Building Trust in Computing

Glenn Schoonover CISSP
Executive Director
Worldwide Defense Technical Strategy

Microsoft Corporation

Glenn.Schoonover@microsoft.com

The Interconnected World

Benefits

- Communication and collaboration
- Network Centric Operations
- Personal & business productivity gains

Challenges

- Evolving malicious code threats
- Maintaining privacy and confidentiality
- New risks: Spyware and phishing

What is the problem?

Programmers can be taught to avoid creating buffer overflows and other well-known vulnerabilities found in commercial software, said Lawrence Hale, speaking at the FOSE 2003 conference on government technology.

Lawrence Hale, former deputy director of the DHS' U.S. Computer Emergency Response Team, said, "the things that are costing us the most pain are preventable."

Microsoft's Security Philosophy: Past

- Ease of Use was most important
- Make components work together seamlessly
- Services usually enabled by default
- Applications and APIs given many privileges (i.e., Outlook object model)
- Security often thought of in terms of "features." – (IPSEC, EFS, etc.)

Microsoft's Security Philosophy: Present & Future

- Security taking precedence over ease-of-use
- Design for security (Secure Windows Initiative)
- New Tools for finding coding flaws
- Unprecedented resources given to Security design and Response groups inside Microsoft

What is Trustworthy Computing?

Trustworthy Computing means that government, commercial and individual users can say:

"I can trust this product or service. It is reliable, safe, and my privacy is respected."

Trustworthy Computing

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

- Individual control of personal data
- Products, online services 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

Microsoft's Security Focus

Vision

Trustworthy solutions for secure software and services with tools and guidance to keep customers safe

Technology Investments

- Excellence in fundamentals
- Security innovations

Prescriptive Guidance

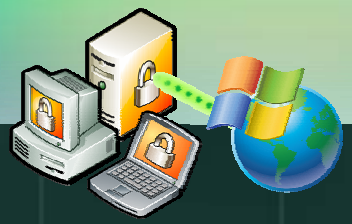
- Scenario-based content and tools
- Authoritative incident response

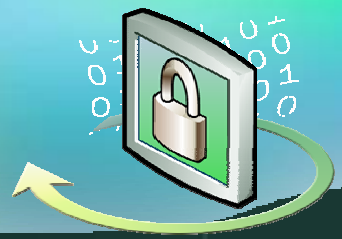
Industry Partnership

- Awareness and education
- Collaboration and partnership

Technology Fundamentals

- Engineering excellence
- Security development lifecycle
- Microsoft Security Response Center
- Sharing best practices with all developers





Technology Investments

Improving the Application Development Process

- Consider security
  - At the start of the process
  - Throughout development
  - Through deployment
  - At all software review milestones
- Do not stop looking for security bugs until the end of the development process

The SD³ Security Framework

SD³

Secure by Design

- Secure architecture and code
- Threat analysis
- Vulnerability reduction

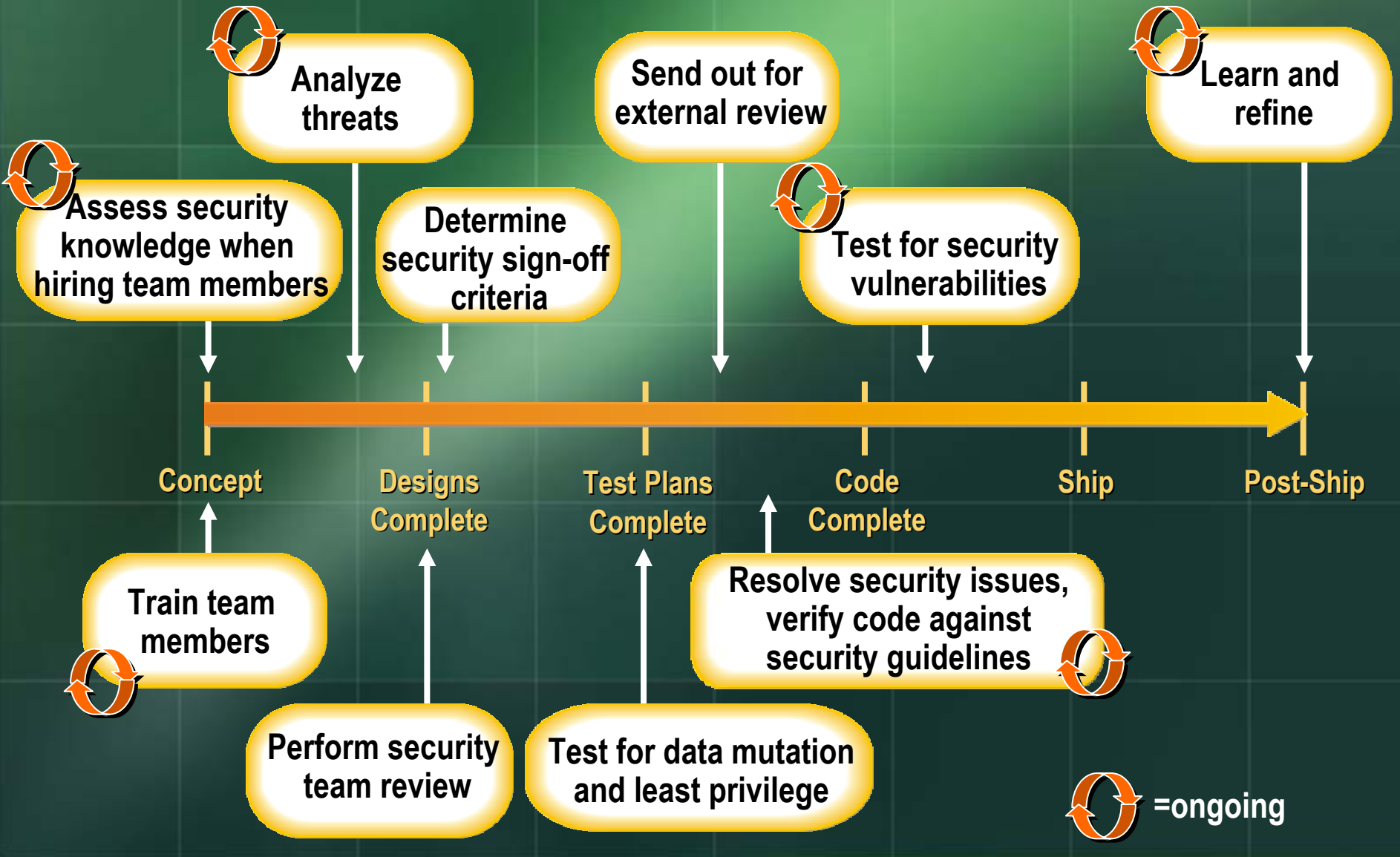
Secure by Default

- Attack surface area reduced
- Unused features turned off by default
- Minimum privileges used

Secure in Deployment

- Protection: Detection, defense, recovery, and management
- Process: How to guides, architecture guides
- People: Training

Secure Product Development Timeline



Secure By Design

- Raise security awareness of design team
  - Use ongoing training
  - Challenge attitudes - “What I don’t know won’t hurt me” does not apply!
- Get security right during the design phase
  - Define product security goals
  - Implement security as a key product feature
  - Use threat modeling during design phase

Never trust user input

- Check all parameters completely
  - Assume all input is harmful until proven otherwise
  - Assume they may be completely random
  - Assume they may be subtly malicious
  - Look for valid data and reject everything else
- Potential problems
  - Buffer overflows
  - Script injection
  - Denial of service

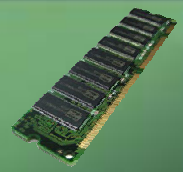
Do Not Trust User Input

- Constrain, reject, and sanitize user input with
  - Type checks
  - Length checks
  - Range checks
  - Format checks

Buffer overflows

- Most attacked vulnerability: 75% of all exploits
- Caused by not checking input buffer sizes
  - Buffers declared on the stack are easiest to exploit, a.k.a. “stack smashing”
    - Attackers send data too big for the buffer and figure out how the stack is affected
    - Instructions are then inserted to take over the app
  - Non-stack exploits (heap, exception handling, object pointers) are possible too

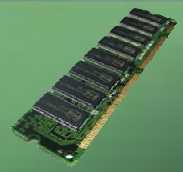
Problem: Memory



Anatomy of a Buffer Overrun

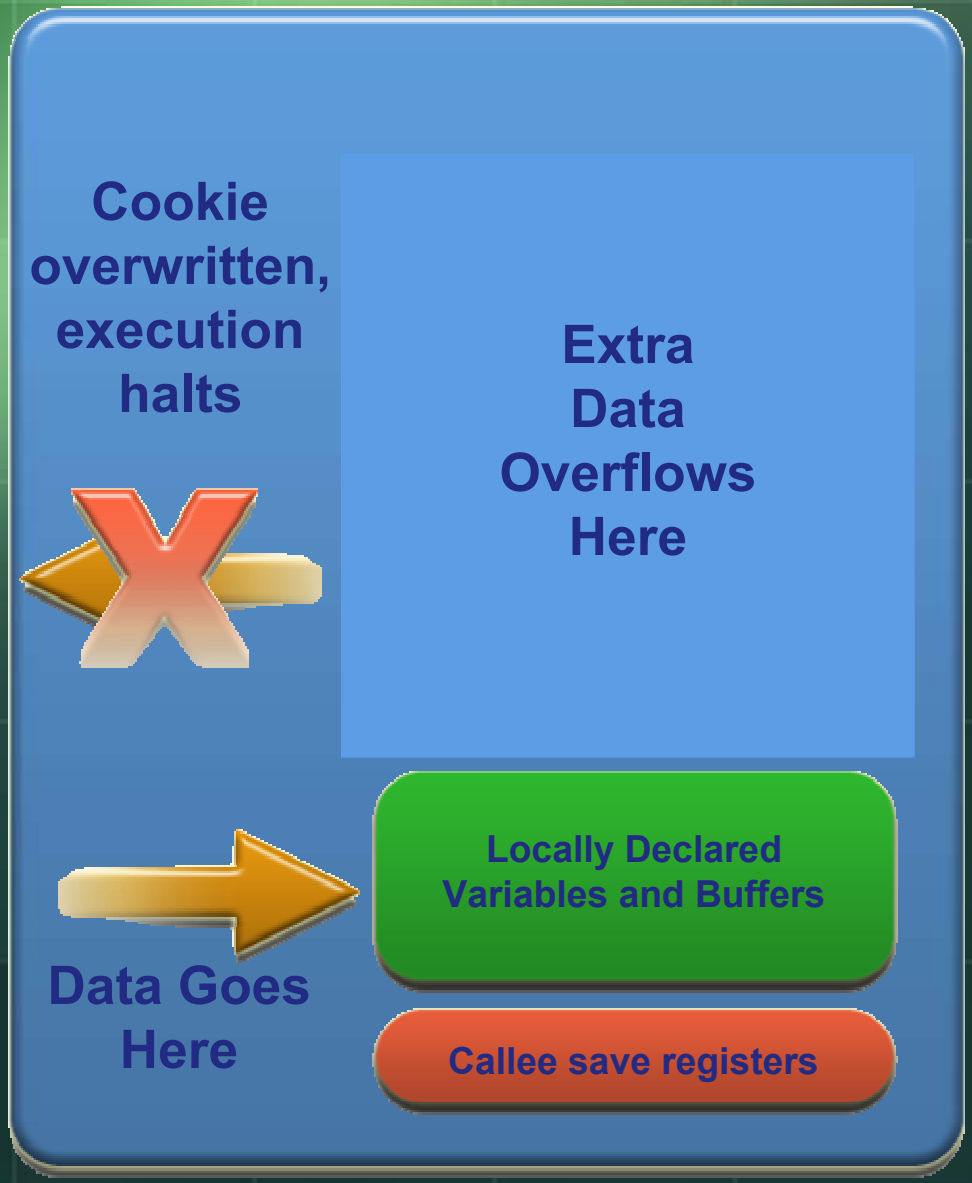
- Some services and applications improperly handle malformed messages
- An attacker can send a message with data that is longer than expected
  - Extra data includes malicious code
- Malicious code is inadvertently written to area of memory where that code is executed

Solution: /GS Switch



Reduce Risk of Buffer Overruns

- To check for buffer overruns in production code, the Visual C++ .NET compiler implements the new /GS switch
- The /GS switch provides a "speed bump," or cookie, between the buffer and the return address
- If an overrun writes over the return address, it will have to overwrite the cookie put in between it and the buffer



Run with Least Privilege

- Well-known security doctrine:

  "Run with just enough privilege to get the job done, and no more!"

- Elevated privilege can lead to disastrous consequences
  - Malicious code executing in a highly privileged process runs with extra privileges too
  - Many viruses spread because the recipient has administrator privileges

Use least privilege

- Require minimal privileges
  - Don’t require Admin privileges unless you really must restrict use to administrators
  - If you have features that require admin privilege, allow lesser accounts to run with those features disabled
  - Run services as LocalService or NetworkService, not LocalSystem
- Set security descriptors to grant the minimum access necessary
- Restrict access tokens to remove privileges and groups memberships you don’t require

Reduce the Attack Surface

- Expose only limited, well documented interfaces from your application
- Use only the services that your application requires
  - The Slammer and CodeRed viruses would not have happened if certain features were not on by default
  - ILoveYou (and other viruses) would not have happened if scripting was disabled
- Turn everything else off

Reduce surface area

- Disable any feature that won't be used by most (80%?) of your customers
  - Those that need the disabled features can enable them
- Retire old features and interfaces ("deprecate")
  - This can pose compatibility problems
- Remove undocumented interfaces and dead code
  - Make sure debug-only features are not in the shipping version

Assume external systems are insecure

- Suspect all data from any outside source
  - Anyone can load and call a DLL
    - Don’t assume you have a known set of clients
  - There is no such thing as client-side security
    - Attackers can write their own client code to send whatever data they want
  - Don’t trust unauthenticated servers

Have secure defaults

- Be secure right out of the box
- Do not rely customers to
  - Read documents
  - Make uninformed security choices
  - Disable any features

Default to a secure mode

- Go to a secure state after failure
  - Access denied?
    - Make sure the operation stops
    - Trace failure paths to verify
  - Special case fails?
    - Make sure the default case is always safe
- Bonus principle: give away minimal information
  - i.e. error messages should not help an attacker

Do Not Rely on Security by Obscurity

- Do not hide security keys in files
- Do not rely on undocumented registry keys
- Assume attackers have
  - Source code
  - Specs
  - Debuggers
  - Network sniffers
- Do not skip any undocumented features or interfaces

Protect secrets correctly

- Secrets include passwords, keys, credentials
- Avoid them if possible
  - Let user supply the password instead of caching it
  - Store a hash of the secret instead of the secret
- Never store secrets in code
- Never write your own encryption/obfuscation code
- Two DPAPI functions:
  - CryptProtectData
  - CryptUnprotectData
- Two stores for data encrypted with DPAPI:
  - User store
  - Machine store
- Erase secrets as soon as you're done with them

Have defense in depth

- Protect yourself
  - Don’t rely on external things (e.g. firewalls) to protect you
- Use as many levels of protection as you can
  - The more you have, the more secure you’ll be
  - Plan for failure
    - Run through the failure scenarios
    - If an attacker gets through one defense, make sure they go up against others

Fail Intelligently (1 of 2)

```
DWORD dwRet = IsAccessAllowed(…);
if (dwRet == ERROR_ACCESS_DENIED) {
      // Security check failed.
      // Inform user that access is denied
} else {
      // Security check OK.
      // Perform task…
}
```

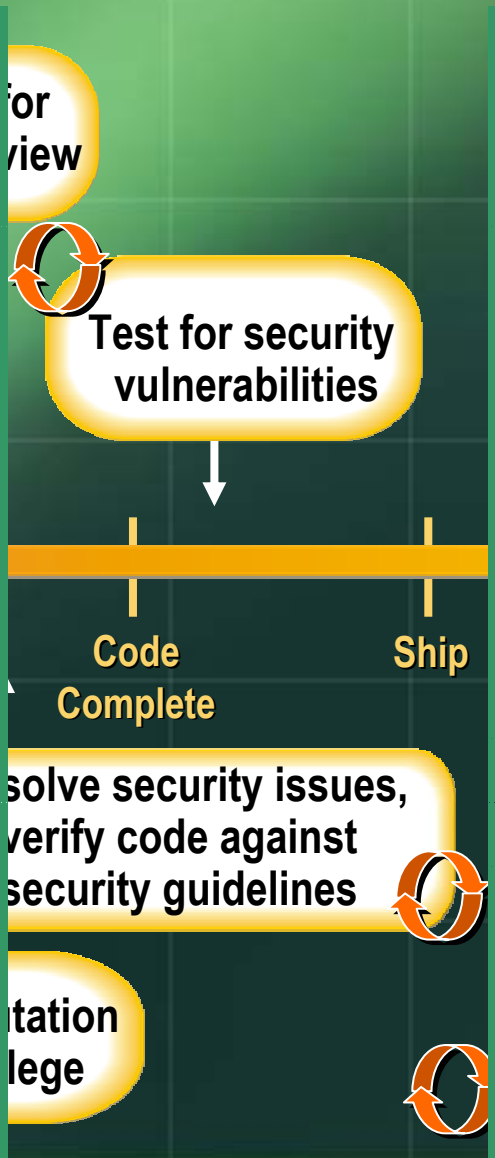
What if IsAccessAllowed() returns ERROR_NOT_ENOUGH_MEMORY?

- If your code does fail, make sure it fails securely

Fail Intelligently (2 of 2)

- Do not:
  - Reveal information in error messages
  - Consume resources for lengthy periods of time after a failure
- Do:
  - Use exception handling blocks to avoid propagating errors back to the caller
  - Write suspicious failures to an event log

Secure Product Development Timeline



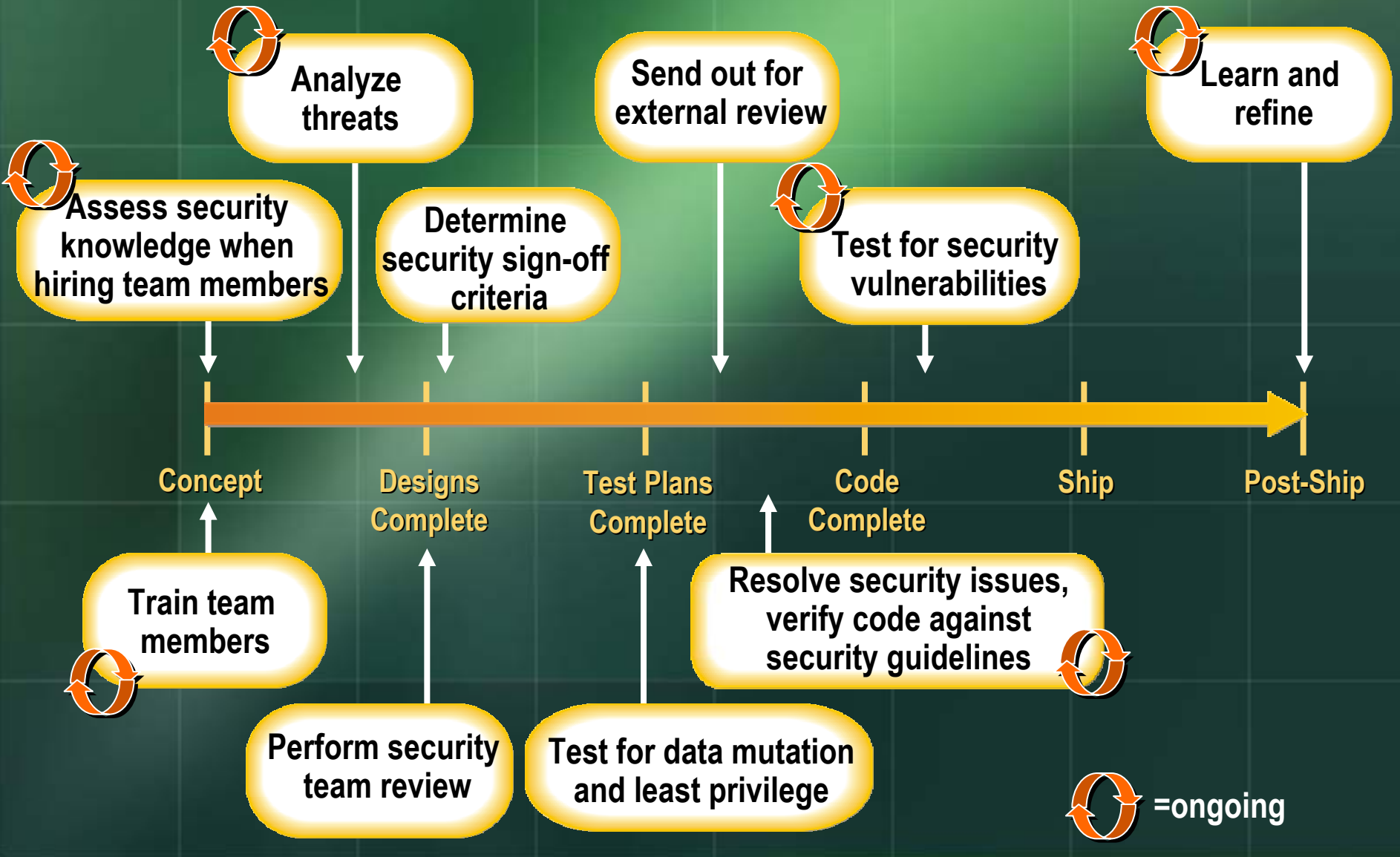
Test Security

- Involve test teams in projects at the beginning
- Use threat modeling to develop security testing strategy
- Think Evil. Be Evil. Test Evil.
  - Automate attacks with scripts and low-level programming languages
  - Submit a variety of invalid data
  - Delete or deny access to files or registry entries
  - Test with an account that is not an administrator account
- Know your enemy and know yourself
  - What techniques and technologies will hackers use?
  - What techniques and technologies can testers use?

Learn from Mistakes

- If you find a security problem, learn from the mistake
  - How did the security error occur?
  - Has the same error been made elsewhere in the code?
  - How could it have been prevented?
  - What should be changed to avoid a repetition of this kind of error?
  - Do you need to update educational material or analysis tools?

Secure Product Development Timeline



Microsoft Security Bulletin MS03-007

Unchecked Buffer In Windows Component Could Cause Server Compromise

- Affected Software:
  - Microsoft Windows NT 4.0
  - Microsoft Windows NT 4.0 Terminal Server Edition
  - Microsoft Windows 2000
  - Microsoft Windows XP
- Not Affected Software:
  - Microsoft Windows Server 2003

SD³ At Work – MS03-007
Windows Server 2003 Unaffected

The underlying DLL (NTDLL.DLL) not vulnerable	Code made more conservative during Security Push
Even if it was vulnerable	IIS 6.0 not running by default on Windows Server 2003
Even if it was running	IIS 6.0 doesn't have WebDAV enabled by default
Even if it did have WebDAV enabled	Maximum URL length in IIS 6.0 is 16kb by default (>64kb needed)
Even if the buffer was large enough	Process halts rather than executes malicious code, due to buffer-overrun detection code (-GS)
Even if it there was an exploitable buffer overrun	Would have occurred in *w3wp.exe* which is now running as 'network service'

It's Not Just About Technology

- Trustworthy Computing Initiative provides a foundation for:
  - Process (procedures, guidelines)
  - Technology (hardware, software, networks)
  - People (culture, knowledge)
- Security needs to be comprehensive
- Technology is neither the whole problem nor the whole solution

Partnerships

Law Enforcement

Public Policy

Global Infrastructure Alliance
for Internet Safety

Virus Information Alliance

Consumer Awareness

Industry Partnership

Session Summary

- Secure Development Process
- Risk Mitigation
- Security Best Practices

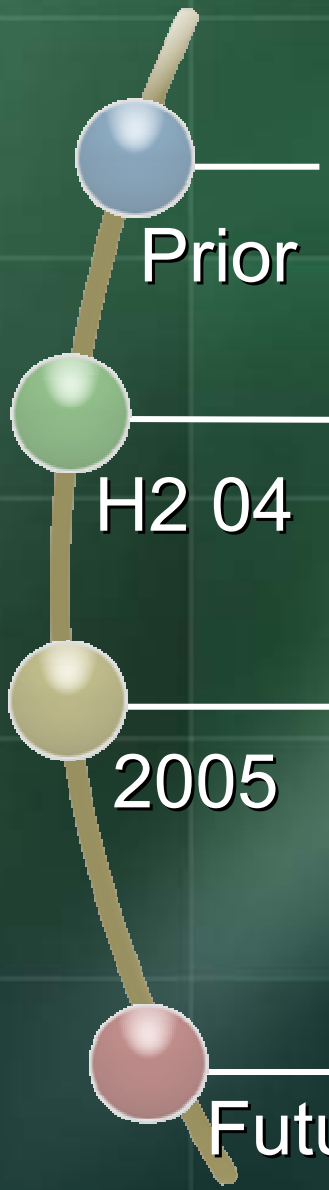
Security

Much to Do

Great Progress

Journey with milestones

Security Timeline



Prior

- Microsoft Baseline Security Analyzer (MBSA) v1.2
- Virus Cleaner Tools
- Systems Management Server (SMS) 2003
- Software Update Services (SUS) SP1
- Internet Security and Acceleration (ISA) Server 2004 Standard Edition

H2 04

- Windows XP Service Pack 2
- Patching Technology Improvements (MSI 3.0)
- Systems Management Server 2003 SP1
- Microsoft Operations Manager 2005

2005

- Windows malicious software removal tool
- Windows Server 2003 Service Pack 1
- Windows Update Services
- ISA Server 2004 Enterprise Edition
- Windows Rights Management Services SP1
- Windows AntiSpyware
- System Center 2005
- Windows Server 2003 “R2”
- Visual Studio 2005

Future

- Vulnerability Assessment and Remediation
- Active Protection Technologies
- Antivirus

For More Information

- MSDN Security Site (developers)
  - http://msdn.microsoft.com/security
- TechNet Security Site (IT professionals)
  - http://www.microsoft.com/technet/security

Microsoft®

Supporting the Warfighter