VDPCollect: Vulnerability Disclosure Programs as a Complement to Web Security Measurements

Philip Decker
Saarland University
decker.philip.3@gmail.com

Florian Hantke
CISPA Helmholtz Center for Information Security
florian.hantke@cispa.de

ABSTRACT

Since the first vulnerability disclosure program (VDP) in 1963, these programs have recently gained more attention throughout the industry, allowing external people to search for and report vulnerabilities. However, current research in this direction primarily conducts surveys with stakeholders or extracts insights into management.

With this work, we shift the focus to the technical side of VDPs and investigate the opportunities for ethical and legal vulnerability research using a VDP dataset. We therefore created a dataset of 3462 websites listed within VDPs along with their policies, and compared them against a set of 9423 popular websites from the CrUX list to gain insights into their usability for web security research.

Our measurements reveal that websites participating in VDPs demonstrate greater security practices and fewer vulnerabilities. Nearly twice as many CrUX websites include outdated libraries with known vulnerabilities. Further, we found and validated more client-side XSS attacks on CrUX domains (0.49%) than on VDP-listed domains (0.16%), and observed insecure CSP use 5% more often in CrUX. While security appears to be improved within such programs, their policies allow researchers to test areas that are otherwise difficult to assess in large-scale, real-world environments, such as server-side vulnerabilities. Our results highlight how VDPs could enhance research, namely offering valuable insights into web security that can serve as a lower bound estimate for the overall Web. This suggests that VDPs could also provide an indirect entry point to study server-side security postures at scale, a hypothesis we outline for future work.

CCS CONCEPTS

• **Security and privacy → Web application security**.

KEYWORDS

Web measurement, Server-side research, Ethics, Law

ACM Reference Format:
Philip Decker and Florian Hantke. 2026. VDPCollect: Vulnerability Disclosure Programs as a Complement to Web Security Measurements. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '26), June 1–5, 2026, Bangalore, India.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3779208.3785373

ASIA CCS '26, June 1–5, 2026, Bangalore, India

ACM ISBN 979-8-4007-2356-8/26/06
https://doi.org/10.1145/3779208.3785373

1 INTRODUCTION

"Get a bug if you find a bug" — This title represents the beginning of bug bounty programs in the digital area. It was the title of the first-ever bug bounty program, which was introduced by Hunter & Ready in 1983 [16]. The prize was a Volkswagen Beetle, also known as *"bug"*, for everyone who could demonstrate a bug in their VRTX real-time operation system. They were the first to offer hackers not only financial rewards for their findings, but also a structured and legal way to report, an agreement today often referred to as *safe harbor*. Since 1983, Bug Bounty and Vulnerability Disclosure programs have seen a rapid growth with an expected market size of USD 5.74 billion by 2033 [9]. Today, many large corporations run such structured programs, often managed by platforms such as HackerOne and Bugcrowd, and the number of programs is growing [25, 26, 75]. Even many government bodies, with support from organizations like CISA, have started programs to improve their cybersecurity [14]. In short, the ecosystem around bug bounty and disclosure programs has grown into a major business.

Yet, despite their immense growth and popularity, these programs have not gained traction widely within academic web security research. While occasionally papers studied the effectiveness, development and key factors of bug bounty programs [1, 5, 23, 75, 76], the web security measurement community continues to rely primarily on widely popular websites such as those from the CrUX or Tranco lists. Conducting measurements on live systems, however, raises ethical and legal challenges [29, 38], particularly in the case of critical experiments such as server-side vulnerability assessments. Legal uncertainties, notably, have produced chilling effects, discouraging researchers from pursuing such studies [24]. As ethical and legal considerations gain increasing importance in our publications [41] and call for papers [73], it becomes essential to explore alternative approaches for conducting web measurements.

To address and overcome these challenges, we explore whether Vulnerability Disclosure Programs (VDPs) can enable empirical research on server-side security issues at scale. Unlike traditional web measurement datasets, VDPs provide *explicit authorization*, *scope definitions*, and *safe-harbor guarantees* for active testing that are otherwise ethically or legally challenging to study. We believe, these programs offer researchers the rare opportunity to test real-world applications within an authorized scope, including critical experiments such as server-side assessments, without facing the same ethical and legal uncertainties. Building on this perspective, we explore *How can researchers leverage these programs for broader empirical studies?* and *What trade-offs do researchers face with these programs compared to traditional web measurement datasets?*

Motivated by these questions, we collected data from bug bounty and disclosure programs and conducted security measurements on

3462 in-scope domains from major bug bounty and disclosure programs, comparing their security practices to 9423 domains from the CrUX list. Our measurements cover HTTP security headers, cookie and their attributes, third-party JavaScript inclusions, TLS configurations and client-side XSS vulnerabilities, which we manually validated. This approach allows us to assess whether VDP participation correlates with improved security and to explore the suitability of VDP-listed sites as an alternative dataset for web security research. While our measurements in this paper focus for good legal and ethical reasons only on client-side aspects, our analysis of VDP rules and scopes suggests that such datasets offer a promising entry point for future studies of server-side vulnerabilities.

In short, this work makes the following key contributions:

- An open-source *python* framework collecting information of bug bounty and disclosure programs, including scope definitions and rules from *Bugcrowd*, *HackerOne*, *Intigriti* and *YesWeHack*, resulting in a public dataset [17].
- Collection and analysis of legal and technical requirements defined in program rules to qualify for *safe-harbor*, ensuring compliance in our experiments and providing guidance for future research.
- A large-scale security measurement of 11 734 webpages from these programs and 81 702 from CrUX, comparing multiple client-side security aspects.
- An evaluation of the benefits and trade-offs of VDP-based datasets, showing that their scope definitions and safe-harbor guarantees enable ethical large-scale measurement research on server-side security compared to popularity-based lists.

Our pipeline VDPCollect and the dataset are open source [17].

2 BACKGROUND & RELATED WORK

Before providing details of our research, we define key terms, review related work, and outline the client-side security metrics used.

2.1 Vulnerability Disclosure Programs

Starting with the central point of our work, we define the term Vulnerability Disclosure Programs (VDP). There exist multiple types of VDPs, those rewarding an accepted vulnerability with a bounty are called Bug Bounty (BB). Programs offering no monetary compensation, but only acting as a platform for coordinated disclosure, are Vulnerability Disclosure (VD). For our work, we summarize both types under the term Vulnerability Disclosure Programs (VDP), since both follow the same concept and offer legal protection when complying to the rules. Despite this advantage, research on the potential of VDP for the scientific community remains limited. Existing studies primarily focus on statistical evaluations of the development [9, 75], investigations to gain a better understanding of the program rules [76], interviews and surveys to identify key success factors [1] and assessments of the costs and benefits for the industry [5, 23, 75]. While these papers present promising results of a growing number of VDPs across different providers and industry sections [9, 75], researchers have not yet considered VDP-listed websites as a dataset for conducting web security measurements. We still miss an assessment of the usability, benefits, limitations and potential biases that measurements using such a dataset would introduce compared to relying on the broader Web.

2.2 Security Aspects

Our analysis focuses on observable client-side security features. With the increasing web security awareness, modern browsers implement client-side protections to ensure the confidentiality, integrity, and safe execution of web content. Key features include HTTP response headers, cookie attributes, SSL/TLS certificates, and vulnerability assessments.

2.2.1 Client-Side XSS. One of the most common vulnerabilities on the Web is Cross-Site-Scripting (XSS) attacks. Defined initially by Microsoft engineers in 2000 [53], XSS refers to attacks where an adversary injects malicious JavaScript, HTML, or other fragments into a victim's browser, causing it to be executed in the victim's context. Over time, many people studied XSS, introducing a variety of subcategories such as server-side, persistent, blind, or client-side XSS [38, 56, 57, 68]. Despite ongoing research, XSS remains one of the most prominent web vulnerabilities, also appearing in the OWASP Top 10 [59]. Our focus is on client-side XSS, which occurs entirely in the browser-side due to insecure handling of user input in client-side code, without involving the server-side at all.

2.2.2 Security Headers. A fundamental mechanism for enforcing browser security policies against XSS and similar attacks is using security- and privacy-related HTTP headers. These headers have become essential tools for mitigating attacks, including clickjacking [12, 32], mixed-content [4, 47], or unauthorized cross-origin interactions [61].

One key HTTP security header is the *Content-Security-Policy* (CSP), which restricts the execution and inclusion of scripts and other content through directives such as *script-src* and *object-src*, blocking untrusted sources. To mitigate XSS, they can also deactivate dangerous functions such as *eval* and inline event handlers. To prevent eavesdropping and mixed-content issues, CSP supports directives like *upgrade-insecure-requests* and *block-all-mixed-content*. Additionally, the *frame-ancestors* directive further lets developers control which sites are allowed to embed their pages. [52]

The predecessor of the CSP directive *frame-ancestors* is the *X-Frame-Options* (XFO) header, which was designed to enable developers to restrict which websites are allowed to embed their pages in framing contexts such as <*iframe*>, <*frame*>, or <*object*> elements. While the CSP offers more fine-grained controls, the original header XFO is used similarly to prevent clickjacking and iframe-based attacks by it either to *SAMEORIGIN*, *ALLOW-FROM*, or *DENY*. [51]

Similar to the *upgrade-insecure-requests* enforcing the upgrading of HTTP connections to HTTPS, the *Strict-Transport-Security* (HSTS) header instructs browsers to access websites over HTTPS. The header contains multiple directives: *max-age* defining how long the browser should enforce the HTTPS upgrades in seconds, *includeSubDomains* instructing the browser to also upgrade requests to its subdomains, and the *preload* directive to ask for inclusions in HSTS preload lists, which then enables the browser to initiate the upgrade already for the first request towards the host. [50]

Irrespective of the protocol used to visit a website, during all requests, the browser adds information about the referring (previously visited) host in outgoing requests and navigation. This information can pose privacy risks. To mitigate this, browsers support the *Referrer-Policy* (RP) header, which allows developers to control

whether the full URL, only the origin, or no referrer is sent when navigating between pages or making subresource requests. [49]

Beyond controlling the exposure of privacy-related information, another header enables the restriction of powerful and privacy-threatening browser features. This is archived by the *Permissions Policy* (PP), formerly known as Feature Policy, allowing developers to explicitly control the availability of powerful browser features and APIs. Developers can selectively turn on or off features such as geolocation, camera access, fullscreen mode, or clipboard operations through a set of policy directives. [48]

To address and mitigate cross-site leaks and related threats, headers such as the *Cross-Origin-Opener-Policy* (COOP), *Cross-Origin-Embedder-Policy* (COEP) and *Cross-Origin-Resource-Policy* (CORP) control the sharing of resources or window references across deviating origins. [43–45, 61]

2.2.3 Cookies. Besides HTTP headers, cookies are also a standard Web mechanism, used to store information on the client-side. Nowadays, cookies are used for various tasks, including session management, advertising, and tracking. Their intent and origin can often be inferred by analyzing their domain and name and comparing those against publicly available tracking lists [74].

Cookies are not secure by default, especially when used for sensitive tasks like authentication. To address this, modern browsers support security attributes that enhance cookie handling: the *Secure* flag ensures cookies are transmitted only over HTTPS, mitigating simple Man-in-the-Middle (MitM) attacks; the *HttpOnly* option blocks JavaScript access to cookies to prevent theft via XSS; and the *SameSite* attribute controls cross-origin cookie transmission. For *SameSite*, the option *Strict* limits cookies to same-origin requests only, *Lax* allows them on top-level navigations and *None* always sends the cookie along, but requires *Secure* to be set. [33, 46]

The usage and effectiveness of these attributes have been extensively studied in earlier work, which analyzed their deployment across websites and their impact on web security at a large scale [8, 36, 65, 69]. Previous work has also studied cookies' purpose and classified them, particularly focusing on third-party cookies for advertising and tracking purposes [22, 31, 74]. As browsers have increasingly started restricting third-party cookies [13, 35, 54, 55], there has been a shift toward using first-party cookies for similar tracking activities, often employing techniques such as *DNS CNAME cloaking* to bypass browser restrictions, which were also already analyzed in prior work. [18] Since the blocking of third-party cookies is only active under certain circumstance (e.g., browsing in incognito mode), still first and third-party cookies are used during normal browsing, which can be analyzed.

2.2.4 JavaScript Inclusions. Besides cookie usage, we also study third-party JavaScript inclusion, which web developers commonly use to extend website functionalities. Research observed that the average number of script inclusions increased steadily from the early 2000s until around 2016, after which it has stabilized at a constant level [28, 57, 69].

Each third-party script introduces security risks, as vulnerable or outdated third-party code can expose websites to attacks. Prior work has shown that reliance on external scripts often leads to vulnerabilities due to poor update practices and insecure code within the included sources [42, 56, 57].

Provider	#Programs	#LinkedIn	Location	Focus
HackerOne	460	314k	USA	-
Bugcrowd	201	124k	USA	-
YesWeHack	74	47k	France	-
Intigriti	125	34k	Belgium	-
Immunefi	316	9k	Singapore	web3/crypto
HackenProof	237	4k	Estonia/Ukraine	web3/crypto
GoBugFree	85	2.6k	Switzerland	-
Huntr	326	2k	UK	ML/AI
VulnScope	162	<1k	Chile	-
BugRap	52	-	Singapore	web3

Table 1: Identified VDP providers

These scripts are not always functional; many can serve purposes such as tracking, fingerprinting, or advertisement. Their classification, based on their origin, path, or by utilizing specific classification frameworks, has been studied by previous research, which demonstrates the extensive use of such privacy-invasive scripts [22, 30, 67].

2.2.5 Transport Layer Security. Another client-side observable security mechanism is the Transport Layer Security (TLS) implemented by the browser based on instructions from the websites. It ensures the confidentiality and authenticity of web communication. By enforcing HTTPS over SSL/TLS, modern browsers protect the connection from interception and Man-in-the-Middle (MitM) attacks during transmission. TLS relies on digital certificates presented by the websites, providing all necessary cryptographic parameters for the encryption and validates through a chain of trust.

Research has shown that vulnerabilities and configuration weaknesses within the process can compromise these security guarantees. Famous vulnerabilities significantly threatening the confidentiality and authenticity, are *Heartbleed* [71], *poodle* [6], or MitM attacks [10], which have been extensively demonstrated and studied in earlier work.

3 VULNERABILITY DISCLOSURE SITES

To enable web security measurements against VDP websites, we first have to create a comprehensive dataset. This section contains the methodology for creating the dataset, along with other important information extractable from the individual programs.

3.1 Collection of Programs

To ensure the data's completeness, correctness, and freshness, we collected programs ourselves rather than relying on potentially outdated existing lists. We identified potential candidates by looking for providers with at least 50 programs, open registration, and allowing us to check programs without a prior application/reviewing process. Based on a public list [20] and manual web searches, we compiled a list of ten providers (see Table 1). To inform our final decision, we reviewed each provider's number of public programs, LinkedIn follower count, headquarters location, and provider's specialization (if applicable), e.g., Web3 or cryptocurrencies.

We chose to analyze Bugcrowd, HackerOne, Intigriti, and YesWeHack based on their great overall coverage. Besides being among the largest and most popular providers, we chose those due to their geographic location. The providers Bugcrowd and HackerOne are

located in the US, and their customer base is similarly concentrated there. To ensure a broader picture, we included Intigriti, a Belgian provider and YesWeHack from France. Another key factor for our selection was the providers' broad scope, including companies from all fields of work (in contrast to other major providers like HackenProof and Immunefi, who focus only on specific fields such as crypto). This selection enables us to gain a complete overview, including VDPs from different areas, similar to the CrUX list variety.

To collect the publicly available VDPs, we implemented a crawler for each provider into our pipeline VDPCollect, each logging into a fresh account and gathering all public programs (Figure 2 part one). The collected data includes the program title, company name, rules or descriptions, scope details and any additional information or prerequisites, such as specific HTTP headers or request rate limits, that should be considered during testing.

On completion of our self-implemented provider crawls, their raw data is stored in a database for further processing. The first step is extracting actual webpages from the raw scope entries collected for the VDPs. This extraction is required since the collected scope entries can describe any type of scope, such as webpages, IPs, mobile applications, or even physical products. For the extraction, we relied on the *RegEx* library of *python3* with an expression matching valid URLs of different writing and structuring.

3.2 Coverage

With our initial crawls, we collected 11 975 in-scope entries of 843 scraped VDPs, and after applying our extraction process, we compiled a list of 9919 scope matching our website scheme. Having a list of websites from VDPs, we compare their inclusion in traditional web measurement datasets. We chose to compare our dataset against the CrUX list, as it has been recognized as the most accurate list of popular websites [66]. We therefore split the CrUX list into the smaller buckets and checked the number of domain names, that occur within any VDP we collected. As presented in Figure 1, the percentage of continuously decreases by broadening the limit. While within the top 1k entries, almost one fifth of the websites have a match within the VDP dataset, less than 5% are reached within the top 50k entries of CrUX. This already indicates that higher-ranked websites are more likely to have a VDP. However, we employed optimistic matching to obtain these percentages, as we did not distinguish between different suffixes and relied solely on the domain name. A case illustrating this optimism during matching is Telenor. While 'telenor.se' (Sweden) is part of a VDP, 'telenor.com' lacks a corresponding VDP, despite both domains belonging to the same company group and sharing the domain name [72].

3.3 Rules

In addition to the websites listed as in-scope in the VDPs, we also collect each program's rules and descriptions to ensure ethical and legal compliance. These include accepted and excluded vulnerability types, required request headers and request rate limits. Adhering to these rules is essential, as violations may result in the loss of safe harbor protections.

Due to the majority of rules being mentioned within the free text field collected as rules and descriptions, we face the challenges of automated natural language processing. None of the providers uses

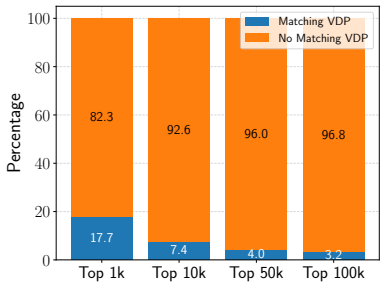


Figure 1: Overlap of VDP and CrUX domain names

Rank	In-Scope Category	#	Out-of-Scope Category	#
1	Access Control / Authentication	386	Misc	694
2	Misc	345	Information Disclosure	523
3	Remote Code Execution	316	Denial of Service	484
4	Cross Site Scripting	253	3rd Party Vulnerabilities	425
5	Information Disclosure	215	Social Engineering	422
6	SQL Injections	127	Automated Vulnerability Scanner	372
7	CSRF	122	Clickjacking	336
8	SSRF	73	Security Best Practices	312
9	Open Redirects	60	Content Spoofing / Injection	309
10	Configuration Issues	47	Configuration Issues	294

Table 2: Top 10 in-scope and out-of-scope vulnerability categories across programs

standardized terms for vulnerability types, requests and headers, leading to different descriptions of the same vulnerability type, spelling errors and further inconsistency. Due to these challenges of natural language, we use the Large Language Model (LLM) *gpt-4o-mini* to extract this information in cases in which the providers do not offer specific fields. The individual prompts used for extraction can be found in our code artifacts.

We start by analysing the vulnerability types allowed to be tested by security individuals within the scope. While Intigriti and YesWeHack present those in a structured list, Bugcrowd and HackerOne offer only a free-text field for their programs. During initial manual validation, we observed that the automatic categorization of terms into vulnerability types did not work perfectly reliably. To improve the accuracy, we added a manual keyword-based classification to increase the validity of the resulting classification. By using this mapping, we additionally categorized around 250 (out of 2530) in-scope and around 2300 (out of 7941) out-of-scope vulnerability types, which were earlier in the fallback category *Misc*. To this end, we create a list of vulnerability categories mentioned for each program with rules specified (840 out of 843) and additionally summarize the most common terms. Table 2 shows the most prominent vulnerability categories across the programs. We can, for example, see that out of the 843 programs, 386 explicitly accept vulnerabilities related to *Access Control*. While this provides a general overview of commonly targeted and excluded areas, it does not capture finer distinctions within each category. For instance, *Information Disclosure* may include severe issues like bulk

personal data leaks, but exclude minor exposures such as software version details. These nuances cannot be reflected using the current high-level classification.

Next, we extracted the required header extensions that must be set during testing. This extraction leads to better results since header descriptions often follow the same structures and often only affect the *User-Agent*. In total, the LLM extracted 150 out of the 843 programs. We manually validated their correctness. In addition, for 74 programs, on the providers *Intigriti* and *YesWeHack*, these headers can be extracted directly via a specific field. To ensure a broader coverage we not only correlate the header extension to each scope of the program, but also to the domain-name. While these entries might lead to falsely set headers, we ensure that specific headers are always set, even for AJAX, XHR, or similar requests against other subdomains or domains, which might not be specified in the original scope. To ensure compliance with the header prerequisite, we accept the risk that the headers are sent to destinations that are not part of VDP. Via this methodology, we can cover one third of the domains of our dataset with an associated header. This aligns with our observation that around 40% of the programs from providers that explicitly allow extraction of these headers via the API demand the use of such a header.

As a final commonly stated precaution in program rules — explicitly supported by a dedicated field in *Intigriti* — we extracted and analyzed request rate limits. These are crucial to adhere to, as VDPs often prohibit *automated scanning* or *tools generating large network traffic*. Once again, we used *ChatGPT* to automatically extract rate limit information from rules and program descriptions. In total, we obtained rate limit data for around 22% (188) of all programs. While this number seems low, our manual investigation confirmed this finding and matches the proportion of rate limits among *Intigriti* programs for which we could extract this information without an LLM. We further manually sampled 25 programs from other providers and validated that the LLM assistant always extracted the right number. Given this information, we can now obtain insights into the expected limits of these 22% of all programs. The data (in more detail in Appendix A) suggests that a limit of 5 requests per second complies with 60% of; reducing it to 2 requests per second covers 80%. This distribution is likely biased towards lower limits, as programs with stricter limits are more inclined to state them to avoid disruptive testing.

> **Key Takeaways:** Restrictive rules are common in VDPs. Many programs require specific identifying headers and recommend a request limit – 2 per second for broad compliance. While server-side testing is generally allowed, inconsistent rule formats hinder deeper analysis, highlighting the need for standardization.

4 MEASURING SECURITY RELEVANT DIFFERENCES

After describing our list of VDPs, we now dive into the methodology of the security measurements. First, we carefully considered the previously discussed findings to ensure we adhere to the rules and act responsibly (more details in Section 6.3). We then created our dataset and initiated the measurement pipeline of VDPCollect as displayed in Figure 2.

4.1 Creating the Dataset

For our experiments, we base the measurements on two datasets, one for VDPs and one for popular websites. For the latter one, we rely on the well-known CrUX list of February 2025 [77]. From this list, we take the top 10k plus 500 randomly sampled domains from each of the less popular buckets, i.e., 50k, 100k, 200k, 500k.

For the VDP dataset, we use websites extracted by our VDPCollect crawlers (see Section 3). Since these entries also contain wildcard domains (e.g., **.example.com*), we start a subdomain discovery using the tool *Sublis3r*, which queries public sources such as DNS servers and Google. Newly discovered subdomains are immediately checked against the list of out-of-scope sites to ensure that only in-scope domains are used to enlarge our VDP dataset.

Afterwards, we have two lists of websites, one from VDPs and one from CrUX. We filter them both with the same method to decrease the number of invalid targets passed to the later measurements. We start with a DNS-based filtering. For each website in the lists, we try to resolve the domain via *dig* to an actual IP; otherwise, the target is not reachable and filtered out. From these filtered lists, we further exclude domains that only serve non-HTML content and domains that redirect to prevent redirecting outside of our scope. This filtering is implemented using *Playwright* to control a headless browser visiting each website.

Once these filtering steps are applied, we compiled lists of valid targets for the VDP and CrUX datasets. Since metrics like HTTP headers, cookies and vulnerabilities can vary across different subpages of the same websites, we extend our dataset by discovering additional subpages. To this end, we visit each website, extract links to subpages and follow them. On valid responses, we add up to ten valid subpages per original website. We limit the crawling depth to two and consider at most 50 subpages per site. The resulting datasets are now enriched with subpages, forming the basis for our subsequent measurements.

4.2 HTTP Headers

This collection is implemented by issuing a single HTTP request.. If we get a valid response — defined as a 2xx status code, no redirects and a response time under 30 seconds — we store all the observable headers as JSON in our database. In case of failure or an exception, we log the issue to allow later evaluation of the success rate and commonly observed problems during the collection.

In addition to inspecting HTTP headers, we parse the HTML content when a *CSP* header is missing, checking whether it is defined via a *meta* tag. Then, we automatically evaluate each CSP using *Google's CSP-Evaluator* and store the result in the database.

The database now contains all headers explained in Section 2. We now focus on the classification of the observable configurations as secure or insecure for use in our analysis.

4.2.1 Cross Origin Policies. We begin with Cross Origin Policies, namely *COEP*, *COOP* and *CORP*. We classify them as insecure if their directives are not restrictive, effectively deactivating the protection offered by the headers. These insecure directives are *unsafe-none* for *COEP* and *COOP* and *cross-origin* for CORP.

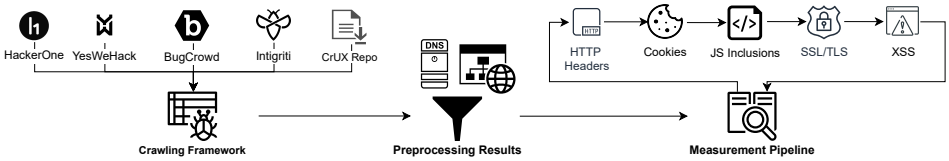


Figure 2: The pipeline of VDPCollect used for our measurements

4.2.2 Content Security Policy (CSP). After providing insights into the classification of headers related to cross-origin isolation, we now focus on the CSP, which offers a broader framework for client-side security controls. For our analysis of *CSP*, we mainly rely on the results of the tool *Google's CSP-Evaluator*, which we directly execute if we observe a *CSP* via header or meta tag. The results are individual ratings of the findings and recommendations to improve the different directives within the *CSP*.
Besides the investigation based on *Google's CSP-Evaluator*, we implemented our own checks for two more interesting use cases of *CSP* directives. Specifically, the first check examines the use of *block-all-mixed-content* and *upgrade-insecure-requests*, both of which aim to mitigate *mixed-content* issues and prevent insecure HTTP requests. The second interesting directive is the *frame-ancestors* controlling which websites are allowed to frame the original webpage. Used values, which we can classify as insecure, are too broadly defined wildcards. In our case, these include *, *http:*, *http://*, *http://**, *https:*, *https://* and *https://**. All of those offer no valid restriction to the framing and are therefore considered insecure by us. The only values we consider secure are: *self*, the own origin, the own domain, or the same full URL, allowing for same-origin framing and no third parties.

4.2.3 X-Frame Options (XFO). Complementary to CSP-based framing control, the legacy X-Frame-Options header is still in use. Though less flexible, it serves the same purpose of preventing clickjacking. The supported directives *sameorigin* and *deny* are classified as secure in our experiments, as they allow framing only by the same origin or block it entirely.

4.2.4 Permission Policy (PP). Beyond CSP, another important mechanism that governs client-side capabilities is the Permission Policy, which we investigate regarding its restrictiveness and common usage. The header contains definitions of browser functionalities, which can be completely disabled, such that no JavaScript can access it, limit access to scripts of specific origins, or allow access to any JavaScript. Our analysis investigates the most frequently set permissions within the header and whether and how restrictive they are.

4.2.5 Referrer Policy (RP). Privacy-related headers like the Referrer Policy are important in controlling what information is shared during navigation. The RP can be set to predefined values that limit what the *referrer* header includes. For our research, we are primarily interested in the insecure usage of the header. As insecure, we consider the absence of the header, the browser's default setting *no-referrer-when-downgrade* and *unsafe-url*. The first would always send the full URL in *referrer*, but only via HTTPS, while *unsafe-url* sends the full URL in any case. These settings can expose the full source URL to attackers, advertisers, or trackers, revealing the origin of previously visited pages.

4.2.6 Strict Transport Security (HSTS). Transport-level security is another common client-side security feature. The HSTS header tells the browser to access a site only via HTTPS. To work properly, a *max-age* must be set to define how long the browser should enforce this rule. We focus on two values: 0, which is insecure, as it disables HSTS immediately and 31 536 000, which is one year in seconds and classified as acceptable, as it is also the minimum required to use *preload*. The two other directives *preload* and *includeSubdomains* are treated as neither secure nor insecure, as they configure scope rather than directly affecting security.

4.3 Cookies

While we collect the HTTP headers with simple requests, we use an instrumented headless *Firefox* with *Playwright* to load dynamic resources for further collection. For each target URL, we create a fresh user profile with the *ISDCAC* extension preinstalled to bypass cookie banners [58]. These kinds of extensions are commonly used in measurement work [19, 37, 40] and help ensure that we can collect a sufficiently large set of cookies and resources for analysis. Further, we specify our proxy to set the program-specific headers. The script then navigates to the target, allowing up to 30 seconds for the DOM to load. If successful, cookies are extracted via *Playwright*'s built-in *cookies()* method and stored in our databases, along with exceptions and errors that occurred. We also map the cookies to their identified purpose for further analysis, leveraging the *Open-Cookie-Database*[34].

4.4 JavaScript Inclusions

Within the process of collecting cookies, we extract externally loaded script sources monitoring the network within *Playwright*. We flag requests as third-party scripts if the *Content-Type* includes *javascript* or the requested URL ends with *.js* and the origin differs from the visited site.

For further analysis, we classify third-party requests using the public filter lists *Disconnect.me* and *Easylists*. *Disconnect.me* maps domains to categories like advertisements, analytics and fingerprinting. Easylists includes two lists *Easylist* (for ads) and *EasyPrivacy* (for fingerprinting and tracking), which we match using the *adblockparser* library.

Furthermore, we analyze the JavaScript libraries used on each webpage for known vulnerabilities using the tool *reitre.js* [63]. While it cannot detect all issues, it checks for over 275 CVEs by identifying library versions through hashes and other mechanisms. Due to the version-based vulnerability detection, these only act as

indicator and a exploitation might not be feasible due to prerequisite (e.g. use of specific functions). Still, it acts as indicator for the overall security hygiene and potential vulnerabilities.

4.5 Transport Layer Security

Our measurements are not solely based on the code of webpages and headers; we also measure the applied transport layer security. The measurement and configuration extraction is done, within our *python3* pipeline VDPCollect, by using the tool *sslscan* [62]. The tool performs a broad set of protocol-level and certificate configuration tests. It offers information about TLS/SSL protocol versions, the corresponding encryption schemes, checks for known vulnerabilities such as *heartbleed* and identifies expired, not-yet-valid and self-signed certificates all compiled in a built-in rating system. Since we investigate certificates and settings shared across webpages of the same website, we only collect these metrics for each website once. The tool is executed with its default configuration, its XML outputs are converted to JSON and stored in our database.

4.6 Client-Side XSS

Besides configuration measurements acting as indicators of the implemented security — such as security headers or transport layer security — we also test for actual client-side vulnerabilities, specifically XSS. We use *Playwright* again to control *Foxhound*, an instrumented *Firefox* browser designed to automatically detect and validate XSS flows by tracing user-controllable input reaching sinks which might enable HTML or JavaScript injection. David Klein and Thomas Barber already used this instrumented browser for their original research investigating custom JavaScript Sanitizer Functions [39]. The browser found repeatedly use in earlier work investigating XSS and showing its capabilities [38, 60, 68]. To capture findings, we developed a custom extension and backend to extract taint flow data and store it in a dedicated database with relevant context for validation. Our script then simply visits the target webpages with a fresh browser profile, the proxy settings and the *ISDCAC* extension to bypass cookie banners, waits for the page load and waits an additional 10 seconds for the flow extraction.

Once the taint collection completed for all targets, we continue with the exploit generation. The framework we chose was initially created by Steffens et al. for their work analyzing XSS in the wild in 2019 [68] and parts of the latter changes were done by Rautenstrauch et al. for their investigations about login landscapes, published in 2024 [60]. The framework uses collected data flows to generate context-aware exploits, including context-specific encoding or escaping. We used `alert(document.domain)` as a proof-of-concept payload and stored the results for automated validation.

After generating exploits for each viable taint flow, we start validating the provided exploits. To do so, we use Firefox in a setup similar to the one for the collection. For exploits requiring the replacement of values within the local storage, cookies, or similar, we automatically set those before navigating to the URL. The script then navigates to the URL and checks whether the alert functionality is triggered and if the message matches the current value of *document.domain*, as specified in our exploit generation. If both conditions are met, the exploit is successfully validated and stored within a separate file for manual review, where we assess real-world exploitability and document the findings. As detailed in Section 6.3, confirmed vulnerabilities are then responsibly disclosed to the affected site operators.

	VDP	CrUX
Scopes Entries	11 975	12 000
Extracted (in-scope) URLs	9919	12 000
Wildcards	104	-
Discovered Subdomains	32 615	-
Discovered Out-Of-Scope Subdomains	4821	-
Valid DNS Entry	15 576	11 960
Valid Response	3462	9423
Subpage Discovery	11 734	81 702

Table 3: Overview preprocessing steps

5 RESULTS

With the insights of the VDP coverage and rules in Section 3, we now present the results of the subsequent crawl, our security measurements, and statistical chi-square tests.

5.1 General Crawling Statistics

Before starting the measurements, we applied our subdomain discovery and filtering to preprocess the collected targets. As presented in Table 3, we started initially with 11 975 entries out of the VDPs and 12 000 entries of the CrUX list. Through our subdomain discovery, we could signify increase the dataset, however the majority of those were removed immediately by the subsequent filtering. After DNS and HTTP filtering, 3462 VDP and 9423 CrUX entries remained. The higher removal rate for VDPs was expected, as they might not be updated as regularly as CrUX, and the VDP exclusive subdomain discovery often finds outdated or unreachable domains. Afterwards, the last step to conclude our dataset creation is the subpage discovery. Our subpage discovery traced the discovered subpages back to 1544 base URLs (44.6%), with an average of 7.14 new pages per active site. For CrUX, 84% of sites yielded new pages, totaling in an average of 9.13 per site. The lower number with the VDP dataset is partly due to more empty pages and a higher rate of duplicate links within the websites of the VDP dataset (27.41% vs. 19.36% in CruX). In total, the final datasets included a total of 11 734 webpages of VDPs and 81 702 webpages for the CrUX dataset.

5.2 HTTP Headers

Throughout our header crawling, we observed a similar failure rate for both datasets, namely 8.31% (6755) for targets of CrUX and 7.42% (1060) for the ones of VDPs. Despite our effort in preprocessing, the most common failures were due to redirects and the HTTP response code *403 Unauthorized*. We prevented redirects due to the risk of leaving the explicitly defined scope and the associated risk of potential legal and ethical issues when testing websites without prior permission. Throughout all the measurements, these repeatedly occur and are blocked by our framework.

Starting with the overall occurrence of headers presented in Table 4 we identified similar adoption rates throughout most of the headers, with only few headers showing larger differences.

	VDP - Domains	CrUX - Domains
COEP	0.24%	0.52%
CORP	0.96%	1.22%
COOP	2.70%	2.42%
CSP	23.31%	22.66%
PP	6.26%	5.22%
RP	20.22%	16.61%
HSTS	37.27%	39.93%
XFO	34.12%	39.54%

Table 4: Header usage overall

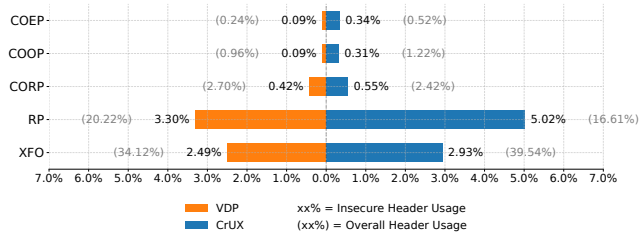


Figure 3: Prevalence of insecure headers

Next, we apply our earlier classifications for five of the headers (see Section 4.2), defining secure and insecure configurations, which results in the percentages presented in Figure 3. A significant difference (chi-square; $p \approx 0.00005 < 0.05$) was only identified in the insecure usage of the *referrer-policy*, for which within the VDP dataset 3.30% of 20.22% domains were flagged insecure, compared to 5.02% of 16.61% of the CrUX dataset. However, the CrUX dataset contains more adult-content, dating, and similar websites, which should not share the referrer insecurely to ensure privacy. In contrast, XFO ($p \approx 0.20$) and CORP ($p \approx 0.46$) show no statistically significant disparity in their insecure usage. For COOP and COEP, we do not report statistical tests, as their very low counts (<5 in the VDP dataset) violate the assumptions required for reliable chi-square testing [15].

For the *HSTS*, which instructs the browser to automatically upgrade HTTP connections to HTTPS, we examined the configuration. While the VDP domains use it slightly more often (see Table 4), the observed configurations revealed only minor differences. For example, the *max-age* of at least one year was set by 28.19% (of 37.27%) of VDP and 26.88% (of 39.93%) of CrUX domains, showing no significant difference. Insecure values (e.g., zero or malformed) were rare overall, but occurred twice as often in CrUX domains (*0.45%* VDP vs. *0.99%* CrUX), leading in a statistically significant difference (chi-square; $p \approx 0.0048$) and allowing insecure HTTP connections. Regarding the *includeSubDomains*, both datasets set it in one quarter of the investigated domains, with no significant difference between configurations.

While the insecure usage of the earlier headers primarily leads to cross-origin or privacy issues, *CSP* is more critical because it mitigates multiple attacks. Our evaluation with the *Google's CSP-Evaluator* provides insights into its recommendations (see Appendix C), reporting their assigned severity and the share of domains where at least one subpage received the recommendation for both datasets. Around 5% more domains within the CrUX dataset completely omit the important directives *script-src* and *object-src* compared to the VDP domains. In contrast, however, within the VDP domains a higher percentage of 2-4% include *unsafe-eval* or *unsafe-inline*. Both values heavily reduce the security guarantees by increasing the attack surface for XSS attacks.

Besides the automated analysis using Google's CSP Evaluator, we also examine two additional directives in more detail. These are the *upgrade-insecure-requests* and the (deprecated but still widely supported) *block-all-mixed-content*. Both directives target mixed content attacks by ensuring that all resources loaded by the page use HTTPS connections. In most domains — 85% in the VDP dataset and 75% in CrUX — where a CSP header was present, neither directive was used in any of the subordinate webpages. In cases where they were used, both datasets relied mainly on *upgrade-insecure-requests*, which is the more modern directive.

The last directive we investigated is *frame-ancestors*. The percentage of domains that did not use the directive on any subpages is nearly identical in both datasets: 11.02% for VDP and 10.82% for CrUX. However, domains within the VDP dataset, including at least one insecure value within their webpages, add up to 0.42% of the total, compared to only 0.20% in the CrUX dataset. Interestingly, when considering the opposite, namely the number of domains that consistently use their own origin or self across all their subpages, the situation is reversed. The domains in the VDP dataset do so in 0.36%, while only 0.17% of the CrUX domains. In summary, our analysis results hint at CrUX showing more consistency and fewer mixed or insecure framing configurations overall. The VDP dataset, on the other hand, demonstrates a greater polarity: domains either use the directive securely with self or are misconfigured and unsafe.

Another restricting header for which our analysis led to interesting insights is the *PermissionPolicy*. The most frequently observed configurations were restrictive and in favor of web security, explicitly blocking access to sensitive features. The most restricted features were *microphone()*, *camera()*, and *geolocation()*, observed in between 1.16% and 4.34% of the domains, with VDP domains consistently applying each of those over twice as often as CrUX. These suggest a stronger awareness of implementing a least-privilege permission strategy among VDP websites. Overall, broad and non-restrictive permissions are used very infrequently and even less for more sensitive capabilities such as *camera*, *microphone*, or *geolocation*, highlighting a general awareness of permission hardening across both datasets.

5.3 Cookies

Another key measure to limit the impact of XSS attacks is securing cookies by setting the appropriate flags. Figure 4 shows the applied cookie flags across the pages in our datasets, broken down by dataset and whether the cookies are first-party or third-party.

Both, first- and third-party, show stronger security within the VDP dataset for the *secure* and *httpOnly* attributes, with both differences statistically significant (chi-square; $p \ll 0.05$). This suggests a more secure cookie handling due to higher awareness of the usage of secure default configurations within frameworks. Additionally, VDP cookies used the *SameSite=Strict* attribute significantly more

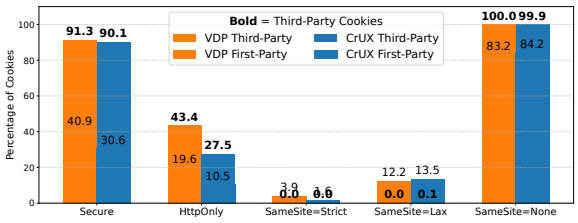
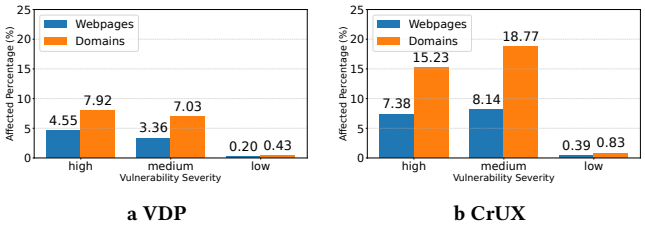


Figure 4: Usage of cookie flags within the datasets - first vs third party



a VDP **b CrUX**

Figure 5: Comparison of JavaScript library vulnerabilities across datasets

	Cookie	Secure	HttpOnly	SameSite
C	*PHPSESSID*	34.1%	37.5%	84.9% None, 14.6% Lax, 0.4% Strict
	JSESSIONID	57.0%	81.0%	93.6% None, 4.1% Lax, 2.3% Strict
	cf_clearance	96.9%	100%	100% None
	__cf_bm	95.6%	100%	100% None
	_cfuvid	91.6%	99.3%	100% None
V	*PHPSESSID*	64.8%	62.8%	76.4% None, 14.0% Lax, 9.6% Strict
	JSESSIONID	90.1%	85.6%	89.3% None, 6.0% Lax, 4.7% Strict
	cf_clearance	98.1%	100%	100% None
	__cf_bm	92.8%	100%	100% None
	_cfuvid	93.3%	98.8%	100% None

Table 5: Sensitive cookies flag usage in the CrUX (C) and VDP (V) datasets

often (chi-square; $p \ll 0.05$). Also, as expected, because cross-origin use requires it, most third-party cookies in both sets use *SameSite=None*, showing no significant difference.

A similar trend is also identified within a sample of security-sensitive session and authentication cookies as presented in Table 5. While third-party cookies such as *__cf_bm* showed consistently strong flag usage in both datasets, traditional first-party cookies (e.g., *PHPSESSID*, *JSESSIONID*) revealed notable differences. VDP websites use necessary flags like *Secure*, *HttpOnly*, and *SameSite* more often, highlighting a stricter focus on cookie security.

In addition to the cookie options, we used the *OpenCookieDatabase* to classify the top 25 most frequently used cookies by their purpose to identify shifts there. Half of the cookies are classified as *analytics* (13) and *marketing* (6), while for the VDP dataset, only three were classified as *analytics* and eight as *marketing*. Conversely, some cookies like *__cf_bm* or *_cfuvid* (Cloudflare cookies) appear more often in the VDP dataset, hinting at increased bot protection and request fingerprinting among sites with a VDP, while still overall focusing more on functionality and using less analytics. In contrast, the higher share of analytics-related cookies in CrUX domains indicates a stronger focus on user tracking and behavioral analysis, which is often leveraged for advertising optimization.

5.4 JavaScript Inclusions

While accessing insecurely configured cookies is advantageous in client-side attacks, the prone code snippets and thus potential vulnerabilities are often enabled by included third-party scripts [42, 56, 57]. Our collection of included scripts within webpages of both datasets yielded a great success rate of 98.74% for VDP and 94.53% for CrUX. By analyzing the number of third-party inclusions, we discovered the VDP dataset includes 11.96 unique domains with a median of six. In contrast, the included domains' numbers are higher for the CrUX dataset, as they include JavaScript from 17.27 unique domains with a median of 11. This indicates a higher dependency on third-party code, which must be carefully maintained to prevent third-party vulnerabilities in their own webpages.

To evaluate the third-party hygiene, we used *retire.js* to check for known vulnerabilities and their severities at the webpage and domain level, as shown in the Figure 5a and the Figure 5b. The separation reveals that the vulnerabilities are not concentrated in a few domains with many insecure pages, but rather distributed across many domains, each with fewer issues, as reflected by the higher percentage of affected domains compared to webpages. Overall *retire.js* identified a statistically significant difference (chi-square; $p < 0.001$) in the prevalence of potentially vulnerable pages, with vulnerable versions detected on 8.09% (1108) of VDP pages compared to 15.91% (12 369) of CrUX pages. Since the discovery relies on library versions, vulnerabilities might not be exploitable due to mitigations or unused code. Still, the measurement provides a great comparison of the update discipline and awareness of third-party vulnerabilities, with VDP sites performing significantly better.

Furthermore, we analyzed CVE-tagged vulnerabilities reported by the tool to estimate their age, based on their CVE ID, and present our results in Figure 6. Observing the plot reveals that websites within VDPs show vulnerabilities mainly between zero and five years, while CrUX websites have even older vulnerabilities. This observation is further validated by the calculated higher average (+0.6) and median (+1) age in years. This analysis further indicates a slightly worse security hygiene and patch management regarding third-party libraries and their vulnerabilities within the CrUX dataset. The complete breakdown of the most frequently identified vulnerable libraries and versions is enclosed in the Appendix B. It reveals further insights into the different libraries responsible for the vulnerabilities included in the webpages of both datasets.

Besides assessing known vulnerabilities of third-party JavaScript inclusions, we manually and automatically classified the top 25 most frequently included domains (details in Appendix B). Our manual investigation reveals that CrUX websites more often included domains linked to *advertising* and *tracking* (e.g., *googlesyndication*, *criteo*, and *amazon-adsystem*), while VDP webpages favored more *functionality*-focused inclusions like *cookielaw* and *adobedtm*. These observations were further supported by the automated classifications. According to *Disconnect.me*, 8 of the top 25 CrUX domains related to *advertising*, 4 to *analytics*, and 4 to *fingerprinting*; VDP

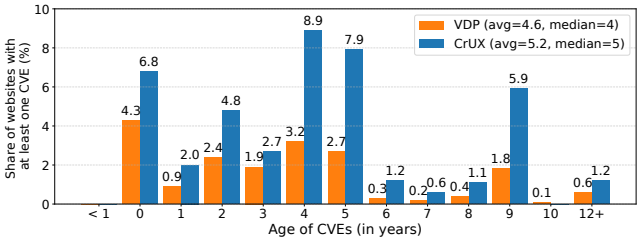


Figure 6: Age of CVEs discovered within websites of VDP vs. CrUX

had only 2 in *advertising*, 6 in *analytics*, and 3 in *fingerprinting*. *EasyList* and *EasyPrivacy* classifications also confirmed this: only 1 of the top VDP URLs was flagged for *advertisement* (vs. 15 in CrUX), and 11 VDP URLs were marked as *trackers* (vs. 6 in CrUX).

To summarize the classifications, they revealed that CrUX sites tend to include more third-party scripts for *advertising* and *monetization*, reflecting their commercial focus. In contrast, VDP sites show fewer third-party inclusions overall, with a stronger emphasis on *functional* and *analytic* purposes, but also higher numbers in *tracking*. This suggests a more cautious and less monetary focus.

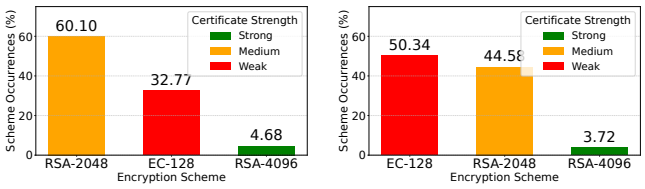
5.5 Client-Side XSS

For the XSS analysis, we collected over 6.3 million unique taints by visiting each webpage of both datasets once. The results of each collection step regarding client-side XSS are presented in Table 6. In the next step, after the automated exploit generation, at least one exploit was generated for 441 unique webpages (3.08%) in the VDP dataset and 3548 webpages (4.37%) in the CrUX dataset. When breaking these down to the domain level, an even higher difference (5.18% VDP vs. 7.16% CrUX) is observed, which is statistically significant under a chi-square test ($p < 0.001$). The higher percentage of generated exploits within the CrUX set suggests a greater prevalence of unprotected flows from user-controllable sources to potentially dangerous sinks.

Nevertheless, while the presence of such flows is already concerning, these might not be exploitable due to sanitization or similar countermeasures applied. The real security risk is confirmed by successfully validated exploits. In total, we validated exploits for 41 domains (0.49%) within CrUX, and only for three domains (0.16%) of the VDP dataset. Similar to some security headers, the sample set is too small to reliably perform a chi-square test [15]. Although these percentages are low, they are still critical, as such fundamental client-side XSS vulnerabilities should no longer be present under modern security best practices. And looking back at the earlier number of websites for which exploits were generated, this difference proved to be significant. Furthermore, the CrUX dataset contained nearly three times as many vulnerable domains and, when vulnerable, often included multiple affected pages. Our manual review revealed this was typically caused by single flows or closely related ones, exploitable across multiple subpages, rather than multiple distinct vulnerabilities.

	VDP	CRUX
Collected Taints	1 063 503	5 595 563
Websites with Generated Exploit (%)	441 (3.08)	3548 (4.37)
Domains with Generated Exploit (%)	97 (5.18)	605 (7.16)
Websites with Validated Exploit (%)	11 (0.08)	252 (0.31)
Domains with Validated Exploit (%)	3 (0.16)	41 (0.49)

Table 6: Overview of XSS detection steps



a VDP **b CrUX**

Figure 7: Strength rating of certificate encryption

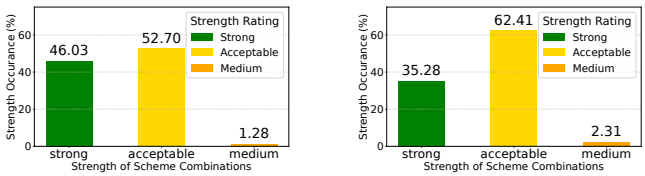
5.6 Transport-Layer Security

For the transport-layer security metrics, especially employed via certificate settings, we relied on the tool *sslscan*. Our measurement achieved a success rate of almost 90% for CrUX targets in comparison to 84% for targets belonging to the VDPs.

Within both datasets, we identified six expired certificates and none that were not yet valid. Interestingly, the non-valid certificates of the CrUX dataset included four certificates that presented *localhost.localdomain* as the certificate subject. This value is often the default or testing value and, therefore, is not trusted by the browsers. It indicates a configuration error on the operator's site. As for our evaluation, we categorized those as insecure as well. Both datasets present a low percentage of self-signed certificates, with only two within the VDP and four within the CrUX dataset. These comparisons present equal results for both datasets, indicating a similar certificate of hygiene.

By investigating the used configurations of the certificate itself, such as the used encryption and offered encryption schemes, the VDP dataset performed slightly better. The observed encryption schemes used for the certificates themselves are presented in Figure 7a and Figure 7b. The most prominent combinations are *RSA-2048* and *EC-128* within both datasets, but for certificates from the VDP dataset, the first is most often used with around 60%. Meanwhile, for the CrUX dataset, the latter is more often used, namely for slightly more than 50%. Even if the Elliptic Curve calculation is more complex than RSA, the key size of 128 bits is considered low for current standards, and thus we consider it weaker than RSA with the key size of 2048 bits [3]. More CrUX certificates rely on this lower security standard, indicating a more outdated encryption implementation, which is also statistically significant according to our chi-square test ($p < 0.0001$).

Even more interesting are the schemes offered by the certificate for the client to chose from to establish the encrypted communication channel. The tool *sslscan* assigns a strength for each TLS/SSL version and cipher combination, which lead to the overview presented in Figure 8a and Figure 8b. Notably, both datasets almost exclusively present combinations classified as *acceptable* or *strong*.



a VDP **b CrUX**

Figure 8: Strength ratings by *sslscan* for offered schemes

These results indicate a general awareness of cryptographic threats towards the transport layer. Nevertheless, the VDP dataset even performs slightly better, including a *medium* rated combination in only 1.26%, while the CrUX dataset does so in almost twice as much (2.31%). In addition, the inclusion of a combination with the rating *strong* is lower within the CrUX dataset at only 35.28%, while the VDP dataset reaches a percentage of 46.03%. A chi-square test confirms the statistical significance of this difference (p = 0.0008), supporting the observation that VDPs more frequently deploy stronger encryption schemes.

> **Key Takeaways:** Throughout our results, the VDP dataset demonstrates a higher security level compared to CrUX. VDP webpages employed most configurations more securely, and significantly fewer (potential) exploitable vulnerabilities were discovered.

6 DISCUSSION

In the following discussion, we first highlight the main insights gained from our security comparison, followed by the limitations of this work to place these insights in the proper context. We then outline our ethical considerations. Finally, with all insights in mind, we discuss how further work can build on the takeaways of this study and how VDP providers could better support their efforts.

6.1 Main Insights

Can VDPs serve as a viable alternative dataset for web security research? The short answer is, yes, provided that existing natural biases are acknowledged.

Our analysis shows that a substantial fraction of programs explicitly permit testing for server-side vulnerability classes such as access control flaws, SQL injection, and SSRF. This authorization removes the primary ethical and legal obstacle that has historically prevented large-scale server-side measurement studies on the Web.

Regarding potential biases, our results indicate that the VDP dataset is skewed toward websites with a stronger security posture. In particular, we identified fewer exploitable client-side XSS issues on sites in our VDP dataset (0.16%) compared to 0.49% on CrUX. This bias is also reflected in the adoption of security policies. Websites listed in VDPs use the *CSP* header more frequently. They also deploy a stronger *Permission-Policy* twice as often, explicitly blocking access to features like the microphone or camera more often, following the least-privileged principle. Additionally, *Referrer-Policy* is deployed more often on VDPs (20.22%) compared to sites listed in CrUX (16.61%), even with more restrictive configurations. Other indicators support this trend, such as the higher use of cookie options like *httpOnly* (43.3% vs. 27.5%), the use of stronger TLS signing algorithms and fewer identified vulnerable or outdated third-party libraries (8.09% vs. 15.91%) on VDP sites.

That said, some metrics show little difference. For example, the general deployment of HTTP security headers, although slightly higher in our VDP dataset, is similar across both datasets. Notably, for CSP, VDPs more often include the insecure CSP directives *unsafe-inline* and *unsafe-eval*, reflecting the well-known challenges in crafting robust CSPs [11, 64].

In general, our findings suggest a clear focus on (potentially) exploitable vulnerabilities on VDP-listed websites rather than vulnerability mitigation policies such as HTTP header configurations. This focus aligns with the goal of such external security programs, namely, identifying and remedying security issues. Consequently, issues like XSS and insecure third-party code rank among the most frequently reported in VDPs [7, 27]. Our findings suggest that VDP-listed sites have fewer vulnerabilities and thus should be seen as a lower-bound relative to the general Web. Moreover, given the focus on impactful issues, we assume that companies selectively include high-impact systems in their VDPs, thus shifting the bias of a VDP dataset towards critical or attack-prone assets. Nevertheless, unlike CrUX, which reflects popularity and adoption, a VDP dataset emphasizes real-world exploitability. In our view, this makes it a valuable resource for studying actual vulnerability exposure and exploring areas otherwise inaccessible.

6.2 Limitations

As mentioned above, our findings highlight exclusively client-side measurable security aspects (for ethical reasons as explained in Section 6.3). As such, our comparison may not fully reflect other aspects, including the server-side. Still, the VDP dataset outperforms the CrUX datasets with actual vulnerabilities (like XSS) commonly reported to VDPs. Since the top ten reported vulnerabilities in VDPs also include server-side issues — such as Insecure Direct Object References (IDOR) or session misconfigurations — it is plausible, though unconfirmed, that a similar disparity exists in server-side security between the datasets [7, 27].

Another limitation concerns the statistical test we used, chi-square. In some cases, sample sizes were too small to yield reliable results. We therefore excluded them from the test in these cases.

We also limit our analysis to the static surface of webpages without interactions, logins, or forced execution. While deeper interactions could reveal more, prior work found minimal differences between logged-in and anonymous views [60].

Lastly, the VDP dataset creation process itself has limitations. First, the dataset represents only a lower bound of programs, as it is solely collected from public VDPs visible to new accounts. Moreover, some program details are available only in natural language, such as allowed vulnerability types, requiring a combined RegEx- and LLM-based extraction. While LLM-based approaches are not fully reproducible, we minimized the model's temperature to make it as deterministic as possible. To ensure research transparency, we made all scripts and data publicly available [17].

6.3 Ethics

Our experiments were performed on live systems, which always necessitate ethical considerations. Following the Menlo Report [2], we balanced potential risks against societal and research benefits and performed a stakeholder analysis. Below, we outline the key stakeholders, along with risks they face and benefits they may gain.

Our first stakeholder group is *program owner*, i.e., organizations offering disclosure programs. Our scans could have affected their websites with additional traffic and false positive security alerts, leading to increased resource usage. However, these organizations explicitly consented to security testing by hosting such programs, thereby accepting this risk. To further minimize it, we compiled common rules from these programs and followed them when crawling (see Section 4). This includes limiting requests to 3 per second and adding HTTP headers with contact information. We responsibly disclosed all vulnerabilities according to program rules, enabling program owners to address potential issues. Furthermore, this paper contributes to a broader understanding of the effectiveness of disclosure programs, supporting owners in informed decisions.

Another core stakeholder is the group of *website owners without disclosure programs* (websites from the CrUX list). Although they did not consent to scanning, they face similar risks as owners of sites with disclosure programs. At the same time, they benefit from our responsible disclosure of vulnerabilities, which we reported to each owner via the most suitable contact addresses found on their sites. They also benefit indirectly from the study's findings, which may inform decisions about adopting disclosure programs. Finally, the paper advocates for responsible scanning practices, helping to reduce the risk of harmful large-scale scanning in future research.

Besides the owners of a website, its *end users* may also face risks such as bad website performance when overwhelmed with scans. However, as mentioned before, our scans do not exceed 3 requests per second. Our monitoring showed no negative effect on the targeted websites, which are the top-visited websites on the Web and sites with disclosure programs. In the long run, end users benefit from more responsible research methodologies and improved web security through disclosure programs.

Lastly, we as *researchers* are also a core stakeholder. We benefit from publishing this work but face risks common in web measurement, such as exposure to disturbing content or unpredictable legal reactions of website owners. Drawing on prior experience, we assessed these risks as low, discussed them within the team, and ensured voluntary participation.

Overall, our methods align with practices successfully used in prior web research. Based on our stakeholder analysis, we believe the societal and academic benefits of improved understanding of VDPs outweigh the limited and mitigated risks of our scans.

6.4 Recommendations For Future Research

Building on the insights of our security comparison and keeping limitations in mind, we now discuss how VDPs can support future measurement research.

Some measurement studies — such as those involving any server-side code executions — are ethically and legally challenging without consent from the website owners [2, 29]. However, these remain important to study as they can help to highlight server-side security trends in the Web ecosystem and shift focus on urgent, understudied or new evolving issues. VDPs can provide a *"safe harbor"* for such work as pre-established agreements. Furthermore, these programs help address another common challenge in large-scale studies, the reporting of findings to the individual vendors [70]. VDPs streamline this process by offering a direct communication channel, which is also useful when unexpected issues arise or when researchers seek further context for qualitative analysis.

For researchers who plan to conduct research in this direction, we recommend using our open-sourced tool [17] to compile a dataset of VDP-covered sites. When conducting scans, researchers should respect the program rules (see Section 3.3) by applying appropriate request rate limits (e.g., two requests per second) and setting program-specific headers. Also, for invasive methods, program rules should be reviewed carefully. In addition, best practices from prior work should also be followed, such as pre-studies in controlled environments, active monitoring and data minimization [21, 29].

While these VDPs allow researchers to conduct large-scale research that might otherwise be infeasible, downsides remain. Of course, these programs represent only a small, security-aware part of the Web. However, Web measurement research is always biased; commonly used datasets like Alexa, CrUX, or Tranco are biased towards popular Western-centric sites, which also make up just a subset of the World Wide Web. As long as such biases are made transparent, using VDP-based datasets to present lower bounds on otherwise hidden security issues is justifiable.

6.5 Recommendations For VDP Providers

This work highlights that compiling a dataset of VDP-listed sites is far from straightforward. Platforms lack a standardized way to offer the often textual scopes and rules. They also miss a uniform reporting process. From our own experience, complex or low-impact reports are sometimes dismissed by impact-driven providers, even when the vendors would welcome them. We therefore call on VDP providers to adopt standard interfaces for (academic) researchers to access program details and report issues at scale.

7 CONCLUSION

We compiled a dataset of websites participating in vulnerability disclosure programs (VDPs), along with their program rules. Using this dataset, we conducted a measurement comparing client-side security aspects of VDP-listed sites against those from a CrUX dataset. Our findings provide the first empirical evidence supporting the widely held belief that VDPs contribute to improved web security.

Building on this work, we encourage researchers to consider VDPs as a valuable dataset for security measurements that would otherwise be difficult or infeasible. At the same time, we urge VDP platform providers to support academic research by offering more standardized mechanisms for compiling datasets and enabling large-scale notifications.

8 ACKNOWLEDGMENT

We thank the AsiaCCS reviewers and the shepherd for their valuable suggestions, improving the presentation of our paper. This work was conducted in the scope of a dissertation at the Saarbrücken Graduate School of Computer Science

REFERENCES

[1] Omer Akgul, Taha Eghtesad, Amit Elazari, Omprakash Gnawali, Jens Grossklags, Michelle L. Mazurek, Daniel Votipka, and Aron Laszka. 2023. Bug Hunters' Perspectives on the Challenges and Benefits of the Bug Bounty Ecosystem. In *32nd USENIX Security Symposium*. https://www.usenix.org/conference/usenixsecurity23/presentation/akgul

[2] Michael Bailey, David Dittrich, Erin Kenneally, and Doug Maughan. 2012. The Menlo Report. *IEEE Security & Privacy* 10, 2 (2012). https://doi.org/10.1109/MSP.2012.52

[3] NIST Elaine Barker. 2020. NIST Special Publication 800-57 Part 1 - Revision 5. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

[4] Pedro Bernardo, Lorenzo Veronese, Valentino Dalla Valle, Stefano Calzavara, Marco Squarcina, Pedro Adão, and Matteo Maffei. 2024. Web Platform Threats: Automated Detection of Web Security Issues With WPT. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/bernardo

[5] Ashish Bhushan, Vivekananda Billa, Minal Sonkar, and Vighnesh Chavan. 2022. The Dynamics of a Bug Bounty Platform. In *2022 5th International Conference on Advances in Science and Technology (ICAST)*. https://doi.org/10.1109/ICAST55766.2022.10039642

[6] Bodo Möller, Thai Duong, Krzysztof Kotowicz. 2025. SSL Poodle Security Advisory. https://openssl-library.org/files/ssl-poodle.pdf.

[7] Bugcrowd. 2025. *Top 10 Vulnerabilities*. https://www.bugcrowd.com/resource/top-10-vulnerabilities/

[8] Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, and Wilayat Khan. 2015. CookiExt: Patching the Browser Against Session Hijacking Attacks. *Journal of Computer Security* 23 (2015). https://doi.org/10.3233/JCS-150529

[9] Business Research Insights. 2024. Bug Bounty Platforms Market Size, Share, Growth, And Industry Analysis, By Type (Cloud, SaaS Web, Mobile-Android Native, Mobile-iOS Native, Host), By Application (Finance & Banking, Software Development, Retail, Government, Other), Regional Forecast By 2033. https://www.businessresearchinsights.com/market-reports/bug-bounty-platforms-market-102501.

[10] Franco Callegati, Walter Cerroni, and Marco Ramilli. 2009. Man-in-the-Middle Attack to the HTTPS Protocol. *IEEE Security Privacy* 7, 1 (2009). https://doi.org/10.1109/MSP.2009.12

[11] Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi. 2016. Content Security Problems? Evaluating the Effectiveness of Content Security Policy in the Wild. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery. https://doi.org/10.1145/2976749.2978338

[12] Stefano Calzavara, Sebastian Roth, Alvise Rabitti, Michael Backes, and Ben Stock. 2020. A Tale of Two Headers: A Formal Analysis of Inconsistent Click-Jacking Protection on the Web. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity20/presentation/calzavara

[13] Chrome. 2024. Chrome Privacy. https://privacysandbox.com/intl/en_us/news/privacy-sandbox-update/.

[14] CISA. 2024. Vulnerability Disclosure Policy Platform 2023 Annual Report. https://industrialcyber.co/vulnerabilities/cisa-releases-2023-annual-report-for-vdp-platform-highlighting-significant-achievements/.

[15] William G Cochran. 1954. Some methods for strengthening the common χ 2 tests. *Biometrics* 10, 4 (1954), 417–451.

[16] Cyberdefense. 2020. "Get a Bug If You Find a Bug". https://www.orangecyberdefense.com/be/blog/get-a-bug-if-you-find-a-bug.

[17] Philip Decker. 2025. *Artifacts*. https://github.com/cispa/VDPCollect

[18] Nurullah Demir, Daniel Theis, Tobias Urban, and Norbert Pohlmann. 2022. Towards Understanding First-Party Cookie Tracking in the Field. *CoRR* abs/2202.01498 (2022). https://arxiv.org/abs/2202.01498

[19] Nurullah Demir, Tobias Urban, Norbert Pohlmann, and Christian Wressnegger. 2024. A Large-Scale Study of Cookie Banner Interaction Tools and Their Impact on Users' Privacy. https://doi.org/10.56553/popets-2024-0001

[20] disclose. 2025. Public List of VDP/BB Providers. https://github.com/disclose/bug-bounty-platforms.

[21] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. 2013. {ZMap}: Fast internet-wide scanning and its security applications. In *22nd USENIX Security Symposium (USENIX Security 13)*.

[22] Steven Englehardt and Arvind Narayanan. 2016. Online Tracking: A 1-million-site Measurement and Analysis. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery. https://doi.org/10.1145/2976749.2978313

[23] Matthew Finifter, Devdatta Akhawe, and David Wagner. 2013. An empirical study of vulnerability rewards programs. In *Proceedings of the 22nd USENIX Conference on Security*. USENIX Association.

[24] Alexander Gamero-Garrido, Stefan Savage, Kirill Levchenko, and Alex C. Snoeren. 2017. Quantifying the Pressure of Legal Risks on Third-party Vulnerability Research. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery. https://doi.org/10.1145/3133956.3134047

[25] Dave Gerry. 2025. *Crowdsourced intelligence in action: Bugcrowd's 2024 year in review*. https://www.bugcrowd.com/blog/crowdsourced-intelligence-in-action-bugcrowds-2024-year-in-review/

[26] HackerOne. 2025. *2022 Hacker-Powered Security Highlights Blog Post*. https://www.hackerone.com/press-release/hackers-discover-over-65000-software-flaws-2022-according-hackerone-report

[27] HackerOne. 2025. *CWE Discovery*. https://hackerone.com/hacktivity/cwe_discovery

[28] Florian Hantke, Stefano Calzavara, Moritz Wilhelm, Alvise Rabitti, and Ben Stock. 2023. You Call This Archaeology? Evaluating Web Archives for Reproducible Web Security Measurements. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery. https://doi.org/10.1145/3576915.3616688

[29] Florian Hantke, Sebastian Roth, Rafael Mrowczynski, Christine Utz, and Ben Stock. 2024. Where Are the Red Lines? Towards Ethical Server-Side Scans in Security and Privacy Research. In *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE.

[30] Florian Hantke, Peter Snyder, Hamed Haddadi, and Ben Stock. 2025. Web Execution Bundles: Reproducible, Accurate, and Archivable Web Measurements. In *34th USENIX Security Symposium (USENIX Security 25)*. https://www.usenix.org/system/files/conference/usenixsecurity25/sec25cycle1-prepub-158-hantke.pdf

[31] Xuehui Hu, Nishanth Sastry, and Mainack Mondal. 2021. CCCC: Corralling Cookies into Categories with CookieMonster. In *Proceedings of the 13th ACM Web Science Conference 2021*. Association for Computing Machinery. https://doi.org/10.1145/3447535.3462509

[32] Lin-Shung Huang, Alex Moshchuk, Helen J. Wang, Stuart Schecter, and Collin Jackson. 2012. Clickjacking: Attacks and Defenses. In *21st USENIX Security Symposium (USENIX Security 12)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity12/technical-sessions/presentation/huang

[33] IETF. 2025. RFC Cookie Usage. https://datatracker.ietf.org/doc/html/rfc6265.

[34] jkwakman. 2025. OpenCookieDatabase - GitHub Repository. https://github.com/jkwakman/Open-Cookie-Database.

[35] John Wilander. 2024. Webkit Privacy. https://webkit.org/blog/10218/full-third-party-cookie-blocking-and-more.

[36] Soheil Khodayari and Giancarlo Pellegrino. 2022. The State of the SameSite: Studying the Usage, Effectiveness, and Adequacy of SameSite Cookies. In *2022 IEEE Symposium on Security and Privacy (SP)*. https://doi.org/10.1109/SP46214.2022.9833637

[37] Soheil Khodayari and Giancarlo Pellegrino. 2023. It's (DOM) Clobbering Time: Attack Techniques, Prevalence, and Defenses. In *2023 IEEE Symposium on Security and Privacy (SP)*. 1041–1058. https://doi.org/10.1109/SP46215.2023.10179403

[38] Robin Kirchner, Jonas Möller, Marius Musch, David Klein, Konrad Rieck, and Martin Johns. 2024. Dancer in the Dark: Synthesizing and Evaluating Polyglots for Blind Cross-Site Scripting. In *33rd USENIX Security Symposium (USENIX Security 24)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity24/presentation/kirchner

[39] David Klein, Thomas Barber, Souphiane Bensalim, Ben Stock, and Martin Johns. 2022. Hand Sanitizers in the Wild: A Large-scale Study of Custom JavaScript Sanitizer Functions. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroSP)*. https://doi.org/10.1109/EuroSP53844.2022.00023

[40] David Klein, Marius Musch, Thomas Barber, Moritz Kopmann, and Martin Johns. 2022. Accept All Exploits: Exploring the Security Impact of Cookie Banners. In *Proceedings of the 38th Annual Computer Security Applications Conference* (Austin, TX, USA) *(ACSAC '22)*. Association for Computing Machinery, New York, NY, USA, 911–922. https://doi.org/10.1145/3564625.3564647

[41] Tadayoshi Kohno, Yasemin Acar, and Wulf Loh. 2023. Ethical Frameworks and Computer Security Trolley Problems: Foundations for Conversations. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association. https://doi.org/10.48550/arXiv.2302.14326

[42] Tobias Lauinger, Abdelberi Chaabane, and Christo Wilson. 2018. Thou Shalt Not Depend on Me: A look at JavaScript libraries in the wild. *Queue* 16, 1 (2018). https://doi.org/10.1145/3194653.3205288

[43] MDN. 2025. Cross-Origin-Embedder-Policy - HTTP MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Cross-Origin-Embedder-Policy

[44] MDN. 2025. Cross-Origin-Opener-Policy - HTTP MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Cross-Origin-Opener-Policy

[45] MDN. 2025. Cross-Origin-Resource-Policy - HTTP MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Cross-Origin-Resource-Policy

[46] MDN. 2025. MDN Web Docs - Cookie. https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies.

[47] MDN. 2025. MDN Web Docs - Mixed Content. https://developer.mozilla.org/en-US/docs/Web/Security/Mixed_content.

[48] MDN. 2025. Permissions Policy - HTTP MDN. https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Permissions_Policy

[49] MDN. 2025. Referrer-Policy - HTTP. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Referrer-Policy
[50] MDN. 2025. Strict-Transport-Security - HTTP. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/Strict-Transport-Security.
[51] MDN. 2025. X-Frame-Options Header - HTTP. https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Headers/X-Frame-Options.
[52] MDN Web Docs. 2025. MDN Web Docs - Content Security Policy. https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/CSP.
[53] Microsoft. 2009. Happy 10th birthday Cross-Site Scripting! https://learn.microsoft.com/en-us/archive/blogs/dross/happy-10th-birthday-cross-site-scripting
[54] Microsoft Edge Team. 2024. Edge Privacy. https://blogs.windows.com/msedgedev/2024/03/05/new-privacy-preserving-ads-api/.
[55] Mozilla. 2024. Firefox Cookie Privacy. https://blog.mozilla.org/en/products/firefox/firefox-rolls-out-total-cookie-protection-by-default-to-all-users-worldwide/.
[56] Marius Musch, Marius Steffens, Sebastian Roth, Ben Stock, and Martin Johns. 2019. ScriptProtect: Mitigating Unsafe Third-Party JavaScript Practices. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. Association for Computing Machinery. https://doi.org/10.1145/3321705.3329841
[57] Nick Nikiforakis, Luca Invernizzi, Alexandros Kapravelos, Steven Van Acker, Wouter Joosen, Christopher Kruegel, Frank Piessens, and Giovanni Vigna. 2012. You are what you include: large-scale evaluation of remote javascript inclusions. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*. Association for Computing Machinery. https://doi.org/10.1145/2382196.2382274
[58] OhMyGuus. 2024. ISDCAC - I still don't care about cookies - Repository. https://github.com/OhMyGuus/I-Still-Dont-Care-About-Cookies.
[59] OWASP. 2024. OWASP TOP 10. https://owasp.org/www-project-top-ten/.
[60] Jannis Rautenstrauch, Metodi Mitkov, Thomas Helbrecht, Lorenz Hetterich, and Ben Stock. 2024. To Auth or Not To Auth? A Comparative Analysis of the Pre- and Post-Login Security Landscape. In *2024 IEEE Symposium on Security and Privacy (SP)*. https://doi.org/10.1109/SP54263.2024.00094
[61] Jannis Rautenstrauch, Giancarlo Pellegrino, and Ben Stock. 2023. The Leaky Web: Automated Discovery of Cross-Site Information Leaks in Browsers and the Web. In *2023 IEEE Symposium on Security and Privacy (SP)* (2023-05). IEEE. https://doi.org/10.1109/SP46215.2023.10179311
[62] rbsec. 2025. SSLScan GitHub Repository. https://github.com/rbsec/sslscan.
[63] RetireJS. 2025. Retire.js CVE list - Repository. https://rawgit.com/RetireJS/retire.js/master/repository/jsrepository.json.
[64] Sebastian Roth, Timothy Barron, Stefano Calzavara, Nick Nikiforakis, and Ben Stock. 2020. Complex Security Policy? A Longitudinal Analysis of Deployed Content Security Policies. In *Network and Distributed System Security Symposium (NDSS)*. https://eref.uni-bayreuth.de/id/eprint/91468/
[65] Sebastian Roth, Stefano Calzavara, Moritz Wilhelm, Alvise Rabitti, and Ben Stock. 2022. The Security Lottery: Measuring Client-Side Web Security Inconsistencies. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity22/presentation/roth
[66] Kimberly Ruth, Deepak Kumar, Brandon Wang, Luke Valenta, and Zakir Durumeric. 2022. Toppling top lists: evaluating the accuracy of popular website lists. *Proceedings of the 22nd ACM Internet Measurement Conference* (2022). https://api.semanticscholar.org/CorpusID:253045702
[67] Nayanamana Samarasinghe and Mohammad Mannan. 2019. Towards a global perspective on web tracking. *Comput. Secur.* 87, C (2019). https://doi.org/10.1016/j.cose.2019.101569
[68] Marius Steffens, Christian Rossow, Martin Johns, and Ben Stock. 2019. Don't Trust The Locals: Investigating the Prevalence of Persistent Client-Side Cross-Site Scripting in the Wild. In *Conference: Network and Distributed System Security Symposium*. https://doi.org/10.14722/ndss.2019.23009
[69] Ben Stock, Martin Johns, Marius Steffens, and Michael Backes. 2017. How the Web Tangled Itself: Uncovering the History of Client-Side Web (In)Security. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/stock
[70] Ben Stock, Giancarlo Pellegrino, Christian Rossow, Martin Johns, and Michael Backes. 2016. Hey, you have a problem: On the feasibility of {Large-Scale} web vulnerability notification. In *25th USENIX Security Symposium (USENIX Security 16)*.
[71] Synopsys Software Integrity Group. 2025. Heartbleed - Website. https://heartbleed.com/.
[72] telenor.se. 2025. YesWeHack - Telenor Sweden BBP. https://yeswehack.com/programs/telenor-sweden-public-bug-bounty-program.
[73] USENIX. 2025. USENIX Security '25 Call for Papers. https://www.usenix.org/conference/usenixsecurity25/call-for-papers.
[74] Thi-Huong-Giang Vu, Hai-Nam Hoang, and Thanh-Quang Le. 2025. A User Privacy Risk Driven Approach to Web Cookie Classification. In *Information and Communication Technology*, Wray Buntine, Morten Fjeld, Truyen Tran, Minh-Triet Tran, Binh Huynh Thi Thanh, and Takumi Miyoshi (Eds.). Springer Nature Singapore.
[75] Thomas Walshe and Andrew Simpson. 2020. An Empirical Study of Bug Bounty Programs. In *2020 IEEE 2nd International Workshop on Intelligent Bug Fixing*. https://doi.org/10.1109/IBF50092.2020.9034828
[76] Thomas Walshe and Andrew Simpson. 2023. Towards a Greater Understanding of Coordinated Vulnerability Disclosure Policy Documents. *Digital Threats* 4, 2, Article 29 (2023). https://doi.org/10.1145/3586180
[77] zakird. 2025. CrUX Top 1Million February 2025 - GitHub. https://github.com/zakird/crux-top-lists/commit/ad2c78421de4fde949b3ce77e241355867f3e664.

A REQUEST LIMITS

We analyzed the the rules typically mentioned in VDPs. One of these rules is the rate limit that researchers should apply when sending requests to a website. Figure 9 shows these limits as requests per seconds, excluded are three outliers. One program allowed up to 500 and two up to 1000 requests per seconds.

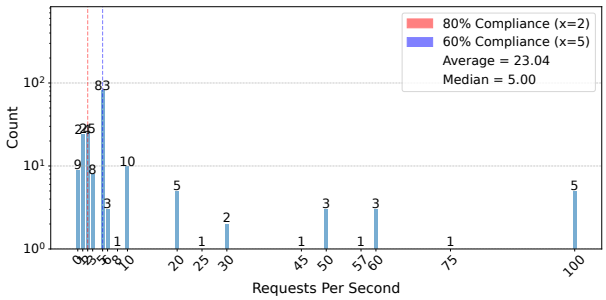


Figure 9: Allowed request rate per second

B LIST OF MOST FREQUENT THIRD-PARTY JAVASCRIPT INCLUSIONS

Our measurement of JavaScript inclusions via *retire.js* revealed multiple libraries which are repeatedly found within webpages having known vulnerabilities. Figure 10 presents the top 20 most frequently reported libraries across both datasets, along with their vulnerable version.

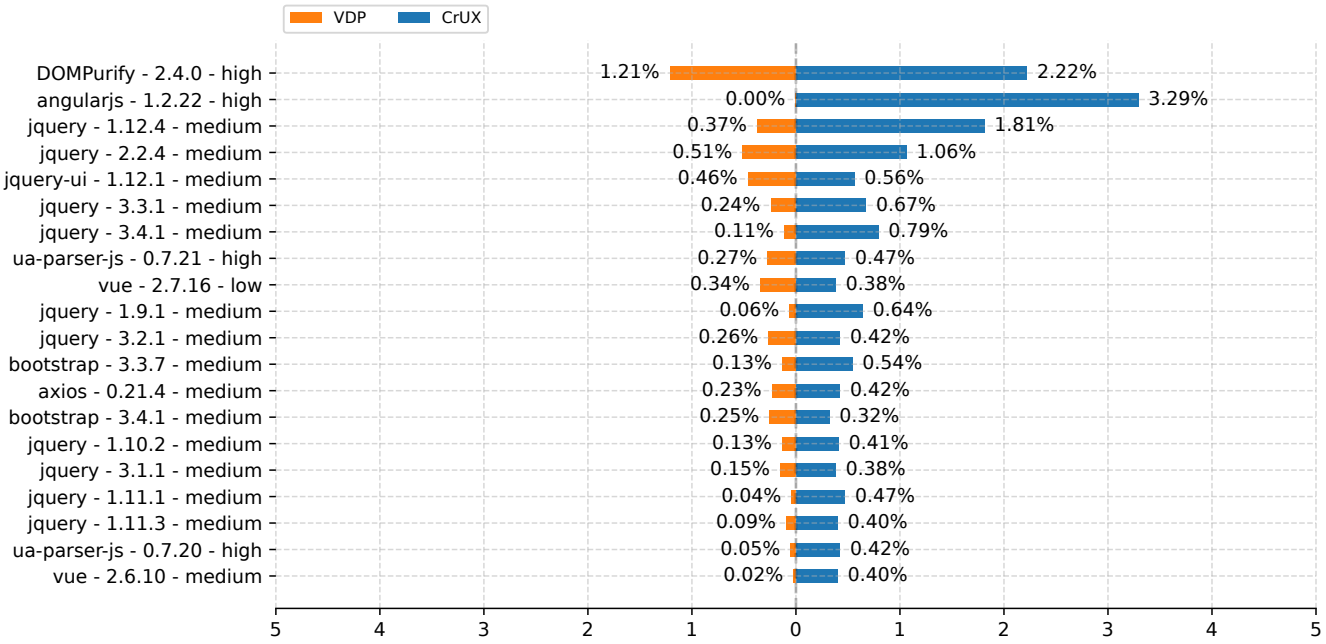


Figure 10: Most Frequent Vulnerable Library and Versions

C CSP EVALUATOR

We compared the CSPs deployed by sites belonging to VDPs with ones from the CrUX dataset by using the Goggle's CSP Validator. The results are listed in Table 7 and show that in general, VDP-listed sites have a larger number of secure CSPs while, however, lacking behind CrUX-listed sites in the high severity cases.

Description	**Affected Directive**	**Severity**	**VDP Domains** (of 23.31%)	**CrUX Domains** (of 22.66%)
Missing object-src allows the injection of plugins which can execute JavaScript	object-src	10	10.52%	15.09%
script-src directive is missing	script-src	10	9.85%	14.69%
'self' can be problematic if you host JSONP, AngularJS or user uploaded files	script-src	50	8.91%	4.51%
'unsafe-eval' allows the execution of code injected into DOM APIs such as eval()	script-src	50	7.65%	5.59%
'unsafe-inline' allows the execution of unsafe in-page scripts and event handlers	script-src	10	8.26%	4.76%
data: URI in default-src allows the execution of unsafe scripts	default-src	10	1.55%	2.20%
'unsafe-inline' allows the execution of unsafe in-page scripts and event handlers	default-src	10	1.32%	1.79%
'unsafe-eval' allows the execution of code injected into DOM APIs such as eval()	default-src	50	1.17%	1.71%
https: URI in script-src allows the execution of unsafe scripts	script-src	10	1.55%	4.97%
'self' can be problematic if you host JSONP, AngularJS or user uploaded files	default-src	50	1.41%	1.14%

Table 7: Recommendations by *CSP-Evaluator*