Akraino Security Development Lifecycle (Akraino SDL)

Akraino SDL Applicability

- Deployed in a business or enterprise environment
- Processes personally identifiable information (PII) or other sensitive information
- Communicates regularly over the Internet or other network

Akraino SDL Overview

- Security training
- Requirements
- Design
- Implementation
- Verification
- Release
- Response

Akraino SDL Roles

- Security advisor/Privacy advisor
    - Auditor
    - Expert
- Team security champion/privacy champion
    - Negotiate, accept, and track of minimum security and privacy requirements
    - Maintain clear lines of communication with advisors and decision makers

Security Training

- Secure design
- Threat modeling
- Secure coding
- Security testing
- Privacy
- Security response processes

Requirements

- Team subject to SDL policies?
- Security bug reporting tools
- Security bug bar
- 3rd-party code licensing security requirements
- Security plan
- Cost Analysis

Security Bug Tracking

- Security bug effect
  - Not a Security Bug
  - **S**poofing
  - **T**ampering
  - **R**epudiation
  - **I**nformation Disclosure
  - **D**enial of Service
  - **E**levation of Privilege
  - Attack Surface Reduction

- Security bug cause
  - Not a security bug
  - Buffer overflow/underflow
  - Arithmetic error (for example, integer overflow)
  - SQL/Script injection
  - Directory traversal
  - Race condition
  - Cross-site scripting
  - Cryptographic weakness
  - Weak authentication
  - Weak authorization/Inappropriate permission or access control list (ACL)
  - Ineffective secret hiding
  - Unlimited resource consumption (Denial of Service [DoS])
  - Incorrect/No error messages
  - Incorrect/No pathname canonicalization
  - Other

Security Plan

- Team training
- Threat modeling
- Security push
- Final security review

Cost Analysis

- Security risk assessment
- Project privacy impact rating

Security Risk Assessment

- What portions of the project will require threat models before release.
- What portions of the project will require security design reviews before release.
- What portions of the project will require penetration testing (*pen testing*)
- Any additional testing or analysis requirements the security advisor deems necessary to mitigate security risks.
- Clarification of the specific scope of *fuzz testing* requirements

Privacy Impact Rating

- P1 high privacy risk
- P2 Moderate privacy risk
- P3 Low privacy risk

Design

- Risk Analysis
- Best Practices

Risk Analysis

- STRIDE threat model analysis
  - Threats and vulnerabilities
  - External code
  - Threat models
  - Design review for P1 privacy projects
  - Detail privacy analysis
- NEAT security user experience

STRIDE

- Spoofing of user identity
- Tampering
- Repudiation
- Information disclosure (privacy breach or data leak)
- Denial of service
- Elevation of privilege

NEAT security UX

- Necessary
- Explained
- Actionable
- Tested

Best Practices

- Secure design principles
- Security design review
- Security architecture
- Assets & threat actors identified and addressed
- Identity and Access Management
- Cryptograph
- Mitigate against XSS
- Use memory-safe languages
- Be careful with error message
- Strong log-out and session management
- Confidentiality
- Integrity
- Availability

Secure Design Principles

- Secure defaults
- Defense-in-depth
- Separation of privilege
- Least privilege
- Least common mechanism
- Psychological acceptability
- Minimize default attack surface
- Input validation with whitelists

Security Architecture

- Attack surface measurement
- Product structure or layering

Cryptograph

- Use AES for symmetric enc/dec.
- Use 128-bit or better symmetric keys.
- Use RSA for asymmetric enc/dec and signatures.
- Use 2048-bit or better RSA keys.
- Use SHA-256 or better for hashing and message-authentication codes.
- Support certificate revocation.
- Limit lifetimes for symmetric keys and asymmetric keys without associated certificates.
- Support cryptographically secure versions of SSL (must not support SSL v2).
- Use cryptographic certificates reasonably and choose reasonable certificate validity periods.
- Properly use Transport Layer Security (TLS) when communicating with another entity
    - Check the **Common Name** attribute to be sure it matches the host with which you intended to communicate.
    - Verify that your service consults a certificate revocation list (CRL) for an updated list of revoked certificates at a frequent interval.
    - If your service is accessible via a browser, confirm that no security warnings appear at any visited URL for any supported browser.

Confidentiality

- Passwords stored on server as iterated salted hashes using bcrypt
- Remember me token: Cryptographic nonce is stored on client & bcrypt digest stored on server
- Email addresses only revealed to owner & admins
- HTTPS

Integrity

- HTTPS
- Data modification requires authorization
- Modifications to official application requires authentication

Availability

- Cloud & CDN deployment
- Timeout
- Can return to operation quickly after DDOS attack stops
- Login disabled mode
- Multiple backups

Implementation

- Common types of vulnerable implementations
- Hardening
- Securely reuse
- Deprecate unsafe functions
- Use approved tools
- Static code analysis

OWASP top 10 vulnerabilities

- Injection (including SQL injection)
- Auth & session
- XSS (Esp. SafeBuffer)
- Insecure object references
- Security misconfiguration
- Sensitive data exposure
- Missing access control
- CSRF
- Known vulnerabilities
- Unvalidated redirect/fwd
- XXE (2017 A4)
- Insecure Deserialization (2017 A8)
- Insufficient logging and monitoring (2017 A10)

Hardening

- Force HTTPS, including via HSTS (Http strict transport security)
- Hardened outgoing HTTP headers, including restrictive CSP
- HTTP-only Cookies
- User secure cookie over HTTPS
- CSRF token hardening
- Incoming rate limits
- Address Space Layout Randomization (ASLR)
- Harden or disable XML entity resolution
- Load DLLs securely
- Reflection and authentication relay defense
- Safe redirect, online only
- Do not use the Javascript eval() or equivalent functions
- Integer overflow/underflow
- Input validation and handling
- Encrypted email addresses
- Gravatar restricted

Securely reuse

- Review before use
- Get authentic version
- Use package manager

Verification

- Dynamic Program Analysis
    - AppVerifier
    - Sandbox
- Fuzz Testing
- Threat Model and Attack Surface review
- Penetration Test

Release

- Incident Response Plan
    - An identified sustained engineering team
    - On-call contacts with decision-making authority
    - Security servicing plan for code inherited from other group
    - Security servicing plan for licensed 3rd-party code
- Final security review
    - Examination of
        - Threat models
        - Exception requests
        - Tool output
        - Performance against the previously determined quality gates or bug bars
- Release/Archive
    - Certify
    - Archive all pertinent information and data

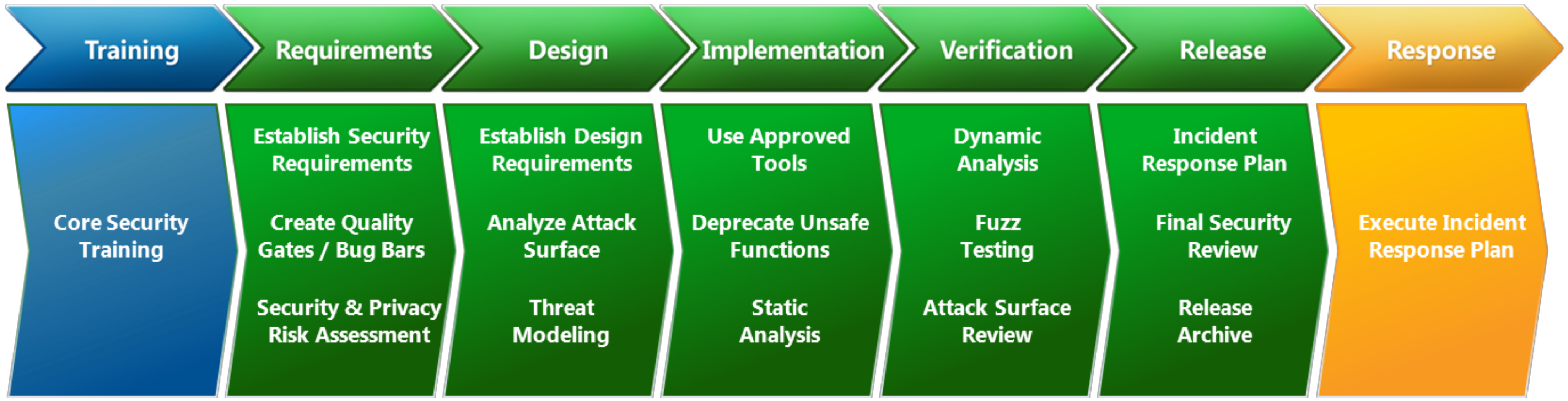
FSR Outcomes

- Passed FSR
- Passed FSR with exceptions
- FSR with escalation

Response

- Security servicing and response execution

Simplified SDL Security Activities



Training	Requirements	Design	Implementation	Verification	Release	Response
Core Security Training	Establish Security Requirements	Establish Design Requirements	Use Approved Tools	Dynamic Analysis	Incident Response Plan	Execute Incident Response Plan
	Create Quality Gates / Bug Bars	Analyze Attack Surface	Deprecate Unsafe Functions	Fuzz Testing	Final Security Review	
	Security & Privacy Risk Assessment	Threat Modeling	Static Analysis	Attack Surface Review	Release Archive	