Foundations of Software Engineering

Security Development Lifecycles

Christian Kästner

(Based on slides by Michael Maass)

Administrativa

- No class Nov 20,
  but HW6a deadline on Nov 20
- HW 5 extended until Nov 13
- In-class interview Thursday

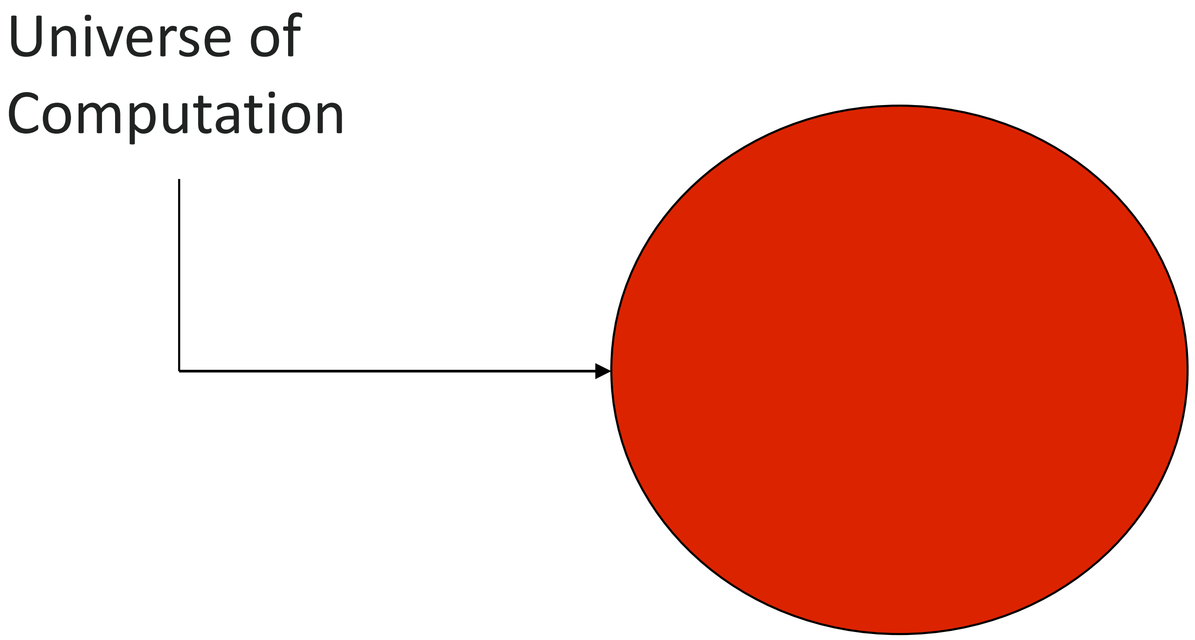
Learning goals

- Understand basic concepts of vulnerabilities and secure software
- Implement security mechanisms across the entire software development lifecycle
- Design and inspect architecture for security with threat modeling
- Decide how do adopt security practices and educate participants. Who, when, and how much?

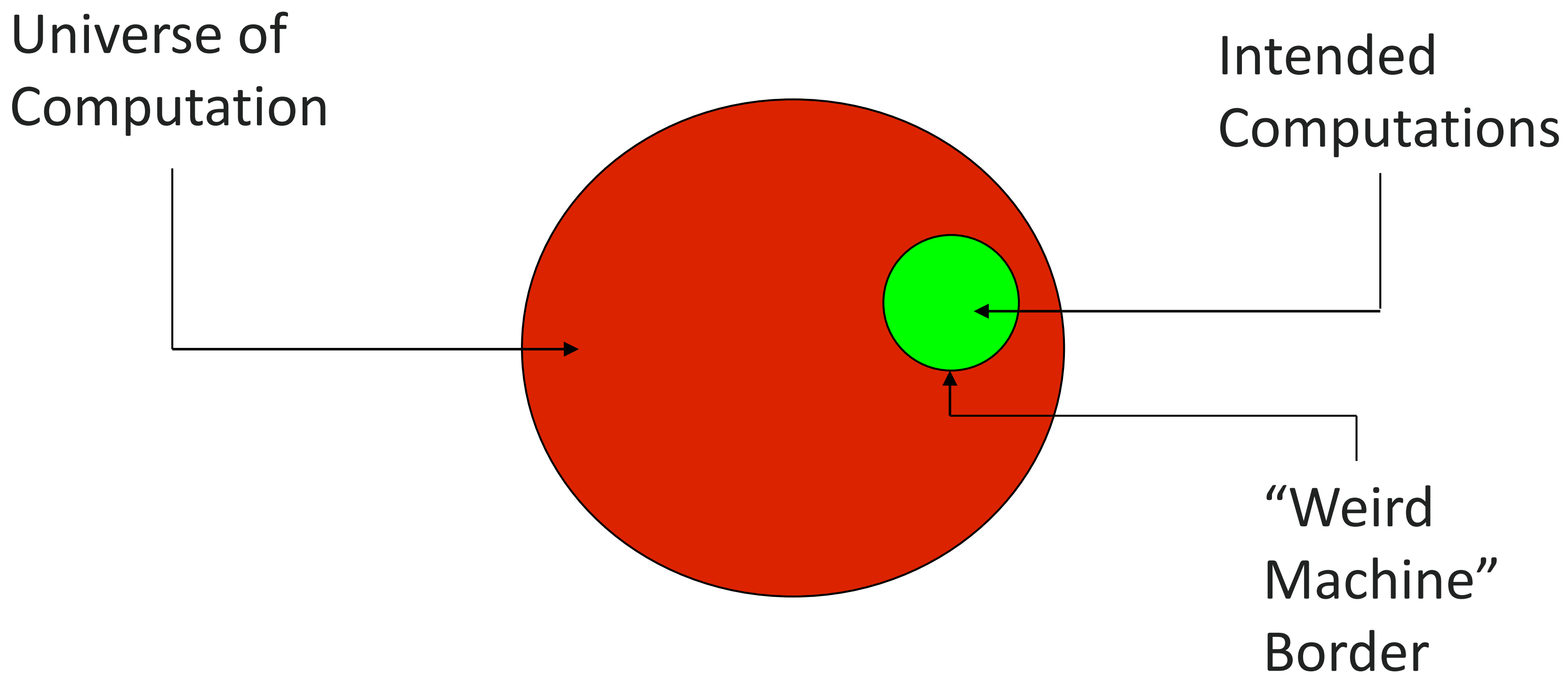
Vulnerability

- A **vulnerability** is a set of conditions that allows an attacker to violate an explicit or implicit security policy
  - Not all software security flaws lead to vulnerabilities. Vulnerabilities require an avenue of attack known as an attack vector
- A software security flaw can cause a program to be vulnerable to attack
  - Software security flaws that do not result in vulnerabilities should still be corrected so that they do not propagate

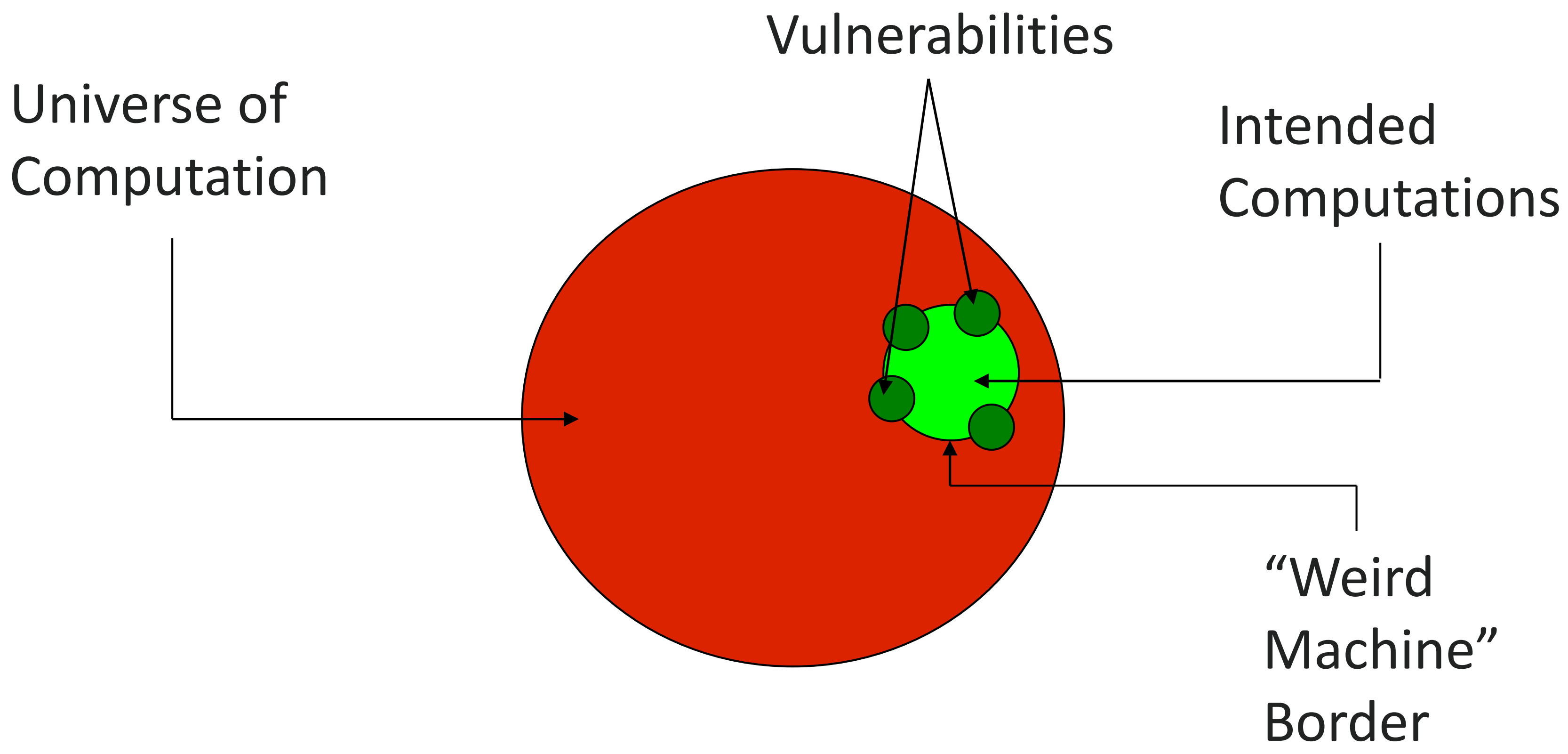
Attack Surfaces



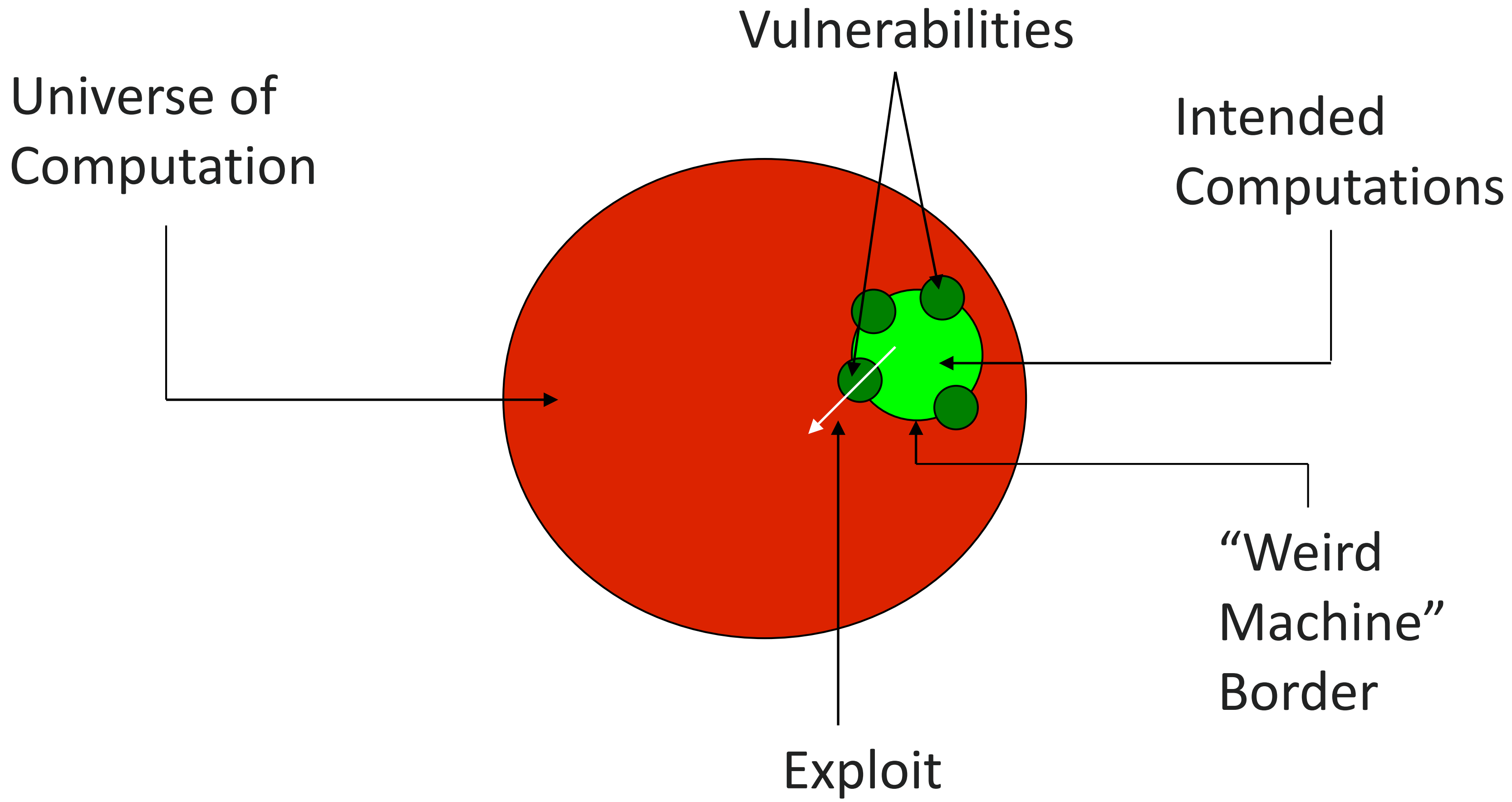
Attack Surfaces



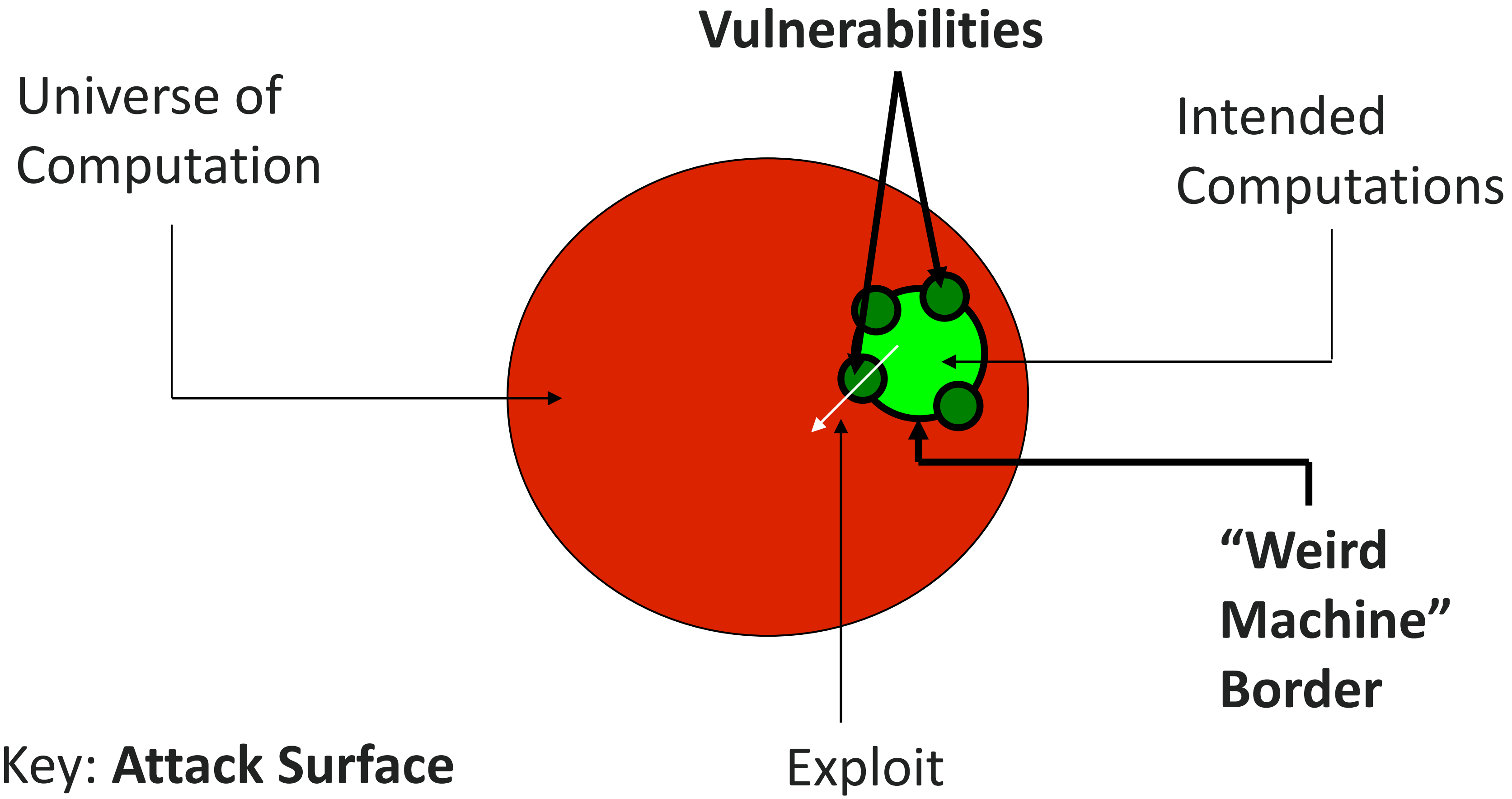
Attack Surfaces



Attack Surfaces



Attack Surfaces



Key: **Attack Surface**

Vulnerabilities Have Utility

- Bugs and vulnerabilities are typically accidentally introduced
- Both can cause a system to fail
- **Bugs** typically cause failures through innocent interactions
- **Bugs** often result in a loss of control with no utility
- **Vulnerabilities** cause failures through intentional and clever interactions initiated by a malicious actor
- **Vulnerabilities** give an attacker a route to seize control

An Airplane Example

- The wings fall off in violent turbulence
- Power shuts off when crossing the international date line

BUGS

- Ground control channels allow anyone to re-route active flights
- The fuel system can be trivially ordered to dump fuel at altitude

VULNERABILITIES

Vulnerabilities and Programming Errors

- 64% of the vulnerabilities in NISTs National Vulnerability Database (NVD) in 2004 are due to programming errors.
  - 51% of those due to classic errors like buffer overflows, cross-site-scripting, injection flaws
- “We wouldn't need so much network security if we didn't have such bad software security.”
  - Bruce Schneier

institute for SOFTWARE RESEARCH

Vulnerability Root-cause Categories

(non-exhaustive list)

- Logic errors
- Synchronization and timing errors
- Insecure configurations
- Protocol errors
- Cryptographic vulnerabilities
- **Input validation errors**
  - Buffer overflow
  - Integer errors
  - SQL injection

Security Issues

Security

- Confidentiality: Data is only available to the people intended to access it.
- Integrity: Data and system resources are only changed in appropriate ways by appropriate people.
- Availability: Systems are ready when needed and perform acceptably.
- Authentication: The identity of users is established (or you’re willing to accept anonymous users).
- Authorization: Users are explicitly allowed or denied access to resources.
- Nonrepudiation: Users can’t perform an action and later deny performing it.

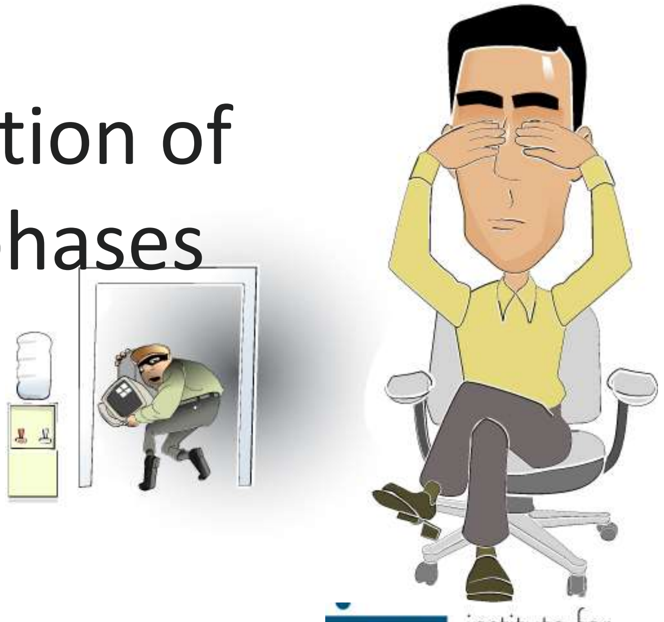
Sources of Software Insecurity

- Complexity, inadequacy, and change
- Incorrect or changing assumptions (capabilities, inputs, outputs)
- Flawed specifications and designs
- Poor implementation of software interfaces (input validation, error and exception handling)
- Inadequate knowledge of secure coding practices

institute for SOFTWARE RESEARCH

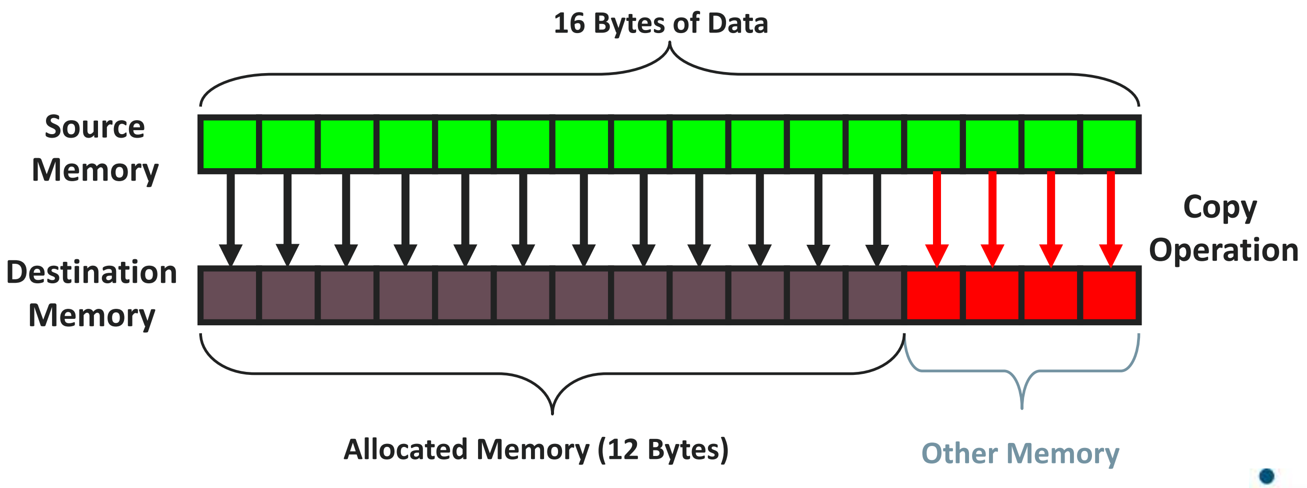
Sources of Software Insecurity - 2

- Unintended, unexpected interactions
  - with other elements
  - with the software's execution environment
- Absent or minimal consideration of security during all life cycle phases
- Not thinking like an attacker



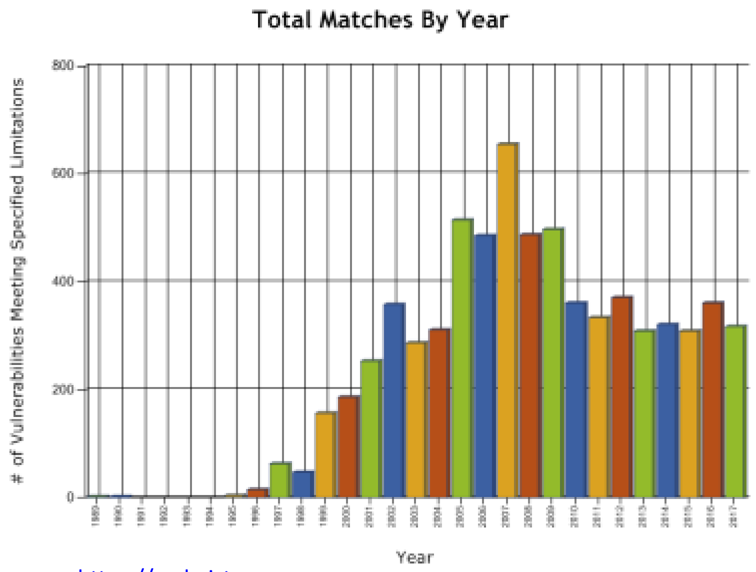
What Is a Buffer Overflow?

- A buffer overflow occurs when data is written outside of the boundaries of the memory allocated to a particular data structure



Why Buffer Overflows Matter

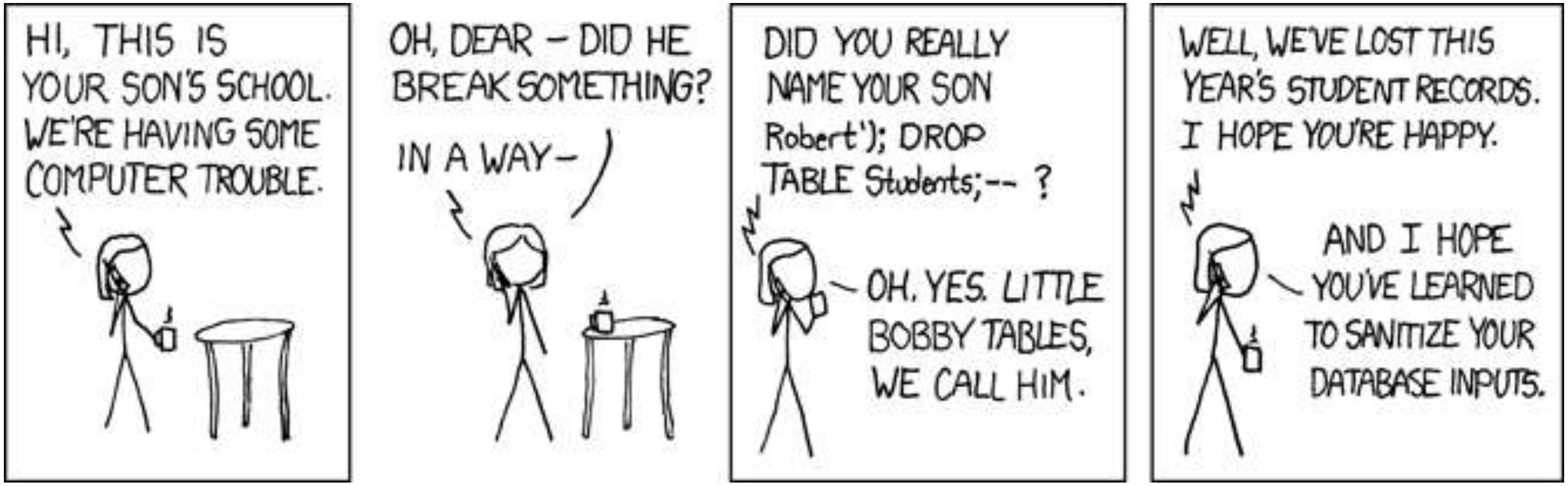
- Buffer overflows can allow an attacker to corrupt memory to execute arbitrary code
  - With the privileges of the running process
- Pervasive
  - Legacy code
  - Insecure coding practices
  - Changing environments



https://nvd.nist.gov

SOFTWARE RESEARCH

SQL Injection


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.

http://xkcd.com/327/

institute for SOFTWARE RESEARCH

EQUIFAX®

Mitigations

Mitigations

- Mitigations are methods, techniques, processes, tools, or runtime libraries that can prevent or limit exploits against vulnerabilities
  - Sometimes called a workaround
- What are some strategies to eliminate software vulnerabilities?

Two General Strategies

- Find and fix vulnerabilities in existing software
  - Reactive
  - Costly in terms of money and reputation
- Prevent vulnerabilities from occurring in new software
  - Proactive
  - Develop processes to find and eliminate vulnerabilities during software development

Strategies: Finding Vulnerabilities

- Security-focused testing
  - Fuzzing
  - Penetration testing
- Inspection/auditing
- Static analysis
- Read the news

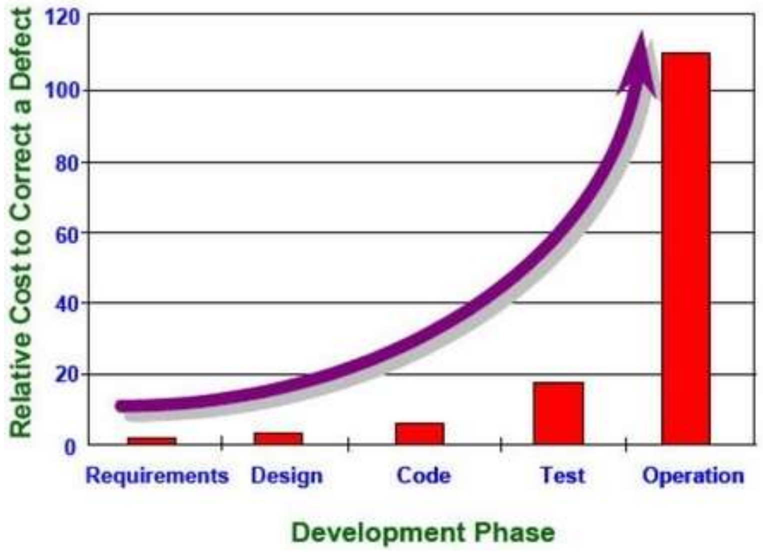
Security Development Lifecycles (SDL)

Security Development Lifecyles (SDLs) prescribe **security practices** for each phase of a software development project.

isr institute for SOFTWARE RESEARCH

Security Practice Goals

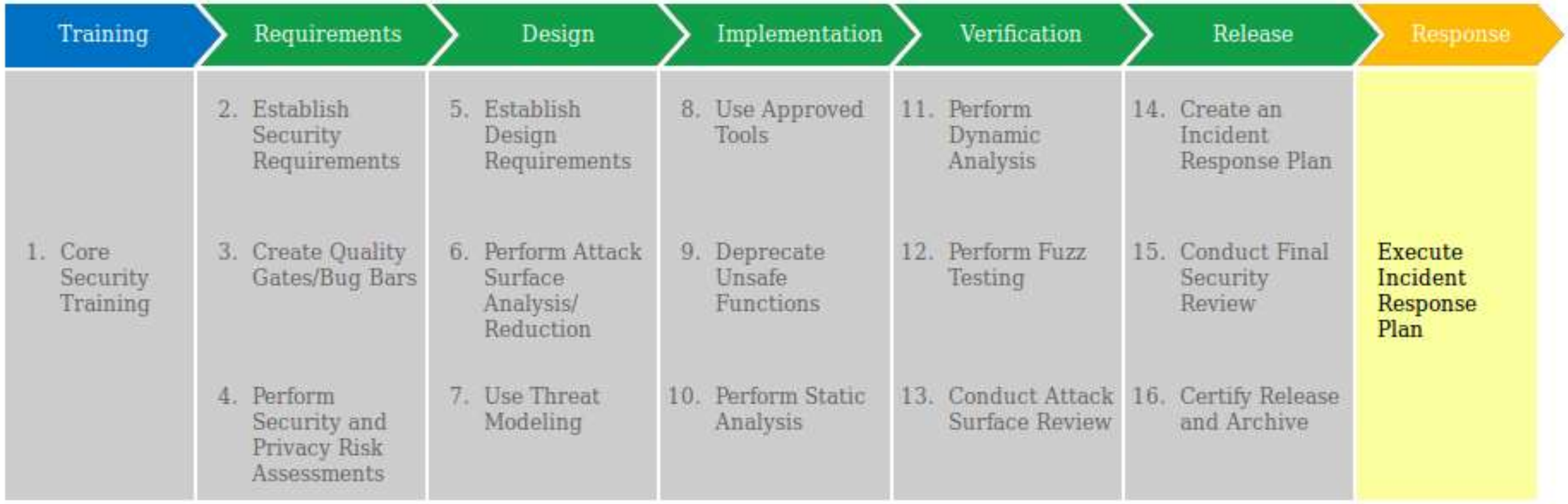
- Find vulnerabilities early
- Identify risks and mitigate them
- Reduce attack surface
- Prepare to fix future vulnerabilities quickly
- Gain confidence that the system is secure
- **Build security in!**



Microsoft Trustworthy Computing Initiative (2002)

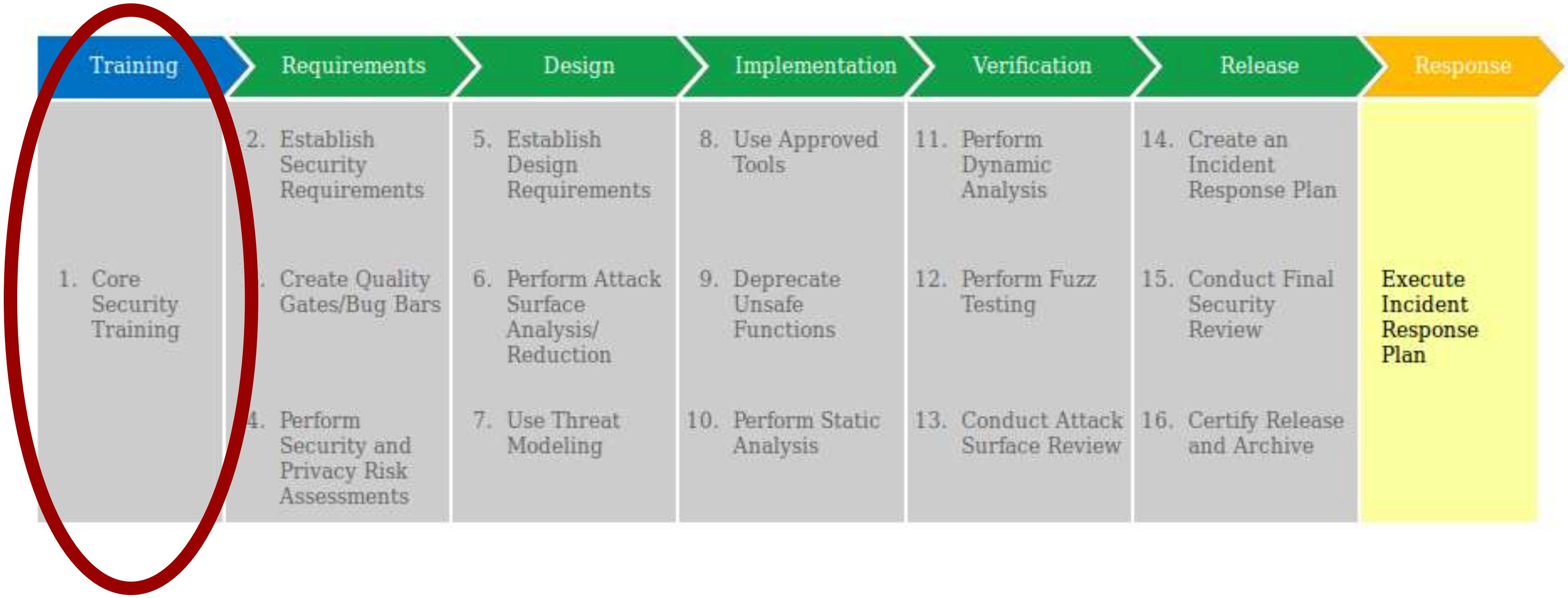
- see memo

Microsoft SDLs



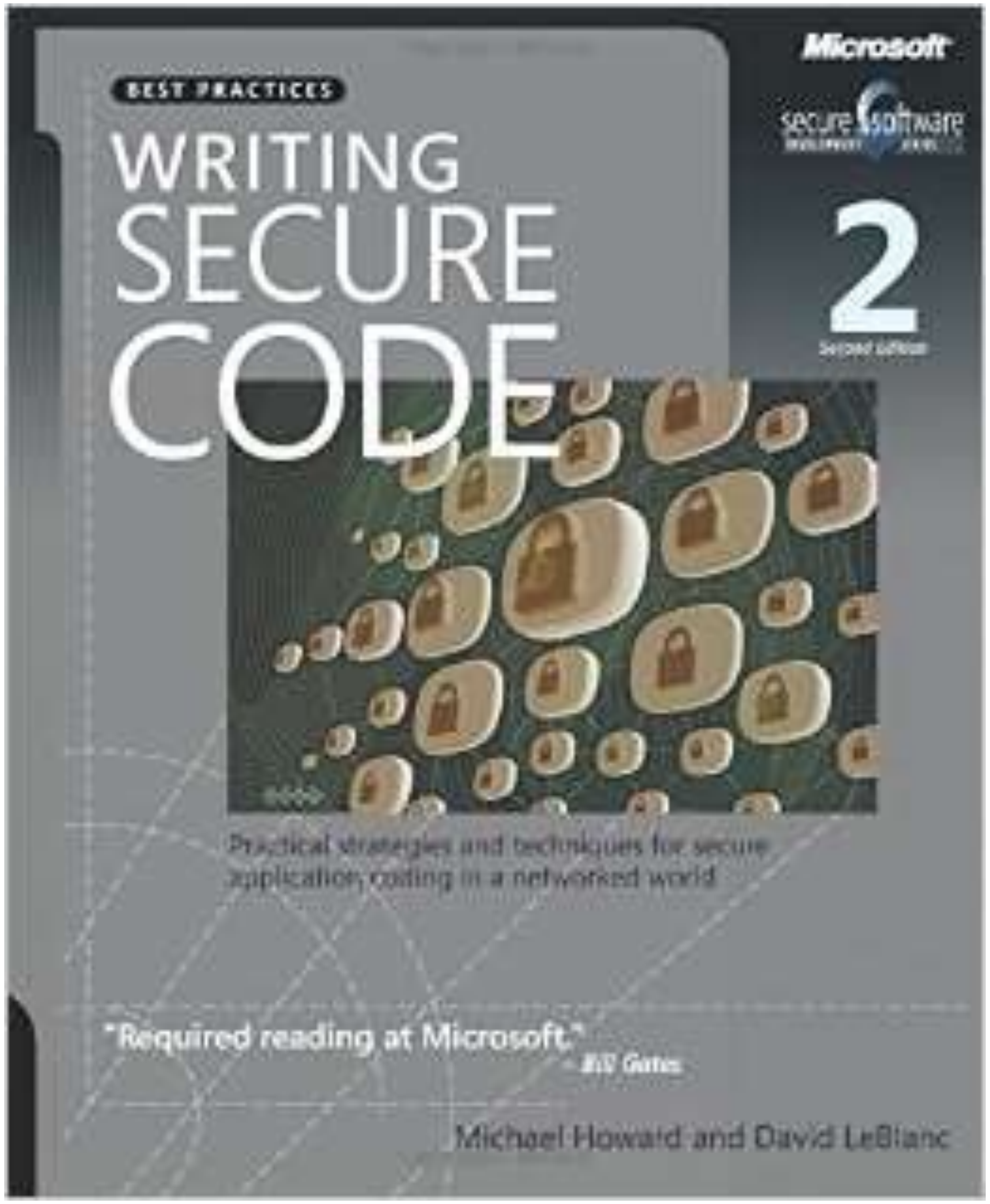
Training	Requirements	Design	Implementation	Verification	Release	Response
	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	
1. Core Security Training	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	Execute Incident Response Plan
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

Microsoft SDLs



Training	Requirements	Design	Implementation	Verification	Release	Response
	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	
1. Core Security Training	. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	**Execute Incident Response Plan**
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

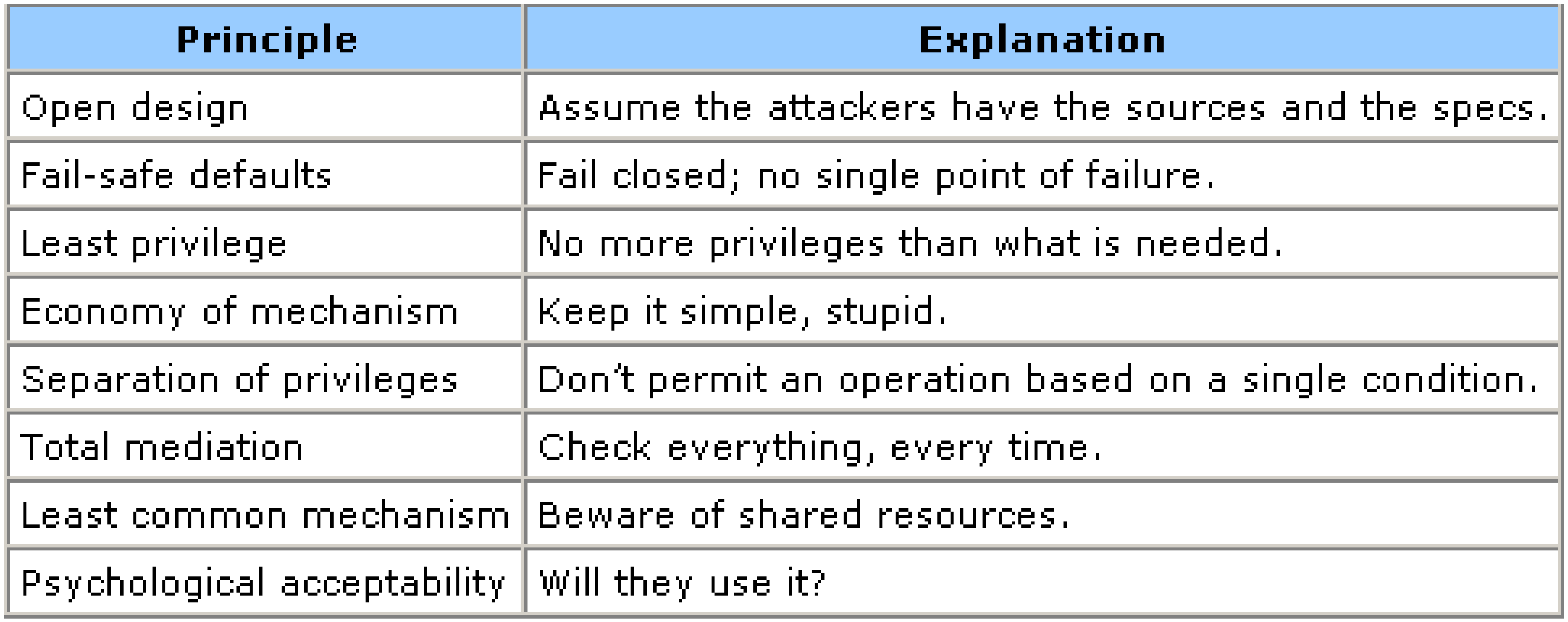
institute for SOFTWARE RESEARCH


Microsoft
secure software
2
Second edition
BEST PRACTICES
WRITING SECURE CODE
Practical strategies and techniques for secure application coding in a networked world
"Required reading at Microsoft." – Bill Gates
Michael Howard and David LeBlanc

CERT: Secure Coding Standards

- https://www.securecoding.cert.org/

(Academic) Design Principles



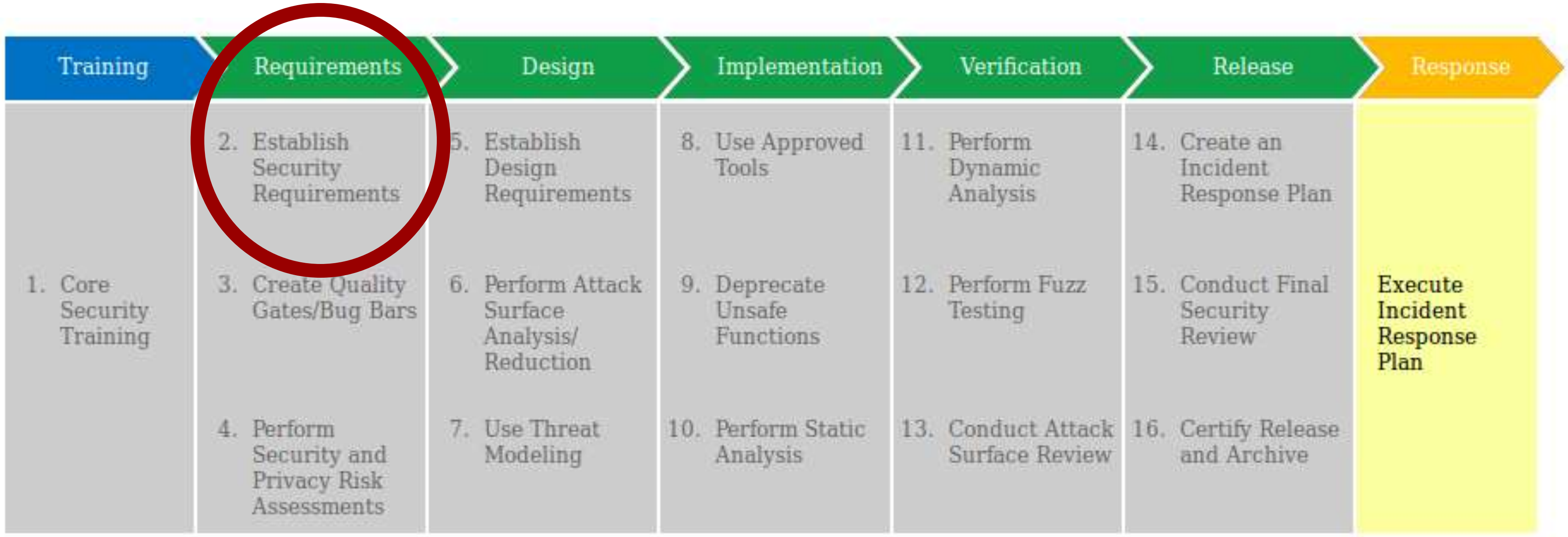
Principle	Explanation
Open design	Assume the attackers have the sources and the specs.
Fail-safe defaults	Fail closed; no single point of failure.
Least privilege	No more privileges than what is needed.
Economy of mechanism	Keep it simple, stupid.
Separation of privileges	Don't permit an operation based on a single condition.
Total mediation	Check everything, every time.
Least common mechanism	Beware of shared resources.
Psychological acceptability	Will they use it?

Saltzer and Schroeder's design principles

“8 Simple Rules for Developing More Secure Code” (M. Howard, MSDN Magazine Nov 2006)

1. Take Responsibility
2. Never Trust Data
3. Model Threats against Your Code
4. Stay One Step Ahead
5. Fuzz!
6. Don’t Write Insecure Code
7. Recognize the Strategic Asymmetry
8. Use the Best Tools You Can

Microsoft SDLs



Training	Requirements	Design	Implementation	Verification	Release	Response
	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	
1. Core Security Training	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	Execute Incident Response Plan
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

isr institute for SOFTWARE RESEARCH

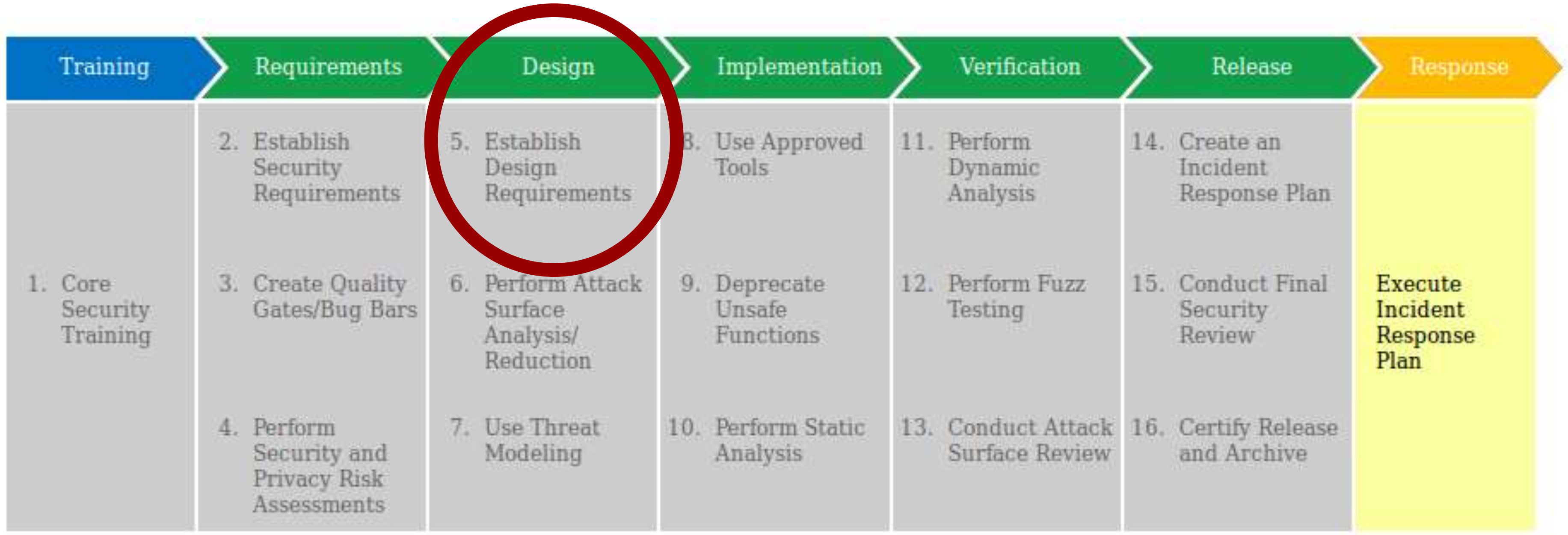
Security Requirements

- Security requirements are as important as any other requirement category
- Must include individuals with security expertise
- Deploy vulnerability tracking system
  - Can be the same as the bug tracker for most projects

Example

- “The application shall provide passwords, smart cards, and one-time passwords to support user authentication.”
- “The mechanisms for performing cryptographic operations shall be easily replaceable at runtime.”

Microsoft SDLs

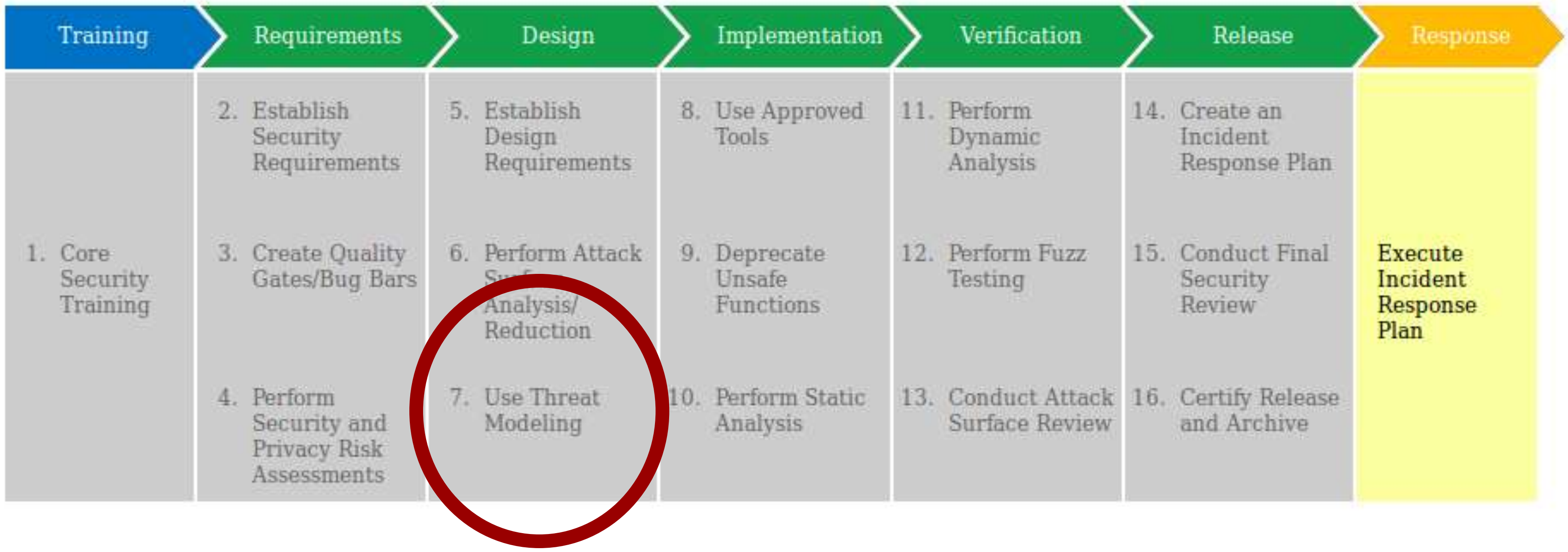


Training	Requirements	Design	Implementation	Verification	Release	Response
	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	
1. Core Security Training	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	Execute Incident Response Plan
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

Certify Security Requirements in Design

- Traceability from security requirements to design (and implementation)
- Inspection of design
- Involve security experts

Microsoft SDLs



Training	Requirements	Design	Implementation	Verification	Release	Response
	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	
1. Core Security Training	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	Execute Incident Response Plan
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

Threat Modeling

- A structured approach to find threat scenarios that apply to a product
- Typically:
  - Create a data flow diagram showing system components and the data flowing between them (requires some expertise in deciding what to model)
  - Apply the STRIDE threat model at each data flow to enumerate threats

STRIDE

- **Spoofing** – can an actor use someone else's data as their own or trick the system into using fake data?
- **Tampering** – is malicious modification of data possible?
- **Repudiation** – can an actor claim they didn't perform an action or easily make it look like someone else did it?
- **Information Disclosure** – is an actor given private or sensitive information they don't need?
- **Denial of Service** – can an actor prevent valid users from using the system?
- **Elevation of Privilege** – can an actor gain higher privileges than they should have?

Inspection per component

STRIDE vs Security Properties

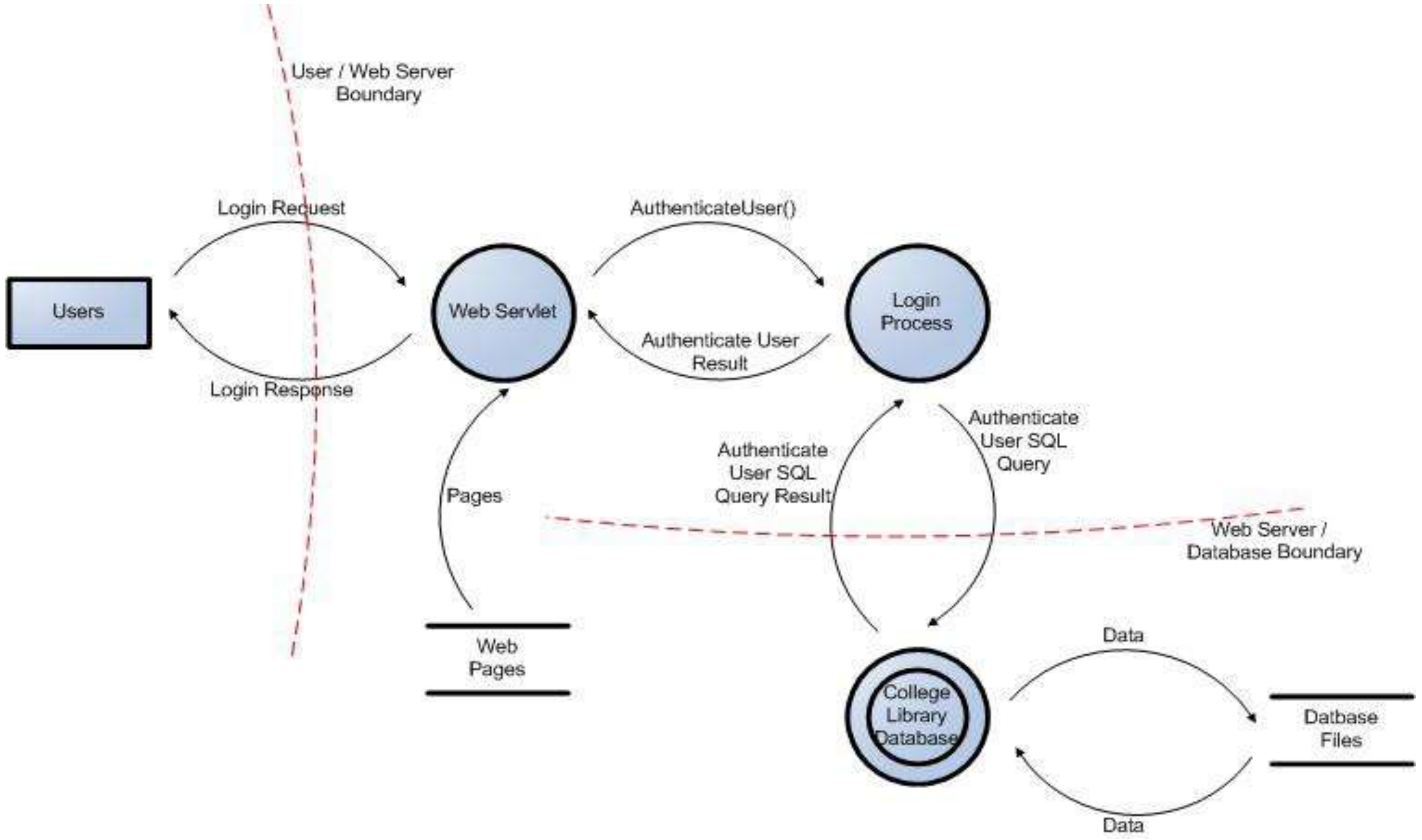
Threat	Security Property
Spoofing	Authentication
Tampering	Integrity
Repudiation	Non-repudiation
Information disclosure	Confidentiality
Denial of service	Availability
Elevation of privilege	Authorization

STRIDE process

- Identify relevant components and data flows
- Analyze each component for each threat
- Mitigate threats

- -> Gain confidence (no proof)

Data Flow Diagram



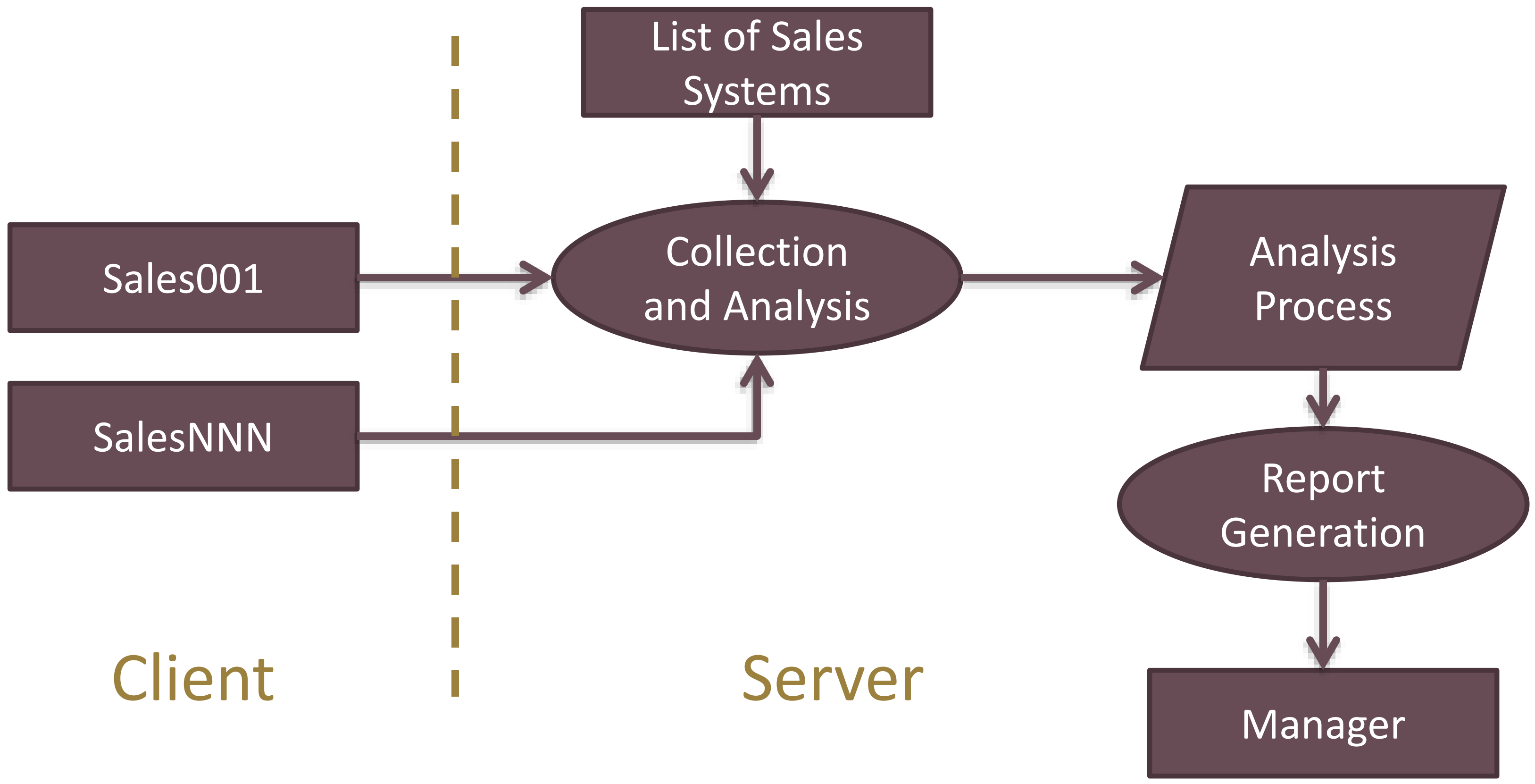
Data flows, data stores, processes, interactors, and trust boundaries

Use case: Sales entry

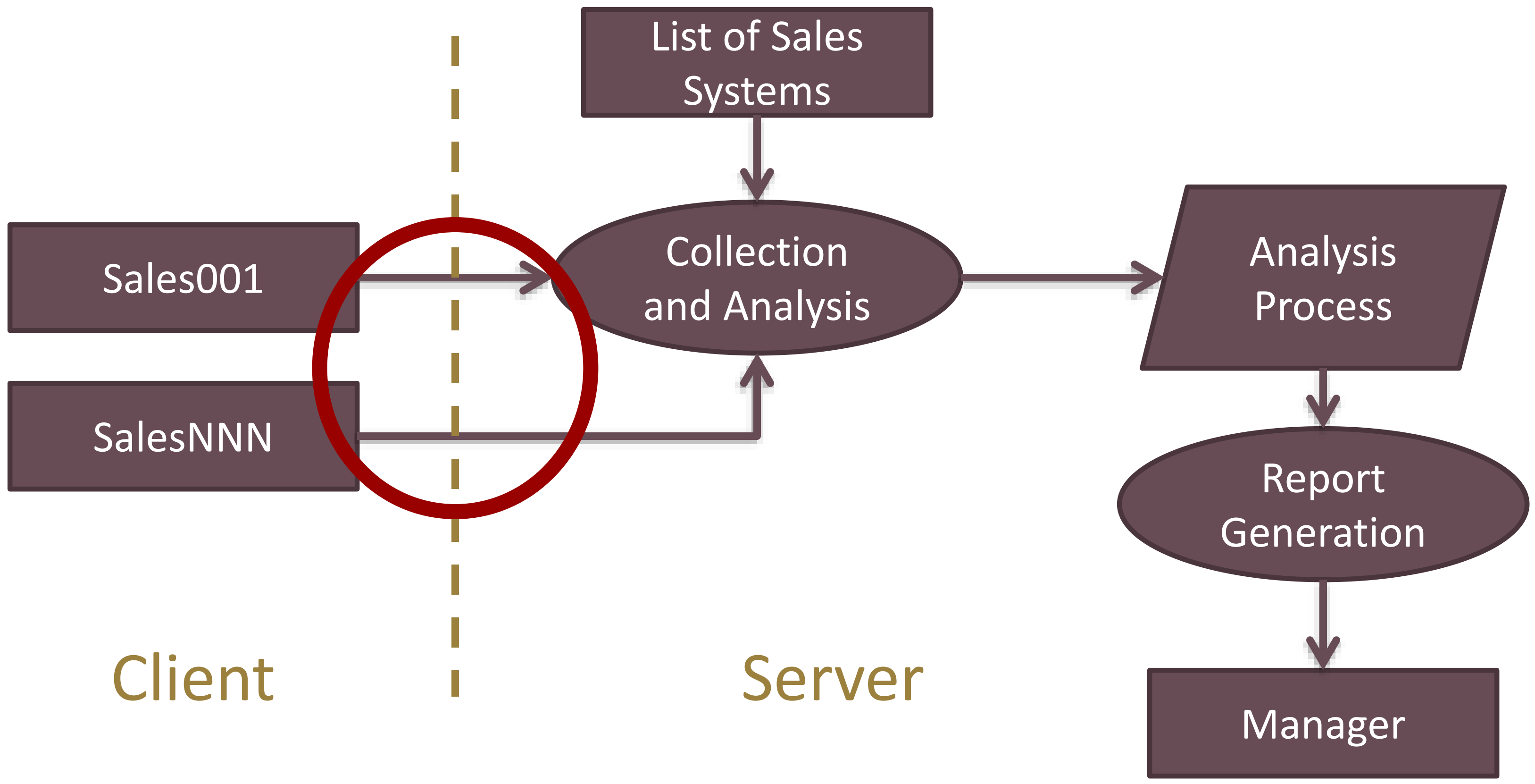
- Collect accounting files from sales force
- Compute sales data
- Produce weekly reports

for details see
Hernan, Shawn, Scott Lambert, Tomasz Ostwald, and Adam Shostack. "Uncover security design flaws using the STRIDE approach (2006)." *MSDN Magazin Nov 2006*

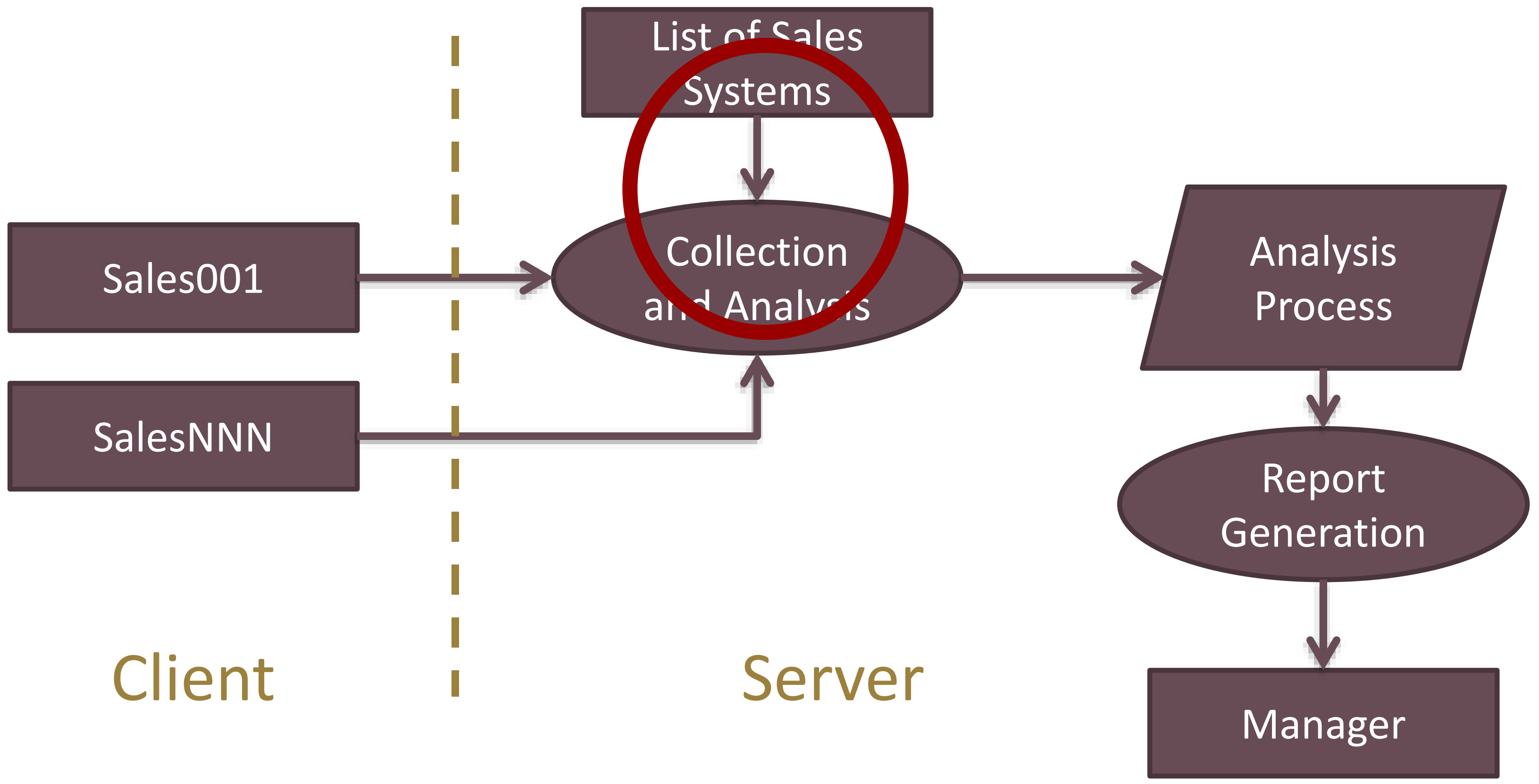
Use case: Sales entry



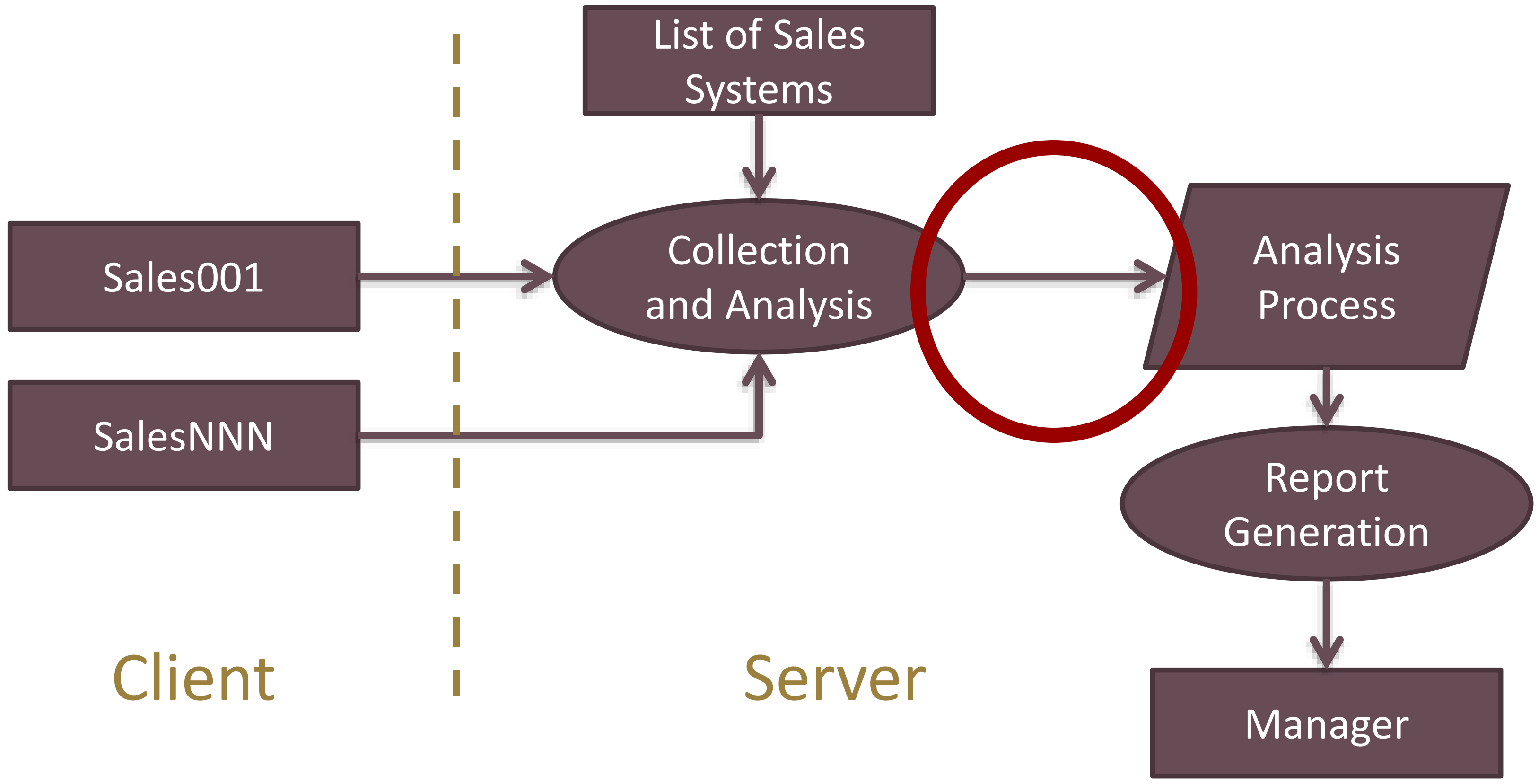
Use case: Sales entry



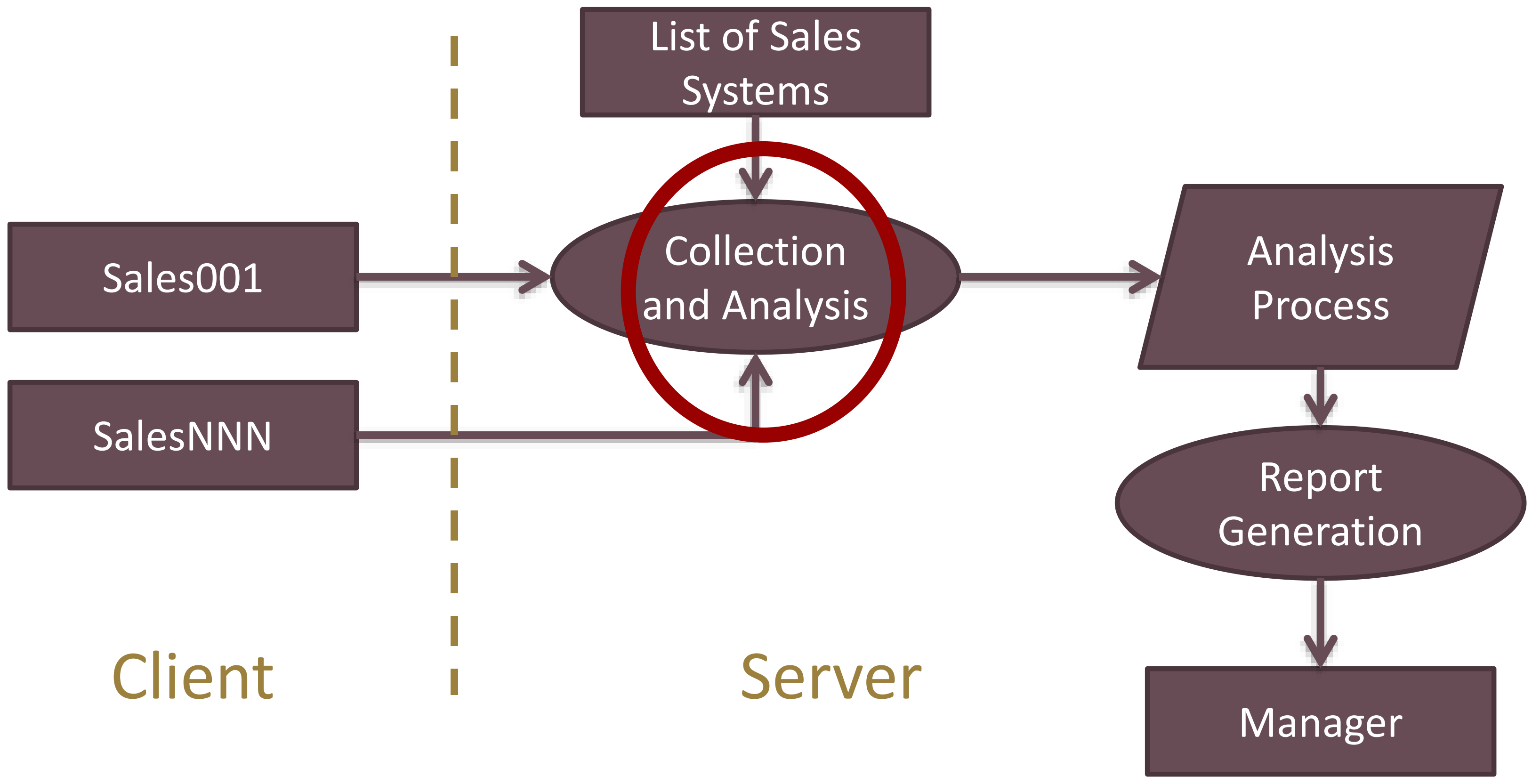
Use case: Sales entry



Use case: Sales entry

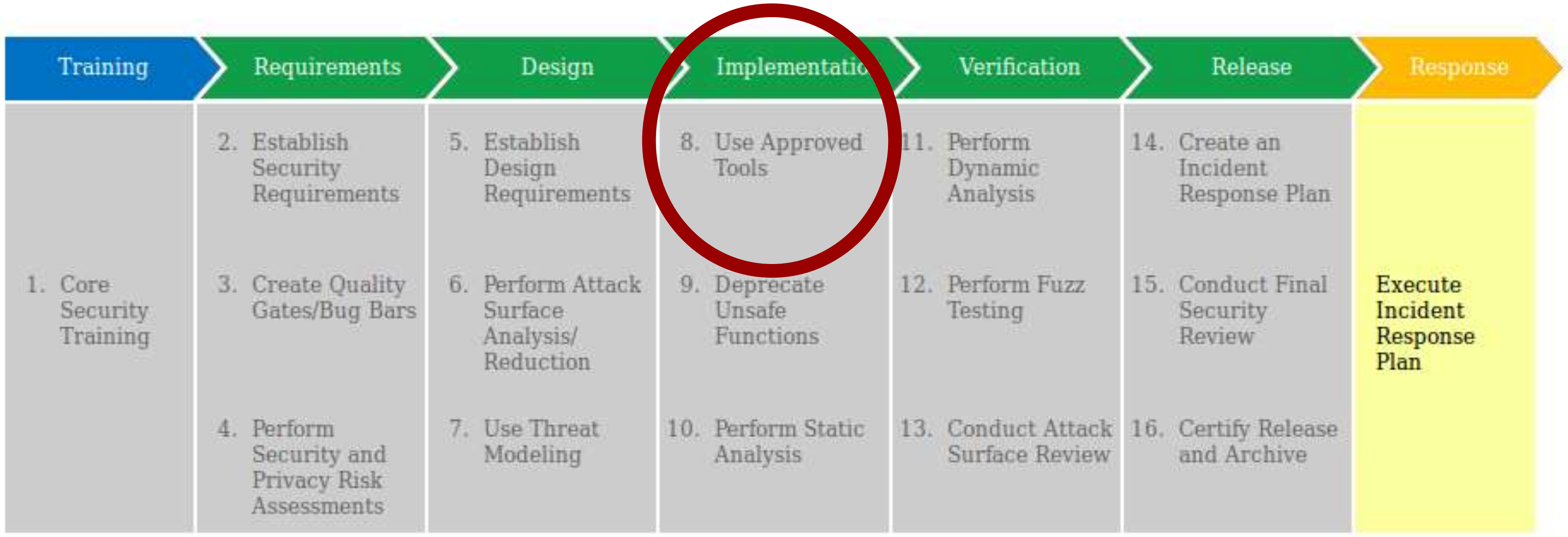


Use case: Sales entry



Any unhandled threats turned up by threat modeling must be tracked!

Microsoft SDLs



Training	Requirements	Design	Implementatio	Verification	Release	Response
	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	
1. Core Security Training	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	Execute Incident Response Plan
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

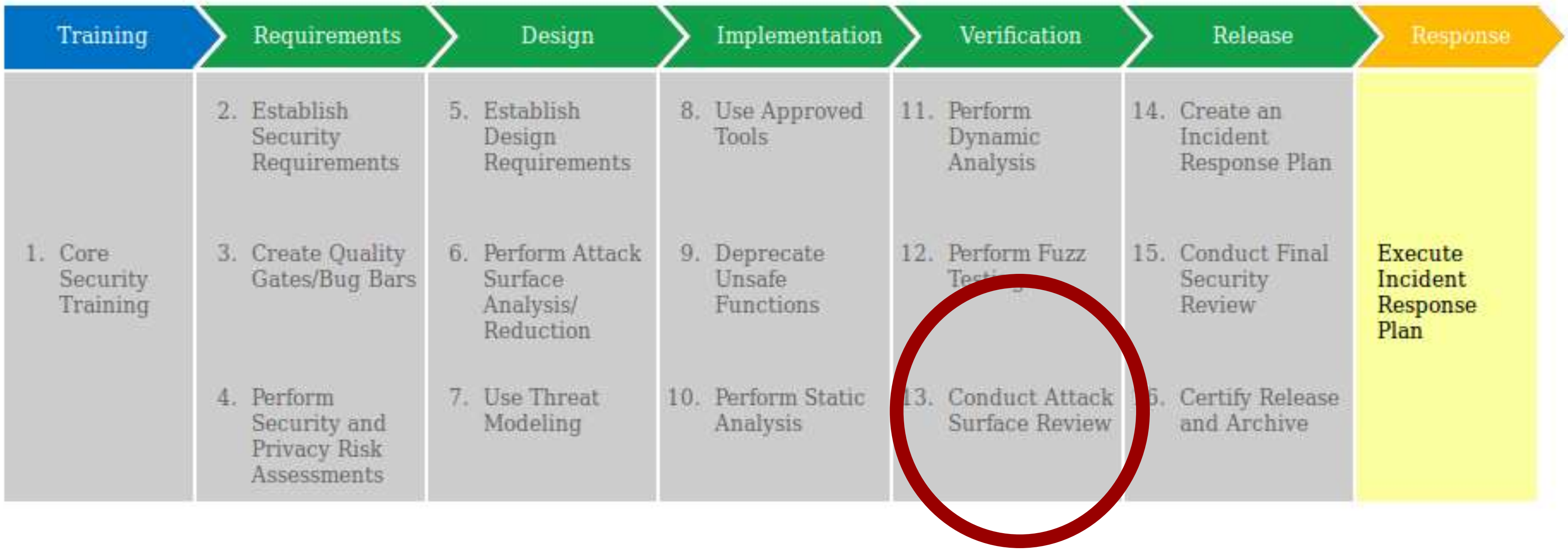
Use Approved Tools

- Some libraries are vulnerable and have safe alternatives (e.g. string.h bad vs strsafe.h good)
- Modern compilers automatically mitigate a number of vulnerabilities (e.g. stack canaries, heap integrity checks, SAFESEH, etc.)
- Appropriate static and dynamic analysis tools automate the enforcement of security practices

Static Analysis, Deprecation

- Microsoft runs static checkers at checking (quality gates)
- Banned over 100 C functions for new code

Microsoft SDLs

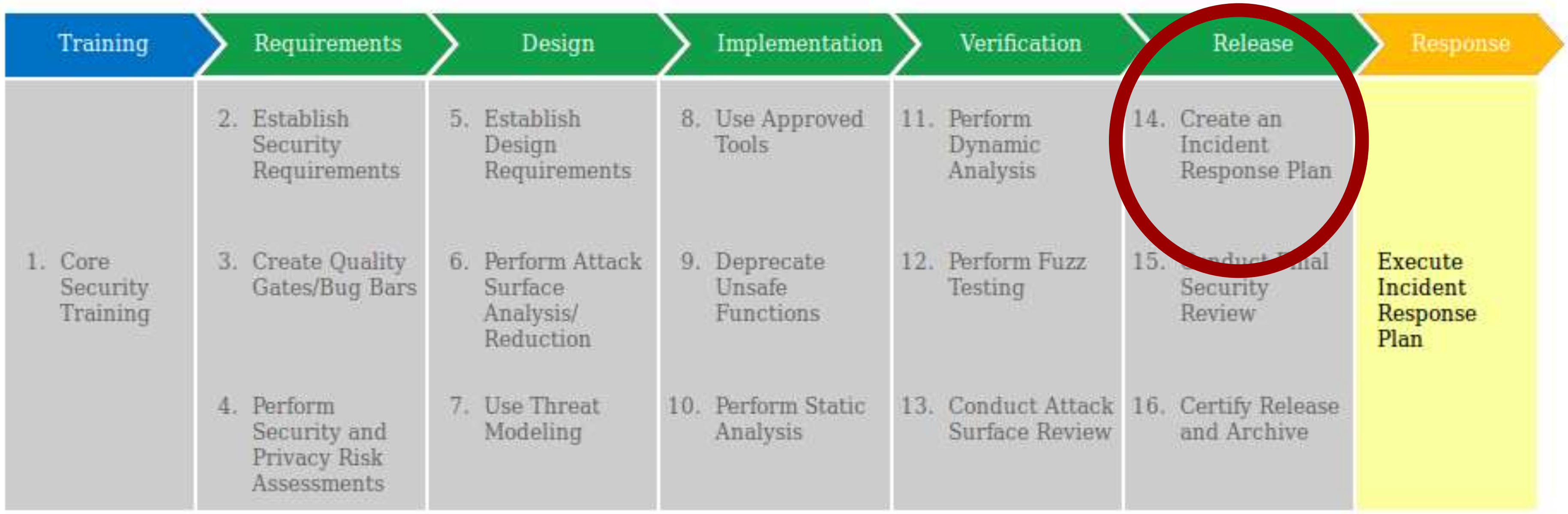


Training	Requirements	Design	Implementation	Verification	Release	Response
	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	
1. Core Security Training	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	**Execute Incident Response Plan**
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

Conduct Attack Surface Review

- What is every source of input to the application?
- Are there any new sources since the last milestone?
- Much more fine grained than threat modeling
- All sources of input must have a defensive approach applied
- Tools help automate this practice

Microsoft SDLs



Training	Requirements	Design	Implementation	Verification	Release	Response
1. Core Security Training	2. Establish Security Requirements	5. Establish Design Requirements	8. Use Approved Tools	11. Perform Dynamic Analysis	14. Create an Incident Response Plan	Execute Incident Response Plan
	3. Create Quality Gates/Bug Bars	6. Perform Attack Surface Analysis/ Reduction	9. Deprecate Unsafe Functions	12. Perform Fuzz Testing	15. Conduct Final Security Review	
	4. Perform Security and Privacy Risk Assessments	7. Use Threat Modeling	10. Perform Static Analysis	13. Conduct Attack Surface Review	16. Certify Release and Archive	

Create Incidence Response Plan

- Attacks always get better
- New threats emerge every day
- Vulnerabilities always exist in non-trivial systems
- Who should be contacted when an incident occurs?
- Who should deal with third-party code?
- What priority should be applied to fixing new vulnerabilities?

`

Who should implement these security practices?

Security Roles

- **Everyone**: “security awareness” – buy into the process
- **Developers**: know the security capabilities of development tools and use them, know how to spot and avoid *relevant*, common vulnerabilities
- **Managers**: *enable* the use of security practices
- **Security specialists**: everything security

https://cwe.mitre.org/

Organizational Architectures

Increased Cost and Coverage

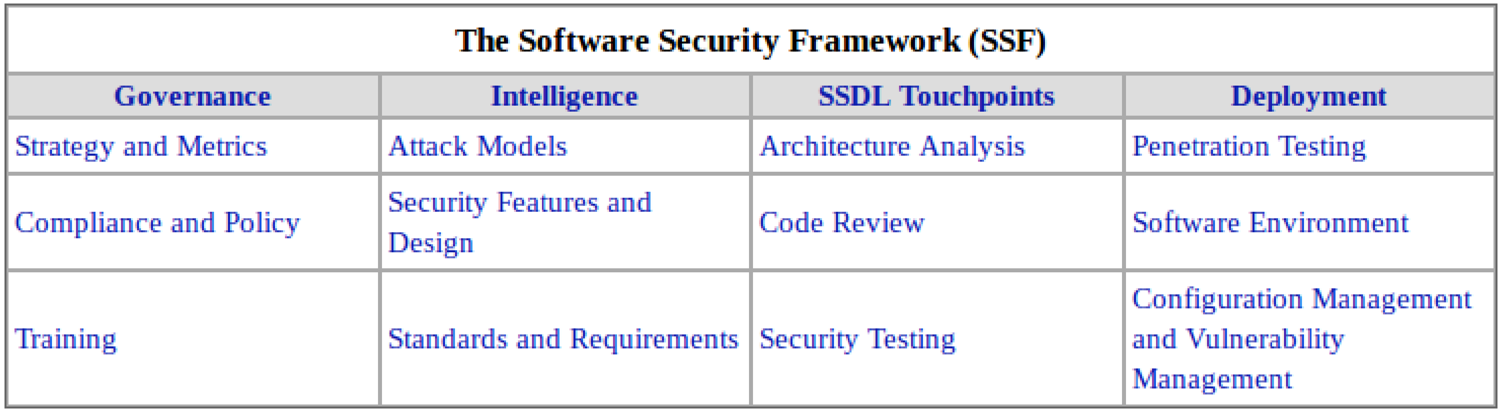
- **Centralized**: development teams consult with a core group of security specialists when they need help
- **Distributed**: development teams hire security specialists to be a first-class member of the team
- **Weak Hybrid**: centralized group of security specialists and teams with security critical applications hire specialists
- **Strong Hybrid**: centralized group of security specialists and most teams also hire specialists

Tuning SDLs

- No one set of security practices work across every industry... or even for every project in a given company
- Expertise is required to determine what set of practices is the most cost effective

BSIMM

- Building Security In Maturity Model
- See what practices other companies utilize
- Understand, measure, and plan software security initiatives



The Software Security Framework (SSF)			
Governance	**Intelligence**	**SSDL Touchpoints**	**Deployment**
Strategy and Metrics	Attack Models	Architecture Analysis	Penetration Testing
Compliance and Policy	Security Features and Design	Code Review	Software Environment
Training	Standards and Requirements	Security Testing	Configuration Management and Vulnerability Management

None of this is scientifically validated.

Future: Measures and Standards

- NHTSA inspired star ratings
- Building Codes for Software
- Security Guarantees
- Liability
- Science

Summary

- Security is a quality among others, often very important
- As all QA, design security QA throughout the process, not only after the fact
- Security requires special expertise, awareness by developers + experts
- Use tools, modeling, automate, …