SECURE SOFTWARE DEVELOPMENT

A Microsoft Perspective

Ed Harrison

- Ed Harrison
- Computer Science grad
  - University of St Andrews, 1996
- Software developer at Metaswitch
  - née Data Connection
- Moved into security role in 2015(ish)
  - Director of Security for Metaswitch
- Metaswitch acquired by Microsoft in 2020
- Moved into Cloud Solution Architect role at end of 2021

CONTENTS

History of the SDL

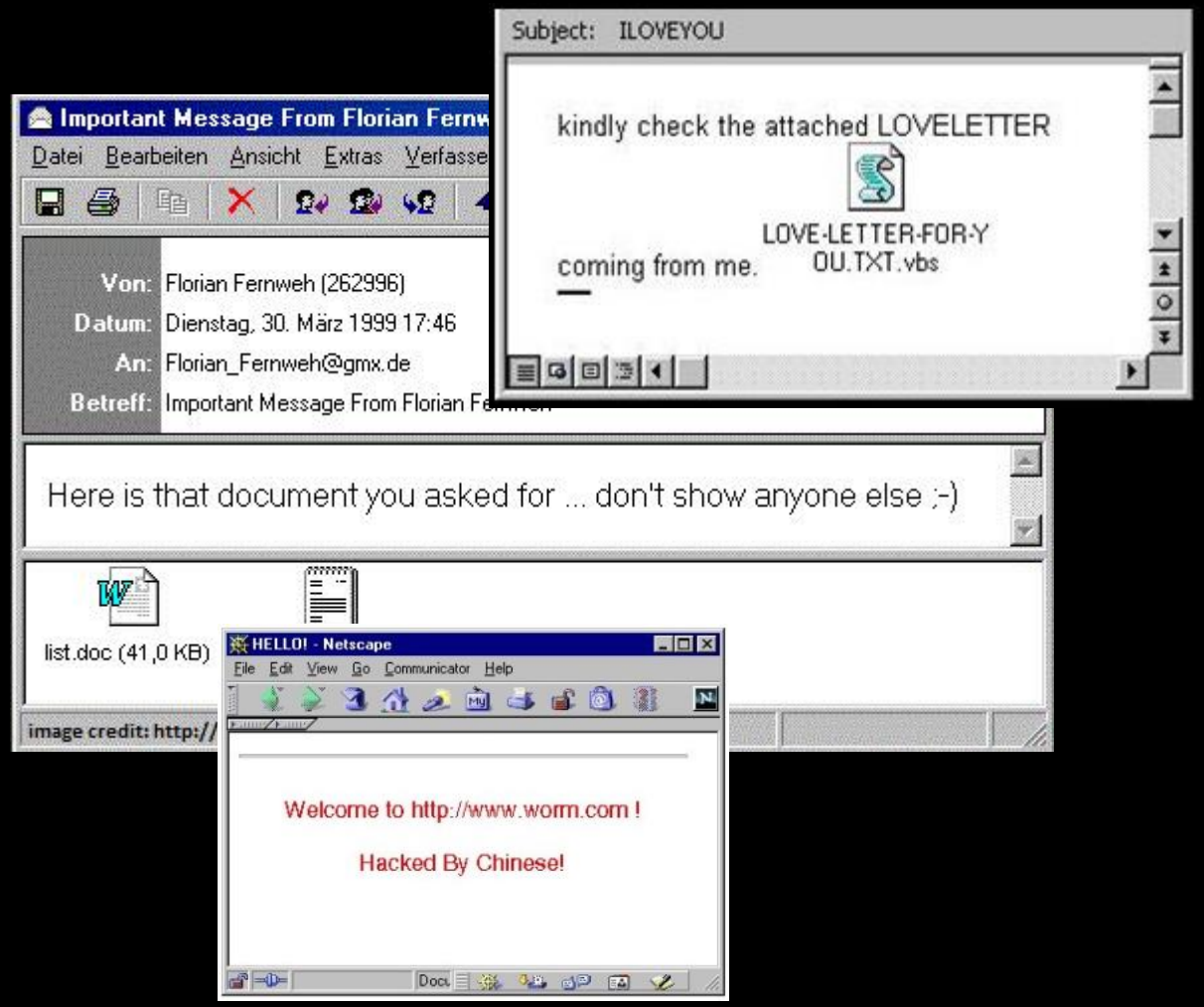
What it all means - Shifting Left

In practice – the SDL at Microsoft


HISTORY

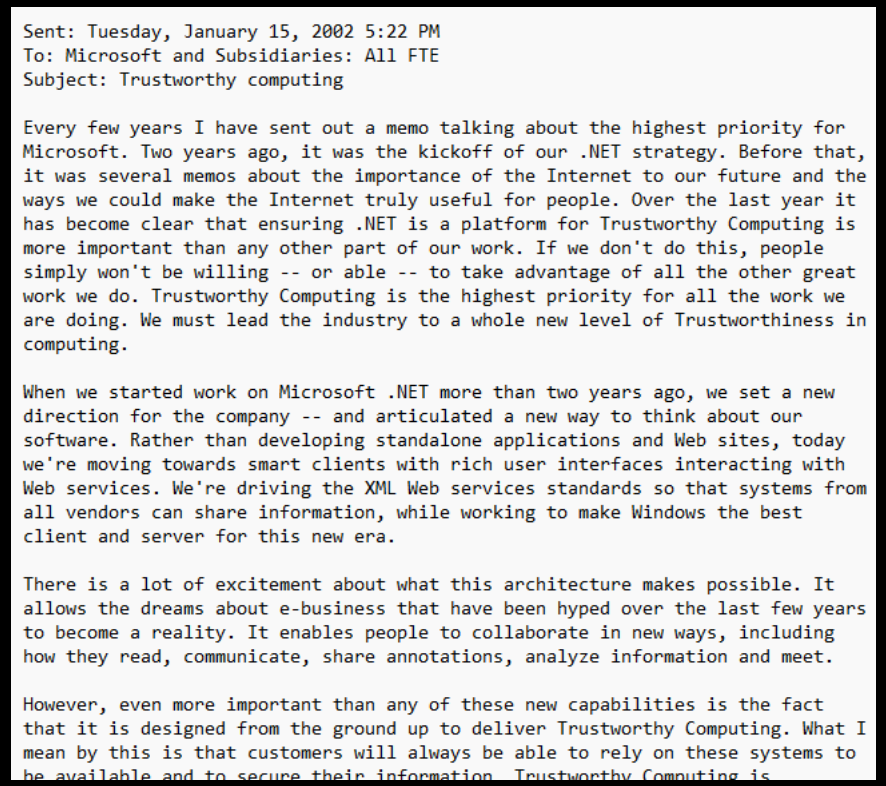
SECURITY NOT ALWAYS MICROSOFT'S SHINING LIGHT

- 1990s massive growth in PC usage
- 1995 – Internet to be a part of all Microsoft Products
- Microsoft was widely regarded as a soft target for viruses and malware
  - Melissa virus – 1999 – email delivered word macro
  - ILOVEYOU – 2000 – email delivered VBS based attack
  - Code Red – 2001 – IIS buffer overflow flaw



SOMETHING HAD TO BE DONE

- January 15 2002 - Bill Gates famous “Trustworthy Computing” memo
- Computing as an essential service like power and water
- Focus on
  - Availability
  - Security
  - Privacy
- Kicked off security reviews of .NET, Windows, etc.



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.

When we started work on Microsoft .NET more than two years ago, we set a new direction for the company -- and articulated a new way to think about our software. Rather than developing standalone applications and Web sites, today we're moving towards smart clients with rich user interfaces interacting with Web services. We're driving the XML Web services standards so that systems from all vendors can share information, while working to make Windows the best client and server for this new era.

There is a lot of excitement about what this architecture makes possible. It allows the dreams about e-business that have been hyped over the last few years to become a reality. It enables people to collaborate in new ways, including how they read, communicate, share annotations, analyze information and meet.

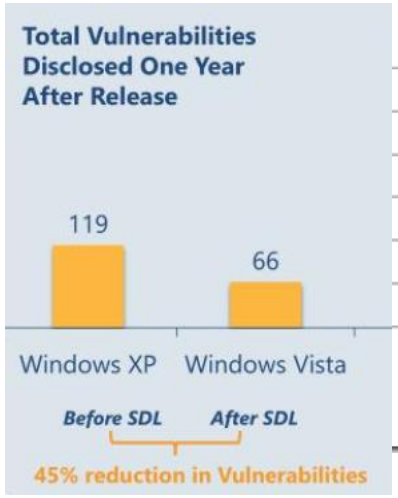
However, even more important than any of these new capabilities is the fact that it is designed from the ground up to deliver Trustworthy Computing. What I mean by this is that customers will always be able to rely on these systems to be available and to secure their information. Trustworthy Computing is

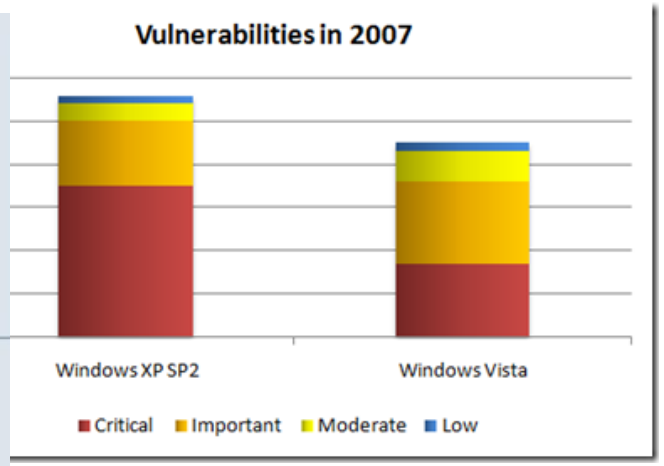
https://news.microsoft.com/2012/01/11/memo-from-bill-gates/

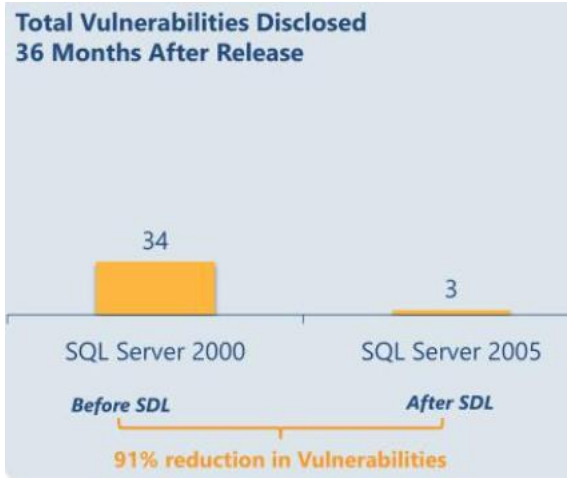
- TwC started culture of building secure, available and reliable software
- Software developers needed something more concrete than “culture”
- Microsoft Security Development Lifecycle (SDL)
  - First released internally in 2004
  - Borne from the development teams as an integral part of the software development process
  - Became mandatory policy for all developers
  - Windows Vista and Office 2007 first products to fully integrate the SDL

TANGIBLE IMPACT

- In first year since launch, Vista had 45% fewer vulnerabilities found than were found in Windows XP's first year
- 24% fewer vulnerabilities in Vista in 2007, compared to XP (which was launched in 2001)
- SQL Server 2005 had 91% fewer vulnerabilities than SQL Server 2000 after 36 months



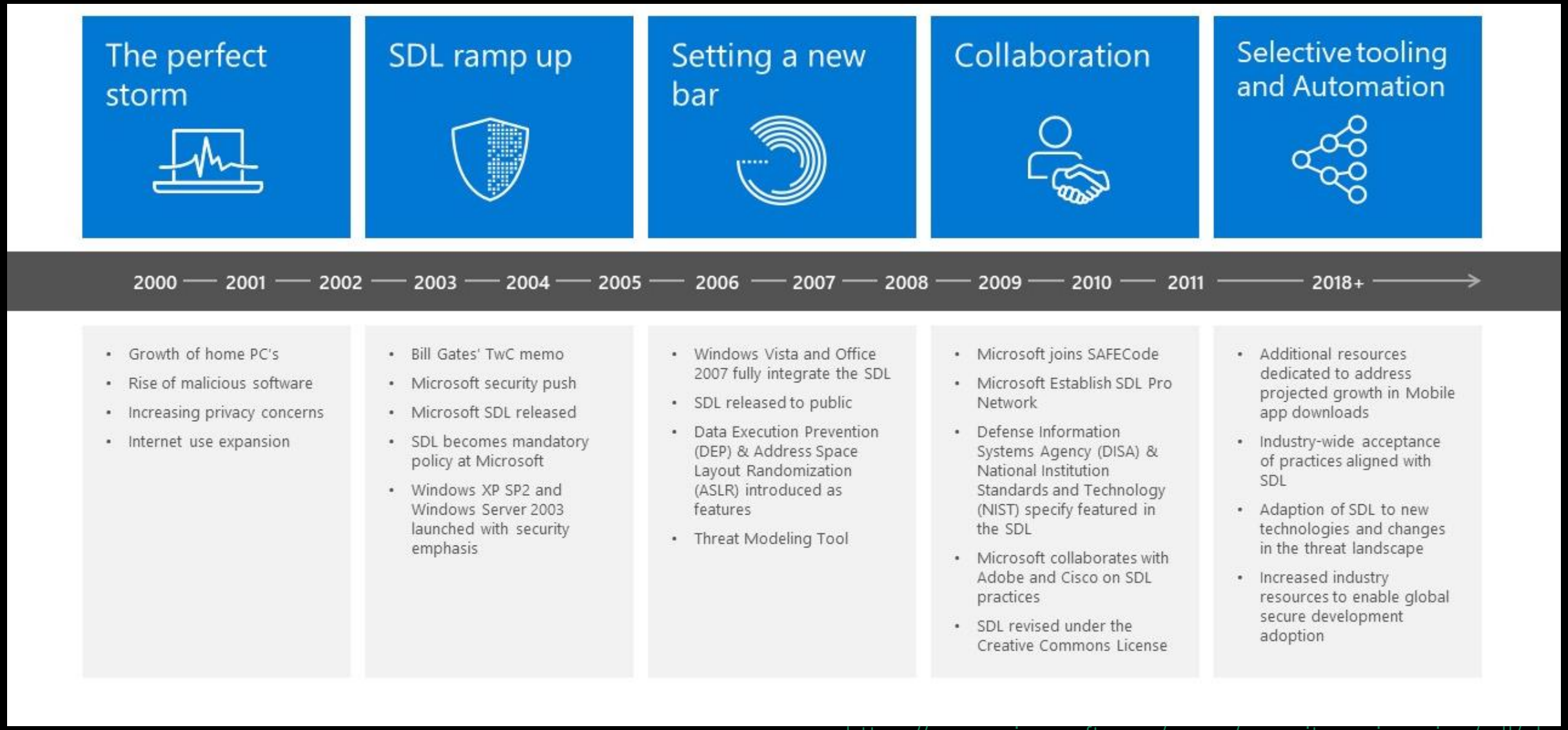




POWER TO THE PEOPLE

- While Windows was the biggest target, by mid-2000s the majority of attacks were against applications
- SDL principles couldn't be kept internal
- Microsoft published the SDL and made a push to evangelise amongst the development community
- SDL has been through continual improvement to address changing landscape
  - Mobile applications
  - Adoption of cloud computing

SDL TIMELINE



https://www.microsoft.com/en-us/securityengineering/sdl/about

SHIFTING LEFT

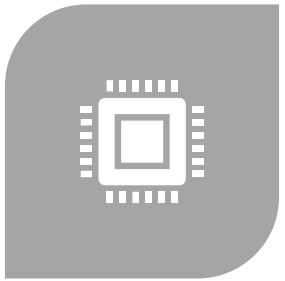
How the SDL turns DevOps into SecDevOps

Cultural change

DEVOPSSEC

SECURITY COMES LAST

A SINGLE SECURITY ASSESSMENT BEFORE DEPLOYMENT OF SILOED APPLICATIONS WILL NOT EVALUATE THE FULL SECURITY OF THE PROGRAMME.



DEVSECOPS

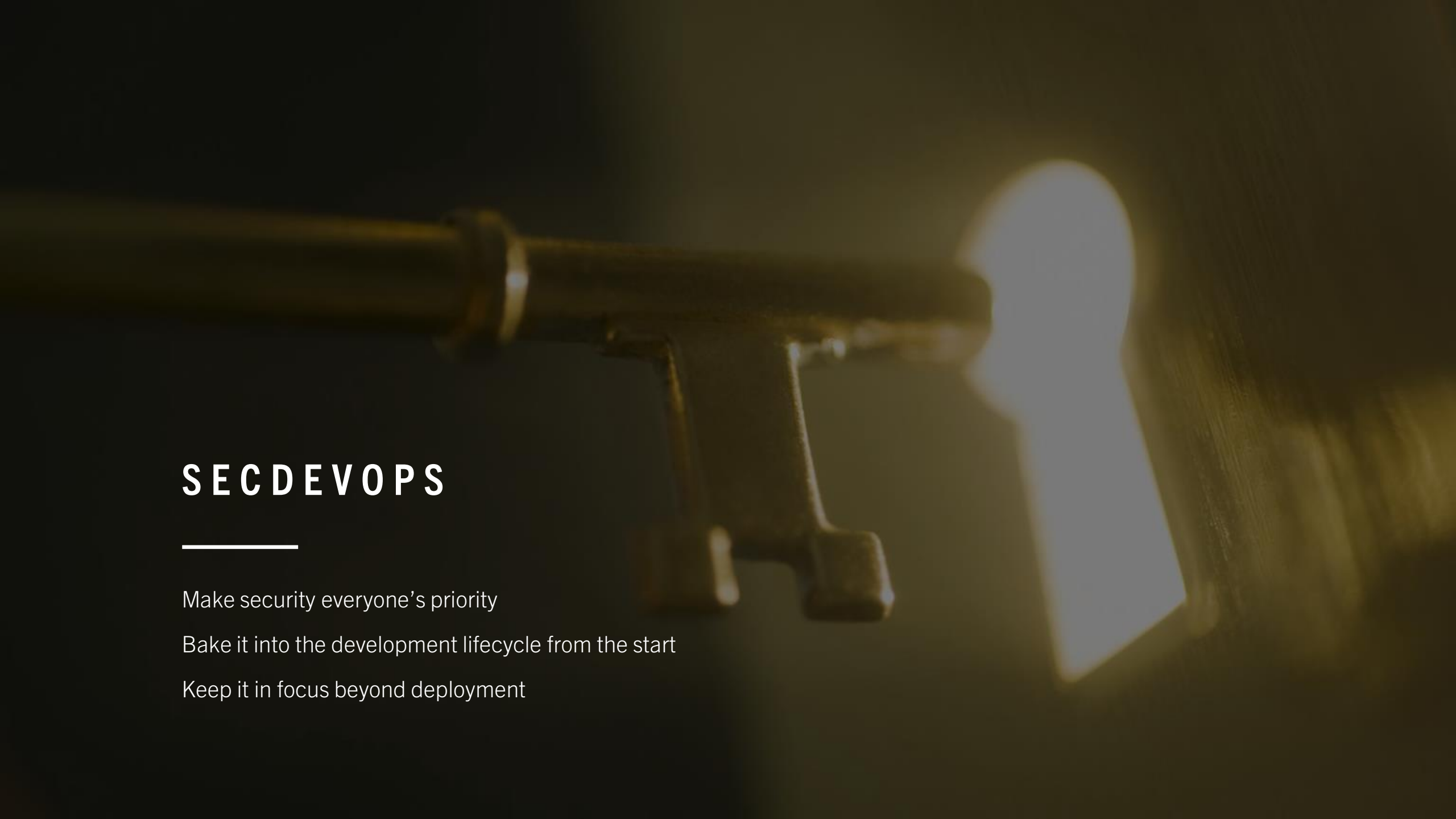
SECURITY BUILT IN

SECURITY BUILT INTO PIPELINES; ENGINEERING TEAMS KNOW THAT SECURITY IS IMPORTANT AND ARE CONTINUOUSLY EVALUATING BEST POSTURE. APPLICATION SECURITY IS STILL AN ENGINEER'S JOB WITH LIMITED VISIBILITY FROM OTHER TEAMS.

SECDEVOPS

SECURITY COMES FIRST

SECURITY IS ELEVATED TO ALL TEAMS, PROVIDING CORE VISIBILITY OVER THE CHALLENGES AND THE BUSINESS PRIORITIES FOR SECURITY FROM DEVELOPER LEVEL TO BUSINESS LEVEL.



SECDEVOPS

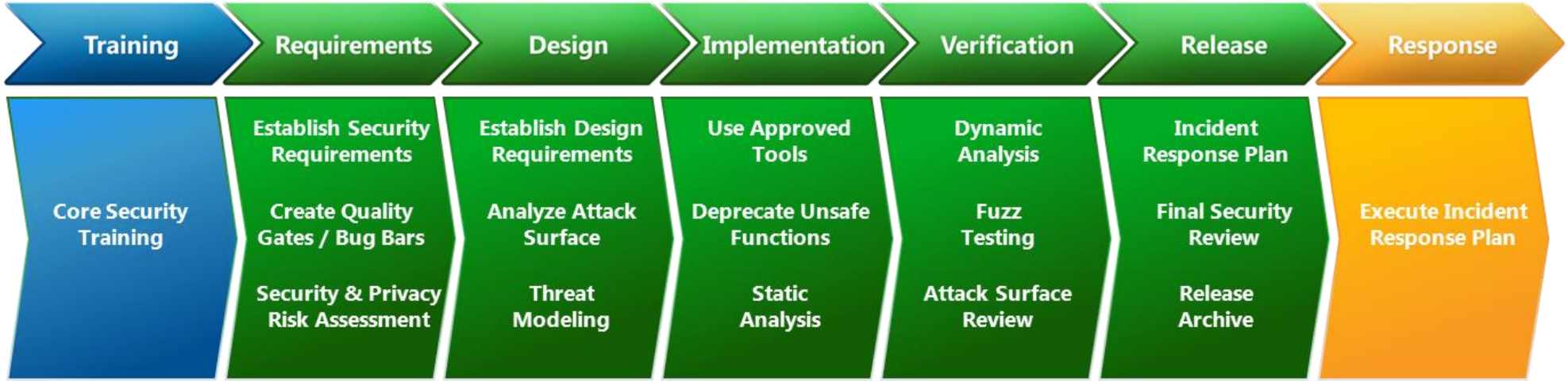
Make security everyone’s priority

Bake it into the development lifecycle from the start

Keep it in focus beyond deployment

WHAT DOES SHIFTING LEFT MEAN?

WHAT DOES SHIFTING LEFT MEAN?



Security Development Lifecycle (SDL)

Provide Security Training

Define and Update Security requirements

Define Metrics and Compliance Reporting

Use Cryptographic Standards

Perform Threat Modelling

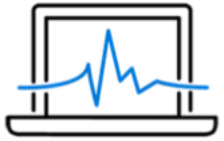
Establish Design Requirements

Manage and Monitor Dependencies

Use Approved Tools

Perform Static Code analysis

Perform Dynamic Analysis Security Testing (DAST)



Perform Penetration Testing

Establish a standard Incident Response Process

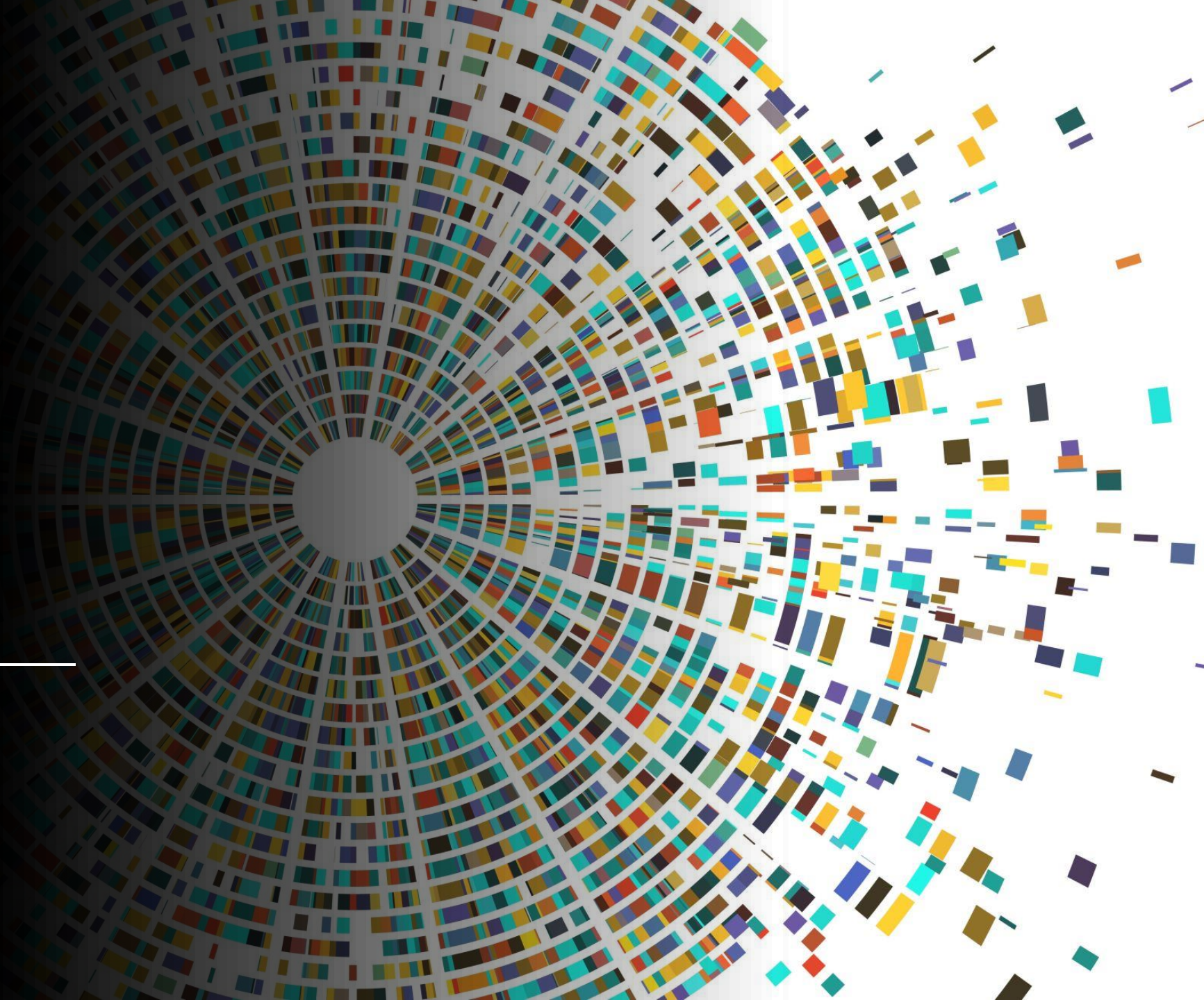
The Security Development Lifecycle (SDL) consists of a set of practices that support security assurance and compliance requirements.

The SDL helps developers build more secure software by reducing the number and severity of vulnerabilities in software, while reducing development cost.

https://www.microsoft.com/en-us/securityengineering/sdl/practices


Security in the Design Process

Provide Security Training



Defining Security Requirements

Design considerations

- Secure coding libraries, software frameworks and standard tools
- Vulnerability scanning and component updates
- Reduce attack surface
- Key management
- Data classification and encryption
- Error handling, logging and alerting
- Identity as the primary security perimeter
- Threat modelling

Identity as the primary security perimeter

- Enforce MFA
  - use a platform that already enforces it for you
- Principle of least privilege
  - Limit access based on user roles
- Reauthenticate for important transactions
- Consider just-in-time access for administrators

Threat Modelling

- Identifies potential security threats to the application
  - Evil brainstorming
- Pulls in reps from each stakeholder group
  - Business / Customer / Security / Dev Team
- Ensures identified threats are “handled”
  - Mitigated – removed or reduced
  - Accepted – signed off by senior management

Threat	Property	Definition	Example
Spoofing	Authentication	Impersonating something or someone else	An example of identity spoofing is illegally accessing and then using another user's authentication information, such as username and password.
Tampering	Integrity	Modifying data or code	Data tampering involves the malicious modification of data. Examples include unauthorized changes made to persistent data, such as that held in a database, and the alteration of data as it flows between two computers over an open network, such as the Internet.
Repudiation	Non-repudiation	Claiming to have not performed an action	Repudiation threats are associated with users who deny performing an action without other parties having any way to prove otherwise—for example, a user performs an illegal operation in a system that lacks the ability to trace the prohibited operations. **Nonrepudiation** refers to the ability of a system to counter repudiation threats. For example, a user who purchases an item might have to sign for the item upon receipt. The vendor can then use the signed receipt as evidence that the user did receive the package.
Information Disclosure	Confidentiality	Exposing information to someone not authorized to see it	Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it—for example, the ability of users to read a file that they were not granted access to, or the ability of an intruder to read data in transit between two computers.
Denial of Service	Availability	Deny or degrade service to users	Denial of service (DoS) attacks deny service to valid users—for example, by making a Web server temporarily unavailable or unusable. You must protect against certain types of DoS threats simply to improve system availability and reliability.
Elevation of Privilege	Authorization	Gain capabilities without proper authorization	In this type of threat, an unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the entire system. Elevation of privilege threats include those situations in which an attacker has effectively penetrated all system defenses and become part of the trusted system itself, a dangerous situation indeed.

STRIDE Threat model

Threat	Property	Definition	Example
Spoofing	Authentication	Impersonating something or someone else	An example of identity spoofing is illegally accessing and then using another user's authentication information, such as username and password.
Tampering	Integrity	Modifying data or code	Data tampering involves the malicious modification of data. Examples include unauthorized changes made to persistent data, such as that held in a database, and the alteration of data as it flows between two computers over an open network, such as the Internet.
Repudiation	Non-repudiation	Claiming to have not performed an action	Repudiation threats are associated with users who deny performing an action without other parties having any way to prove otherwise—for example, a user performs an illegal operation in a system that lacks the ability to trace the prohibited operations. **Nonrepudiation** refers to the ability of a system to counter repudiation threats. For example, a user who purchases an item might have to sign for the item upon receipt. The vendor can then use the signed receipt as evidence that the user did receive the package.
Information Disclosure	Confidentiality	Exposing information to someone not authorized to see it	Information disclosure threats involve the exposure of information to individuals who are not supposed to have access to it—for example, the ability of users to read a file that they were not granted access to, or the ability of an intruder to read data in transit between two computers.
Denial of Service	Availability	Deny or degrade service to users	Denial of service (DoS) attacks deny service to valid users—for example, by making a Web server temporarily unavailable or unusable. You must protect against certain types of DoS threats simply to improve system availability and reliability.
Elevation of Privilege	Authorization	Gain capabilities without proper authorization	In this type of threat, an unprivileged user gains privileged access and thereby has sufficient access to compromise or destroy the entire system. Elevation of privilege threats include those situations in which an attacker has effectively penetrated all system defenses and become part of the trusted system itself, a dangerous situation indeed.

Development Phase

Implementation considerations

- Perform and require code reviews
- Static code analysis
- Validate every input and sanitize every output
  - Including file uploads
- Segregate production and test data

Verification considerations

- Keep on top of application dependencies
- Test the application in an operating state
  - DAST – Dynamic application security testing
  - Penetration testing
- Fuzz testing
- Security verification testing
- Attack surface review

Deployment

Release

- Consider your infrastructure
  - e.g. deploy a Web Application Firewall
- Run load tests and check performance
- Create an incident response plan
- Final security review
- Sign off and archive the release

Post-release

- Monitor
  - Application performance
  - Security logs and events
- Execute the incident response plan
- Look for continual improvement

IN PRACTICE

INTERNAL SECURITY TRAINING

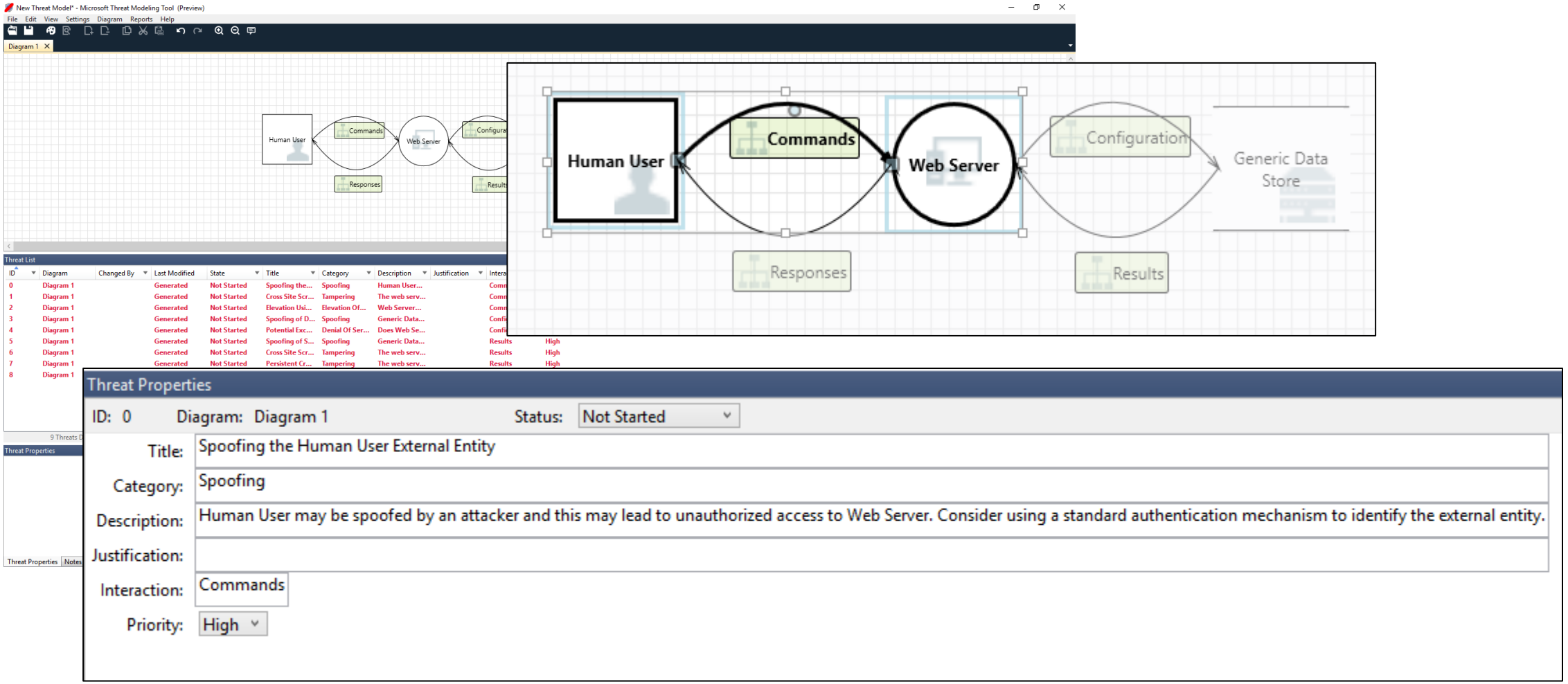
- Ongoing program of developer led security
- Providing training and building a culture of security
- Combination of widely delivered training briefing, capture the flag contests and shorter “lightning” sessions
- Participation is mandatory for all in development and technical roles

- The SDL has been encoded into over 90 specific requirements
- For example
  - Applications must adhere to the least privilege principle
  - Secrets must not be present in code, documentation, telemetry or pipelines
  - Service specific DDoS mitigations must be implemented
  - etc…
- Each requirement is backed up with more details, including
  - Guidance on applicability
  - Technical information to help with implementation
- We've also developed tools to help determine the applicability on given code

THREAT MODELLING

- Required part of the SDL process
- Starts with a list of use cases for the system
- Adds “data flow diagrams” that show the
  - High level view of components and data flows
  - Entry points
  - Trust boundaries
- Brings in any assets to be protected (databases, storage accounts, keys, servers, etc.)
- From this develops a list of threats, mitigations and accepted risks

Threat modelling



https://docs.microsoft.com/en-us/azure/security/develop/threat-modeling-tool

EMBEDDING SECURITY IN THE WORKFLOW

Automation is good…

- … automated processes can't get forgotten
  - Credential scanning
  - Static Application Security Testing (SAST) in the pipeline
  - Antivirus
  - Package vulnerability scanning
- Introduction of a major security issue needs to break the build

… but mustn't kill developer efficiency

- Security has to go into the workflow efficiently
  - parallel security pipeline
  - tune tools which always generate false positives
- Push tooling further left if possible
  - Use a package manager that *only* serves approved packages
  - Add security testing into unit test
    - Negative unit testing – test how we handle unexpected inputs
  - Use pre-commit checks for things like credential scanning

AUTOMATION AT MICROSOFT

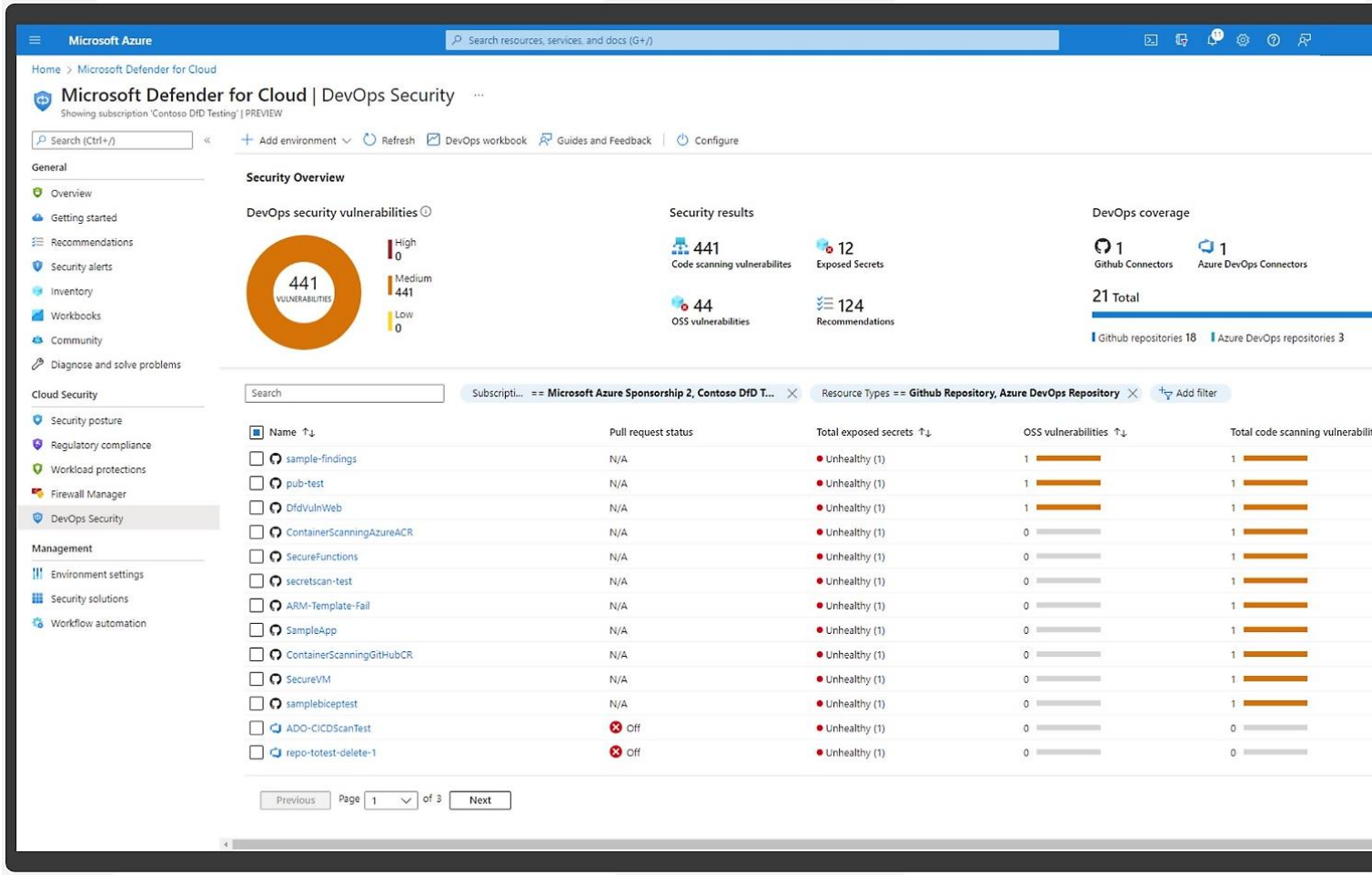
- One engineering system across all development teams
- Based on Azure DevOps
- Includes extensions that pull together a broad set of security tasks into the pipeline
- Also provides workflows to ensure things like Threat model review and sign off are complete
  - … and reviewed at least every six months

Security Code Analysis tools

- Credscan
  - Microsoft developed tool to identify credential leaks in source code and configuration files
- BinSkim - https://github.com/Microsoft/binskim
  - Lightweight scanner to validate security related compiler and linker settings
- Bandit - https://github.com/PyCQA/bandit
  - Python security linter
- ESLint - https://github.com/eslint/eslint
  - Javascript security linter
- Terrascan - https://github.com/tenable/terrascan
  - Static analysis for IaaC
  - Terraform, AWS CFT, Azure ARM, dockerfiles, etc
- Trivy - https://github.com/aquasecurity/trivy
  - Vulnerability detection across multiple platforms including k8s, Docker, Terraform

Defender for DevOps

- Brings visibility of code security from across repos and sources into one pane of glass
- Same tools that the security team use to monitor and protect the infrastructure



https://www.microsoft.com/en-us/security/business/cloud-security/microsoft-defender-devops

Thank You!