Measuring Developers' Web Security Awareness from Attack and Defense Perspectives

Merve Sahin*, Tolga Ünlü†, Cédric Hébert*, Lynsay A. Shepherd†, Natalie Coull†, Colin McLean†
*SAP Security Research, France
{merve.sahin, cedric.hebert}@sap.com
†Division of Cyber Security, School of Design and Informatics, Abertay University, Dundee, United Kingdom
{t.unlu1200, lynsay.shepherd, n.coull, c.mclean}@abertay.ac.uk

***Abstract*—Web applications are the public-facing components of information systems, which makes them an easy entry point for various types of attacks. While it is often the responsibility of web developers to implement the proper security controls, it remains a challenge for them to develop a good understanding of the whole attack surface.**

This paper aims to understand developers' familiarity with a number of web attack and defense mechanisms. In particular, we conducted two different experiments: First, we employed a questionnaire to understand the perceived attack surface and the types of security controls that are often considered. Second, we designed a Capture the Flag challenge aiming to push participants to discover as many attack points as possible on a given web application. We found that one third of developers are not aware of the clients' ability to intercept and modify all parts of an HTTP request. Moreover, developers' attack awareness focuses on a limited set of attacks (such as Cross-site scripting and SQL injection), overlooking a large part of the attack surface.

***Index Terms*—Web, Framework, Security Awareness, Secure Software Development, CTF**

1. Introduction

The security posture of web applications stands and falls with their developers. While recent developments in browser platforms and web application frameworks have introduced built-in security controls, these do not always guarantee protection as they require a developers' awareness and implementation of such controls. Moreover, the security-relevant information sources available to developers can lack in quality [56], usability [16], and positioning [27] aspects, as observed in recent work. On the other hand, developers' awareness and expertise on attacks and security controls have shown to be correlated with their ability to implement effective and robust defenses [35], [26].

With simple to exploit and impactful threats such as Insecure Direct Object Reference (IDOR) attacks commonly found in the wild [30], [29], it raises the question if developers often lack the awareness for attack probes that require nothing more than the submission of an arbitrary input using a browsers' developer console or an interception proxy.

In this work, we aim to gain an understanding of developers' security and attack awareness from two perspectives: the defender and the attacker. Essentially, we seek to find out: (i) to which level they are familiar with the different types of security controls (e.g., client/server side input validation, vulnerability-specific controls, logging&monitoring) from a defensive perspective, and (ii) to which level they are able to identify and carry out different types of web attacks from an offensive perspective. We address (i) by designing and conducting a questionnaire, where the developers report their familiarity with security controls and answer scenario-based questions. We then address (ii) with a Capture the Flag (CTF) based experiment that includes a large variety of attack points for the participants to exhaust. These two experiments allow us to make several observations on the possible root causes of web application vulnerabilities. Some of our main findings can be summarized as follows:

- A large majority of the developers reported to be familiar with input validation controls, however, they are not always aware that all parts of an HTTP request (including the headers and method) may be tampered with.
- Some developers are completely unfamiliar with security controls that are vulnerability-specific, access-control related, or logging/monitoring related. This result also goes in parallel with the OWASP Top 10 vulnerabilities.
- Developers' familiarity with web attacks is mostly limited to injection and credential guessing. This might have an impact on their ability to implement strong defenses against the large attack surface of web applications.
- Popular web frameworks often lack proper security-relevant documentation, as they address a limited set of attack vectors and do not mention all the built-in security controls.

2. Related Work

Previous work employs surveys, design studies, and programming tasks to evaluate the impact of developers' information sources on code security [16], the usability of security-related information in API documentations [27], [65], and the impact of console warnings on the use of security-related APIs [24], [25]. Our work uses similar methods, but instead focuses on understanding developers' overall awareness of the web security controls. As stated by Acar et al. [17], security is often a secondary concern for developers as opposed to aspects such as functionality, compliance and speed. Thus, understanding developers' experiences with security controls can help to reveal the root causes of security issues, and see how these issues can be addressed.

More recent studies that are closely relevant to our work can be summarized as follows. Roth et al. [56] investigated the practical issues in Content Security Policy (CSP) deployment, by employing semi-structured interviews with coding and drawing tasks, on 12 web developers. The study found that "knowledge gaps" (e.g., about CSP related conceptual issues or built-in security features) are one of the main roadblocks for the adoption of CSP.

Likaj et al. [35] studied the defense techniques related to CSRF vulnerabilities by analyzing 44 popular web frameworks, in terms of the supported defense methods, if they are enabled by default, how much additional coding is required by developers, and if the documentation is clear. In addition to identifying several security risks in the frameworks' defense mechanisms, they also found the correct and robust use of the defense mechanisms require the developers' "awareness and expertise about CSRF attacks". This result reinforces our motivation to understand developers' awareness on different types of web attacks. We also find CSRF to be one of the least known attack types among the participants (Section 4).

In another study, Braz et al. [19] conducted a task-based experiment to understand to what extent developers can detect Improper Input Validation (IIV) practices during a code review, on two example IIV attack scenarios: SQL injection (SQLi) and improper validation of integer boundaries. They found that participants are seven times more likely to identify SQLi, as it is a well-known, stereotype attack vector with high visibility (e.g., in textbook examples and in the OWASP Top 10 list). In relation to this observation, our study also investigates to what extent web framework documentations (specifically, the dedicated security chapters) make their built-in security controls or their coverage of attack scenarios visible to the developers.

Finally, several studies used Capture The Flag (CTF) challenges to evaluate the effectiveness of attack-aware applications, in particular through the use of deceptive elements (such as honeytokens), in attack detection [18], [32], [57]. Our CTF experiment instead focuses on the attack awareness integration aspect from developers' perspective.

3. Experiment I: An Online Survey

The motivation of this online survey was to investigate whether participants with web development skills can detect specific indicators of attack attempts, their familiarity with specific security controls and how they perceive aspects of input validation that can determine the effectiveness of validation based detectors. The survey was developed based on related work in usable security research on security controls and APIs ([66], [33], [28]) as well as the OWASP Top 10 proactive controls project [42]. The first section of the survey investigates participants' expertise with important security controls in application security, specifically with input validation controls due to its prevalence in many security incidents as well as being a first step in the prevention and detection of security issues. However, self-reporting expertise and familiarity has its limitations [17] and thus may not be accurate. For that reason, a scenario-based question has been designed for the second section of the survey, which requires understanding of the scenario and the ability to apply the expertise that the participant responded with in the first section. The scenario was inspired by a theoretic case study of a rapidly deployed web application documented in the OWASP AppSensor guide [64]. In this case study, input validation controls are utilized to detect and warn the development team about forced browsing to non-existing endpoints, missing required parameters and the submission of additional parameters, among others. Lastly, the third section consists of questions to identify demographics and developer profiles to ascertain participant diversity and reveal potential relationships between participant groups and findings.

A full listing of the questionnaire can be found in Appendix A.1.

3.1. Ethical Considerations

The survey was conducted after receiving full approval by Abertay Universitys' research ethics committee. We informed participants about the research objectives, what they will be required to do, the approximate duration of the survey, and how the data will be retained. Participants also had to complete a consent form with a privacy notice on the legal basis of data processing, which is in accordance with the UK Data Protection Act 2018 and the EU General Data Protection Regulation (GDPR).

3.2. Pilot Survey

We conducted a pilot survey to receive feedback on the questions and the estimated time to complete the questionnaire. Participants of the pilot were recruited via snowball sampling of personally known contacts. A total of 10 participants were invited and 7 have completed the pilot survey. On average, completion of the survey took about 20 minutes. Feedback received after this pilot helped identify common issues the participants encountered, and allowed survey questions to be refactored to its current form.

3.3. Recruitment

The recruitment process for the final survey focused on gathering potential participants with experience in web application development as a primary requirement. However, reaching out to this specific community and finding people who would volunteer to participate was a difficult task. The initial recruitment of participants took place on social media platforms such as Twitter and LinkedIn, and some development specific platforms such as the DEV community and Reddit channels dedicated to web development topics. Although we wanted to advertise the survey among other developer communities such as on Discord or Slack channels, our initial requests for permission were declined by the moderators, as posting of surveys is undesired.

Participation in the survey was on a voluntary basis, and participants did not receive any monetary benefit. In total, 21 participants completed the survey.

3.4. Participants' Demographics and Profile

Our survey contains 14 questions (listed as Q7 to Q20 in Appendix A.1.), in order to understand participants' demographics, professional experience and work environment. Although 21 participants corresponds to a small sample size, and there might be a potential bias of gathering the participants from social media platforms; the demographics of the participants indicate a diverse group in terms of the industries and job titles (see Appendix A.2). Participants came from 7 different countries, mostly from the UK (8 / 21) and Germany (5 / 21). The majority of the participants had 1-5 years experience (11 / 21) with web development, often working in organizations with more than 100 employees (10 / 21), and primarily within teams of 6-10 people (7 / 21). Most participants (9 / 21) reported to have a single person dealing with security in their teams, followed by 1-5 people (8 / 21) and > 10 people (3 / 21). Finally, 7 participants considered themselves as security champions (promoting security practices within their team), while 8 participants do not and 6 did not know.

3.5. Results and Discussion

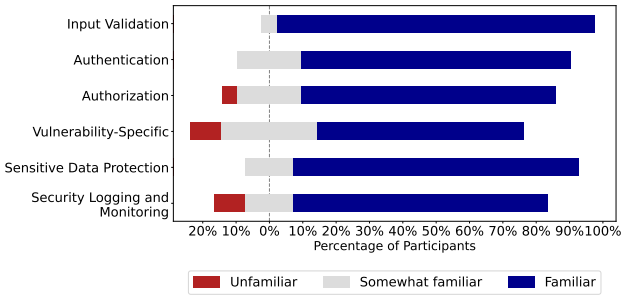


Figure 1. Participants familiarity with security controls.

3.5.1. Familiarity with Security Controls. The first question (Q1) asked the participants about their familiarity (understanding and implementation experience) with a set of security controls such as input validation, authentication, authorization, logging, and vulnerability-specific controls. As shown in Figure 1, participants are the most familiar with input validation controls (20 / 21), followed by controls for sensitive data protection (18 / 21) and authentication (17 / 21) . Participants also indicated a high familiarity with security logging and monitoring (16 / 21).

We found that the participants were least familiar with vulnerability-specific security controls such as prepared statements and context-aware escaping, as exemplified in the question. Six participants reported to be somewhat familiar, and 2 participants reported to be unfamiliar with these controls. Note that, as Q1 includes a description of what familiarity means[1], reported unfamiliarity can be to some extent understood as participants lacking implementation experience, even if they have an understanding of the security control. This does not necessarily mean that the applications the participants work on do not have these controls in place, these could be covered by controls implemented by their colleagues, the libraries or framework components used in their applications.

Looking further into unfamiliarity, only 3 categories of controls involved participants that were completely unfamiliar with controls: Security logging and monitoring (2 / 21), vulnerability-specific controls (2 / 21) and authorization controls (1 / 21). This reflects to some extent the OWASP Top 10 [41], [50], which indicates, for example, *Broken Access Control* and *Insufficient Logging and Monitoring* issues moving up on the list from 2017 to 2021. While *Injection* has moved down in the list to the third place, it is still a relevant security threat when considering that injection attacks, such as XSS and SQLi, have been consistently within the Top 10 of the CWE Top 25 weaknesses of the last three years [36], [37], [38] - all requiring a vulnerability-specific control for effective prevention such as context-aware escaping and prepared statements.

3.5.2. Focusing on Input Validation. Questions two to four further focus on input validation controls. In question two (Q2), the participants replied that they work on tasks involving input validation controls *Frequently*: 7 / 21, *Occasionally*: 7 / 21, *Rarely*: 6 / 21 and *Never*: 1 / 21. Based on these results, more than 60% of participants deal with input validation on a regular basis.

Question three (Q3) asked participants which properties of an input (e.g., origin, content, structure, semantic) they considered as security-critical to be validated. The results indicated that a majority of participants find validating the content (19 / 21) and the structure (19 / 21) of an input as critical. The least security-critical is considered the semantic validation of an input (12 / 21) although validating the semantics can be relevant in the prevention of business logic attacks as a recent incident at Coinbase demonstrated [21], where a missing validation check allowed trades to a specific order book using a mismatched source account. Among all 21 participants, only 4 considered all of the available properties in the answer options as security-critical.

For the last question in this section (Q4), we asked if the participants considered client-side input validation as optional or essential. The responses reveal that 9 participants consider it as essential, while 12 participants think it is optional.

```
                    3.
1. POST /settings HTTP/1.1
2. Host: www.app.com
   User-Agent: Mozilla/5.0
   Content-Type: application/json 4.
   Content-Length: 114

   {                    5.
     "Username" : "admin",
     "Password" : "1234",
     "Email" : "admin@app.com,
     "Country" : "United Kingdom"
   }
```

Figure 2. Example HTTP request shown in question five (Q5) with references to its individual parts.

1. "Note: Familiarity means that you understand a security control and have experience implementing it"

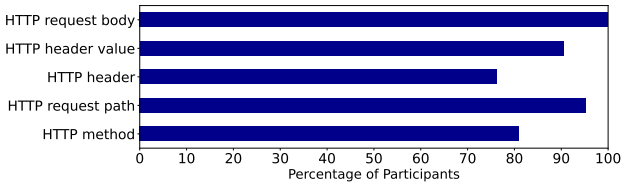


Figure 3. Percentage of participants who select parts of the HTTP request to be modifiable at the client-side.

3.5.3. Perception of Attack Attempts. The last two questions (Q5 and Q6) of the main part of the survey aimed to understand which type of anomalies the participants would consider as attack attempts. Q5 shows an example HTTP request (Figure 2) and asks which parts of the request can be modified by the client-side. All participants agreed that the HTTP request body can be modified, however, the HTTP header and HTTP method were selected by 76% (16 / 21) and 81% (17 / 21) of the participants respectively - see Figure 3. Since all parts of an HTTP request can be modified by the client-side, it is also important to evaluate how many of the participants selected all answers. In total, only 14 participants (67%) selected all answers.

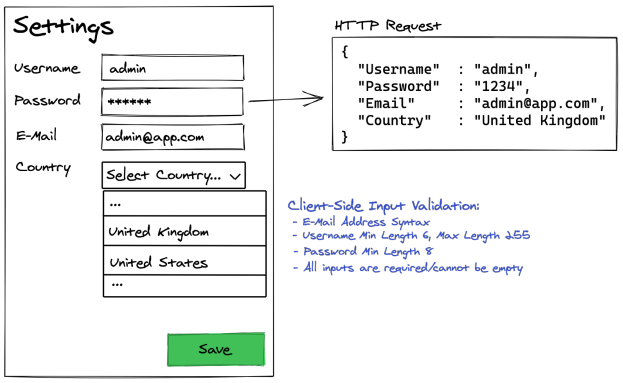


Figure 4. A graphical illustration of the settings form scenario in question six (Q6).

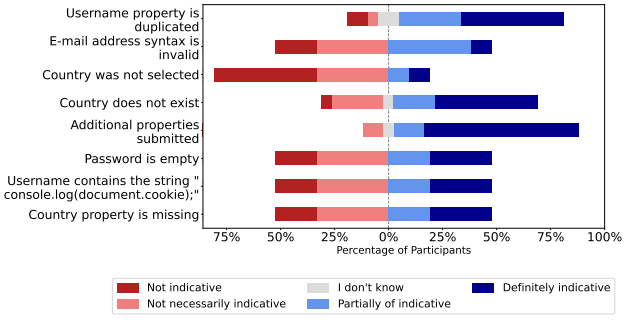


Figure 5. Participants selection of the server-side input validation controls which are indicative of blatant attack attempts.

Finally, Q6 provides a scenario to the participants: Given Figure 4 with an HTML form, a set of client-side input validation rules, and an example request; we asked how likely a set of events (given in Figure 5) observed on the server-side would be indicative of blatant attack attempts. Note that, all the events listed in the scenario would either require the HTTP request to be intercepted or modified (as well, to avoid client-side validation), or they contain obvious attack payloads.

None of the participants selected all the events as *Definitely indicative* of attack attempts in Q6. Only 3 participants selected either *Partially of indicative* or *Definitely indicative* for all the events. This result is surprising, as Q6 provides several assisting elements in the scenario description (such as the client-side validation rules, an example HTTP request with the expected parameters - depicted in Figure 4), which give hints that the listed events are only possible if the request is tampered with. Note that, some of these events might be interpreted by the participants as not malicious due to the nature of the event (e.g., invalid email address syntax, empty 'Country' field). However, even the event where a suspicious looking JavaScript payload was received as username, was not considered as an attack indication by half of the participants.

3.5.4. Discussion. In our survey, developers reported to be mostly familiar with different types of security controls, especially with input validation. However, one key finding is that, around one third of the participants are not aware that all parts of an HTTP request (including the headers, verb) can be tampered with. Thus, developers seem to be more likely to validate the HTTP request body, but might miss the validation of other fields and related attack vectors. The Apache Struts vulnerability (CVE-2017-5638) exploited in the high-profile attack against Equifax is a good example of how important the validation of other parts of an HTTP request can be, as the attack payload was submitted as a value of the Content-Type header [22].

As 18 participants reporting utilising a web framework, its influence on the developers' perception of what can and should be validated may play an important role. For example, web frameworks often come with a router which maps specific URL endpoints to code that handles requests. Depending on how the framework expects routes and route handlers to be defined, parts of a HTTP request such as the expected method may not require explicit validation - see Listing 1 for a code example where the HTTP method is part of the route definition. In addition, the findings by Braz et al. [19] may also apply here by means of which part of the request the developers mostly work on. For instance, developers will most likely spend their time with business logic code which gets executed as part of a route handler, it is therefore sensible to hypothesize that the focus is more on the request body, the primary carrier of payload for the business logic.

```
1 app.post('/settings', (request, response) => { /* Route handling */ });
```

Listing 1. Basic route definition for a web application built with the Express framework [10].

Our concern with Q5 was initially that asking developers directly what parts of an HTTP request can be tampered with might be straightforward to answer. In other words, our expectation and thus the underlying hypothesis was that developers know that the client-side can submit HTTP requests with arbitrary content. However, being aware of the small number of participants in the survey, a third (7 / 21), through their choice of answers,

indicated that the client-side can submit arbitrary content only to a limited extent. For future investigations, research hypotheses could revolve around what makes developers think that, e.g., only the HTTP request body can be tampered with and what needs to be done to make it more evident to developers that anything coming from the client-side cannot be trusted.

Considering Q5 and Q6 together, we find that only 2 participants were able to identify all the correct answers. These two participants also stated their familiarity with all security controls in Q1. However, using Fishers' exact test, the familiarity with all controls seems not to be significantly correlated with the ability to understand request tampering in Q5 and Q6 ($p = 0.214$). Furthermore, both participants consider client-side input validation as optional in Q4. Our expectation was that the participants who understand request tampering may also see client-side input validation controls as essential, as they can be beneficial in intrusion detection ([39], [64]) and form the essence of the scenario in Q6. To give a concrete example, assuming the input validation control in Listing 2, having this input validation control available on both the client-side and the server-side of a web application can make request tampering detectable as the control on the client-side would prevent benign users from submitting a request without the required properties - and whenever the same control is triggered on the server-side, it is safe to assume that the client-side control has been bypassed.

```
const requiredProperties = ['Username','Password','Email','Country'];

/* Check if requestBody is missing required properties */
const requestProperties = Object.keys(requestBody);
const missingProperties = requiredProperties.filter(property =>
    !requestProperties.includes(property)
);

if(missingProperties.length > 0)
{
   console.log('Required properties are missing');
   /* Abort submission/processing of request */
}
```

Listing 2. Input validation control to check if the request body is missing required properties.

Note that, although scenario based questions can lead to more concrete results, they also take more time to respond, extending the duration of the survey. Future work can focus on more specific scenarios and combine them with practical experiments in which participants complete a programming or reviewing task, similar to [24], [19]. As around 62% of participants reported to find it very useful to work on side-projects, they may be motivated to participate in a practical experiment, solving the challenges in a development environment familiar to them, thus providing a more robust dataset to draw conclusions from. Furthermore, the results in a practical experiment would also more accurately reflect the developers' real expertise and would also address the issues of self-reporting.

4. Experiment II: A CTF Challenge

While the survey measures the developers' awareness from a defense perspective (i.e., security controls they need to keep in mind), it does not really evaluate their awareness on the the web application attack vectors.

An attack vector might require one or more types of security controls for a robust defense mechanism. For example, mitigating XSS attacks involves proper input validation, but also vulnerability-specific controls such as CSP or the HttpOnly cookie flag [40], [49]. However, as observed in [35] and [56], such vulnerability-specific controls may be difficult to configure or enable, or may require additional coding from the developers, even though they have built-in support by the web frameworks or browsers. Thus, developers' awareness and expertise of the attack vector can affect the strength of the defense mechanisms they can implement.

To this end, this section aims to explore the familiarity of developers with a variety of web attacks. Employing a CTF challenge is a useful method for this purpose, as it requires the developers to understand the inner-workings of the vulnerability, and to have hands-on experience with the attack mechanism.

4.1. CTF Challenge Design

The purpose of this CTF challenge is to exhaust the players' imagination and to push them to try as many attacks as possible, on a given web application. To explore this, we designed a small web application with the following goals in mind: First, the application should allow for a large variety of web-application attacks to be performed. Second, it should have a minimal set of endpoints, so that the participants are not distracted by the structure of the application, and the order by which attacks would be tried would not depend on the structure.

Overall, our application contains 7 endpoints: a home page from which most of the attacks can be launched, and 6 other endpoints accessible from the home page, to allow different types of attack vectors. For the list of attack vectors, we took the inspiration from PortSwigger Labs [54], an online web security training platform. We ignored the attacks which would be hard to detect from the server side such as clickjacking. We also ignored the attacks that are automatically prevented by our application engine (Node.js), such as HTTP request smuggling, where the engine drops the request before our server can analyze it to grant points. The complete list of the attacks, their mapping to the OWASP Top 10 list, and the CTF implementation details can be found in the Appendix B.4.

The idea of the challenge was to put the participants in the position of a penetration tester, and let them define their own approach to navigate the application and find attack vectors. The challenge is named "GIVE UP" and the challenge description states: "This application has so many vulnerabilities. Exploit them all, and you'll be rewarded. But you may as well give up...".

The application keeps track of the different types of attack attempts silently, however it is not particularly vulnerable or exploitable. Thus, the challenge encourages participants to try out different attack vectors. Once all the 17 configured attack vectors are tried, the flag of the challenge is unlocked. To avoid frustration, the application includes a button to "Give up", where the participant is asked if they want to end the challenge. Once a participant gives-up, they would see a message explaining which attacks were detected and how many more were left. This allowed for the creation of two datasets: attacks tried

'blindly' until the first give-up, and attacks tried once the player knew that the goal was to trigger a maximum of different attacks.

Moreover, we provided a set of hints within the application to motivate attack vectors that require some context to be tried at all. For example, as SSRF requires the existence of a second server, we developed the /login page to be dynamically loaded from a different server via client-side JavaScript code. The request to load the page also includes the server URL as a GET parameter. Other examples are: the `path` URL parameter of the file upload page that can be used for path traversal or null-byte injection, and the /restricted endpoint that replies with '403 - local users only' to hint an HTTP host header attack.

While the hints aim to signal the possibility of certain attack points, participants would still need to be familiar with the attack vector to be able to make use of the hints. Thus, we do not expect the hints to significantly impact the participants' ability to try out the attacks. The list of the hints, and some screenshots from the CTF application can be found in Appendix B.2 and B.1.

Participation. The CTF challenge was part of a large CTF competition at a software company, employed on an enterprise CTF platform used for internal security training exercises. The competition took place during the cybersecurity awareness month (October) in 2021, and included a variety of challenges (e.g., web, forensics, cryptography). Participants were security enthusiasts who voluntarily joined the CTF competition. This might bring a bias towards a higher attack awareness ratio, compared to the general population average. Overall, there were 82 participants who carried out at least one type of attack during the CTF.

4.2. Results

4.2.1. Overall popularity of attacks. For each attack vector, Table 1 shows the ratio of participants who tried this attack (i) until they give up for first time, and (ii) during the total duration of the CTF. (i) can be considered as a lower-bound, and (ii) can be considered as an upper-bound on participants' awareness, as the participants can take their time to do additional research, and make as many attempts as they want during the total CTF duration.

We found that XSS, SQLi, forced browsing, and credential guessing were the attacks that were the most frequently tried. Note that these attacks are rather easy to try out, e.g., they do not require intercepting the HTTP traffic. On the other hand, the attacks that require intercepting and modifying the HTTP request (e.g., Cookie and verb tampering, client-side bypass, Content-Type and Host header attacks) had significantly lower percentage on average (10%), compared to the average of top four popular attacks (48%)[2]. This result is also in line with the survey (Section 3.5.3), showing that, 10 to 20% of participants do not think that HTTP method, HTTP header, and header values can be modified at the client-side; while all participants know that the request body can be modified.

Moreover, certain attack vectors such as insecure deserialization, CSRF and SSRF were tried by very few participants. These are rather complex attacks, probably more difficult to understand, and at the same time more difficult to implement a proper defense against [35], [34].

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%

TABLE 1. PERCENTAGE OF PARTICIPANTS WHO HAVE TRIED EACH ATTACK VECTOR.

4.2.2. Clustering participants' attack behavior. In this section, we aim to analyze if there are certain groups of participants with similar attack behavior. We focused on the attacks carried out until the participant gave-up for the first time, as this data is more reliable to capture the inherent knowledge of the participants. We used the KMeans clustering algorithm [59] where each attack vector becomes a separate feature and takes the value of 1 if the participant have tried out this attack, and the value of 0 otherwise. Applying the elbow method [51], we found the optimal number of cluster to be 4. Appendix B.3 visualizes the clusters in 2 dimensions, using the t-SNE method [62]. Looking into the clusters manually, we interpret the 4 different profiles:

Cluster 0: 27% of participants who fall into this cluster tried only a single attack - either SQLi, cookie tampering, or forced browsing. We do not observe any attempts of the XSS or credential guessing attacks in this cluster, although they have the highest overall popularity.

Cluster 1: 28% of participants fell into this cluster, where the differentiating feature (tried by all participants) becomes the credential guessing attempts. These participants tried 2 other attacks on average, mostly XSS (43%) and client-side bypass (35%).

Cluster 2: This group of participants (21%) tried 6 different attacks on average. Almost all of them tried the top 4 most popular attacks, and often cookie tampering and client-side bypass in addition to these. This cluster also had the largest variety of attacks, including deserialization and Content-Type header attacks, path traversal, XXE, and OS command injection. However, we still do not observe certain categories such as CSRF and SSRF.

Cluster 3: 23% of participants fell into this cluster. Similar to Cluster 1, it has a differentiating feature, that is the XSS attack carried out by all participants. On average the participants tried 2 other attacks, mostly SQLi (47%) and forced browsing (26%). The only participant who tried the CSRF attack is also in this category.

This analysis provides a preliminary look into different types of participant profiles in terms of attack awareness. For instance, Cluster 0 seems to correspond to the least experienced, while Cluster 2 represents the participants

2. Applying a 2-proportions Z-test: z-score=-5.3, p-value<.00001, result is significant at p<.05

with the highest attack awareness. Overall, we still observe that participants' awareness of the variety of attack vectors is often quite limited: 79% of participants fail to consider most of the attacks, even the ones that are rather easy to try out (e.g., cookie tampering). Future work seeks to investigate the correlation between the different profiles and the developers' security and work experience, which may show the need for different types of security education.

Limitations. Although we designed the CTF application in a way to minimize its influence on the participants' approach, it is still possible that the CTF setup creates a bias on the participants (i) to try attacks that are easier or more obvious to them, (ii) to not consider certain attacks as they did not see an explicit scenario, or (iii) to press the give-up button rather early, thinking that they will be able to replay the challenge. However, there was no time limitation on completing the challenge or replaying it, thus they could take as much time to think and search for different attacks.

Another limitation is about the possible false positives in the detection of attack attempts. Attacks were detected with a set of hard-coded rules (Appendix B.4) to search for known payloads (e.g. collected from PortSwigger or OWASP cheat sheets). However, especially for the injection category of attacks it is possible to have collisions, i.e., the same payload being identified as multiple attacks. We did our best to minimize this possibility, and as shown in Section 4.2.2, we can observe different injection attacks in different clusters. For the future work, it can be a more reliable approach to replay the traffic to a Web Application Firewall (WAF) for attack detection. However, WAFs can also result in false positives.

4.2.3. Web Frameworks' Security Control Support.

Based on top three most and least frequently attempted attack vectors in the CTF challenge, we have also compared how common web frameworks make developers aware of these attacks, and what security controls they suggest. We choose the frameworks to investigate based on what the survey participants have reported to use, which is also in line with the most popular web frameworks reported in last years' StackOverflow developer survey [60].

We looked for the dedicated security chapters in the selected frameworks' documentations, to see if the relevant attack vectors are covered or referenced, and if the frameworks come with preventive security controls built-in. This does not exclude the possibility of security relevant information being mentioned in other chapters, however, we think that these should be referenced in a general security chapter to give developers a more accessible entry point, e.g., for those who might not be familiar with terminology and abbreviations used in web security.

With the exception of React.js and jQuery, all other evaluated frameworks have provided a dedicated section or chapter for security in their documentation. However, we can already see in Table 2 that these dedicated chapters do not always mention or reference to built-in security controls. For those attack vectors that get a mentioning, we also observe that only vulnerability-specific controls get recommended such as context-aware escaping, prepared statements or synchronizer tokens. A more general explanation of web security and how submission of arbitrary input is a core enabler of the attacks is not provided.

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)	-	-	-	-	-	-

TABLE 2. WEB FRAMEWORKS: A: ASP.NET[4], [3], [1], [2], ANJS: ANGULAR.JS[9], AN: ANGULAR[8], B: BLAZOR[5], D: DJANGO[6], E: EXPRESS[13], F: FLASK[14], L: LARAVEL[11], S: SPRING[7], [12], SY: SYMFONY[15].

For example, a general explanation could have included a vulnerability description with a practical example and a reference to the relevant input validation components of a framework. Basic validation such as ensuring that a value is an integer can be an important first line of defense [58] and can establish a defense-in-depth together with the vulnerability-specific controls, another aspect that was not covered in the dedicated security pages.

Regarding deserialization attacks and SSRF, we observe that these do not get mentioned in the dedicated security chapters. While deserialization attacks can be mostly avoided through a safer built-in API enforced by the framework, it is more complicated with SSRF as there are multiple aspects that need to be validated that are not straightforward [34].

5. Limitations

Our study contains two experiments with two different sets of participants. Thus, the results of the experiments should be interpreted individually. For instance, the CTF participants are likely to be more knowledgeable and enthusiastic about security, compared to the survey respondents. The ideal case would be to have the same set of participants for both studies, however, this would either require to hire the developers for a lengthy study, or to work with university students, who may not represent the developers. Acquiring a representative and voluntary set of respondents for such a study remains a challenge.

Another limitation is that, our study does not weight the experiment results according to the participants' years of development experience. It is possible that more experienced developers are more aware of security, and as mentioned in Section 3.5, a developers' security awareness does not necessarily correlate to the security level of the web applications they develop, as the development teams may have specific people dealing with security, or more experienced developers may be supervising the less experienced ones.

6. Conclusion and Outlook

Our experiments allow us to draw a number of conclusions. First, a diverse group of survey participants working in various industries have reported their familiarity with different types of security controls. Especially input validation was selected by most participants of the survey, however, almost one third of the participants were reporting that only certain parts of a HTTP request can be mod-

ified. This includes participants with >1 year experience in web development and even one participant in a leading technical position. This indicates a lack of awareness of a clients' ability to submit arbitrary input. Although the CTF experiment was conducted on a different population of developers, who are possibly more enthusiastic about security, the attacks that were the least attempted would have required to intercept and modify the HTTP request. Future work could further investigate to what extent this awareness is lacking, and whether awareness-raising abstractions of APIs and security controls can be designed [20].

Second, we observe that *Security logging and monitoring* was one of the controls that the participants are the most unfamiliar with. In the future work, a task-based experiment where the developers augment a prototype application with monitoring and attack detection points could provide more reliable data on their familiarity with such controls. This could also help to identify if application frameworks provide useful artifacts, e.g., event or exception classes, that can be utilized for logging and monitoring purposes. This could complement related research on automatically modelling security incidents for logging [55] and application intrusion detection approaches to create attack-aware software applications [61], [31], [63].

Another insight from the CTF experiment is that SSRF attack has been attempted the least. SSRF has made its entry in the most recent OWASP Top 10 list, and with URL fetching features being common in modern web applications, web frameworks should provide security controls for this. However, as highlighted in [34], defending against SSRF programmatically can be non-trivial as various validations need to be in place with each of them having their own limitations. Still, frameworks can be in the best position in order to prevent the developer from becoming the weakest link, by introducing secure by default controls [52], [23], [58]. This would complement our previous suggestion to conduct further studies on the design of APIs and their abstractions, which is also in line with recommendations on how to advance usable security for developers [17].

Finally, while our study sheds light into the lack of security awareness from defense and offense perspectives, future studies are required to better understand the correlation between the two: In particular, future work can explore if an educational approach can make the overlooked aspects in both experiments more salient to the developers, and whether the focus should be on training developers with an offensive or defensive take on application security. Current work suggests that having a good understanding of certain attacks helps in building stronger defenses [56]. However, as security is often just a secondary concern [17], a more promising path may lie in utilizing and improving the resources in the immediate vicinity of a developers' work environment. Gorski et al. [27] demonstrates this on the example of integrating security information in a non-security API documentation, this makes security information more visible to developers, and thus supports the findings by Braz et al. [19]. Tackling the blindspots in API and framework documentations, such as those in Table 2, could be therefore a promising area of research. Frameworks themselves may be another promising subject of study: especially regarding our findings of the lack of awareness on certain attacks and HTTP request tampering, future work can investigate to what extent the development methodology of current frameworks has an impact on this.

References

[1] "Security | Microsoft Docs," https://docs.microsoft.com/en-us/aspnet/web-pages/overview/security/, 2020, Accessed: 31/03/2022.

[2] "Security, Authentication, and Authorization in ASP.NET MVC | Microsoft Docs," https://docs.microsoft.com/en-us/aspnet/mvc/overview/security/, 2020, Accessed: 31/03/2022.

[3] "Security, Authentication, and Authorization in ASP.NET Web API | Microsoft Docs," https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/, 2020, Accessed: 31/03/2022.

[4] "Security, Authentication, and Authorization in ASP.NET Web Forms | Microsoft Docs," https://docs.microsoft.com/en-us/aspnet/web-forms/overview/security/, 2020, Accessed: 23/02/2022.

[5] "ASP.NET Core Blazor authentication and authorization | Microsoft Docs," https://docs.microsoft.com/en-us/aspnet/core/blazor/security/?view=aspnetcore-6.0, 2022, Accessed: 23/02/2022.

[6] "Security in Django | Django documentation | Django," https://docs.djangoproject.com/en/4.0/topics/security/, 2022, Accessed: 23/02/2022.

[7] "Web on Servlet Stack," https://docs.spring.io/spring-framework/docs/current/reference/html/web.html#mvc-web-security, 2022, Accessed: 23/02/2022.

[8] "Angular - Security," https://angular.io/guide/security, n.d., Accessed: 23/02/2022.

[9] "AngularJS: Developer Guide: Security," https://code.angularjs.org/snapshot/docs/guide/security, n.d., Accessed: 23/02/2022.

[10] "Express basic routing," https://expressjs.com/en/starter/basic-routing.html, n.d., Accessed: 31/03/2022.

[11] "Installation - Laravel - The PHP Framework For Web Artisans," https://laravel.com/docs/9.x/, n.d., Accessed: 23/02/2022.

[12] "Protection Against Exploits :: Spring Security," https://docs.spring.io/spring-security/reference/features/exploits/index.html, n.d., Accessed: 23/02/2022.

[13] "Security Best Practices for Express in Production," https://expressjs.com/en/advanced/best-practice-security.html#production-best-practices-security, n.d., Accessed: 23/02/2022.

[14] "Security Considerations - Flask Documentation (2.0.x)," https://flask.palletsprojects.com/en/2.0.x/security/, n.d., Accessed: 23/02/2022.

[15] "Security (Symfony Docs)," https://symfony.com/doc/current/security.html, n.d., Accessed: 23/02/2022.

[16] Y. Acar, M. Backes, S. Fahl, D. Kim, M. L. Mazurek, and C. Stransky, "You Get Where You're Looking for: The Impact of Information Sources on Code Security," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 289–305.

[17] Y. Acar, S. Fahl, and M. L. Mazurek, "You are Not Your Developer, Either: A Research Agenda for Usable Security and Privacy Research Beyond End Users," in *2016 IEEE Cybersecurity Development (SecDev)*. IEEE, 2016, pp. 3–8.

[18] F. Araujo, M. Shapouri, S. Pandey, and K. Hamlen, "Experiences with Honey-Patching in Active Cyber Security Education," in *8th Workshop on Cyber Security Experimentation and Test (CSET 15)*, 2015.

[19] L. Braz, E. Fregnan, G. Çalikli, and A. Bacchelli, "Why Don't Developers Detect Improper Input Validation?'; DROP TABLE Papers;–," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 499–511.

[20] P. D. Chowdhury, J. Hallett, N. Patnaik, M. Tahaei, and A. Rashid, "Developers Are Neither Enemies Nor Users: They Are Collaborators," in *2021 IEEE Secure Development Conference (SecDev)*. IEEE, 2021, pp. 47–55.

[21] Coinbase, "Retrospective: Recent Coinbase Bug Bounty Award | by Coinbase | Feb, 2022 | The Coinbase Blog," https://blog.coinbase.com/retrospective-recent-coinbase-bug-bounty-award-9f127e04f060, 2022, Accessed: 20/02/2022.

[22] N. V. Database, "NVD - CVE-2017-5638," https://nvd.nist.gov/vuln/detail/CVE-2017-5638, 2017, Accessed: 22/02/2022.

[23] M. Finifter and D. Wagner, "Exploring the Relationship Between Web Application Development Tools and Security," in *2nd USENIX Conference on Web Application Development (WebApps 11)*, 2011.

[24] P. L. Gorski, Y. Acar, L. Lo Iacono, and S. Fahl, *Listen to Developers! A Participatory Design Study on Security Warnings for Cryptographic APIs*, 2020, pp. 1–13.

[25] P. L. Gorski, L. L. Iacono, D. Wermke, C. Stransky, S. Möller, Y. Acar, and S. Fahl, "Developers Deserve Security Warnings, Too: On the Effect of Integrated Security Advice on Cryptographic API Misuse," in *Fourteenth Symposium on Usable Privacy and Security (SOUPS 2018)*, 2018, pp. 265–281.

[26] P. L. Gorski, L. L. Iacono, S. Wiefling, and S. Möller, "Warn if Secure or How to Deal with Security by Default in Software Development?" in *HAISA*, 2018, pp. 170–190.

[27] P. L. Gorski, S. Moller, S. Wiefling, and L. L. Iacono, "" I just looked for the solution!" - On Integrating Security-Relevant Information in Non-Security API Documentation to Support Secure Coding Practices," *IEEE Transactions on Software Engineering*, 2021.

[28] M. Green and M. Smith, "Developers are Not the Enemy!: The Need for Usable Security APIs," *IEEE Security & Privacy*, vol. 14, no. 5, pp. 40–46, 2016.

[29] HackerOne, "Organizations Paid Hackers $23.5 Million for These 10 Vulnerabilities in One Year | HackerOne," https://www.hackerone.com/press-release/organizations-paid-hackers-235-million-these-10-vulnerabilities-one-year-4, 2020, Accessed: 21/02/2022.

[30] ——, "The Rise of IDOR | HackerOne," https://www.hackerone.com/company-news/rise-idor, 2021, Accessed: 21/02/2022.

[31] C. C. Hall, L. A. Shepherd, and N. Coull, "BlackWatch: Increasing Attack Awareness within Web Applications," *Future Internet*, vol. 11, no. 2, p. 44, 2019.

[32] X. Han, N. Kheir, and D. Balzarotti, "Evaluation of Deception-Based Web Attacks Detection," in *Proceedings of the 2017 Workshop on Moving Target Defense*, 2017, pp. 65–73.

[33] L. L. Iacono and P. L. Gorski, "I Do and I Understand. Not Yet True for Security APIs. So Sad," in *European Workshop on Usable Security*, vol. 4, 2017.

[34] B. Jabiyev, O. Mirzaei, A. Kharraz, and E. Kirda, "Preventing Server-Side Request Forgery Attacks," in *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, 2021, pp. 1626–1635.

[35] X. Likaj, S. Khodayari, and G. Pellegrino, "Where We Stand (or Fall): An Analysis of CSRF Defenses in Web Frameworks," in *24th International Symposium on Research in Attacks, Intrusions and Defenses*, 2021, pp. 370–385.

[36] MITRE, "CWE - 2019 CWE Top 25 Most Dangerous Software Weaknesses," https://cwe.mitre.org/top25/archive/2019/2019_cwe_top25.html, 2019, Accessed: 08/02/2022.

[37] ——, "CWE - 2020 CWE Top 25 Most Dangerous Software Weaknesses," https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html, 2020, Accessed: 08/02/2022.

[38] ——, "CWE - 2021 CWE Top 25 Most Dangerous Software Weaknesses," https://cwe.mitre.org/top25/archive/2021/2021_cwe_top25.html, 2021, Accessed: 08/02/2022.

[39] ——, "CWE - CWE-20: Improper Input Validation (4.6)," https://cwe.mitre.org/data/definitions/20.html, 2021, Accessed: 08/02/2022.

[40] MITRE, "CWE - CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')," https://cwe.mitre.org/data/definitions/79.html, 2021, Accessed: 23/02/2022.

[41] OWASP, "OWASP Top Ten 2017 | Table of Contents | OWASP Foundation," https://owasp.org/www-project-top-ten/2017/, 2017, Accessed: 15/02/2022.

[42] ——, "OWASP Proactive Controls | OWASP Foundation," https://owasp.org/www-project-proactive-controls/, 2018, Accessed: 29/03/2022.

[43] ——, "A01:2021 – Broken Access Control," https://owasp.org/Top10/A01_2021-Broken_Access_Control/, 2021, Accessed: 24/02/2022.

[44] ——, "A03:2021 – Injection," https://owasp.org/Top10/A03_2021-Injection/, 2021, Accessed: 24/02/2022.

[45] ——, "A05:2021 – Security Misconfiguration," https://owasp.org/Top10/A05_2021-Security_Misconfiguration/, 2021, Accessed: 24/02/2022.

[46] ——, "A07:2021 – Identification and Authentication Failures," https://owasp.org/Top10/A07_2021-Identification_and_Authentication_Failures/, 2021, Accessed: 24/02/2022.

[47] ——, "A08:2021 – Software and Data Integrity Failures," https://owasp.org/Top10/A08_2021-Software_and_Data_Integrity_Failures/, 2021, Accessed: 24/02/2022.

[48] ——, "A10:2021 – Server-Side Request Forgery (SSRF)," https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_%28SSRF%29/, 2021, Accessed: 24/02/2022.

[49] OWASP, "Cross-Site Scripting Prevention - OWASP Cheat Sheet Series," https://cheatsheetseries.owasp.org/cheatsheets/Cross_Site_Scripting_Prevention_Cheat_Sheet.html, 2021, Accessed: 23/02/2022.

[50] OWASP, "OWASP Top 10:2021," https://owasp.org/Top10/, 2021, Accessed: 15/02/2022.

[51] P. Vatsal , "Explaining and Implementing kMeans Algorithm in Python," https://towardsdatascience.com/k-means-explained-10349949bd10, 2021, Accessed: 23/02/2022.

[52] K. Peguero, N. Zhang, and X. Cheng, "An Empirical Study of the Framework Impact on the Security of JavaScript Web Applications," in *Companion Proceedings of the The Web Conference 2018*, 2018, pp. 753–758.

[53] PortSwigger, "Cross-site scripting (XSS) cheat sheet," https://portswigger.net/web-security/cross-site-scripting/cheat-sheet, 2022, Accessed: 31/03/2022.

[54] ——, "Web Security Academy - All labs," https://portswigger.net/web-security/all-labs, 2022, Accessed: 07/02/2022.

[55] F. Rivera-Ortiz and L. Pasquale, "Automated Modelling of Security Incidents to represent Logging Requirements in Software Systems," in *Proceedings of the 15th International Conference on Availability, Reliability and Security*, 2020, pp. 1–8.

[56] S. Roth, L. Gröber, M. Backes, K. Krombholz, and B. Stock, "12 Angry Developers-A Qualitative Study on Developers' Struggles with CSP," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 3085–3103.

[57] M. Sahin, C. Hebert, and A. S. De Oliveira, "Lessons Learned from SunDEW: A Self Defense Environment for Web Applications," in *Proceedings of the 2020 Measurements, Attacks, and Defenses for the Web (MADWeb) Workshop in the Network and Distributed System Security Symposium (NDSS)*, 2020.

[58] T. Scholte, W. Robertson, D. Balzarotti, and E. Kirda, "An Empirical Analysis of Input Validation Mechanisms in Web Applications and Languages," in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, 2012, pp. 1419–1426.

[59] Scikit-learn developers , "KMeans clustering," https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html, 2022, Accessed: 23/02/2022.

[60] StackOverflow, "Stack Overflow Developer Survey 2021," https://insights.stackoverflow.com/survey/2021#section-most-popular-technologies-web-frameworks, 2021, Accessed: 20/02/2022.

[61] T. Ünlü, L. A. Shepherd, N. Coull, and C. McLean, "A Taxonomy of Approaches for Integrating Attack Awareness in Applications," in *2020 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*. IEEE, 2020, pp. 1–4.

[62] L. van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 11 2008.

[63] C. Watson, M. Coates, J. Melton, and D. Groves, "Creating Attack-Aware Software Applications with Real-Time Defenses," *CrossTalk The Journal of Defense Software Engineering*, vol. 24, no. 5, 2011.

[64] C. Watson, D. Groves, and J. Melton, "OWASP AppSensor Guide - Application-Specific Real Time Attack Detection & Response - Version 2.0," https://owasp.org/www-pdf-archive/Owasp-appsensor-guide-v2.pdf, 2015, Accessed: 08/02/2022.

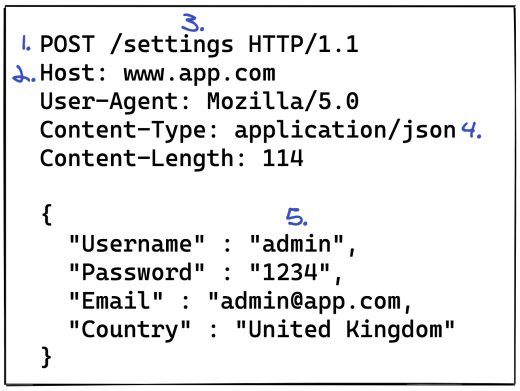
[65] C. Wijayarathna and N. A. G. Arachchilage, "Fighting Against XSS Attacks: A Usability Evaluation of OWASP ESAPI Output Encoding," in *Proceedings of the 52nd Hawaii International Conference on System Sciences (HICSS)*, 2019.

[66] C. Wijayarathna, N. A. G. Arachchilage, and J. Slay, "A Generic Cognitive Dimensions Questionnaire to Evaluate the Usability of Security APIs," in *International Conference on Human Aspects of Information Security, Privacy, and Trust*. Springer, 2017, pp. 160–173.

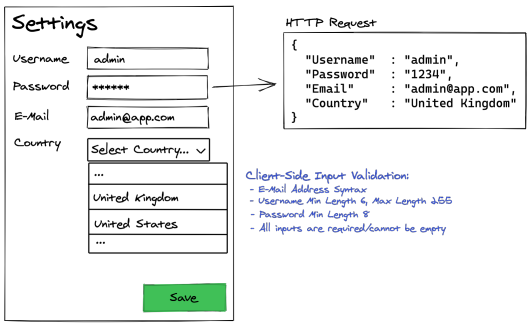
Appendix A. Questionnaire-based Online Survey

A.1. Survey Questions

- **Q1**: Please select how familiar you are with the following security controls - Note: Familiarity means that you understand a security control and have experience in implementing it (Likert: Unfamiliar, Somewhat familiar, Familiar)
  - Input validation controls (e.g. Is the provided input a number? Is the provided input whitelisted?)
  - Authentication controls (e.g. Did User A provide correct credentials?)
  - Authorization controls (e.g. Is User A allowed to do action X?)
  - Security controls that prevent specific types of vulnerabilities (e.g. Context-Aware Escaping, Prepared Statements)
  - Security controls that protect sensitive data (e.g. Applying Encryption, Preventing Storage of Secret Keys)
  - Security controls that log security events (e.g. Log multiple failed login attempts)
- **Q2**: Please select how often you work on tasks that involve the development of input validation controls - Note: Both client-side and server-side input validation (Single Choice: Never, Rarely, Occasionally, Frequently)
- **Q3**: Please select which of the following types of validations you consider as security-critical (Multiple Choice: Origin - Who sent the input?, Size - Is the input size reasonable?, Content - Does the input contain the right values and encoding?, Structure - Is the input in the correct format?, Semantics - Does the input make sense in the current context?)
- **Q4**: From a security perspective, do you consider client-side input validation as optional or essential? (Single Choice: Optional, Essential)
- **Q5**: Please select which of the parts of the HTTP request in the figure below can be modified by the client-side (Multiple Choice: 1. The HTTP method, 2. The HTTP header, 3. The HTTP request path, 4. The value of a HTTP header, 5. The HTTP request body)



- **Q6**: You are given the following figure of a settings page below. Please select whether the following input validation controls on the server-side would be indicative for blatant attack attempts (Likert: Not indicative, Not necessarily indicative, I don't know, Partially of indicative, Definitely indicative)



  - Username property is duplicated
  - E-Mail address syntax is invalid
  - Country was not selected
  - Country does not exist
  - Additional properties submitted
  - Password is empty
  - Username contains the string "console.log(document.cookie);"
  - Country property is missing
- **Q7**: Please describe your current job title (Short-answer text)
- **Q8**: Please select your current age (Single choice: 18 - 24, 25 - 39, 40 - 60, $>$ 60)
- **Q9**: Please select your current country of residence (Drop-down)
- **Q10**: Please select your highest level of educational attainment (Drop-down: Doctoral degree (Ph.D., Ed.D., etc.), Master's degree (M.A., M.S., M.Eng., MBA, etc.), Bachelor's degree (B.A., B.S., B.Eng., etc.), Associate degree (A.A., A.S., etc.), Professional degree (JD, MD, etc.), Secondary school (e.g. American High School, German Realschule or Gymnasium, etc.), Some college/university study without earning a degree, No formal education)
- **Q11**: Please select the programming and scripting languages of the web application(s) you are working on - Note: Use comma's (",") to separate multiple values when using the "Other..." option (Multiple

Choice: C#, Go, Java, JavaScript, Perl, PHP, Python, Ruby, TypeScript, WebAssembly, Other...)

- **Q12**: Please select the web frameworks of the web application(s) you are working on - Note: Use comma's (",") to separate multiple values when using the "Other..." option - Do not select any option if you are not making use of a web framework (Multiple Choice: Angular, Angular.js, ASP.NET, Django, Express, Flask, jQuery, Laravel, React.js, Ruby on Rails, Spring, Symfony, Vue.js, Other...)
- **Q13**: Please select the development environments or code editors that you are using for web application development - Note: Use comma's (",") to separate multiple values when using the "Other..." option (Multiple Choice: Atom, Coda, Eclipse, IntelliJ, Komodo, Light Table, NetBeans, PHPStorm, PyCharm, RubyMine, Sublime Text, Visual Studio, Visual Studio Code, Zend, Other...)
- **Q14**: Please select the years of experience you have in web application development (Single Choice: <1 Year, 1 - 5 Years, 6 - 10 Years, >10 Years)
- **Q15**: Please select the size of the development team you work with (Single Choice: 1 Person, 1 - 5 People, 6 - 10 People, >10 People)
- **Q16**: Please select how many in your development team are dealing with security (Single Choice: 1 Person, 1 - 5 People, 6 - 10 People, >10 People)
- **Q17**: Please select the size of the organization you work for (Single Choice: 1 Employee, 1 - 10 Employees, 11 - 20 Employees, 21 - 100 Employees, >100 Employees)
- **Q18**: Please select the type of industry you work for (Drop-down: Cloud-based Solutions or Services, Consulting, Data and Analytics, Education and Training, Energy or Utilities, Financial and Banking, Government or Public Administration, Health Care or Social Services, Information Technology, Internet, Manufacturing, Marketing, Media, Publishing Advertising or Entertainment, Nonprofit, Real Estate, Research – Academic or Scientific, Retail or E-Commerce, Security, Software as a Service (SaaS) Development, Software Development, Telecommunications, Transportation, Travel, Web Development and Design)
- **Q19**: Please select how useful you think the following activities are to learning new skills, technologies, programming languages or frameworks (Likert: Very useless, Somewhat useless, I don't know, Somewhat useful, Very useful)
  - Reading books
  - Reading online articles (e.g. Medium, DEV Community, Hacker Noon)
  - Reading in online communities (e.g. Twitter, Reddit, Hacker News, StackOverflow/StackExchange)
  - Participating in online communities (e.g. Twitter, Reddit, Hacker News, StackOverflow/StackExchange)
  - Watching online videos (e.g. YouTube, Twitch)
  - Taking courses (e.g. Coursera, Udemy, University Courses)
  - Working on side projects
  - Contributing to open source projects
  - Talking with developers I know/work with
  - Attending events (e.g. Meetups, Conferences, Workshops, Trainings)
- **Q20**: Would you consider yourself as a "security champion"? - A person who enables and promotes security practices within a team and is often the go-to person for security-related inquiries (Single Choice: Yes, I don't know, No)
- If you want to give us feedback on this survey, please use the text field below. Your feedback is very valuable to us and will help us in improving our future research activities (Short-answer text)
- As part of my PhD, I will conduct further research activities on the topic of attack-aware web applications. If you are keen to take part in them, please use the text field below to enter your E-mail address (Short-answer text)

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.

Appendix B.
Details related to the CTF challenge

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.2. Screenshots from the CTF challenge

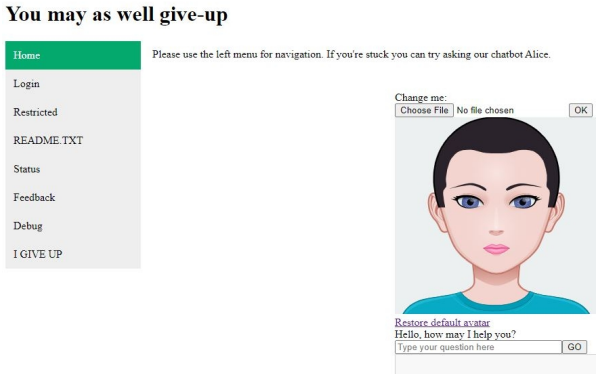


Figure 6. Home page of the "Give Up" CTF application.

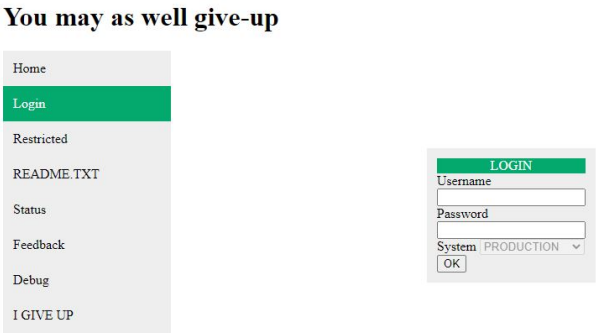


Figure 7. Login page of the "Give Up" CTF application.

B.3. Clustering of participants by attack types

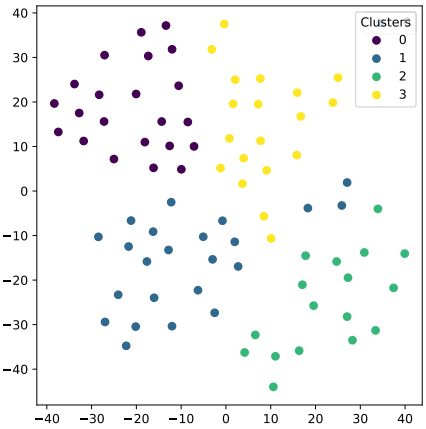


Figure 8. Clustering of participants by the attacks tried, using t-SNE method for dimensionality reduction.

B.4. Complete list of CTF attacks and implementation details

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.

TABLE 6. LIST OF ATTACKS, THEIR OWASP CATEGORY, AND HOW THEY ARE IMPLEMENTED IN THE CHALLENGE.