The Security Development Lifecycle

Steven B. Lipner, CISSP
SLipner@microsoft.com
Senior Director
Security Engineering Strategy
Microsoft Corp.
Overview

- Introduction
- A look back
- Trustworthy Computing and security pushes
- The Security Development Lifecycle
- Training
- Tools
- Results
- What next?
Introduction

- SLipner@microsoft.com
- Microsoft employee for 6+ years
- 35+ years in security
  - Mathematical models
  - A1 systems
  - Firewalls
  - Patents in system and network security
- Learned from my (and industry’s mistakes)
A Look Back

- Security integrated into Windows NT
- Microsoft Security Response Team
- Secure Windows Initiative (SWI) team
- Each OS release placed more emphasis on security
- And then there was 2001
  - Code Red
  - Nimda
  - UPNP
Trustworthy Computing

- Deliver products and services that customers can rely on
- Integrate trust into Microsoft culture
Make significant security improvement

Jumpstart Trustworthy Computing

Apply (then) current understanding of security best practices

- .NET Framework
- Windows Server 2003
- Office 2003
- SQL Server 2000 SP3
- Exchange 2000 SP3
Building a Process

“Security Audit” introduced with Windows Server 2003

“How good a job did we do?”

Through 2003, SWI team

Provided training

Supported security pushes

Conducted “audits”

By late 2003, time to formalize
The Security Development Lifecycle
Security Engineering Framework

SD³ + Communications

Secure by Design
- Threat Modeling
- Code Inspection
- Security Testing

Secure by Default
- Reduce attack surface area
- Unused features off by default
- Least privilege

Secure in Deployment
- Prescriptive Guidance
- Security Tools
- Enterprise Management

Communications
- Clear security commitment
- Full member of the security community
- Microsoft Security Response Center
Security Development Lifecycle Tasks and Processes

Security Kickoff & Register with SWI
Security Design Best Practices
Security Arch & Attack Surface Review
Threat Modeling
Create Security Docs and Tools for Product
Prepare Security Response Plan
Security Push Pen Testing
Final Security Review
Security Servicing & Response Execution

Requirements → Design → Implementation → Verification → Release → Support & Servicing
Key Components of the SDL

- Threat modeling
  - Enumerate external entities, processes, stores, flows
  - Identify “threats” – STRIDE
  - Determine risks
  - Plan mitigations – security features and code quality

- Development best practices
  - Banned APIs
  - Static analysis
  - SAL Annotation
Key Components of the SDL

- Fuzz testing
  - Supply well-formed random input; investigate failures
  - Applies to file parsers, RPC, ActiveX controls
- Tools
  - Static analysis and fuzz testing must be automated
  - Other tools developed as needed – some “weapons grade”
  - Improves efficiency and consistency
Final Security Review (FSR)

- Successor to 2002-3 “audit”
- “From a security viewpoint, is this software ready to deliver to customers?”
- Did the team meet the SDL requirements?
- How secure is this software?
- NOT “penetrate and patch”
- If FSR finds a pattern of vulnerabilities, revisit earlier phases and address root causes (e.g., improve training, enhance tools) – don’t just fix the bugs and ship!
Maintaining the SDL

- SDL is NOT static!
  - Respond to new threats
  - Exploit new tools, techniques, processes
- SDL updated on a six-month cycle
  - Proposal (with costs and benefits)
  - Beta
  - Final review and release
- Proposed updates reviewed broadly by Microsoft security and engineering teams
Knowledge of security is not the norm among developers/college graduates

Microsoft operates its own training program

- Basics of Secure Software Design, Development, and Test
- Introduction to Fuzz Testing
- Threat Modeling
- Security Design and Architecture - Time Tested Design Principles
- Implementing Threat Mitigations
- Introduction to SDL and the FSR Process
- Security Tools Overview
- Security Code Reviews
- Network Fuzz Testing
- Security for Upper Management
- Vista Security Features
- Classes of Security Defects
- Defects in Detail
- Attack Surface Analysis and Reduction
- Exploit Development
- Defect Estimation and Management
- Introduction to Cryptography
- Security Response
SDL Tools

- Visual Studio 2005
  - Native code static analysis (/analyze aka PREfast)
  - Safe CRT libraries
  - Managed code static analysis (FXCop)
  - Code Access Security/Least Privilege (.Net)
  - /GS improvements
  - Testing tools (AppVerifier)
  - Unit/Stress testing framework

- Full integrated solutions in VS Team System
  - Stand alone versions available shortly

Results

1111 Days After Product Release

* As of May 9, 2006

Copyright © Microsoft Corp 2006
“We actually consider Microsoft to be leading the software industry now in improvements in their security development life cycle [SDL].”

John Pescatore
Vice President and Distinguished Analyst Gartner, Inc
(From CRN, Feb 13th 2006)
What Next?

- Making the SDL available to the community
  - Howard/Lipner book a first step
  - Additional tools, training, partners to follow
- Continuing to improve the SDL
  - Process improvements
  - More and better tools
  - Blocking new classes of vulnerabilities and exploits
- Working to improve the Common Criteria security evaluation scheme
Summary

- SDL has proven effective at improving software security

- Key success factors
  - Executive commitment
  - Education and training
  - Continuous improvement

- Process is widely applicable to software development teams
Emulating the SDL

- Get management support
- Build a core security team
  - Including a security response center
- Define your SDL process
  - Select your initial tools
- Train your people
- Learn from your mistakes
  - Update your products and your processes