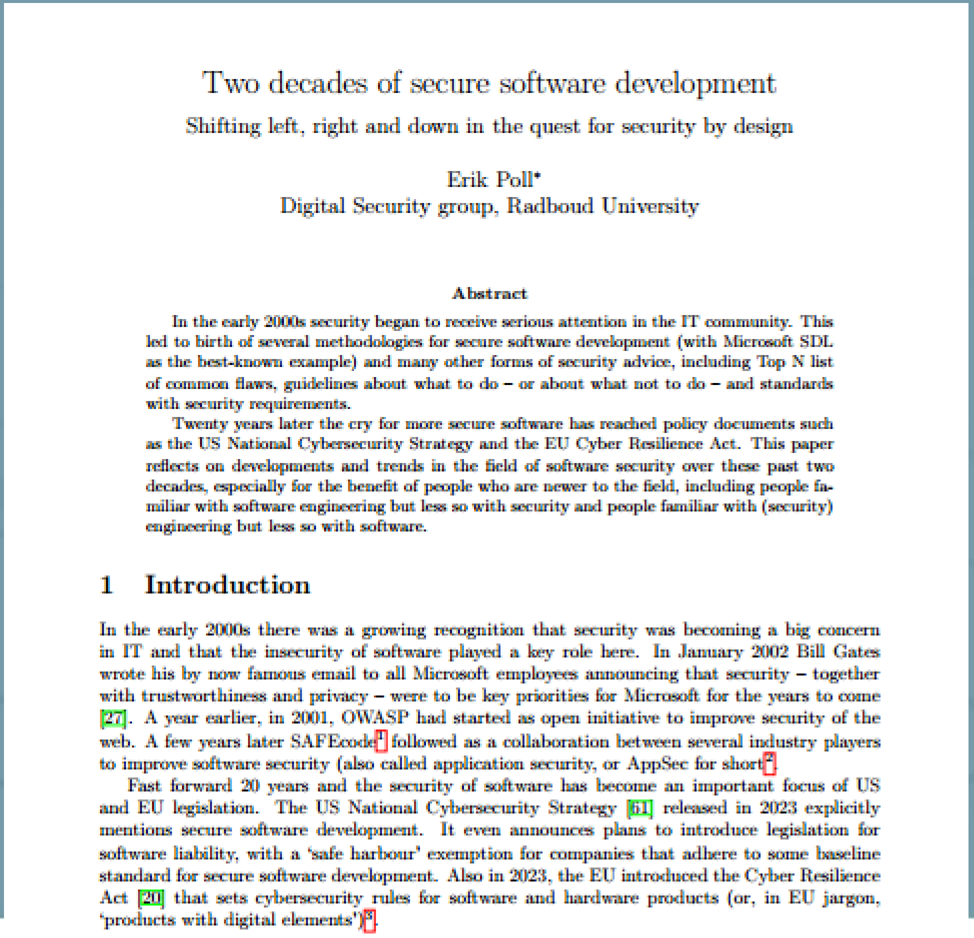
Twenty years of secure software development

What have we learned?

What's changed?

in our quest for security-by-design

by shifting left, right and down



Two decades of secure software development

Shifting left, right and down in the quest for security by design

Erik Poll*
Digital Security group, Radboud University

Abstract

In the early 2000s security began to receive serious attention in the IT community. This led to birth of several methodologies for secure software development (with Microsoft SDL as the best-known example) and many other forms of security advice, including Top N list of common flaws, guidelines about what to do – or about what not to do – and standards with security requirements.

Twenty years later the cry for more secure software has reached policy documents such as the US National Cybersecurity Strategy and the EU Cyber Resilience Act. This paper reflects on developments and trends in the field of software security over these past two decades, especially for the benefit of people who are newer to the field, including people familiar with software engineering but less so with security and people familiar with (security) engineering but less so with software.

1 Introduction

In the early 2000s there was a growing recognition that security was becoming a big concern in IT and that the insecurity of software played a key role here. In January 2002 Bill Gates wrote his by now famous email to all Microsoft employees announcing that security – together with trustworthiness and privacy – were to be key priorities for Microsoft for the years to come [27]. A year earlier, in 2001, OWASP had started as open initiative to improve security of the web. A few years later SAFEcode[1] followed as a collaboration between several industry players to improve software security (also called application security, or AppSec for short[2].

Fast forward 20 years and the security of software has become an important focus of US and EU legislation. The US National Cybersecurity Strategy [61] released in 2023 explicitly mentions secure software development. It even announces plans to introduce legislation for software liability, with a 'safe harbour' exemption for companies that adhere to some baseline standard for secure software development. Also in 2023, the EU introduced the Cyber Resilience Act [20] that sets cybersecurity rules for software and hardware products (or, in EU jargon, 'products with digital elements')[3].

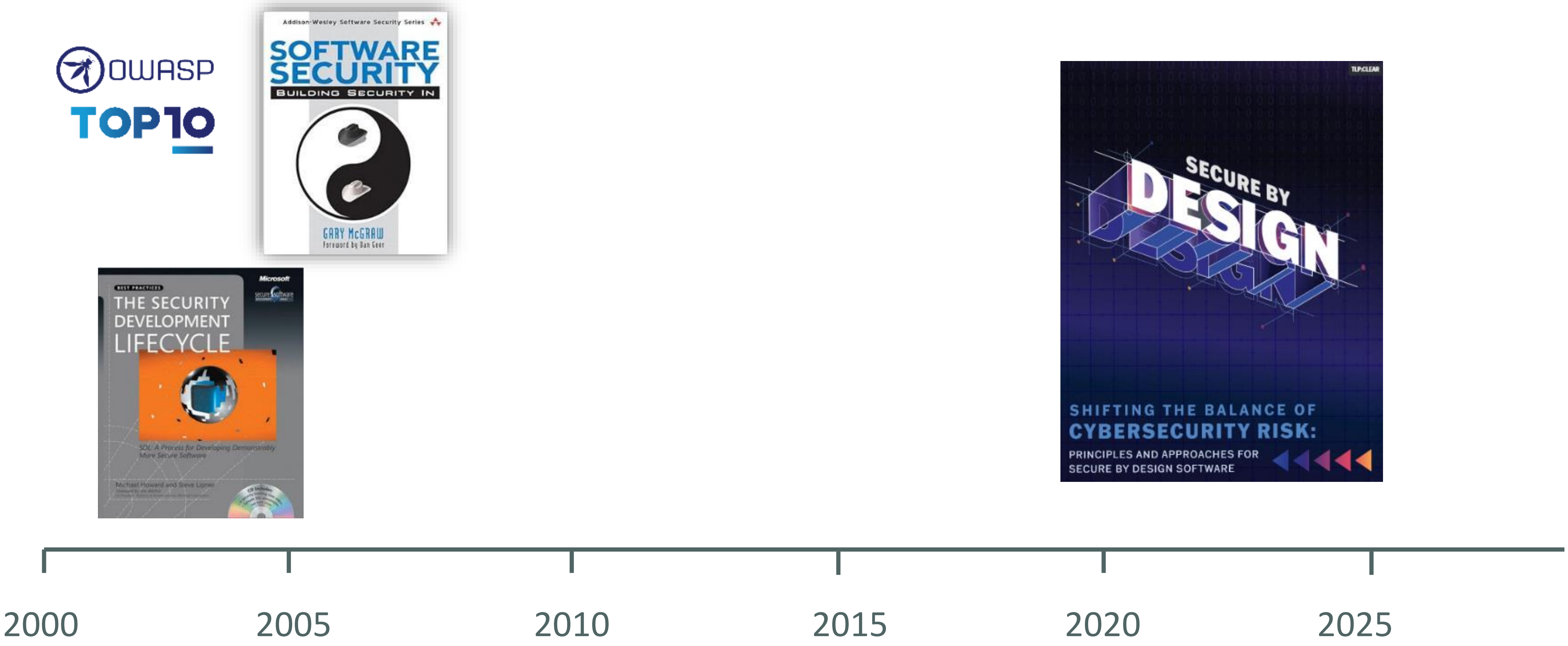
Caveat

This talk is about ***software engineering***
not about ***systems engineering***

But: hardware is simply software that you cannot update

or: hardware is software that you *do not need to* update?

A brief history of software security


OWASP
TOP10
Addison-Wesley Software Security Series
SOFTWARE SECURITY
BUILDING SECURITY IN
GARY McGRAW
Foreword by Dan Geer
BEST PRACTICES
THE SECURITY DEVELOPMENT LIFECYCLE
Microsoft
secure software
SDL: A Process for Developing Demonstrably More Secure Software
Michael Howard and Steve Lipner
CD Includes
TLP:CLEAR
SECURE BY DESIGN
SHIFTING THE BALANCE OF CYBERSECURITY RISK:
PRINCIPLES AND APPROACHES FOR SECURE BY DESIGN SOFTWARE
2000
2005
2010
2015
2020
2025

LOTS of (different kinds of) security advice

Hard to see the forests for the trees

LOTS of (different kinds of) security advice

Hard to see the forests for the trees

- forest of *vulnerabilities* (CVEs) with CVSS, KEV, EPSS, CPR, SSVC, ... to navigate it
- forest of *vulnerability categories* (CWEs) OWASP Top 10, CWE Top 25 & Top 1000

  maybe here lies the difference between *systems* engineering and *software* engineering?
- forest of *secure development technologies* SDL, SAMM, BSIMM, NIST SSDF, ...

  focused on the **process**
- forest of *security tools* DAST (incl. fuzzing), SAST, SCA, SecretScanning, ...
- forest of *security requirements* OWASP ASVS, MASVS, SVCS, ...

  focused on the **product**
- some *good practices and design patterns*

Methodologies (for the software engineering process)

- **Microsoft SDL** incl. DevSecOps
- **OWASP SAMM**
- **NIST SSDF** Secure Software Development Framework, 2022
- **ISO/IEC 27034** for application security
- **ISO/IEC 62433** for industrial automation and control systems
- **NIST CSF** Cyber Security Framework
- NIST IR 7628 for smart grid security
- Grip on SSD by CIP-overheid.nl
- SAFEcode Fundamental practices for Secure Software Development
- BSA Framework for Secure Software Development
- ...

Which of these do you use? Which important ones am I missing?

Maturity models

Many secure development methodologies, each can get quite complex

Introducing one, and then improving by *shifting left*, is a lengthy process

Hence: *maturity models for 1) measuring and 2) comparing*

- **BSIMM** by Synopsis, since 2009
  - lists 126 activities grouped in 12 practices across 4 domains
- **OWASP SAMM**

Has anyone of you ever used such a maturity model?

Tools

Secure software development methodologies can be supported by tools, esp.

- **DAST (incl fuzzing)**

- **SAST**
  - eg Coverity, Fortify, Checkmarx, VeraCode, SonarCube, ...

Which (type of) tools do you use?

Changes in software engineering over the past 20 years

1. Agile & DevOps

   Some security activities trickier; more need to ***shift left*** and ***automate***

2. Supply chain risks

   Huge rise in the use of 3rd party code thanks to github, sourceforge, PyPI, NPM, Maven, ...

   Risks of *accidental flaws* and *deliberate backdoors*

   Hence: 1) SCA tools and 2) SBOMs

3. Risks of leaking credentials

   Many more credentials around: for SaaS APIs, code repos, cloud environments, CI/CD pipelines with Jira, Confluence, Jenkins, Azure DevOps, Slack, Teams, ...

   Risks of *leaking these secrets*

   Hence 1) Secret Scanning tools eg TruffleHog, Nosey Parker, ABN-AMRO Repository Scanner and maybe 2) SaaSBOMs?

Which of these do you use? Which important changes am I missing?

Security advice for the product (as opposed to process)

Very different kinds of security advice, some very specific to certain tech stack/application type

- Lists of common vulnerabilities, eg. OWASP Top 10, CWE Top 25, KEV Top 10, ...
  - Also Mobile Top 10, API Top 10, Top 10 for LLM applications, ...
- Coding guidelines, eg. SEI/CERT guidelines for C and for C++
- Standards with security requirements & controls, eg.
  - OWASP ASVS (Application Security Verification Standard)
    can be used as metric, as guidance, or in procurement
  - SVCS (Software Component Verification Standard)

Some standards (eg ISO/IEC 62443) combine requirements for process & for product.

Which of these do you use? Which important ones am I missing?

The forest of security vulnerability categories

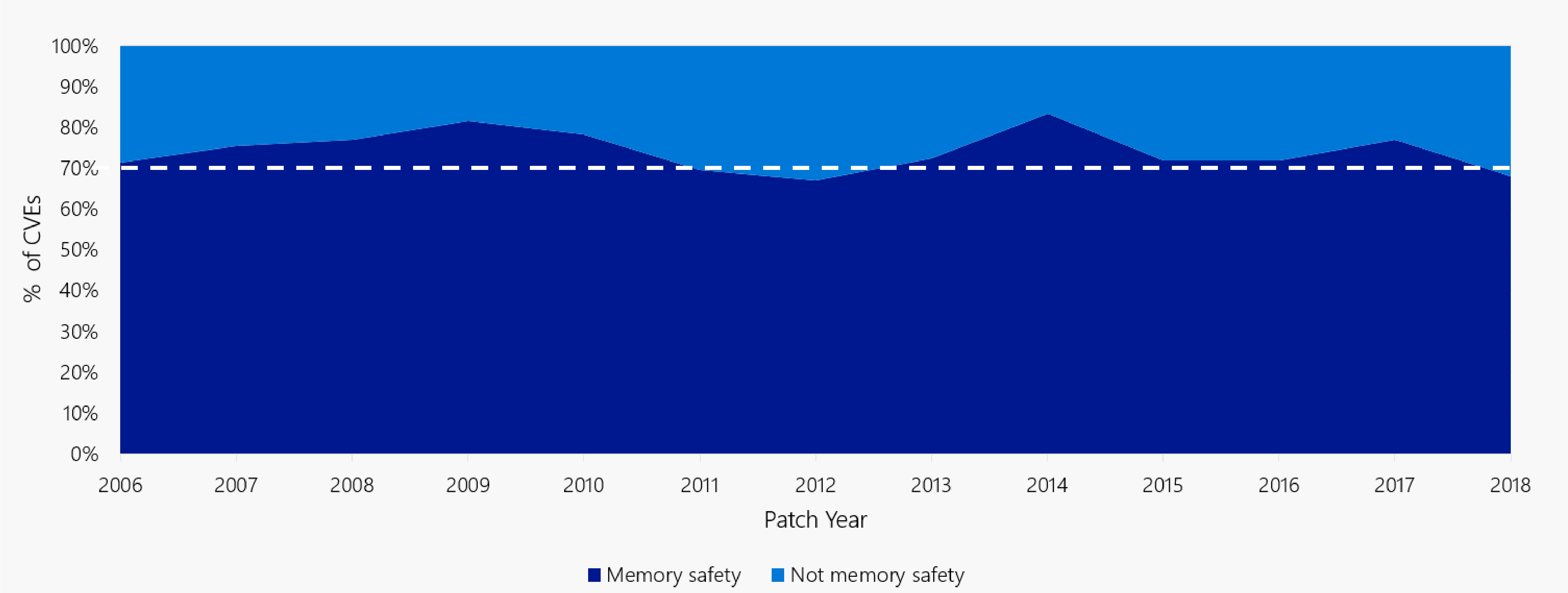
There are many types of security vulnerability:

- the CWE classification includes over a 1000 categories

But: the bulk of them come down to only three kinds of problems:

1) **memory corruption**
2) **input handling,** esp. **injection attacks**
3) **access control** (incl. authorization, authentication, monitoring, and response)

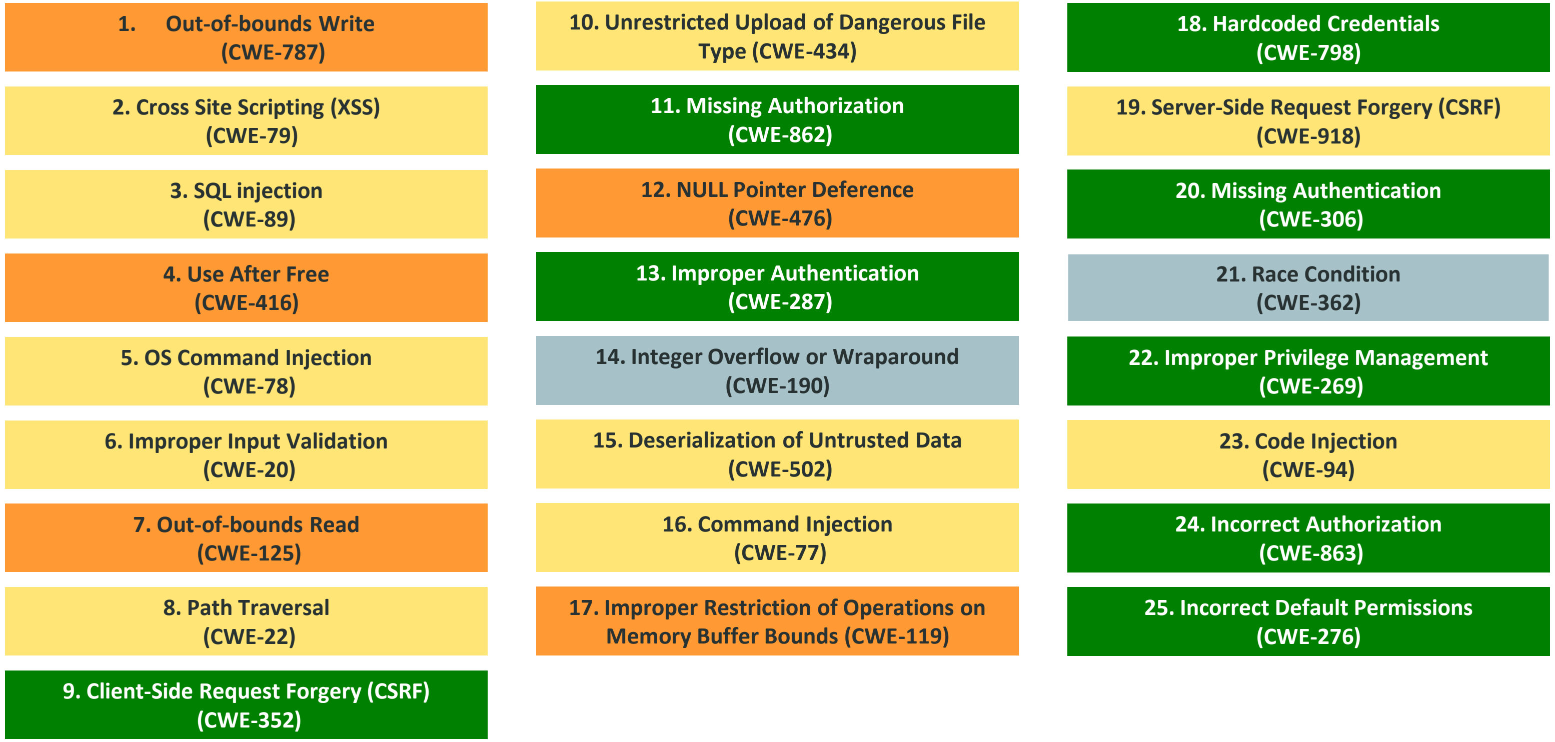
A known known: memory corruption bugs at Microsoft 2006-2018



[Source: https://msrc-blog.microsoft.com/2019/07/16/a-proactive-approach-to-more-secure-code
and *“Trends, challenge, and shifts in software vulnerability mitigation*”, presentation by Matt Miller at BlueHat IL 2019]

Another known known: CWE Top 25

with memory corruption, access control, and input handling



🟧 1. Out-of-bounds Write (CWE-787)

🟨 2. Cross Site Scripting (XSS) (CWE-79)

🟨 3. SQL injection (CWE-89)

🟧 4. Use After Free (CWE-416)

🟨 5. OS Command Injection (CWE-78)

🟨 6. Improper Input Validation (CWE-20)

🟧 7. Out-of-bounds Read (CWE-125)

🟨 8. Path Traversal (CWE-22)

🟩 9. Client-Side Request Forgery (CSRF) (CWE-352)

🟨 10. Unrestricted Upload of Dangerous File Type (CWE-434)

🟩 11. Missing Authorization (CWE-862)

🟧 12. NULL Pointer Deference (CWE-476)

🟩 13. Improper Authentication (CWE-287)

⬜ 14. Integer Overflow or Wraparound (CWE-190)

🟨 15. Deserialization of Untrusted Data (CWE-502)

🟨 16. Command Injection (CWE-77)

🟧 17. Improper Restriction of Operations on Memory Buffer Bounds (CWE-119)

🟩 18. Hardcoded Credentials (CWE-798)

🟨 19. Server-Side Request Forgery (CSRF) (CWE-918)

🟩 20. Missing Authentication (CWE-306)

⬜ 21. Race Condition (CWE-362)

🟩 22. Improper Privilege Management (CWE-269)

🟨 23. Code Injection (CWE-94)

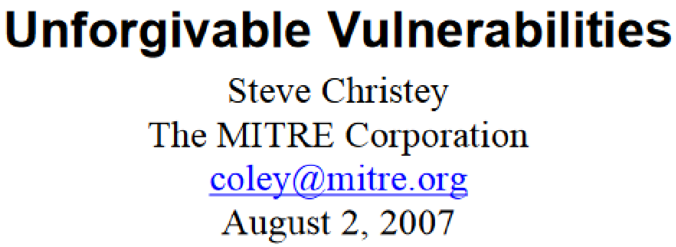
🟩 24. Incorrect Authorization (CWE-863)

🟩 25. Incorrect Default Permissions (CWE-276)

Another known known: CWE Top 25

with memory corruption, access control, and input handling

1. Out-of-bounds Write (CWE-787)
2. Cross Site Scripting (XSS) (CWE-79)
3. SQL injection (CWE-89)
4. Use After Free (CWE-416)
5. OS Command Injection (CWE-78)
6. Improper Input Validation (CWE-20)
7. Out-of-bounds Read (CWE-125)
8. Path Traversal (CWE-22)
9. Client-Side Request Forgery (CSRF) (CWE-352)
10. Unrestricted Upload of Dangerous File Type (CWE-434)
11. Missing Authorization (CWE-862)
12. NULL Pointer Deference (CWE-476)
13. Improper Authentication (CWE-287)
14. Integer Overflow or Wraparound (CWE-190)
15. Deserialization of Untrusted Data (CWE-502)
16. Command Injection (CWE-77)
17. Improper Restriction of Operations on Memory Buffer Bounds (CWE-119)
18. Hardcoded Credentials (CWE-798)
19. Server-Side Request Forgery (CSRF) (CWE-918)
20. Missing Authentication (CWE-306)
21. Race Condition (CWE-362)
22. Improper Privilege Management (CWE-269)
23. Code Injection (CWE-94)
24. Incorrect Authorization (CWE-863)
25. Incorrect Default Permissions (CWE-276)



Unforgivable Vulnerabilities

Steve Christey
The MITRE Corporation
coley@mitre.org
August 2, 2007

Shifting down

Best way to shift left: *shift down*

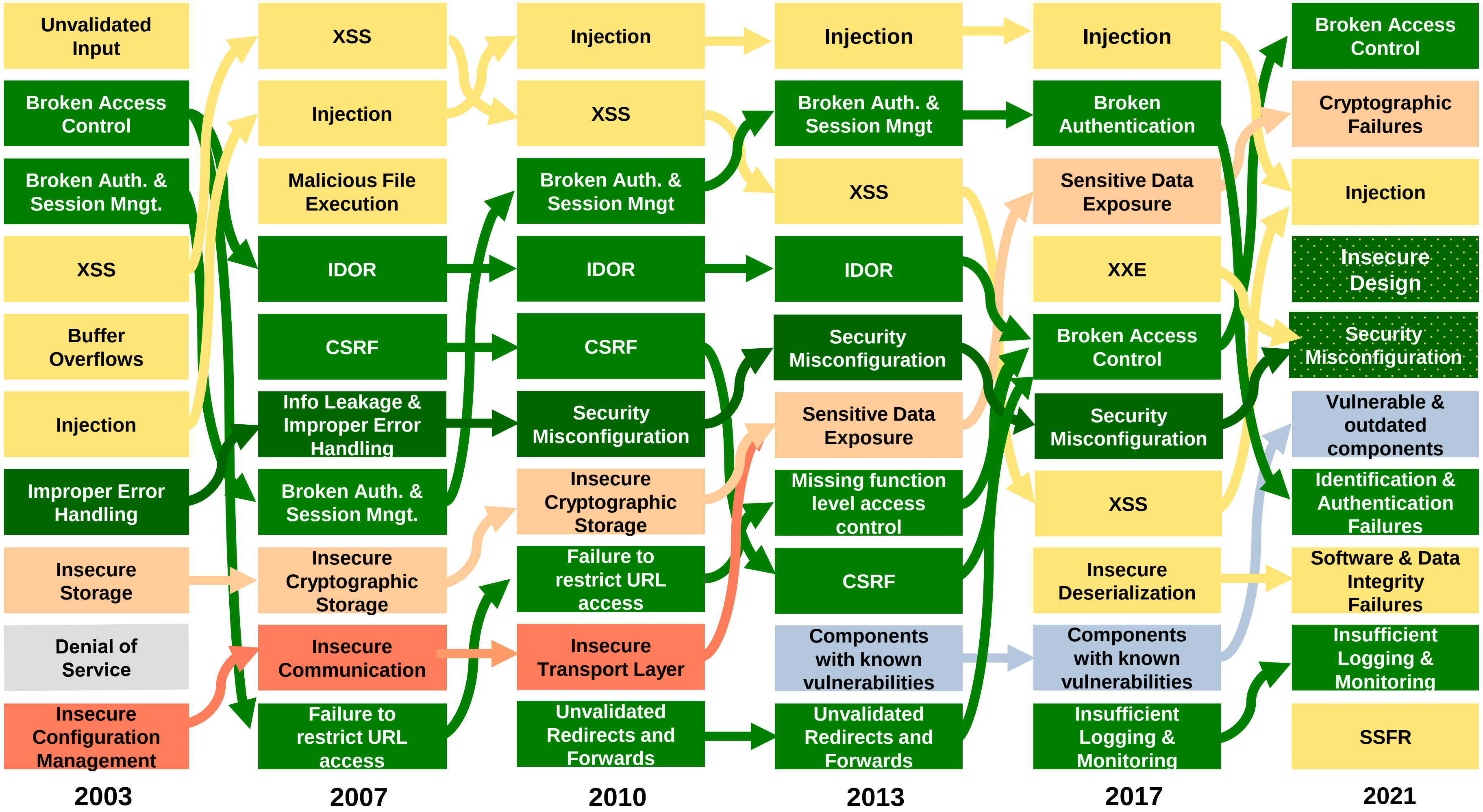
ie. address security lower down in technology stack, in platform or APIs

Examples

- safe(r) programming languages, notably for memory-safety
- safer APIs, that are less prone to injection attacks
  - eg using 'Safe Builder' approach leveraging typing
- built-in security mechanisms in platforms

  eg built-in session mechanism that resistant to CSRF
- LangSec, to prevent input handling problems by paying attention to (parsing of) input languages

OWASP Top 25 over the past 20 years



2003	2007	2010	2013	2017	2021
Unvalidated Input	XSS	Injection	Injection	Injection	Broken Access Control
Broken Access Control	Injection	XSS	Broken Auth. & Session Mngt	Broken Authentication	Cryptographic Failures
Broken Auth. & Session Mngt.	Malicious File Execution	Broken Auth. & Session Mngt	XSS	Sensitive Data Exposure	Injection
XSS	IDOR	IDOR	IDOR	XXE	Insecure Design
Buffer Overflows	CSRF	CSRF	Security Misconfiguration	Broken Access Control	Security Misconfiguration
Injection	Info Leakage & Improper Error Handling	Security Misconfiguration	Sensitive Data Exposure	Security Misconfiguration	Vulnerable & outdated components
Improper Error Handling	Broken Auth. & Session Mngt.	Insecure Cryptographic Storage	Missing function level access control	XSS	Identification & Authentication Failures
Insecure Storage	Insecure Cryptographic Storage	Failure to restrict URL access	CSRF	Insecure Deserialization	Software & Data Integrity Failures
Denial of Service	Insecure Communication	Insecure Transport Layer	Components with known vulnerabilities	Components with known vulnerabilities	Insufficient Logging & Monitoring
Insecure Configuration Management	Failure to restrict URL access	Unvalidated Redirects and Forwards	Unvalidated Redirects and Forwards	Insufficient Logging & Monitoring	SSFR

Conclusion

- The good news:

  There is a lot of – different kinds of – security guidance out there!

- Bad news:

  It is hard to see the forest for the trees

  Which development methodology, (type of) tools, set of security requirements, ... to use?

  Little hard evidence statistics to use as basis for decisions here.

Nice initiative to combat some of the confusion: OpenCRE (Open Common Requirement Enumeration, https://www.opencre.org) to link all the standards, frameworks and guidelines