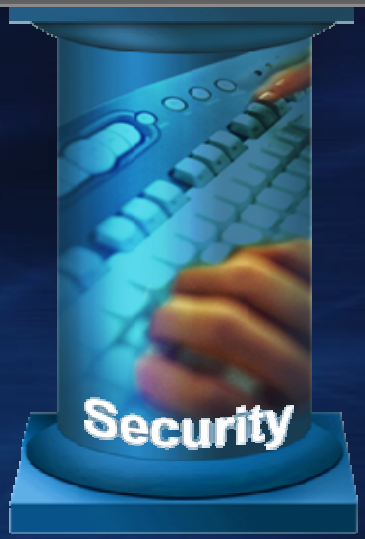
The Security Development Lifecycle

Steven B. Lipner
Director of Security Engineering Strategy
Security Business and Technology Unit
Microsoft Corporation

Context and History

- 1960s – penetrate and patch
- 1970s – theories and research security kernels
- 1980s – the Orange Book and high assurance product efforts
- Late 1980s - viruses
- Early 1990s – the Internet, firewalls, and C2 as commodity
- Late 1990s – security features, vulnerability discovery, worms
- 2000s – the quest for real world assurance

Trustworthy Computing



- Resilient to attack
- Protects confidentiality, integrity, availability of data and systems

- Individual control of personal data
- Adhere to fair information principles
- Protects individual's right to be left alone

- Engineering Excellence
- Dependable, performs at expected levels
- Available when needed

- Open, transparent interaction with customers
- Address issues with products and services
- Help customers find appropriate solutions

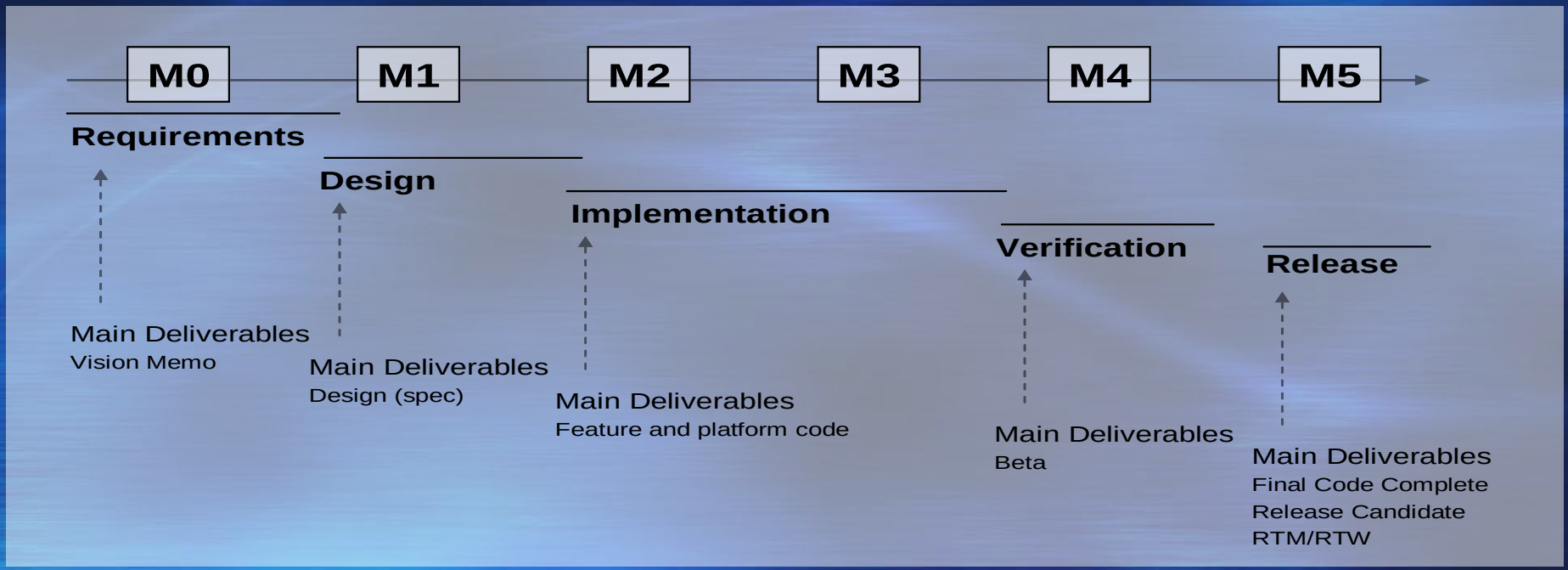
What is the Security Development Lifecycle?

A PROCESS by which Microsoft develops software, that defines security requirements and milestones

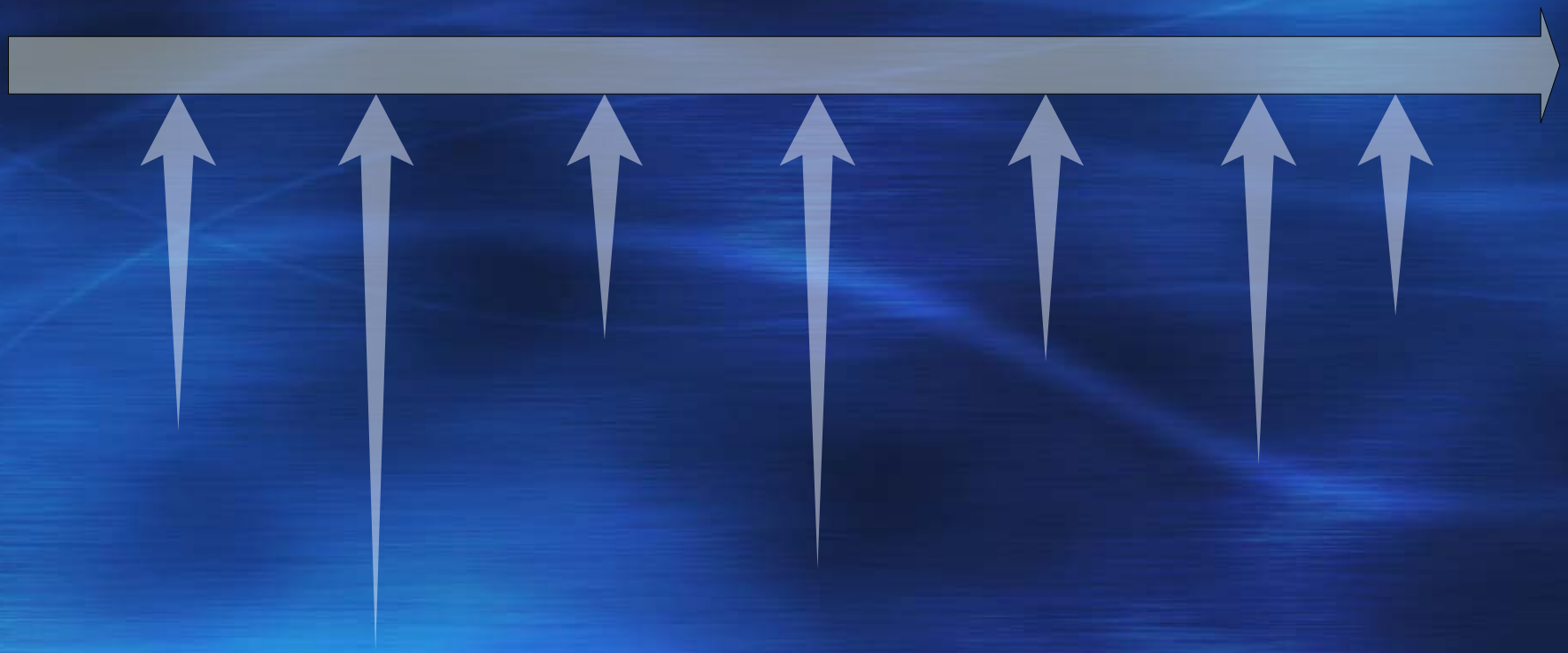
- MANDATORY for products that are exposed to meaningful security risk
- EVOLVING and new factors, such as privacy, are being added
- COMPATIBLE with COTS product development processes
- EFFECTIVE at addressing security issues; designed to produce *DEMONSTRATABLE RESULTS* (not all methodologies do this)
- It has shown itself to highly effective at reducing vulnerabilities in commercial software

The SDL puts Microsoft on a path toward real world assurance and more secure software

Traditional Baseline Process



Integrating the SDL into the Process



Training for the SDL

- New employees do not arrive with ability to develop secure software
  - Microsoft trains staff as part of New Employee Orientation
  - Microsoft trains staff as part of a security push
  - Microsoft trains developers, testers, program managers, user education staff and architects annually
  - Microsoft funds curriculum development through Microsoft Research
  - Microsoft publishes training on writing secure code, threat modeling and SDL (pending) and offers courses (see http://www.microsoft.com/learning/)

Stages of the SDL

- Requirements
  - Consider security “up front”
- Design
  - Architecture, TCB, least privilege, threat models
- Development
  - Code reviews, fuzz testing, static analysis tools
- Verification
  - “Security push” for new and legacy code
- Release
  - Final Security Review (FSR)
- Response
  - Closing the feedback loop

Requirements Phase

- Opportunity to consider security at the outset
- Central Security Team assigns Security Advisor (SA)
- Development team identifies security requirements
- SA reviews product plan, makes recommendations, ensures resources allocated by management
- SA assess security milestones and exit criteria
- (NOTE: This SA will stay with the project through the Final Security Review)

Design

- Design Stage
  - Define and document security architecture
  - Identify security critical components (“trusted base”)
  - Identify design techniques (e.g., layering, managed code, least privilege, attack surface minimization)
  - Document attack surface and limit through default settings
  - Create threat models (e.g., identify assets, interfaces, threats, risk) and mitigate threats through countermeasures
  - Identify specialized test tools
  - Define supplemental ship criteria due to unique product issues (e.g., cross-site scripting tests)
  - Confer with SA on questions
- Exit Criteria: Design review complete and signed off by development team *and* security advisor

Development

- Apply coding and testing standards (e.g., safe string handling)
- Apply fuzz testing tools (structured invalid inputs to network protocol and file parsers)
- Apply static code analysis tools (to find, e.g., buffer overruns, integer overruns, uninitialized variables)
- Conduct code reviews

Verification

- Software functionally complete and enters Beta
- Because code complete, testing both new and legacy code
- Security Push
  - Code reviews – focus on legacy code
  - Penetration and other security testing
  - Review design, architecture, threat models in light of new threats

Final Security Review (FSR)

- “From a security viewpoint, is this software ready to deliver to customers?”
  - Two to six months prior to software completion, depending on the scope of the software
  - Software must be in a stable state with only minimal non-security changes expected prior to release
- FSR Results: If the FSR finds a pattern of remaining vulnerabilities, the proper response is not just to fix the vulnerabilities found, but to revisit the earlier phases and take pointed actions to address root causes (e.g., improve training, enhance tools)

Final Security Review (FSR)

- What is in the FSR
  - Completion of a questionnaire by the product team
  - Interview by a security team member assigned to the FSR
  - Review of bugs that were initially identified as security bugs, but on further analysis were determined not to have impact on security, to ensure that the analysis was done correctly
  - Analysis of any newly reported vulnerabilities affecting similar software to check for resiliency
  - Additional penetration testing, possibly by outside contractors to supplement the security team
- “Penetrate and patch?”
  - Penetration test is only one component of FSR
  - Overall assessment of fitness is the primary output
  - Individual findings support these findings

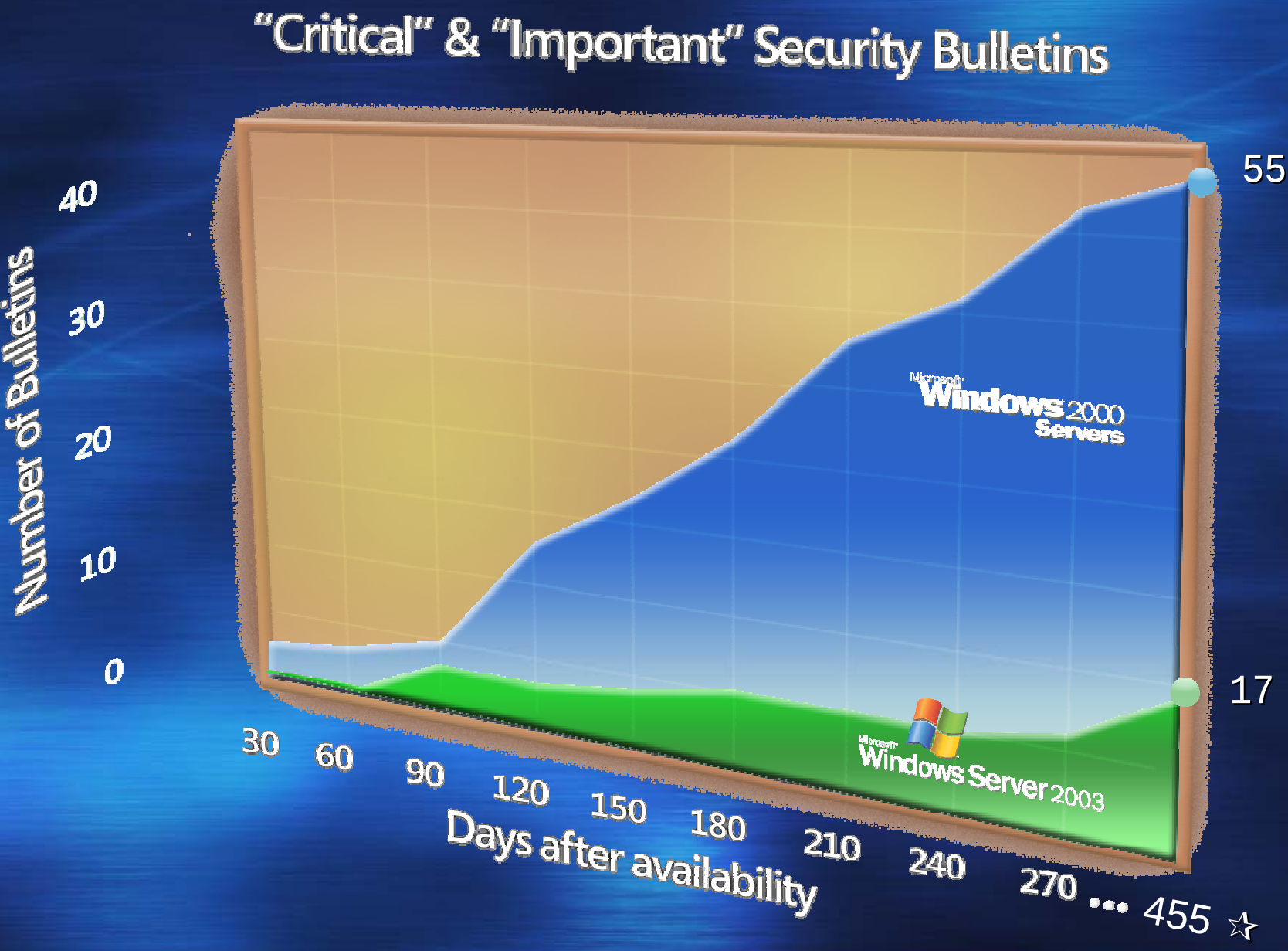
Response Phase

- Microsoft Security Response Center
- Sustained Engineering Teams
- Patch Management
- Post mortems and feedback to the SDL

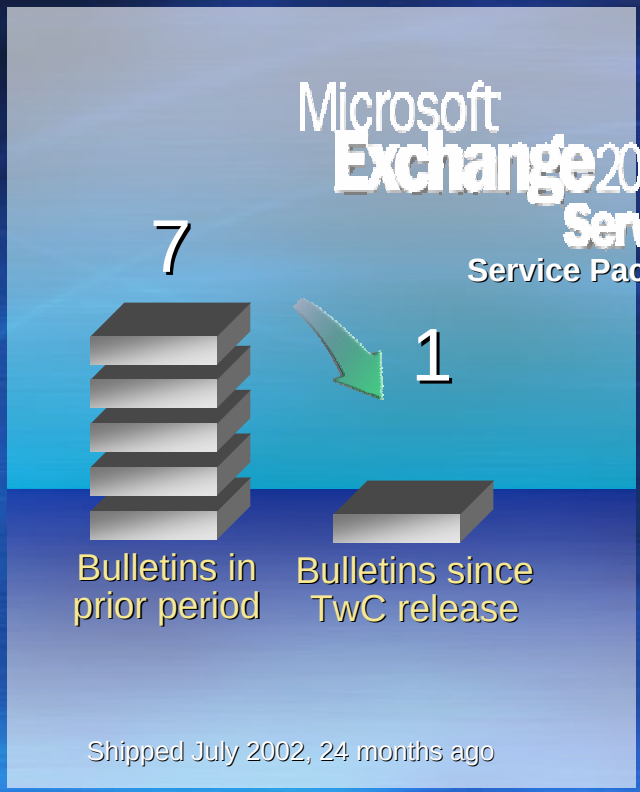
Evolving the SDL

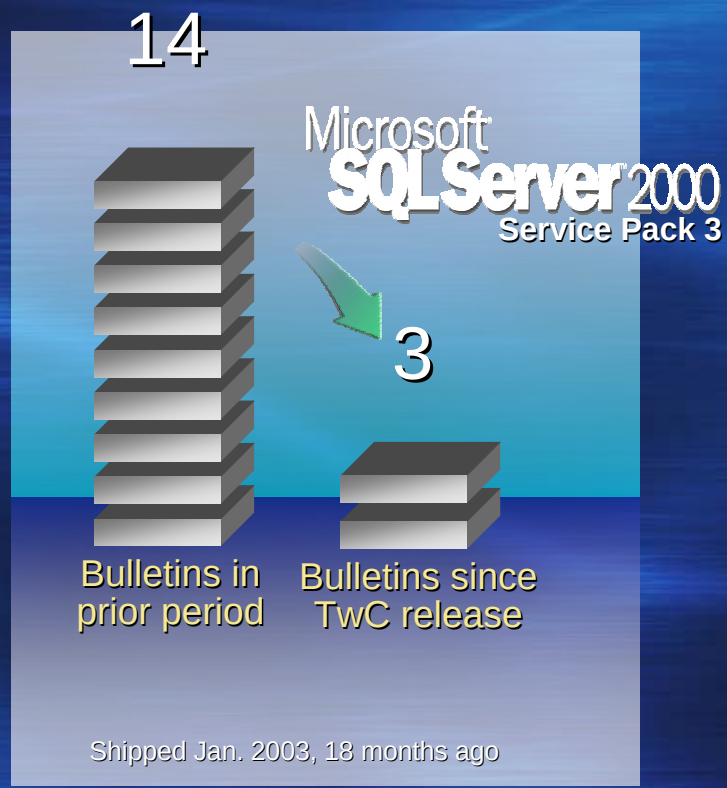
- The SDL is documented
- Updates are released twice a year
  - New objectives
  - New techniques
  - New threats
- “We reserve the right to change the rules at the drop of a hat if the threat environment changes”

Measuring Results of the SDL



Measuring Results of the SDL





Measuring Results of the SDL

- Measured results to date based on early implementation of the SDL
  - Process has evolved from January 2002
- Results to date demonstrate that incremental application of process yields incremental benefits
  - Improved security in place when software is released
  - Evidence of SDL is software and development artifacts (e.g. threat models) – *not* primarily documentation
  - Process is *not* “all or nothing”
  - Process *is* expensive, but benefits justify investment

The SDL and the Common Criteria

- Common Criteria has been successful in many ways
  - Mutual recognition
  - Flexibility to evaluate new classes of products and configurations
  - Vendor commitment to process
- But there are still some hard questions
  - Common Criteria focuses on security features
    - Attackers don't
  - At commercially viable assurance levels, Common Criteria does not reduce vulnerability rates
  - Common Criteria evaluation is often after-the-fact
  - Common Criteria evaluation is relatively expensive (though much cheaper than implementation of the SDL

Closing Thoughts and Recommendations

- To the academic community
  - *Teach secure* features, not *security* features
  - Help us evolve techniques to make the SDL even more effective
- To NSA and other Common Criteria partners
  - Consider evolution of the Common Criteria to recognize processes such as (based on) the SDL
  - Combination of commercial viability and reduced vulnerability will ensure relevance and market acceptance

Microsoft®

Your potential. Our passion.™