The Security Development Lifecycle

Steve Lipner
Senior Director of Security Engineering Strategy
Trustworthy Computing
Microsoft Corporation
SLipner@microsoft.com
+1 425 705-5082

OWASP
24 June 2010

Copyright © The OWASP Foundation
Permission is granted to copy, distribute and/or modify this document under the terms of the OWASP License.

The OWASP Foundation
http://www.owasp.org
Overview

- How we got here
- Selling the process
- The SDL at Microsoft
- Managing change
- Automation and tools
- The Simplified SDL: Adapting the SDL to new organizations
- Objections
- Resources
- Questions?
How We Got Here

- Through 1980s, security was about insiders
  - Studies and experiments demonstrated potential for attacks on software
  - No real examples
  - “Nobody would ever...”

- Computer security treated as a theoretical problem
  - Prove it’s secure and you’re done forever
  - Market proved unsympathetic (or absent) – projects canceled, no real products
How We Got Here

■ PC and Internet changed the rules
  ‣ Viruses, information sharing, “outside” and “inside” indistinguishable
  ‣ Vulnerability research for reputation

■ Vulnerability research led to security response process
  ‣ Fix the problems when they’re found

■ “Secure Windows Initiative” to make software secure
  ‣ Assigned three program managers to review Windows
  ‣ Evolved to training and “bug bashes”
How We Got Here

- Thought we’d done “better” with XP, and then...
  - Code Red
  - Nimda
  - UPNP

From: Bill Gates
Sent: Thursday, 18, 2002
Subject: Trustworthy Computing

As I've talked with customers over the last year - from individual consumers to big enterprise customers - it's clear that everyone recognizes that computers play an increasingly important and useful role in our lives. At the same time, many of the people I talk to are concerned about the security of the technologies they depend on...
How We Got Here: The Security Push Era

- **Security push**
  - Team-wide stand-downs and training
  - Threat model, review code, run tools, conduct tests, modify defaults
  - (Relatively) quick way to significant improvement
  - Immature and ad hoc processes

- **“Security science”**
  - Identify and remove new classes of vulnerabilities

- **Security “audit”**
  - Independent review – what did the push miss?
Selling the Process

- Security pushes were an “obviously” necessary response...
- Security pushes achieved rapid improvements (some dramatic) but...
- Leverage comes from early (design time) focus on security
- Ongoing attacks demonstrated continued need
- Executive buy-in surprisingly easy in retrospect
  - Everyone understood what bad things could happen
  - Security pushes had accomplished enough to allow us to claim we could do this
The Classic SDL at Microsoft

- Training:
  - Core training

- Requirements:
  - Analyze security and privacy risk
  - Define quality gates

- Design:
  - Threat modeling
  - Attack surface analysis

- Implementation:
  - Specify tools
  - Enforce banned functions
  - Static analysis

- Verification:
  - Dynamic/Fuzz testing
  - Verify threat models/attack surface

- Release:
  - Response plan
  - Final security review
  - Release archive

- Response:
  - Response execution

Education → Technology and Process → Accountability

Ongoing Process Improvements
SDL for Agile at Microsoft

- Requirements defined by frequency, not phase
  - Every-Sprint (most critical)
  - One-Time (non-repeating)
  - Bucket (all others)
- Great for projects without end dates, like cloud services
Managing Change

- The first (2004) iteration of the SDL was pretty rough
  - Developed rapidly based on security push lessons

- Initial updates at 6-month intervals
  - Responses to new threats
  - New application classes (privacy, online services)
  - New requirements and techniques (e.g. banned APIs, new fuzzers)

- Since SDL v4 (October 2007), annual updates
  - More time for tool development
  - More time for beta and feedback
  - More time for usability

- Every update receives both broad and senior review
Automation and Tools

- At Microsoft today, the SDL requires three classes of tools
  - Automated tools to help find (and remove or mitigate) security problems
  - Automated tools to help product teams record and track their compliance with the SDL
  - Automated tools to help the MSEC PM (security advisor) help the product teams
- We started with only the first (problem finders)
- All three are critical to our implementation of the SDL – and we’ve changed our release cadence largely in recognition of this fact
Who Needs the SDL?

Subject: I swear, i'm giving our kids normal names...

Today’s XKCD (http://xkcd.com/327/)

Hi, this is your son’s school. We’re having some computer trouble.

Oh, dear—did he break something? In a way—

Did you really name your son Robert? Drop Table Students;--?

Oh, yes. Little Bobby Tables, we call him.

Well, we’ve lost this year’s student records. I hope you’re happy.

And I hope you’ve learned to sanitize your database inputs.
Adapting the SDL to Organizations Beyond Microsoft

- Non-proprietary
- Scalable to organizations of any size
- Platform agnostic
- Based on the SDL process used at Microsoft
Pre-SDL Requirement: Security Training

Assess organizational knowledge – establish training program as necessary

- Establish training criteria
  - Content covering secure design, development, test and privacy
- Establish minimum training frequency
  - Employees must attend $n$ classes per year
- Establish minimum acceptable group training thresholds
  - Organizational training targets (e.g. 80% of all technical personnel trained prior to product RTM)
Phase One: Requirements

Opportunity to consider security at the outset of a project

- Establish Security Requirements
  - Project wide requirements – security leads identified, security bug tracking process mandated, architectural requirements set given the planned operational environment

- Create Quality Gates / Bug Bars
  - Minimum performance and quality criteria for each stage and for the project as a whole,

- Security and Privacy Risk Assessment
  - Risk assessment performed to determine critical components for the purposes of deep security and privacy review
Define and document security architecture, identify security critical components

- Establish Design Requirements
  - Required activities which include creation of design specifications, analysis of proposed security technologies (e.g. crypto requirements) and reconciliation of plans against functional specs.

- Analyze Attack Surface
  - Defense in depth strategies employed – use of layered defenses used to mitigate severity.

- Threat Modeling
  - Structured, component-level analysis of the security implications of a proposed design.
Phase Three: Implementation

Determine processes, documentation and tools necessary to ensure secure development

- Use approved tools
  - Approved list for compilers, security test tools, switches and flags; enforced project wide.
- Deprecate Unsafe Functions
  - Ban unsafe functions, APIs, when using native (C/C++) code.
- Static Code Analysis
  - Scalable in-depth code review, augmentation by other methods as necessary to address weaknesses in static analysis tools.
Phase Four: Verification

Verification of SDL security and privacy activities performed earlier in the process

- **Dynamic Analysis**
  - Runtime verification and analysis of programs to identify critical security problems

- **Fuzz Testing**
  - Specialized dynamic analysis technique used to deliberately cause program failure by injection of random, deliberately malformed inputs.

- **Attack Surface / TM review**
  - Re-review of attack surface and threat models when the program is “code complete” to ensure security assumptions and mitigations specified at design time are still relevant.
Phase Five: Release

- 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

- Response

Satisfaction of clearly defined release criteria – consistent with organizational policy

- Incident Response Plan
  - Creation of a plan that outlines engineering, management and “on-call” contacts, security servicing plans for all code, including 3rd party artifacts.

- Final Security Review
  - Deliberate examination of all security and privacy activities conducted during development

- Release Archive
  - SDL compliance certification and archival of all information and data necessary for post-release servicing of the software.
Post-SDL Requirement: Response

“Plan the work, work the plan…”

- **Execute Incident Response Plan**
  - Performance of activities outlined in response plan created during Release phase

- **Other non-development, post-release process requirements**
  - Root cause analysis of found vulnerabilities; failure of human, process, or automation. Addressed immediately and tagged for inclusion in next revision of SDL
Objections to the SDL

“...only for Windows”
  ▶ Based on proven, generally accepted security practices
  ▶ Appropriate for non-Microsoft platforms

“...for shrink-wrapped products”
  ▶ Also covers Line of Business (LOB) and online services development

“...for waterfall or spiral development”
  ▶ Agile methods are also supported

“...requires Microsoft tools”
  ▶ Use the appropriate tools for the job

“...requires Microsoft-level resources to implement”
  ▶ SDL as its applied at Microsoft != SDL for other development organizations
  ▶ Some smaller organizations have adopted
Who Uses the SDL?

- Short answer: we don’t know
- You have to click through a EULA to download the tools, but you don’t have to register so...
- We have worked with some large organizations on adopting and adapting the SDL (mostly not public)
- We’ve seen the Errata survey, and had some users (large and small) tell us they’re using the SDL
- Finding the answer is one of our objectives for the next year
Resources at a glance...
SDL Threat Modeling Tool

Transforms threat modeling from an expert-led process into a process that any software architect can perform effectively.

Provides:
- Guidance in drawing threat diagrams
- Guided analysis of threats and mitigations
- Integration with bug tracking systems
- Robust reporting capabilities
SDL Template for VSTS (Spiral)

- Incorporates
  - SDL requirements as work items
  - SDL-based check-in policies
  - Generates Final Security Review report
  - Third-party security tools
  - Security bugs and custom queries
  - A library of SDL how-to guidance

- Integrates with previously released free SDL tools
  - SDL Threat Modeling Tool
  - Binscope Binary Analyzer
  - Minifuzz File Fuzzer

The SDL Process Template integrates SDL 4.1 directly into the VSTS software development environment.
MSF Agile + SDL Template for VSTS

- Automatically creates new security workflow items for SDL requirements whenever users check in code or create new sprints
- Ensures important security processes are not accidentally skipped or forgotten
- Integrates with previously released free SDL tools
  - SDL Threat Modeling Tool
  - Binscope Binary Analyzer
  - Minifuzz File Fuzzer
- Will be updated for VS2010

Incorporates SDL-Agile secure development practices directly into the Visual Studio IDE - now available as beta (planned release at the end of Q2CY10)
Binscope Binary Analyzer

- Provides an extensive analysis of an application binary

- Checks done by Binscope
  - /GS - to prevent buffer overflows
  - /SafeSEH - to ensure safe exception handling
  - /NXCOMPAT - to prevent data execution
  - /DYNAMICBASE - to enable ASLR
  - Strong-Named Assemblies - to ensure unique key pairs and strong integrity checks
  - Known good ATL headers are being used

- Use either standalone or integrated with Visual Studio (VS) and Team Foundation Server (TFS)
MiniFuzz File Fuzzer

MiniFuzz is a basic testing tool designed to help detect code flaws that may expose security vulnerabilities in file-handling code.

- Creates corrupted variations of valid input files
- Exercises the code in an attempt to expose unexpected application behaviors.
- Lightweight, for beginner or advanced security testing
- Use either standalone or integrated with Visual Studio (VS) and Team Foundation Server (TFS)
Summary

- You’re here, so you all understand the importance of building secure software.
- Integrating security into a development process and organization requires commitment and time.
- Our experience has shown that the SDL is an effective process – and that it can be applied beyond Microsoft.
- We’ve made a lot of resources freely available to help other organizations apply the SDL.
Online Resources

SDL Portal
http://www.microsoft.com/sdl

SDL Blog
http://blogs.msdn.com/sdl/

SDL Process on MSDN (Web)

Simplified Implementation of the Microsoft SDL
http://go.microsoft.com/?linkid=9708425
Questions?