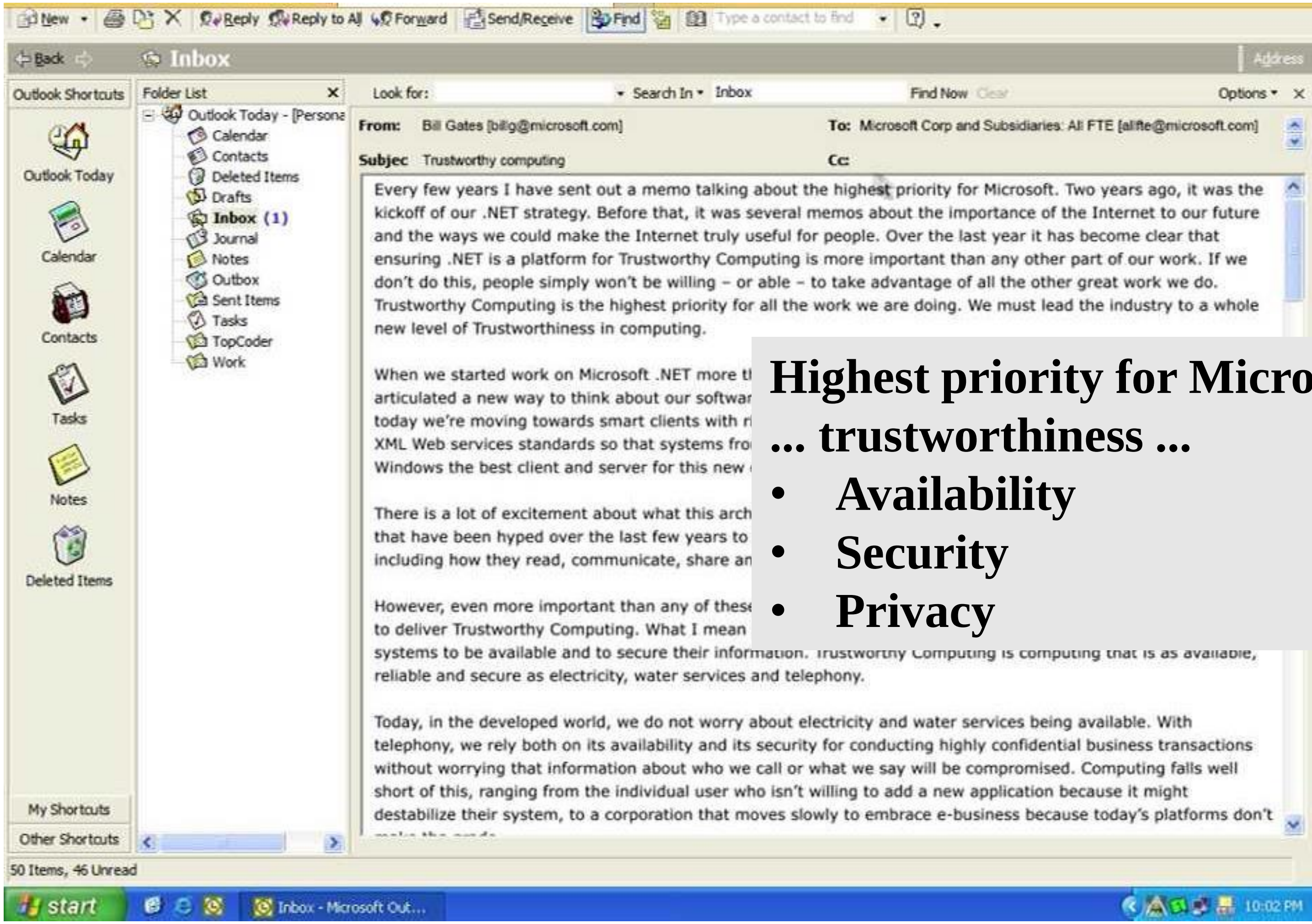
Twenty years of secure software development

Erik Poll

Digital Security

Radboud University Nijmegen

A brief history of software security: January 2002



Highest priority for Microsoft:
... trustworthiness ...

- **Availability**
- **Security**
- **Privacy**

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

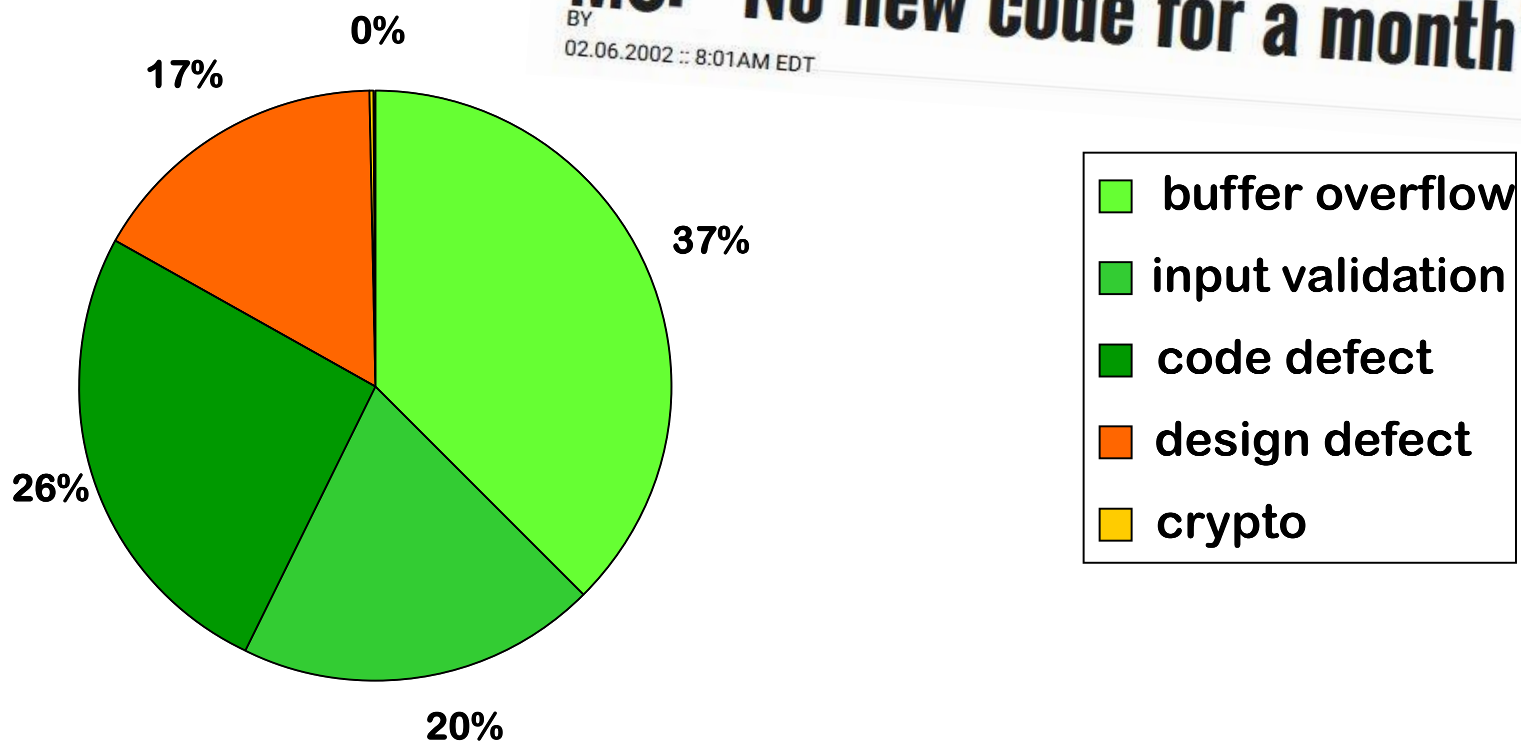
Flaws found in Microsoft's first security bug fix month

NEWS

MS: "No new code for a month"

BY

02.06.2002 :: 8:01AM EDT



Twenty years later

EU & US announce regulation for software security

(Sept 2022: proposed regulation
to complement NIS2 framework)

(May 2023)

https://digital-strategy.ec.europa.eu/en/policies/cyber-resilience-act

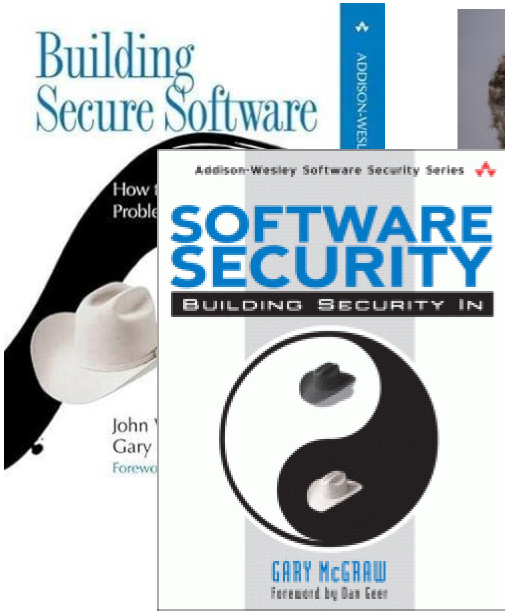
https://www.whitehouse.gov/briefing-room/statements-releases/2023/03/02/fact-sheet-biden-harris-administration-announces-national-cybersecurity-strategy

Software Security

- Software is <u>*the*</u> cause of cybersecurity problems

- Software security = *everything* we can do to *reduce* or *manage the risks* of security problems involving software
  - covers all aspects of software engineering (from requirement engineering & initial design to static analysis, testing, monitoring & patching), programming languages, ‘platforms’ / tech stacks, protocols, APIs, ...
  - aka AppSec (Application Security),
    but AppSec can have narrower meaning

Early 2000s



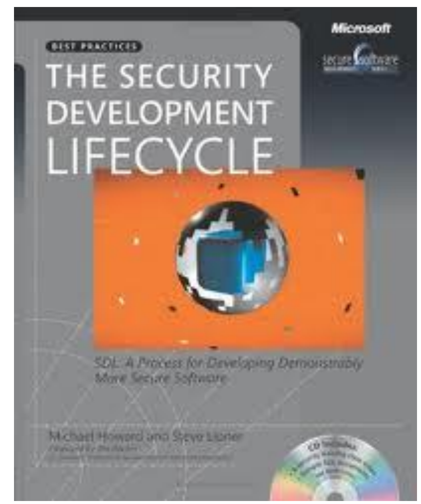
Gary McGraw

`Building Security In' aka
Cigital Touchpoints

Software [In]security: Nine Things Everybody Does: Software Security Activities from the BSIMM

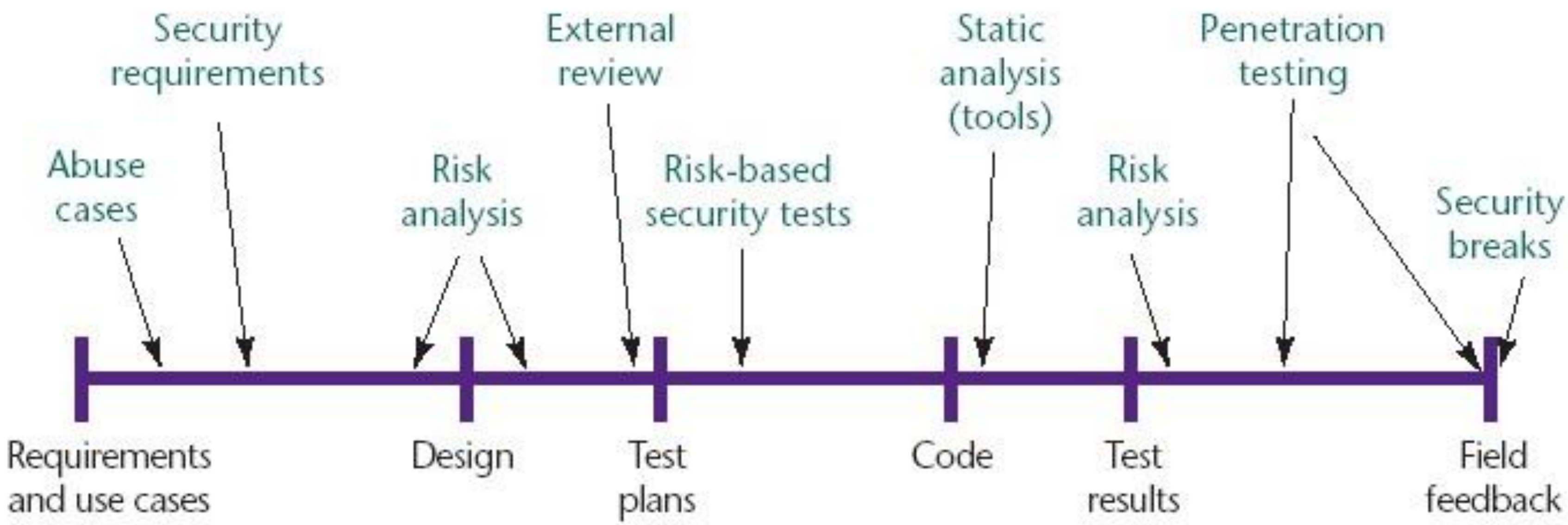
BSIMM by Synopsis

CLASP and SAMM by OWASP



Microsoft SDL (2004)

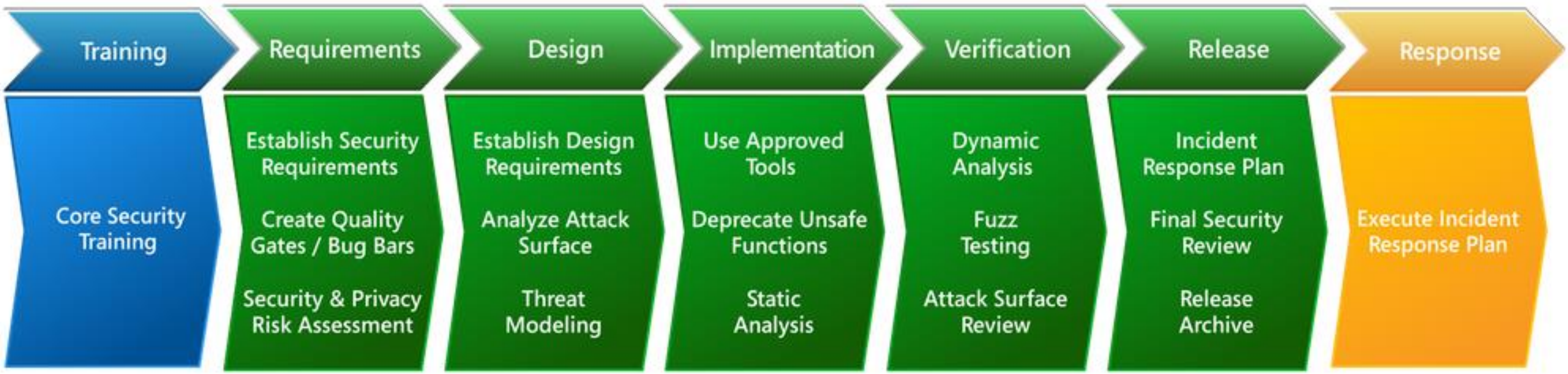
McGraw's Touchpoints

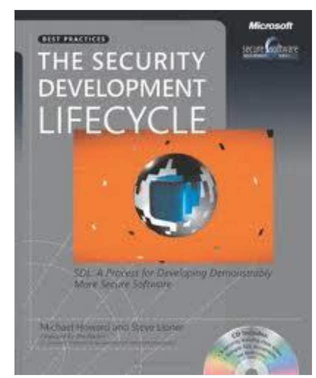


Security activities throughout the software development life cycle (SDLC)

[Gary McGraw, Software security, Security & Privacy Magazine, IEEE, Vol 2, No. 2, pp. 80-83, 2004.]

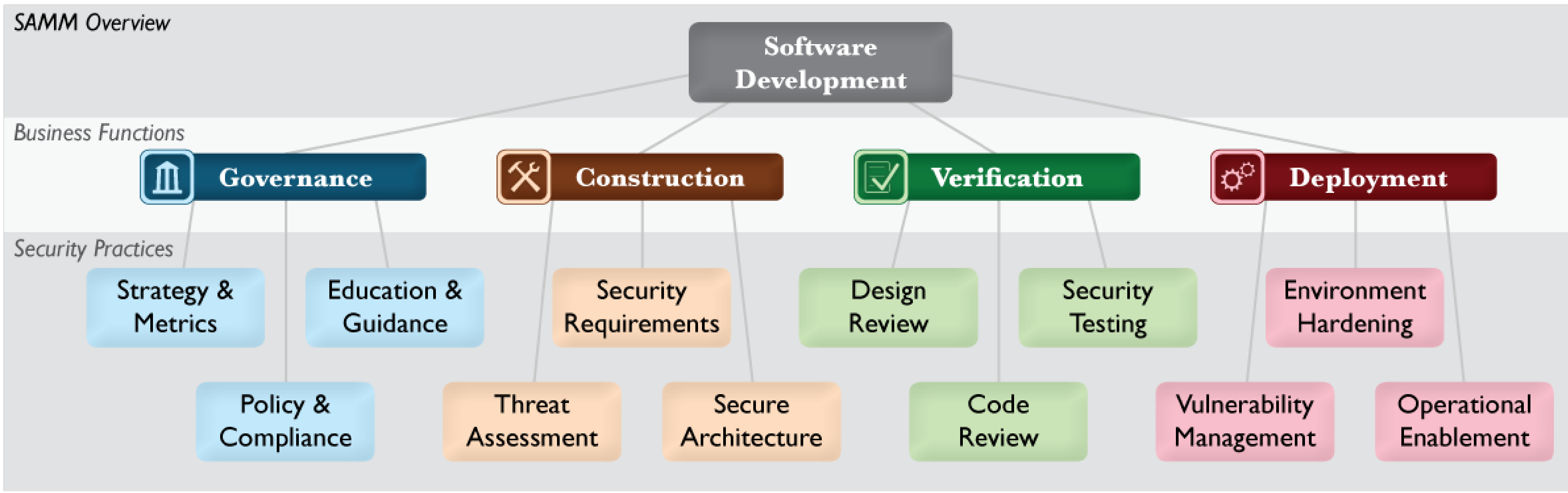
Microsoft SDL





OWASP OpenSAMM

12 security practices in 4 business functions



BSIMM (Building Security In Maturity Model)

12 practices across 4 domains, subdivided into 100 activities

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 Manage- ment

https://www.bsimm.com/framework/

BSIMM to compare security maturity

Software security in slogans

- **Security by Design:** thinking of security *from the start*
  - But: we will never foresee or prevent all security problems
- **Shifting Left:** tackling security *earlier*
  - eg. not (only) relying on pen-testing but (also) having security tests or even static analysis to catch problems
- **Shifting Down:** tackling security *lower* in the tech stack
  - moving from C/C++ to Rust
  - using a web framework for session management instead of making your own
  - using 'safe' APIs instead of injection-prone APIs (more later)
  - LangSec to tackle root causes of insecure input handling (more later)

What has changed in software engineering in the past 20 years?

What's changed? More acronyms

- **SAST:** *static* application security testing
  static analysis to catch security flaws
- **DAST:** *dynamic* application security testing
  testing to catch security flaws
- **IAST:** *interactive* application security testing
  (tool-supported) penetration testing
- **RASP:** runtime application self-protection
  instrumentation to detect weird things at runtime

Many more methodologies, frameworks, and guidelines

Most methodologies for secure software lifecycles are very similar

Arina Kudriavtseva & Olga Gadyatskaya of Leiden University recently compared 28 of them [arXiv:2211.16987, 2022]

More concrete 'guidelines' to supplement such methodologies include

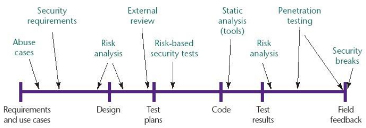
OWASP ASVS (Application Security Verification Requirements)
NIST SP 800-218 SSDF (Secure Software Development Framework)

Hard to see the forest for the trees!

- OWASP OpenCRE by a.o. Rob van der Veer of SIG in Amsterdam is recent initiative to relate entries between methodologies, guidelines and standards [https://www.opencre.org]

What's changed? Agile & DevOps

All approaches for secure SDLC use waterfall model frame of reference



- *How can we cope with Agile development?*
  You cannot use pen-testers for every new feature...
  - Hence: *more important to shift left!*
    Eg using SAST & DAST. And train developers to give them more security expertise?
- *How can we cope with DevOps ?*
  You cannot hire pen-testers or run tests for every new release...
  - Hence: *even more important to shift left!*
    Eg integrate SAST (& DAST?) into CD/CI pipelines
  - Some proposals for DevSecOps as new buzzword

What's changed? Code repositories

Modern software development relies heavily on reusing components from code repositories

- github, Maven, PyPi, RubyGems, ...

- New attack vector: supply chain attacks
  - Eg Log4J , SolarWinds

NCSC slaat alarm om kwetsbaarheid in Apache Log4j

11 december 2021 11:55 | Rik Sanders

BY KIM ZETTER BACKCHANNEL MAY 2, 2023 6:00 AM

The Untold Story of the Boldest Supply-Chain Hack Ever

The attackers were in thousands of corporate and government networks. They might still be there now. Behind the scenes of the SolarWinds investigation.

- New countermeasures

1) SCA (Software Composition Analysis):
   static analysis tools to check software supply chain for CVEs

2) SBOM (Software Bill of Materials)
   Required by executive Order 14028 'Improving the Nation's Cybersecurity' (May 2021)

What's changed? 'Services'

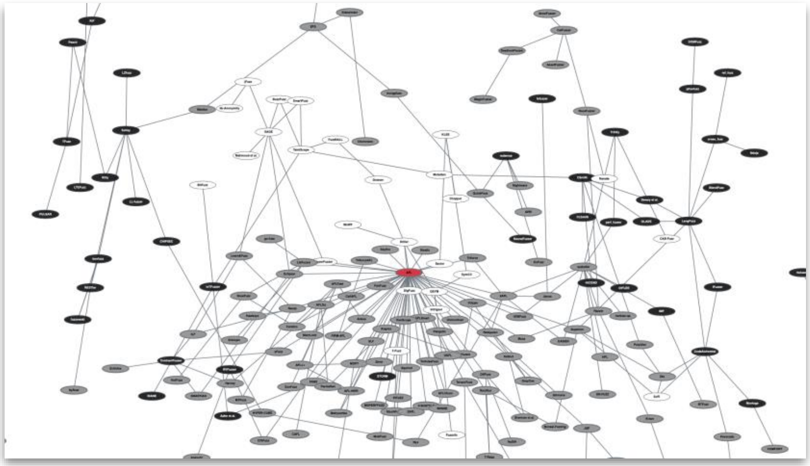
Software increasingly built not only with libraries as components but also using (cloud-based) services

- **eg micro-services, SaaS, cloud APIs, ...**

- **This introduces**
  - **new attack surfaces**
  - **need for authentication to cloud APIs**
- **New security flaw: leaking credentials**
  (JWT tokens, AWS security tokens, ...)
- **New countermeasure: SAST tools for secret scanning**
- **Also: first proposals for SaaSBOMs**

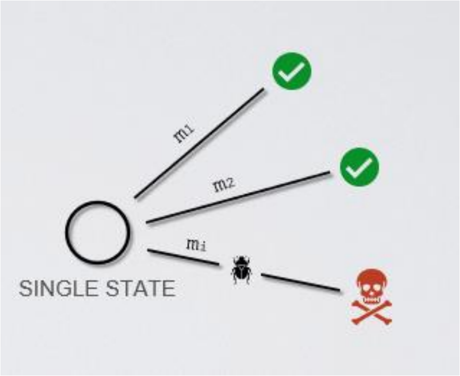
What has changed? Fuzzing

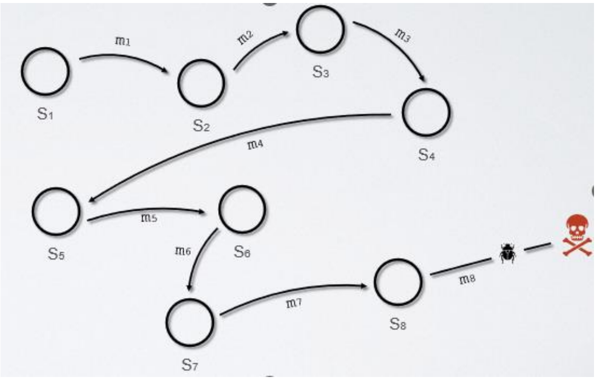
- **Fuzzing as (semi)-automated testing technique has proved very successful at finding security flaws, esp. memory corruption**
- **Esp. with afl as evolutionary coverage-guided fuzzer**
- **Google OSS Fuzz initiative is continuously fuzzing open source projects**



https://fuzzing-survey.org

One of remaining challenges: fuzzing stateful systems





[Fuzzers for Stateful Systems, Cristian Daniele, Seyed Benham Andarzian, Erik Poll arXiv:2301.02490, 2023]

What has *not* changed in software engineering in the past 20 years?

What has not changed?

Organisations are

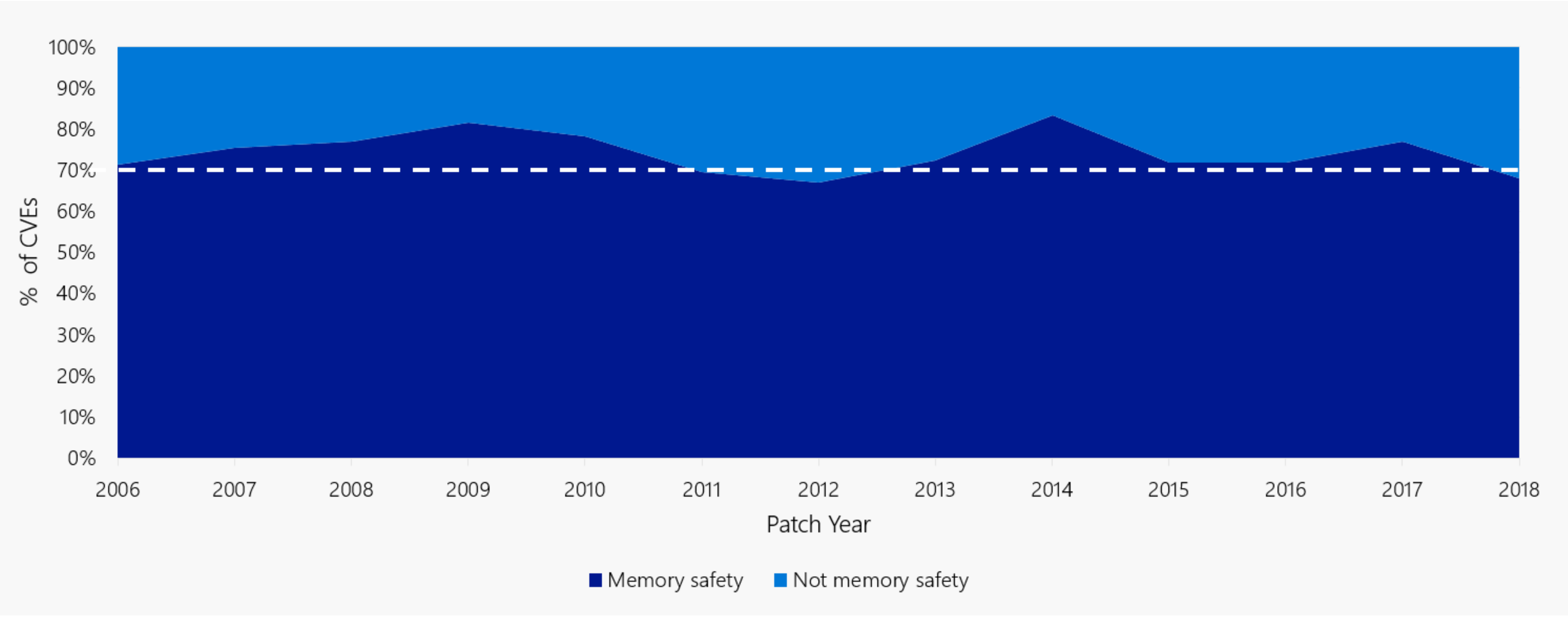
- still trying to shift left
- or even still getting started with security in the SDLC

Ongoing initiative by Dutch government organisations:

Grip op SSD (Secure Software Development)

https://www.cip-overheid.nl/en/category/products/secure-software/

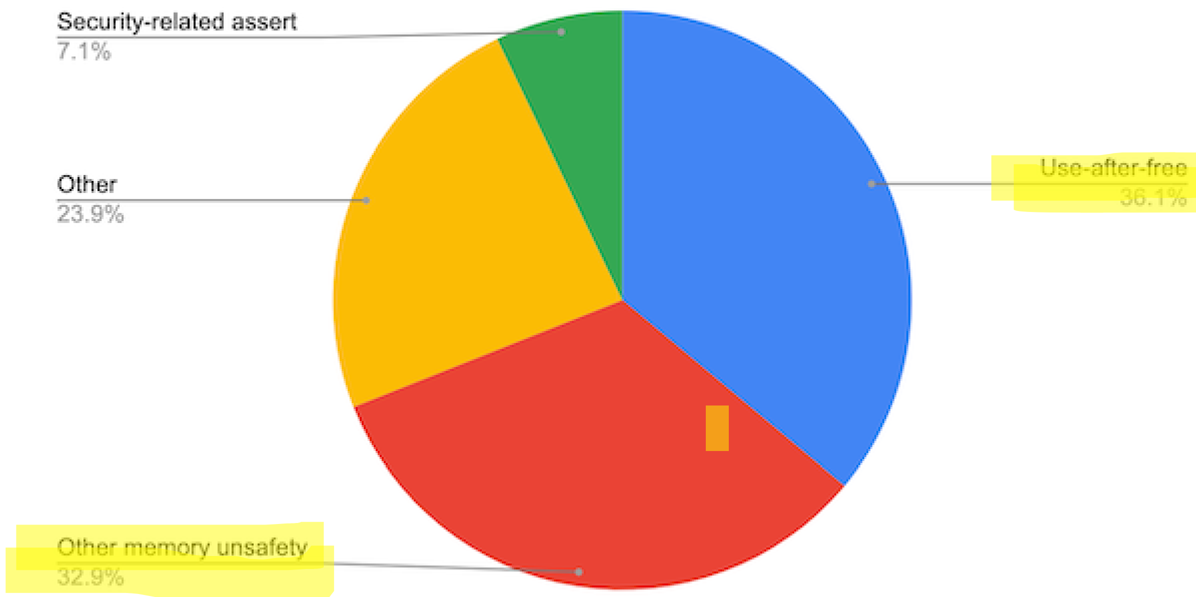
What has not changed? Memory corruption bugs



[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]

Memory corruption bugs in Chromium project – since 2015

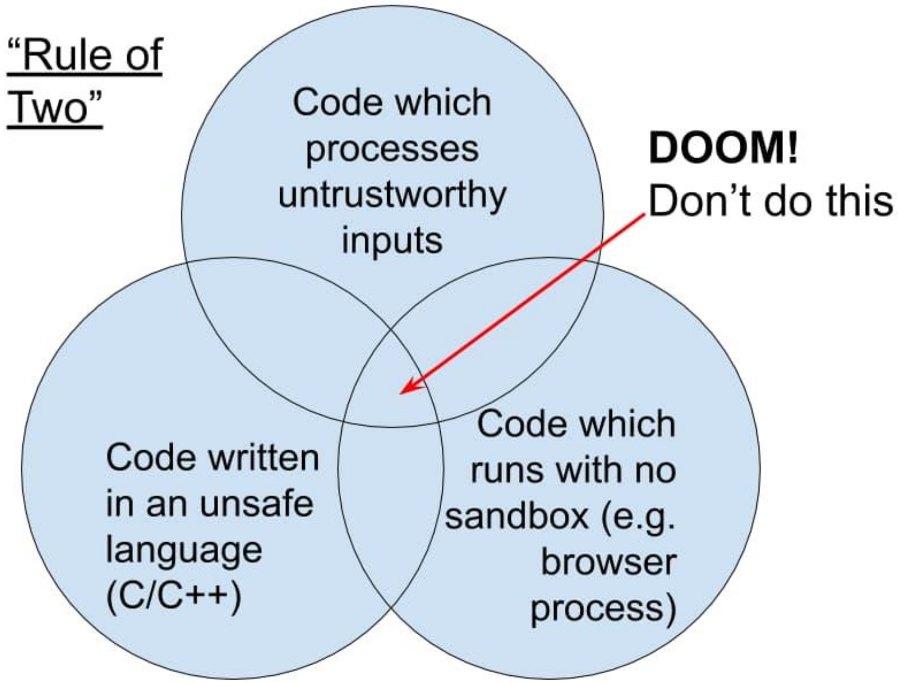
70% of high severity & critical security bugs are memory safety problems



[Source: https://www.chromium.org/Home/chromium-security/memory-safety]

Rule of 2 in Chromium project

"When you write code to parse, evaluate, or otherwise handle untrustworthy inputs from the Internet, don't do more than 2 of …"



[https://chromium.googlesource.com/chromium/src/+/refs/heads/main/docs/security/rule-of-2.md]

What has not changed?

Many bugs arise in INPUT handling

Eg flood of bugs in handling WebP image format past weeks:

Apple squashes security bugs after iPhone flaws exploited by Predator spyware

Holes in iOS, macOS and more fixed following tip off from Google, Citizen Lab

Chris Williams Fri 22 Sep 2023 // 19:58 UTC

Critical vuln in libwebp: Go get updates to Chrome, Firefox, Edge, Slack and more.

Critical New 1Password, Signal, Chrome, Edge, Firefox Emergency Security Updates

Davey Winder Senior Contributor

Co-founder, Straight Talking Cyber

Follow

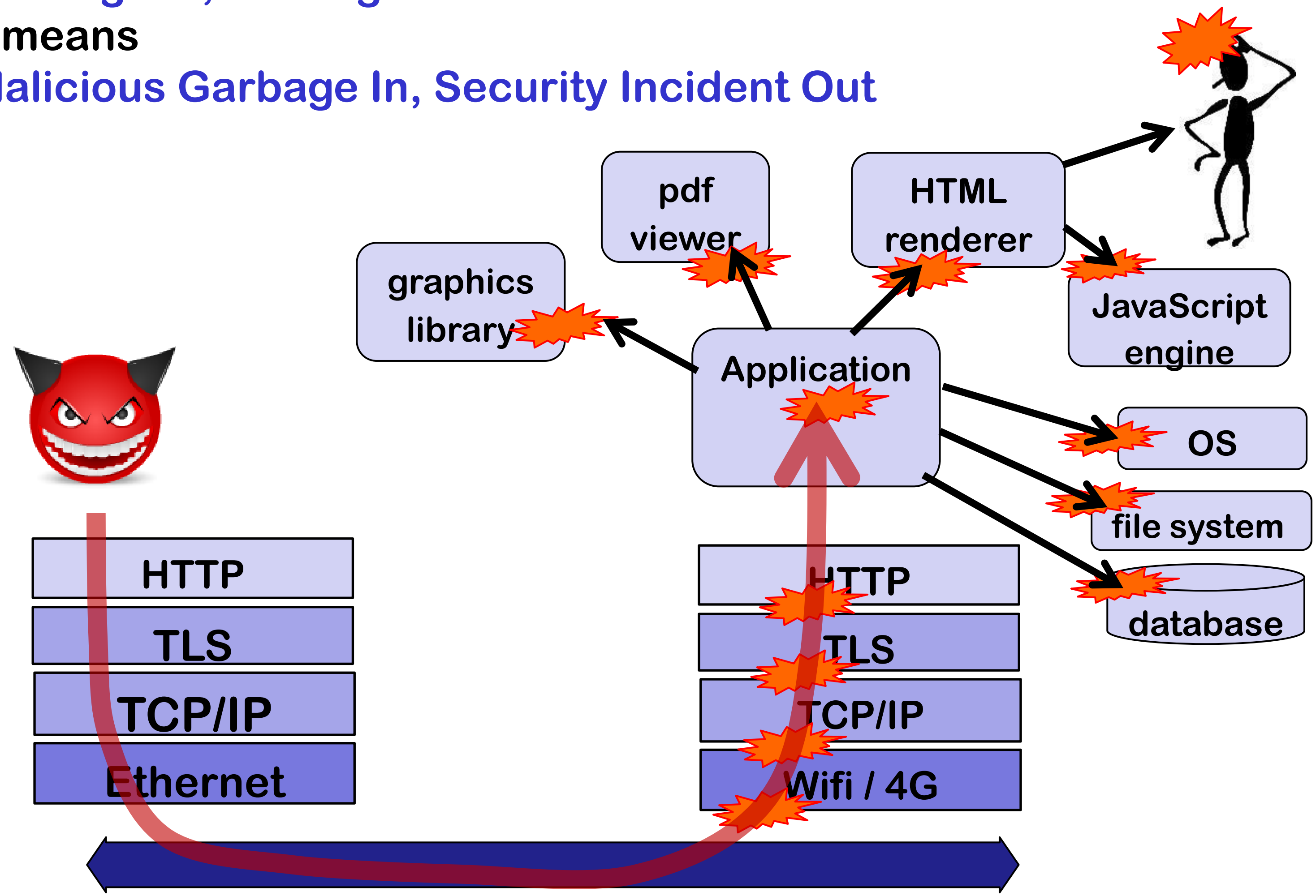
0 Sep 14, 2023, 05:27am EDT

INPUT handling problems

Garbage In, Garbage out

means

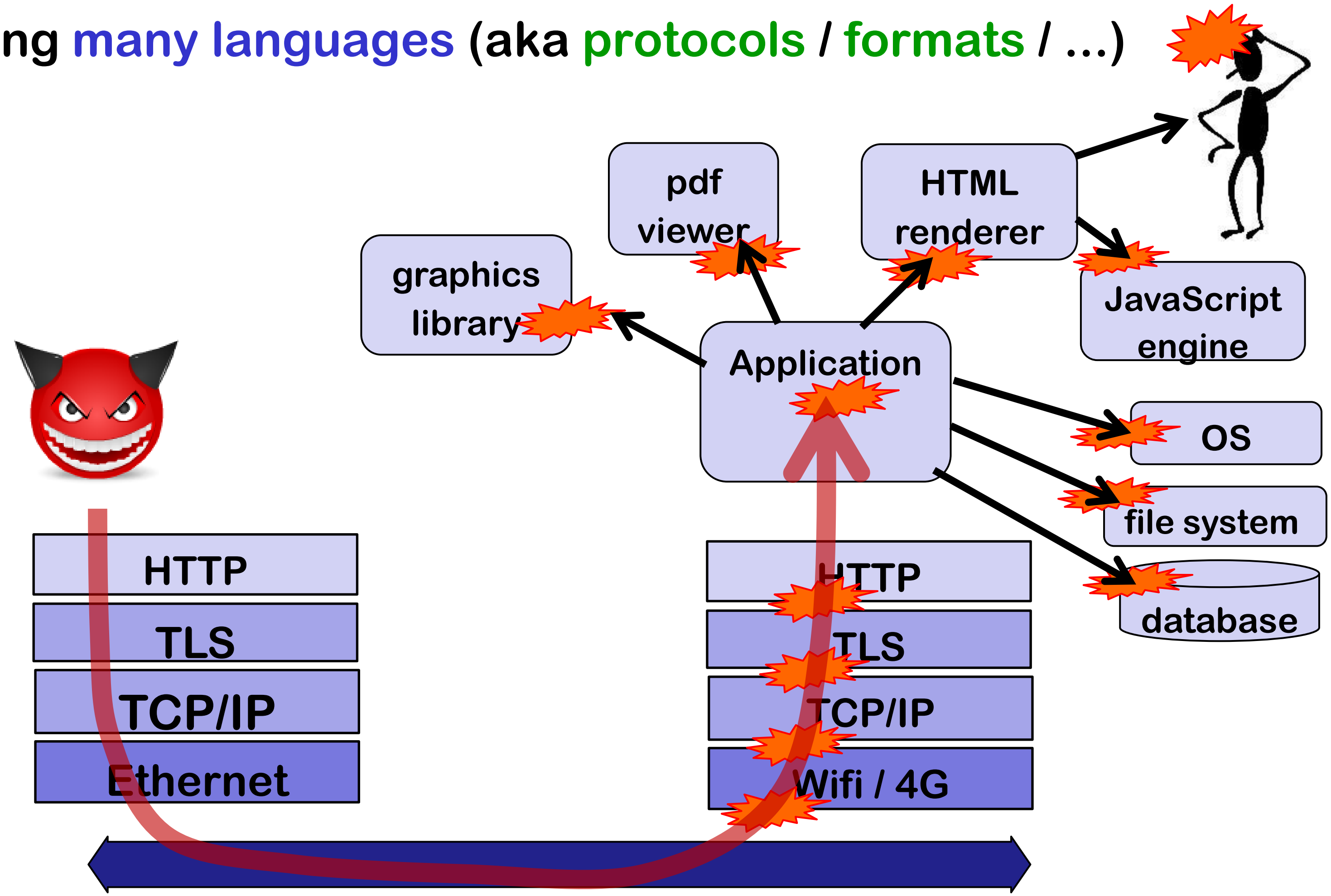
Malicious Garbage In, Security Incident Out



INPUT problems involve *parsing* & *languages*

Input is parsed (aka decoded / interpreted/...) in many places.

Involving many languages (aka protocols / formats / ...)



Typical bug categories

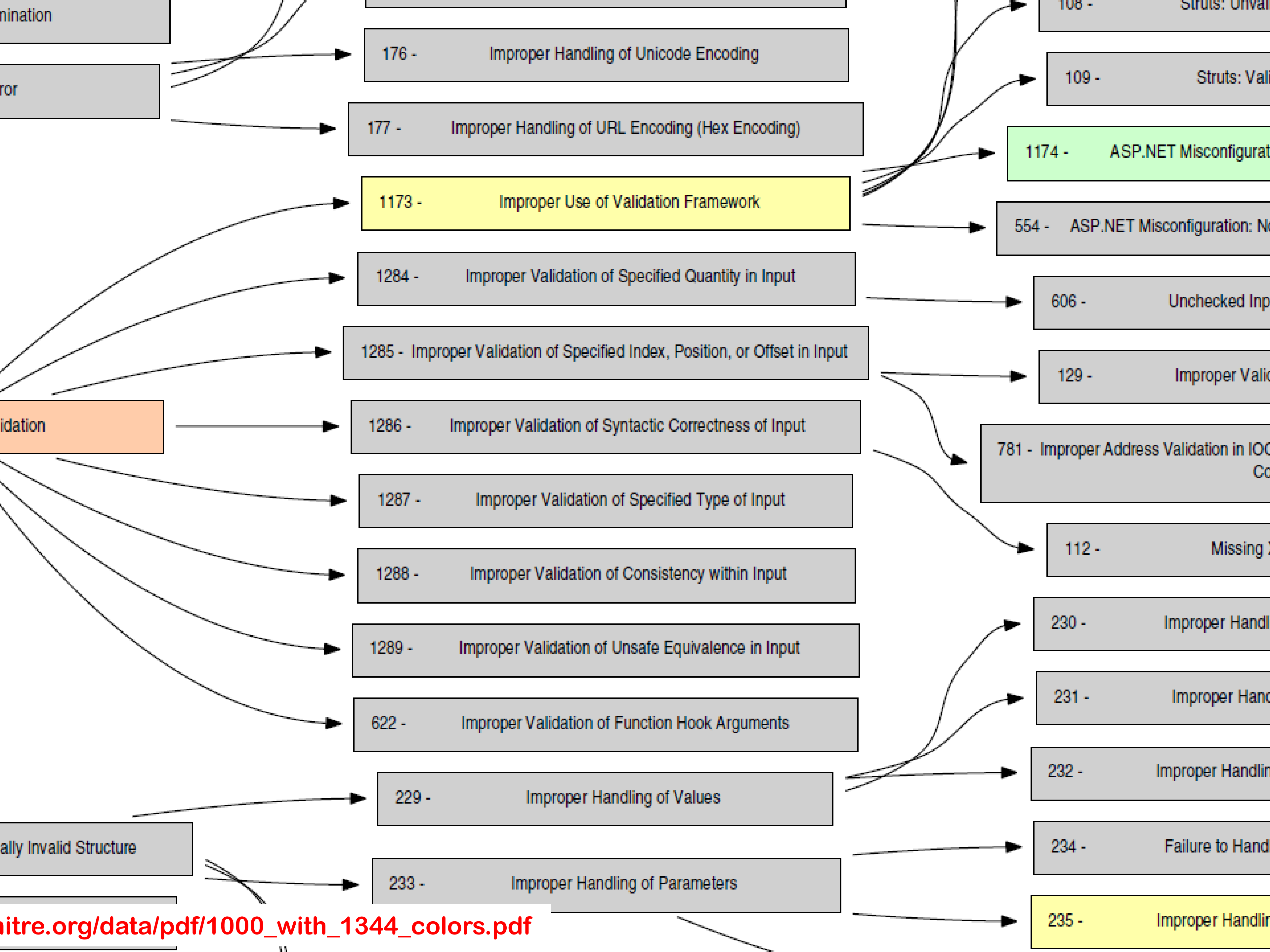
OWASP Top 10 [2017]

1. Injection
2. Broken Authentication
3. Sensitive Data Exposure
4. XML External Entities (XXE)
5. Broken Access Control
6. Security Misconfiguration
7. Cross-Site Scripting (XSS)
8. Insecure Deserialization
9. Using Components with Known Vulnerabilities
10. Insufficient Logging & Monitoring

CWE TOP 25 [2022]

1 Out-of-bounds Write
2 Cross-site Scripting
3 SQL Injection
4 Improper Input Validation
5 Out-of-bounds Read
6 OS Command Injection
7 Use After Free
8 Path Traversal
9 Cross-Site Request Forgery (CSRF)
10 Unrestricted Upload of File with Dangerous Type
11 NULL Pointer Dereference
12 Deserialization of Untrusted Data
13 Integer Overflow or Wraparound
14 Improper Authentication
15 Use of Hard-coded Credentials
16 Missing Authorization
17 Command Injection
18 Missing Authentication for Critical Function
19 Improper Restriction of Bounds of Memory Buffer
20 Incorrect Default Permissions
21 Server-Side Request Forgery (SSRF)
22 Race Condition
23 Uncontrolled Resource Consumption
24 Improper Restriction of XML External Entity Reference
25 Code Injection

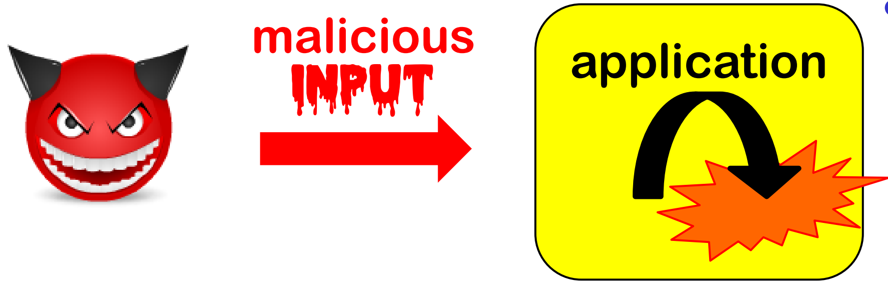
MITRE CWE TOP 1000


mination
ror
176 - Improper Handling of Unicode Encoding
177 - Improper Handling of URL Encoding (Hex Encoding)
1173 - Improper Use of Validation Framework
1284 - Improper Validation of Specified Quantity in Input
1285 - Improper Validation of Specified Index, Position, or Offset in Input
idation
1286 - Improper Validation of Syntactic Correctness of Input
1287 - Improper Validation of Specified Type of Input
1288 - Improper Validation of Consistency within Input
1289 - Improper Validation of Unsafe Equivalence in Input
622 - Improper Validation of Function Hook Arguments
229 - Improper Handling of Values
ally Invalid Structure
233 - Improper Handling of Parameters
108 - Struts: Unva
109 - Struts: Vali
1174 - ASP.NET Misconfigurat
554 - ASP.NET Misconfiguration: N
606 - Unchecked Inp
129 - Improper Valid
781 - Improper Address Validation in IOC
Co
112 - Missing
230 - Improper Handl
231 - Improper Hand
232 - Improper Handlin
234 - Failure to Hand
235 - Improper Handlin

The <u>two</u> problems in input handling

1. *Insecure, buggy* parsing

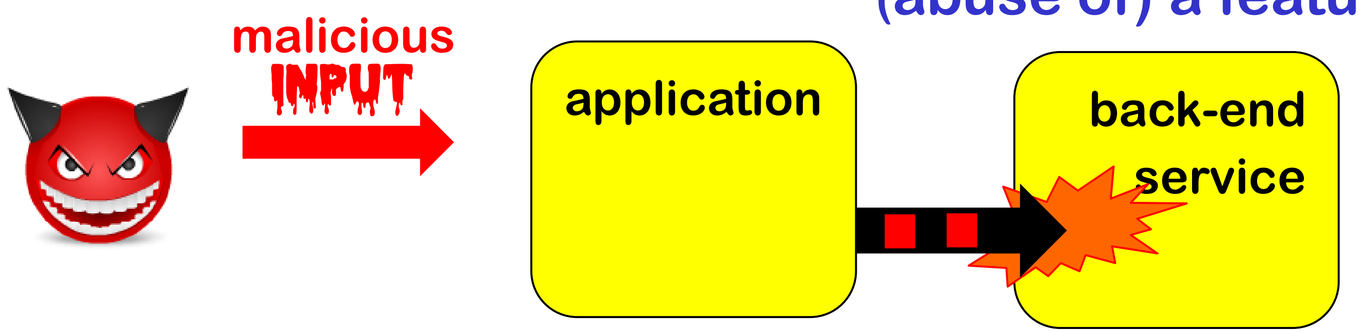
a bug !



eg. buffer overflow
in WebP library

2. *Unintended* parsing

(abuse of) a feature !



eg. SQL injection, XSS, ...

Tackling *buggy* parsing:

using the LangSec approach

Root causes of buggy parsing

1. Many input languages / formats / protocols

   Wifi, Ethernet, Bluetooth, GSM/3G, 4G, 5G, ...

   TCP/IP, UDP, HTTP(S), TLS, SSH, OpenVPN, ...

   URLs, X509 certificates, domain names, ...

   JPG, MP3, MPEG, WebP, ...

   HTML, PDF, Word, Excel, Powerpoint, ...

   Often these are complex and/or poorly specified

2. Hand-written parser code, often in unsafe languages like C(++)

Fuzzing – aka fuzz testing – is a great way to find these bugs!

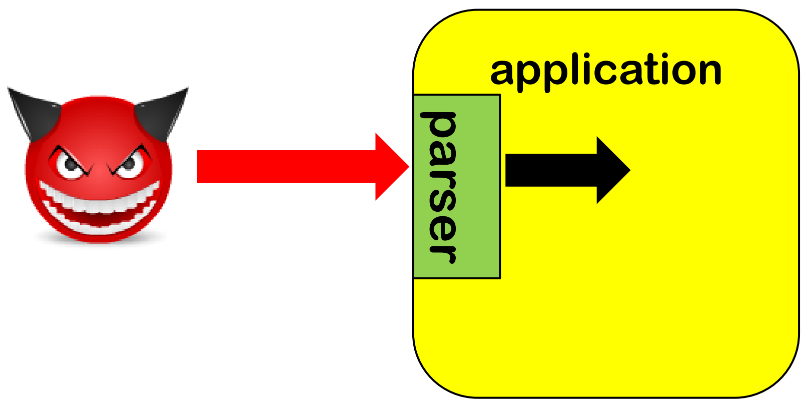
LangSec: tackling buggy parsing

1. Provide clear, formal spec of input language

   eg as regular expression or BNF grammar

2. Generate parser code

   using a parser generator tool

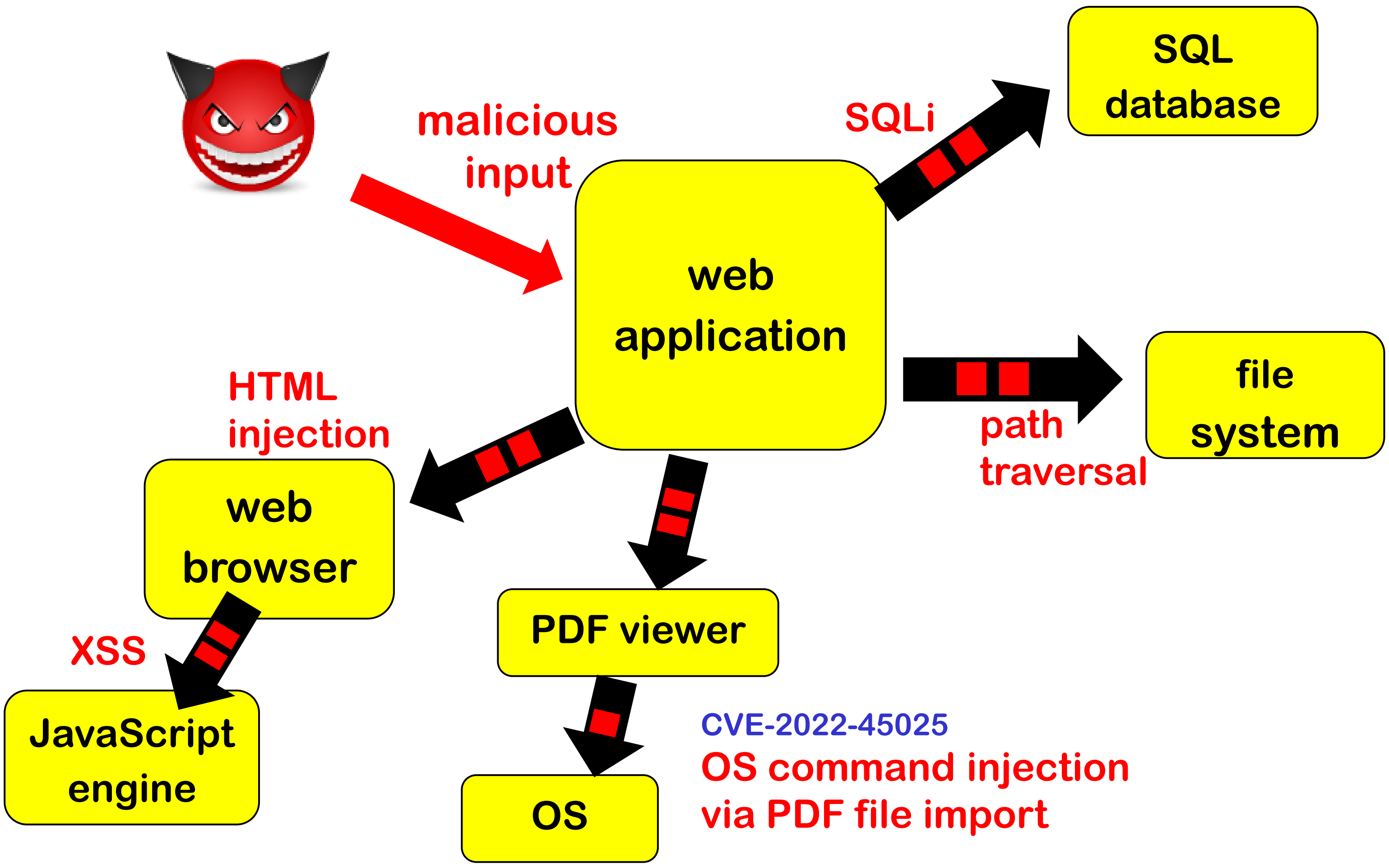


More info at langsec.org

Tackling *unintended* parsing
(ie injection attacks)

use types!

Many back-ends, with input languages, more problems with unintended parsing …



Root causes of unintended parsing

1. **Many** languages: e.g **HTML, SQL, PDF, OS commands**
   - Also **output** languages, not just **input** languages
   - Possibly combined or nested in complex way
2. **Complex data flows** where user input can end up being interpreted as one of these languages
3. **Powerful, expressive** languages

   JavaScript in HTML,

   JavaScript or ActionScript in PDF,

   SQL commands,

   OS commands, ...

Anti-pattern: STRINGS

Strings are *useful*, because you use them to represent many things

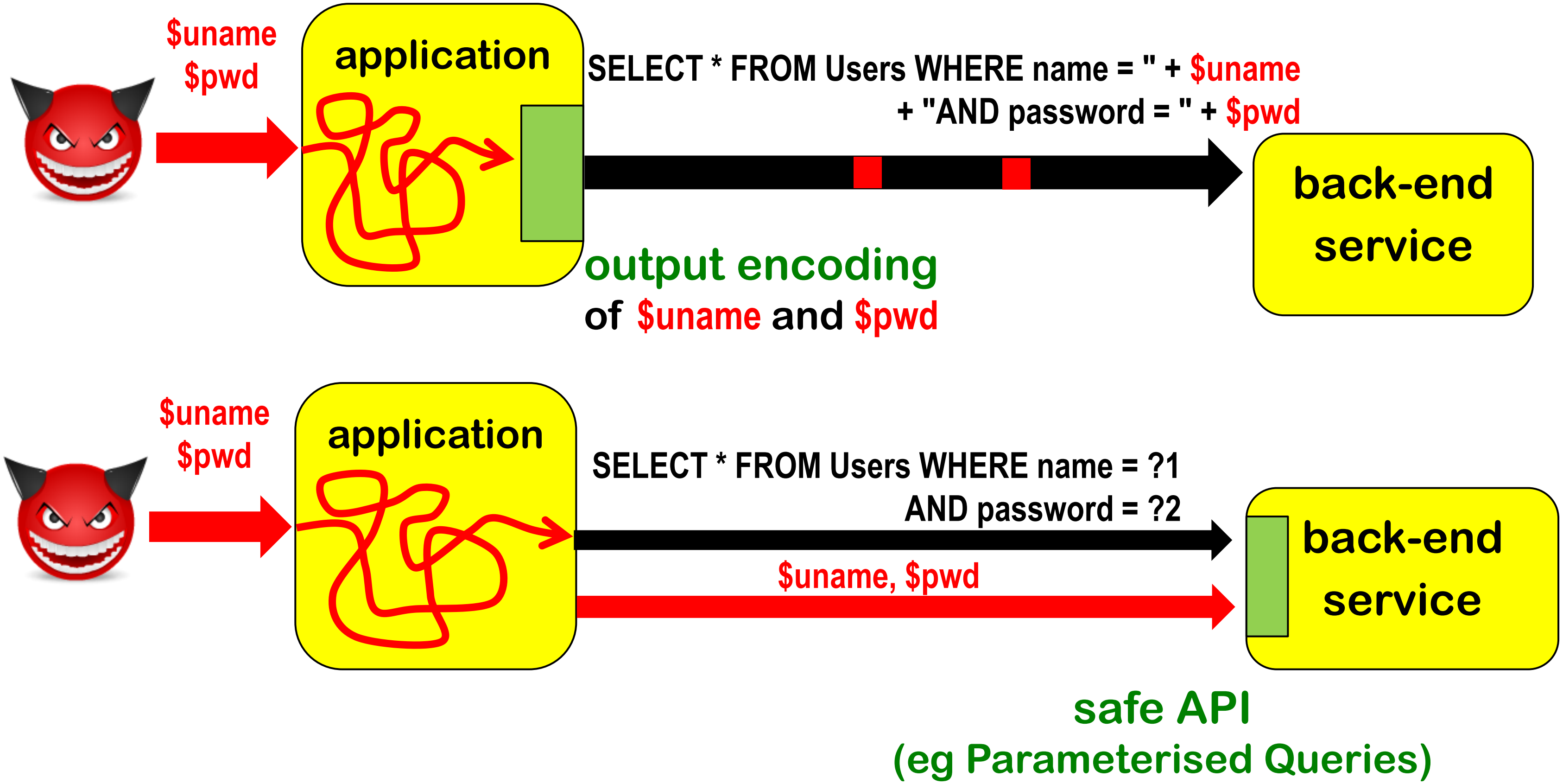
eg. user name, file name, email address, URL, shell command, snippet of SQL, HTML, or JavaScript, ...

- Not just `String` but also `char*`, `char[]`, `StringBuilder`, `...`

This also make strings *dangerous:*

1. A string is unstructured & unparsed data, and processing it often involves some interpretation – incl. parsing
2. The same string may be handled & interpreted in many – possibly unexpected – ways
3. A string parameter in an API call can – and often does – hide a very expressive & powerful language

Solutions: output encoding or safe APIs



This is about avoiding parsing

Safe Builder Approach

- Classic approach to finding injection flaws in SAST tools:

  tainting

- More structural approach (in coding phase):

  'safe builder approach'

  i.e. introduce a dedicated type for a specific format /language with a restricted set of operations to construct values of that type

[Christoph Kern. Preventing Security Bugs through Software Design.
Presentation at OWAPS AppSec California 2016. 2016. https://www.youtube.com/watch?v=ccfEu-Jj0as]

Example: Safe builder for SQL injection

- Suppose we have an unsafe API method

```
void executeDynamicSQLQuery (String s)
```

- We define a new 'wrapper' String type `SQLquery` and a function that executes such a wrapped string

```
void safeExecuteSQLQuery (SafeSQLquery s){
        executeDynamicSQLCommand(the string in s );
 }
```

- We now define functions to create `SafeSQLqueries`

  1. any compiled-time constant can be turned into a `SQLquery`

     ```
     SafeSQLquery create (@CompiletimeConstant String s)
     ```

  2. we can append a string to an `SafeSQLquery` using a function

     ```
     SafeSQLquery appendSQL (SafeSQLquery q, String s)
     ```

     which applies the right encoding to `s`

Type system guarantees that user inputs in queries are properly encoded.
We can gradually disallow use of the old unsafe `executeDynamicSQLQuery`.

Safe builders for several contexts

If we use string-like data in several contexts, each with their own encoding, we can introduce a different String-like type for each, e.g.

`SafeSQLquery, SafeHTML, SafeOSCommand, SafeFilename`

with association constructors or factory methods for each, e.g.

```
SafeHTML create (@CompiletimeConstant String s)

SafeHTML concatHTML (SafeHTML h1, SafeHTML h2)

SafeHTML appendHTML (SafeHTML h, String s)
```

`appendHTML(h,s)` and `appendSQL(h,s)` would use different encodings (aka sanitisations) for the parameter `s`

We could introduce unsafe loopholes that we evaluate by hand

```
SafeHTML unsafeCreate (String s)
```

Example: Trusted Types DOM API in Chrome browser

Trusted Types initiative to root out DOM-based XSS

replaces string-based DOM API with typed API

- **Type checking ensures that untrusted data can only reach dangerous APIs after passing (carefully vetted) validation or encoding operations**

```
TrustedHTML htmlEncode(String str)
TrustedHTML create(@Compiletimeconstant String str)
```

[Wang et al., If It's Not Secure, It Should Not Compile: Preventing DOM-Based XSS in Large-Scale Web Development with API Hardening, ICSE'21, ACM/IEEE, 2021]

[https://github.com/WICG/trusted-types]

Summary

- We know how to make software more secure:
  just pick one of the many secure development methodologies
- Agile & DevOps only highlight the importance of shifting left
- The use of repos increases risk of supply chain attacks:
  hence SCA and SBOMs
- Using more 'services' means more authentication to APIs and more credentials that can leaks.
  hence secret scanning tools as part of SAST. And SaaSBOMs?
- Structural way to improve security by shifting down:
  eg recognise the role of input languages and parsing of them
  - use LangSec approach to prevent them
  - use types to track different kinds of data

Thanks for your attention!

[Strings considered harmful USENIX :login; 2019]