data should be masked if involved in any operation where it might be seen by anyone other than the owner (and possibly even then)
  - On a client system, this data must not be allowed to persist in an unencrypted form

Secure Coding Principles
Secure Data at Rest - Tips

- Data that is private to certain groups
  - This data should be encrypted if there is any danger that the a non-privileged entity might see it
  - This data does not typically have to be masked
- Data that is not private
  - There is no need to encrypt or mask this data

- When masking data, it should be done when it leaves the database, not when it is viewed
  - You may be very careful, but the risk is very high
- Client-side data is always in danger
  - Encrypt it in files, databases or in memory
  - All decrypted copies should be overwritten immediately after use

Creating Secure Code - Principles - Java

Secure By Implementation

- Psychological Acceptability
- Least Common Mechanism
- Validate Inputs
- Prevent Bypass Attacks
- Audit and Verify
- Defend in Depth

Secure Coding Principles
Avoid Bypass Attacks - Introduction

- Principle: Attacks that bypass authentication or authorization gates are among the most dangerous
  - Corollary: Bypass attacks are favorites among the evil-doers
- In a bypass attack, an attacker:
  - Is able to act as an authenticated user without providing valid credentials
  - Is able to access resources by evading the authorization checks
- Many vulnerabilities can result in a bypass, but the focus here is on the authentication and authorization system

Secure Coding Principles
Avoid Bypass Attacks - Introduction

- Authentication bypass can occur by:
  - Forceful browsing to parts of the application that fail to authenticate
  - Stealing or forging session tokens
  - Stealing/guessing credentials
  - Masquerading
- Authorization bypass can occur by:
  - An authentication bypass
  - Forceful browsing to parts of the application that fail check privileges
  - Forging credentials for the authorization system

Secure Coding Principles
Avoid Bypass Attacks - Introduction

- Credentials are almost always stolen from the client or the exchange between the client and server
  - Cookies can be co-opted by attacks like Cross-site Request Forgery
  - Cookies can be stolen or forged if the attacker has access to a system on which a valid user has authenticated
  - Session or authorization data stored in the URL is subject to being cached and then viewed at some later time
  - Poor messaging might allow the attacker to guess credentials
  - A poorly designed forgotten system could allow attackers to breach user accounts
  - Weak passwords are easy to guess
  - Weak cryptography can allow credentials to be stolen

Secure Coding Principles
Avoid Bypass Attacks - Security Design Pattern

- Alias: Complete Mediation
- Some Forces:
  - Authentication is an obvious attack vector
  - Clients must maintain some form of application state
  - Developers may not protect all pages
- Consequences:
  - Authentication and authorization bypass threats are remediated
  - A more reliable session management system
  - Forceful browsing remediated

Secure Coding Principles
Avoid Bypass Attacks - Tips

- Authentication cookies
  - Must have their contents encrypted
  - Should contain information that locates it (browser info, IP address)
  - Must expire in a reasonable time (minutes-to-hours)
- Session data should not be in the URL and should be encrypted if it includes information useful to the attacker
- Direct high-value transactions authenticated with client-side data should not be trusted
  - Example: http://mybank.com/transfer?from=...&to=...&amt=...
  - Don't execute the transaction directly unless preceded with the proper page request within a short time length

Secure Coding Principles
Avoid Bypass Attacks - Tips

- Using IP Addresses for authentication is insufficient
- For highly critical systems, use two-factor authentication
- For stateless systems, re-authenticate and re-authorize on every access
  - This done via the session data, but it must be done on every request
- The login/password/forgotten_password system must be designed carefully
  - Prevent information leakage
  - Strong configurable passwords
  - Strong encryption
  - No bypass opportunities

Creating Secure Code - Principles - Java

Secure By Implementation

- Psychological Acceptability
- Least Common Mechanism
- Validate Inputs
- Prevent Bypass Attacks
- Audit and Verify
- Defend in Depth

Secure Coding Principles
Audit and Verify - Introduction

- Principle: A system can't be secure if its operation can't be audited and verified
  - There must be an audit trail for all operations that have a security component
    - Login/Logout/Password change
    - Data read/write/update/delete
    - Administrative operations, user add/delete, change in access controls
    - Resource access
    - Network connect/disconnect
    - Errors/Failures
  - The audit trail must extend to into the past as long as required
    - Contractual obligations to customers, compliance organizations
    - Legal obligations

Secure Coding Principles
Audit and Verify - Introduction

- Security incident investigations require information to verify the extent of damages
  - When the attack started/ended
  - What the attacker accomplished
  - What areas of the product have been exploited
- Compliance certifications will request logs and audit reports
  - To verify that no sensitive data is being exposed
  - That sufficient information is logged to meet security needs

Secure Coding Principles
Audit and Verify - Security Design Pattern

- Alias: none
- Some Forces:
  - Without an audit trail, most security attacks can be repudiated
  - Logs are typically an after-thought, designed primarily for developers
- Consequences:
  - Logs are designed to meet security needs
  - Logs are protected throughout the product lifecycle
  - Audit trail reports are available to meet security and compliance needs

Secure Coding Principles
Audit and Verify - Tips

- Logs
  - Should be treated as critical data
  - Must not contain sensitive data
    - Passwords, encryption keys, payment card information
    - It is possible to allow this data if properly masked
  - Should not be kept on a server that allows user access
  - Must be encrypted if moved outside of a Trust Boundary
  - Must be in locked storage if stored on movable media
- Audit reports should be available for easy access in the event of a security incident

Creating Secure Code - Principles - Java

Secure By Implementation

- Psychological Acceptability
- Least Common Mechanism
- Validate Inputs
- Prevent Bypass Attacks
- Audit and Verify
- Defend in Depth

Secure Coding Principles
Defense in Depth - Introduction

- Principle: Use layered security defenses
  - Corollary: Defense in Depth is not just opportunism
- There many ways to defend your application
  - A given defense may fail if:
    - You make any kind of an error
    - Someone else is negligent or malicious
    - Circumstances change
    - New attack methods are developed
  - So it is better to have more than one layer of defense for every possible threat
  - Many defense strategies protect for multiple threats

Secure Coding Principles
Defense in Depth - Example

- To protect from SQL Injection, you use Input Validation, but decide not to use Prepared Statements
  - Error 1:
    - Later someone decides that certain characters need to be allowed in the input, like single quotes and dashes
  - Error 2:
    - A new encoding method devised by an attacker evades your validation
  - Error 3:
    - A new feature enhancements adds columns to the DB tables and input variables, but no one validates the new variable
  - Error 4:
    - A bug fix in the validation code creates an opportunity for the validation to be evaded

Secure Coding Principles
Defense in Depth - Introduction

- A fortress contains multiple defense systems
  - A perimeter moat
  - Outer wall
  - Outer archer towers
  - Higher inner wall
  - Inner archer towers
- Combined, you have a formidable defense
- You want your software to be a fortress for your assets

Secure Coding Principles
Defense in Depth - Introduction

- You have great system for managing administrative access to your application
  - It uses the IP address to grant or deny access
  - Unfortunately, the deployment staff is setting this value to 0.0.0.0 which permits all addresses
  - And anyone can access your administrative interface
- You solve this problem by requiring a username and password
  - But the deployment staff creates an account for test and install with credentials test/test
  - And leaves it active in production

Secure Coding Principles
Defense in Depth - Security Design Pattern

- Alias: Layered Security, Belt and Suspenders
- Some Forces:
  - Timelines and laziness can lead developers to ignore defense in depth
  - There is substantial evidence that security is very difficult to get right
  - Changing conditions can result in expensive patches to restore a secure situation
  - Negligence is responsible for up to a third of exploits
- Consequences:
  - Robust defenses are developed
  - Developers actively seek layered defenses

Secure Coding Principles
Defense in Depth - Tips

- When architecting an application, plan for layered defenses
  - Assume that any defense will eventually fail
  - Recognize the real threat from insiders
    - Maliciousness
    - Negligence
- If a defense can be overridden by a human mistake, plan for reviews to validate that the settings are correct
  - Deployment testing and reviews of configurations
  - Testing customer configurations and informing them of dangerous choices (cloud service offerings)
  - Validate that backups are encrypted

- Creating Secure Code - Principles
  - Understanding Secure Coding Principles
  - Common Secure Coding Principles
  - Summary

Common Secure Coding Principles
Summary

- Can we think of a one-line description for each principle?

Common Secure Coding Principles Summary

- Secure By Design
  - Establish Trust Boundaries
  - Don't Reinvent the Wheel
  - Economy of Mechanism
  - Trust Reluctance
  - Open Design
  - Minimize the Attack Surface
  - Secure the Weakest Link
- Secure By Default
  - Use Least Privilege
  - Use Default Deny
  - Fail Securely

Common Secure Coding Principles Summary

- Secure By Communication
  - Secure Trust Relationships
- Secure by Implementation
  - Psychological Acceptability
  - Least Common Mechanism
  - Validate Inputs
  - Secure Data at Rest
  - Prevent Bypass Attacks
  - Audit and Verify
  - Defense in Depth

Common Secure Coding Principles Summary

- Which of the design principles do you think is the most important? Why?
- Which might be the hardest sell to a development team?

References
Summary – Online References

- www.securityinnovation.com
- www.owasp.org
- www.us-cert.gov
- http://searchsecurity.techtarget.com

References
Summary – Books

- Introduction to Computer Security, Matt Bishop, Addison-Wesley
- Secure Coding: Principles and Practices, Mark Graff, et al, O'Reilley