"I just looked for the solution!" On Integrating Security-Relevant Information in Non-Security API Documentation to Support Secure Coding Practices

Peter Leo Gorski, Sebastian Möller, Stephan Wiefling, and Luigi Lo Iacono

Abstract—Software developers build complex systems using plenty of third-party libraries. Documentation is key to understand and use the functionality provided via the libraries' APIs. Therefore, functionality is the main focus of contemporary API documentation, while cross-cutting concerns such as security are almost never considered at all, especially when the API itself does not provide security features. Documentations of JavaScript libraries for use in web applications, e.g., do not specify how to add or adapt a Content Security Policy (CSP) to mitigate content injection attacks like Cross-Site Scripting (XSS). This is unfortunate, as security-relevant API documentation might have an influence on secure coding practices and prevailing major vulnerabilities such as XSS.
For the first time, we study the effects of integrating security-relevant information in non-security API documentation. For this purpose, we took CSP as an exemplary study object and extended the official Google Maps JavaScript API documentation with security-relevant CSP information in three distinct manners. Then, we evaluated the usage of these variations in a between-group eye-tracking lab study involving N=49 participants. Our observations suggest: (1) Developers are focused on elements with code examples. They mostly skim the documentation while searching for a quick solution to their programming task. This finding gives further evidence to results of related studies. (2) The location where CSP-related code examples are placed in non-security API documentation significantly impacts the time it takes to find this security-relevant information. In particular, the study results showed that the proximity to functional-related code examples in documentation is a decisive factor. (3) Examples significantly help to produce secure CSP solutions. (4) Developers have additional information needs that our approach cannot meet.
Overall, our study contributes to a first understanding of the impact of security-relevant information in non-security API documentation on CSP implementation. Although further research is required, our findings emphasize that API producers should take responsibility for adequately documenting security aspects and thus supporting the sensibility and training of developers to implement secure systems. This responsibility also holds in seemingly non-security relevant contexts.

Index Terms—API Documentation, Content Security Policies, Secure Coding Practices, Developer Centered Security, Usable Security

1 Introduction

In 2020, approximately twenty-four million developers [1], [2] worked to meet the software needs of over four billion Internet users [3]. Web development is an attractive field for many developers because the Web, with its countless application areas, is very closely linked to our daily lives. However, many software products are produced without the know-how to consider data and application security during the development life cycle [4]. Due to the high degree of network connectivity, the sensitivity of user data, and legal requirements [5], many secure coding practices for APIs [6] and web applications [7] are mandatory in web engineering. Especially developers who are not familiar with security need support to protect the data of their users and to make their applications secure. This group also includes nonprofessional and prospective developers such as computer science students and junior developers who are at the beginning of their career.

Due to the complexity resulting from the data and application security mechanisms, challenges exist for many components of a software development environment. Usability flaws in the design of security APIs lead to improper usage and, in the end, to vulnerable software [8], [9], [10], [11], [12]. Static analysis tools or extensions for Integrated Development Environments (IDEs) to find and fix vulnerabilities can overwhelm developers [13], [14], or available tools are not used at all [15]. Documentation is sparse [16] or does not provide the information a developer needs [17]. Furthermore, online resources, despite their popularity, can spread unsafe code examples [18].

Since security is often a secondary task in software development, there are also efforts to equip development environments with security defaults [19], [20]. This support can relieve developers to a certain degree from implementing security components themselves, which are necessary to build a development project on a secure foundation. However, not every security requirement in web development can be covered by defaults without the developer actively

- *Peter Leo Gorski, Stephan Wiefling, and Luigi Lo Iacono are with the Department of Computer Science, Data and Application Security Group, H-BRS University of Applied Sciences, Sankt Augustin, Germany. E-mail: {peter.gorski,stephan.wiefling,luigi.lo_iacono}@h-brs.de*
- *Sebastian Möller is with the Quality and Usability Lab, TU Berlin, Germany. E-mail: sebastian.moeller@tu-berlin.de*

Manuscript received October 26, 2020; revised May 10, 2021.

Postprint version. Article accepted for IEEE Transactions On Software Engineering. DOI: 10.1109/TSE.2021.3094171

configuring the framework. A proven example of this are Content Security Policies (CSPs) [21].

CSPs [21] can be an effective measure to protect the integrity of website and web application elements against code injection attacks, such as Cross-Site Scripting (XSS) [22]. However, studies have shown that CSPs are rarely used in practice and are error-prone [23], [24], [25], [26]. Some of the flaws are caused by a combination of usability problems in web development frameworks, browser warnings, and API documentation [27]. Developers miss essential information and support when having little experience with security mechanisms such as CSPs [27]. Facing these challenges, framework developers may decide to avoid or remove a security default feature [28], to avoid putting their users in an uncomfortable development situation that can hinder the completion of a primary task. To prevent such decisions and situations, it should be investigated both the type of additional information web developers need to implement a secure application, and how tools and documentation can reliably provide this information.

For popular development frameworks and libraries, a diverse range of API documentation is usually available. Therefore developers can choose the information sources that best meet their information needs. When having a free choice, they do not only use secondary web sources such as blogs and developer centered question and answer platforms like Stack Overflow [18], [29]. Instead, they also use official information of software producers [17]. Recent studies conclude that content and form of official security API documentation should better address users' needs through easily accessible and practical recommendations for action and code examples [10], [11], [16], [17], [30], [31].

However, these requirements do not only concern the documentation of security APIs [32]. Since many aspects of web development include network connectivity, these implicate security requirements, which should be also addressed by non-security Web API documentation. As a non-security API we understand an API that is not classified as a security API [33]. Developers can take advantage of over 22,000 Web APIs [34] to add functionalities to their applications. Even if the vast majority of these libraries do not directly relate to security, like, e.g., a cryptographic API does, the integration of a Web API such as Google Maps can lead to security considerations that a developer must take into account.

Thus, web developers are required to meet this requirement and responsibility. In addition to the challenges of supporting API users with security-relevant information in a usable manner, as is the case with security APIs, other issues arise for non-security APIs. Security-relevant information and code examples do not relate to the primary feature set offered by an API. From an API producer's point of view, the question arises, where to place specific security-related information and code examples to support users in applying security measures. Furthermore, developers should perceive content and locations as meaningful and appropriate.

Recent studies found that neither professional developers nor students are sufficiently familiar with security concepts to be able to apply them securely. Still, we assume that this task is a particular challenge for inexperienced developers, such as prospective developers who are at the end of their studies and junior developers who have just started their careers. Usually, all developer groups share the same documentation offered by framework and API producers. For these reasons we selected a students sample (cf. Section 4) where most of the participants were near the end of their studies, and some already got paid for developing software (cf. Section 5).

Research Questions — Security-related information should also be part of the non-security API documentation, to assist developers in dealing with these considerations reliably and comprehensively. API producers provide essential and trustworthy first-hand information. In this way, each API provider could, e.g., supply its consumers with a tailored CSP for their services. We assume this approach to be useful for developers since API providers should know best about the required whitelist entries, i.e., the resources that their software library require and the entities that deliver these resources. To the best of our knowledge, the approach to integrate CSP information in non-security API documentation is not applied in practice to date.

Before we can address such a demand to the numerous API providers, it should first be understood whether this approach can help developers to implement a secure CSP in a web development framework. Research has not yet answered the question of how to usefully integrate security-relevant information into the documentation of a non-security Web API. Using the concrete application context of CSPs as a starting point, where extensive adoption problems exist in web development practice [23], [24], [25], [26], [35], we examine the following research questions:

RQ1: *Does the position of where documentation places security-relevant CSP information make a difference in transporting this information to a developer?* In a between-group lab study, we compare the original Google Maps JavaScript API documentation with three extended prototypes. By analyzing gaze data distributions, we study placement effects of security-relevant CSP information on the developers' performance.

RQ2: *How do developers read API documentation, and what elements in the documentation do they pay attention to?* We use eye-tracking to analyze the behavior of developers and to identify areas of visual attention by evaluating heat maps.

RQ3: *Does CSP documentation affect the functionality and security of participants' programming task solutions?* We analyze programming task results to evaluate whether our documentation approach supports developers with configuring CSPs and thus helps to integrate a functional and secure Google Maps API into a web page.

To answer these research questions, we present the following work:

- We design, implement, and test three extended documentation prototypes of the original Google Maps API documentation that add security-relevant CSP examples and information.
- We conduct a between-group eye-tracking lab study with 49 junior software developers.
- We evaluate in detail how junior developers work with API documentation and evaluate the effectiveness of our novel approach to integrate CSP examples in non-security API documentation.

- We highlight limitations and conclude with a recommendation for API vendors on how to include CSP examples in their API documentation.

Our results confirm previous work showing that usability problems exist in the application of CSP [27], [36]. While existing work illustrated that easily accessible and application-related code examples in security API documentation improve its usability and developer's code security [9], [10], [37], we found evidence that also security-relevant information in non-security API documentation is important to support secure coding practices. Additionally, our study extends knowledge offering fine-grained eye-tracking insights in the developer's behavior while using API documentation. Our participants mostly skimmed the documentation while searching for a quick solution to their programming task. We found that the location where CSP code examples are placed in non-security API documentation significantly impacts the time it takes to find this security-relevant information. In particular, the study results showed that the proximity to functional-related code examples is a decisive factor. In the context of CSPs, security-relevant code examples and information should be placed close to API examples. Alternatively, code comments can help point developers to security-relevant information located elsewhere in the documentation. We also found evidence that the examined novel approach of integrating CSP examples in non-security API documentation significantly helps to produce a secure CSP implementation. These results specifically address API producers to improve their API documentation in these aspects to support secure coding practices.

Outline — Our paper is structured as follows: We first introduce CSP in Section 2, discuss related work in Section 3, and describe our study methodology in Section 4. We present the results of our analysis and answer our research questions in Section 5. We outline limitations in Section 6 and discuss our findings in Section 7 before we conclude the paper in Section 8.

2 Content Security Policy (CSP)

CSP was initially proposed by Stamm et al. [38] in 2010. It became a W3C security standard [21] since 2012 and is supported by all modern web browsers. CSP's primary goal is to prevent, mitigate, and report code injection attacks such as XSS [7]. These attacks result from executing malicious content on web pages [39]. CSP restricts to include content for web pages by whitelisting origins for content that browsers are approved to load. This means that even if attackers were able to inject content, this content would not be executed or rendered if it did not match the whitelist. Since level 2, the feature set of CSP is no longer limited to code injection mitigation but has been extended to, e.g., interface redress vulnerabilities like Clickjacking [40].

CSP directives bind content types to lists of sources from which a CSP protected web page is allowed to fetch and include resources of that specific type. CSP specifies directives like `img-src` for images, `script-src` for scripts, and `style-src` for styles (cf. Table 1 for a selected list of specified directives). If a policy directive specifies, e.g., the source `https://some.cdn/` for the `script-src` directive, the protected web page fetches scripts only from this specified source. If a malicious code injection tries to fetch scripts from another site, e.g., `https://evil.site/`, the browser will refuse to load them.

TABLE 1
List of selected CSP directives web developers can use to declaratively restrict content types to be loaded from allowed sources only.

Directive	Description
`connect-src`	Defines allowed sources for connecting via XHR, WebSockets, and EventSource.
`default-src`	Is applied as a fallback to content types for which the according directive is missing.
`font-src`	Defines allowed sources that can serve web fonts.
`frame-src`	Defines allowed sources for nested browsing contexts using elements such as <frame> and <iframe>.
`img-src`	Defines allowed sources of images and favicons.
`media-src`	Defines allowed sources for loading media using the <audio>, <video> and <track> elements.
`object-src`	Defines allowed sources for the <object>, <embed>, and <applet> elements.
`script-src`	Defines allowed sources for scripts (JavaScript).
`style-src`	Defines allowed sources for style sheets (CSS).
`worker-src`	Defines allowed sources for Worker, SharedWorker, or ServiceWorker scripts.

CSP directives are wide open by default, i.e., if no specific policy is set, no restrictions apply when fetching resources for the corresponding content type. The `default-src` directive overrides this default behavior. When no explicit CSP directive is present for a given content type, the `default-src` directive comes into effect when being present. In general, this applies to any directive that ends with `-src`.

The source list in each policy directive can specify sources by the scheme (`blob:`, `data:`, or `https:`), ranging from hostname-only to a fully qualified URI. Wildcards are accepted, but only as a scheme, a port, or in the leftmost position of the hostname. As web pages require to load a bunch of external resources, the list of sources can enumerate individual sources separated with spaces. The source list also accepts four keywords that require to be specified single-quoted. The `'none'` keyword effectively disables security for assigned content types. The current origin, but not its subdomains, is matched by `'self'`. With `'unsafe-inline'`, inline JavaScript and CSS is allowed. Accordingly, `'unsafe-eval'` allows text-to-JavaScript mechanisms like the `eval()` function in JavaScript.

The CSP is a string containing the policy directives that a web browser should apply to a given web page, e.g.:

```
content-security-policy: default-src 'self'; script-src
    ↪ 'self' https://maps.googleapis.com/
```

The policy directives are separated with semicolons. Web pages can use as many CSP directives as they make sense for the given application and its context. One pitfall, however, is to make sure that all required sources of a given content type are listed in a single corresponding policy directive.

CSPs are specified by web developers using HTTP headers or meta elements in HTML pages. However, the use of HTTP headers dominates [26]. The HTTP response

message transports the policies from the webserver to the browser. The browser side then enforces them on a page-by-page basis. Thus, the server needs to send a CSP in every response that requires protection. The CSP standard specifies two HTTP response header types for this purpose. The `Content-Security-Policy` header transfers the policy to the browser so that the browser can enforce it. In contrast to that, the `Content-Security-Policy-Report-Only` header instructs the browser to report any violation of the CSP directive to a specified server endpoint. The latter allows web developers to test policy effects – without enforcing them – and to monitor any violation events, which may indicate resource changes on the service provider side or attack attempts. Figure 2 contains a complete example CSP, which is part of an HTTP response message carrying a web page to the browser.

3 Related Work

We first discuss previous CSP research results on their practical application and usability. We also summarize related work dealing with API documentation and eye-tracking in software engineering research.

3.1 Usability of CSP

In practice, the adoption of CSP is a problem. Large-scale studies on deployed policies revealed adoption problems, resulting in low usage rates and comprehensive misconfiguration [23], [24], [25], [26], [35]. In 2016, over 90% of deployed CSPs were not effective, mainly because users compiled insecure whitelists [25]. Furthermore, a lack of reporting, harsh policies, vulnerable policies, and frequent maintenance are identified as problems [26], [35], [36]. To support developers, Weichselbaum et al. developed a CSP evaluator [25], [41] that developers can use to check their policies for security. However, developers still have to create CSPs by a reverse engineering approach, even with semi-automated policy generation [23], [25], [42], and problems in CSP adoption remain [26]. We propose and evaluate a different approach, in which the API provider takes the responsibility of integrating a service tailored CSP into the API documentation.

Although authors of previous work called for research on CSP design and usability [27], [36], results of usability studies on the application of CSP are rare. One qualitative lab study with 30 participants identified usability problems that occurred when working with CSP [27]. The results showed that the given information and support by warnings in browser developer tools, the web development framework and its documentation, the API documentation, and third party resources from the Internet were not sufficient. Available documentation resources were not providing information that participants needed to create a functional and secure CSP. We address this problem by integrating security-relevant information into non-security API documentation. To this end, we adopted the technical design of this study, which consists of a web development framework with a default CSP setup, the Google Maps documentation, and the Chrome browser.

3.2 Security in API Documentation

Based on survey results, Uddin et al. presented ten types of API documentation problems, mostly related to content and presentation [32]. In our study, we precisely study these two aspects, i.e., how the presentation of security-related content in API documentation affects the security of a web page using the API. Inzunza et al. compiled a list of seven essential documentation elements representing minimum requirements for complete API documentation [43]: "API Overview", "Get Started Guide", "Sample Code", "Video Tutorials", "Document API Reference", "Document the API Directives", and "Document Status and Error Codes". The list does not suggest to document security-relevant information. In our study, we enhance a Google Maps API documentation page, representing a "Get Started Guide" and containing "Sample Code". As it is a part of an extensive documentation, it does not meet all the seven requirements.

Meng et al. concluded, after conducting a study including interviews and a survey, that developers mainly follow two different concepts when working with API documentation [44]. They either adopt a concepts-oriented or a code-oriented learning strategy. In the context of our study, the participants clearly showed code-oriented behavior. In an observational study with 11 developers, Meng et al. further studied how developers use API documentation [45]. Based on the results, they proposed high-level guidelines for designing API documentation with three main aspects: (1) Enable efficient access to relevant content, (2) facilitate initial entry into the API, and (3) support different development strategies. All guidelines are focused on the main functionality of an API and do not consider cross-cutting requirements like security. One important guideline when supporting different developer strategies is "Signal text-to-code connections". This guideline suggests that the connection between text and code should be easily identifiable by a suitable design with, e.g., using color-coding to facilitate the reading transition. To the best of our knowledge, we are the first to investigate the opposite approach, "Signal code-to-text connections". This guideline means that code examples should provide references to elements in the text, raising a developer's awareness for security-relevant information. Meng et al. also identified the need for "*research providing a more fine-grained analysis of the information units, such as text versus code examples, which developers attend to when using a certain section of the documentation*" [45]. We address this need and present, to the best of our knowledge, the first results of an eye-tracking study offering fine-grained results in how developers use Web API documentation.

Previous Usable Security studies recommended to provide software developers with easily accessible and application-related code examples in the documentation of security APIs [9], [10], [30], [37]. Mindermann et al. developed an open-source web platform that collects such examples for many different cryptographic APIs [46]. Moreover, the general relevance of solution-oriented code examples in software development proposed by other developers manifests in the popularity of online question and answer platforms [29] also for security professionals [47].

Based on this knowledge, this study examines whether and how security-relevant CSP information can be inte-

grated into online documentation of Web APIs, which do not have a primary focus on security, in a usable way. This study's primary goal is not to reconfirm that code examples improve the usability of APIs and their documentation. Instead, we want to study how we can increase the attention of developers to security-relevant information through appropriate API documentation extensions and to what extent this can contribute to the implementation of secure CSPs. Therefore, by placing security-relevant information near and even inside functional-related code examples, we also study if the developers' focus for these vital documentation components can be utilized for this purpose.

3.3 Eye-Tracking in Software Engineering Research

Since 2006, the use of eye-tracking in software engineering research has steadily increased as a method to study various topics, including program/model understanding, code review, and debugging [48]. Eye-tracker-recorded gaze data show both the target of the participant's attention, and the effort and time required to understand the stimulus. Eye-tracking enables an objective, quantitative measurement of the eye-tracking process in real time, without any intentional filtering by the study participants. Therefore, eye trackers help researchers to examine processes and intentions that participants cannot articulate [49]. Thus, they can complement and enhance data that was collected with semi-objective data collection methods (e.g., screen and audio recordings) and subjective data collection methods (e.g., surveys and questionnaires). In other words, eye movement measurement provides additional insight into participants' actions and reasons for them, based on where they focused their attention during a task.

Recent studies review and illustrate the use of eye-tracking in software engineering research and provide guidance [48], [50], [51], [52]. Working with code examples as part of API documentation shares aspects of typical eye-tracking application areas in software engineering research [52]. The developers need to comprehend a given program, review code examples, and own code implementations. Previous studies focused, e.g., on debugging and reading code [53], [54], [55], [56], and reading compiler error messages in the console [57]. Used metrics in eye-tracking studies in software engineering were often inconsistent [50], [51], making different studies difficult to compare. Furthermore, as an empirical method, eye-tracking is context-sensitive. Contextual aspects like the programming language and coding style can impact the visual effort when working with buggy code [53], [55]. Our study setup uses a Go framework and JavaScript code (cf. Section 4). Also, the gender and experience of participants impact the way of source code reading [54], [56]. We analyze our sample in Section 5.

Considering recent application of eye-tracking in software engineering research and the available guidelines, we first ensured that eye-tracking is an appropriate additional data source to answer our research questions. As mentioned in the beginning of this section, most available research used eye-tracking to study tasks related to programming and debugging as well as reading code and models. In our research, we focus on API documentation, which has been comparatively underrepresented in the literature to date (see Section 3.2). Although there are not many eye-tracking studies on API documentation, the methods are similar to studies that focus on reading code at various levels of abstraction – including prose pseudo-code. We oriented our methodology on previous eye-tracking studies related to reading documentation or code with a particular focus on identifying Areas of Interest (AOI) that attract more attention in the documentation. In addition to screen/audio recordings and post-task questionnaires, we chose eye-tracking as an objective measure to answer RQ1 and RQ2 in particular. Eye-tracking allowed us to quantify developers' behavior when working with API documentation. Thereby, we were able to study placement effects of security-relevant CSP information and measured the elements in the documentation that developers pay attention to.

As in previous eye-tracking studies, we used the number of fixations in the whole stimulus or in specific parts of it—the so-called *fixation count*—to find the AOIs [58], [59], [60]. A fixation is a spatially-stable eye-gaze during which the participant's visual attention is focused on a specific area of the stimulus [61]. To analyze fixation counts, we used heat maps as visualizations that map gaze data to the stimulus. See Section 4.5 and Section 4.10 for more details on the concrete setup and captured as well as analyzed metrics.

Our intention is to understand whether conventional API documentation can support the awareness and transfer of cross-cutting concerns, such as security. We also intend to understand whether the placement of such non-functional information in the documentation has an impact on developer adoption. This is particularly challenging because neither the API documentation nor the programming task itself includes security as a primary objective. Hence, the Average Fixation Duration (AFD), and related metrics used to measure and compare the amount of visual effort (or difficulty) to perform a task [58], [62], [63], [64], [65], [66], [67], [68], [69], [70], [71], [72], [73], [74] and to find the AOIs that are most important for the participants to perform their tasks [63], [64], [65], [74], can not be adopted. This is due to the fact that there is no direct task to include a CSP, but this is rather implicit.

4 Methodology

In a between-group lab study, we asked our participants to solve four web development tasks. We provided them with a web development framework for this purpose. Several security measures, including a strict CSP, went into effect by default in this framework, intending to support users to implement a secure application. The core of the study is the integration of a Google Map into a web page. In this step, the default security setting from the server side intervenes and prevents browser clients from loading external resources like scripts, styles, fonts, or images. This means that participants have to react to this default CSP security setting during the experiment because it prevents them from solving the given primary programming tasks. This behavior corresponds to real-life implementation attempts of secure defaults to help developers bring security features into their applications [28]. Also, this situation creates a

need for information and thus a motivation to work with the documentation.

Google Maps had been chosen as it is a widespread API and has a mature and well maintained documentation. In addition the API allowed us to keep tasks appropriatly short for a laboratory study with voluntary participants, but was complex enough for studying the research questions.

For the web page to display the map, participants have to configure a CSP whitelist for third party resources correctly (cf. Section 2). Previous research has shown that computer science students and junior developers with little professional experience have severe difficulties to cope with this security default because they needed information that was missing in the API documentation [27]. To support the participants with the secondary security task of implementing a secure CSP for their web page requirements, we enhanced an original Google Maps API documentation page. The enhancement contained required information, including an API specific CSP example. We integrated security-related information in three different ways (cf. Section 4.1) into the original Google Maps API documentation and examined the effects of these different versions.

The study took about two hours per participant. It consisted of a preliminary discussion, a briefing, the experiment, a structured interview, and a debriefing. The experiment was limited to a maximum time of one hour. The studies were all carried out by the first author.

4.1 Documentation and Study Conditions

We tested the original Google Maps API documentation and three modified versions of it. We describe the four API documentation variants, each representing a different study condition, in the following (cf. Table 2).

(G1) **CONTROL**: The original documentation page [75][1], i.e., the control condition, starts with a headline, a short introduction, and shows a Google map as a practical example of the API. The developers can implement this example by simply copying and pasting the corresponding code from the *Try it yourself* section. For quick access, the *Getting started* section provides hyperlinks that lead directly to three following sections, which explain step-by-step details of the code example. *Step 1* describes the structure of the HTML page, *Step 2* the JavaScript code with API calls, and *Step 3* the API key handling. The latter regulates access to the service and serves for billing purposes. At the very bottom of the page, readers will find the *Tips and troubleshooting* section, which outlines known issues when using the API. This page of the Google Maps API documentation does not contain any security advice other than instructions on how to get and use the mandatory Google API key.

Aiming to support developers in adopting secure programming practices of web engineering, we specifically changed the content of the original Google Maps API documentation and investigated the effects in this lab study.

1. At the end of 2019, Google changed its website design. However, the content of the page has not changed at the time of writing. The original website used in the study is accessible via Internet Archive [76].

We have taken care to select design options that are practically feasible. The primary purpose of the documentation is to support users in implementing a Google map. Therefore, being realistic, we excluded for the study design that the page can present security-relevant aspects prominently in a section at the topmost positions. This decision raises the practical problem of how to integrate security-related information in a way that users will find it when they need it. Thus, focusing on **RQ1**, we have chosen approaches that differ in the position and integration style of the security-relevant content.

TABLE 2
Overview of the documentation structure and locations where CSP information is added in each study condition. The documentation in the CONTROL condition does not contain any CSP information. A '∗' indicates structure elements appearing identical in all four conditions.

Study Condition	Documentation Structure
∗	**Adding a Google Map with a Marker to Your Website**
∗	Introduction
∗	Try it yourself
CODE-COMMENT	*Code Comment: Notes on Content Security Policies*
CODE-COMMENT	*+ Link to Tips and troubleshooting*
∗	Getting started
∗	*Link to Step 1*
∗	*Link to Step 2*
∗	*Link to Step 3*
STEP4	*Link to Step 4*
∗	Step 1: Create an HTML page
∗	Step 2: Add a map with a marker
CODE-COMMENT	*Code Comment: Notes on Content Security Policies*
CODE-COMMENT	*+ Link to Tips and troubleshooting*
∗	Step 3: Get an API key
STEP4	Step 4: Set a Content Security Policy (CSP)
STEP4	*CSP Header String (raw)*
STEP4	*CSP Header String (pretty print with comments)*
∗	Tips and troubleshooting
TIPS,CODE-COMMENT	*Content Security Policy (CSP)*
TIPS,CODE-COMMENT	*CSP Header String (raw)*
TIPS,CODE-COMMENT	*CSP Header String (pretty print with comments)*

We designed and tested three different approaches to display CSP information in the documentation (cf. Figure 1):

(G2) **TIPS**: We added a subsection in the *Tips and troubleshooting* section at the bottom of the documentation page.
The CSP section explains how a developer can identify problems and gives a recommendation for action. Since problems with the CSP configuration typically manifest themselves in the fact that implemented functionalities do not work, we tried to pick up the user with the wording "If the map does not show up..." (cf. Figure 2). The formulations also use the keywords “directive”, “hash”, and “inline”, which the Chrome browser developer tools use in their CSP warnings. We could have written detailed step-by-step instructions on how to define a CSP in our study web framework. However, since it is not practical for an API provider to do this for every available web framework, we chose a general formulation. The section provides a code example that developers can copy and paste directly into their development environment. It also provides an illustrative example that explains the individual CSP components. The end of the section contains links to a comprehensive CSP guide from Google [41] and to a tool that checks the security of a CSP [25].

(G3) **STEP4**: We added a fourth section entitled *Set a Content Security Policy (CSP)* to the original documentation

Fig. 1. Overview of the documentation and locations where CSP information is added in each study condition.

page and placed it before the section *Tips and troubleshooting*. The content of the section is identical to that of the TIPS condition. We also added a link to this section to the navigation sidebar and the "Getting started" section (cf. Figure 3).

(G4) **CODE-COMMENT**: Based on observations from the pilot studies, we assumed that the participants would likely use the Google Maps code example in the documentation to solve their primary programming task. Hence, we tested the effect of a source code comment (cf. Figure 4) in the code examples of the API documentation.

We assumed that this approach could help developers who would potentially copy sample code without reading the documentation. The comment briefly points out a potential problem and provides a link to the CSP section. Identical to the TIPS approach, we placed the CSP section at the end of the documentation page.

4.2 Preliminary Discussion and Ethics

In the preliminary discussion, the study participants first read and signed a consent form confirming that they were of legal age. There was no formal IRB (institutional review board) process at our university. Nevertheless, we followed the data handling requirements of the European General Data Protection Regulation (GDPR) [5]. The study required video and audio recording for exact time measures. The data has been stored on encrypted hard disks to ensure confi-

- If the map does not show up, one reason might be a CSP is set by your web server and enforced by your web browser. Check the **Developer Tools Console** in your web browser for error messages to see if elements are blocked.
- You **need all** of the following CSP directives to configure your web server for the Google Maps Demo code on this page and allow Google maps resources on your web site.
- You can open the Developer Tools and then reload your page. The Console tab will contain error messages with the **correct sha256 hash** for each of your refused inline elements. Alternatively, you can use a hashing tool.

CSP Header String (raw):

```
default-src 'self'; script-src 'self' https://maps.googleapis.com/ 'sha256-purvb6auV8U0MiZWQ6SKq7Tjza
```

CSP Header String (pretty print including comments):

```
default-src     // Secure default
                'self';

script-src      // JavaScript from your host
                'self'

                // Maps JavaScript API
                https://maps.googleapis.com/
                // Inline script hash for the Demo code on this page
                // If you change the script in the Demo code
                // this hash needs to be updated
                'sha256-purvb6auV8U0MiZWQ6SKq7TzalqXH7qN1KpVUeXr6E=';

style-src       // Styles from your host
                'self'
                // Fonts specification
                https://fonts.googleapis.com/css
                // Inline style hash for the Demo code on this page
                // If you change the style in the Demo code
                // this hash needs to be updated
                'sha256-rXxm3Q5rCvjaI3w7x0f/JJDqGECh8mxg68PKLP8Jo7pQ='
                // Styles
                'sha256-Uvs35gtL0c/whxmyVt4YoNv7YnPUd0tANZOlq3NshXE='
                // Control active
                'sha256-VjkqXV9i0mo5RzxvaQpz7qQA91PkjLVqLQGYNI4Cc/I='
                // UI hover effects
                'sha256-g9aHNH7iF2hhGZYtVVd5mKQSnyLPmXWw5gwiuxBVonI='
                // Styles
                'sha256-2WQZQFa8KGAig8CPptpS8JDqetQ2jb5arMlI6fTGWiU='
                // Media print
                'sha256-/VVOq+Ws/EiUxf2CU6tsqsHdOWqBgHSgw8PqCTjYD3U='
                // Styles
                'sha256-a2VR/Wq1VPr0+3GRY+lEmAQm7wjwwnDtPpcCPs2zTrw='
                // Styles
                'sha256-mmA4m52ZZWPKWAzDvKQbF7Qhx9VHCZ2pcEdC0f9Xn/Po='
                // Google Maps JavaScript API error
                'sha256-+1AnUMTxYqnxCnuWd5ie3PIFYaEJTJj6eAy7VkZ7jyc=';

img-src         // Images from your host
                'self'
                // UI images using data scheme
                data:
                // Satellite images
                https://maps.gstatic.com/mapfiles/
                https://maps.googleapis.com/maps/
                https://khms0.googleapis.com/kh
                https://khms1.googleapis.com/kh;


font-src        // Fonts
                https://fonts.gstatic.com/;


object-src      // Secure default
                'none';
```

- For an advanced guide to creating a CSP, read the Content Security Policy guide.
- Use the CSP Evaluator to check if a CSP serves as a strong mitigation against cross-site scripting attacks.

Fig. 2. CSP section with guidance and CSP example integrated into the original Google Maps JavaScript API documentation and tested in the conditions TIPS, STEP4, and CODE-COMMENT.

Getting started

There are three steps to creating a Google map with a marker on your web page:

1. Create an HTML page
2. Add a map with a marker
3. Get an API key
4. Set a Content Security Policy (CSP)

You need a web browser. Choose a well-known one like Google Chrome (recommended), Firefox, Safari or Internet Explorer, based on your platform.

Fig. 3. "Getting started" section with an additional link to the CSP section integrated into the original Google Maps JavaScript API documentation and tested in the STEP4 condition.

```
/**
 * Notes on Content Security Policies
 * If the map does not show up, check the Developer Tools Console in your web browser
 * for error messages to see if elements are blocked by a Content Security Policy
 * (e.g., default-src 'self';). Find tips and troubleshooting at:
 * https://developers.google.com/maps/documentation/javascript/adding-a-google-map#CSP
 */
```

Fig. 4. Source code comment integrated into the original Google Maps JavaScript API documentation and tested in the CODE-COMMENT condition.

dentiality. Participants can request deletion via a provided random token.

We framed [77] the tasks in order to explicitly emphasize to our participants that they should implement a secure solution. Therefore, the study moderator pointed out at the end of the preliminary discussion that "it should be a secure solution". By pointing this out, we tried to ensure that the participants see the application's security as a positive and desired aspect. We kept the term "secure" general and did not describe it by a specific threat model. We consider this to be a common requirement in web development, where a wide range of threats exists [7]. It is also questionable whether a threat model is generally available to developers when they take on the task of implementing a web application securely. In our view, API documentation has a special responsibility to provide such information. Finally, we did not want to influence the participants in the subject matter by mentioning the term CSP, as we did not want to influence the handling of the documentation by a specific security task. In order to still have all groups focusing on a protective CSP as security measure, in the study, by constraints of the web framework, our participants were required to consider CSP. The CSP by default setup prevented participants solving their programming tasks without considering CSP. However, they could decide to deactivate the security mechanism.

During the study, the participants tried to implement or search only for the CSP security mechanism. This observation confirms that the framing and the study design proved to be appropriately tailored to our research questions (cf. Section 5.3).

4.3 Development Environment Briefing

We briefed all study participants on the experiment's development environment to mitigate potential differences in their experience with the tools of the study environment. The 15 minute briefing included the Visual Studio Code integrated development environment (IDE), the folder and file structure of the Go [78] framework, required Go packages (app dependencies), and the Developer Tools of the Chrome browser. Participants were allowed to ask questions about the development environment. After the briefing, the study moderator asked them to read through the four tasks of the experiment. The participants were also able to ask questions for understanding the tasks. The study moderator asked the developers to think aloud and explained the meaning of the term. Then, the moderator calibrated the eye-tracking (9-point calibration), started the experiment, and left the room. The moderator followed the experiment via video, audio, and screencast in an observation room near the study room.

4.4 Programming Tasks

The experiment consisted of four programming tasks (cf. Figure 5). The first two served as warm-up tasks for the study participants to get used to the study situation. We selected short tasks that we expected our participants to complete quickly and that did not relate to the framework's default CSP setting. In **task one**, the participants integrated a web page favicon. In **task two**, our participants used Cascading Style Sheets (CSS) to integrate an image as a background for the page.

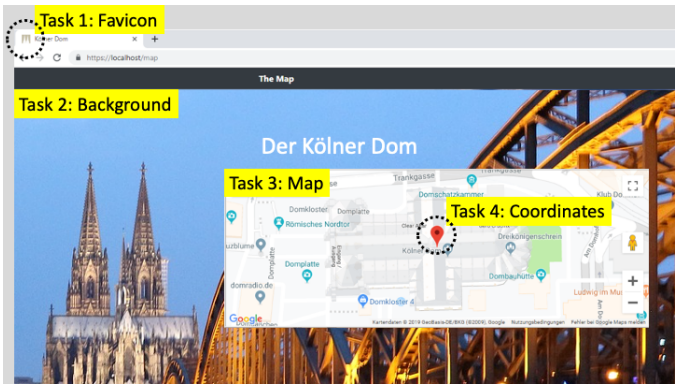

Fig. 5. Illustration of the four study tasks.

In **task three** – the main task – the participants had to integrate a map into the web page using the Google Maps JavaScript API. They needed to implement the tutorial example of the official Google Maps API documentation. The task description provided an API key for the service and a link to the web page. However, due to the default CSP, the browser did not execute the code example from the documentation. As a result, the browser displayed a warning inside the browser's JavaScript console. For the browser to display a map, the required secondary task – not mentioned in the task description – was to compile a CSP whitelist containing the required Google maps resources. Except in the CONTROL group, we provided the appropriate CSP for the tutorial example in the documentation. Thus, this condition represents the current real-world situation and allows us to measure effects of the documentation approaches.

The final **task four** was to set other coordinates for the map and a marker. To complete the task, the participant had to replace the CSP hash value because of the changed inline JavaScript code. This task should answer the question of whether the participants were sufficiently supported by the code example to be able to adapt it to their requirements, including a securely customized CSP.

We did not implement any real risk for the experiment, as it would have been somewhat artificial in the laboratory situation anyway. Hence, the study tasks missed a clear security and privacy context. Previous work found, however, that fewer study participants ignored security in laboratory studies when it was an explicit part of the task description [79]. We considered this in our study design and framing (see Section 4.2).

4.5 Implementation and Study Environment

The study took place in our usability lab. The developers worked with a Windows computer and two computer monitors. On the left monitor, they used the Visual Studio Code IDE for programming with the Go framework. We enabled code completion, and syntax highlighting and parsing for the code editor. We provided a code framework for the tasks to keep complexity low. Like most web development frameworks, the Go framework implemented the model view controller pattern [80]. We integrated security functionality with the Secure middleware [81] and kept the secure CSP default configuration `default-src 'self'`. On the right monitor, the participants used the Chrome browser with Internet access and Chrome's integrated developer tools. We implemented the changes of the official Google Maps developers documentation web page with three custom Chrome extensions, one for each of the three CSP conditions TIPS, STEP4, and CODE-COMMENT. The extensions monitored the requested URLs and replaced the original documentation with the corresponding CSP-extended variants, according to the condition. The participants accessed the documentation as they usually would. The software iMotions7 recorded eye-tracking data for the browser monitor with a Gazepoint GP3 HD sensor (150Hz sampling rate, 0.5 – 1.0 degree of visual angle accuracy). We equipped the workplace with a microphone and two cameras to record the experiments and streamed them live to the study moderator. The moderator annotated activities, statements, and key events during the sessions.

4.6 Structured Interview

After the experiment, the study moderator conducted a structured interview with each of the participants. The interview was structured into three parts. Part one contained questions regarding the experiment. We collected previous experiences in the field of software development in the second part. In the third part, we collected demographic data.

4.7 Debriefing

The study concluded with a twenty minute debriefing session. The study moderator explained the participants the programming problems they experienced in the study. The dialogue clarified the purpose of the study and raised awareness for the topic of Usable Security. The moderator asked every participant not to share the contents of the study with other persons.

4.8 Pilot Studies

We developed and improved the study design by piloting it twice. For both rounds, we invited two members of our working group, who were not involved in the study design. On the technical side, the pilot studies helped to adjust the hardware setup, and to fix minor bugs in the web development framework. On the usability side, we could clarify descriptions of the briefing and the task descriptions, and improve explanations and elements of the CSP section. Most importantly, the observations from the pilot studies led us to the idea of integrating security-relevant information into the API documentation in different ways, resulting in the four study conditions.

4.9 Recruiting and Compensation

We conducted the study at the end of a web programming course. Data and application security was not part of the curriculum. Participation in the study was an offer for a total of 103 participants in the course to reduce the workload of a semester project. In return, several required project features were waived. It was not a prerequisite for passing

the course. Also, participation did not replace a graded course component. Course attendance was required for admission to an exam. Since we had no means to offer other compensations like vouchers, we felt this was appropriate. We announced the study in the lecture as a study on web development with the Go programming language. Students interested in study participation could register online for an appointment. They chose one of the offered appointments themselves, and we assigned them round-robin to the four groups. During the study, we offered water and candy bars for their personal well-being.

4.10 Performance Measures

We used several metrics to evaluate the experiment. With eye-tracking, we measured the total time participants spent on the API documentation page and used the gaze data to determine the areas of the documentation that participants paid visual attention to. We measured the time participants worked on each programming task by analyzing the video recordings. The study moderator took notes on key moments while watching the study live stream. After the study, we used the session recordings to check and correct each timestamp precisely before using them for statistical analysis. To gain some supplementing insights on our qualitative laboratory study, we evaluated the structured interview answers.

5 Analysis

The analysis answers the research questions formulated in the introduction. First, we characterize the study sample based on the demographics and software development experience. Then, we present the results of the experiments. For our analysis, we set 0.05 as the threshold for statistical significance, i.e., $p < 0.05$ is significant.

The study took place over a period of four weeks from June to July 2019. 49 participants took part in the study (41 male, 8 female). Ten participants were in the CONTROL group and 13 participants each in the other three conditions. The participants were students and mainly reported to study in the sixth semester (SD: 2). They were, on mean average, 25 years old (SD: 4). The standard period of study is seven semesters. 38 recruited participants, with more than five study semesters, were near the end of their Bachelor's degree. Thus, our participants were typical candidates who start their professional life as junior software developers after their Bachelor's thesis. The fact that 17 of the students (35%) had already been paid for programming jobs before the study supports this assumption.

The participants reported having four years of programming experience on average (SD: 2). All of them started learning the Go programming language in the web programming course about half a year before the study and implemented at least one Go web application. However, 23 had also developed other web applications before. Thus, they had a mean JavaScript experience of two years (SD: 2). Even if the participants were assigned round-robin, the groups were balanced in terms of demographic data and previous development experience.

Results from the structured interview show that the developers of the study sample had little experience in applying data and application security measures. None of the participants named CSPs when being asked about the security measures they had ever considered in the development of software. When being asked about previous experience with CSPs, only three stated that they had. All three remembered problems with the CSP implementation. Participant G4P5 explained for example[2]:

> *"I briefly touched on it, but I mostly let it go because it was too complicated, and it didn't want to work."*

Eleven participants stated that they never considered any security measure.

In the following, we present the study results ordered by the research questions. We answer the corresponding research question after the results have been presented.

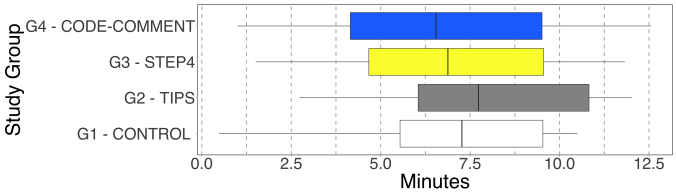

Fig. 6. RQ1: Total time each group spent on the documentation site, based on gaze data.

5.1 Information Placement Effects (RQ1)

During the programming tasks, participants spent a mean average of seven minutes on the API documentation page (SD: 3). There were no significant differences between the groups. (Kruskal-Wallis H(3) = .727, p = .867; cf. Figure 6).

5.1.1 Fixation Distribution

We analyzed the regions that our participants focused in the corresponding API documentation. The eye-tracking sensors measured 150 individual gaze points per second . If gaze points are close together in time and space, this is an indication for visual attention. The eye-tracking software clustered these gaze points to "fixations" for a defined area of interest.

Figure 7 shows the participants' fixation distribution on the documentation page per each of the four conditions (1) for all elements (all), (2) only for elements of the original documentation (original), and (3) only for the added CSP elements (CSP). Consistent to the time spent, there were no significant differences in the visual attention that the groups paid to the entire documentation page (Kruskal-Wallis H(3) = 2.538, p = .468). We neither found significant differences between the three groups in the distribution of fixations across the CSP documentation elements (Kruskal-Wallis H(2) = 2.040, p = .361).

However, we found a significant difference in the distributions of fixations on the elements of the original page (Kruskal-Wallis H(3) = 9.243, p = .026). Dunn-Bonferroni tests show that group CONTROL and STEP4 (z = 2.553, p = .011, r=.532), as well as CONTROL and CODE-COMMENT (z = 2.451, p = .014, r=.511), differ significantly both with

2. All quotes were translated from German into English.

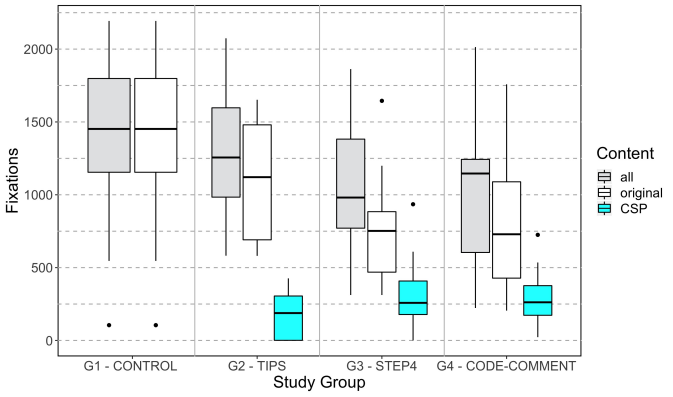

Fig. 7. RQ1: Fixation distributions of the conditions for the entire page (all), elements of the original documentation (original), and added CSP elements (CSP). Differences between CONTROL, and both STEP4 and CODE-COMMENT were significant for the original elements.

an effect size over 0.5. From these results, we conclude that the characteristics of the conditions led to a shift of visual attention to the CSP documentation elements. The designs of the groups STEP4 and CODE-COMMENT are suitable to transport the security-relevant information to the developer.

5.1.2 Time to Find CSP Documentation

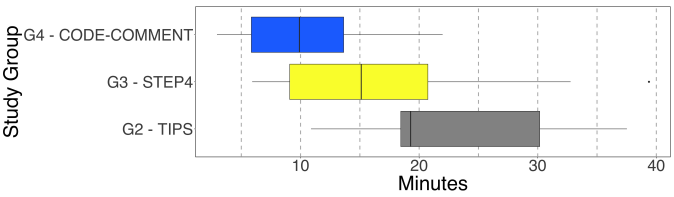

Fig. 8. RQ1: Time to CSP Documentation after starting task three. CODE-COMMENT participants were significantly faster than those of TIPS.

As presented before, the statistical results for the visual attention that the groups paid to the entire documentation page and the time spent on it did not show significant differences between the three CSP groups. However, there were significant differences in the time it took the participants after starting task three to find the CSP documentation (Kruskal-Wallis H(2) = 9.048, p = .011; cf. Figure 8). Dunn-Bonferroni post-hoc tests showed a significant difference between the TIPS and CODE-COMMENT group (z = 3.0, p = .003, r=.671), with an effect size greater than 0.6.

5.1.3 Effect of Code Comment and Link

Ten out of 13 participants of the CODE-COMMENT group copied the CSP code comment from the documentation and inserted it into their IDE. Three deleted it, two immediately after pasting and another one after a few minutes:

> *"I'll delete all the unnecessary comments. It's just annoying."* (G4P1)

We found the code comment in seven results. However, nine of 13 participants became aware of the documentation by the reference in the code comment:

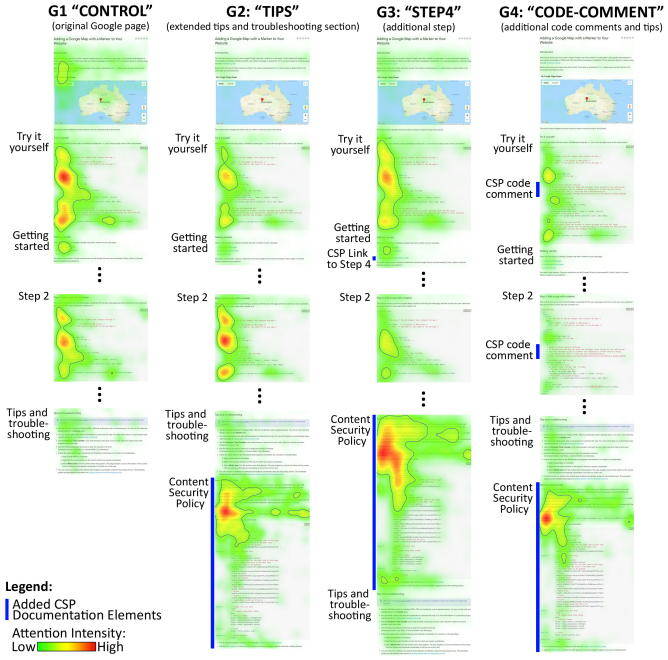

Fig. 9. RQ2: Heat map showing the visual attention intensity on the Google Maps API documentation, aggregated for all participants in each of the four study conditions.

> *"This is beautiful here, just as you wish. You couldn't have a simpler bug-fix."* (G4P7)

Two CODE-COMMENT participants found the documentation by scrolling down the site. Another two did not find the documentation at all, compared to four in the TIPS group. Only three developers in the STEP4 group found the CSP elements by clicking the "Getting started" link. In total, ten participants directly detected the note on the browser developer tools from the CSP documentation. Thus, they directly found the CSP warning messages in the browser console and were able to identify their problem.

Thus, we conclude for **RQ1**, that placing security-related CSP information has an impact on the time it takes to find the required information. Remarkably, security-related information and code examples prove to be difficult to detect if they are placed too far away from the API code examples. Participants gave their initial attention to the functional code example to solve programming tasks (cf. section 5.2). In the further course, the participants searched for more information using individual approaches. Starting from the previously used code example, they usually scrolled further down the document skimming the contents. In the process, participants frequently switch attention between the documentation and their code. A considerable amount of active searching is required to bridge longer distances between functional code examples and security-related information.

5.2 Areas of Attention (RQ2)

Based on the fixations, the heat maps show the distribution of visual attention inside the four tested API documentations (cf. Figure 9). The green, yellow, and red colors represent the visual attention intensity from low to high in ascending order. Blue bars, on the left side of each tested

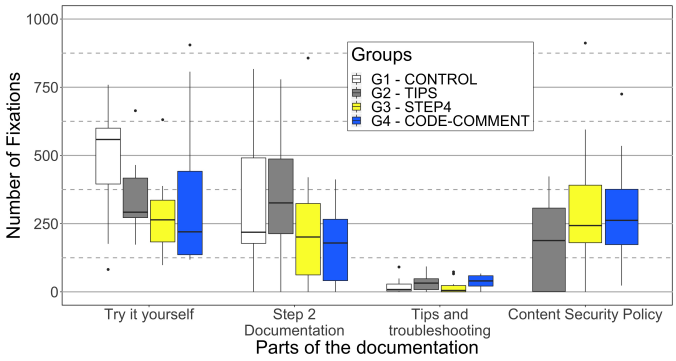

Fig. 10. RQ2: Box plot diagram of fixation distribution for parts of the documentation per each condition.

documentation, highlight our added CSP documentation elements. We answer **RQ2** using the heat maps by comparing the areas of visual attention of our experimental groups (cf. Figure 10).

In the main task of the experiment, the participants opened the Google Maps API documentation. After that, they started reading and explored the top of the documentation web page. However, they did not strictly read the page in the intended order from top to bottom. All groups skipped page areas due to their selective search behavior for the task solution, i.e., they scrolled and repeatedly visited the page.

The participants paid most visual attention to the code snippets in the sections "Try it yourself", "Step 2," and "CSP". All red areas in the heat map, which indicate the most visual attention, are related to code examples. The participants immediately found the Google Maps code example and started adapting it to their development task. This behavior corresponds to the primary task of implementing the map and the more complex secondary task of implementing a CSP.

In the STEP4 and CODE-COMMENT groups, the participant's visual attention shifted to the CSP areas of the modified documentation. The middle page areas clearly had fewer points of primary visual attention than the other areas. Participants of the CODE-COMMENT group concentrated their visual attention to only eight main areas compared with ten in the STEP4, 12 in the CONTROL, and 14 main areas in the TIPS group. The code comment itself is not part of these areas. The comment only needed short attention, as it simply referred to the "CSP" section. Participants of CONTROL and TIPS groups searched more for information solving their problem in original page elements than those participants of the other conditions (cf. Figure 7). Thus, they repeatedly focused on the Google Maps code examples in the sections "Try it yourself" and "Step 2". Group TIPS is in chapter "Step 2" more similar to the CONTROL group.

The green areas in Figure 9 show that the participants worked with all elements of the site but with limited visual attention. This impression is correct for the upper elements, including "Step 1". But it is only limited right for the areas below because some participants did not regard these. Nine participants of the three groups with CSP elements did not see or find the CSP section.

To answer **RQ2**, based on these results, we conclude that the developers of our study were focused on code examples. They mostly skimmed the documentation while searching for a quick solution to their programming task. One participant accurately described this behavior in the interview:

> *"There's a page where they explain all this [CSP]. But honestly, I didn't read it carefully. I just looked for the solution!"* (G4P6)

We observed that the participants were motivated to solve the tasks even if they searched for required information following the least effort approach. We consider this behavior legitimate in real programming contexts as well. The time limit of one hour was appropriate for most participants to deal with the documentation without time pressure at the beginning of task three (cf. section 5.3.1). Also, to counteract a sense of urgency, the study moderator told each participant before the experiment that they could do no wrong in the study.

5.3 Programming Task Results (RQ3)

We first analyze how much time the developers spent on each task within the overall time frame of one hour. Next, we evaluate the extent to which the tested approach could support the developers in achieving functional and secure results. Then we summarize CSP integration problems.

We rated the tasks as functional by the following criteria: In task one if the favicon was displayed, in task two when the background image was visible, and in tasks three and four if the Google Map appeared on the web page.

Focusing on **RQ3**, the rating of secure task solutions was limited to the CSP configurations. We evaluated the security of final CSP configurations applied for task three and four by criteria proposed and applied by Weichselbaum et al. [25]. The sample solution from the CSP documentation (cf. Figure 2) was also based on these characteristics and was therefore considered secure.

In this study we observed that the participants tried to implement or search only the CSP security mechanism suggesting that our task design and framing were effective. Moreover, we found the following insecure CSP configurations: (1) intentionally disabled, (2) unintentionally disabled because of syntax errors, and (3) 'unsafe-inline' was used without a nonce or a hash. All solutions could be unambiguously rated without discriminating potential secure approaches. Briefly, the existing solutions were either clearly non-functional, functional but insecure, or functional and based on the sample solution from the CSP documentation. Table 3 shows the detailed rating results for functionality and security.

5.3.1 Time Spent on Tasks

The programming tasks evaluation shows that most of the participants handled the warm-up tasks well. All developers solved task one in a mean average of six minutes (SD: 4), and only one developer each in the TIPS and CODE-COMMENT groups was unable to implement a functional solution for task two. Some participants in these two groups

TABLE 3
RQ3: Programming task results for each condition and in total.

	CONTROL	TIPS	STEP4	CODE-COMMENT	Total
Number of Participants	10	13	13	13	49
	NUMBER OF RATED TASK SOLUTIONS				
Task One: Functional	10	13	13	13	49
Task Two: Functional	10	12	13	12	47
Task Three:					
Functional	6	2	8	4	20
Functional and Secure	**0**	**2**	**5**	**3**	**10**
Task Four:					
Functional	6	2	3	3	14
Functional and Secure	**0**	**2**	**3**	**2**	**7**

spent a comparatively long time on the second task because they had difficulty debugging their entries in the CSS file. On mean average, developers worked for 11 minutes (SD: 9) on task two. Participants spent a mean average of 38 minutes (SD: 12) on the main task three. A total of 17 people started task four and worked on it for 8 minutes (SD: 5) until they either reached the total study time of one hour, or finished the tasks.

5.3.2 Functional Task Results

In the first attempt, the participants copied the Google Maps sample code (complete or partial) from the documentation and pasted it into their development environment. However, the low numbers of functional and secure solutions of tasks three and four reflect that the CSP default mechanism was a problem for the participants (cf. Table 3):

> *"I am confused, very confused, I just have to copy, and it has to appear there now. Why am I blocked?"* (G2P4)

20 and thus less than half of the study participants were able to implement a functional solution for the third task. Eight of them disabled the CSP mechanism in their framework. Participant G1P9 commented:

> *"I can't get it right! Why? Okay, let's do it another way. Let's just say 'screw security' for now. I'm gonna put a 'none' here, and then we'll move on - just for the record."*

After disabling CSP, changing the coordinates in the final task was easy for these developers. Nevertheless, we were able to lower the number of CSP deactivations by the framing in the preliminary discussion.

> *"Since it was just said that the site should be secure, I'll do that!"* (G3P5)

Overall, the CONTROL group achieved more functional but insecure solutions in task four than the other groups.

5.3.3 Functional and Secure Task Results

Fifteen of 49 participants only copied and pasted the CSP header string from the documentation and two only used parts of the illustrative CSP example. In contrast to that, eight participants copied and pasted both CSP versions during their attempts to integrate the CSP into the framework. A total of ten developers, who did not disable the CSP mechanism, were able to integrate a secure and working solution into the web page (cf. Table 3). However, they subsequently faced further problems of understanding when dealing with the customization task:

TABLE 4
RQ3: Contingency table for functional and secure solutions and used CSP documentation in task three and four. The solutions were significantly more functional and secure with CSP documentation.

		Functional and Secure Solution			
		Task 3		Task 4	
		true	false	true	false
Used CSP Documentation	true	10	20	7	23
	false	0	19	0	19

> *"But why is it a security problem if all I do is change the coordinates?"* (G3P7)

Seven of them understood the CSP concept to such an extent that they were able to achieve a functional and secure solution.

All four groups had worked with the documentation for about the same amount of time (cf. Section 5.1). Also, we did not see any major differences in the numbers of functional and secure solutions between the groups with CSP documentation. For these reasons, we compared the effect of CSP support on the security of task solutions across all groups (cf. Table 4). Fisher's exact test is significant for both task three (p=0.004) and task four (p=0.0338).

Thus, the security-relevant CSP information significantly improved the security of the solutions. 33% of the participants who worked with the CSP documentation were able to implement a CSP securely in task three, and 30% in task four. None of the CONTROL group participants, who did not have the support of our CSP documentation approach, achieved a secure solution.

However, these results also clearly show that the majority of the participants would have needed more support. Our approach could only help to bring security-relevant information to developers and thus provide them with a sample CSP. From the moment of integration into the framework, the experiment showed that there were still other factors where support was lacking, which we summarize below.

5.3.4 CSP Integration Problems

The integration into the Go framework, especially where the CSP code example had to be inserted, turned out to be a problem:

> *"I understand that I have to copy this string somewhere, I just don't know where."* (G2P12)

Searching for a solution on the Internet, participants mainly found meta-tag CSP examples, although the use of HTTP headers dominated in practice [26]. However, meta-tag CSP examples are suitable as self-contained code examples that can be easily integrated into HTML files and tested in a browser. When defining a secure CSP using a meta tag, which was the case for nine of 49 participants, the CSPs of both – the configuration file and the meta tag – get into effect. The study participants did not understand this standardized [21] behavior of the development environment:

> *"And how do I get it [the browser] to just take that? [...] It ignores the line completely, is that possible?"* (G3P8)

It was difficult for the participants to trace the origin of the policy, i.e., server-side or client-side. A hint, that a CSP was

set on both server and client side, was missing to resolve this double policy confusion. Answers from the Stack Overflow community were also not considered helpful:

> *"Ok, that's how it works, but I don't think that's how the security policy was meant to be. But I found it very hard to figure out how to configure something via Stack Overflow."* (G1P5)

Our results also confirm usability problems with CSP warning messages in the Chrome browser console, which had already been discovered in related work [27].

Concluding for **RQ3**, adding CSP into non-security API documentation significantly improved the security of the solutions. It also helped participants to configure CSP until the moment of integrating it into the framework. Our approach could not compensate for all problems that prevented participants from implementing a working and secure solution. Thus, service providers can help software developers to define CSPs by providing additional information in their documentation. Other aspects like the CSP integration into a specific web development framework or the usability of CSP warning messages in browsers lie outside their direct sphere of influence.

6 Limitations

The results of the study are subject to several limitations, which we describe in the following.

First, we recruited a student sample. Since having little programming experience, we assume that students are precisely the group that could benefit most of good documentation [82]. The participants took a course in web development, most were near the end of their studies, and some have already worked as software developers (cf. Section 5). Some of the participants had already gained experience in web development before the course. Our sample nevertheless has a narrow range of programming experience and occurring experience levels have been comparably distributed among the groups. We did not find a significant correlation between secure task solutions and programming experience.

However, the population of this study is not representative for all web developers. Previous studies could not find significant differences between student samples and professional developers [11], [83], [84]. They found that neither professional developers nor students are sufficiently familiar with security concepts like cryptography and password storage to be able to apply them securely. Nevertheless, in the context of our study, we assume that experienced developers would have had fewer problems with configuring a CSP in the framework and therefore would have been able to use the support of the documentation more effectively. We observed that the behavior of our participants solving programming problems varied widely (cf. Section 5.3).

The methodological strength of the presented laboratory study lies, particularly in the qualitative results. The small sample size of the groups limits the statistical power of the presented quantitative results by nature. However, the statistical results coherently substantiate and support the qualitative results.

We used a specific software development scenario for our lab study (cf. Section 4). The results are, therefore, only conditionally transferable to other software development contexts. API documentation, in general, also has many different forms. Therefore, this study uses the real and well-maintained documentation web page of the popular Google Maps JavaScript API as a baseline and starting point for studying the integration of security-relevant CSP information in non-security API documentation. Nevertheless, the structure and content are not representative for all API documentations. Furthermore, CSPs are only one specific security mechanism to reduce security risks in web development [7]. Thus, the effectiveness and generalizability of the tested approach can only be shown by follow-up studies. Future work should consider other documentation types, security measures, and software engineering contexts, like auto-generated API documentation (e.g., Javadoc), or mobile development documentation (e.g., Android).

We used a Go software development environment to run the experiments, which has some technical limitations. In this study, we refer to CSP level 2 features to mitigate XSS attacks, as browsers offer only incomplete support for level 3 features [85], [86], [87], [88]. Therefore, we did not consider CSP level 3 security features [21], [35] for the CSP example in our prototypes. However, the basic approach of providing a developer with API tailored CSPs does not change by considering additional CSP features, like clickjacking mitigation. Additionally, we used a local web server as an embedded part of the framework. Thus, we excluded another potential source of error. Under real circumstances, standalone web servers may also set HTTP headers. How to support developers in even more complex scenarios is left for future work.

The documentation was written in English even though most of the test persons were German native speakers. This situation is realistic since software development documentation is typically written in English, and translations are usually not available. For example, the Google Maps API documentation used in this study is only available in English. While The students' study curriculum requires a basic knowledge of the English language, a subject's language comprehension may impact the performance.

Finally, participants could not find the contents that we added into our prototypes by a search engine. However, we assume that depending on search queries, a search engine could have improved the discoverability of the CSP documentation.

7 Discussion

Based on our results, we discuss recommendations addressing practitioners such as API service providers who are in charge of API documentation creation and design. Further, we illustrate future work that researchers should investigate.

7.1 Recommendations

API providers have an unique responsibility for their users since they can provide high quality and trustworthy first-hand information. Thus, they can supply software developers directly with secure CSPs for their service. Instead of imposing a reverse engineering approach on their API users, i.e., they have to build a CSP following the try and error

principle, API providers could proactively provide CSPs tailored for their services.

Based on the results of this work, we recommend to follow our approach of adding CSP, including examples, into the API documentation. We suggest two ways to implement this approach: The security-relevant information should clearly be stated in the documentation's content structure (group STEP4). As an alternative, the developer should be made aware of necessary secondary tasks via a code comment (group CODE-COMMENT). If API providers specify a CSP for their services, we also assume a positive training effect for their users. Developers, whether young or experienced, would repeatedly deal with CSPs. This can increase the attention of developers for CSPs over a short time. At the same time, this can illustrate very clearly how secure CSPs have to be structured and implemented. Educators could use these application-oriented examples in their lessons. Search engines would also be able to display more suitable solutions to configuration problems. Secure examples could reduce visibility of the numerous existing tutorials on circumventing the CSP security mechanism. More documentation would allow best practices to evolve and may potentially gain secure CSP usage on more websites.

7.2 Future Work

We see several relevant work which should be addressed by researchers in future. First, a quantitative follow-up study should supplement the presented qualitative laboratory study to further evaluate if the identified styles to integrate CSP templates into non-security API documentation can effectively support developers in implementing security best practices. Furthermore, it is interesting future work to quantify how much cross-cutting concerns, such as security, are currently taken into account by non-security API documentation.

If the proposed approaches would become established in practice, hopefully leading to increasing secure CSP usage, future work should also find effective ways to support software developers in emerging issues like frequent merging CSPs from different providers. As web services frequently change over time, API users need to adjust their policies several times a month [35]. Also during this study, we had to modify the example CSP once. API producers could provide their users with up-to-date CSP examples as a valid reference for action. Here, we assume potential to extend the CSP standard. CSP examples are not limited to written documentation. They can also be made available by an API supporting automated processing. Researchers should investigate suitable interaction designs of semi-automated routines merging multiple CSPs and helping to compile policies. Further research could adopt such tools for teaching and evaluate whether these can help students understanding CSP configuration.

Facing automation, we see also challenges in maintainability. Following, e.g., the security principle "least common mechanism" [89] strictly by listing those specific endpoints that are explicitly needed instead of bundling full hosts would lead to whitelists with many entries. Another open challenge are CSP constellations in which the technical characteristics of individual providers impair the effectiveness of an entire CSP, e.g., by JSONP endpoints [25]. In such a case, we recommend to clearly state or even warn in the documentation or via tool dialogues, that if a user will implement the service it could break the customer's CSP security measure. In the long run, prominent placement in the API documentation may no longer be necessary. Empirical studies should evaluate if software developers would usually expect service providers to indicate the CSPs in a suitable and maybe standardized location.

Based on the interviews we also see potential to further investigate how documentation can better support developers to achieve a profound understanding during the implementation of CSPs. It can also be important to investigate the developers' confidence in their result after implementation. Besides CSP documentation, it is also important to examine and improve other tools involved (cf. Section 5.3). These tools can be, for instance, CSP warning messages in browser consoles and web development frameworks. Also, security defaults are difficult for developers to understand. Getting an idea of a web development framework's security concepts and procedures is not easy, either. It is challenging to inform developers in a quick and transparent way about reliable security support in the background and the mechanisms needed to adapt. On the one hand, these mechanisms should still work securely for their requirements. On the other hand, these should not get developers in the way. Moreover, it must be apparent which security services developers have to take care of themselves.

Beyond CSPs, which we examined as a concrete use case in this study, it is vital to explore the integration of security-relevant information into non-security API documentation, also concerning further security mechanisms. This information is necessary to ensure that developers will get sufficient support from API producers to apply API specific security characteristics. We assume this approach will help to increase attention for the existing variety of secure development practices in web development.

8 Conclusion

A basic contribution of the presented eye-tracking laboratory study is that it reconfirms the relevance of official API documentation [17] and carefully designed code examples [9], [10], [37] for secure software development. Our results show that developers' visual attention to API documentation strongly focuses on code examples and that in the specific case of CSP the integration of security-relevant information into non-security API documentation can support inexperienced software developers in applying secure coding practices successfully.

Here, we found evidence that the novel approach of integrating CSP examples significantly helps developers to configure this particular security mechanism in a secure way. The feedback from our participants also confirms the assumptions of previous studies about the lack of usability in practice. Current practices can overwhelm developers by missing to mediate a basic understanding of what a CSP mechanism is meant to do. Outside the scope of our experiment, we believe that the approach could partially contribute to improving security. However, we are eager to

design and conduct a field study to obtain more evidence on whether this assumption holds true or not.

Furthermore, our work substantiates the knowledge about developers' behavior when reading JavaScript API documentation and contribute an initial understanding of integrating security-relevant information into non-security API documentation. Our participants mostly skimmed the documentation while searching for a quick solution to their programming task. Thus, We found that the location where security-related CSP examples are placed in non-security API documentation significantly impacts the time it takes to find this information. In particular, the study results showed that the proximity to functional-related code examples in documentation is a decisive factor for secure CSP implementations. We conclude that security-relevant CSP examples and information should place close to API examples to be better recognized by developers. Code comments have proven to be very helpful in pointing developers to security-relevant information locating elsewhere in the documentation. More general conclusions on the effectiveness of security-relevant information in the context of other security mechanisms and documentation types require further research.

If we pursue the goal of supporting developers in the technical protection of software systems, we should urgently gain knowledge about how security-relevant information can be reliably transported to developers. Neuralgic locations like official API documentation should provide security awareness and usable support during software development. Therefore, we see particular opportunities for API providers using our findings to assist developers in securely integrating their provided APIs and, in the end, improving the data protection of end-users.

Acknowledgments

The authors would like to thank the anonymous reviewers for providing valuable feedback; and all participants of this study for their participation. This work has been partially funded by the German Federal Ministry of Education and Research within the funding program "Forschung an Fachhochschulen" (contract no. 13FH016IX6).

References

[1] Evans Data Corporation, "Worldwide Professional Developer Population of 24 Million Projected to Grow amid Shifting Geographical Concentrations," Website. URL: https://evansdata.com/press/viewRelease.php?pressID=278 (visited on 12/04/2021), EDC, Tech. Rep., 5 2019.

[2] A. Dayaratna, "IDC's Worldwide Developer Census, 2018: Part-Time Developers Lead the Expansion of the Global Developer Population," Website. URL: https://web.archive.org/web/20200117065523/https://www.idc.com/getdoc.jsp?containerId=US44363318 (visited on 12/04/2021), International Data Corporation, Tech. Rep., 10 2018.

[3] International Telecommunication Union (ITU), "Measuring digital development - facts and figures 2019," Document. URL: https://www.itu.int/en/ITU-D/Statistics/Documents/facts/FactsFigures2019.pdf (visited on 12/04/2021), 2019.

[4] H. Assal and S. Chiasson, "Security in the software development lifecycle," in *Fourteenth Symposium on Usable Privacy and Security*, ser. SOUPS. Baltimore, MD: USENIX Association, Aug. 2018, pp. 281–296.

[5] The European Parliament and the Council of the European Union, "Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation)," *Official Journal of the European Union* , L119/1. URL: http://data.europa.eu/eli/reg/2016/679/oj (visited on 12/04/2021), 2016.

[6] OWASP, "API Security Top 10 2019," Website. URL: https://owasp.org/www-project-api-security/ (visited on 12/04/2021), The Open Web Application Security Project, 2019.

[7] ——, "OWASP top 10 - 2017 - the ten most critical web application security risks," Website. URL: https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project (visited on 12/04/2021), The Open Web Application Security Project, 2017.

[8] 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, Sep. 2016.

[9] S. Nadi, S. Krüger, M. Mezini, and E. Bodden, "'Jumping Through Hoops': Why do Java Developers Struggle With Cryptography APIs?" in *38th International Conference on Software Engineering*, ser. ICSE. Austin, Texas: ACM, 2016, pp. 935–946.

[10] Y. Acar, M. Backes, S. Fahl, S. Garfinkel, D. Kim, M. L. Mazurek, and C. Stransky, "Comparing the usability of cryptographic APIs," in *2017 IEEE Symposium on Security and Privacy*, ser. SP. San Jose, CA, USA: IEEE, 2017, pp. 154–171.

[11] A. Naiakshina, A. Danilova, C. Tiefenau, M. Herzog, S. Dechand, and M. Smith, "Why do developers get password storage wrong?: A qualitative usability study," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS. Dallas, Texas, USA: ACM, 2017, pp. 311–328.

[12] 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*, ser. SOUPS. Baltimore, MD: USENIX Association, Aug. 2018, pp. 265–281.

[13] J. Smith, L. Nguyen Quang Do, and E. Murphy-Hill, "Why can't johnny fix vulnerabilities: A usability evaluation of static analysis tools for security," in *Sixteenth Symposium on Usable Privacy and Security*, ser. SOUPS. USENIX Association, Aug. 2020.

[14] M. Christakis and C. Bird, "What developers want and need from program analysis: An empirical study," in *31st IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE. New York, NY, USA: ACM, 2016, pp. 332–343.

[15] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Proceedings of the 35th International Conference on Software Engineering*, ser. ICSE. San Francisco, CA, USA: IEEE, 2013, pp. 672–681.

[16] Y. Acar, C. Stransky, D. Wermke, C. Weir, M. L. Mazurek, and S. Fahl, "Developers need support, too: A survey of security advice for software developers," in *2017 IEEE Cybersecurity Development*, ser. SecDev, Sep. 2017, pp. 22–26.

[17] 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*, ser. S&P. San Jose, CA, USA: IEEE, May 2016, pp. 289–305.

[18] W. Bai, O. Akgul, and M. L. Mazurek, "A qualitative investigation of insecure code propagation from online forums," in *2019 IEEE Cybersecurity Development*, ser. SecDev. Tysons Corner, VA, USA: IEEE, 2019, pp. 34–48.

[19] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking ssl development in an appified world," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '13. ACM, 2013, pp. 49–60.

[20] S. Krüger, S. Nadi, M. Reif, K. Ali, M. Mezini, E. Bodden, F. Göpfert, F. Günther, C. Weinert, D. Demmler, and R. Kamath, "Cognicrypt: Supporting developers in using cryptography," in *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE. Urbana-Champaign, IL, USA: IEEE, 2017, pp. 931–936.

[21] World Wide Web Consortium (W3C), "Content security policy level 3," W3C Working Draft. URL: https://www.w3.org/TR/CSP3/ (visited on 12/04/2021), 10 2018.

[22] OWASP, "Cross site scripting (xss)," Website. URL: https://owasp.org/www-community/attacks/xss/ (visited on 12/04/2021), 2020.

[23] M. Weissbacher, T. Lauinger, and W. Robertson, "Why is csp failing? trends and challenges in csp adoption," in *Research in Attacks, Intrusions and Defenses*, A. Stavrou, H. Bos, and G. Portokalidis, Eds. Cham: Springer International Publishing, 2014, pp. 212–233.

[24] K. Patil and F. Braun, "A measurement study of the content security policy on real-world applications," *International Journal of Network Security*, vol. 18, no. 2, pp. 383–392, Mar. 2016.

[25] L. Weichselbaum, M. Spagnuolo, S. Lekies, and A. Janc, "Csp is dead, long live csp! on the insecurity of whitelists and the future of content security policy," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 1376–1387.

[26] S. Calzavara, A. Rabitti, and M. Bugliesi, "Semantics-based analysis of content security policy deployment," *ACM Trans. Web*, vol. 12, no. 2, pp. 10:1–10:36, Jan. 2018.

[27] P. L. Gorski, L. Lo Iacono, and S. Wiefling, "Warn if secure or how to deal with security by default in software development?" in *The Twelfth International Symposium on Human Aspects of Information Security & Assurance*, ser. HAISA, 2018.

[28] Playframework, "Playframework - Content Security Policy Filter," Website. URL: https://github.com/playframework/playframework/pull/8242 (visited on 12/04/2021), 2018.

[29] Stack Overflow, "2020 developer survey," Website. URL: https://insights.stackoverflow.com/survey/2020 (visited on 12/04/2021), 2020.

[30] K. Mindermann, P. Keck, and S. Wagner, "How usable are rust cryptography apis?" in *2018 IEEE International Conference on Software Quality, Reliability and Security*, ser. QRS, Lisbon, Portugal, 2018, pp. 143–154.

[31] R. Balebako and L. Cranor, "Improving app privacy: Nudging app developers to protect user privacy," *IEEE Security Privacy*, vol. 12, no. 4, pp. 55–58, 2014.

[32] G. Uddin and M. P. Robillard, "How api documentation fails," *IEEE Software*, vol. 32, no. 4, pp. 68–75, July 2015.

[33] L. L. Iacono and P. L. Gorski, "I do and i understand. not yet true for security apis. so sad," in *Second European Workshop on Usable Security*, ser. EuroUSEC. Paris, France: Internet Society, 2017, pp. 1–11.

[34] ProgrammableWeb, "Apis show faster growth rate in 2019 than previous years," Website. URL: https://www.programmableweb.com/news/apis-show-faster-growth-rate-2019-previous-years/research/2019/07/17 (visited on 12/04/2021), 2019.

[35] S. Roth, T. Barron, S. Calzavara, N. Nikiforakis, and B. Stock, "Complex security policy? a longitudinal analysis of deployed content security policies," in *The Network and Distributed System Security Symposium*, ser. NDSS, 2020.

[36] S. Calzavara, A. Rabitti, and M. Bugliesi, "Content security problems?: Evaluating the effectiveness of content security policy in the wild," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS. New York, NY, USA: ACM, 2016, pp. 1365–1375.

[37] C. Wijayarathna and N. A. G. Arachchilage, "Why johnny can't store passwords securely? a usability evaluation of bouncycastle password hashing," in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, ser. EASE. ACM, 2018, p. 205–210.

[38] S. Stamm, B. Sterne, and G. Markham, "Reining in the web with content security policy," in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW. New York, NY, USA: ACM, 2010, pp. 921–930.

[39] C. Cimpanu, "Hackers are collecting payment details, user passwords from thousands of sites - Servers of at least seven companies compromised to deliver malicious code to thousands of sites." Website. URL: https://www.zdnet.com/article/hackers-are-collecting-payment-details-user-passwords-from-4600-sites/ (visited on 12/04/2021), ZDNet, 2019.

[40] OWASP, "Clickjacking," Website. URL: https://owasp.org/www-community/attacks/Clickjacking (visited on 12/04/2021), The Open Web Application Security Project, 2020.

[41] M. West and J. Medley, "Content Security Policy," Website. URL: https://developers.google.com/web/fundamentals/security/csp/ (visited on 12/04/2021), 2019.

[42] K. Patil, T. Vyas, F. Braun, M. Goodwin, and Z. Liang, "Poster: Usercsp-user specified content security policies," in *In the proceedings of the Symposium On Usable Privacy and Security*, ser. SOUPS, 2013.

[43] S. Inzunza, R. Juárez-Ramírez, and S. Jiménez, "Api documentation," in *Trends and Advances in Information Systems and Technologies*, Á. Rocha, H. Adeli, L. P. Reis, and S. Costanzo, Eds. Cham: Springer International Publishing, 2018, pp. 229–239.

[44] M. Meng, S. Steinhardt, and A. Schubert, "Application programming interface documentation: What do software developers want?" *Journal of Technical Writing and Communication*, vol. 48, no. 3, pp. 295–330, 2018.

[45] ——, "How Developers Use API Documentation: An Observation Study," *Commun. Des. Q. Rev*, vol. 7, no. 2, p. 40–49, Aug. 2019.

[46] K. Mindermann and S. Wagner, "Usability and Security Effects of Code Examples on Crypto APIs," in *2018 16th Annual Conference on Privacy, Security and Trust*, ser. PST, Aug 2018, pp. 1–2.

[47] Stack Exchange, "Welcome to information security stack exchange," 2021. [Online]. Available: https://security.stackexchange.com/tour

[48] Z. Sharafi, Z. Soh, and Y.-G. Guéhéneuc, "A systematic literature review on the usage of eye-tracking in software engineering," *Information and Software Technology*, vol. 67, pp. 79–107, 2015.

[49] J. Ross, "Eyetracking: Is it worth it?" UXmatters Online Article URL: https://www.uxmatters.com/mt/archives/2009/10/eyetracking-is-it-worth-it.php/ (visited on 12/04/2021), 10 2009.

[50] Z. Sharafi, T. Shaffer, B. Sharif, and Y. Guéhéneuc, "Eye-tracking metrics in software engineering," in *2015 Asia-Pacific Software Engineering Conference*, ser. APSEC, 2015, pp. 96–103.

[51] F. Hauser, J. Mottok, and H. Gruber, "Eye tracking metrics in software engineering," in *Proceedings of the 3rd European Conference of Software Engineering Education*, ser. ECSEE. New York, NY, USA: Association for Computing Machinery, 2018, p. 39–44.

[52] Z. Sharafi, B. Sharif, Y.-G. Guéhéneuc, A. Begel, R. Bednarik, and M. Crosby, "A practical guide on conducting eye tracking studies in software engineering," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3128–3174, Sep. 2020.

[53] R. Turner, M. Falcone, B. Sharif, and A. Lazar, "An eye-tracking study assessing the comprehension of c++ and python source code," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 231–234.

[54] B. Sharif, M. Falcone, and J. I. Maletic, "An eye-tracking study on the role of scan time in finding source code defects," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 381–384.

[55] B. Sharif and J. I. Maletic, "An eye tracking study on camelcase and under_score identifier styles," in *2010 IEEE 18th International Conference on Program Comprehension*, 2010, pp. 196–205.

[56] Z. Sharafi, Z. Soh, Y.-G. Guéhéneuc, and G. Antoniol, "Women and men — different but equal: On the impact of identifier style on source code reading," in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, 2012, pp. 27–36.

[57] T. Barik, J. Smith, K. Lubick, E. Holmes, J. Feng, E. R. Murphy-Hill, and C. Parnin, "Do developers read compiler error messages," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, 2017, pp. 575–585.

[58] M. Crosby and J. Stelovsky, "How do we read algorithms? a case study," *Computer*, vol. 23, no. 1, pp. 25–35, 1990.

[59] M. E. Crosby, J. Scholtz, and S. Wiedenbeck, "The roles beacons play in comprehension for novice and expert programmers," in *Programmers, 14th Workshop of the Psychology of Programming Interest Group, Brunel University*, 2002, pp. 18–21.

[60] H. Uwano, M. Nakamura, A. Monden, and K.-i. Matsumoto, "Analyzing individual performance of source code review using reviewers' eye movement," in *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications*, ser. ETRA '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 133–140.

[61] M. A. Just and P. A. Carpenter, "A theory of reading: from eye fixations to comprehension," *Psychological Review*, vol. 87, no. 4, 1980.

[62] R. Bednarik and M. Tukiainen, "Effects of display blurring on the behavior of novices and experts during program debugging," in *CHI '05 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '05. New York, NY, USA: Association for Computing Machinery, 2005, p. 1204–1207.

[63] ——, "An eye-tracking methodology for characterizing program comprehension processes," in *Proceedings of the 2006 Symposium on*

Eye Tracking Research & Applications, ser. ETRA '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 125–132.

[64] S. Jeanmart, Y.-G. Gueheneuc, H. Sahraoui, and N. Habra, "Impact of the visitor pattern on program comprehension and maintenance," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '09. USA: IEEE Computer Society, 2009, p. 69–78.

[65] G. Cepeda Porras and Y.-G. Guéhéneuc, "An empirical study on the efficiency of different design pattern representations in UML class diagrams," *Empirical Software Engineering*, vol. 15, no. 5, pp. 493–522, Oct. 2010.

[66] T. Busjahn, C. Schulte, and A. Busjahn, "Analysis of code reading to gain more insight in program comprehension," in *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, ser. Koli Calling '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 1–9.

[67] R. Bednarik, "Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations," *International Journal of Human-Computer Studies*, vol. 70, no. 2, pp. 143–155, 2012.

[68] Z. Soh, Z. Sharafi, B. V. den Plas, G. Porras, Y. Gueheneuc, and G. Antoniol, "Professional status and expertise for uml class diagram comprehension: An empirical study," in *International Conference on Program Comprehension*. Los Alamitos, CA, USA: IEEE Computer Society, jun 2012, pp. 163–172.

[69] Z. Sharafi, Z. Soh, Y.-G. Guéhéneuc, and G. Antoniol, "Women and men — different but equal: On the impact of identifier style on source code reading," in *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, 2012, pp. 27–36.

[70] Z. Sharafi, A. Marchetto, A. Susi, G. Antoniol, and Y.-G. Guéhéneuc, "An empirical study on the efficiency of graphical vs. textual representations in requirements comprehension," in *2013 21st International Conference on Program Comprehension (ICPC)*, 2013, pp. 33–42.

[71] R. Petrusel and J. Mendling, "Eye-tracking the factors of process model comprehension tasks," in *Proceedings of the 25th International Conference on Advanced Information Systems Engineering*, ser. CAiSE'13. Berlin, Heidelberg: Springer-Verlag, 2013, p. 224–239.

[72] D. Binkley, M. Davis, D. Lawrie, J. I. Maletic, C. Morrell, and B. Sharif, "The impact of identifier style on effort and comprehension," *Empirical Softw. Engg.*, vol. 18, no. 2, p. 219–276, Apr. 2013.

[73] N. E. Cagiltay, G. Tokdemir, O. Kilic, and D. Topalli, "Performing and analyzing non-formal inspections of entity relationship diagram (erd)," *Journal of Systems and Software*, vol. 86, no. 8, pp. 2184–2195, 2013.

[74] B. De Smet, L. Lempereur, Z. Sharafi, Y.-G. Guéhéneuc, G. Antoniol, and N. Habra, "Taupe: Visualizing and analyzing eye-tracking data," *Science of Computer Programming*, vol. 79, pp. 260–278, 2014, experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010).

[75] Google, "Adding a google map with a marker to your website," Website. URL: https://developers.google.com/maps/documentation/javascript/adding-a-google-map (visited on 12/04/2021), 2019.

[76] ——, "Adding a google map with a marker to your website," Website. URL: https://web.archive.org/web/20190715140525/https://developers.google.com/maps/documentation/javascript/adding-a-google-map (visited on 12/04/2021), 2019.

[77] A. Tversky and D. Kahneman, "The framing of decisions and the psychology of choice," *Science*, vol. 211, no. 4481, pp. 453–458, 1981.

[78] Go, "The go programming language," Website. URL: https://golang.org/ (visited on 12/04/2021), 2020.

[79] A. Naiakshina, A. Danilova, C. Tiefenau, and M. Smith, "Deception task design in developer password studies: Exploring a student sample," in *Fourteenth Symposium on Usable Privacy and Security*, ser. SOUPS. Baltimore, MD: USENIX Association, Aug. 2018, pp. 297–313.

[80] G. E. Krasner and S. T. Pope, "A cookbook for using the model-view controller user interface paradigm in smalltalk-80," *J. Object Oriented Program.*, vol. 1, no. 3, p. 26–49, Aug. 1988.

[81] C. Jacobsen, "Secure - http middleware for go that facilitates some quick security wins," Website. URL: https://github.com/unrolled/secure (visited on 12/04/2021), 2020.

[82] M. Tahaei, A. Jenkins, K. Vaniea, and M. K. Wolters, "'i don't know too much about it': On the security mindsets of computer science students," in *9th International Workshop on Socio-Technical Aspects in Security and Trust*, ser. STAST. Luxembourg City, Luxembourg: Springer International Publishing, 2019.

[83] Y. Acar, C. Stransky, D. Wermke, M. L. Mazurek, and S. Fahl, "Security developer studies with github users: Exploring a convenience sample," in *Thirteenth Symposium on Usable Privacy and Security*, ser. SOUPS. Santa Clara, CA, USA: USENIX Association, 2017, pp. 81–95.

[84] A. Naiakshina, A. Danilova, E. Gerlitz, E. von Zezschwitz, and M. Smith, ""if you want, i can store the encrypted password": A password-storage field study with freelance developers," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI. ACM, 2019.

[85] MDN Web Docs, "Content-Security-Policy - Browser compatibility," Website. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy#Browser_compatibility (visited on 12/04/2021), Mozilla, 2019.

[86] Google, "Chrome Platform Status," Website. URL: https://www.chromestatus.com/features#csp (visited on 12/04/2021), 2019.

[87] A. Deveria, "Can I use content security policy?" Website. URL: https://caniuse.com/#search=content%20security%20policy (visited on 12/04/2021), Can I use, 2019.

[88] Apple, "Documentation Archive - What's New in Safari - Safari 10.0," Website. URL: https://developer.apple.com/library/archive/releasenotes/General/WhatsNewInSafari/Articles/Safari_10_0.html#//apple_ref/doc/uid/TP40014305-CH11-SW1 (visited on 12/04/2021), 2018.

[89] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *Proceedings of the IEEE*, vol. 63, no. 9, pp. 1278–1308, Sept 1975.

Peter Leo Gorski is a Ph.D. candidate at TU Berlin and senior researcher of the Data and Application Security Group (DAS) at H-BRS University of Applied Sciences in Sankt Augustin, Germany. He has been researching the Usability of Security APIs since 2015. In his work, he focuses on information flows between API producers and API consumers.

Sebastian Möller received the Electrical Engineering degree from the University of Bochum, Germany, University of Orléans, France, and University of Bologna, Italy, and the Doctor-of-Engineering degree in 1999, with a book on the Assessment and Prediction of Speech Quality in Telecommunications, and the Venia Legendi with a book on the Quality of Telephone-Based Spoken Dialogue Systems in 2004. In 2005, he joined Telekom Innovation Laboratories and in 2007, he was appointed as a Full Professor of quality and usability with TU Berlin. In addition, he is a Scientific Director with the German Research Center for Artificial Intelligence (DFKI), Berlin, and an Adjunct Professor with the University of Technology Sydney. Since 1997, he has taken part in ITU–T Study Group 12, where he is currently a Co-Rapporteur of Question Q.15/12. His primary interests are in speech and multimedia quality, speech and language technology, user experience, usable security and privacy, as well as crowdsourcing.

Stephan Wiefling is a research associate of the Data and Application Security Group (DAS) at H-BRS University of Applied Sciences in Sankt Augustin, Germany. He is also a PhD student at the Horst Görtz Institute for IT Security (HGI) of Ruhr University Bochum in Bochum, Germany. His current research spans several areas in the field of usable security, including topics of developer-centered security and risk-based authentication.

Luigi Lo Iacono leads the Data and Application Security Group (DAS) at H-BRS University of Applied Sciences in Sankt Augustin, Germany. His research interests include security- and privacy-enhancing technologies with a specific focus on developer-centered security and privacy cockpits.