"I have no idea how to make it safer": Studying Security and Privacy Mindsets of Browser Extension Developers

Shubham Agarwal, Rafael Mrowczynski, Maria Hellenthal, Ben Stock

CISPA Helmholtz Center for Information Security, Saarbruecken, Germany
{shubham.agarwal, mrowczynski, hellenthal, stock}@cispa.de

Abstract

Browser extensions play a vital role in the Web ecosystem: they enable users to customize their experience while browsing. However, the higher privileges of extensions compared to the Web applications require in-depth security considerations to not threaten the security and privacy (S&P) of their users; the security and privacy mindset of developers has not been studied yet, though. In this paper, we close this research gap.

To that end, we conducted a qualitative study with extension developers from diverse backgrounds and experience levels (N=21) to identify the root causes for vulnerable extensions existing in the ecosystem. Our findings suggest that developers often implicitly acknowledge the S&P risks associated with their extensions, but they frequently lack the necessary knowledge and resources to implement effective security and privacy-protecting mechanisms. Additionally, socio-technical barriers, such as insufficient incentives and external pressures, including platform-imposed restrictions, further hinder secure development practices. Based on our findings, we offer empirically grounded takeaways for the browser extension ecosystem to help strengthen security practices and ultimately provide better protection for users.

1 Introduction

Web browsers have made significant advances to allow users to customize their experience. Browser extensions, or add-ons, are a prime example of this customization. Much like their historical predecessors, the *ActiveX plugins* and the *bookmarklets* [15, 56], these extensions, quite literally, extend the native capabilities of browsers and offer a diverse range of functionalities to users. Today, they are integral to the average user's browsing experience, with everyday utility (such as password management or ad blocking) and widespread adoption across platforms [97]. Thus, browser extensions today have a large user base across different Web platforms. However, their popularity has also attracted digital miscreants, exploiting extensions for nefarious intentions.

The research community, penetration testers, and security enthusiasts have extensively investigated browser extensions over the last decade. They reported numerous critical S&P issues with browser extension [33, 72, 79]. Much of the early investigations gravitated around the overtly malicious intent of extension developers [5, 43, 44, 73] which was followed by strict measures and architectural changes by browsers to enhance the overall security of the ecosystem [18, 64]. However, recent findings have also shown that even harmless extensions could be abused by malicious actors on the Web through various overt or covert channels. The attacks on extensions could range from exploiting code vulnerabilities to perform Universal Cross-Site Scripting (UXSS), to data theft, to tracking users online, and so on [17, 53, 84, 85]. Unfortunately, many of these exploitable vulnerabilities in the extension could be an inadvertent coding mistake or even desired behavior of the extension; however, they are exposed to malicious actors due to the inherent architectural limitations [4]. This is further complicated by the fact that browser vendors may differ in terms of extension support and API (in-)compatibilities [24, 66]. Last but not least, the extension also needs to be feature-rich to meet users' expectations.

To date, while research has focused on detecting *benign-but-buggy* extensions and developing understanding in other contexts, the community misses an analysis of browser extension developers' S&P mindsets [95, 100]. In this study, we close this gap in the literature. Concretely, we perform a semi-structured interviews with extension developers (N=21) to assess the S&P awareness through two coding tasks and additional interview questions on security mindsets and past experiences with the extension ecosystem.

Our findings highlight that developers often implicitly consider the S&P aspects of their extensions' behavior; however, they lack sufficient knowledge to implement corresponding mechanisms. Additionally, socio-technical drivers, such as monetary incentives (or lack thereof), and ecosystemic and extrinsic influences, such as platforms' restrictions and business goals, may also influence the development style and pattern.

To summarize, we answer the following research questions:

RQ1 *What is the security and privacy mindset of extension developers?*

RQ2 *What do extension developers regard as threats, and how do they model these threats in their own extension?*

RQ3 *What are the critical factors and roadblocks that influence extension development practices?*

2 Technical Background

We first outline the relevant background topics on the extension ecosystem, their components, and the publishing process.

Extension Architecture & Distribution: Browser extensions are client-side add-ons that users install to get additional features while surfing the Web. Currently, there are two popular extension marketplaces, the *Chrome Web Store (CWS)* for all chromium-based browsers and the *Mozilla Add-ons Store (AMO)* for the Firefox family of browsers. Their users can access and install extensions [36, 66] from there. While *Microsoft Edge* and *Opera* also have their own extensions marketplace, their users can nevertheless install extensions from the CWS, along with *Brave* and *Arc*, since they are all Chromium-based browsers. Thus, the CWS currently holds the largest market share in the extension ecosystem. Although there are minor differences between how the two extension stores operate, the underlying architecture and development process are very similar since they adhere to the common *WebExtensions* API standards [97].

Extension Components: Browser extensions, in essence, contain three main components: the *extension core*, the *content scripts*, and the interactive *UI pop-ups* [25]. The extension core is an isolated and highly privileged component that runs in the background (as service workers). It can access the privileged *chrome* APIs and includes page-agnostic functionalities. Content script(s) are less privileged and can interact with the DOM of the visited Web page, enabling page-related features of the extension. The extension may also inject a script directly into the context and the JavaScript namespace of the visited page from the extension core using the privileged *scripting* APIs. The UI pop-ups enable users to interact and configure the behaviors of their extensions defined by the developers. The extension package also has a *manifest* file that contains the metadata of the extension itself: all the API and host permissions that the developer requests, the list of scripts, stylesheets, and other resources that it includes, and the description of the extension shown to the user at install-time [19]. Extension components can communicate with each other or with other extensions through dedicated channels, such as exchanging *postMessages*.

Browser vendors take proactive measures to mitigate against existing threats in the extension ecosystem. Google launched the latest extension standards in 2021, called *MV3*, to boost the security and performance of the extensions overall [24]. However, the MV3-driven changes and the transition process have stirred controversies among developers as they also restrict the desired extensions' capabilities (e.g., deprecation of the `webRequestBlocking` API used by ad blockers).

Publishing process: Given the highly privileged nature of browser extensions and the range of sensitive operations they can perform, a regulatory system is essential to restrict malicious and vulnerable extensions in the ecosystem. Hence, every extension published on the store goes through a review process [20, 65]: When an extension developer decides to publish their extension on any of the stores, they first need to submit their extension along with additional information about the technical and non-technical aspects of their extensions to the platform for review. The browser vendors may ask the extension developer to make further changes to comply with their review standards [21, 68]. The review process is a black box, and the browser vendors have not documented the underlying technical details. We assume this to be intentional to avoid any bypass by malicious extension developers [22, 63]. This way, every extension published on the store goes through a detailed scrutiny when submitted for the first time and may undergo additional periodic vetting while they are available on the store.

Client-side Storage: Web applications have access to different client-side storage APIs, such as the *Web Storage* APIs and the `IndexedDB` API, that enable them to store origin-isolated and persistent data in the browser [58]. Browser extensions can utilize these storage APIs to store or retrieve data through JavaScript since the extension JavaScript (e.g., content script or page-injected JavaScript) also executes in the page context. In addition, extensions also have dedicated access to a privacy-friendly *storage* API [57]. The *storage* API is not accessible to Web applications and is even extension-isolated, i.e., an extension can only read data stored by itself and not by other extensions.

Security-related HTTP Headers Web application servers deploy various HTTP response headers [71], such as `Content-Security-Policy` (CSP) [59] and `X-Frame-Options` (XFO) [62], to instruct the browser about the desired security restrictions they wish to enforce on the client side. For instance, a Web server may send an XFO header with `SAMEORIGIN` value to deny any framing attempts by cross-origin applications. Unfortunately, extensions may find these strict security restrictions interfering with or blocking their intended functionality on the client side. However, tampering with these headers may also inadvertently expose the corresponding Web application to client side vulnerabilities (e.g., cross-site scripting, clickjacking, etc.) [39].

3 Related Work

In this section, we discuss various research studies related to our work that highlight the S&P issues with browser extensions. We also refer to other related developer-centric studies focused on the security and privacy mindsets of software developers since they also inspired our study design.

3.1 Research on Browser Extensions

Prior studies in this field primarily focused on either of the two behavioral aspects in terms of security and privacy: *malicious* or *benign-but-buggy* extensions. We focus our attention primarily on the *benign-but-buggy* extensions in this study.

Vulnerable Extensions: Researchers have also emphasized the fact that even benign-but-buggy extensions could inadvertently expose their users to client side security issues or allow nefarious actors to access privacy-sensitive data. For instance, as early as 2010, Barth et al. [9] showed that extensions are often over-privileged, posing grave security and privacy risks to the users, arguing in favor of enforcing isolation, privilege separation, and the principle of least privilege. Carlini et al. [14] manually reviewed extensions and reported a large number of vulnerabilities that could be abused by malicious websites to attack them. Further, Somé [85] and Fass et al. [30] conducted a large-scale analysis of extensions and identified vulnerable cases where malicious websites could exploit the buggy communication channels of these extensions to execute code in the privileged context, leading to universal-XSS or sensitive data exfiltration. Similarly, Eriksson et al. [29] highlighted that extensions could be subject to being attacked by other installed extensions, leading to XSS when poorly implemented. Yu et al. [101] built a static analysis framework utilizing the abstract interpretation of the extension code to detect vulnerable endpoints. Starov and Nikiforakis [88] added that while extensions often leak privacy-sensitive data, most happen accidentally due to poor implementation. A long tail of work on extension fingerprinting also makes clear that they present an additional surface for online trackers to indirectly track users through the extensions' observable set of actions on the client side [4, 45, 46, 51, 80, 81, 82, 83, 84, 89, 90, 95]. Lee and Kim [53] investigated the extension architecture and argued that a compromised renderer process of the browser might allow a malicious website to bypass the isolation mechanism and execute unauthenticated code or exfiltrate sensitive data. Orthogonally, Bui et al. [12] analyzed the privacy policies of Chrome extensions against their runtime behavior and reported inconsistencies among their data collection practices.

In this study, we uncover the fundamental factors behind such benign-but-buggy implementations by extension developers and their knowledge on associated threats.

Human-centered Extension Studies: Kariryaa et al. [47] inquired about the S&P-focused mental model of extension users through an online survey. They reported that users are not fully aware of the potential of browser extensions and their S&P implications and, instead, trust the developers to handle such issues. Orthogonally, Nisenoff et al. [70] performed an extensive study and built a taxonomy of breakages incurred by ad-blocking and anti-tracking extensions by analyzing store reviews and GitHub reviews along with interviewing extension users. Whereas, Roongta and Greenstadt [74] proposed revised benchmarking metrics to address usability concerns of privacy-preserving browser extensions.

In contrast, our study addresses the research gap by understanding the developers' S&P mindset and their considerations while developing and publishing extensions.

3.2 Developer-centered Security Studies

Security researchers have investigated the S&P-knowledge and the development style of Web developers using either qualitative or quantitative methods or both in the past. Roth et al. [75] performed a qualitative study with Web developers to understand the root cause for misconfigured CSP policies deployed by websites. They interviewed 12 Web developers and asked them to work on a CSP-related coding task and explain the associated threat model through a drawing board. As a result, they reported an apparent lack of knowledge, misaligned expectations, and incentives for Web developers to use CSP. In a follow-up study, Roth et al. [76] also investigated the roadblocks for deploying secure sanitizers required for the newly introduced *Trusted Types* API by applying similar methods. This study with 13 Web developers revealed that the impact of misleading information sources, in combination with the complexity of the mechanism and incorrect understanding, leads to deploying trivially bypassable sanitizers.

More generally, Gorski et al. [37] conducted an online experiment with 53 Python developers to identify and address insecure usage of cryptographic APIs to improve code security. They found that API-integrated security advice tools, when placed close to the development environment, are effective in curbing API misuse and insecure design choices. In an orthogonal direction, Fischer et al. [31] conducted a quantitative study on the security-related code snippets available on ∼4K *Stack Overflow* posts and their usage in 1.3M Android applications to understand the prevalence of copy-pasted code in Android ecosystem. They reported over 15% of all Android applications to use one or more code snippets from these posts, where 97% of them were insecure. Naiakshina et al. [69] conducted a mixed-methods study with 43 freelance developers to understand their password storage strategy. They uncovered weaknesses in the secure storage strategy and a false sense of awareness (e.g., using `base64` to "encrypt") that developers may often have. Recently, Serafini et al. [78] performed a literature review of the research studies using qualitative or quantitative methods from selected conferences to identify the hurdles in recruiting candidates for security studies. Through their findings, they advocated for increased sample size and compensation, using reliable recruitment channels and transparency for recruitment.

Our work is orthogonal to prior work in the space by explicitly focusing on developers of extensions, for which security issues affect not just a single site or sites relying on specific third-party components, instead, could unknowingly affect *every* website the user visits.

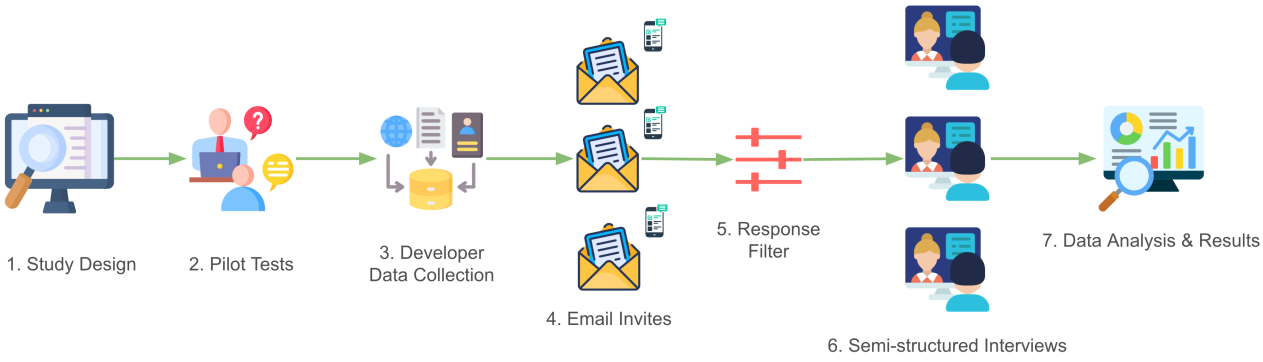

Figure 1: Summary of the overall steps of our study procedure.

4 Methods

Our study aims to understand the overall S&P mindset of extension developers and their specific courses of action while programming browser extensions. The latter should demonstrate how developers exercise (or do not exercise) knowledge relevant to the security and privacy of extensions in their programming practices. Hence, our research design aimed at the analysis of empirical data collected through (1) the video-recorded observation of developers' programming behavior in combination with (2) more general statements about their S&P-relevant knowledge, mindset, and biographic background elicited in semi-structured interview setting. These general factors (2) cannot be directly observed during coding tasks; they must be asked. Hence, we developed a qualitative interviewing strategy that combined a semi-structured interview with video-recorded coding tasks, with the latter embedded within the former. The coding task served as the primary focus of our data analysis, which was then complemented by findings from the interview data. Figure 1 describes the overall steps in our study procedure.

The interviews lasted, on average, 120 minutes and were conducted via Zoom between July and November 2024. As compensation, we offered all interview participants an Amazon Gift Card worth $50 (or its equivalent). Further, the responsible ERB for our institution reviewed and approved our study procedure (refer to Section 7 for a detailed description of our ethical considerations).

To verify the feasibility of our methods, we conducted five pilot interviews before recruiting the actual extension developers (we excluded these interviews from our analysis). These tests enabled us to refine the coding tasks by adding comments and improving task instructions to reduce unnecessary cognitive burden for participants. It also highlighted the need to reduce the overall complexity of the tasks, which is why we removed one of the originally considered tasks. For more information on the rationale, please refer to Appendix A of the supplementary material [3].

4.1 Semi-structured Interview

To gather data needed for answering our research questions, we conducted semi-structured interviews [38, 52]. For this purpose, we designed an interview guide that focused on the following topical areas: 1) developer's background and experience, including their purpose and their motivation for developing browser extensions; 2) their programming and publishing practices including their general approach to extension development, key considerations, and guidelines they follow during the process; and 3) their S&P mindset covering their general S&P awareness, their specific S&P considerations during development, and the roadblocks they face in the process (refer to the repository for the Interview guide [3]).

The coding tasks (explained in Section 4.2) were strategically placed between topical areas 2 and 3. This arrangement ensured that questions about developers' S&P mindset (3) would not influence their performance on the programming tasks. Once participants completed these tasks, and before we continued with topical area 3) of the interview, we further asked follow-up questions about participants' programming decisions and activities during the tasks and additional steps they would consider necessary to publish the coding task extensions. In the last phase of the interview, we also inquired about their perception and experience of transitioning to *MV3* standards, in case the participant did not mention this already.

4.2 Coding Tasks

The second stage of our interview required participants to work on two consecutive coding tasks implicitly focused on their security and privacy mindset. For this, we created a mock e-commerce website. Participants were tasked with working on the extensions that enabled additional features on this website. Notably, the website did not include any actual logic to place orders or send data to our servers. It only contained the pages and HTML forms absolutely necessary to facilitate the coding tasks. Participants were then provided with task descriptions and corresponding boilerplate code.

Due to the nature of the tasks and the varying levels of expertise among participants, we explicitly informed all participants that, while they were encouraged to complete the coding tasks, this was not strictly required. They were allowed to verbally outline their approach in as much detail as possible in case they ran out of time, although we did not impose an explicit time limit. We believe this approach aligned with our research questions and provided an acceptable trade-off, as our primary goal was to understand the S&P mindset and practices that developers followed, along with their reasoning, rather than focusing on the exact lines of code.

At the beginning of this interview phase, we provided participants with a link to the artifact [3], which included a PDF with all the necessary information to complete the tasks and the extension folders corresponding to each task. The extension contained the necessary code (e.g., DOM interactions code, stylesheets, etc.), unrelated to the core of the task and was labeled accordingly for clarity. Participants were allowed to ask the questions and seek help from online resources or platforms, as they typically do when working on their extensions. Importantly, we explicitly asked participants to verbalize their thoughts and consider a browser-agnostic, publication-ready extension while outlining their approach.

Next, we explain the rationale behind selecting individual coding tasks and describe their structure and design.

Coding Task 1: Save Address For Later

Researchers and practitioners have repeatedly demonstrated that malicious websites could compromise browser extensions to steal privacy-sensitive data. Lee and Kim [53] showed that a compromised browser renderer could allow a malicious actor to steal all the extension-stored data and argued that extensions should not store privacy-sensitive information on the client side. This is in line with the recommendations put forth by Chrome and the Mozilla Developer documentation [23, 57]. Agarwal et al. [4] reported that many extensions still use the traditional *Web Storage* APIs instead of the dedicated *chrome.storage* APIs for the extensions to store data and showed that this behavior could be reliably used to fingerprint the existence of an extension. Recently, Solomos et al. [84] affirmed that extension-stored data could be abused to detect their presence on the client side. Hsu et al. [42] also reported the *storage* to be the widely-used permissions by all classes of extensions. We modeled the prevalence of the data storage behavior among extensions and the associated privacy threat highlighted by these studies in our first coding task.

In line with these considerations, Coding Task 1 required participants to enable a *"Save Address for Later"* feature on the website through an extension. The feature was intended to allow users to "record" and "retrieve" their address data when ordering any product from the provided website, but it lacked this basic feature. Given that the address data could be, at the very least, classified as semi-sensitive, we believe that handling such cases through extensions is a privacy-sensitive action, and developers should be equipped to handle it appropriately. Thus, we designed this task to capture the developers' privacy awareness and to determine the set of actions along with their underlying explanations. The extension scripts already contained the code for DOM interactions and extract/-fill the HTML form. Participants were only required to store and retrieve the data based on their preference and choice of API(s) (as in Listing 1 in the supplementary material [3]).

Coding Task 2: Check prices on Amazon

Contrary to the first coding task, the second task focused on their security-relevant decisions. Prior studies on browser extensions reported that malicious extensions often tamper with security-related HTTP headers to bypass security restrictions on the client side to execute untrusted JavaScript or perform clickjacking attacks, for instance [43, 44]. Contrastingly, Hausknecht et al. [41], and later, Agarwal [2] reported that even benign browser extensions often modify or even drop security-related HTTP headers, such as `Content-Security-Policy` (CSP) and `X-Frame-Options` (XFO), that may interfere with their client-side functionality and are benign in nature. For instance, an extension may modify or drop an all-blocking CSP header to inject and execute its JavaScript in the page's context. Similarly, extensions may have to alter XFO if they wish to embed a Web page in an iframe that does not allow cross-origin framing. Naturally, modifying HTTP headers may have unwanted consequences if not handled cautiously. Since both malicious and benign extensions may exhibit similar header-modifying behavior on arbitrary domains, it is infeasible to discern their intent and take down the corresponding extensions proactively. In such cases, the extension developers must consider the security implications of their functionality and the header-modification strategy. Coding Task 2 aimed at understanding the generic security mindset of the extension developers. It used HTTP header modifications as a focal point, given that this behavior is exhibited by both malicious and benign extensions alike.

In Coding Task 2, participants were required to enable the *"Check Price on Amazon"* feature for the users of the e-commerce website such that they would be able to compare the discounted price on the website against the price listed on Amazon. To achieve this, the extension should add a button for each product in the DOM, which should open the amazon.com domain inside an iframe overlay. The *iframe overlay* contacted the publicly available Amazon search endpoint and passed the corresponding book title as the search query here, which was then supposed to be loaded inside the *iframe* [7]. The extension scripts already included the code to add the button on the UI, contact the search endpoint with an appropriate query, and the *overlay* to show the result. However, the iframe did not successfully load due to the `X-Frame-Options` header deployed by the website, which

restricted any cross-origin attempt to load the page within an *iframe*. The goal of the task was to provide a security-aware solution that allowed the *iframe* to load as expected. Essentially, participants needed to deal with the response headers here to enable the desired feature, however, only as per *MV3*.

CT-Independent Additional Hidden Flags

While the above two coding tasks capture the S&P mindset through specific tasks and functionalities, acting as an anchor point, there are certain development patterns that apply to all kinds of extensions. From a security standpoint, API and host permissions of browser extensions could pose threats when not specified adequately. Many studies have reported privilege escalation attacks on extensions [29, 30, 53], while the extension stores also strongly recommend developers to apply the *Principle of Least Privilege* to avert any exploitation attempts [20, 65]. Hence, we also inquired about the developers' knowledge of permissions in our coding tasks.

Specifically, we intentionally included redundant API permissions and set the host permissions of the content script and the web-accessible resources to `<all_urls>` in the manifest for each extension. Additionally, we also added the `all_frames` primitive for the content script set to *true* and the `use_dynamic_url` primitive set to *false* to capture the participants' awareness and knowledge on these primitives. The `all_frames` key (dis-)allowed the content script to be injected into all the *iframes* of the parent page while the `use_dynamic_url` key was used to configure whether the extension identifier is static or rotated for each session. A static extension identifier exposes itself to WAR-based extension fingerprinting [45, 80, 81]. Importantly, these redundant configurations are task-agnostic and do not affect the outcome of the solution of the two coding tasks above. Towards the end of the coding tasks, we explicitly asked the participants if they would like to make any changes whatsoever in the manifest to hypothetically publish this extension today. This supplemented our understanding of the development and publishing practices against their actions and observations here.

4.3 Sampling & Recruitment

We reached out to browser extension developers who had published at least one extension that was live on either of the two largest extension distribution stores (*CWS* & *AMO*) at the time the study was conducted. Hence, we collected the publicly available details of the developers and their extensions from the respective distribution stores through the sitemap of the stores [26, 67]. We then applied specific selection criteria to filter the developers and discarded those that did not fulfill the following fundamental factors - i.) they had to have published or updated one of their extensions since 1st January 2020, and, ii.) they had to have a valid email address. Many extensions on these stores had been first published as early as 2011 and had not been updated since. We assumed it was more meaningful to gather insights from developers who actively contributed recently. This ensured that the selected participants were likely to be aware of current security concepts and practices, aligning with our study's goals.

From the selected pool of candidate developers, we randomly chose 2,500 Chrome and 1,000 Mozilla extension developers separately for each store and sent email invites to participate in our study. The email invites included details on the study's background, its goal, and how we identified participants to establish the legitimacy of the sender and the email. The email also contained the link to a pre-screening questionnaire allowing participants to sign up for the interview study [3]. The questionnaire itself consisted of a total of ten questions that inquired about the technical experience, their educational and professional background, and their demographic information. Extension-related questions included the developers' years of experience with extensions, the categories of extensions developed (as visible on the stores), and the underlying motivation. Last, we asked them to provide their email address to schedule interviews (refer to Appendix B in the supplementary material [3] for more details).

Following the sampling principle of maximal structural variation [48], we then used the answers to these questions to identify developers with varying levels of expertise, motivations, and categories of extensions they had developed [54].

While reviewing the data from the pre-screening survey of participants who expressed interest in our study, we observed that the majority of responses came from developers who had published only one extensions with small user bases (10 users), and primarily as a leisure activity. Indeed, based on the statistics collected from the distribution stores, we found that only 1% (99th percentile) of all the published extensions have more than 10K users. Similarly, only 1% of all the developers have published more than 7 extensions. However, to adequately address our research questions, we deemed it necessary to also interview participants with larger user bases or having published multiple extensions, as their S&P mindsets might differ significantly from those creating extensions as a leisure activity. To ensure diversity in extension visibility and user base, which is critical for the validity of our results, we decided to supplement our initial sampling strategy by reaching out to popular developers in subsequent recruitment batches. To achieve this, we identified developers with at least one published extension having more than 10,000 users or those who have developed at least 7 extensions and exclusively sent email invitations to these developers.

4.4 Qualitative Data Analysis

As a result of our general interviewing strategy, each full-fledged interview yielded two complementary types of data: (1) an audio recording of the entire interview, which we transformed later into a written interview transcript by using our

institution's internal LLM-based transcription tool, and (2) a video recording of the participants' shared screen during the coding task stage. Technically, the entire interview, including the embedded coding tasks, was video-recorded using the recording tool of the video-conferencing platform that we deployed. However, we only used the audio recording of all verbal interactions with our participants for transcription and video recordings were exclusively used to analyze their programming activities, visible on their shared screens.

The sequential order of the data analysis differed from the sequential arrangement of the data-collection process: we started our videographic [49] analysis with coding-task segments of several interviews selected according to the principle of maximal variation between cases [48, 54]. From the entire pool of full-fledged interviews, we initially selected pairs of interviews that displayed different patterns of coding-task solutions. In this way, we identified six different clusters of participants. Members of each of these clusters had in common that they went for the same solution options in *CT1* and *CT2*. Then we selected two cases (whenever available) from each cluster and analyzed them in depth.

The first step of the in-depth analysis was the formal coding of screen-sharing videos and the corresponding transcript sections. The goal of this procedure was to identify different activity steps of the programmer and code them accordingly using a CAQDAS package [99]. In the next step, we created concise content-related descriptions of every step made by the respective developer. Then, we compiled workflow diagrams for every participant by putting step descriptions into a table where each column represented one interviewee from the sample core. This table allowed us to compare paths of action taken by these participants and to identify crucial decisions [27, 28, 98] as well as resulting activities which, in the end, led to particular coding-task solutions. Finally, we identified axial categories, i.e., abstract descriptions of key characteristics of each developer's coding-task "journey" [91].

Interview transcripts were analyzed using a combination of the "top-down" coding approach, as proposed by the Qualitative Content Analysis [55, 77], with some elements of the 'bottom-up' coding approach inspired by the Grounded Theory [11, 16, 34, 52, 91]. On the one hand, our interview guide determined the main topical areas addressed by our participants in their answers. Hence, we derived an initial set of codes from our interview-guide questions. On the other hand, however, we also developed some new codes in an "open-coding" manner while reading through the interview transcripts. Most of these "bottom-up" codes specified the content of participants' statements made within one of the topical contexts introduced by our interview-guide questions. Hence, new codes, created in the "bottom-up" process, aligned with more abstract "top-down" codes derived from the interview guide: the earlier turned into the specification of the latter. We created the final version of the interview code book by putting our initial set of codes and newly emergent codes into one hierarchical system of codes. We note that while a single researcher performed the entire content analysis, two additional researchers with >10 years of experience in qualitative studies were involved during the analysis and regularly discussed the approach to arrive at the resulting set of codes. We adopted this methodology as an acceptable trade-off due to the lack of a subject-matter expert on extensions who could also sufficiently analyze qualitative data and vice versa.

4.5 Limitations

Generalizability: Our study focused on developers with live extensions from the two largest stores, *CWS* and *AMO*, which may limit to some extent our sample diversity. However, this ensures our findings are rooted in real-world practices, as these developers have firsthand experience with widely applicable platform-specific guidelines and challenges. While we observed unequal gender representation and note that our results are shaped by participants' technical backgrounds and experiences, we assume that the insights gained still provide valuable insights to addressing key concerns in the ecosystem.

Study Environment: We explicitly instructed our participants to take all necessary actions during the coding tasks as they would in their own development process. However, we acknowledge the potential influence of the artificial interview situation on their behavior. To minimize discomfort, we gave them the option to explain their solution strategies verbally and ask for clarifications at any time during the task.

Biases: We acknowledge the possibility that our results may have been influenced by social desirability, opt-in biases, or demand effects [10]. To mitigate these risks, we refrained from disclosing the study's specific focus on security and privacy (S&P) topics during recruitment and encouraged open and candid discussions throughout the interviews.

5 Study Results

In the following, we outline the demographics of our study participants and conduct an in-depth analysis of our findings from the interviews.

5.1 Participants' Characteristics

We sent email invites to the extension developers, as listed on 4th July 2024 in *CWS* & *AMO*, based on our sampling strategy discussed in Section 4.3. We reached out to 4,042 developers on *CWS* and 2,449 developers on *AMO*, distributed across four batches of variable size between July 2024 – Nov 2024 This accounted for a total of 8,490 and 3,275 extensions, respectively. We received 322 survey responses, where 170 people finished the survey, and 165 of these consented to participate in the study. These 165 people span from all over the world and published extensions across various categories and with differing motivations (refer to Figure 2 and Table 2 in

Coding Tasks	CT2: *Drop XFO*	CT2: *Used ALLOW-FROM*	CT2: *No Solution*
CT1: *storage.local*	*P01*, *P08*, *P16*, *P17*, *P20*	*P11*	*P05*$^{+}$, *P06**, *P12*, *P21**
CT1: *storage.sync*	*P03*, *P09*	*P02*, *P07*	–
CT1: *storage.localStorage*	*P04*, *P10*	–	–
CT1: *Store in Background*	–	–	*P18*
CT1: *No Solution*	–	–	*P13*, *P15*, *P19*

Table 1: Combinations of individual participants' solution strategies for both coding tasks. (* – Did not finish, but outlined approach, $^{+}$ – the Amazon website went under maintenance, thus, rendering the task to be incomplete.)

the supplementary material [3] for details). We analyzed the pre-screening responses and sent interview scheduling instructions to 29 people. Unfortunately, we faced language-based restrictions with six people (since we planned to interview in English) while two of them did not respond. We successfully interviewed 21 of these interested participants, from 15 different countries in the world. As detailed in Table 2, 17/21 participants have an educational background in Computer Science, 15/21 are currently employed in Computer Science or related profession. 15 participants also had prior Web development experience. The participants also differed in terms of their extension development experience (min: 1, max: 20, mean: 4.58) and their age (min: 19, max: 56, mean: 32.44). The extensions developed by these participants had a combined installation of 6,162,000+ (6M+) across 37 Chrome, and 239,983 users across 19 Firefox extensions. Please refer to Table 2 for demographic, technical and extension-related statistics of individual participants and their extensions. While 4/21 participants had <100 install counts for their extensions, we still included them in our study as they showed apparent signs of S&P mindset (refer to our detailed assessment in the supplementary material (Appendix C) [3].)

Notably, three interviewees (*P13*, *P19* & *P15*) only participated in the non-coding task stages of the interview as the first two did not *directly* develop their extensions but were *Product Managers* in an extension-developing organization. Whereas *P15* was uncomfortable working on the tasks and mentioned that they did not read the interview objectives carefully. On the contrary, we discarded the interview of *P14* from the data analysis as this person mentioned during off-boarding that they had investigated the researchers' background and prepared for the interview accordingly.

5.2 Coding Tasks

In this section, we discuss the concrete solution strategies outlined by the participants and the associated critical factors they considered while working on the task. As summarized in Table 1, we compare and contrast the path of actions and decisions taken by individual participants for each task based on two factors - i) their internalized knowledge and ii) the critical junctures. The term "internalized knowledge" refers to participant's ideas derived from prior experience or their established patterns of thought related to the task or its particular solution. It is grounded in social-scientific studies on knowledge internalization by individuals and human groups like organizations [8, 50, 86, 96]. We contrast it with "external knowledge" not directly possessed by our research participants, which must be acquired from different (online) sources while working on coding tasks. The term "critical juncture" denotes situations where our participants had to make crucial decisions that led to a particular solution strategy. We borrowed it from the literature on institutionalist social sciences, which refers to large-scale societal developments [13] and adapted it to the analysis of individual actions taking place under structural constraints (requirements of coding tasks, publicly available data etc.). Figure 3 demonstrates our participants' flow of actions across coding tasks.

5.2.1 Task 1: Store Address Data

As explained above in Section 4.2, Coding Task 1 captured participants' privacy mindset and considerations. It required developers to implement a "Save Address for Later" feature on the website using the extension. 14/21 participants finished the first task, two implemented their approach semantically, i.e., included the code to record and retrieve the data using the data storage API of their choice. However, the final code did not execute without error due to syntactical issues while retrieving the data. One participant verbally outlined their exact approach instead of writing the code. Four different solutions emerged from this coding task. 14 participants chose the privacy-focused means to save the data – the *storage.sync* & *storage.local* APIs dedicated for extensions, two of them used the *Web Storage API*, and one of them argued in favor of storing the data in the extension background. Notably, the two storage APIs: *storage.sync* and *storage.local* only differ in terms of their functionality (i.e., the former enables cross-device sync) while providing identical isolation and privacy guarantees from other extensions and the websites [57]. Thus, we consider these two to be privacy-focused solutions here.

Internalized vs External Knowledge: 7/14 participants who chose to use the *storage* APIs explicitly indicated that they are familiar with the API and/or have worked with it. Thus, they followed a linear set of steps aimed at the solution envisioned by them from the very beginning. *P01* referred to their previously developed extension to extract the associated code snippet, whereas *P03, P06, P09, P11, P20* and *P21* referred to the official documentation (i.e., *Chrome* or *MDN*) to

implement their own solution (*...we're gonna use storage.sync ...I just prefer sync most of the time – P09*). *P11* and *P21* could not solve the syntactical issues with the getter API due to their lack of knowledge in handling the *async/await* constructs in JavaScript. Interestingly, while *P17* did not make an explicit statement to indicate their familiarity with the API, they arrived at the solution strategy during implementation by using *Supermaven* – an AI code-completion plugin for IDEs [92]. The plugin instantly auto-generated the code without even having to provide the input seed character to clarify their solution strategy. In contrast, *P09* used *GitHub Copilot* but also provided an initial seed to auto-complete the rest of the code.

6/14 participants did not have any prior knowledge on the task, three of these searched on the Web for related resources and arrived at the common solution strategy – the *storage* APIs. One of them used the entire task description as a prompt to *ChatGPT* and copy-pasted the solution from there. On the contrary, the remaining 2/14 (*P12* & *P16*) initially decided to use the *localStorage* API and searched for related resources on the Web. However, the search engine results directed them to use the *storage* API for extensions as their final strategy.

P04 decided to use the *localStorage* API very early and did not deviate from their strategy. Whereas, *P10* searched on the Web and had a cursory view on a *Stack Overflow* post suggesting the *storage* API. However, they chose to refer to the MDN docs listed in the search engine result and decided to use the *localStorage* API. Orthogonally, *P18* suggested to store the data in the extension core with *IndexedDB (...register a new background script, create index database so it's going to be like a persistent thing)*. Interestingly, the solution here does provide the same privacy guarantees as the *storage* APIs since the extension core is contextually isolated.

The prior knowledge and experience of the developers play a crucial role in developing strategies to implement privacy-sensitive functionality for their extensions.

Information Sources: The search engine results and the information sources referred to by the participants were pivotal for the end strategy. For instance, *P16* initially indicated to use the *localStorage* API. However, when searching for the syntax and the semantics, they ended up on the Chrome docs on the *storage* API and copy-pasted the sample code to formulate their own solution.

"...we would collect I guess the address we would then store that in local storage" – P16

Similarly, *P12* also contemplated different storage mechanisms they would ideally use for the task: *cookies* or *Web Storage*, and searched for related resources. However, the search results again listed the MDN docs on the *storage* API here, which they then followed to arrive at their solution.

P10, contrastingly, did not follow the *storage* API strategy they came across in the search results but opted to look specifically at their preferred resources – MDN docs.

"...I'll see Firefox, I like Firefox documentation" – P10

Interestingly, the *Google* search result to the query only showed MDN resources on client-side storage APIs [58] and not the extension storage APIs [57], contrary to the case of *P12* who used *DuckDuckGo*. Naturally, the search results differed based on the query itself. However, we empirically tested this to compare and reproduce the results and noticed that the chosen search engine and the indices may influence the top search results shown to the users, as was the case here.

To summarize, we observe that the *accessed and available* information sources, coupled with personal biases, may also influence the development practice, thus leading to more or less S&P-focused strategies.

Privacy Considerations: After the participants stopped working on the task, we asked each of them to explain their solution strategy, the underlying reason(s) for choosing a particular strategy, and any alternative strategy that they could think of. Four participants suggested using encryption before storing data on the client side. However, *P11* underlines that encryption on the client side is merely a security theater, as they would then have to also protect the cipher keys. *P04* was only concerned with storing data in plain and mentioned that they would store the data in a session-bound manner such that it is deleted once the user is logged out. *P03* and *P09* also mentioned that since users implicitly trust the browsers, it makes sense to store data in the browser. *P11* further added that storing data locally offers better privacy guarantees, and mechanisms like *Apple Keychain* may even allow to preserve its confidentiality. Contrastingly, *P03, P06, P09* and *P17*, who previously used the *storage* API based on their prior knowledge, suggested using any of the *Web Storage* APIs, as they are functionally similar and even easier to implement *(...start storing it in local storage maybe tag it to the session storage – (P06))*. On the contrary, five participants recommended storing data, optionally with encryption, on a remote server or using third-party service providers.

"would stored it through Firebase. It takes care of the encryption on its own" – P05

We observe that while developers may resort to specific developmental patterns that seem to be privacy-focused, unfortunately, those patterns are often driven by functional benefits rather than their concrete understanding and considerations of the associated S&P risks. For instance, *P03* mentioned that they did not use the *localStorage* API in the first place only because it is not supported in the background script, and Chrome may altogether scrap the API in the future. *P17* mentioned that the deciding factor was the persistence of data with the *storage* API, as websites can wipe out the *localStorage*. Similarly, many participants considered *Web Storage* APIs as an alternative as they were unaware of the differences with the *storage* API. *P07* only realized the differences after the task and confirmed with us.

"...if I were to call Chrome store.sync here and save this address, would other extensions have access to this" – P07

Key Takeaway: Although most developers solved the task using the privacy-focused *storage* API, they were unaware of its benefits and mentioned using the *Web Storage* API as an alternative.

5.2.2 Task 2: Check Price On Amazon

The second coding task focused on the security-related mindset and considerations of extension developers. The task required them to handle the XFO headers sent by the test application such that it allowed the extension-injected *iframe* to be successfully displayed on the website. 12/17 participants completed the task with a working solution while three of them did not finish the task. *P12* and *P18* opted out from the task due to the lack of background knowledge and expertise, and time constraints, respectively.

Internalized vs External Knowledge & Path of Actions: Contrary to *CT1*, only one participant explicitly indicated having internalized knowledge of the problem statement and the solution strategy. The rest of the participants never modified HTTP headers through browser extensions before and obtained familiarity with the task by seeking help from different information sources (acquiring external knowledge). Some participants were even surprised to find out that extension could modify HTTP headers. Thus, we focus on the statements made and the paths taken by the participants to formulate their solution strategy here based on the information sources they referred to. Seven participants took a sequential set of steps to arrive at the solution based on the concrete suggestions from the *Stack Overflow* post or *ChatGPT* they followed. Whereas two of them verbally presented their initial solution strategies to get feedback from the interviewer, and then took a *detour* to navigate to the correct solution. The remaining *three* participants tried and tested different unrelated approaches (e.g., using web accessible resources, configuring HTML attributes for the *iframe*, enabling CORS headers, etc.), as pointed by *ChatGPT* or based on their whim before the interviewer directed them to the right path. Notably, while we occasionally felt compelled to explain the task requirements in further detail to save time, we made sure not to influence the end solution or participants' mindset in any sense.

Information Sources & their Impact: The participants referred to various resources on the Web to familiarize themselves with the task and devise a solution strategy on the right path. The most prominently referred sources were *Stack Overflow* (SO) and *ChatGPT*. In fact, 6/12 participants implemented the entire solution based on the answer from a popular *Stack Overflow* post on header modification [87], which also includes *P03* with prior experience with header modifications. Two others were also inclined to implement their solution based on the popular SO post here, but only after reading the entire official docs on the corresponding *declarativeNetRequest* permission required to intercept the headers.

Seven participants consulted *ChatGPT* to obtain a solution *(...this is a good case to just ask like an ai assistant – P09)*. However, in most cases, they only received suggestions that used web-accessible resources to inject a static HTML page, which is an incorrect solution here. Two of them, however, were able to formulate their entire solution through *ChatGPT-derived* suggestions on dropping the XFO header, while *two* participants used the suggestion here together with the popular SO post above and other blog posts to implement the solution.

Two remaining participants copy-pasted code from the sample extensions provided by the MDN [60]. *P10* also dropped CSP in addition to the XFO header, as suggested by the GitHub resource they referred to. We note that the participants often asked us to help with the correct configuration of specific primitives within the header-modifying ruleset (e.g., *initiatorDomain*, *resourceTypes*, etc.) they copied from different resources. However, we limited our feedback to avoid unintended influence on the final outcome.

We observed a direct impact of the resources referred to by most developers where they simply copy-pasted the code and did not change anything further for the functionality to work, except in three cases. All three of these participants implemented a solution that would keep the XFO header intact and allow-list the embedding domain for framing through the long-deprecated *ALLOW-FROM* attribute. Additionally, the complexity of the task may also have influenced developers to avoid any deviations from the available solutions.

Security Considerations: As highlighted earlier, all but one participant did not know that browser extensions could modify the HTTP headers or have not modified them through extensions. Nevertheless, 10/15 participants explicitly stated that meddling with HTTP headers is dangerous and were skeptical of performing such operations in their own extension. This is irrespective of the fact that only 6/15 participants were vaguely familiar with the XFO header *(...it seemed a little bit, bit hacky – (P16))*. Two participants also felt that such behavior in extensions could lead to less trust among users. *P03* added that, unfortunately, there may not be a feasible alternative to enable expected behavior within extensions *(...can't really think of another way to get around this – P03)*.

Fortunately, eight participants were open to exploring alternative approaches and even suggested opening the page in a new tab. Three participants intended to explicitly allow the framing behavior through modifying the values of the XFO using the *ALLOW-FROM* attribute. *P02* searched on the Web about the header and extracted the attribute from the summary auto-generated by the Google Search AI [35]. Whereas *P07* and *P11* did not perceive the deprecation notice for the attribute on MDN docs [61]. Unfortunately, their attempt to preserve the header and provide a "better" solution only resulted in a *no-op* since XFO with *ALLOW-FROM* is ignored by most modern browsers, and no security policy is enforced.

In light of these findings, we can assume that extension developers act responsibly and are mindful of the security posture of their extensions. As evident in the cases above, the

general notion of certain actions, such as script injection or header modification, positively influences the developers even when they are unaware of the exact details. However, lack of S&P awareness and attention to detail could also give them a false sense of security, similar to our observation in *CT1*. This was evident with the three participants resorting to the deprecated *ALLOW-FROM* primitive.

Key Takeaway: Despite not being aware of the exact solution strategy, most developers were hesitant and skeptical of publishing an extension that modified HTTP headers. In fact, they readily suggested enabling the expected behavior through new tabs as a trade-off.

5.2.3 Permission-related Flags

As outlined in Section 4.2, the manifest of both extensions contained two redundant API permissions and host permissions set to `<all_urls>`. We explicitly asked all our participants to review and make any necessary changes they would like to make throughout the extension, to hypothetically publish the extension right away. Importantly, we did not explicitly mention or lead them to the *manifest* or any other file in particular. Interestingly, only 8/17 participants could identify at least one redundant API permission in either of the two coding tasks. In contrast, host permissions were not apparent to most participants as only six of them suggested changing them across the coding tasks. Additionally, only one of them could detect host over-permissions among *web_accessible_permissions*, while none of them actually reported the redundant use of `all_frames`. Two participants (*P11* & *P16*) *"suspected"* host and API over-permissions and mentioned *"to review it later"*. Overall, 8/17 did not report any permission issues in the manifest. Similarly, 15/17 participants did not comment at all on `use_dynamic_url`, while two of them inquired what it was. We believe that the detection of API and host over-permissions in even one extension is representative of the associated development practices of participants to account for the cognitive load and the influence of the study environment.

We observe significant deviation from what the participants believed to have inculcated in their development habits versus the patterns exhibited during the coding tasks. For instance, *P04* claims to use the minimum possible set of permissions; however, did not notice any permission bloats. Similarly, *P09* also mentioned that they are generally very careful about permissions *as it may determine the length and difficulty of the review process*; however, they could not identify the issues. While *P20* accurately removed the *scripting* permission from the manifest of *CT2*, they supported the use of the redundant *cookies* permission – *"...cookies permissions are fine"*. In general, while most participants reported being cautious with permissions and reviewing them before submitting them to the stores to avoid rejection, they only considered the API permissions and did not show any concerns for host permissions.

Key Takeaway: Developers may implicitly follow the *Principle of Least Privilege* for APIs, however, potentially only to reduce the reviewing and publishing overhead. This may also realistically differ based on their (lack of) knowledge of related primitives and desired values.

5.3 S&P Threats and Awareness

We asked our participants about their awareness of S&P threats associated with browser extensions and how they model them for their own extensions. All but one participant reported at least one security or privacy threat with extensions. The most commonly mentioned threats were data, history, and credentials theft, XSS, script injection, and key logging. Most participants with knowledge of security attacks had Web development experience and mentioned that it could apply to extensions, too. However, three participants also explicitly expressed their lack of knowledge and experience in modeling S&P threats with extensions. *Three* other participants were concerned with future malicious updates of popular extensions and the fact that the users do not notice such behaviors. *P11* disregarded the Web history of users as PII *"since browsers collect them anyways"*. *P08*, interestingly, believed that data collection is not a privacy problem and platforms should relax the restriction around it to incentivize developers and denied cases of data theft through extensions in the past.

"I want to make some money of it, how do I do that without collecting data?" – P08

In general, most developers showed a high level of understanding of the S&P threats with extensions, but lacked knowledge on how best to mitigate against them *(...I have no idea how to make it safer – P01)*.

We also asked our participants about their knowledge of specific S&P threats and related concepts.

Inclusion of Third-party SEO & Analytics Script: Five participants reported that they get emails every week by *"third-party SEO & Analytics companies"* to include JavaScript into the extension to get better insights on their user base. These entities even offered a monetary compensation between $50–$100 per 1,000 users with possible negotiation on call or even showed interest in buying the extension for $10,000, as mentioned by *P21*. *P04* was concerned that these scripts may inject ads or even be "malware-ish," which could affect their business and credibility, while *P17* believed they might be interested in affiliate fraud. *P19* added that these companies are often interested in popular extensions and contact them to include third-party scripts. Interestingly, *P13* worked in such a company and was aware of such malpractices.

"...these companies just scrape a lot of information... I think actually the stores don't do so much against it" – P13

We believe this to be dangerous and an anti-S&P pattern in the ecosystem and would urge the platforms to strengthen the review process to thwart such behavior.

Impact of Malicious Websites: Six participants said they did not consider this before. Among these, *P07* retrospectively realized this being an issue with one of the ad-blockers they used, while *P09* mentioned that they were worried about *Twitter/X* blocking their extension, as it operated on the platform. Orthogonally, five participants denied the fact that extension could be attacked, stating that this is unrealistic. Three of them could think of theoretical ways (e.g., malicious data flow, *Web Storage* APIs, etc.) to exploit the extension but were not sure if this would be feasible (*"I don't know even if it's possible" – P18*). Unfortunately, we observe an apparent lack of knowledge of the threat surface among extension developers, which may negatively influence their secure development processes.

Extension Fingerprinting: Seven participants reported that they had never heard of fingerprinting through extensions and were interested to learn more about it. Six of them were familiar with the overall concept but did not model them for their own extensions. Of these, *P07* mentioned that they do not know how to protect against such issues, while *P15* mentioned that it is impossible. *P08* did not see this as an S&P problem and said that it is statistically difficult to fingerprint extensions. Interestingly, *P11* believed that platforms may be interested in tracking users; thus, no solution exists *(don't think Google would want to mitigate that. They want to track people – (P11))*. *Three* participants showed a clear understanding of the topic and mentioned having deployed appropriate measures. *P20* reported that their users, who perform penetration testing and archiving with the extension, were concerned about being fingerprinted by websites, and thus, the extension company removed all the web-accessible resources to avoid tracking through identifiers. *P18* also mentioned having defensive measures in place against fingerprinting; however, on manual inspection of their live extension, we found that it was not actually the case. At the very least, they injected web-accessible resources on *://*/* without setting `use_dynamic_url` to *true*. Given that the extension offered a privacy-critical solution to their free and premium customers, we believe such misconceptions may have grave consequences. Importantly, none of them knew about the `use_dynamic_url` primitive and its usage before, which leads us to believe that developers are not aware of mitigation against extension fingerprinting.

Isolation Contexts and Mechanisms: We asked participants about their understanding of the isolation across different extension components and its relevance. Four of them did not know about the concept. This included *P18* who worked in an organization that developed S&P solutions as extensions. On the contrary, five of them assumed this to have something to do with the security of the extension, while two of them had clear misconceptions: *P05* thought it existed to prevent bots, while *P10* confused isolation with permissions. *P09*, *P19*, and *P21* seemed to have thoroughly understood the concept and mentioned that it prevented privilege escalation and restricted the interaction between malicious websites and extensions.

Key Takeaway: Extension developers are familiar with the S&P threats and are concerned about their extensions' behavior. However, our participants do not have sufficient knowledge to act accordingly.

5.4 S&P Practices & Developmental Patterns

We captured the implicit S&P patterns of the participants during the coding tasks and asked explicit questions about their best practices. As highlighted in Section 5.2.3, we observed many developers to show caution against using excessive API permissions. Further, they were open to exploring less intrusive solutions for *CT2* and were hesitant to publish extensions that modified HTTP headers unless absolutely required. *Three* participants even suggested avoiding interaction with the DOM as much as possible and, instead, enabling similar features in the extension UI popups. However, this was not the case for *CT1*, where most participants focused on the ease of implementation. Participants who opted for the *storage.sync* API also reasoned along the lines of its cross-sync capabilities rather than its privacy benefits against their Web counterparts.

When explicitly asked about specific S&P-related practices, we observed various development patterns and a few anti-patterns in their responses. *P03* and *13* said to have enforced the Principle of Least Privilege. *P18* mentioned that they have a secure end-to-end software development life cycle and zero-knowledge structure enforced for both free and premium tier extension users. *P11* argued that remote servers should only be contacted for synchronization across devices and nothing else, while *P20*, *P09*, and *P02* favored for data storage on remote servers for better privacy guarantees. Contrastingly, *P08* believed that privacy depends on the users' definition. *P04* also mentioned that privacy is a minor concern here. *P04* also said that they avoid the hassle of going through the review process by only pushing annual updates to their extension. They do so by offloading most of the logic to their website on which the extension operates and, further, injecting the code to the extension at runtime. Notably, this behavior is against the store policy, and they are even concerned that the extension might be taken down in the future [18]. However, they believe that the review process might slow down their business needs and their users' expectations.

S&P-related Resources: Most participants followed the official docs of the platforms as a single source of information for most of their learning. *P03* added that their Web development experience also adds to the understanding, while *P21* said that much of their expertise comes from the security training they undergo for their employment. *P17* relied on community support for S&P knowledge, such as Discord servers and Slack channels, common social groups for techies and entrepreneurs, and YouTube videos, whereas *P06* followed SubReddits. However, all of them agreed upon the fact that one needs to actively look upon the Web for best practices and guidelines, as the official docs does not always help.

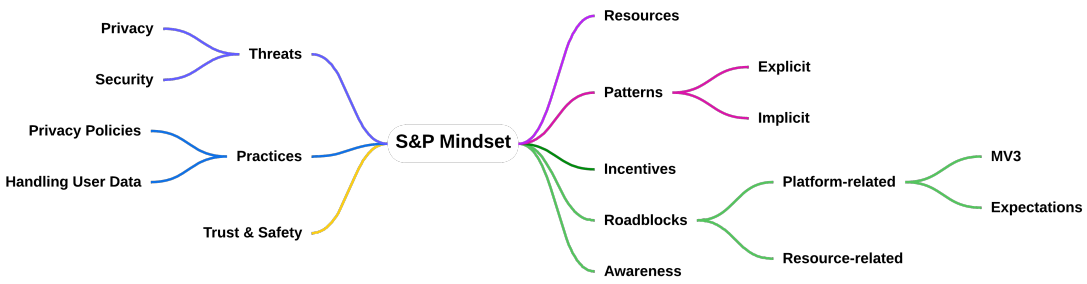

Figure 2: The mindmap of the S&P-related themes that we identified in our data analysis.

Privacy Policies & User Data: 14 participants mentioned having a privacy policy for their extension either because it was a mandatory step or the platforms nudged them to include one. Three of them complained about the lack of support for creating these policies. Business organizations, as in case of *P13* and *P20*, consulted legal experts to form privacy policies compliant with global regulatory frameworks. In general, several participants emphasized that they do not store sensitive data or only store data that the user consented to. *P18* talked about maintaining a zero-knowledge structure about their users' data, whereas *P09* believed in being as transparent as possible to their users about their data and privacy policies. *P08*, however, argued that developers should be allowed to collect user statistics or anonymized data to incentivize development and maintenance. *P13* said that some extensions harvest a lot of user data and monetize these.

Trust & Safety of Users: Many participants stated that they considered different trust and safety factors and related practices during the interview. *P08* and *P11* mentioned that they trust *Google* to perform thorough review checks on submitted extensions and their importance in building users' trust. *P03* and *P09* argued that since users inherently place trust in browsers by using them, it makes sense to utilize the browser APIs for storing sensitive data to ensure users' safety, whereas *P06* contradicted this, and preferred to do this remotely. *P21* believed that open-sourcing the code helps build trust among users, which they did, while *P10* recommends fellow developers to *"do exactly what you describe"* to instill trust among users. On the contrary, *P04* said that the review process is not foolproof, and the extension takedown can create trust issues with users, which can have implications for businesses.

Key Takeaway: Developers often follow basic data processing and user consent standards but struggle with navigating to the correct resources to build further. Other factors, such as business needs and professional background, may also influence the development pattern.

6 Developer-External Influences on S&P

While prior research has extensively documented benign-but-buggy behavior in browser extensions through technical measurements [30, 53, 85, 101], these studies did not focus on identifying the root causes of such issues, or explored the S&P mindsets of extension developers. Our research addresses this research gap, offering insights into the challenges that hinder developers from creating S&P-friendly extensions (as summarized in Figure 2). We, now, examine the socio-technical, ecosystemic, and external factors, as reported by our participants, that influence the S&P mindset of extension developers.

Economic Friction: Developers highlighted a lack of incentive for them to publish and further maintain extensions. Of the four participants who already ran businesses and generated revenue through their extensions, one even said that they would be nervous if their business depended entirely on the extension. Among other reasons, the lack of platform support combined with an opaque review process concerned them the most. It is a common belief that while extensions with few features neither lure enough users nor have the potential to generate monetary benefits, a feature-rich extension also incurs additional infrastructural costs. As per our participants, both direct monetary expenses (e.g., server rent) and time resources to maintain extensions (to remain compliant with the latest S&P best practices) necessitate some form of revenue.

Platform Dissonance & the TINA Effect: Participants showed a general consensus on the fact that Google's influence on the extension ecosystem and the act of technological control is an anti-pattern and concerning. Drawing comparisons to Mozilla, they mentioned that, while Mozilla is not perfect, Google is not developer-friendly and community-oriented, neither does it support new developers *(P10)*. The introduction of the *MV3* standards further fueled these arguments since the changes deviated from the established framework *(P04)*, and Mozilla was forced to adapt to the Google-induced changes *(P09)*. Beyond our study, ad-blockers also claimed to have been significantly affected by the controversial push by Google *(P03)* [1, 93]. Two participants even argued that privacy is a lost cause with Google since *"wants to track users anyway" (P01, P11)*. Given the market share Google holds, the developers showed the symptoms of *TINA effect*: simply put, *there is no alternative* given Google's large user base. However, the developers strictly argue in favor of Mozilla and even Apple to raise their voice against what they called an *"Internet monopoly" (P09)*. In general, our partici-

pants dislike the *"autocratic" (P17)* attitude of platforms in shaping the ecosystem. Although, they also collectively agree that the platform should act as the responsible stakeholder, but only through incorporating developers' and users' feedback.

Security Apathy: Although Google and Mozilla, as the main vendors of extensions markets, offer similar features, developers often find it painful to maintain a cross-browser product. This is because even when *most* of the APIs are supported by both browsers [97], they may have distinctive behavior, incurring additional overhead for developers to adapt. Further, the build and publishing steps, the review process, and the decision timeline differ for the two platforms. There exists no CI/CD pipeline to build and publish extensions without additional manual effort. Safari adds insult to injury since it has different publishing requirements. Many participants also complained about the inconsistent review process across the two stores and even across updates within the same store. The complex error codes that they may receive in case of rejections/takedowns are also difficult to parse, given that there are no *store liaisons* assigned to talk to. We believe that the overall tedious process of publishing extensions, compliance with the *blackbox* review process, and the lack of *unified* set of APIs across browsers lead to a sense of complexity fatigue. Unfortunately, this may result in considering S&P only as an afterthought over functionality and not by design.

7 Conclusion and Call to Action

We investigated the S&P mindsets of browser extension developers to identify the root causes of vulnerabilities in extensions. Our semi-structured interviews with 21 extension developers reveal that while developers often demonstrate an implicit awareness of generic security threats in the broader Web ecosystem, many reported having little to no knowledge of how to best protect their extensions against them. Developers attributed their lack of S&P awareness to the scarcity of relevant resources and insufficient support from platform providers to encourage secure development practices. Other contributing factors included misaligned incentives, tooling limitations in the extension publishing process, inconsistent platform behaviors, and opaque vetting mechanisms. Overall, these challenges often relegated S&P considerations to an afterthought in the development process. Here, we provide high-level recommendations to individual stakeholders, based on our observations in this study, for a sustainable ecosystem.

API documentation: 19/21 participants explicitly stated that they refer to the Chrome or Mozilla Developer docs for development. This was also reflected in the resources they referred to during the coding tasks. However, the official docs does not include the S&P benefits or issues around using certain APIs for development. For instance, neither the Chrome nor Mozilla docs on the *storage* API [57, 58] emphasized that the data stored through the *Web Storage* APIs is accessible to Web applications, and thus, not private. To exacerbate the issue, developers may either operate under the assumption that no S&P issues exist as they strictly followed the official guidelines or even give up due to lack of supporting resources. Here, we ask *browser vendors* to take concrete steps towards providing necessary details around S&P-critical APIs so that developers can make an educated decision.

Consistent Submission/Vetting Guidelines Browser vendors use a black-box vetting system to detect *suspicious* extensions. Yet many participants described it as inconsistent, with extensions wrongly taken down and later reinstated after clarification. While the system may favor false positives over false negatives, this discourages developers. For example, *P04* lost users and business when their extension was wrongly removed during the Christmas holidays. The problem is worsened by differences in *how* and *when* reviews occur. Mozilla requires both source code and distribution build for manual review, while Chrome only asks for the latter. Developers are reluctant to share source code with AMO, citing its monetary value. AMO allows instant publishing with review delayed up to six months, while Chrome vets and acts immediately, as noted by *P10*, a *Mozilla Recommended Extensions* panelist. To make matters worse, reviewers within or across stores may flag extensions for different, undocumented reasons due to the system's opacity. We recommend *vendors* improve transparency and consistency in the review process so developers can plan accordingly. We also urge *vendors* to align submission and vetting practices across extension stores.

S&P Practices: While adequate security and privacy-related information is imperative to take informed decisions, as observed in our study, this is unfortunately not always enough. Most participants were either unaware of security and privacy threats associated with extensions or only had surface-level knowledge of user data exfiltration through extensions. Moreover, none of them explicitly modeled any of the threat vectors for their extension. This also underlined by our findings from the first coding task, where the participants only considered the *privacy* of address data only when explicitly asked about alternatives. Thus, we recommend *developers* to proactively consider the threats and the usability vs S&P tradeoffs during individual steps of extension development.

Developer-friendly Policies Lastly, several participants questioned the deprecation or refactoring of existing APIs in favor of newer, limited alternatives. This reflects broader concerns around the restrictions introduced by Chrome's MV3 standards [32, 40]. For example, *P03* preferred sticking with `chrome.storage`, citing fears that Chrome may deprecate other APIs without notice, increasing migration burden. Similarly, *P07* and *P09* noted that browser vendors often make impactful policy changes without accounting for developer input. To ease developer fatigue, we urge *vendors* to incorporate developer feedback into policy decisions, and encourage *developers* to proactively communicate legitimate concerns to vendors when possible.

Ethics Considerations

We proactively considered and modeled the ethical and legal factors associated with the individual stakeholders that may be directly or indirectly affected throughout the course of the design and execution of this study. Additionally, our Institutional Review Board approved our study design and the methodology before we moved to the execution stage. Below, we discuss in detail the potential risks and benefits associated with these stakeholders in line with our adopted methodology.

Recruitment Strategy: We collected the publicly available email addresses of the extension developers and associated details on their extensions that were available on *CWS* and *AMO*. However, we only sent invites to those developers who fulfilled our study criteria. While we could have recruited participants through social media platforms and developer-focused forums, we would have been unable to derive concrete details and statistics on their live extensions. Further, none of the participants expressed any discomfort or anguish; some were even positively surprised to be contacted for the study.

Participants' Rights: In the first stage of our study, we sent email invites to the extension developers, inviting them to participate in our study by completing the pre-screening survey. The questionnaire included a detailed explanation of the purpose of the data collection, and we only collected data from those individuals who provided explicit consent to participate. Further, we separately collected individuals' consent for data collection during the interview (i.e., audio, video, and screen recording), provided to them along with the scheduling details. In addition, right before starting the interview, we verbally explained their rights, including the option to quit at any time or to skip answering questions they felt uncomfortable with during the interview. Participants could also switch off the video feed during the interview if preferred. During recruitment, we only informed participants that we were Computer Science researchers to minimize potential biases in their responses. However, we made sure to inform them about our full background and the purpose of the study after the interview.

Coding Task Design: We asked our participants to download the extension files provided by us to work on the coding tasks and share their screens. An alternative approach could have been to provide them with remote access to our machines for the coding tasks to avoid any risk of capturing user-sensitive data during screen-sharing. However, we provided them with the extension files to allow them to work on their accustomed setup that they use for development: the IDE, the AI tools and plugins used for development, and the preferred browser. This approach allowed us to capture their real-world development tools and environment while providing them with the flexibility to work as they usually would. As a precautionary step, we explicitly asked them to close any sensitive applications before starting screen-sharing, both in the email containing interview details and verbally before the task began. Unfortunately, one of the participants did not carefully read the email and refused to work on the coding task, as they were reluctant to download files to their machine.

The test website and the extensions designed for the coding tasks did not request or capture any privacy-sensitive identifiers (e.g., IP address, device identifiers, etc.) at any point and were hosted within our institutional network. The second coding task involved embedding a live website – Amazon, in an `iframe` as a solution by appropriately handling the framing policy enforced by the website. We resorted to using Amazon for our study to factor in the proximity to the real-world scenario of the coding task in terms of the participants' familiarity with the website along with the severity of the perceived threat. Additionally, the task only required to query the publicly exposed Amazon search endpoint for products and load inside an `iframe`, with no further interactions required. This ensured not violating any terms of service [6, 94]. Hence, we believe that our coding task did not incur any tangible or intangible damages to the live website.

Data Collection & Transparency: As discussed before, we collected user data at different stages of our study for analysis. Importantly, we stored all the data collected through the pre-screening survey and the interviews within our institutional premises. Further, we deleted all the personally identifiable information (e.g., demographics, audio/video recordings, etc.) associated with the participants upon the completion of data analysis and only stored the anonymized and aggregated results for later verification. We used the in-house instance of the OpenAI Whisper model for transcription services. Naturally, the artifacts related to this study are accessible exclusively to the members directly involved in the study. At no stage do we share any raw or processed data with any third-party entities.

Responsible Disclosure: We believe that the insights gathered from our study participants may help facilitate changes and guide towards an improved ecosystem, including, but not limited to, the security and privacy aspects. Thus, we plan to create an additional artifact that would include the participants' views, perceptions, and expectations beyond what is discussed in this paper. We plan to reach out to the store vendors and provide them with gathered insights so that they can take the necessary steps in the future. Naturally, we will not include any PIIs related to any of the participants in our study.

Open Science

As part of our commitment to the Open Science Policy, transparency, and the reproducibility of our study, we open-source the interview guide, the survey, the coding task package, the source code of the website, as well as (only) the code book generated from the raw data coded from our analysis [3]. We will *not* share raw data such as transcripts or video recordings for privacy reasons.

Acknowledgements

We would like to thank our reviewers and the shepherd for their valuable feedback to help improve our work. We thank all the pilot and study participants for their time and efforts. We also thank Matthias Fassl for his constructive suggestions.

This work was conducted in the scope of a dissertation at the Saarbrücken Graduate School of Computer Science.

References

[1] AdGuard. Mozilla solves the manifest v3 puzzle to save ad blockers from chromapocalypse, 2023. URL https://adguard.com/en/blog/firefox-manifestv3-chrome-adblocking.html.

[2] Shubham Agarwal. Helping or Hindering? How Browser Extensions Undermine Security. In *CCS*, 2022.

[3] Shubham Agarwal. Zenodo Artifact Repository, 2025. URL https://doi.org/10.5281/zenodo.15631753.

[4] Shubham Agarwal, Aurore Fass, and Ben Stock. Peeking through the window: Fingerprinting browser extensions through page-visible execution traces and interactions. In *CCS*, 2024.

[5] Anupama Aggarwal, Bimal Viswanath, Liang Zhang, Saravana Kumar, Ayush Shah, and Ponnurangam Kumaraguru. I spy with my little eye: Analysis and detection of spying browser extensions. In *IEEE Euro S&P*, 2018.

[6] Amazon. Conditions of Use, 2022. URL https://www.amazon.com/gp/help/customer/display.html/ref=ap_desktop_footer_cou?ie=UTF8&nodeId=508088.

[7] Amazon. Amazon search endpoint, 2025. URL https://www.amazon.com/s?k=.

[8] Helen Aquino and José Márcio de Castro. Knowledge internalization as a measure of results for organizational knowledge transfer: the proposition of a theoretical framework. In *Tourism & Management Studies*, 2017.

[9] Adam Barth, Adrienne Porter Felt, Prateek Saxena, and Aaron Boodman. Protecting browsers from extension vulnerabilities. In *NDSS*, 2010.

[10] Nicole Bergen and Ronald Labonté. "everything is perfect, and we have no problems": detecting and limiting social desirability bias in qualitative research. In *Qualitative health research*. Sage Publications, 2020.

[11] Nicola Bücker. Kodieren – aber wie? varianten der grounded-theory-methodologie und der qualitativen inhaltsanalyse im vergleich. In *Forum: Qualitative Sozialforschung / Forum: Qualitative Social Research*, 2020.

[12] Duc Bui, Brian Tang, and Kang G Shin. Detection of inconsistencies in privacy practices of browser extensions. In *IEEE Euro S&P*, 2023.

[13] Giovanni Capoccia and R. Daniel Kelemen. The study of critical junctures: Theory, narrative, and counterfactuals in historical institutionalism. In *World Politics*, 2007.

[14] Nicholas Carlini, Adrienne Porter Felt, and David Wagner. An evaluation of the google chrome extension security architecture. In *USENIX Security*, 2012.

[15] caseywatts. Making bookmarklet, 2017. URL https://gist.github.com/caseywatts/c0cec1f89ccdb8b469b1.

[16] Kathy C. Charmaz. Constructing grounded theory: A practical guide through qualitative analysis. In *Sage*, 2006.

[17] Quan Chen and Alexandros Kapravelos. Mystique: Uncovering information leakage from browser extensions. In *CCS*, 2018.

[18] Chrome Extensions. Stay secure, 2025. URL https://developer.chrome.com/docs/extensions/develop/security-privacy/stay-secure.

[19] Chrome For Developers. Manifest file format, 2012. URL https://developer.chrome.com/docs/extensions/reference/manifest.

[20] Chrome For Developers. Publish in the chrome web store, 2014. URL https://developer.chrome.com/docs/webstore/publish.

[21] Chrome For Developers. Follow-up on rejections and takedowns, 2021. URL https://developer.chrome.com/docs/webstore/check-review#follow-up_on_rejections_and_takedowns.

[22] Chrome For Developers. Chrome Web Store review process, 2021. URL https://developer.chrome.com/docs/webstore/review-process.

[23] Chrome For Developers. Can extensions use web storage APIs?, 2024. URL https://developer.chrome.com/docs/extensions/reference/api/storage#can_extensions_use_web_storage_apis.

[24] Chrome For Developers. Extensions / Manifest V3, 2025. URL https://developer.chrome.com/docs/extensions/develop/migrate/what-is-mv3.

[25] Chrome For Developers. Get started, 2025. URL https://developer.chrome.com/docs/extensions/get-started.

[26] Chrome Webstore. Chrome extensions sitemap, 2025. URL https://chrome.google.com/webstore/sitemap.

[27] David Collier and Gerardo L Munck. Building blocks and methodological challenges: A framework for studying critical junctures. In *QMMR*, 2017.

[28] Thad Dunning. Contingency and determinism in research on critical junctures: Avoiding the" inevitability framework". In *QMMR*, 2017.

[29] Benjamin Eriksson, Pablo Picazo-Sanchez, and Andrei Sabelfeld. Hardening the security analysis of browser extensions. In *ACM SAC*, 2022.

[30] Aurore Fass, Dolière Francis Somé, Michael Backes, and Ben Stock. Doublex: Statically detecting vulnerable data flows in browser extensions at scale. In *CCS*, 2021.

[31] Felix Fischer, Konstantin Böttinger, Huang Xiao, Christian Stransky, Yasemin Acar, Michael Backes, and Sascha Fahl. Stack overflow considered harmful? the impact of copy&paste on android application security. In *IEEE S&P*, 2017.

[32] Ghostery. Chrome's manifest v3 - improving privacy?, 2025. URL https://www.ghostery.com/blog/manifest-v3-privacy.

[33] GitHub Security Lab. Attacking browser extensions, 2024. URL https://github.blog/security/vulnerability-research/attacking-browser-extensions/.

[34] Barney G. Glaser and Anselm L. Strauss. The discovery of grounded theory: Strategies for qualitative research. In *Aldine Publishing Company*, 1967.

[35] Google. Generative AI in Search: Let Google do the searching for you, 2024. URL https://blog.google/products/search/generative-ai-google-search-may-2024/.

[36] Google Chrome. Chrome web store, 2025. URL https://chromewebstore.google.com/.

[37] Peter Leo Gorski, Luigi Lo Iacono, Dominik Wermke, Christian Stransky, Sebastian Möller, Yasemin Acar, and Sascha Fahl. Developers deserve security warnings, too: On the effect of integrated security advice on cryptographic api misuse. In *SOUPS*, 2018.

[38] Jaber F. Gubrium, James A. Holstein, Amir B. Marvasti, and Karyn D. McKinney. The sage handbook of interview research: The complexity of the craft. In *Sage*, 2012.

[39] Hacker News. Chrome extensions are tampering with security headers, 2021. URL https://news.ycombinator.com/item?id=27284224.

[40] Hacker News. Chrome users beware: Manifest v3 is deceitful and threatening, 2023. URL https://news.ycombinator.com/item?id=38301801.

[41] Daniel Hausknecht, Jonas Magazinius, and Andrei Sabelfeld. May i?-content security policy endorsement for browser extensions. In *DIMVA*, 2015.

[42] Sheryl Hsu, Manda Tran, and Aurore Fass. What is in the chrome web store? In *AsiaCCS*, 2024.

[43] Nav Jagpal, Eric Dingle, Jean-Philippe Gravel, Panayiotis Mavrommatis, Niels Provos, Moheeb Abu Rajab, and Kurt Thomas. Trends and lessons from three years fighting malicious extensions. In *USENIX Security*, 2015.

[44] Alexandros Kapravelos, Chris Grier, Neha Chachra, Christopher Kruegel, Giovanni Vigna, and Vern Paxson. Hulk: Eliciting malicious behavior in browser extensions. In *USENIX Security*, 2014.

[45] Soroush Karami, Panagiotis Ilia, Konstantinos Solomos, and Jason Polakis. Carnus: Exploring the privacy threats of browser extension fingerprinting. In *NDSS*, 2020.

[46] Soroush Karami, Faezeh Kalantari, Mehrnoosh Zaeifi, Xavier J Maso, Erik Trickel, Panagiotis Ilia, Yan Shoshitaishvili, Adam Doupé, and Jason Polakis. Unleash the simulacrum: Shifting browser realities for robust extension-fingerprinting prevention. In *USENIX Security*, 2022.

[47] Ankit Kariryaa, Gian-Luca Savino, Carolin Stellmacher, and Johannes Schöning. Understanding users' knowledge about the privacy and security of browser extensions. In *SOUPS*, 2021.

[48] Gerhard Kleining. Umriss zu einer methodologie qualitativer sozialforschung. In *Kölner Zeitschrift für Soziologie und Sozialpsychologie*, 1982.

[49] Hubert Knoblauch, René Tuma, and Bernt Schnettler. Videography: Introduction to interpretive videoanalysis of social situations. In *Peter Lang*, 2014.

[50] Tatiana Kostova and Kendall Roth. Adoption of an organizational practice by subsidiaries of multinational corporations: Institutional and relational effects. In *The Academy of Management Journal*, 2002.

[51] Pierre Laperdrix, Oleksii Starov, Quan Chen, Alexandros Kapravelos, and Nick Nikiforakis. Fingerprinting

in style: Detecting browser extensions via injected style sheets. In *USENIX Security*, 2021.

[52] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. *Research methods in human-computer interaction*. Morgan Kaufmann, 2017.

[53] Byoungyoung Lee and Young Min Kim. Extending a hand to attackers: Browser privilege escalation attacks via extensions. In *USENIX Security*, 2023.

[54] Heidi M Levitt. Qualitative generalization, not to the population but to the phenomenon: Reconceptualizing variation in qualitative research. In *Qualitative psychology*, 2021.

[55] Philipp Mayring. Qualitative content analysis. In *Forum: Qualitative Sozialforschung / Forum: Qualitative Social Research*, 2000.

[56] Microsoft. Activex controls, 2021. URL https://learn.microsoft.com/en-us/cpp/mfc/activex-controls.

[57] Mozilla Developer Network. storage, 2024. URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/storage.

[58] Mozilla Developer Networks. Client-side storage, 2025. URL https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Client-side_APIs/Client-side_storage.

[59] Mozilla Developer Networks. Content security policy (csp), 2025. URL https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP.

[60] Mozilla Developer Networks. Example extensions, 2025. URL https://developer.mozilla.org/en-US/docs/Mozilla/Add-ons/WebExtensions/API/declarativeNetRequest#example_extensions.

[61] Mozilla Developer Networks. X-Frame-Options, 2025. URL https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options.

[62] Mozilla Developer Networks. X-frame-options, 2025. URL https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/X-Frame-Options.

[63] Mozilla Firefox. What does review rejection mean to users?, 2019. URL https://extensionworkshop.com/documentation/publish/what-does-review-rejection-mean-to-users/.

[64] Mozilla Firefox. Security over choice, 2021. URL https://extensionworkshop.com/documentation/publish/add-ons-blocking-process/#security-over-choice.

[65] Mozilla Firefox. Submitting an add-on, 2024. URL https://extensionworkshop.com/documentation/publish/submitting-an-add-on/.

[66] Mozilla Firefox. Firefox browser add-ons, 2025. URL https://addons.mozilla.org/en-US/firefox/extensions/.

[67] Mozilla Firefox. Firefox Add-ons, 2025. URL https://addons.mozilla.org/api/v5/addons/search/?app=firefox&type=extension.

[68] mozilla wiki. Add-ons/reviewers/guide/review decision, 2024. URL https://wiki.mozilla.org/Add-ons/Reviewers/Guide/Review_Decision.

[69] Alena Naiakshina, Anastasia Danilova, Eva Gerlitz, Emanuel Von Zezschwitz, and Matthew Smith. "if you want, i can store the encrypted password" a password-storage field study with freelance developers. In *ACM CHI*, 2019.

[70] Alexandra Nisenoff, Arthur Borem, Madison Pickering, Grant Nakanishi, Maya Thumpasery, and Blase Ur. Defining "broken": User experiences and remediation tactics when ad-blocking or tracking-protection tools break a website's user experience. In *USENIX Security*, 2023.

[71] OWASP Cheat Sheet Series. Http security response headers cheat sheet, 2025. URL https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Headers_Cheat_Sheet.html.

[72] OWASP Cheat Sheet Series Team. Browser Extension Security Vulnerabilities, 2025. URL https://cheatsheetseries.owasp.org/cheatsheets/Browser_Extension_Vulnerabilities_Cheat_Sheet.html.

[73] Raffaello Perrotta and Feng Hao. Botnet in the browser: Understanding threats caused by malicious browser extensions. In *IEEE S&P*, 2018.

[74] Ritik Roongta and Rachel Greenstadt. From user insights to actionable metrics: A user-focused evaluation of privacy-preserving browser extensions. In *AsiaCCS*, 2024.

[75] Sebastian Roth, Lea Gröber, Michael Backes, Katharina Krombholz, and Ben Stock. 12 angry developers-a qualitative study on developers' struggles with csp. In *CCS*, 2021.

[76] Sebastian Roth, Lea Gröber, Philipp Baus, Katharina Krombholz, and Ben Stock. Trust me if you can – how usable is trusted types in practice? In *USENIX Security*, 2024.

[77] Margrit Schreier. Qualitative content analysis. In *Sage*, 2014.

[78] Raphael Serafini, Stefan Albert Horstmann, and Alena Naiakshina. Engaging company developers in security research studies: A comprehensive literature review and quantitative survey. In *USENIX Security*, 2024.

[79] Seraphic Cyber Security. Malicious browser extensions are on the rise, 2025. URL https://seraphicsecurity.com/resources/blog/malicious-browser-extensions-are-on-the-rise/.

[80] Alexander Sjösten, Steven Van Acker, and Andrei Sabelfeld. Discovering browser extensions via web accessible resources. In *CODASPY*, 2017.

[81] Alexander Sjösten, Steven Van Acker, Pablo Picazo-Sanchez, and Andrei Sabelfeld. Latex gloves: Protecting browser extensions from probing and revelation attacks. In *NDSS*, 2019.

[82] Konstantinos Solomos, Panagiotis Ilia, Soroush Karami, Nick Nikiforakis, and Jason Polakis. The dangers of human touch: fingerprinting browser extensions through user actions. In *USENIX Security*, 2022.

[83] Konstantinos Solomos, Panagiotis Ilia, Nick Nikiforakis, and Jason Polakis. Escaping the confines of time: Continuous browser extension fingerprinting through ephemeral modifications. In *CCS*, 2022.

[84] Konstantinos Solomos, Nick Nikiforakis, and Jason Polakis. Harnessing multiplicity: Granular browser extension fingerprinting through user configurations. In *ACSAC*, 2024.

[85] Dolière Francis Somé. Empoweb: empowering web applications with browser extensions. In *IEEE S&P*, 2019.

[86] John Sparrow. Knowledge in organizations: access to thinking at work. In *Sage*, 1998.

[87] Stack Overflow. ManifestV3 example using declarativeNetRequest, 2021. URL https://stackoverflow.com/a/69177790.

[88] Oleksii Starov and Nick Nikiforakis. Extended tracking powers: Measuring the privacy diffusion enabled by browser extensions. In *WWW*, 2017.

[89] Oleksii Starov and Nick Nikiforakis. Xhound: Quantifying the fingerprintability of browser extensions. In *IEEE S&P*, 2017.

[90] Oleksii Starov, Pierre Laperdrix, Alexandros Kapravelos, and Nick Nikiforakis. Unnecessarily identifiable: Quantifying the fingerprintability of browser extensions due to bloat. In *WWW*, 2019.

[91] Anselm L. Strauss and Juliet M. Corbin. Basics of qualitative research: Techniques and procedures for developing grounded theory. In *Sage Publications*, 2008.

[92] supermaven. supermaven, 2025. URL https://supermaven.com/.

[93] TechRadar. How chrome's manifest v3 will change the game for ad blockers, 2024. URL https://www.techradar.com/pro/how-chromes-manifest-v3-will-change-the-game-for-ad-blockers.

[94] ToS;DR. Amazon, 2025. URL https://tosdr.org/en/service/190.

[95] Erik Trickel, Oleksii Starov, Alexandros Kapravelos, Nick Nikiforakis, and Adam Doupé. Everyone is different: Client-side diversification for defending against extension fingerprinting. In *USENIX Security*, 2019.

[96] Ming-Ten Tsai and Kuo-Wei Lee. A study of knowledge internalization: From the perspective of learning cycle theory. In *Journal of Knowledge Management*, 2006.

[97] W3C/WebExtensions. webextensions, 2024. URL https://github.com/w3c/webextensions.

[98] David Waldner. Qualitative causal inference and critical junctures. In *Critical Junctures and Historical Legacies: Insights and Methods for Comparative Social Science*, 2022.

[99] Wikipedia. Computer-assisted qualitative data analysis software, 2024. URL https://en.wikipedia.org/wiki/Computer-assisted_qualitative_data_analysis_software.

[100] Qinge Xie, Manoj Vignesh Kasi Murali, Paul Pearce, and Frank Li. Arcanum: Detecting and evaluating the privacy risks of browser extensions on web pages and web content. In *USENIX Security*, 2024.

[101] Jianjia Yu, Song Li, Junmin Zhu, and Yinzhi Cao. CoCo: Efficient Browser Extension Vulnerability Detection via Coverage-guided, Concurrent Abstract Interpretation. In *CCS*, 2023.

PID	Country	Age	Gender	CompSci BG	ExtDev YoE	WebDev BG	#Extensions Developed	Chrome Install Counts (max, min, mean, stddev)	Firefox Install Counts (max, min, mean, stddev)
P01	Ukraine	32	♂	None	1	🕗	1	272 / 272 / 272.0 / 0.0	41 / 41 / 41.0 / 0.0
P02	India	23	♂	UG	1	🕗	1	63 / 63 / 63.0 / 0.0	-
P03	Canada	19	♂	**UG***	5	🕗	**8**	**10,000 / 22 / 4,035.0 / 4,137.7**	-
P04	Canada	31	♂	None	**9**	✗	1	**10,000 / 10,000 / 10,000.0 / 0.0**	-
P05	U.S.A.	26	♂	UG	1.5	✓	1	3 / 3 / 3.0 / 0.0	-
P06	India	31	♂	UG	**9**	🕗	2	640 / 75 / 357.5 / 282.8	-
P07	Norway	25	♂	None	5	🕗	3	3,000 / 473 / 1,375.3 / 1,091.7	45 / 45 / 45.0 / 0.0
P08	Switzerland	34	♂	UG	1	🕗	1	413 / 413 / 413.0 / 0.0	-
P09	Bangladesh	28	♂	**DNF**	3	🕗	1	87 / 87 / 87.0 / 0.0	-
P10	India	29	♀	UG	2	🕗	1	-	7 / 7 / 7.0 / 0.0
P11	U.S.A.	–	♂	UG	5	🕗	2	-	2,682 / 317 / 1,499.5 / 1,186.5
P12	Italy	47	♂	UG	3	🕗	1	-	1,837 / 1,837 / 1,837.0 / 0.0
P13	Germany	–	♂	UG	2	🕗	1	744 / 744 / 744.0 / 0.0	136 / 136 / 136.0 / 0.0
P14	Japan	29	♂	UG	3	🕗	2	2,000 / 1,000 / 1,500.0 / 500.0	-
P15	Taiwan	43	♂	DNF	3	🕗	1	3,000 / 3,000 / 3,000.0 / 0.0	-
P16	Australia	27	♂	UG	4	🕗	1	10,000 / 10,000 / 10,000.0 / 0.0	876 / 876 / 876.0 / 0.0
P17	Israel	27	♂	**DNF**	**7**	✓	7	100,000 / 1,000 / 17,714.3 / 33,356.3	5,945 / 40 / 1,447.4 / 2,184.5
P18	Lithuania	29	♂	UG	5	🕗	1	**5,000,000 / 5,000,000 / 5,000,000.0 / 0.0**	**203,601 / 203,601 / 203,601.0 / 0.0**
P19	Germany	56	♂	**PG**	**20**	🕗	3	**800,000 / 1,000 / 300,500.0 / 399,500.0**	**10,000 / 4,766 / 7,383.0 / 3,617.0**
P20	India	40	♀	**PG**	3	🕗	1	10,000 / 10,000 / 10,000.0 / 0.0	-
P21	New Zealand	57	♂	UG	9	🕗	1	**70,000 / 70,000 / 70,000.0 / 0.0**	**9,077 / 9,077 / 9,077.0 / 0.0**

Table 2: Demographics of our interview participants. BG: *Background*, UG: *Studied or *currently studying Computer Science in Under Graduate*, PG: *Post Graduate*, DNF: *Did not finish/Dropped out of college*, YoE: *Year(s) of experience*, 🕗: *Currently working in Web Development*, ✓: *Prior Web development experience*, ✗: *No Web development experience*, ♀: *Female*, ♂: *Male*

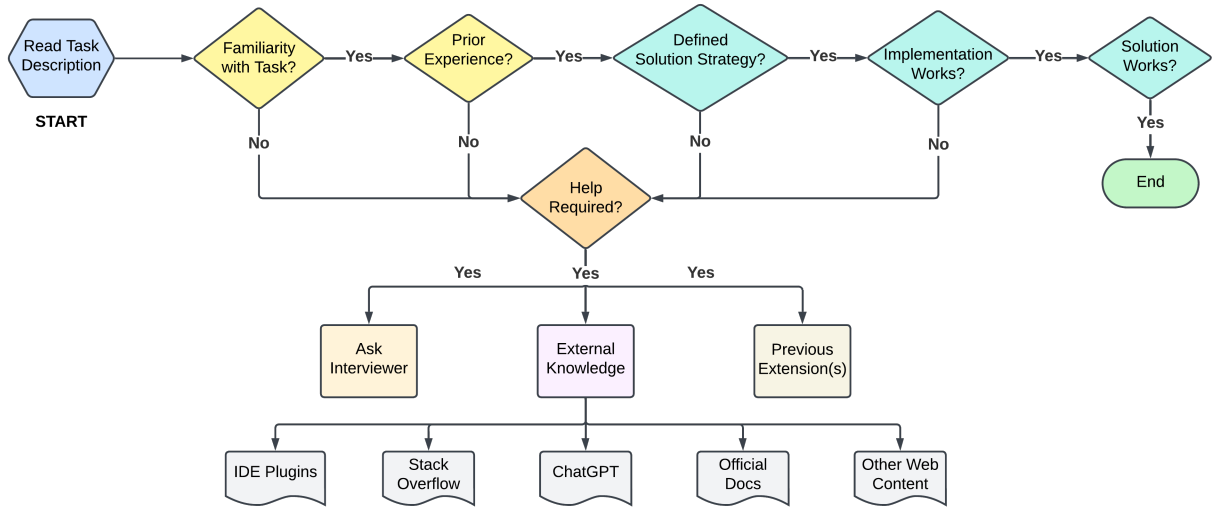

Figure 3: Generic interaction and workflow diagram demonstrating the participants' sequential actions during the coding tasks.