Secure Systems Engineering and Management

A Data-driven Approach

Secure by Design: Overview

Michael Hicks

UPenn CIS 7000-003
Spring 2026

You Know the Attacks. Now What?

Scenario: You're hired to build the university's new course management system.

You've studied the CWE Top 25:

- Buffer overflows, SQL injection, XSS, path traversal, …

Question: How do you make sure *none* of those end up in production?

Limes
Dashboard
Courses
Calendar
Resources
Log In
Sign Up
Empowering Learning, Simplifying Education
Your Complete Course Management System
Easy Course Access
Secure & Reliable
Engage & Inspire
Modern & Intuitive Interface
Easy to navigate, easy to use.
Powerful Tools & Integrations
Quizzes, Assignments, Video & More.
24/7 Support
We're here wheiever you need us.
Explore Our Features
Course Management
Gradebook & Assessments
Virtual Classrooms
About
Support
Privacy
Contact Us
(not real! Drawn by CoPilot)

Our Running Example: Limes

A university course management system (Canvas, Blackboard, …)

- **Users:** Students, TAs, instructors, admins
- **Data:** Grades, submissions, personal records (FERPA)
- **Features:** File upload, discussion forums, quizzes, video integration
- **Integrations:** SSO, plagiarism detection, LTI plugins, email

We'll use this system throughout to ground every concept

The Reactive Security Trap

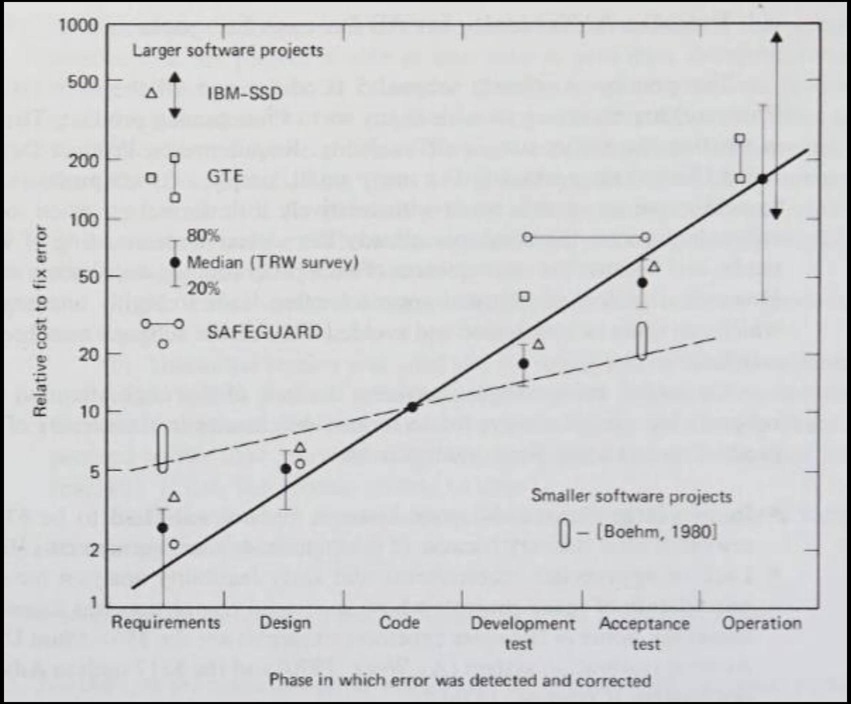
Most of cybersecurity grew up **reacting** to insecure systems:

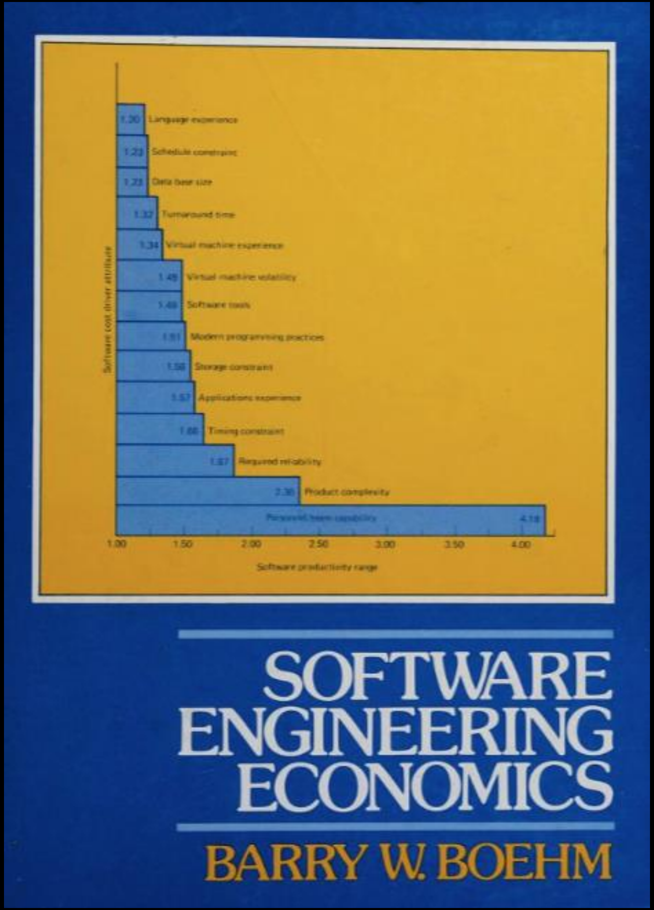
- Patch management
- Configuration hardening
- Malware detection
- Incident response

. . .

“Frequent security bulletins and advisories, disruptive patching, complex configuration management, malware analysis and detection, and vulnerability disclosure may have created more security ‘fatigue’ than progress.”
– CIS/SAFECode, *Secure by Design* (2025)

The Cost of Finding Bugs Late





Published 1981

What Does “Secure by Design” Mean?

Secure by Design = building security in from the start, as a first-class design constraint

Not a binary state. A continuum of risk management.

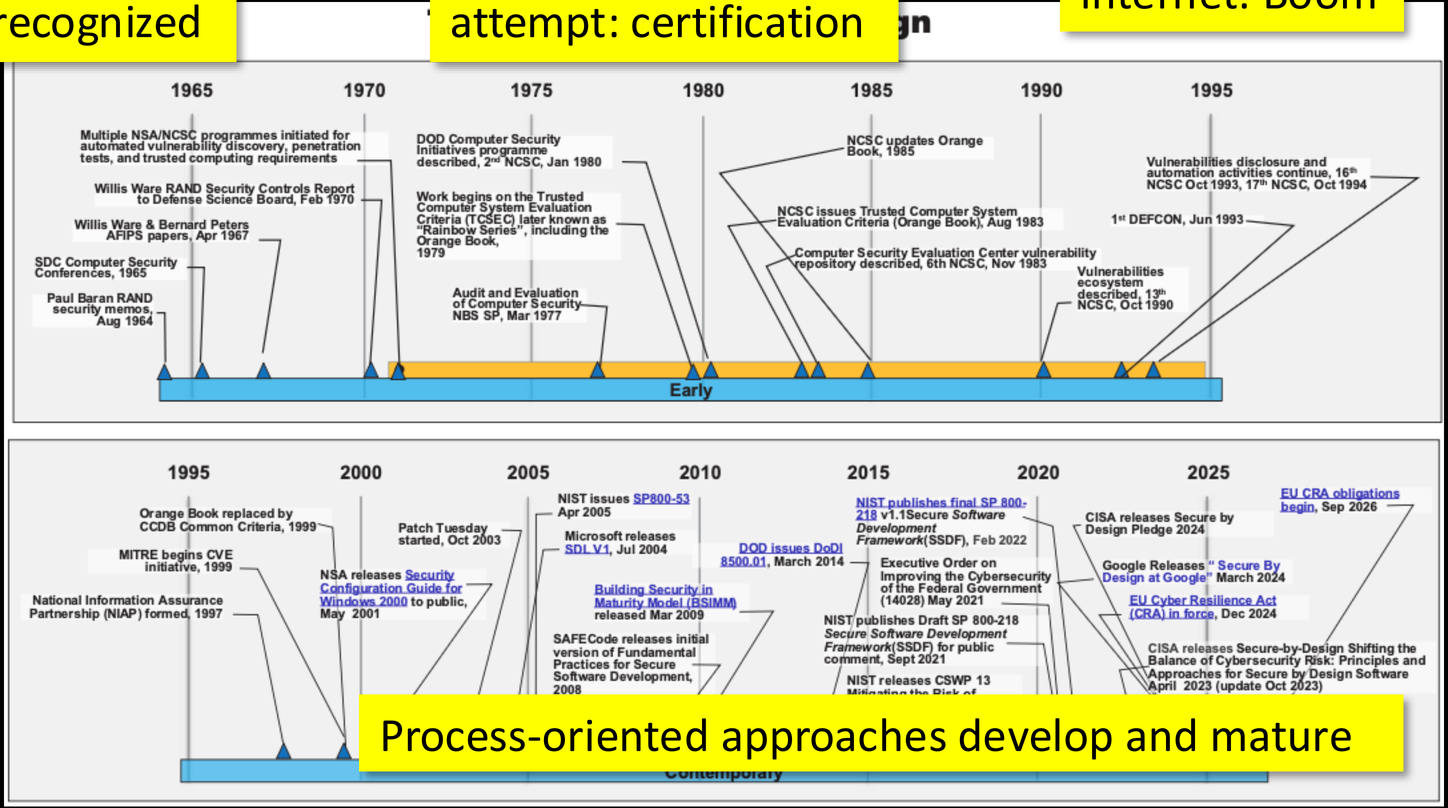
Three foundational truths:

1. **Perfect security is impossible.** The goal is to manage risk.
2. **Not all security practices are equal.** Prioritize.
3. **“Secure” is never finished.** It’s a continuous feedback loop.

The problem is recognized

Government solution attempt: certification

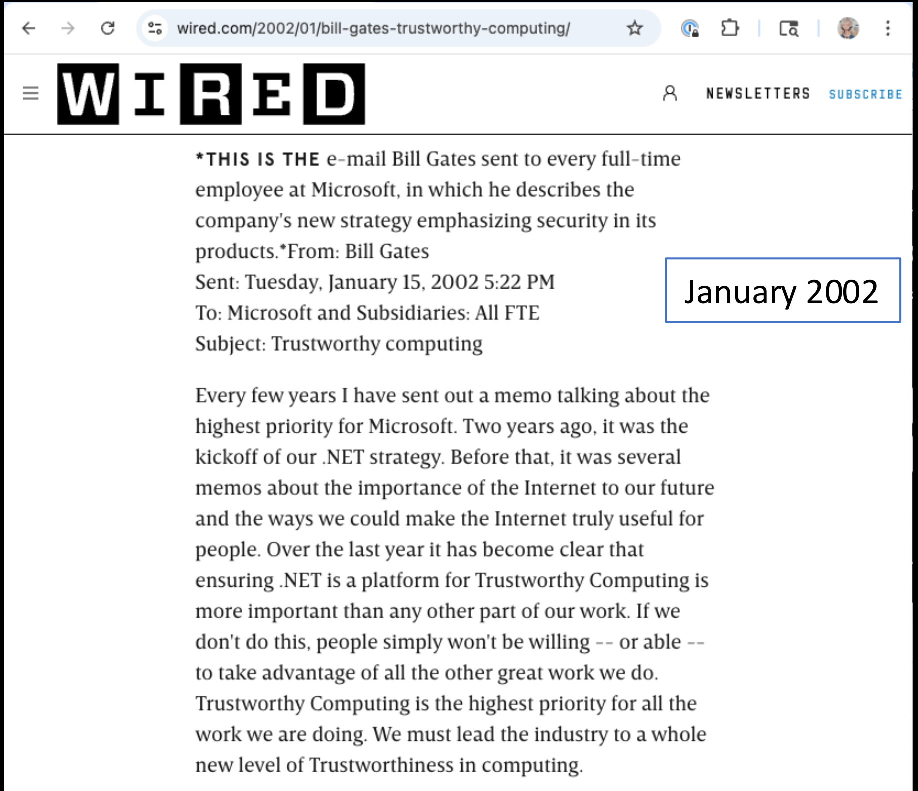
Internet: Boom



1965
1970
1975
1980
1985
1990
1995
Multiple NSA/NCSC programmes initiated for automated vulnerability discovery, penetration tests, and trusted computing requirements
Willis Ware RAND Security Controls Report to Defense Science Board, Feb 1970
Willis Ware & Bernard Peters AFIPS papers, Apr 1967
SDC Computer Security Conferences, 1965
Paul Baran RAND security memos, Aug 1964
DOD Computer Security Initiatives programme described, 2nd NCSC, Jan 1980
Work begins on the Trusted Computer System Evaluation Criteria (TCSEC) later known as "Rainbow Series", including the Orange Book, 1979
Audit and Evaluation of Computer Security NBS SP, Mar 1977
NCSC updates Orange Book, 1985
NCSC issues Trusted Computer System Evaluation Criteria (Orange Book), Aug 1983
Computer Security Evaluation Center vulnerability repository described, 6th NCSC, Nov 1983
Vulnerabilities disclosure and automation activities continue, 16th NCSC Oct 1993, 17th NCSC, Oct 1994
1st DEFCON, Jun 1993
Vulnerabilities ecosystem described, 13th NCSC, Oct 1990
Early
1995
2000
2005
2010
2015
2020
2025
Orange Book replaced by CCDB Common Criteria, 1999
MITRE begins CVE initiative, 1999
National Information Assurance Partnership (NIAP) formed, 1997
Patch Tuesday started, Oct 2003
NSA releases Security Configuration Guide for Windows 2000 to public, May 2001
NIST issues SP800-53 Apr 2005
Microsoft releases SDL V1, Jul 2004
Building Security in Maturity Model (BSIMM) released Mar 2009
SAFECode releases initial version of Fundamental Practices for Secure Software Development, 2008
DOD issues DoDI 8500.01, March 2014
NIST publishes final SP 800-218 v1.1Secure Software Development Framework(SSDF), Feb 2022
Executive Order on Improving the Cybersecurity of the Federal Government (14028) May 2021
NIST publishes Draft SP 800-218 Secure Software Development Framework(SSDF) for public comment, Sept 2021
NIST releases CSWP 13
CISA releases Secure by Design Pledge 2024
Google Releases " Secure By Design at Google" March 2024
EU Cyber Resilience Act (CRA) in force, Dec 2024
EU CRA obligations begin, Sep 2026
CISA releases Secure-by-Design Shifting the Balance of Cybersecurity Risk: Principles and Approaches for Secure by Design Software April 2023 (update Oct 2023)

Process-oriented approaches develop and mature

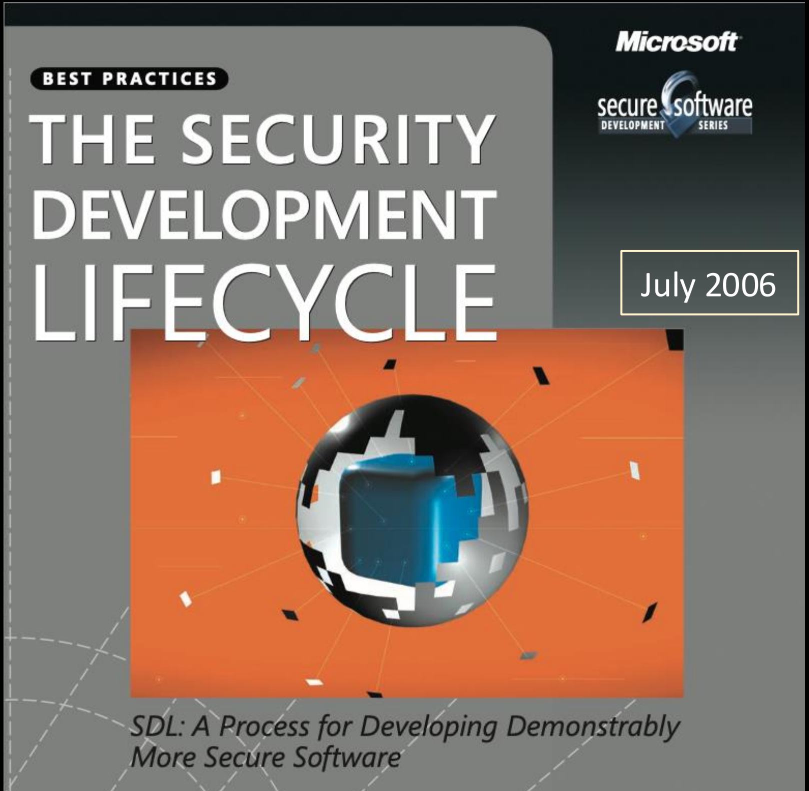
Microsoft was a target, then became a leader



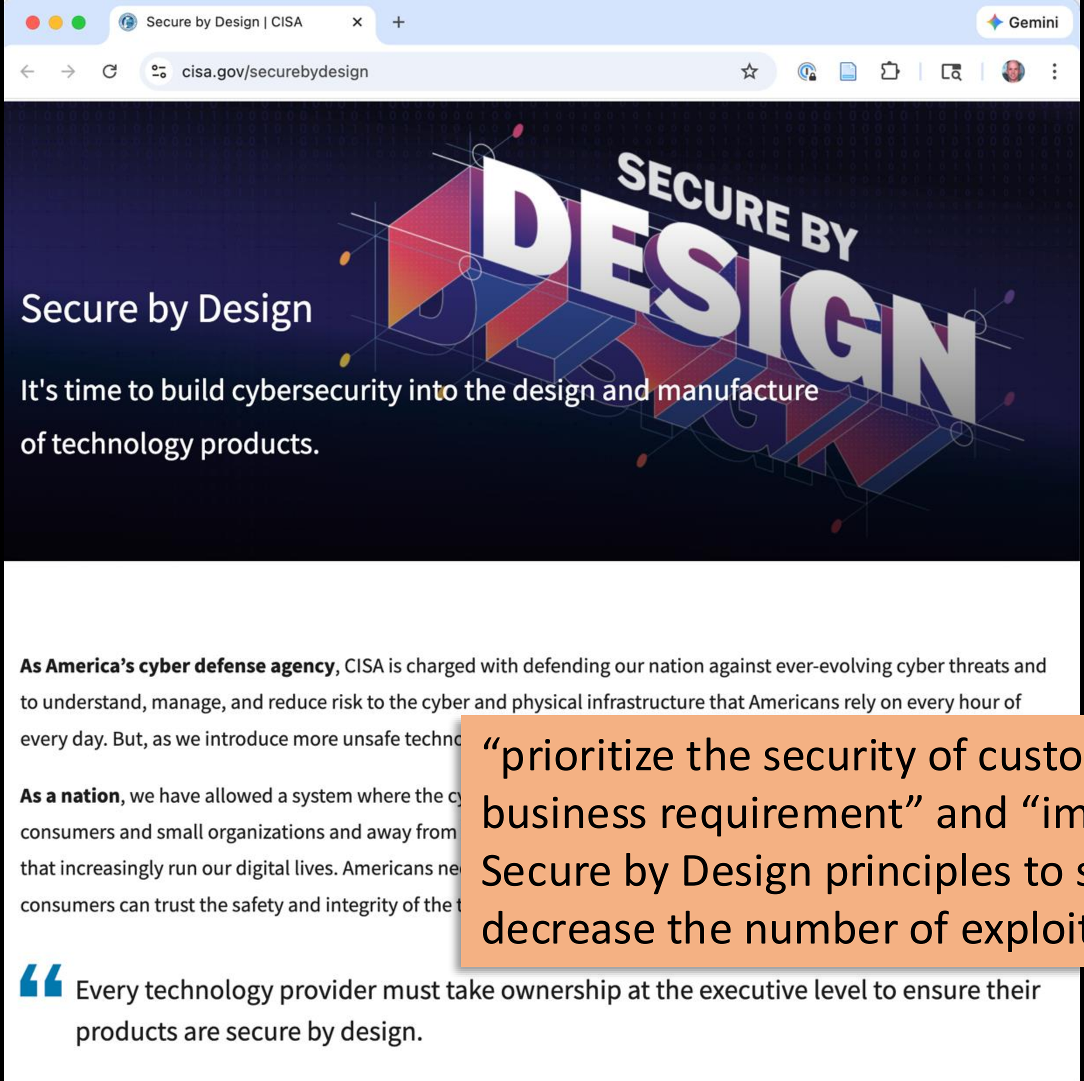
*THIS IS THE e-mail Bill Gates sent to every full-time employee at Microsoft, in which he describes the company's new strategy emphasizing security in its products.*From: Bill Gates
Sent: Tuesday, January 15, 2002 5:22 PM
To: Microsoft and Subsidiaries: All FTE
Subject: Trustworthy computing

Every few years I have sent out a memo talking about the highest priority for Microsoft. Two years ago, it was the kickoff of our .NET strategy. Before that, it was several memos about the importance of the Internet to our future and the ways we could make the Internet truly useful for people. Over the last year it has become clear that ensuring .NET is a platform for Trustworthy Computing is more important than any other part of our work. If we don't do this, people simply won't be willing -- or able -- to take advantage of all the other great work we do. Trustworthy Computing is the highest priority for all the work we are doing. We must lead the industry to a whole new level of Trustworthiness in computing.

January 2002


BEST PRACTICES
THE SECURITY DEVELOPMENT LIFECYCLE
Microsoft
secure software DEVELOPMENT SERIES
SDL: A Process for Developing Demonstrably More Secure Software

July 2006



Secure by Design

It's time to build cybersecurity into the design and manufacture of technology products.

As America's cyber defense agency, CISA is charged with defending our nation against ever-evolving cyber threats and to understand, manage, and reduce risk to the cyber and physical infrastructure that Americans rely on every hour of every day. But, as we introduce more unsafe techno

As a nation, we have allowed a system where the cy
consumers and small organizations and away from
that increasingly run our digital lives. Americans ne
consumers can trust the safety and integrity of the t

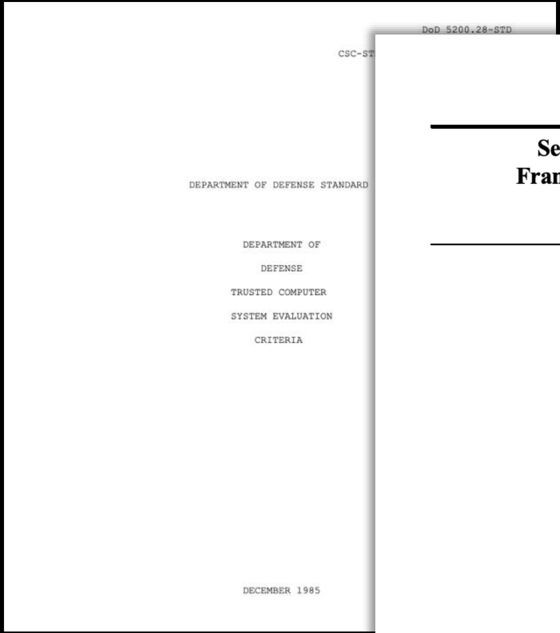
“ Every technology provider must take ownership at the executive level to ensure their products are secure by design.

2023

“prioritize the security of customers as a core business requirement” and “implement Secure by Design principles to significantly decrease the number of exploitable flaws.”

Lessons from History

- Principles understood since the 1960s-70s
- Early approaches (Orange Book) were too rigid to scale
- The Internet era outpaced security processes
  - Microsoft pivoted to catch up
- Modern frameworks (SSDF, CIS Controls) aim for the **sweet spot**: practical, evaluable, adaptable



DoD 5200.28-STD

CSC-ST

DEPARTMENT OF DEFENSE STANDARD

DEPARTMENT OF

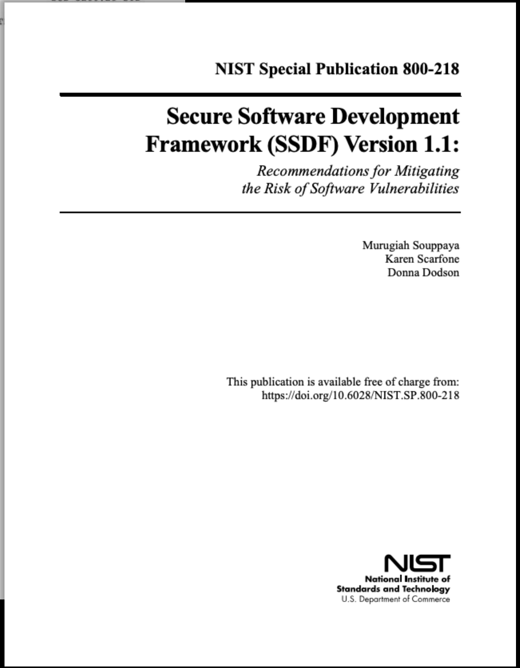
DEFENSE

TRUSTED COMPUTER

SYSTEM EVALUATION

CRITERIA

DECEMBER 1985



NIST Special Publication 800-218

Secure Software Development Framework (SSDF) Version 1.1:

Recommendations for Mitigating the Risk of Software Vulnerabilities

Murugiah Souppaya
Karen Scarfone
Donna Dodson

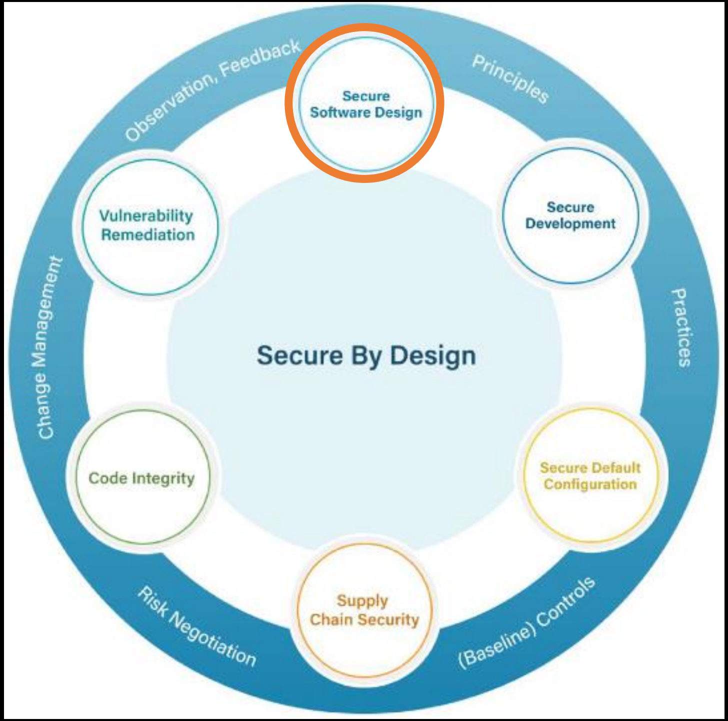
This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-218

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

The CIS/SAFECode paper builds on the NIST SSDF to create **independent, practical, and actionable guidance** – bridging broad principles and concrete practices.

Six Considerations: Overview

1. Secure Software Design
2. Secure Development
3. Secure Default Configuration



4. Supply Chain Security
5. Code Integrity
6. Vulnerability Remediation

Consideration 1: **Secure Software Design**

Secure Software Design
Principles
Secure Development
Practices
Secure Default Configuration
(Baseline) Controls
Supply Chain Security
Risk Negotiation
Code Integrity
Change Management
Vulnerability Remediation
Observation, Feedback
Secure By Design

Design begins with **security objectives**:

Requirement	Definition
Confidentiality	Protect information from unauthorized disclosure
Integrity	Protect information from unauthorized modification
Availability	Provide access when required
Accountability	Non-repudiation via permanent audit records

Secure Design: Security Features

These objectives are typically achieved through:

Feature	Intent
Authentication	Reliably identifying users (passwords, MFA, SSO)
Authorization	Granting access based on roles and rights
Auditing	Creating reliable, immutable records of actions

"The gold standard"

These must be **correctly implemented** and **impossible to bypass**.

DRAFT

date October 3, 1974

THE PROTECTION OF INFORMATION IN COMPUTER SYSTEMS

by

Jerome H. Saltzer and Michael D. Schroeder

Abstract--This tutorial paper explores the mechanics of protecting information stored in computer systems. It begins by discussing desired functions, design principles, and examples of elementary protection and user authentication mechanisms. The second major part examines in depth the principles of modern descriptor-based protection architectures and the relation between capability systems and access-control-list systems. This discussion ends with a brief analysis of protected subsystems and protected objects. The last part of the paper reviews the state of the art and current research problems and also provides suggestions for further reading.

The authors are with Project MAC and the Computer Science section of the Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, 02139

Secure Design: Saltzer & Schroeder's Principles

Classic design principles (1975) that remain essential:

- **Least privilege:** Grant minimum access needed for the task
- **Fail-safe defaults:** If in doubt, *deny* access
- **Complete mediation:** Check *every* access to *every* object
- **Economy of mechanism:** Keep security-critical code simple
- **Separation of privilege:** Require multiple conditions for access

Secure Design: The Confused Deputy

The **confused deputy problem**: a program with elevated privileges is tricked into misusing them on an attacker's behalf.

CMS example: The server-side PDF renderer runs with system privileges to read uploaded files. A malicious "assignment" triggers it to read `/etc/passwd` instead.

Confused deputies appear everywhere: CSRF, SSRF, insecure deserialization, …

Secure Design Requires Threat Modeling

"Secure system design teams in industry frequently rely on a process called threat modeling to review their designs."
– CIS/SAFECode, *Secure by Design*

- Analyze data flows to identify where an attacker might strike
- Identify countermeasures for each threat
- Supported by frameworks, tools, and training

We'll do this for our CMS in the second half of today's lecture.

Consideration 2: **Secure Development**

Secure By Design
Secure Software Design
Secure Development
Secure Default Configuration
Supply Chain Security
Code Integrity
Vulnerability Remediation
Principles
Practices
(Baseline) Controls
Risk Negotiation
Change Management
Observation, Feedback

The process of writing, testing, and maintaining code securely.

A secure development process must include:

- A **“bug bar”**: which security bugs block release?
- **Coding standards**: memory-safe languages, approved libraries, safe patterns
- **Security tooling**: static analysis, dynamic analysis, dependency scanning
- **Code review** focused on security-critical paths
- **Security testing** beyond functional tests (regression and pen testing)
- **Developer training** on secure coding

The Secure Development Toolchain

Tool Class	What It Does	When
SAST (Static Analysis)	Scans source code for vulnerability patterns	During coding & CI
SCA (Software Composition Analysis)	Checks dependencies for known CVEs	Build time
DAST (Dynamic Analysis)	Tests running application for vulnerabilities	Testing/staging
Fuzzing	Feeds unexpected inputs to find crashes	Testing
Code Review	Human analysis of security-critical code	Before merge

Consideration 3: **Secure Default Config**

Secure By Design
Secure Software Design
Secure Development
Secure Default Configuration
Supply Chain Security
Code Integrity
Vulnerability Remediation
Observation, Feedback
Principles
Practices
(Baseline) Controls
Risk Negotiation
Change Management

Ship it **locked down**. Users opt *out* of security, not *in*.

The 80% rule: If a feature, privilege, or interface is not needed by 80% of users, **disable it by default**.

CMS examples:

- New courses: all content **private** by default
- File uploads: restricted file types (.pdf, .docx – not .exe, .html)
- API access: **disabled** by default
- Admin panel: not accessible from public internet
- Session timeout: reasonable default (not “forever”)

Consideration 4: **Supply Chain Security**

Secure By Design
Secure Software Design
Secure Development
Secure Default Configuration
Supply Chain Security
Code Integrity
Vulnerability Remediation
Principles
Practices
(Baseline) Controls
Risk Negotiation
Change Management
Observation, Feedback

Modern software is **mostly code you didn't write**.

Each dependency is a **trust decision**:

- Is it actively maintained?
- Has it been audited for security?
- What happens when a vulnerability is found in it?

Supply Chain: Log4Shell as Case Study

December 2021: CVE-2021-44228 in Apache Log4j

- A logging library used by thousands of Java applications
- Remote code execution via a simple log message
- Many organizations didn't know they were using it

Key lesson: You must know what's in your software (**SBOM**) so you can respond when a dependency is compromised.

Consideration 5: **Code Integrity**

Secure By Design
Secure Software Design
Secure Development
Secure Default Configuration
Supply Chain Security
Code Integrity
Vulnerability Remediation
Observation, Feedback
Principles
Practices
(Baseline) Controls
Risk Negotiation
Change Management

Ensuring the software delivered to users is **what you intended to build**.

Threat: Malicious code inserted during development or delivery

Real-world attacks:

- **SolarWinds (2020):** Build system compromised; malicious update sent to 18,000 organizations
- **XZ Utils (2024):** Backdoor inserted by a trusted maintainer into a core Linux library
- **CodeCov (2021):** CI/CD tool compromised; credentials stolen from thousands of repos

Consideration 6: Vulnerability Remediation

Secure By Design
Secure Software Design
Secure Development
Secure Default Configuration
Supply Chain Security
Code Integrity
Vulnerability Remediation
Principles
Practices
(Baseline) Controls
Risk Negotiation
Change Management
Observation, Feedback

No software ships bug-free. The question is: **what happens when bugs are found?**

Three phases:

1. **Immediate remediation:** Fix the reported vulnerability, ship a patch
2. **Variant analysis:** Search for *similar* vulnerabilities elsewhere in the codebase
3. **Process improvement:** Update tools, training, and practices to prevent recurrence

The Remediation Learning Loop

Every vulnerability is **data about your process**.

- Repeated XSS findings → rethink your templating approach
- Repeated authz bugs → redesign your authorization model
- Repeated dependency vulns → improve your vetting process

AI and Secure Software Development

Four intersections of AI/ML with software security:

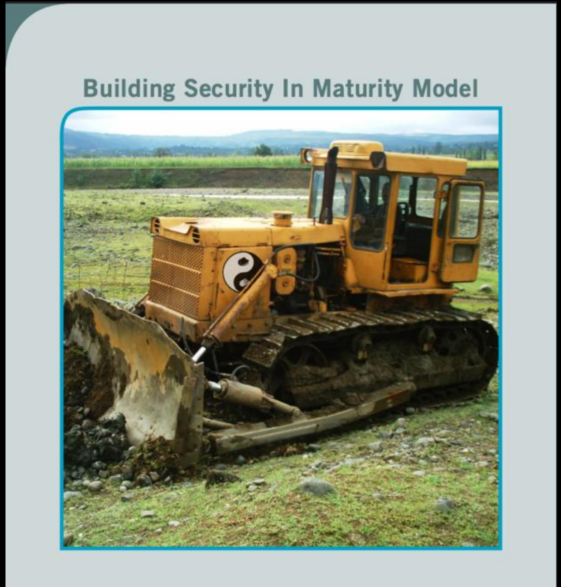
#	Intersection	Implication
1	AI components *in* your software	Their security must be assured
2	AI tools used to *write* code	AI-generated code needs the same security scrutiny
3	AI tools to *find* vulnerabilities	Emerging aid for threat modeling and analysis
4	Adversaries using AI to *attack*	Offensive capabilities evolving rapidly

It’s Not Just Technical – It’s Organizational

“An organizational culture that prioritizes security must underlie implementation of engineering and operational practices for SbD.”
– CIS/SAFECode, *Secure by Design*

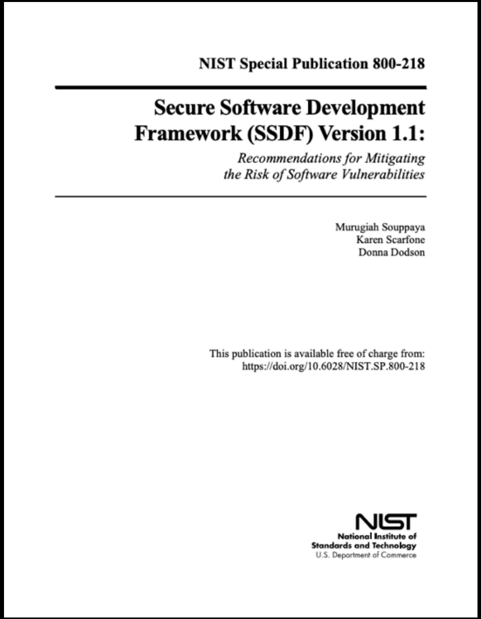
Security by Design requires:

- **Management commitment** and budget
- **Defined roles** and responsibilities
- **Training** for all development staff
- **Security champions** embedded in teams
- **Artifacts** produced naturally by the process



The NIST SSDF: Four Practice Groups

Practice Group	Focus
Prepare the Organization (PO)	Define process, assign roles, train people, choose tools
Protect the Software (PS)	Access control on code, code signing, release integrity
Produce Well-Secured Software (PW)	Design, code, test, and configure securely
Respond to Vulnerabilities (RV)	Identify, assess, remediate, and learn



NIST Special Publication 800-218

Secure Software Development Framework (SSDF) Version 1.1:

Recommendations for Mitigating the Risk of Software Vulnerabilities

Murugiah Souppaya
Karen Scarfone
Donna Dodson

This publication is available free of charge from:
https://doi.org/10.6028/NIST.SP.800-218

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Assessment: How Do You Know It’s Working?

“Secure by Design” doesn’t mean “vulnerability-free.”

It means the organization has **addressed the six considerations** with evidence.

Evidence comes from **artifacts** naturally produced by development:

- Threat models and design documents
- Code review records and tool outputs
- Bug tracking history (including security bugs)
- Test results (including security tests)
- Vulnerability response records

“Evidence created solely to satisfy an assessor should be considered with great suspicion.”

How Does Your Experience Compare?

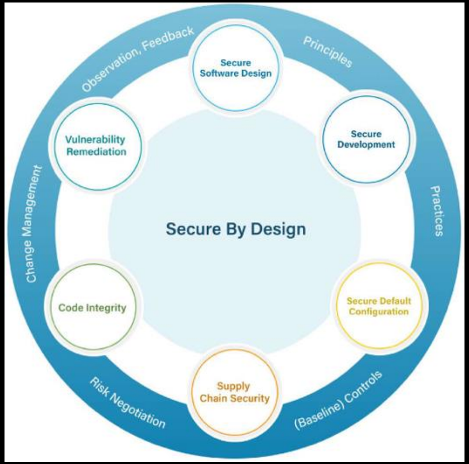
Think about a software project you've worked on:

- A class project, internship, open source contribution, personal project

Which of the six considerations did you address?

1. Secure Design
2. Secure Development
3. Secure Default Configuration
4. Supply Chain Security
5. Code Integrity
6. Vulnerability Remediation

Which did you completely ignore?



Putting It All Together

We’ve seen that secure software requires:

- **Secure Design:** Architecture that resists attack (CIA, access control, least privilege)
- **Secure Development:** Practices and tools throughout the coding process
- **Secure Defaults:** Ship locked down; minimize attack surface
- **Supply Chain Security:** Know and vet what you depend on
- **Code Integrity:** Protect the development and delivery pipeline
- **Vulnerability Remediation:** Find, fix, and learn from bugs continuously

All of this rests on **organizational commitment** and the SSDF framework.

Further Reading

Assigned:

- CIS/SAFECode, *Secure by Design: A Guide to Assessing Software Security Practices* (2025)
  - **See also:** Appendix A of the paper for a comprehensive resource list.

Foundational:

- Saltzer & Schroeder, “The Protection of Information in Computer Systems” (1975)
- NIST SP 800-218, *Secure Software Development Framework*

Practical:

- Google, “Secure by Design at Google” (2024)
- CISA Secure by Design resources