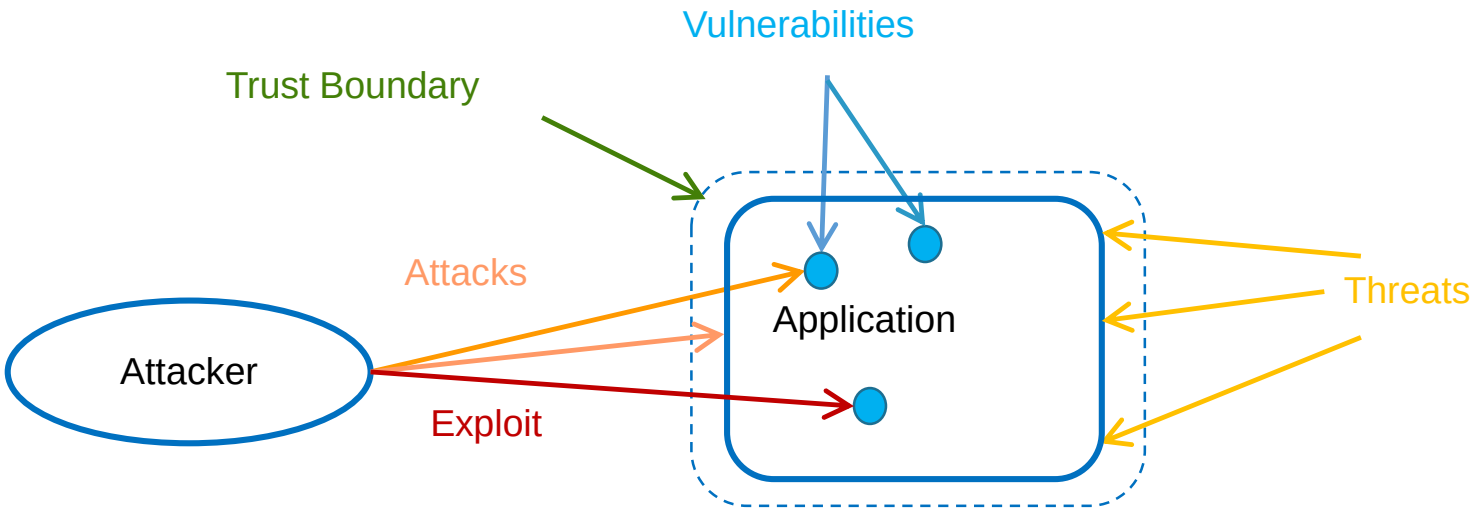
Engineering Secure Software

The Basics of Software Security Terminology



Vulnerability: a software defect with security consequences

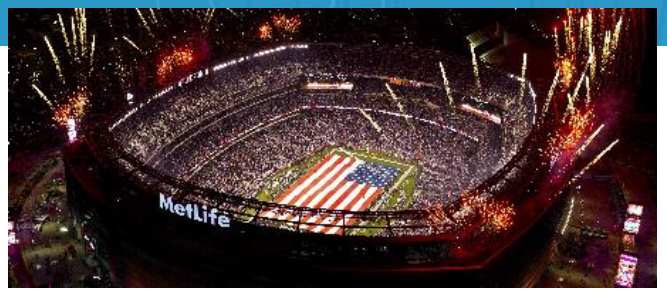
Threat: a potential danger to the software

Attack: an attempt to damage or gain access to the system

Exploit: a successful attack

Trust Boundary: where the level of trust changes for data or code

The Basics of Software Security
The Superbowl Example



- You are preparing for the Superbowl
- Who is your opponent?

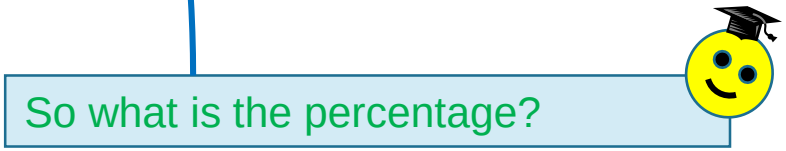
Football

- The Over-the-Hill Gang
- The Walking Wounded
- Last Year's Team
- Next Year's Team
- The Best Thing Ever

App Security

- Script Kiddies
- Hacktivists
- The Criminal Element
- A Disgruntled Employee
- Corporate Spy
- Black Hat Warrior

- Attackers can be anywhere
- And they may or may not know they are attackers
- The number one cause of data breaches is "employee error"
- But the malicious or criminally motivated elite hacker is still the focus
- AKA: A Threat Agent



- What are the threats to your team?

Football	App Security
– The Long Bomb – Three Yards and a Cloud of Dust – Run and Gun – T-Option – The Nickel Defense	– Spoofing – Tampering – Repudiation – Information Disclosure – Denial of Service – Elevation of Privilege

- The STRIDE Model defines 6 general threat types
  - Spoofing
  - Tampering
  - Repudiation
  - Information Disclosure
  - Denial of Service
  - Elevation of Privilege

Give an example of each?

The Basics of Software Security
Threats

- Threats represent a potential danger to the security of one or more assets or components
  - Threats could be malicious, accidental, due to a natural event, an insider, an outsider, …
  - A single software choice can result in many threats.
  - Threats exist even if there are no vulnerabilities
    - No relaxing
    - Threats change with system changes

How can a change in software result in either or fewer threats?

- Social threats: people are the primary attack vector
- Operational threats: failures of policy and procedure
- Technological threats: technical issues with the system
- Environmental threats: from natural or physical facility factors
- The threats themselves are the same, but this is a different view
  - Threats have certain sources (Social, Operational, Technical, Environments)
  - And certain security impacts (STRIDE)

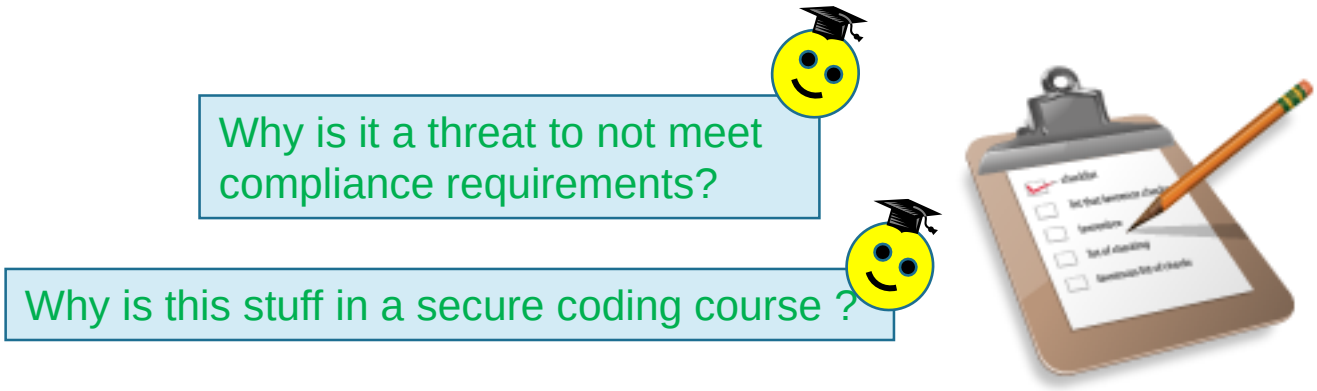
Which of these is likely to have the biggest impact on your development? Most likely to result in vulnerabilities? Hardest to contain?

- Intentional or accidental (carelessness or ignorance)
- Failure to restrict information and systems to the minimal set of people
- Failure to have adequate staff for security services
- Failure to adequately vet personnel with sensitive positions
- Malicious behavior

Would anyone like to offer an example?

The Basics of Software Security
Operational Threats

- Inadequate or improper policies, procedures or internal controls
- Inadequate change management or application monitoring
- Inadequate business continuity plans
- Failure to comply with legal or regulatory requirements or contractual agreements



- Implementation failures
- Design failures
- Interoperability issues
- Hardware and software compatibility
- Deployment failures

The Basics of Software Security
Environmental Threats

- Backups, data recovery and disaster recovery must be given sufficient attention
- The physical location of the systems and data must be secured
- Access to physical spaces must be limited and monitored

- What are your vulnerabilities?

Football	App Security
– Sore-armed quarterback	– Injection weakness
– Injured secondary	– Security Misconfiguration
– Poor pass defense	– Insecure logs
– Inaccurate kicker	– Weak Access Control
	– Open Redirect

The Basics of Software Security
Vulnerability

- A defect in software or the surrounding processes that could result in the compromise of system assets
- Vulnerabilities can be classified as:
  - Design vulnerability
  - Implementation vulnerability
  - Testing vulnerability
  - Deployment vulnerability
  - Patch and update vulnerability
  - Maintenance vulnerability
  - Environmental vulnerability

- What type of attacks can you expect?

Football	App Security
– Sideline buttonhook route – Crossing route – Fullback off-tackle – Bootleg right – Safety blitz – Seven-man front – Missed field goals	– SQL injection – Phishing against customer – Forceful browsing – Parameter Tampering – Facility break-in – SYN Flood

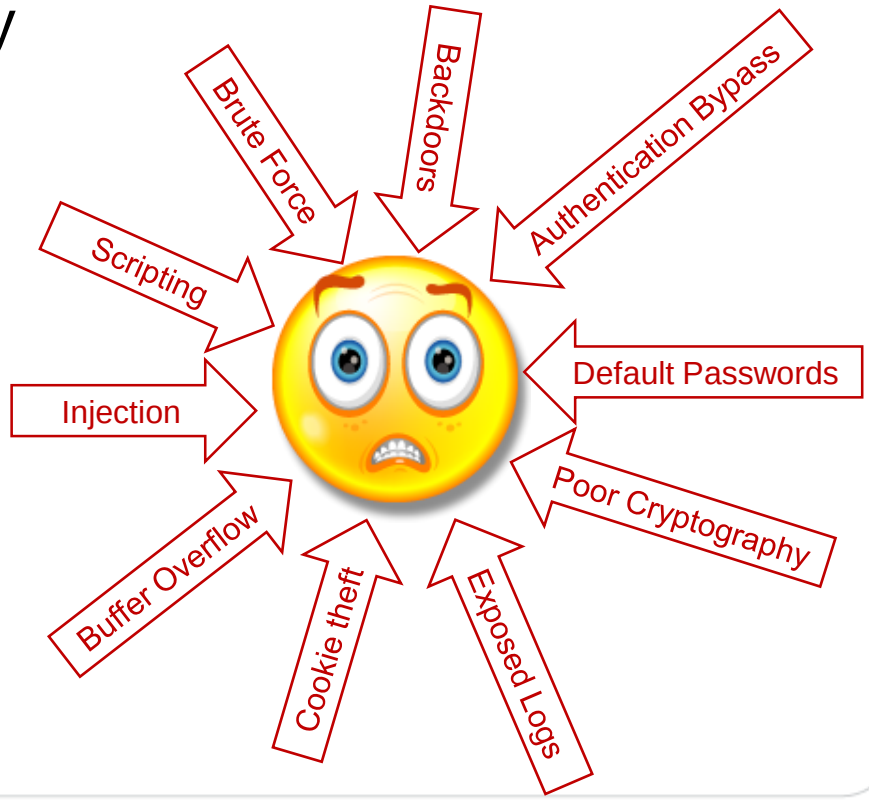
- Attacks
  - An attempt by someone, known as the attacker, to realize a threat against a system
  - The attacker can determine the various threats that exist for a system
    - And select different attacks and attack vectors to try
  - Successful attackers are generally clever and knowledgeable
  - There are many ways to implement attacks
  - An Attack Vector is a specific method of implementing an attack
    - Modifying data in a form field to issue an SQL Injection attack
    - Tampering with data in a POST request to issue a Cross-site Scripting attack
    - Entering a very long input to see what happens

How does an attacker know the Attack Surface?

Some Common Attack Vectors

- Malware: malicious software introduced into the system
- Denial of Service: hindering the proper functioning of the system
- Injection: forcing attacker code into the execution stream of the application
- Buffer Overflow: overwriting memory areas to create unexpected conditions
- Forceful Browsing: accessing areas of the code that should not be exposed to the user
- Parameter Tampering: modifying data during communication

- All of the threats impinging on a system are called the Attack Surface.
- As an architect or developer, you seek to minimize the Attack Surface by developing securely



Exploits

- When the attacker succeeds in an attack and can harm your system
- The exploit is a series of steps that attackers can use to do damage and in some cases, cover their tracks
- Once known, easily communicated through the Internet
- Zero-day attack exploit
  - An exploit against a previous unknown vulnerability that cannot be addressed quickly enough to prevent damage
- Breach: an exploit, especially one that involves the exposure of data

The Basics of Software Security
Risk

- The potential cost of a threat
- Risk = Prob(Exploit) x Expected Cost
- Direct and indirect damages, reputation loss, etc.
- The exploit causes some or all of those costs to be realized, but the potential cost is there because of the threat
- You don’t have to wait for the exploit to know what the cost might be, and you shouldn't

What would be the reputation damage to your company from an exploit that results in the exposure of data?

DREAD Model

$$Risk = (D + R + E + A + D)/5$$

- Where each of the following are evaluated 1 (low) - 5
  - D = Damage Potential
  - R = Reproducibility
  - E = Exploitability
  - A = Affected Users
  - D = Discoverability

- Evaluate the risk of each attack

Football

- Sideline buttonhook route (3.5)
- Crossing route (3.2)
- Fullback off-tackle (2.1)
- Bootleg right (4.1)
- Safety blitz (4.4)
- Seven-man front (1.6)
- Missed field goals (4.8)

App Security

- SQL injection (3.5)
- Phishing against customer (2.2)
- Forceful browsing (2.0)
- Parameter Tampering (3.6)
- Facility break-in (0.6)
- SYN Flood (1.1)

- Do Nothing; accept the risk
- Outsource the risk; transfer it to someone else
- Eliminate the asset
- Reduce the risk
  - Mitigate the vulnerability
  - Remove the threat

- Reduce the attack surface

Football	App Security
– Activate a new kicker	– Use Secure Design methods
– Go with the no-huddle offense	– Penetration Testing
– Use a 3-4 defense	– Use Code Reviews
– Quick kick on third down	– Centralized input processing
– Fake field goal	– Use secure communication

- During the design process, you need to determine:
  - What systems are inside Trust Boundary and what must be done with these systems to make them secure
  - What data is inside the Trust Boundary and how it becomes trusted
  - Perimeter security
  - Deployment and Post-deployment
- Can you relax inside the TB?
  - Inside attacks
  - Something could fail or have a flaw
  - The TB assures you that you have done everything you could think of
  - It does not assure you that you have done everything

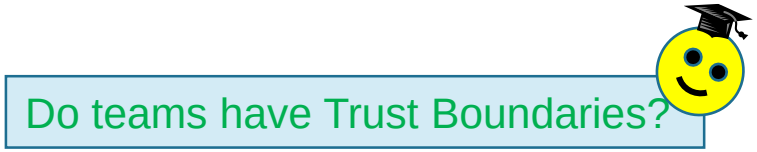
What are some things on the secure deployment checklist?

- When you have properly designed your system, you can be certain that everything inside your TB is secure
- Does that mean everything relaxes in the TB - NO
  - Inside attacks
  - Don't assume you are safe; something could fail or have a flaw
  - The TB assures you that you have done everything you could think of
  - It does not assure you that you have done everything
    - Malcontents are always busy
    - New attacks are found
    - Negligence is substantial

A new feature requires a server inside your pod. It needs to link directly to foreign systems. Comment.

- The TB helps you identify where you need to focus and where you need to be careful

- Are there Trust Boundaries at the Superbowl?
  - TB 1: Who gets into the stadium; what can they bring in
  - TB 2: Who gets on the field; when; how
  - TB 3: Who gets into the broadcast area
  - TB 4: What is allowed during the half-time show;



The Goals of Software Security

The Goals of Software Security
The Parkerian Hexad

- How do we define what security actually means?
- The Parkerian Model uses the hexad
- Defining 6 attributes of secure systems
- These are atomic and non-overlapping
- Every threat/exploit can be defined as affecting one or more of these attributes

The Goals of Software Security
Confidentiality

- To help understand, consider a single credit card number
  - Confidentiality: Only the owner of the card can see the credit card number (CCN) and CCV, or those to whom the owner gives that information
  - Possession: No one else has a duplicate card or a copy of the CCN and CCV
  - Integrity: The CCN and CCV are correct
  - Authenticity: The owner of the card provides the CCN and CCV and verifies ownership with a photo id. Identity is validated
  - Availability: When a purchase is attempted, the card can be verified by the processing center
  - Utility: There is sufficient credit for the intended purchase and the card is not expired

The Goals of Software Security
The Superbowl Example

- Is your game plan secure?

  - Confidentiality: No unauthorized person sees it
  - Possession: The middle linebacker doesn't leave his at Starbucks
  - Integrity: No one can modify the game plan
  - Authenticity: Every book is embossed with the picture of the coach
  - Availability: The books don't get left at home for the trip
  - Utility: The books are not printed as mirror images

The Goals of Software Security
Confidentiality and Possession

- Confidentiality: only authorized entities can access a unit of data
- Possession: there are no unauthorized duplicates
- Threats to Confidentiality and Possession come from
  - The application accessing assets
  - Inadvertent disclosure of information
  - Observing/Monitoring of users
  - Processes that copy the data

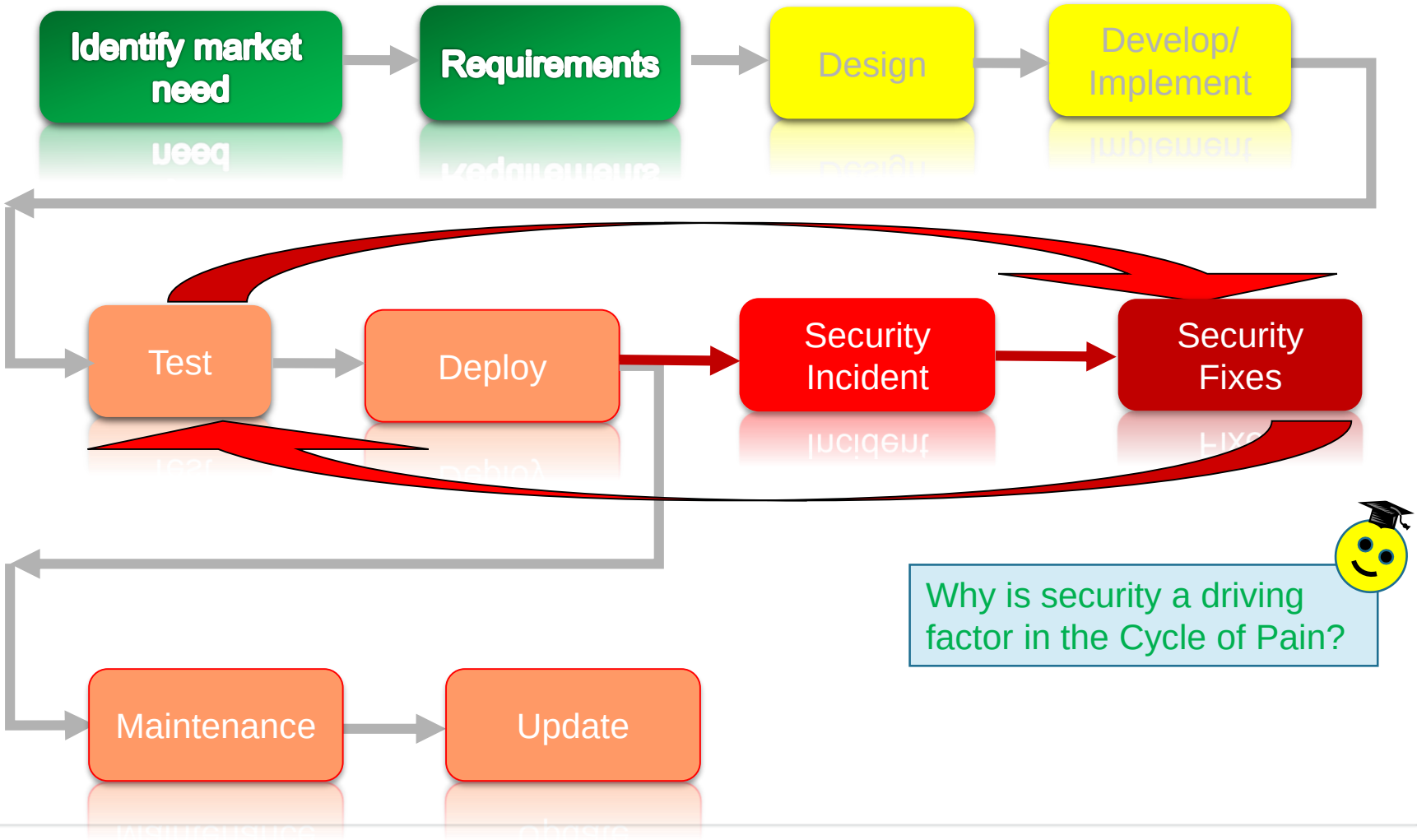
The Goals of Software Security
Integrity and Authenticity

- Integrity: the state of the system is consistent with the intended state
- Authenticity: claims of origin or authorship of the information are valid
- Threats to Integrity and Authentication come from
  - Entering, use or producing false data
  - Modify, replace or re-order data
  - SSL certificate passwords lostMisrepresent data
  - Repudiation (disavowal)
  - Misuse of data

The Goals of Software Security
Availability and Utility

- Availability: the system provides timely access to information
- Utility: the system and data are useful
- Threats to Availability and Utility come from:
  - Destruction
  - Damage
  - Disruption
  - Contamination
  - Denial, prolonging or delay of access

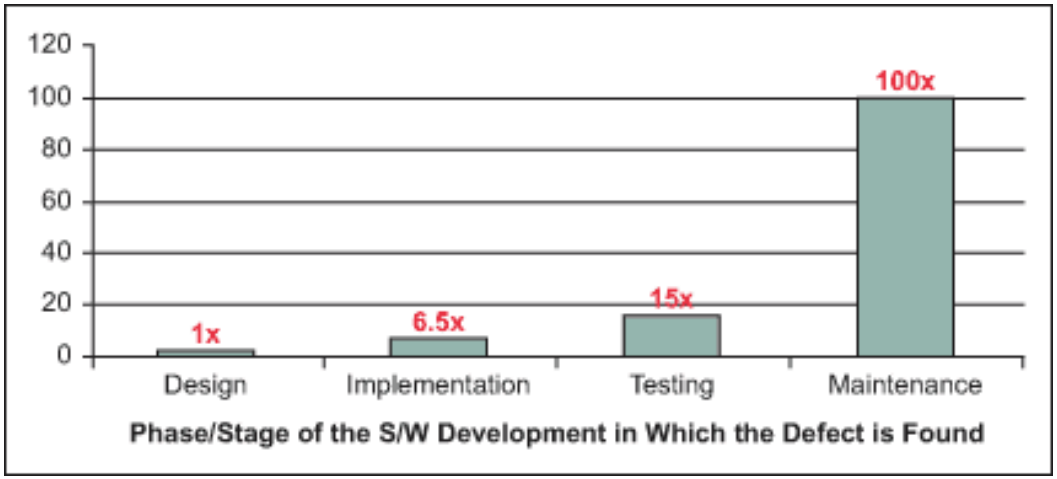
The Goals of Software Security
The Cycle of Pain



Why is security a driving factor in the Cycle of Pain?

The Goals of Software Security
Exploits and their impact on software development

- The phase-cost relationship for fixes
  - Vulnerabilities will occur
  - You want to find them early to avoid the exponentially increasing costs of finding them later (Gartner Group)
  - A secure development lifecycle can achieve this



IBM System Sciences Institute

Why would the cost of fixes increase exponentially with phase?

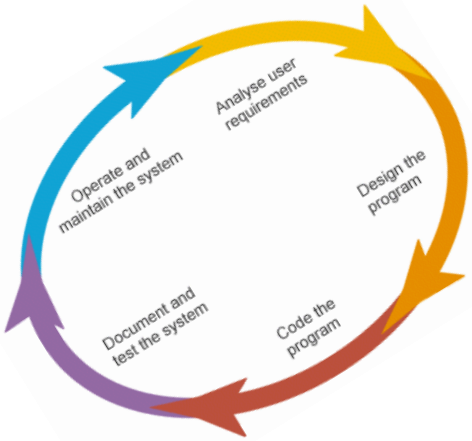
The Goals of Software Security
Tension between Security and Traditional Software Goals

- Complexity is the enemy of security
- Secure software takes longer to develop
- More security means less user-friendly, less convenience
- More security means more difficult design

The Software Development Lifecycle

The Software Development Lifecycle
Introduction

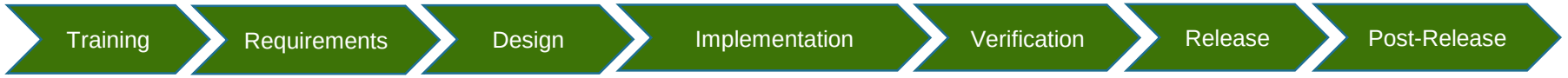
- The Secure Development Lifecycle is much like the traditional Software Development Lifecycle



  - It has additional steps related to security
  - It is really about enhancing the quality of software development
  - It seeks to avoid the cycle of pain by avoiding vulnerabilities
    - And increasing the likelihood that all vulnerabilities are found early

The Software Development Lifecycle
Phases of the SDL

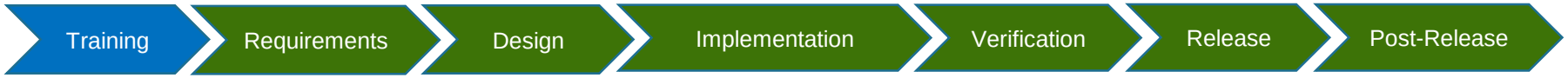
- An Overview of the Phases
  - Education and Awareness
  - Project Initiation
  - Design
  - Implementation
  - Verification
  - Release
  - Post-release



The Software Development Lifecycle
Education and Awareness

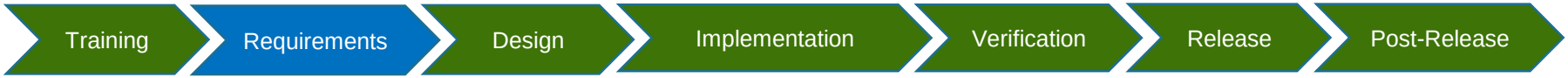
- Education is the key to a successful SDL
  - Security involves every employee
  - Every employee should have security awareness training
    - Learn the responsibilities of the individual
    - Common social and operational threats
    - Reporting procedures
  - All employees involved in the SDL are required to have:
    - Training on the entire SDL
    - Specialized training on the parts of the SDL in which they participate

What percentage of employees claim that they don't know much about their companies security goals?



The Software Development Lifecycle
Project Initiation

- Determine the security requirements for the project
  - Sensitive data involved
  - Legal requirements (COPPA, HIPAA, Sarbanes-Oxley, …)
  - Compliance requirements (PCI-DSS, DISA, …)
  - Communication between internal networks and the outside world (email, customer forums, …)
  - Establish quality gates: the minimum acceptable levels of privacy and security
- Assign a Security Advisor
  - A member of the security team
  - Acts as a point of contact between the development and security teams
  - Aids the development team in conducting security operations



- Plan for scope broadening and feature creep. Be flexible.
- Be precise in describing security requirements. Assumptions are the enemy of good design
- Pay attention to
  - The deployment environments
  - Any software or other components this software must work with and any security standards adhered to
- Explicit requirements for behavior and constraint
  - What should happen
  - What cannot happen

What would be an example of planning for the future?

Requirement-PDH-13.1

- Web Form: PDH
- Name: Operation
- Values: Deposit, Withdraw, Transfer, Check Balance
- Must: Allow user to choose one of the options and reset web form for specific format (See PDH-1.0)
- Must Not:
  - Allow user to choose any other value
  - Allow user operation value to be changed after submission
  - User cannot be allowed to choose an operation that is not permitted by account authorization

Why is the list of Must Nots larger than the list of Musts.

- What would the requirements be?

  - Red Zone: must score 75% of the time; must not have turnovers
  - Possession: must control the ball at least 60% of the time; must not control it less than 50%
  - Quarterback: must limit sacks to less than 5; must not have any blind side sacks

  - Must validate inputs; must not have injection attacks
  - Must encrypt all communications; must not allow the release of SSL certificates
  - Must patch security defects; must not install unvalidated patches

The Software Development Lifecycle
Design

- Best time to identify and avoid vulnerabilities
  - Many security vulnerabilities start here
  - Defects that originate here are the most difficult to fix because they are “baked-in”
  - The goal of the Secure Design Process
    - Will focus your attention on prioritized threats
    - Will help you create a process that protects the security of the application

What would be a problematical baked-in design?

Training | Requirements | **Design** | Implementation | Verification | Release | Post-Release

- Commit to follow design best practices (a later course)
- Perform a risk assessment
  - What portions of the design will require Threat Models
  - What portions of the design will require Security Reviews
  - What portions of the design will require penetration testing
  - What are the other security testing requirements
  - Helps you decide where you need to invest in security
  - Every aspect of the lifecycle is under consideration

✓ Design a secure process	✓ Maintain the product securely
✓ Code securely	✓ Update and patch securely
✓ Test securely	✓ Document securely
✓ Deploy securely	

- Perform a risk analysis (not the same as assessment)
  - Create Threat Models
    - They will help you identify threats to the application
  - Analyze the threats
    - Determine how the threats could impact the application
    - Where is the application likely to be attacked and how
  - Evaluate the risk
    - *risk = probability of exploit x expected damage*
    - Knowing the risk for each threat allows the best allocation of mitigation effort
  - Plan the mitigations
    - This covers everything from creating an authentication model to a database access structure to the content of cookies

- The risk analysis gets you to here in an organized fashion

Football	App Security
– Sideline buttonhook route (3.5) – Crossing route (3.2) – Fullback off-tackle (2.1) – Bootleg right (4.1) – Safety blitz (4.4) – Seven-man front (1.6) – Missed field goals (4.8)	– SQL injection (3.5) – Phishing against customer (2.2) – Forceful browsing (2.0) – Parameter Tampering (3.6) – Facility break-in (0.6) – SYN Flood (1.1)

- Create a secure design and coding plan
  - It details designs and procedures to be followed with respect to
    - How code reviews are to be conducted
    - Code to be developed by Super Teams
    - Tools to be used
    - How changes to design are to be reviews
    - How specific coding situations are to be handled

✓ The authentication model
✓ The authorization model
✓ Session management
✓ Input data handling
✓ Output data handling

✓ Database access
✓ Error handling
✓ Communication
✓ Cryptography
✓ Other security related items

Does this sounds like a game plan?

- This is your game plan
  - In a new project, you start from zero, otherwise, you work with what you have
  - Who are our opponents; what are their strengths?
  - What we going to do to protect ourselves
  - How we will execute each part of the development process
  - The roles of the team members
  - What tools we have to make the work easier
  - The next step – how do we evaluate our plan

- Create a Security Test Plan
  - There is no other part of the organization that can create security tests better than the Development Team in collaboration with the Security Team
    - Unit tests to be run during Implementation
    - Final security tests
    - Final review process
    - Tools to be used

0. Security Test PT-12-121
1. Use a web proxy and turn on intercept
2. Open the Payment Interface and select a test user
3. Open the Credit Card dialog
4. Click on the current invoice and then click Pay
5. In the web proxy, modify the id to a different value and forward
6. You should get an error return for a type 124 error – all others are incorrect returns
7. Repeat for all parameters

- Create a Secure Incident Handling Plan
  - Security incidents cannot be handled like normal defects
  - Create a Security Response Team
    - Security Team
    - Development
    - Legal and compliance
    - Product Management
    - Management

  - Create a process for handling a security incident
    - Define a triage process
    - Define information flows
    - Establish a final authority

The Software Development Lifecycle
Implementation

- Most vulnerabilities originate here
  - Coders don't see the big picture
  - Coders may be given too much leeway

- Train your coders to understand security issues
  - Alert coders can identify security issues before they get "baked-in
- Coders must be held accountable to the Secure Coding Plan
  - Code reviews
  - Automated static code analysis
  - Unit testing

What is the number one reason coders give for security defects?

Training | Requirements | Design | **Implementation** | Verification | Release | Post-Release

- Use compiler defenses
  - /GS in .Net compilers helps prevent buffer overflows
  - /SAFESH in .Net compilers to prevent exception hijacking
- Static code analysis
  - Not a guarantee that all vulnerabilities will be found
  - They generate many false positives
  - Don't work well in mixed language situations
  - Not all languages covered
  - Obviously can't find defects that occur at run-time or due to environmental, social or operational threats
  - HP Fortify, Checkmarx, many others

- Avoid certain common coding mistakes
  - Failure to check input lengths
  - Failure to check for integer overflow/underflow
  - Failure to test inputs for malicious data
  - Allowing user data into executable streams
  - Failure to properly handle cryptography
  - Using functions with dangerous properties (gets, printf, SetSecurityDescriptorDacl)
  - Hidden HTML fields

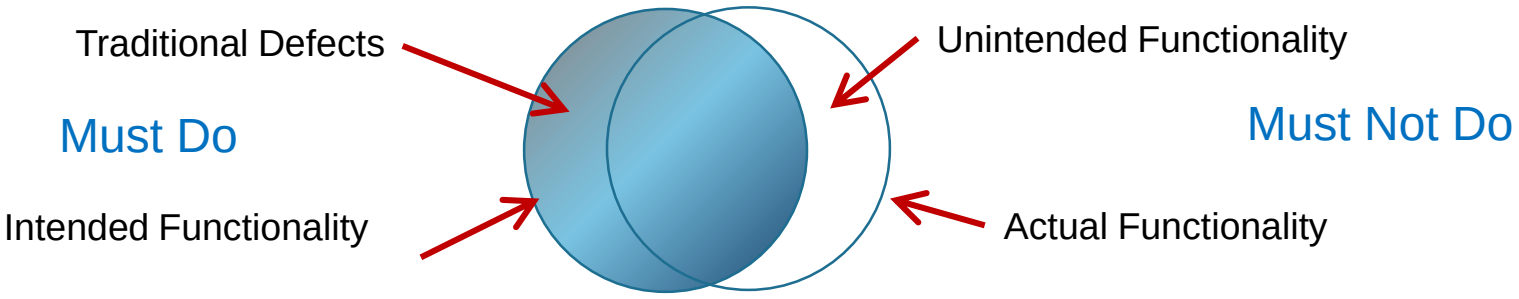
- Assumptions are the bane of security: assume nothing

Can you think of some times where assumptions ended badly?

- The best way to prevent implementation phase errors
  - 75% reduction in defects is possible (Software Management, Boehm)
- It achieves accountability plus the “extra set of eyes’ advantage
- Typically increases coding time by one-fourth to one-third
- Follow the code review rules
  - Train your coders to review properly
  - Define and follow a prescribed format
  - Use a checklist and a written code review report
  - Change coder/reviewer pairs constantly

The Software Development Lifecycle
Verification

- Security testing is very different from functional testing
  - Functional tests look for things that the software should do, but doesn't.
  - Security tests look for things the software shouldn't do, but does
  - There are a limited number of ways software can be right, but it the ways it can be wrong are many and varied



Eclipse Model

Training | Requirements | Design | Implementation | Verification | Release | Post-Release

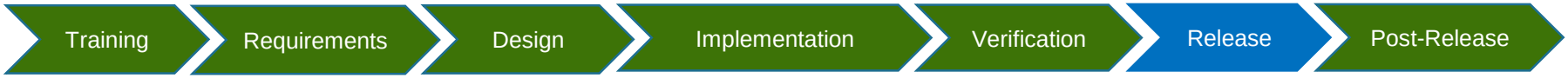
The Software Development Lifecycle
Verification

- To security test, think like an attacker
  - Where is the unintended functionality
  - How are things working underneath the visible components
- Security testing must be continuous throughout the lifecycle
  - Code churn, new features and bug fixes
  - You can't be certain that work will be done well
  - Or that the Secure Coding Plan will be followed
  - Many good security choices are defeated in the update cycle

- The Final Security Review is that last stage of verification
  - Review the current state
    - Any known existing vulnerabilities
    - Triage and assign for mitigation
  - Review Threat Models and compare to current software
    - Generate a gap analysis and plan for addressing gaps
  - Review the current Attack Surface
    - Plan future efforts for reduction
  - Verify that all testing has been completed
    - Review testing for indications of problems
  - Review the Security Incident Handling Plan
  - Determine if product can proceed to Release

The Software Development Lifecycle
Release

- Many vulnerabilities result from deployment processes
- Deployment must
  - Protect production data
  - Remove all testing and deployment access accounts
  - Remove all deployment tools
  - Modify all support account passwords
  - Insure that the installed configuration is secure
- Checklists and flowcharts are a valuable tools
- Have the deployment reviewed before final acceptance



Post-Release: Updating and patching

- There will be patches and updates because attackers don't rest
- Customers must understand the need for security patches
  - Use detailed advisories
  - Make the process simple, timely and effective
  - Be prepared to back it out
- Use patch and update verification to prevent spoofing
  - Protect the patch and update servers
  - Validate all patches and updates before installation

- Security continues with the same importance
  - Manage backups and logs just like production data
    - Secure storage and encryption
  - Control access to all systems, even those assumed to be safe
- Support
  - Support personnel may be the first to see a security vulnerability or breach report.
  - Security awareness training is critical
  - Knowledge of the Security Incident Response Plan is necessary
  - Support personnel must be trained and have resources to help customers configure systems securely.

- As shown, the SDL follows the Microsoft SDL
  - It appears to follow the Waterfall development method
  - It can be applied to any methodology
  - It works much better on a new project
  - It is difficult to backport security fixes without breaking existing functionality
- Re-engineering for security
  - Create Threat Models, analyze risk and prioritize threats
  - Combine threats to minimize code churn
  - Where possible, combine mitigations with upgrade work
  - Create a Security Test Plan to aid in finding existing vulnerabilities

- One-time practices
  - Requirements phase
    - Establish security requirements
    - Conduct security and privacy risk assessments
  - Design phase
    - Establish design requirements
    - Create Threat Models and perform a risk analysis
    - Perform attack surface reduction
    - Create the Security Incident Response Plan
  - Release phase
    - Implement the Security Incident Response Plan

Agile Model: Bucket Practices

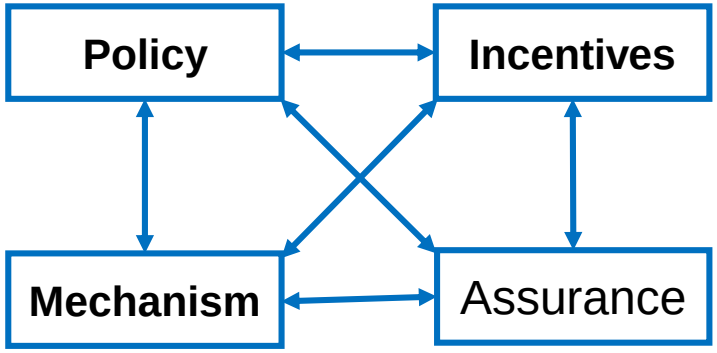
- Requirements phase
  - Create quality gates
- Verification phase
  - Execute Security Plan testing
  - Apply dynamic analysis
  - Conduct an attack surface review

Agile Model: Every-sprint Practices

- Design phase
  - Create threat models
  - Perform risk analyses
  - Create remediation plans
- Implementation phase
  - Follow the Secure Coding Guidelines
  - Perform a static analysis
- Release phase
  - Conduct the Final Security Review
  - Assess for release

Security Engineering

- A specialized field that focuses on the security of software systems
- SSE attempts to bring four things together
  - Policy: what you need to achieve
  - Incentives: The reasons that the systems need to be secured; and the reasons the attackers want to evade security
  - Mechanism: the tools, components processes and procedures that you can use to secure a system
  - Assurance: the reliability of the system and the mechanisms



Policy

- Policy is defined by the project requirements
  - What needs to be private –
  - What needs to be confidential – who has access
  - What are the operational requirements for the system
  - Legal and compliance requirements for the system
  - Who is allowed to access which assets
  - Required access procedures

Policy

- Incentives are defined by
  - The assets of the system: what is to be protected
    - PII, credit card numbers, systems, network access, student grades
  - The value of the assets: to the owner and to others
    - The cost you if you lose PII
    - The value of your sales contacts
    - The value of a stolen credit card number or the ransom value of damaging information
  - The convertibility of the assets: can they be converted to use outside of the system
    - Can you sell a stolen credit card number?
    - Is there any value to a list of user names?

Software Security Engineering
Mechanisms

- Mechanisms are the available defenses for the assets
  - Procedures to create more secure code
    - Threat Models, Risk Analyses, Secure Coding Guidelines
  - Methods to protect data
    - Cryptography, compiler security options, Principles of Secure Development
  - Tools to test and validate software
    - Static Code Analyzers, Dynamic Web Application Scanners, Fuzzing Tools, Web Proxies
  - Tools to secure perimeters and systems
    - Firewalls
    - Intrusion Detection Systems
    - Network Sniffers

- Assurance means to pledge or declare a degree of confidence. SSE is sometimes called Information Assurance
  - This is the part of SSE where the results of the efforts are validated
  - How reliable is the security of an application
    - What is the probability of an exploit?
    - How much interference in use of the software is created by security measures?
  - The answers are impossible to define exactly, but can be estimated
    - By looking at the results of code reviews and testing
    - By assessing Post-release security incidents reported
  - This is important to validate and improve the secure design process
  - And to develop trust

Assurance - Example

- What is your level of assurance regarding airport passenger security
  - What percentage of weapons escape detection (50%)
  - What percentage of items confiscated are actually weapons
    - It is vanishingly small
  - What is the level of interference with normal use of the system
    - Dramatically worse
- How would you decide if the costs are worthwhile?
  - What is the cost of a failure?

- What attributes are found in successful software security engineers
  - Knowledge of
    - Operating systems (MS Windows, Linux, OSX, …)
    - Languages (HTML, JavaScript, Java, C#, C/C++, Python, …)
    - Protocols (TCP/IP, HTTP, SSL, …)
    - The SDL
    - Programming and programming methods
    - Networks
    - Security tools
    - Testing tools

Personal Attributes for Security Engineers

- Sufficient knowledge of security to know where to look
- An inquiring mind
- The ability to think like an attacker
- An evil streak

- Think about what the software might do that is unexpected
- Be contrarian; do the opposite of what is expected
- Think about what the developers might have missed
- Think about what is going on underneath the surface
- Think about how you might defend the application and where they might be a weakness
- Model the attack surface; plan your tests, execute meticulously and take copious notes
- Make no assumptions because you know something about the product and the development process. The attacker won't. Assumptions will destroy you.

- A Security Engineer hopes to accomplish four things
  - Often called the Four Pillars of Software Security

Secure by Design
Secure by Default
Secure by Implementation
Secure in Communications

- Secure by Design
  - The design of the software is secure
  - The design process is secure
- Secure by Default
  - Optional features are configured to be secure
  - Default settings are the “secure” choice
  - Less secure is a user’s choice

- Secure by Implementation
  - Secure coding standards followed
  - The design is properly tested
  - Installation, provisioning and updates are all done securely
  - Maintenance procedures maintain security
- Secure in Communications
  - The network is secure
  - Communication methods are as secure as possible
  - The software accounts for all communication unknowns