Measuring Developers' Web Security Awareness from Attack and Defense Perspectives

IEEE S&P SecWeb Workshop 2022

Merve Şahin, Tolga Ünlü, Cédric Hebert,
Lynsay Shepherd, Natalie Coull and Colin McLean

SAP Security Research

Abertay University

Motivation

“Security of web applications stands and falls with their developers.”

Measuring developers’ awareness of **web attacks** and **available defenses** can help to:

→ Understand the root causes of security issues (e.g. Simple access control vulns such as IDOR).

→ Identify the knowledge gaps in security concepts and see how they can be addressed. [1]

→ Understand how the available security mechanisms and framework/browser features can be better utilized. [2]

```
POST /administration/user
HTTP/1.1
…
{ “Name”: “…”,
  “companyId”: “…”
}
```

[1] Roth et al., “12 Angry Developers-A Qualitative Study on Developers’ Struggles with CSP”, CCS’21.

[2] Likaj et al., “Where We Stand (or Fall): An Analysis of CSRF Defenses in Web Frameworks”, RAID’21.

Method

Questionnaire-based Online Survey

Defenders' Perspective

Measuring developers' awareness of common security controls, esp. Input Validation (IV), and their ability to detect indicators of attacks in a scenario.

Capture-the-Flag (CTF) Challenge

Attackers' Perspective

Measuring developers' awareness of attack vectors and to what extent they attempt different vectors to win the CTF challenge.

Participant Recruitment

Voluntary/Self-motivated participation with no monetary reward.

Online Survey

Participants:
21

Source(s):
Social Media (Twitter, Linkedin, Reddit), DEV Community

→ 7 Countries (8 UK, 5 DE)

→ Diverse Professions

CTF Challenge

Participants:
82

Source(s):
Enterprise CTF Platform

→ Security Enthusiasts

Participant Recruitment - Limitations

Different Participant Sets

Both experiments have a separate set of participants, requiring individual analysis of the results.

Possible Biases

Security enthusiasts may bring bias towards a higher attack-awareness ratio.

Further Considerations

Participants have different years/levels of experience.

Development is Teamwork: Awareness of an individual developer does not necessarily correlate to the security level of the application they develop.

Experiment I: Online Survey

Security Controls and Input Validation (IV)
General familiarity (understanding and impl. experience) of common security controls with focus on IV.

Detecting Attack Attempts - Request Tampering
Understanding of what makes request tampering possible and evaluation through a scenario-based question.

Participant Demographics
Participants’ job title, years of experience, frameworks they work with and other information.

Observations from Survey: Security Controls and Input Validation (IV)

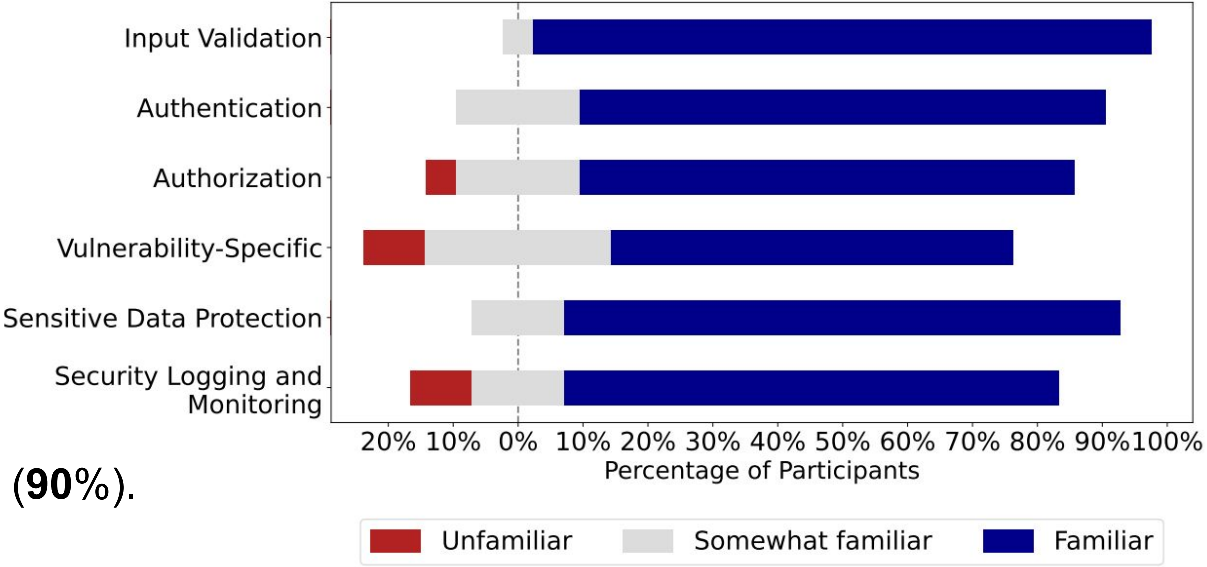
Overall, high familiarity (self-reported) with the available controls.

Some unfamiliarity with:
→ Logging and Monitoring,
→ Vulnerability-Specific,
→ Authorization

Regularly involved in tasks with IV (**66**%).

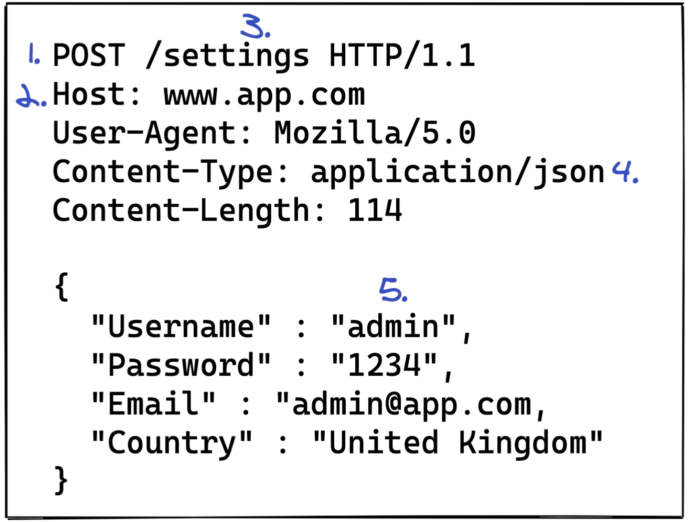
IV Focus: **Content** & **Structure** (**90**%).

Client-Side IV:
Considered **Optional** > **Essential** (**57**% > **43**%).



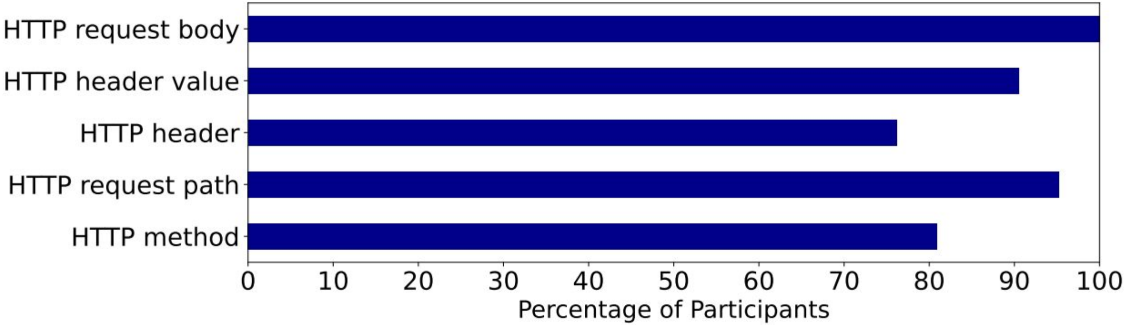
Observations from Survey: Request Modifiability

Participants report which parts of a HTTP request (**1.-5.**) can be modified by the client:



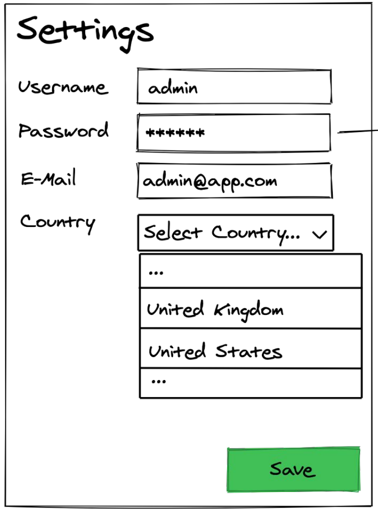
```
POST /settings HTTP/1.1
Host: www.app.com
User-Agent: Mozilla/5.0
Content-Type: application/json
Content-Length: 114

{
  "Username" : "admin",
  "Password" : "1234",
  "Email" : "admin@app.com,
  "Country" : "United Kingdom"
}
```



7 (**33%**) participants are not aware that all parts of an HTTP request can be modified.
→ Limited IV and awareness of client-side control.

Observations from Survey:
Detecting Request Tampering Scenario

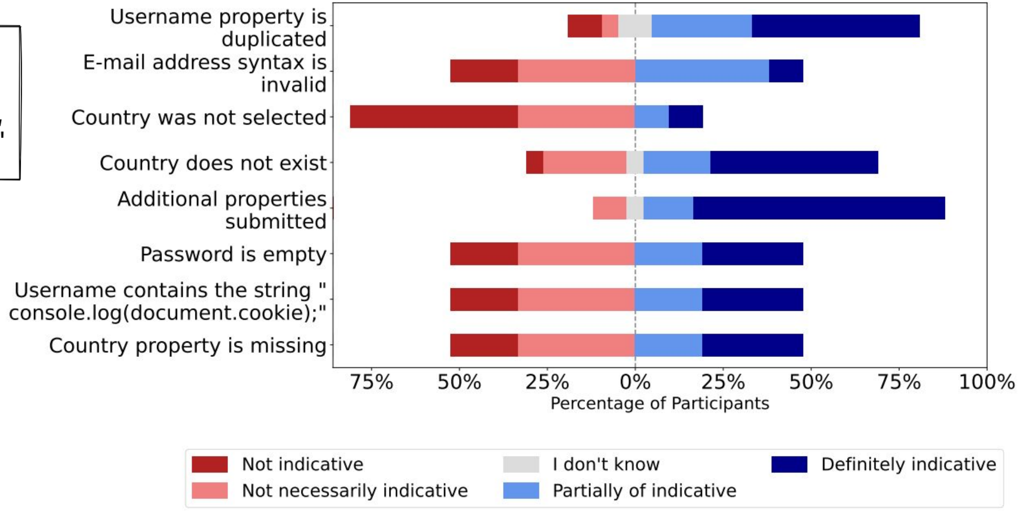


HTTP Request

```
{
   "Username"  : "admin",
   "Password"  : "1234",
   "Email"     : "admin@app.com",
   "Country"   : "United Kingdom"
}
```

Client-Side Input Validation:

- E-Mail Address Syntax
- Username Min Length 6, Max Length 255
- Password Min Length 8
- All inputs are required/cannot be empty



Given a scenario (Example HTML form and HTTP request, set of **client-side IV rules**), → Participant asked whether certain events observed on server-side indicate an attack.

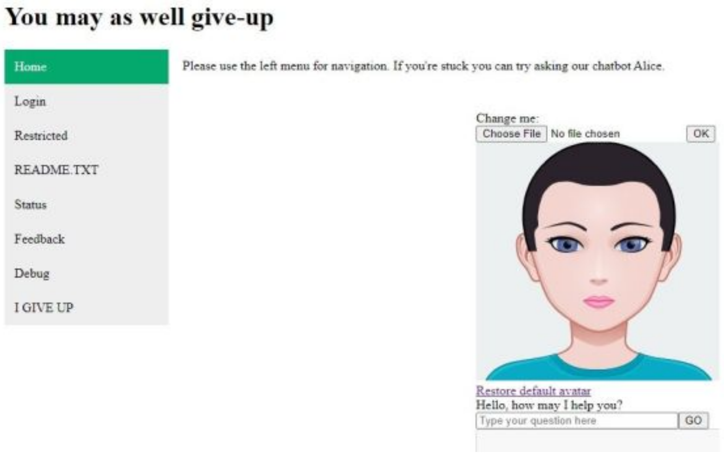
Only 3 (**14%**) participants reported all events as definitely/partially indicative of attack.

Experiment II: The “Give Up” CTF Challenge

Push participants to try as many attacks as possible - Flag unlocked when **all attack vectors** are **attempted** (no exploitation).

CTF Application
→ 7 Endpoints
→ 17 Attack vectors [3]
→ Attack attempts silently tracked
→ Hints within application

I GIVE UP: Shows detected attacks and how many are left to unlock the flag.



You may as well give-up

Please use the left menu for navigation. If you're stuck you can try asking our chatbot Alice.

Change me:

Hello, how may I help you?

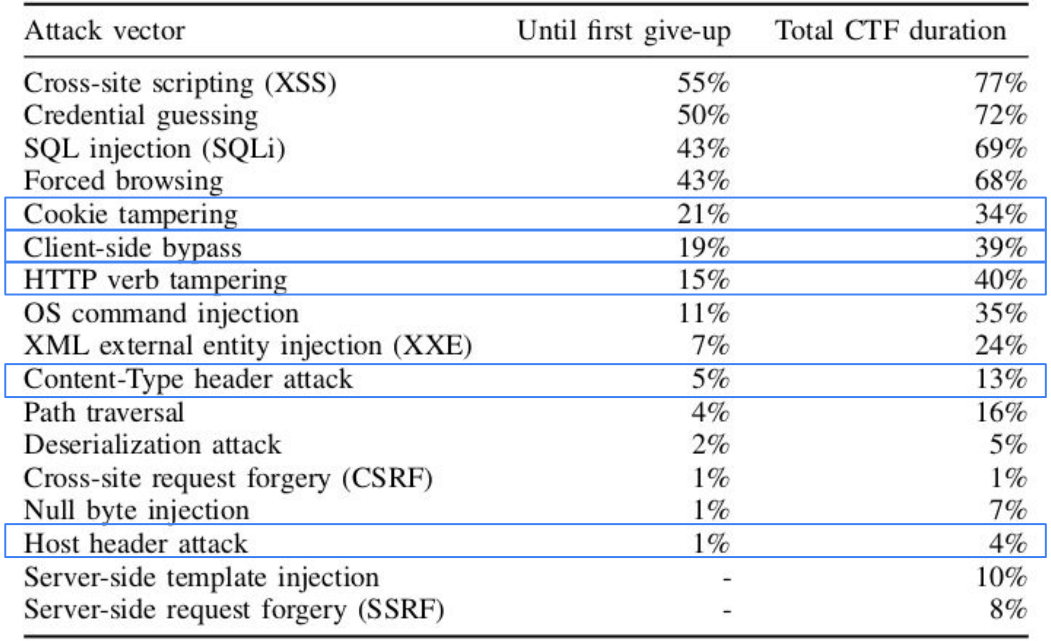
“This application has so many vulnerabilities. Exploit them all, and you’ll be rewarded. But you may as well give up…”.

[3] PortSwigger Web Security Academy - https://portswigger.net/web-security/all-labs

Observations from the CTF

Significantly lower ratio on attacks that require intercepting the request, e.g., Cookie and verb tampering, Client-side bypass, Content-Type and Host header attacks.

→ Survey: Lower awareness on tampering possibility of HTTP method & headers.



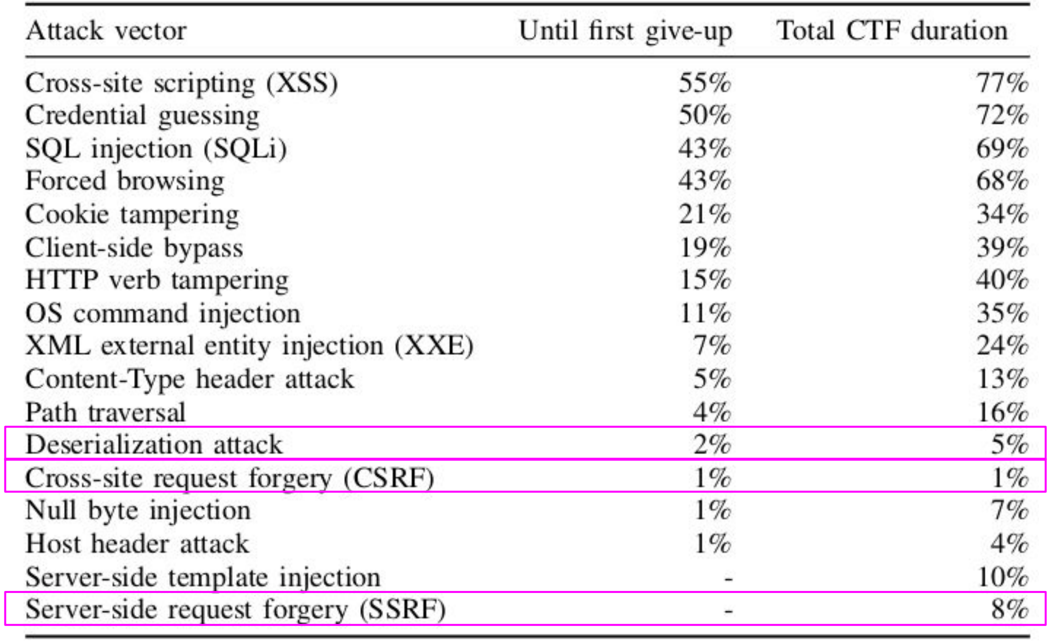
Attack vector	Until first give-up	Total CTF duration
Cross-site scripting (XSS)	55%	77%
Credential guessing	50%	72%
SQL injection (SQLi)	43%	69%
Forced browsing	43%	68%
Cookie tampering	21%	34%
Client-side bypass	19%	39%
HTTP verb tampering	15%	40%
OS command injection	11%	35%
XML external entity injection (XXE)	7%	24%
Content-Type header attack	5%	13%
Path traversal	4%	16%
Deserialization attack	2%	5%
Cross-site request forgery (CSRF)	1%	1%
Null byte injection	1%	7%
Host header attack	1%	4%
Server-side template injection	-	10%
Server-side request forgery (SSRF)	-	8%

Percentage of participants who tried each attack vector.

Observations from the CTF

Deserialization, CSRF, SSRF attacks are attempted by very few → Rather complex attacks, also more difficult to build defenses.

Overall: Limited awareness on attacks → 79% of participants try only ~3 attacks before their first give-up.



Attack vector	Until first give-up	Total CTF duration
Cross-site scripting (XSS)	55%	77%
Credential guessing	50%	72%
SQL injection (SQLi)	43%	69%
Forced browsing	43%	68%
Cookie tampering	21%	34%
Client-side bypass	19%	39%
HTTP verb tampering	15%	40%
OS command injection	11%	35%
XML external entity injection (XXE)	7%	24%
Content-Type header attack	5%	13%
Path traversal	4%	16%
Deserialization attack	2%	5%
Cross-site request forgery (CSRF)	1%	1%
Null byte injection	1%	7%
Host header attack	1%	4%
Server-side template injection	-	10%
Server-side request forgery (SSRF)	-	8%

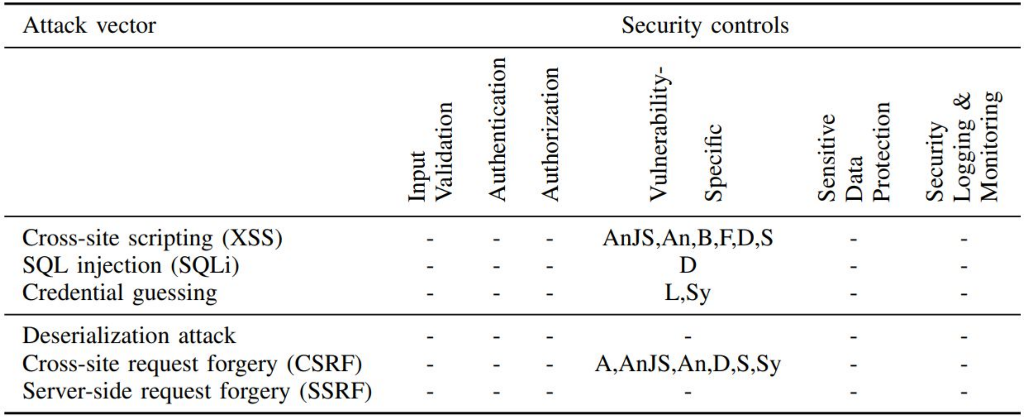
Percentage of participants who tried each attack vector.

Security Documentation of Web Frameworks

Review of framework docs and available **referencing of built-in security controls**.

Focusing on **dedicated security chapters** in documentations.

Framework Selection
Survey participants selection
(In line with Stack Overflow Dev Survey 21').



Attack vector	Security controls					
	Input Validation	Authentication	Authorization	Vulnerability-Specific	Sensitive Data Protection	Security Logging & Monitoring
Cross-site scripting (XSS)	-	-	-	AnJS,An,B,F,D,S	-	-
SQL injection (SQLi)	-	-	-	D	-	-
Credential guessing	-	-	-	L,Sy	-	-
Deserialization attack	-	-	-	-	-	-
Cross-site request forgery (CSRF)	-	-	-	A,AnJS,An,D,S,Sy	-	-
Server-side request forgery (SSRF)	-	-	-	-	-	-

A: ASP.NET, AnJS: AngularJS, AN: Angular, B: Blazor, D: Django, E: Express, F: Flask, L: Laravel, S: Spring, Sy: Symfony

→ Revolve around vulnerability-specific controls
→ Not referenced: Deserialization and SSRF
→ Core enabler of web attacks not discussed:
Arbitrary submission of data.

Conclusions and Outlook

Lack of awareness that the client can submit arbitrary input.

→ Defenders' Perspective:
Request tampering **not fully understood**
→ Attackers' Perspective:
Request tampering **less attempted**

Awareness on certain attacks (SSRF, CSRF) is very limited.

How can we make web attacks and defenses **more salient** to developers?

Future Directions:
Leveling up developers and their common resources (e.g., frameworks and docs) to build with security in mind:

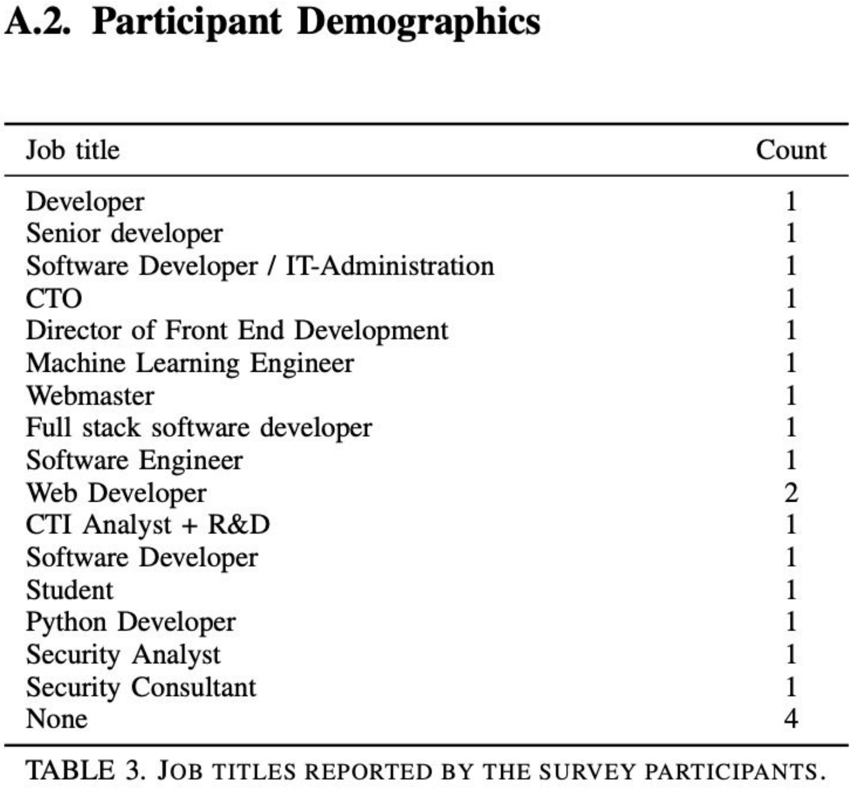
→ Incorporate both attack and defense perspectives within the resources.

→ Security controls that are in line with the developer's workflow, e.g., through Secure by default or Autoconfiguration.

Thank you!

Backup Slides

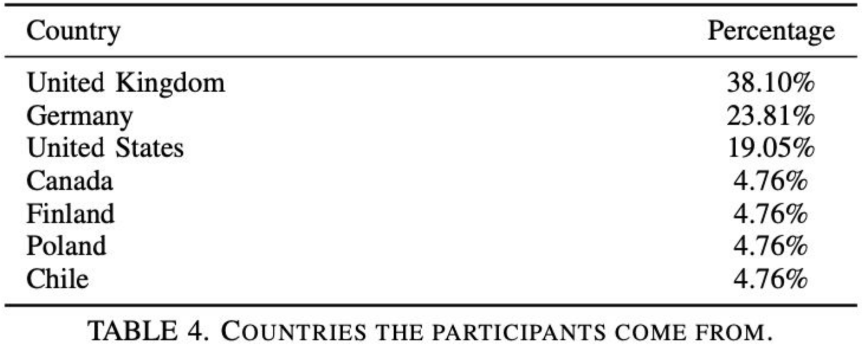
Survey Further Details



A.2. Participant Demographics

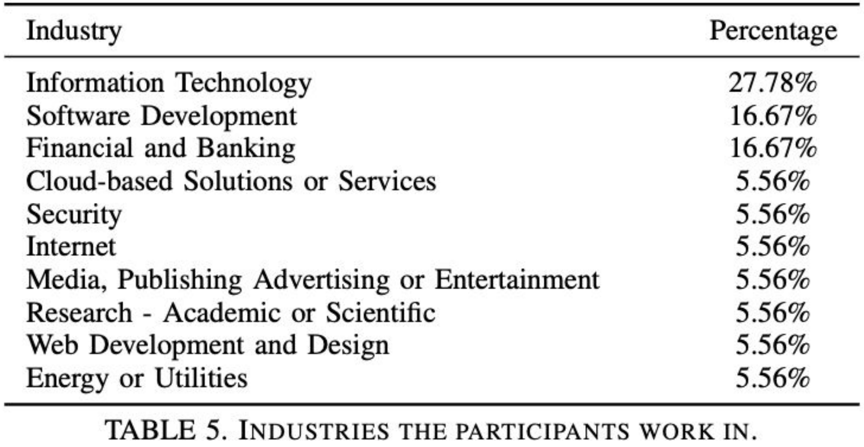
Job title	Count
Developer	1
Senior developer	1
Software Developer / IT-Administration	1
CTO	1
Director of Front End Development	1
Machine Learning Engineer	1
Webmaster	1
Full stack software developer	1
Software Engineer	1
Web Developer	2
CTI Analyst + R&D	1
Software Developer	1
Student	1
Python Developer	1
Security Analyst	1
Security Consultant	1
None	4

TABLE 3. JOB TITLES REPORTED BY THE SURVEY PARTICIPANTS.



Country	Percentage
United Kingdom	38.10%
Germany	23.81%
United States	19.05%
Canada	4.76%
Finland	4.76%
Poland	4.76%
Chile	4.76%

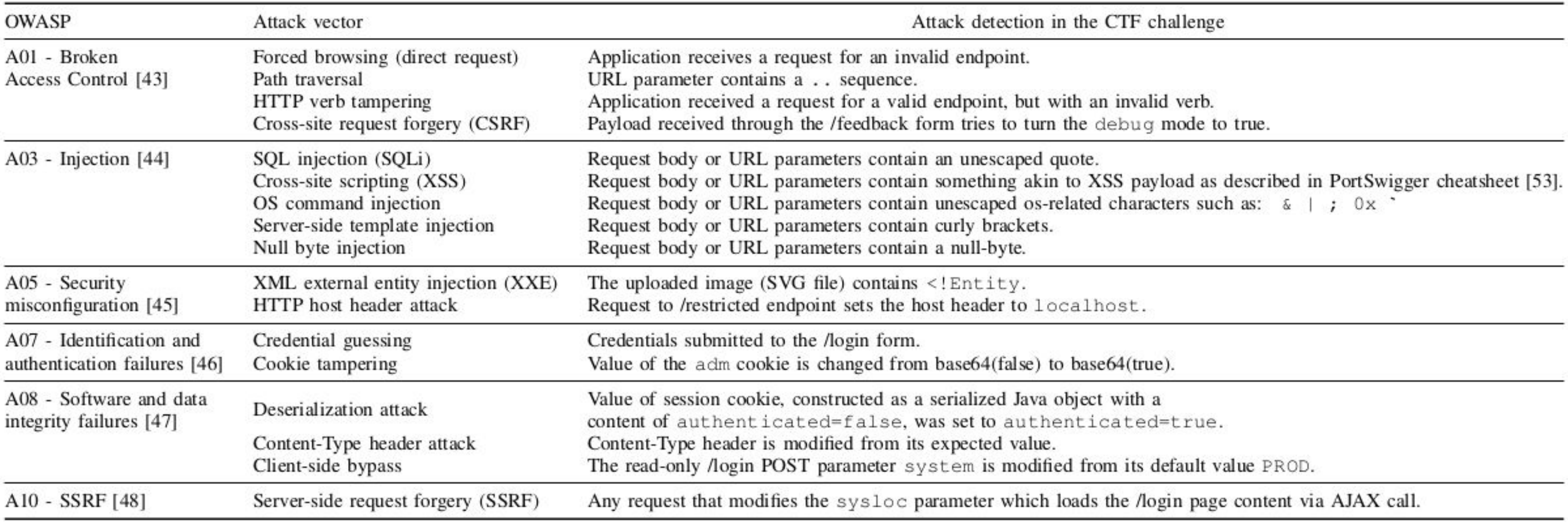
TABLE 4. COUNTRIES THE PARTICIPANTS COME FROM.



Industry	Percentage
Information Technology	27.78%
Software Development	16.67%
Financial and Banking	16.67%
Cloud-based Solutions or Services	5.56%
Security	5.56%
Internet	5.56%
Media, Publishing Advertising or Entertainment	5.56%
Research - Academic or Scientific	5.56%
Web Development and Design	5.56%
Energy or Utilities	5.56%

TABLE 5. INDUSTRIES THE PARTICIPANTS WORK IN.

CTF Attack Vectors



OWASP	Attack vector	Attack detection in the CTF challenge
A01 - Broken Access Control [43]	Forced browsing (direct request)	Application receives a request for an invalid endpoint.
	Path traversal	URL parameter contains a .. sequence.
	HTTP verb tampering	Application received a request for a valid endpoint, but with an invalid verb.
	Cross-site request forgery (CSRF)	Payload received through the /feedback form tries to turn the debug mode to true.
A03 - Injection [44]	SQL injection (SQLi)	Request body or URL parameters contain an unescaped quote.
	Cross-site scripting (XSS)	Request body or URL parameters contain something akin to XSS payload as described in PortSwigger cheatsheet [53].
	OS command injection	Request body or URL parameters contain unescaped os-related characters such as: & | ; 0x `
	Server-side template injection	Request body or URL parameters contain curly brackets.
	Null byte injection	Request body or URL parameters contain a null-byte.
A05 - Security misconfiguration [45]	XML external entity injection (XXE)	The uploaded image (SVG file) contains <!Entity.
	HTTP host header attack	Request to /restricted endpoint sets the host header to localhost.
A07 - Identification and authentication failures [46]	Credential guessing	Credentials submitted to the /login form.
	Cookie tampering	Value of the adm cookie is changed from base64(false) to base64(true).
A08 - Software and data integrity failures [47]	Deserialization attack	Value of session cookie, constructed as a serialized Java object with a content of authenticated=false, was set to authenticated=true.
	Content-Type header attack	Content-Type header is modified from its expected value.
	Client-side bypass	The read-only /login POST parameter system is modified from its default value PROD.
A10 - SSRF [48]	Server-side request forgery (SSRF)	Any request that modifies the sysloc parameter which loads the /login page content via AJAX call.

CTF Related Limitations

Participants might:
→ Not consider certain attacks as they did not see an explicit scenario.

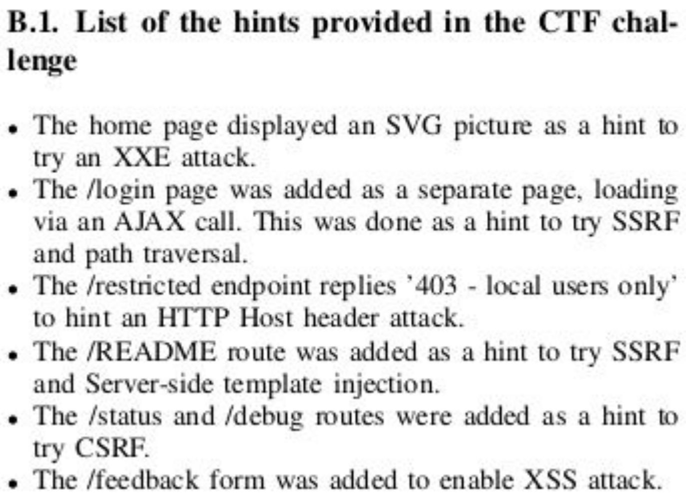
→ Prefer attacks that are easier or more obvious.

→ Press the give-up button rather early, thinking they can replay.

Detection rules might result in:
→ **False Positives**: For example, collisions on injection based attacks are possible.

→ **False Negatives**: For example, we might miss certain payloads.

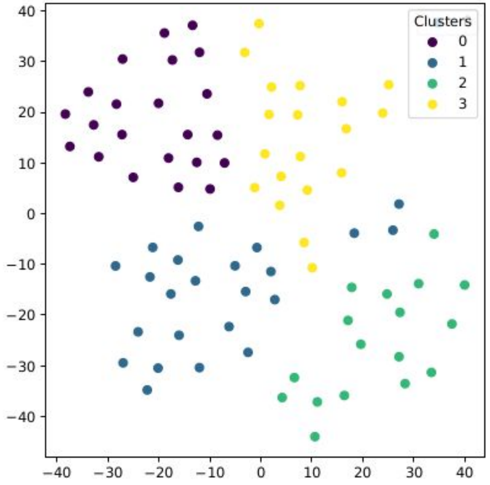
CTF Further Details



B.1. List of the hints provided in the CTF challenge

- The home page displayed an SVG picture as a hint to try an XXE attack.
- The /login page was added as a separate page, loading via an AJAX call. This was done as a hint to try SSRF and path traversal.
- The /restricted endpoint replies ’403 - local users only’ to hint an HTTP Host header attack.
- The /README route was added as a hint to try SSRF and Server-side template injection.
- The /status and /debug routes were added as a hint to try CSRF.
- The /feedback form was added to enable XSS attack.

B.3. Clustering of participants by attack types



Cluster 0: 27% - Single attack: SQLi, Cookie Tampering, or Forced Browsing
Cluster 1: 28% - Avg 3 attacks: Credential guessing + XSS and Client-side Bypass
Cluster 2: 21% - Avg 6 attacks: Large variety
Cluster 3: 23% - Avg 3 attacks: XSS + SQLi and Forced Browsing