Development Process for Secure Software
Main Thing About Secure Software

- Secure software comes from a disciplined security-oriented development process
- You cannot just pen-test or bug bounty the security into a product that is insecure
The Processes Discussed in This Talk

- Microsoft SDL [Microsoft, 2016, Microsoft, 2011]
- BSIMM [BSIMM, 2022]
But There Is More!

- Rugged Software [Rugged, 2012]
- NIST Secure Software Development Framework (SSDF) [Souppaya et al., 2022]
- Software Assurance Maturity Model (SAMM) [OpenSAMM, 2009]
Security Touchpoints
Security Touchpoints: the Basics

- Developed by Gary McGraw (Cigital)
- More oriented towards programming
  - “All software projects produce at least one artifact: source code”
- Small, prescriptive, easy to implement
The Seven Touchpoints

Code Review with a Tool

- Aim: catching implementation bugs early
- Tool helps to achieve good code coverage
- Aim for good, not perfect
First step: create description of architecture
- Start with one page
- Forest-level view

Attack resistance
- Use checklists of known attacks
- Example: Microsoft STRIDE (Spoofing, Tampering, Repudiation, Info disclosure, Denial of service, Elevation of privilege)
Architectural Risk Analysis 2/3

- Ambiguity analysis
  - Discover new risks
  - Find unclear parts in how the system works
  - Trust, data sensitivity, threat models

- Weakness analysis
  - Impact of external software dependencies
  - Platform (hardware, OS)
  - Frameworks
  - Called services
Architectural Risk Analysis 3/3

- Combine risks and consider business impact
- Rank risks
- Find solutions
Penetration Testing 1/2

- Use the source (white-box testing)
  - Otherwise people send time on reverse-engineering your system
- Apply business priorities
  - Logic flaw vs. XSS flaw
  - XSS is important if it contributes towards compromising business logic
Penetration testing 2/2

- Use in-house QA department
  - They already know the system
  - Use tools and training to add security testing skills
- Test more than once
  - Incorporate the findings back into development
- Test based on priorities
  - Architectural risks
  - Risks discovered during code review
- Test malicious input
  - Use fuzzing tool
Abuse Cases

- Security is not a set of features
- How system should react to illegitimate use
- Like use cases, but with malicious users
Microsoft Security Development Lifecycle (SDL)
Microsoft SDL: the Basics

- Started as Trustworthy Computing Directive, published in 2002
- Main principles, called the SD3+C
  - Secure by Design
  - Secure by Default
  - Secure in Deployment
  - Communications
- An analogous concept to SD3+C for privacy is known as PD3+C
Secure By Design

- Secure architecture, design, and structure
- Threat modeling and mitigation
- Elimination of vulnerabilities
- Improvements in security
Secure by Default

- Least privilege
- Defense in depth
- Conservative default settings
- Avoidance of risky default changes
- Less commonly used services off by default
Secure in Deployment

- Deployment guides
- Analysis and management tools
- Patch deployment tools
Communications

- Security response
- Community engagement
SDL Development Process Phases

- **Training**: Core Security Training
  - Establish Security Requirements
  - Create Quality Gates / Bug Bars
  - Security & Privacy Risk Assessment

- **Requirements**: Establish Design Requirements
  - Analyze Attack Surface
  - Threat Modeling

- **Design**: Use Approved Tools
  - Deprecate Unsafe Functions
  - Static Analysis

- **Implementation**: Dynamic Analysis
  - Fuzz Testing
  - Attack Surface Review

- **Verification**: Incident Response Plan
  - Final Security Review
  - Release Archive

- **Release**: Execute Incident Response Plan
Building Security In Maturity Model (BSIMM)
BSIMM Basics

- Describes software security related practices in different organizations
- “Real practices, not best practices”
- Systematic overview of development practices of 128 organizations
  - Companies from different domains: software vendors, SaaS providers, banks, medical companies, etc.
  - For every practice, it is given how much of the selection uses it
### BSIMM Software Security Framework

#### DOMAINS

<table>
<thead>
<tr>
<th>GOVERNANCE</th>
<th>INTELLIGENCE</th>
<th>SSDL TOUCHPOINTS</th>
<th>DEPLOYMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Practices that help organize, manage, and measure a software security initiative. Staff development is also a central governance practice.</td>
<td>Practices that result in collections of corporate knowledge used in carrying out software security activities throughout the organization. Collections include both proactive security guidance and organizational threat modeling.</td>
<td>Practices associated with analysis and assurance of particular software development artifacts and processes. All software security methodologies include these practices.</td>
<td>Practices that interface with traditional network security and software maintenance organizations. Software configuration, maintenance, and other environment issues have direct impact on software security.</td>
</tr>
</tbody>
</table>

#### PRACTICES

<table>
<thead>
<tr>
<th>GOVERNANCE</th>
<th>INTELLIGENCE</th>
<th>SSDL TOUCHPOINTS</th>
<th>DEPLOYMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEVEL 1</td>
<td>LEVEL 1</td>
<td>LEVEL 1</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td></td>
</tr>
</tbody>
</table>
| - [SM1.1] Publish process and evolve as necessary.  
- [SM1.3] Educate executives on software security.  
- [SM1.4] Implement lifecycle instrumentation and use to define governance. | - [CP1.1] Unify regulatory pressures.  
- [CP1.2] Identify PII obligations.  
- [CP1.3] Create policy. | - [T1.1] Conduct software security awareness training.  
- [T1.7] Deliver on-demand individual training.  
- [T1.8] Include security resources in onboarding. |

<table>
<thead>
<tr>
<th>LEVEL 2</th>
<th>LEVEL 2</th>
<th>LEVEL 2</th>
</tr>
</thead>
</table>
| - [SM2.1] Publish data about software security internally and drive change.  
- [SM2.2] Verify release conditions with measurements and track exceptions.  
- [SM2.3] Create or grow a satellite.  
- [SM2.6] Require security sign-off prior to software release.  
- [SM2.7] Create evangelism role and perform internal marketing. | - [CP2.1] Build PII inventory.  
- [CP2.2] Require security sign-off for compliance-related risk.  
- [CP2.3] Implement and track controls for compliance.  
- [CP2.4] Include software security SLAs in all vendor contracts.  
- [CP2.5] Ensure executive awareness of compliance and privacy obligations. | - [T2.5] Enhance satellite through training and events.  
- [T2.8] Create and use material specific to company history.  
- [T2.9] Deliver role-specific advanced curriculum. |

<table>
<thead>
<tr>
<th>LEVEL 3</th>
<th>LEVEL 3</th>
<th>LEVEL 3</th>
</tr>
</thead>
</table>
| - [SM3.1] Use an internal tracking application with portfolio view.  
- [SM3.2] SSI efforts are part of external marketing.  
- [SM3.3] Identify metrics and use them to drive resourcing.  
- [CP3.2] Impose policy on vendors.  
- [CP3.3] Drive feedback from software lifecycle data back to policy. | - [T3.1] Reward progression through curriculum.  
- [T3.2] Provide training for vendors and outsourced workers.  
- [T3.3] Host software security events.  
- [T3.4] Require an annual refresher.  
- [T3.5] Establish SSG office hours.  
- [T3.6] Identify new satellite members through observation. |
<table>
<thead>
<tr>
<th>LEVEL 1</th>
<th>LEVEL 1</th>
<th>LEVEL 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>· [AM1.2] Create a data classification scheme and inventory.</td>
<td>· [SFD1.1] Integrate and deliver security features.</td>
<td>· [SR1.1] Create security standards.</td>
</tr>
<tr>
<td>· [AM1.3] Identify potential attackers.</td>
<td>· [SFD1.2] Engage the SSG with architecture teams.</td>
<td>· [SR1.2] Create a security portal.</td>
</tr>
<tr>
<td>· [AM1.5] Gather and use attack intelligence.</td>
<td></td>
<td>· [SR1.3] Translate compliance constraints to requirements.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LEVEL 2</th>
<th>LEVEL 2</th>
<th>LEVEL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>· [AM2.1] Build attack patterns and abuse cases tied to potential attackers.</td>
<td>· [SFD2.1] Leverage secure-by-design components and services.</td>
<td>· [SR2.2] Create a standards review board.</td>
</tr>
<tr>
<td>· [AM2.2] Create technology-specific attack patterns.</td>
<td>· [SFD2.2] Create capability to solve difficult design problems.</td>
<td>· [SR2.4] Identify open source.</td>
</tr>
<tr>
<td>· [AM2.5] Maintain and use a top N possible attacks list.</td>
<td></td>
<td>· [SR2.5] Create SLA boilerplate.</td>
</tr>
<tr>
<td>· [AM2.6] Collect and publish attack stories.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>· [AM2.7] Build an internal forum to discuss attacks.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LEVEL 3</th>
<th>LEVEL 3</th>
<th>LEVEL 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>· [AM3.1] Have a research group that develops new attack methods.</td>
<td>· [SFD3.1] Form a review board or central committee to approve and maintain secure design patterns.</td>
<td>· [SR3.1] Control open source risk.</td>
</tr>
<tr>
<td>· [AM3.2] Create and use automation to mimic attackers.</td>
<td>· [SFD3.2] Require use of approved security features and frameworks.</td>
<td>· [SR3.2] Communicate standards to vendors.</td>
</tr>
<tr>
<td>· [AM3.3] Monitor automated asset creation.</td>
<td>· [SFD3.3] Find and publish secure design patterns from the organization.</td>
<td>· [SR3.3] Use secure coding standards.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· [SR3.4] Create standards for technology stacks.</td>
</tr>
<tr>
<td>SSDL TOUCHPOINTS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>------------</td>
</tr>
<tr>
<td><strong>ARCHITECTURE ANALYSIS (AA)</strong></td>
<td><strong>CODE REVIEW (CR)</strong></td>
<td><strong>SECURITY TESTING (ST)</strong></td>
</tr>
<tr>
<td><strong>LEVEL 1</strong></td>
<td><strong>LEVEL 1</strong></td>
<td><strong>LEVEL 1</strong></td>
</tr>
<tr>
<td>- [AA1.1] Perform security feature review.</td>
<td>- [CR1.2] Perform opportunistic code review.</td>
<td>- [ST1.1] Ensure QA performs edge/boundary value condition testing.</td>
</tr>
<tr>
<td>- [AA1.3] Have SSG lead design review efforts.</td>
<td>- [CR1.5] Make code review mandatory for all projects.</td>
<td>- [ST1.4] Integrate opaque-box security tools into the QA process.</td>
</tr>
<tr>
<td>- [AA1.4] Use a risk methodology to rank applications.</td>
<td>- [CR1.6] Use centralized reporting to close the knowledge loop.</td>
<td></td>
</tr>
<tr>
<td><strong>LEVEL 2</strong></td>
<td><strong>LEVEL 2</strong></td>
<td><strong>LEVEL 2</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>- [ST2.6] Perform fuzz testing customized to application APIs.</td>
</tr>
<tr>
<td><strong>LEVEL 3</strong></td>
<td><strong>LEVEL 3</strong></td>
<td><strong>LEVEL 3</strong></td>
</tr>
<tr>
<td>- [AA3.2] Drive analysis results into standard design patterns.</td>
<td>- [CR3.3] Create capability to eradicate bugs.</td>
<td>- [ST3.4] Leverage coverage analysis.</td>
</tr>
<tr>
<td>LEVEL 1</td>
<td>LEVEL 1</td>
<td>LEVEL 1</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>- [PT1.1] Use external penetration testers to find problems.</td>
<td>- [SE1.1] Use application input monitoring.</td>
<td>- [CMVM1.1] Create or interface with incident response.</td>
</tr>
<tr>
<td>- [PT1.2] Feed results to the defect management and mitigation system.</td>
<td>- [SE1.2] Ensure host and network security basics are in place.</td>
<td>- [CMVM1.2] Identify software defects found in operations monitoring and feed them back to development.</td>
</tr>
<tr>
<td>- [PT1.3] Use penetration testing tools internally.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LEVEL 2</th>
<th>LEVEL 2</th>
<th>LEVEL 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>- [PT2.2] Penetration testers use all available information.</td>
<td>- [SE2.2] Define secure deployment parameters and configurations.</td>
<td>- [CMVM2.1] Have emergency response.</td>
</tr>
<tr>
<td>- [PT2.3] Schedule periodic penetration tests for application coverage.</td>
<td>- [SE2.4] Protect code integrity.</td>
<td>- [CMVM2.2] Track software bugs found in operations through the fix process.</td>
</tr>
<tr>
<td></td>
<td>- [SE2.5] Use application containers to support security goals.</td>
<td>- [CMVM2.3] Develop an operations inventory of software delivery value streams.</td>
</tr>
<tr>
<td></td>
<td>- [SE2.6] Ensure cloud security basics.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- [SE2.7] Use orchestration for containers and virtualized environments.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LEVEL 3</th>
<th>LEVEL 3</th>
<th>LEVEL 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>- [PT3.1] Use external penetration testers to perform deep-dive analysis.</td>
<td>- [SE3.2] Use code protection.</td>
<td>- [CMVM3.1] Fix all occurrences of software bugs found in operations.</td>
</tr>
<tr>
<td>- [PT3.2] Customize penetration testing tools.</td>
<td>- [SE3.3] Use application behavior monitoring and diagnostics.</td>
<td>- [CMVM3.2] Enhance the SSDL to prevent software bugs found in operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- [CMVM3.4] Operate a bug bounty program.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- [CMVM3.5] Automate verification of operational infrastructure security.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- [CMVM3.6] Publish risk data for deployable artifacts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- [CMVM3.7] Streamline incoming responsible vulnerability disclosure.</td>
</tr>
</tbody>
</table>
You Can Measure Yourself
First Step in BSIMM Implementation

- According to BSIMM, the first step (100% of examined companies) is to create a software security group (SSG)
  - “The internal group charged with carrying out and facilitating software security. According to our observations, the first step of a software security initiative (SSI) is to form an SSG.”
- Can be one person
- SSG is helped with satellite
  - A group of individuals, often called champions, that is organized and leveraged by a software security group (SSG).
Questions?
References I


References II

Simplified Implementation of the Microsoft SDL.


Software Assurance Maturity Model.
https://www.opensamm.org/.
References III

   Rugged Software.
   https://ruggedsoftware.org/.

https://doi.org/10.6028/NIST.SP.800-218.