The Security Lottery: Measuring Client-Side Web Security Inconsistencies

Sebastian Roth, *CISPA Helmholtz Center for Information Security;* Stefano Calzavara, *Università Ca' Foscari Venezia;* Moritz Wilhelm, *CISPA Helmholtz Center for Information Security;* Alvise Rabitti, *Università Ca' Foscari Venezia;* Ben Stock, *CISPA Helmholtz Center for Information Security*

https://www.usenix.org/conference/usenixsecurity22/presentation/roth

This paper is included in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

The Security Lottery: Measuring Client-Side Web Security Inconsistencies

Sebastian Roth†, Stefano Calzavara‡, Moritz Wilhelm†, Alvise Rabitti‡, Ben Stock†

{sebastian.roth,moritz.wilhelm,stock}@cispa.de; {stefano.calzavara,alvise.rabitti}@unive.it

† *CISPA Helmholtz Center for Information Security* ‡ *Università Ca' Foscari Venezia*

Abstract

To mitigate a myriad of Web attacks, modern browsers support client-side security policies shipped through HTTP response headers. To enforce these defenses, the server needs to communicate them to the client, a seemingly straightforward process. However, users may access the same site in variegate ways, e.g., using different User-Agents, network access methods, or language settings. All these usage scenarios should enforce the same security policies, otherwise a *security lottery* would take place: depending on specific client characteristics, different levels of Web application security would be provided to users (*inconsistencies*). We formalize security guarantees provided through four popular mechanisms and apply this to measure the prevalence of inconsistencies in the security policies of top sites across different client characteristics. Based on our insights, we investigate the security implications of both deterministic and non-deterministic inconsistencies, and show how even prominent services are affected by them.

1 Introduction

Web applications are one of the primary access points to security-sensitive data and functionality which we use on a daily basis, hence they represent a primary target for attackers. Unfortunately, the attack surface against Web applications is very large and Web application security is a complicated topic which requires actions at very different levels, including the use of transport layer encryption via HTTPS, the implementation of server-side sanitization routines against attacks like SQL injection, and much more. An important and increasingly popular defense layer of Web applications is *client-side Web security*, i.e., the adoption of appropriate browser-side defenses to prevent or mitigate relevant Web threats [37]. Examples of such defenses which are quite popular on high-profile Web sites include cookie security attributes, Content Security Policy (CSP) and HTTP Strict Transport Security (HSTS). Several papers already studied (and criticized) the adoption of different client-side security mechanisms in the wild. For example, cookie security attributes turned out to be underused by site operators [5, 39], CSP is most often configured incorrectly [28, 43] and HSTS had a hard time getting traction even on top sites [22, 33]. However, these studies were performed in a fixed and often unspecified measurement setting, e.g., measuring security using a single crawler running a specific browser on a machine with a static IP.

In this paper, we investigate the client-side security of top sites from a new angle. Our analysis starts from the observation that client-side security inherently depends on the correct communication of security policies from the server to the client. This seemingly straightforward process might hide subtleties, which can affect Web application security and its large-scale measurement [18]. In particular, we observe that there is no guarantee that two clients accessing the same URL receive the same security policies. For example, clients accessing the same Web application from different geolocations might be served by different servers, due to the existence of several localized variants of the same site. Moreover, the same person might access the same Web application through different User-Agents, e.g., Chrome on their desktop computer and Safari on their iOS mobile device, while the same client might access the same Web application using different forms of network access, e.g., through a VPN. Finally, even the same HTTP request might receive different HTTP responses when it is sent multiple times by the same client apparently under the very same conditions, due to the DNS system resolving the same hostname to different IP addresses and the possible intervention of load balancers and HTTP middleboxes [16].

Intuitively, we (and Web users) would like all the responses served across these multiple legitimate scenarios to enforce the same security policies, otherwise a *security lottery* would take place: depending on specific client characteristics, different levels of Web application security would be provided to users. We refer to this class of potential security flaws as *inconsistencies*. Unfortunately, our research shows that multiple client characteristics might inadvertently affect Web application security in the wild, thus leaving users of prominent services unprotected against both Web and network attacks.

Contributions

In this paper we measure the prevalence of inconsistencies in the security policies of top sites across different client characteristics and we quantify their security implications:

1. We propose a data collection methodology tailored to our analysis and we build a dataset of 13,626,145 responses collected from the 10,000 highest-ranking sites available through HTTPS (based on Tranco [25]), while testing a number of different client characteristics. Our tests include the use of different User-Agents, network access methods and language settings.

2. We introduce general definitions of consistency for client-side security mechanisms and we instantiate them to a set of popular defenses available in modern browsers. Our definitions are *semantics-based*, i.e., they only capture inconsistencies with a potential security import, rather than superficial syntactic differences or other types of false positives coming from the different enforcement models of different security mechanisms.

3. We apply our definitions to the collected data and we report on the key findings. Our measurement shows that a significant fraction of the analyzed Web sites suffers from different types of client-side security inconsistencies. Remarkably, the majority of them can be attributed to specific client characteristics, which identify weak spots in the security configuration, while the others can be attributed to non-deterministic factors, which may nevertheless be exploited by attackers.

2 Background and Related Work

We first outline the relevant security mechanisms for our paper and then review the related work in this area.

2.1 Client-Side Security Mechanisms

All the security mechanisms in the present section are activated by means of HTTP response headers or (in some cases) by making use of meta tags within the HTML. In our analysis we only focus on security policies set via response headers, which is the most common deployment scenario indicated by prior work [6, 28].

Threat Model The client-side security mechanisms under study are designed to prevent different threats coming from *Web attackers* and *network attackers*, the standard threat models of the Web security literature. Web attackers operate a malicious site, say `evil.com`, and leverage it to launch attacks against target Web applications. Relevant Web threats include Cross Site Scripting (XSS), Cross Site Request Forgery (CSRF) and click-jacking. Network attackers extend the capabilities of Web attackers with full control of the unencrypted HTTP traffic, e.g., because they have access to the access point where the victim is connected.

Cookie Security Attributes Cookies are the traditional state management mechanism of the HTTP protocol, yet they suffer from a range of security problems in their default configuration. Site operators are thus recommended to improve the level of protection of their cookies by marking them with appropriate *security attributes*.

The *HttpOnly* attribute ensures that cookies are not accessible from JavaScript, which prevents cookie theft through malicious scripts, e.g., injected through XSS. The *Secure* attribute guarantees that cookies are never sent in plain HTTP requests, but only over encrypted HTTPS connections, which rules out network sniffing attempts. The combination of HttpOnly and Secure significantly raises the confidentiality guarantees of cookies, which is particularly important to protect against session hijacking [5, 11].

The latest addition to the set of attributes is called *SameSite*, which is meant to protect against CSRF attacks. If the attribute is set to Lax, cookies are only sent on cross-site requests when the main frame is navigated using a safe method like GET. If instead the attribute is set to Strict, cookies are never sent across sites. Some modern browsers like the latest versions of Chrome automatically promote all cookies to SameSite Lax, hence site operators can opt out from protection by setting the SameSite attribute to None for compatibility reasons.

Content Security Policy *Content Security Policy* (CSP) is a security mechanism originally aimed at mitigating the dangers of XSS and later extended to cover additional threats. In essence, a CSP is meant to ensure that only resources explicitly allowed by the developer of a page can be included therein. This is achieved by binding directives of the form *type*-src (for different content types, e.g., scripts, images, etc.) to a set of allowed sources, which can be as specific as a full URL or as unspecific as `https://*`. Script execution can also be controlled by allowing only script tags bearing a valid nonce attribute or matching a given hash, which should be preferred over allowlists [43].

Enforcing a CSP with a script-src (or alternatively a default-src) directive implicitly disables a page's ability to run inline scripts, inline event handlers, and string-to-code transformation functions like eval, which are the most common XSS vectors. These restrictions can be lifted by the 'unsafe-inline' or 'unsafe-eval' source-expression, respectively, although 'unsafe-inline' is voided by the use of nonces / hashes in modern browsers to authorize individual scripts. To support dynamic inclusion of scripts, scripts with a valid nonce can propagate trust to recursively included scripts via the 'strict-dynamic' source-expression. This source-expression voids any allowed hosts for script inclusion, forcing the exclusive use of nonces / hashes to control scripts.

Other popular use cases of CSP include *framing control* and *TLS enforcement* [28]. To support the former, CSP introduced the frame-ancestors directive, which defines an allowlist for framing. For the latter, CSP supports two useful directives: upgrade-insecure-requests forces an automated upgrade from HTTP to HTTPS for all resources loaded by the page, while block-all-mixed-content strengthens the traditional mixed content policy implemented by major browsers to rule out all forms of HTTP communication from HTTPS pages. Note that upgrade-insecure-requests effectively subsumes block-all-mixed-content.

X-Frame-Options X-Frame-Options (XFO) is one of the oldest security headers, originally introduced in Internet Explorer to defend against click-jacking attacks by enforcing a framing control policy. Though now deprecated in favor of the frame-ancestors directive of CSP, XFO is still massively deployed in the wild [7].

In modern browsers, XFO can be set to two different values: SAMEORIGIN allows framing only on pages with the same origin of the framed content; DENY instead forbids any form of framing. XFO also used to support a third option, called ALLOW-FROM, which could be used to allow framing only from a given URL, however major browsers like Chrome and Firefox do not support it anymore or even never supported it.

HTTP Strict Transport Security Although the Web is fast progressing towards a full usage of transport layer encryption through TLS, browsers do not automatically upgrade all connections to HTTPS to avoid breakage. This introduces the danger of attackers who force a victim's browser to make a request towards the HTTP version of a site, thus allowing the attacker to perform impersonation attempts, e.g., for phishing or to sniff non-Secure cookies.

To prevent these threats, *HTTP Strict Transport Security* (HSTS) was introduced. In particular, once set for a specific HTTPS host via the Strict-Transport-Security header, any connection towards that host is automatically upgraded to HTTPS by the browser for the duration specified in the max-age attribute (or until a HSTS header with max-age set to 0 is received). Optionally, HSTS can specify the includeSubDomains directive, which extends protection to all the subdomains of the host setting the security header. This is important to defend against network attackers who could otherwise forge cookies from HTTP subdomains and to prevent the exfiltration of domain cookies lacking the Secure attribute.

Due to its design, HSTS faces a Trust-On-First-Use (TOFU) problem because network attacks can be performed before TLS connections are enforced via the Strict-Transport-Security header in the first response. In order to get rid of this issue, hosts supporting HSTS can also ask for inclusion in the HSTS preload list [26], which is a public list of known hosts where browsers should activate HSTS by default. To be accepted into the HSTS preload list, a host must serve a valid HSTS header with max-age set to at least one year, have enabled includeSubDomains, and include the preload directive. Since preloading implies a fully functional HTTPS setup for all of a site's subdomains, which might cause problems in practice, the preload list offers a feature for removal. For this removal request to go through, a site has to serve a valid HSTS header (i.e., at least specifying a max-age value) *without* the preload directive [27]. After this, the site is removed from the preload list without further notice to the site operator.

2.2 Related Work

We categorize related work in three key areas: client-side Web security, Web security inconsistencies and Web measurements from different vantage points.

Client-Side Web Security Client-side Web security received an increasing amount of attention by the research community in the last few years. Prior research studied the adoption of different client-side security mechanisms, including cookie security attributes [5, 39], CSP [6, 28, 43] and HSTS [22, 33]. Stock et al. [37] investigated the historical evolution of the most popular client-side security mechanisms using archival data. However, none of these works focused on client-side Web security inconsistencies, because they analyzed the considered security mechanisms using a single, fixed client with a specific network access method.

Web Security Inconsistencies Other related studies are those on Web security inconsistencies, i.e., conflicting configurations of protection mechanisms leading to insecurity. This problem has been explored from different angles. A first work investigated inconsistencies between the desktop and the mobile version of the same site [21]. This work is largely complementary to ours, because it analyzes vulnerabilities enabled by security inconsistencies between a desktop site like `www.foo.com` and its mobile variant `m.foo.com`. Our analysis instead disregards such cases, because `www.foo.com` and `m.foo.com` are not necessarily the same Web application, hence they might legitimately have different security requirements. Our methodology (see Section 5) is designed to minimize false positives by taking the specific enforcement models of existing security mechanisms into account and focuses on security inconsistencies in the same Web application enabled by a wide range of different client characteristics (including the use of a mobile client, but not limited to that). Similar considerations apply to an analysis of HTTPS security mismatches between `www.foo.com` and its parent [1].

Another complementary piece of work includes a large-scale study of how different User-Agents enforce the same click-jacking protection policy differently [7]. This is a different form of Web security inconsistency coming from the variegate cross-browser support of specific security mechanisms [31, 32], something which we abstract from by assuming the use of a modern, fully compliant client. A last work on Web security inconsistencies is the recent Site Policy proposal, designed to tame inconsistent configurations of the same security mechanism across different pages of the same site, which is yet another orthogonal security issue [8].

Web Measurements Prior work proposed the use of multiple vantage points for Web measurements [17, 18] and several papers measured the impact of different vantage points on specific aspects. Notable examples include papers analyzing how geolocation may affect the behavior of Web trackers [13, 14, 30, 40] and the security guarantees of HTTPS [29]. The latter work also analyzes downgrades in the use of security headers as part of a broader study, though its treatment is not nearly as comprehensive as that of our paper: it considers a more limited set of headers and use cases, it only covers a specific type of inconsistency (lack of header) and it only focuses on a single factor (geolocation). Other work studied suspicious content manipulation performed by free proxies, VPNs and middleboxes [12, 16, 19, 20, 24, 41, 42]. These papers identified a range of malicious behaviors and shady practices, including script injection, cookie injection, TLS downgrades and generic header manipulations. Orthogonally, we are only interested in server-side responses *without* any network manipulation, i.e., we measure the impact of different legitimate access methods on Web application security.

3 Motivation for Our Study

In contrast to prior work, we measure client-side Web security across different client characteristics to identify specific conditions that attackers might exploit to identify weak spots in protection. We discuss a few relevant examples below.

A Web site may set the Secure attribute on its session cookies when it is accessed using Chrome, but may forget the attribute when it is accessed using Opera. This would leave Opera users vulnerable to cookie sniffing attempts over HTTP, which may enable session hijacking attacks against a specific user population. In terms of exploitation, a network attacker could just try a traditional cookie leak attempt, e.g., injecting an HTTP image pointing to the target site, and profit from the presence of Opera users on the network under her control.

A Web site may configure its CSP differently when accessed from different countries, e.g., due to the use of different ad networks, and there is no guarantee that all these CSPs enjoy the same security guarantees. For example, users from specific countries may be left unprotected against XSS, so a Web attacker might attempt targeted attacks where a link containing an XSS payload is only shared on social media platforms which are most popular on the vulnerable country.

Additionally, security mechanisms might change when sites are accessed from different geolocations. These can either occur because a visitor originates from a specific country or because they rely on a VPN or the Onion network to spoof their geolocation. Both the geolocation and the fact that a user is connecting through the Onion network may have an impact on security, which can be leveraged by an attacker if these changes are deterministic based on the (spoofed) geolocation.

An orthogonal issue is that sites may change security headers in a non-deterministic fashion. Indeed, we observed different security policies on the same Web page even when the page was accessed under the very same client characteristics, due to non-deterministic factors like load balancers. This means that users may occasionally enjoy different levels of protection even when no observable condition changes, hence in our threat model we also consider determined attackers who actively try to abuse such inconsistencies opportunistically.

In the following two sections, we lay the ground work for our analysis. First, we explain our measurement setup and chosen client characteristics, as well as how we selected sites to ensure a measurement without network-level interference. Second, we formalize a definition of consistency and apply it to the previously outlined security mechanisms.

4 Data Collection Framework

Here, we discuss which types of factors we investigate to understand differences in client-side Web security guarantees. We then outline which information we use to collect data for each of the factors and we discuss our key design choices.

4.1 Scope of the Study

We assume the use of a modern client implementing all the security mechanisms in Section 2, e.g., the latest versions of Chrome and Firefox as of January 1, 2022. We thus exclude the use of legacy clients, because it is clear that this discouraged practice may severely downgrade Web application security, e.g., because CSP is not supported by the browser. We also assume that modern clients correctly implement all the security mechanisms according to their official specifications, i.e., if two different clients receive the same security headers, we assume they enforce the same intended level of protection. Finding bugs in the implementation of client-side security mechanisms is an orthogonal issue [32].

We identify three *factors* which users may legitimately manipulate as part of their everyday Web browsing experience, without realizing that they can unintendedly affect Web application security:

1. User-Agent: users have different tastes and might prefer different browsers, e.g., due to the privacy policies they implement by default. Moreover, the same user might use different browsers on different devices, possibly running different operating systems. As long as users make use of a modern, up-to-date browser, they likely do not

Factor	Set of tests	Tests
User-Agent	Windows client	User-Agent header: Chrome 96, Firefox 95, Edge 96, Opera 82
	Linux client	User-Agent header: Chrome 96, Firefox 95, Opera 82
	macOS client	User-Agent header: Chrome 96, Firefox 95, Edge 96, Opera 82, Safari 15.2
	Android client	User-Agent header: Chrome 96, Firefox 95, Opera 96
	iOS client	User-Agent header: Chrome 96, Firefox 95, Edge 86, Safari 15.2
Vantage Point	VPN service	Servers from `hidemyass.com` - 1 per country (218 countries)
	Onion network	Standard Onion client - 1 end-node per country (49 countries)
Client Configuration	Language	Accept-Language header: en, es, cn, ru, de

Table 1: Selected client conditions that might influence the received security headers

expect Web application security to be affected, yet it is possible that a site sets up different configurations based on the value of the User-Agent header of the incoming requests for generic reasons. This practice, known as *User-Agent sniffing*, might leave some User-Agents unprotected against specific classes of attacks. Note that we use the terms *browser* and *User-Agent* interchangeably.

2. Vantage Point: users may access a given site from different geographical vantage points. Users may not expect this practice to affect Web application security, yet it is possible that the geolocation has an impact on dynamically loaded advertisement, which in turn might require a different server-side configuration of CSP. Further, the exit nodes of the Onion network are publicly known, hence a connection through the Onion network might result in a different response, possibly introducing a difference in security.
3. Client Configuration: some configuration settings might influence the way clients interact with Web sites. For example, the language of the client is normally advertised in the Accept-Language header and the site might use this information to redirect the client to a localized homepage served by a different host, possibly with a different security configuration. We only focus on this aspect (language) in our analysis for simplicity.

Table 1 identifies for each factor a set of possible *tests*, which can be easily simulated in a black-box fashion by a Web crawler. We identify the respective User-Agent strings for the browsers in the table via a public online repository [45].

4.2 Challenges and Design Choices

The discussion in the previous section does not directly yield a dataset construction procedure, due to a couple of problems we have to deal with. The first is related to the sheer *number of requests* to send to each Web site, because it is possible that security policies are influenced by a combination of multiple factors. To mitigate this, we only cover a subset of all the possible combinations by testing different factors in isolation:

1. User-Agent: when testing different User-Agents, we access the network through a local machine with a German IP, and we do not set the Accept-Language header.
2. Vantage Point: when testing different vantage points, we set the User-Agent header to Chrome 96 for Windows and we do not set the Accept-Language header. We use hidemyass VPN[1] to access the sites from 218 countries and additional 49 different countries through for the Onion network.
3. Client Configuration: when testing different language settings, we access the network through a local machine with a German IP address and we set the User-Agent header to Chrome 96 for Windows.

This way, we can measure meaningful *intra-factor* variations in the level of Web application security, e.g., by estimating the impact of the choice of a specific User-Agent when the other two factors are fixed.

The second problem is related to *non-determinism*, because the same request does not necessarily always receive the same response. For example, DNS might resolve the same hostname to different IP addresses at different times and load balancers can forward requests for the same resource to different backend servers to improve performance. In either case, there is no guarantee that all the hosts which might process the request enforce the same security policies. Security inconsistencies introduced by non-deterministic factors are in the scope of our study, yet they complicate the *attribution* of security flaws. To exemplify the problem, assume that Chrome appears to be less protected than Firefox because it did not receive any security headers at all. In this case, the User-Agent itself may not be the actual cause of insecurity because non-determinism may have played a role on the received response, e.g., the DNS resolution accidentally redirected Chrome to a poorly configured host. To mitigate this problem, each Web site is visited multiple times (five in our collection) for each test and all the corresponding responses are stored, which allows us to detect non-deterministic security inconsistencies.

Our crawler takes as input a set of *Factors*, a set of *Tests* associated to each factor, a set of *URLs* to access and a number of visits n to perform for each test. For each factor $f \in \mathit{Factors}$ and each associated test $t \in \mathit{Tests}[f]$, each $u \in \mathit{URLs}$ is visited n times setting f to t. For the finally reached URL (after potential redirects), we resolve its origin o and save the response r in the dataset at the entry $D[u,t]$, enriched with the origin o. We refer to o as the *end origin* of the response r.

[1]https://www.hidemyass.com/

5 Formalizing Inconsistencies

Before presenting the formal details, we present an overview of our analysis methodology to explain its design and subtleties. A simple notion of consistent security might be as follows: all the responses collected from the same URL must enforce the same security policies. However, this intuitive definition of consistency is too strong to be useful in practice.

The first point we make is that requiring the *same* security policies for all the collected responses is overly restrictive. As a matter of fact, two security policies can be *syntactically* different, yet provide an equivalent level of protection. For example, two syntactically different CSPs may both effectively mitigate the dangers of XSS. As another example, a host may configure HSTS with tiny fluctuations in the value of the max-age attribute which do not play any role in terms of practical security. To abstract from syntactic differences without significant security implications, we define *equivalence* relations $\sim_m$ for each security mechanism m under study.

A second challenge of our analysis is related to *legitimately* different policies we might get for different client characteristics: for example, a Web site which activates CSP for desktop clients might redirect mobile clients to a static error page which requires no protection and thus enforces no CSP. We do not want to consider these cases as security inconsistencies, because the Web pages are different and legitimately require a different level of protection. To filter out false positives, we define *compatibility* relations $\bowtie_m$ for each security mechanism m under study: incompatible responses cannot lead to security inconsistencies, because m protects different objects. For the sake of generality, our framework supports different compatibility relations for different security mechanisms, because they may be based on different enforcement models, e.g., CSP operates at the page level, while HSTS operates at the host level. In our Web measurement, however, we use the same compatibility definition $\bowtie_m$ for each security mechanism m, because we want to be conservative in our findings and avoid over-reporting (see Section 5.2).

Finally, we observe that not all inconsistencies are equal in terms of real-world *exploitation*. Inconsistencies enabled by non-deterministic factors can be exploited by attackers who are determined (or lucky) enough to eventually stumble into them, while inconsistencies enabled by deterministic factors like the adoption of a specific User-Agent identify weak spots that knowledgeable attackers can more easily take advantage of. We discriminate these two cases by having two different definitions of consistency, as detailed below.

5.1 Consistency

For any security mechanism m, we assume a reflexive and transitive relation $r \lesssim_m r'$ reading as: response r configures m no more securely than response r'. We write $r \sim_m r'$ if and only if r configures m equivalently to r', i.e., we have that both $r \lesssim_m r'$ and $r' \lesssim_m r$ hold. Finally, we assume reflexive and symmetric relations $r \bowtie_m r'$ reading as: response r and r' are compatible with respect to the security mechanism m. We later instantiate $\lesssim_m$ and $\bowtie_m$ to the different security mechanisms considered in our study to capture specific security properties.

Chrome 96	Firefox 95
H,H,H,H,H	H,H,H,H,H
H,H,H,L,H	H,H,H,H,H
H,H,H,H,H	L,L,L,L,L

Table 2: Example observations upon crawling

The first definition we introduce is called *intra-test consistency*. It requires all compatible responses collected within the same test to provide an equivalent level of protection. Violations to this consistency property are likely attributed to non-deterministic factors, because all the observable client conditions are the same across the received responses.

Definition 1 (Intra-Test Consistency)**.** The page with URL u satisfies *intra-test consistency* for the security mechanism m within the test t if and only if for all responses $r \in D[u,t]$ and $r' \in D[u,t]$ such that $r \bowtie_m r'$ we have $r \sim_m r'$.

The second definition of consistency which we introduce is called *inter-test consistency*. It requires all compatible responses collected within two different tests (defined for the same factor) to provide an equivalent level of protection. We require the two tests to satisfy intra-test consistency to rule out inconsistencies enabled by non-deterministic factors, e.g., occasionally missing headers on responses collected within the same test. This way, inter-test inconsistencies can be realistically attributed to specific client characteristics.

Definition 2 (Inter-Test Consistency)**.** The page with URL u satisfies *inter-test consistency* for the security mechanism m across the tests t,t', defined for the same factor and satisfying intra-test consistency, if and only if for all responses $r \in D[u,t]$ and $r' \in D[u,t']$ such that $r \bowtie_m r'$ we have $r \sim_m r'$.

We exemplify the definitions at work on a few toy examples. Let us focus on just two tests for the User-Agent factor for simplicity: Chrome 96 for Windows and Firefox 95 for Linux. Assume that pages are visited five times for each test and may be classified in two security levels: low (L) and high (H) with $L \lesssim_m H$. Consider now the example observations in Table 2, that we assume to be all pairwise compatible. The first row models a straightforward scenario where both intra-test consistency and inter-test consistency are satisfied. The second row models a scenario where intra-test consistency does not hold, due to the L observation for Chrome, hence inter-test consistency is undefined: this case captures a non-deterministic security downgrade. The third row represents a scenario where intra-test consistency is satisfied, but inter-test consistency is not: this captures a deterministic security downgrade occurring when the page is visited using Firefox.

5.2 Compatibility Relations

Arguably, certain security mechanisms such as CSP are not applicable to an origin per se, but rather to the *content* provided under a given URL. However, not every URL returns the same content on each load, in particular in the presence of errors or block pages. For such pages, which might originate from CDNs like Cloudflare, enforcing the CSP of the original page might not make sense. Hence, when we encounter an inconsistency, we need to ensure that this inconsistency is not due to different content being delivered. We leverage a *similarity score* on Web pages for this task.

Page Similarity Based on preliminary analyses of the collected data, our page similarity score takes into account four factors: first, we rely on JavaScript as a proxy to implement the pages' functionality. Therefore we created sets of the hosts from which scripts are loaded and computed their Jaccard similarity. By manually investigating the script data that we got from analyzing our responses, we encountered cases where the Jaccard similarity of the script hosts was 1, e.g., because the page only used inline scripts. Notably, the number of inline scripts differed significantly; hence, we also consider the number of scripts for each host as a second factor for our similarity. However, these first two factors do not work well for pages that only rely on a few scripts or do not even use JavaScript on their main page. To lower the impact of this, we manually investigated those pages, and we observed that the title of the page often changes in case of errors, e.g., showing just `domain.com`. We, therefore, compute the longest common substring between the titles of two documents and compute the ratio of this overlap as our third factor. In addition to that, we observed that the response size also differed between error/block pages and pages with content. Thus we define the content size of the response as our fourth factor by assigning a value between 0 and 1 (indicating the relative sizes). We finally combine our factors by computing their average. The resulting value (between 0 and 1) is then used to determine the similarity between two pages. We consider two pages as similar if their similarity score is at least 0.8.

To find the page similarity threshold, we computed the similarity score of pages where we have seen *syntactically* different security headers after normalization (e.g., normalizing CSP nonces or report URLs). Specifically, for each such case, we took the largest response as the baseline (under the assumption that content pages are larger than error pages). Then, we computed the similarity to this baseline for each of the other responses. Figure 1 shows the result of this, both as a histogram (bucket size 0.05), as well as the CDF for the entirety of comparisons. We find that the peak of the histogram is in the right-most bucket, i.e., similarity above 0.95. In the CDF, we observe that the similarity for most of the cases in that bucket is even beyond 0.98. Moreover, the shallow slope of the CDF around 0.8 leads to taking this value as a candidate

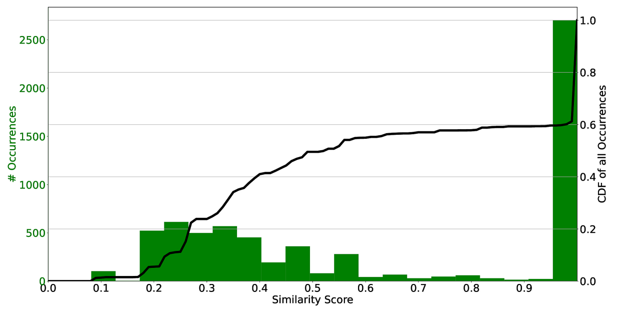

Figure 1: Histogram and CDF of the similarity values for syntactically different header values.

for a threshold to distinguish between content and error/block pages. Notably, error or block pages are just one instance of a potential difference. Our notion generally discerns dissimilar pages, which could require differing levels of protection, so as to avoid reasoning about security inconsistencies when these are in fact legitimate.

To assess the effectiveness of the threshold, we performed analyses to identify false negatives (similar pages marked as dissimilar) and false positives (dissimilar pages marked as similar). To confirm that pages below our threshold are indeed no content pages, we spawned Chromium instances to render those pages *below* the threshold and take screenshots (after 5s) for further analysis. We then manually looked at those 1,939 cases: for the very vast majority, the pages were clearly error pages or block pages showing information about robot detection / CAPTCHAs. Among the edge cases that were close to the threshold, we found one site where the page skeleton looked like the actual content pages for the domain, but without any content. We confirmed this case as a true negative, because the page was under the threshold and seemed different in terms of the content.

Another case that was close to the threshold was a domain that randomly showed a slightly different page that additionally included items in sale. Therefore their title changed from *Mercado Libre Argentina* to *Hot Sale 2021*, and due to the additional content, the file size increased, bringing the similarity down to 0.78. Nevertheless, both pages belonged to the same application and were not empty, error, or block pages, and hence we consider this case as a false negative. Notably, we only faced this one false negative in our dataset.

In order to also assess the number of false positives, we investigated the similarity of pages above the threshold and below 0.95. For the remaining cases beyond 0.95, we are confident to have no error pages in there, given the significant overlap through all four metrics. By looking at those positive (similar) pages, we have seen one single false positive. This case had a similarity score of 0.82, although one HTML file was clearly an error page. This happened due to the error page also including the scripts from the content page. In addition,

False Negatives	False Positives
1/1,939 (0.05%)	1/93 (1.08%)

Table 3: False positive & false negative rates for similarity

the pages were similar in size, yet different in the title (*Hosting Platform of Choice* vs *404 Error | cPanel*). Because this is clearly an error page, we manually removed this error case from our results. Except for this one case, we could not spot any false positives in the pages above 0.8.

Our experiments show that the chosen threshold is appropriate to reason about inconsistencies, because it might suffer from occasional false negatives, but produced just one false positive and one false negative in our experiments (see Table 3). This means that we might lose some inconsistencies, but we are confident not to incorrectly report on inconsistencies where there are, in fact, none (because the content to be protected is different). We now use this *page similarity* notion, to define the following compatibility relations:

Compatibility Given two responses r and r', we let $r \bowtie_m r'$ if and only if the end origins of r and r' are the same and, additionally, the page similarity between r and r' reaches the stipulated threshold. Note that we use the same compatibility relation for each security mechanism m. This may be overly conservative for host-based security mechanisms like HSTS, because different pages under the same host may enforce different HSTS policies for the same object (host), hence one may legitimately disregard page similarity in the compatibility relation for HSTS. However, we empirically noticed that this weaker compatibility notion leads to over-reporting inconsistencies for HSTS. In particular, for sites hosted by CDNs, depending on our vantage point or frequency of requests, we received block or CAPTCHA pages. For Cloudflare, these lacked HSTS. However, it can be argued that this has no significant security implications. In particular, if the site normally uses HSTS, the browser will likely be aware that communication should be performed over HTTPS and the lack of the HSTS header does not deactivate HSTS. For this reason, we prefer to be conservative in our analysis and reuse the same compatibility relation (with page similarity) for all the security mechanisms to avoid potential over-reporting.

5.3 Equivalence Relations

We now define the $\lesssim_m$ relations ("no more secure than") for the different security mechanisms considered in the paper, leading to corresponding security equivalence relations $\sim_m$. These definitions are motivated by the semantics of the security mechanisms under study.

Cookie Security Attributes Defining inconsistencies for cookie security attributes is straightforward, because the HttpOnly and Secure attributes require no configuration, while the SameSite attribute has three different configurations with increasing level of protection: None, Lax, Strict.

We identify cookies with the triple including their name, Domain and Path, as mandated by the corresponding RFC [3]. Formally, we let $r \lesssim_{ck} r'$ if and only if all cookies c occurring in both r and r' satisfy the following conditions:

1. If c is marked as HttpOnly in r, then c is marked as HttpOnly also in r'.
2. If c is marked as Secure in r, then c is marked as Secure also in r'.
3. If c is marked as SameSite in r, then c is marked as SameSite also in r' with at least the same level of protection, e.g., if the SameSite attribute of c is set to Lax in r, then it must be set to Lax or Strict in r'.

Content Security Policy Defining inconsistencies for CSP is more complicated, since it is an expressive security mechanism, which supports many use cases and can thus be analyzed from multiple angles. To address this, we build multiple equivalence relations for CSP to cover different use cases [28].

A first use case for CSP is XSS mitigation, which we study by leveraging a definition of *safe CSP* for CSP Level 3 [8]. This definition ensures that the CSP puts some meaningful restrictions against XSS: policies which do not comply with the definition can be trivially bypassed by an attacker upon any content injection.

Definition 3 (Safe CSP [8]). A CSP is *safe* if and only if it contains a script-src directive (or a default-src directive in its absence) bound to a value v satisfying both the following conditions:

1. v does not contain the 'unsafe-inline' source-expression, unless nonces or hashes are also present in v.
2. v does not contain the wildcard * or any full scheme from the following: http:, https:, data:, unless 'strict-dynamic' is also present in v.

We let $r \lesssim_{csp-xss} r'$ if and only if, whenever r sets a safe CSP, then also r' sets a safe CSP.

The second use case for CSP is framing control. To define an equivalence relation for this use case, we divide responses in four classes based on the enforced framing restrictions:

1. Framing is allowed on all origins.
2. Cross-origin framing is allowed only on selected origins.
3. Only same-origin framing is allowed.
4. Framing is not allowed on any origin.

We then let $r \lesssim_{csp-frm} r'$ if and only if the class of r is less than or equal to the class of r'.

The last use case for CSP is TLS enforcement. Its equivalence relation is defined by having $r \lesssim_{csp-tls} r'$ if and only if, whenever r activates upgrade-insecure-requests or block-all-mixed-content, then also r' does it. In other words, when r forbids the use of HTTP, then also r' enforces the same security restriction.

X-Frame-Options We just focus on SAMEORIGIN and DENY as possible values of XFO, since ALLOW-FROM is unsupported by the modern clients considered in the present study. This implies that responses can be categorized in just three different classes:

1. Framing is allowed on all origins.
2. Only same-origin framing is allowed.
3. Framing is not allowed on any origin.

We then let $r \lesssim_{xfo} r'$ if and only if the class of r is less than or equal to the class of r'.

Strict Transport Security Defining inconsistencies for HSTS requires some care, due to possible differences in the max-age attribute which arguably have little to no impact in terms of real-world security. Our choice is discriminating four classes of responses, as follows:

1. Responses with max-age set to 0, thus forcing HSTS deactivation for their host.
2. Responses without any HSTS header. These responses do not activate HSTS, but do not forcibly deactivate it.
3. Responses with max-age enforcing protection for less than one year. This practice can be useful, but does not comply with the minimal required duration for inclusion in the HSTS preload list.
4. Responses with max-age enforcing protection for at least one year, qualifying the host for preload list inclusion.

We then let $r \lesssim_{hsts} r'$ iff all the following conditions hold:

1. The class of r is less than or equal to the class of r'.
2. If r sets the includeSubDomains directive, so does r'.
3. If r sets the preload directive, then also r' sets it.

Handling Multiple Headers Careful readers may have noticed that the above definitions assume responses to contain at most one header of each type, yet real-world responses might violate this assumption because headers can be set multiple times. Our dataset still contains at most a single header of each type, because the Requests library used in our data collection folds multiple headers into a single header set to a comma-separated concatenation of their values. For handling of multiple headers, we follow specifications where possible:

- If the same cookie (identified by name, Domain and Path) is set in multiple headers, the last one should be prioritized [3]. We thus normalize the collected headers to reflect this behavior within a single header.
- If a response contains multiple CSP headers, all of them should be enforced at the same time [44]. We thus normalize the collected headers by replacing them with a single header enforcing the conjunction of all CSPs.
- If a response contains multiple XFO headers, the correct browser behavior is undefined in the specification. Since prior research showed that different clients handle multiple XFO headers quite differently [7], we check for syntactic differences if multiple values are present.
- If a response contains multiple HSTS headers, the first one should be prioritized [15]. We thus clean our data to keep just the first HSTS header.

6 Measuring Inconsistencies

We use the data collection framework in Section 4 to collect data from live Web sites and apply the formalization in Section 5 to measure inconsistencies. The focus of our study is to understand inconsistencies caused by the servers of highly ranked sites. To ensure that any data we collect could not be tainted through network proxies or firewalls, we decided to only include sites which were served through HTTPS. Specifically, we visited each of the sites in the Tranco list [25] through `https://site.com` and `https://www.site.com`, disregarding those which were not accessible through HTTPS. Further, for each final URL, we determined if this was still under the same eTLD+1 as the originally visited one and not a localized version, e.g., `https://site.eu`, so as to avoid selecting a site which is actually not highly ranked. As a result, this process yields the list of the 10,000 highest-ranked sites available over HTTPS.

Based on this methodology, we arrived at the set of top 10,000 HTTPS sites based on the Tranco list of January 1, 2022[2]. We ran our first crawl, on which we report in the following, from January 2 through January 4, 2022. To ensure that our measurement was not merely a single measurement which is not repeatable, we ran three more confirmation crawls (January 6, 10, and 14, 2022). For each crawl, we collected between 13,626,145 and 13,742,760 responses. While we focus on the results of the first crawl, the appendix lists the overlap in findings between the first and the respective follow-up crawls (Appendix B), which highlights that our results can be confirmed over multiple crawls within 12 days. To ease the confirmation and reproducibility of our findings we made our crawling and analytics pipeline publicly available [10].

In the following, we outline the key results supported by the analysis of the collected data. We first present a high-level overview of the findings and then discuss security inconsistencies introduced by different factors, as well as the security implications of the inconsistencies.

6.1 Overview of the Findings

Usage Statistics To give an overview of the deployed security mechanisms in the wild, we computed the number of sites which activated a specific security mechanism at least once across our data collection. Table 4 shows the resulting usage statistics for each of the selected security mechanisms in the second column. Note that this is an aggregate over all different tests, i.e., it combines checks for different User-Agents, vantage points and languages. In total, 8,174 sites made use

[2]Available at `https://tranco-list.eu/list/XVWN`

Mechanism	Usage	# Sites w/ intra-test inconsistencies					# Sites w/ inter-test inconsistencies					# Sites w/ only inter-test inconsistencies				
		UA	Lang.	VPN	Tor	**Any**	UA	Lang.	VPN	Tor	**Any**	UA	Lang.	VPN	Tor	**Any**
Content Security Policy	1,998	12	11	31	23	36	15	-	29	18	47	15	-	11	3	28
- for XSS mitigation	360	1	-	1	1	3	9	-	1	1	10	9	-	1	-	10
- for framing control	1,288	6	5	15	9	16	2	-	16	5	20	2	-	9	1	12
- for TLS enforcement	661	7	7	19	14	22	4	-	12	12	17	4	-	1	2	6
X-Frame-Options	5,692	20	18	43	22	50	7	-	29	13	37	7	-	9	5	20
Strict-Transport-Security	4,562	15	13	28	23	38	8	-	23	16	35	8	-	12	5	22
- w/o page similarity	-	*42*	*33*	*148*	*593*	*693*	*19*	*2*	*576*	*218*	*643*	*17*	*2*	*524*	*20*	*552*
- preload	920	3	3	6	6	10	-	-	9	4	10	-	-	6	-	6
↳ w/o page similarity	-	*5*	*6*	*20*	*113*	*124*	*1*	*1*	*124*	*48*	*137*	*1*	*1*	*117*	*2*	*119*
Cookie Security	3,876	10	9	11	12	16	150	1	13	8	167	149	1	9	2	160
- Secure attribute	2,937	4	4	5	6	8	144	-	8	3	152	144	-	7	1	151
- SameSite attribute	788	5	5	5	6	7	6	1	4	4	14	6	1	2	-	9
- HttpOnly attribute	3,104	1	-	2	2	3	2	-	3	2	6	2	-	2	1	5
Any	8,174	51	44	103	75	127	177	1	82	49	267	174	1	34	12	194
Any (incl. HSTS w/o similarity)	*8,174*	*77*	*64*	*222*	*634*	*765*	*188*	*3*	*631*	*252*	*833*	*183*	*3*	*541*	*26*	*429*

Table 4: Detected intra-test and inter-test inconsistencies by factor (321 sites in total). We present the numbers with and without page similarity for HSTS to highlight the impact of this choice on the measurement.

of any of the security mechanisms. The most widely used mechanism was X-Frame-Options with 5,692 occurrences. HSTS was used on 4,562 sites, whereas at least one cookie was configured with any of the security attributes on 3,876 sites. The vast majority of these cases stem from the usage of HttpOnly or Secure attributes, with only 788 sites making use of SameSite cookies. The least widely used header was Content-Security-Policy with 1,998 sites which deployed it. Notably, the vast majority of sites used CSP for framing control rather than for its original purpose of XSS mitigation [35]. It is worth noting that for XSS mitigation, we only count those cases which have a policy that is not trivial to bypass (Definition 3). Since our definition of inconsistency revolves around such policies, any site that did not have any meaningful XSS mitigation is not counted. Note that the number of sites for the subclasses of CSP and cookie security do not add to the overall usage, since a site may, e.g., configure a CSP that both mitigates XSS and enforces TLS.

Detected Inconsistencies In total we detected some inconsistency in 321 sites. Table 4 further shows three groups of columns: intra-test inconsistent, inter-test inconsistent, and *only* inter-test inconsistent sites. For the final column group, we removed all those sites for which we found an intra-test inconsistency for the given mechanism. This is to ensure that a site exhibiting a non-deterministic behavior is not accidentally flagged as suffering from inter-test inconsistencies, as required by our formal definition. Hence, the last column is a confident lower bound for the number of sites affected by inter-test inconsistencies. Overall, our crawl detected 127 sites which have some type of intra-test inconsistency and from 194 to 267 sites with inter-test inconsistencies. Notably, our confirmation crawls exhibited two interesting phenomena, i.e., the instability of intra-test inconsistencies and the stability of inter-test inconsistencies. Considering the union of all sites that suffered from intra-test inconsistencies at least once in our crawls, we found a total of 210 sites (Appendix Table 5). This likely means that the actual dangers of non-deterministic intra-test inconsistencies is more severe than what we could measure through our five observations. Conversely, the confirmation crawls showed that the number of sites with inter-test inconsistencies is stable over time (Appendix Table 6).

6.2 Intra-Test Inconsistencies

Intra-test inconsistencies come with particular security risks, as an attacker can abuse these to attack users opportunistically. In the following, we present case studies of intra-test inconsistencies for every mechanism and explain the corresponding security implications.

Cookie Security If a cookie may non-deterministically lack the HttpOnly attribute, an attacker could steal the cookie via malicious JavaScript by performing an XSS attack multiple times until access to the cookie succeeds. One of the three sites where we encountered this kind of inconsistency sets its authentication cookie named *authcookie_loggedIn* sometimes with and sometimes without the HttpOnly attribute.

A non-deterministically missing Secure attribute, as it happened on eight sites, allows a network attacker to steal the corresponding cookie. One site for example non-deterministically set their *csrfToken* cookie as Secure or not. Thus, attackers can steal this cookie and perform CSRF attacks because they know the anti-CSRF token. A similar issue happens on another site, for which the Secure attribute is inconsistently set on the session identifier *JSESSIONID*, thus potentially leading to session hijacking.

In seven sites, we found intra-test inconsistent deployments of the SameSite attribute, which is sometimes set to Lax and sometimes missing. This behavior might not be a problem for modern browsers, because Chromium-based browsers default to Lax in case of a missing SameSite attribute. However, all Safari-based browsers still face the problem that cross-site attacks such as CSRF are possible due to this misconfiguration. One site, for example, sometimes set their *ASP.NET_SessionId* cookie with SameSite attribute set to Lax, and sometimes the SameSite attribute was not set, which enables an attacker to perform attacks such as CSRF.

Content Security Policy Overall, we found three sites for which XSS mitigation was enforced non-deterministically. For example, the responses from one site sometimes did not have any CSP for clients from Germany or Australia. Thus, an attacker can succeed by performing the attack multiple times until one of the responses does not carry a CSP.

For framing control, we found a total of 16 inconsistent sites across our tests. Note that the majority of inconsistencies were detected in the VPN crawl. This is because, in the VPN crawl, we test from 218 vantage point (compared to 49 tests for Onion, and 20 and five respectively, for User-Agent and language), which increases the chances of eventually getting an inconsistent response. For example, a site non-deterministically deployed frame-ancestors or not, hence an attacker can perform the attack multiple times (or load the target in multiple iframes) until the attack succeeds.

Finally, TLS was inconsistently enforced on 22 sites. For example, one site in our dataset deploys a CSP that aims to enforce TLS. However, irrespective of the factors that we checked, this CSP is not present in some responses. Nowadays, Chromium-based browsers auto-upgrade mixed content [4], whereas Firefox and Safari merely block it. Therefore, the security implications of missing TLS enforcement is limited. However, inconsistencies in this feature can lead to functionality issues. In 2020, Roth et al. [28] showed that 77/251 sites which use CSP for TLS enforcement have HTTP resources linked from their front page. Thus, these inconsistencies might lead to essential resources being blocked in Firefox and Safari.

X-Frame-Options The most common intra-test inconsistency for X-Frame-Options was alternating between a deployed header and not deploying XFO at all (41 sites). This behavior enables an attacker to attempt the attack multiple times (or load the target in multiple iframes) until it succeeds. For the other nine cases, the Web applications alternate between a valid XFO header and a malformed one (e.g. sometimes prepending a `:` to its XFO header) or one using an unsupported feature such as ALLOW-FROM. As with omitting the header, an attacker can opportunistically exploit this.

Strict Transport Security Of the 38 sites with intra-test inconsistencies on HSTS, only six are present in the preload list, for which the issue has no implication on the client's security. For 23 sites the inconsistency is related to headers which are sometimes entirely omitted. For those not preloaded, this is problematic since the non-deterministic absence of the header might prolong the time frame where an attack is possible due to the trust-on-first-use problem of HSTS.

While inconsistencies for preloaded sites have no direct impact on a client, they nevertheless pose a threat. In our dataset seven hosts deploy an HSTS header sometimes with, sometimes without the preload directive. Here an attacker can remove the affected site from the HSTS preload list by asking for removal of the site [27]. If the HSTS preload crawler hits a case without preload being present in the HSTS header, the site will be removed without the operator even noticing it. According to our tests with an author-owned preloaded site, there seems to be no rate-limiting in place to stop such abuse.

An intra-test inconsistent deployment of the HSTS includeSubdomains directive can also lead to problems, as it happened for four sites. For example, a payment service provider showed this behavior on several islands (e.g., Falkland Islands, Antigua and Barbuda, Bahamas, and Bermuda). Here an attacker could, in case of a lacking includeSubDomains directive, abuse the subdomains to attack the main domain (e.g., using subdomains to inject cookies into the top-level domain).

The most critical inconsistency in terms of exploitability is a host that randomly alternates between enabled and disabled HSTS, or has multiple enabled and disabled HSTS headers in random order (only the first entry is processed). Three sites have this kind of intra-test inconsistency. They, for example, alternate between `max-age=0, max-age=15768001` and `max-age=15768001`. Since the HSTS specification mandates adhering to the first observed HSTS header, the first case always deactivates HSTS. Similarly, one site non-deterministically disabled HSTS in certain (mostly eastern Europe) countries such as Lithuania, Moldova, Romania, and Ukraine. In both cases, an attacker could perform the attack several times until it succeeds because HSTS has been disabled in the last response. We also identified one site that alternate between a short max-age (<= 5 min) and a properly configured header. In those cases, the site allows an attacker to abuse the HSTS TOFU problem at a higher frequency because if the last HSTS header received was a short one, the next visit after a short time period (e.g., 5 min) will be vulnerable again, because the browser no longer enforces TLS.

Reasons for Intra-Test Inconsistencies In order to find the cause behind the intra-test inconsistencies, we took a closer look into the gathered data from the responses, such as peer IP addresses or cache headers. Here we noticed that indeed some of the different response headers were caused by caching, because all misconfigured header values also had a different `cache-control` header (18 sites) or a different `x-cache` header (five sites). For two sites, our data indicate that depending on the geolocation, we were redirected to a different end URL (under the same origin), which then causes the inconsistency. In the case of five sites, we were also able to attribute the inconsistency to certain peer IP addresses, which indicates that misconfigured origin servers might be the underlying problem. This hypothesis is also supported by one answer from the notification campaign, which indicates that "one of the origin servers seems to be configured differently". However, here we observed that this inconsistency does not depend on the peer IP address, which indicated that what we see as the peer IP might only be a load balancer, sending our request to different origin servers on a back channel. Notably, the probability of getting a different origin server is much higher in the case of different geolocations. This, together with the fact that the number of crawls for VPN and onion are much higher than in the case of user-agent and language settings, explains the comparatively higher number of intra-test inconsistencies.

One inconsistency that is standing out, due to its prevalence in our dataset, was the inconsistent setting of SameSite for the *ASP.NET_SessionId* cookie. According to Microsoft, the framework does not support ".NET versions lower than 4.7.2 for writing the same-site cookie attribute"[23]. Thus, if some of the origin servers have a new version of .NET while others still use the old version, the cookie would show exactly the behavior we observed, which is why we believe this to be a contributing factor.

6.3 Inter-Test Inconsistencies

This section sheds light on the inter-test inconsistencies, i.e., for a single deterministic factor such as the User-Agent, our crawls revealed different security guarantees (see middle column of Table 4).

Cookie Security The vast majority of sites (144/150) that have inter-test inconsistencies for cookie security are those that deterministically gave back cookies without the Secure attribute to some User-Agents. Notably, the cause of this inconsistency is in most cases (130), special handling for the User-Agent for Firefox on iOS. For example, one site set their *sid* cookie Secure for all clients except Firefox on iOS, leaving those clients unprotected against network attackers. Other sites gave non-Secure cookies to a group of User-Agents that visited their page, another site for Safari-based clients, one for mobile clients, and another one for all iOS clients.

Two sites inconsistently deployed HttpOnly cookies for their clients. In one case, a site delivered *CM_SESSIONID* without HttpOnly attribute to clients that use Firefox on iOS. In another case a site only gave out HttpOnly cookies to non-Safari-based clients. In both cases, attackers can steal or manipulate cookies via an XSS attack and eventually perform state-changing actions on behalf of the user.

For inconsistencies of the SameSite attribute, we found 14 cases where sites either send cookies with the attribute or do not set it at all. One site, for example, only gives SameSite cookies if the Accept-Language header of the client is *not* set to English. As mentioned in the intra-test inconsistencies, this behavior might not be a problem in Chromium-based browsers, because those browsers default to Lax in case of a missing SameSite attribute. However, all Safari-based browsers and Firefox still face the problem that cross-site attacks such as CSRF are possible due to this misconfiguration.

Content Security Policy XSS mitigation as the original use-case of CSP also faced inter-test inconsistencies in ten cases. In general, if a site's CSP alternates between a safe policy and a trivially bypassable one based on some client characteristics, an attacker can specifically target the affected user population. Due to the (at the time of writing) porous support for the 'strict-dynamic' source-expression, some sites had inter-test inconsistencies that only deployed a CSP with this source-expression to clients that actually support it. Numerous sites removed 'strict-dynamic' from their CSP for all Safari (and thus all iOS) clients. The problem here is that `https:` is also present in the policy, i.e., clients without support for 'strict-dynamic' would allow script inclusion from any HTTPS host, which is insecure. Removing 'strict-dynamic' is a bad practice, because the CSP design is backward compatible and unknown source-expressions are just ignored by browsers. Importantly, Safari recently announced support for 'strict-dynamic' and already supports it in its technology preview [9], hence dropping 'strict-dynamic' may unduly leave Safari users unprotected. Other sites dropped their entire CSP for XSS mitigation for all Safari clients, while again others did not send a CSP at all for Android clients. One Web site only deployed XSS mitigation to some countries (like Russia, Spain, or Sweden), but did not deploy CSP for others (e.g., US, Pakistan, or South Africa).

CSP for framing control is also used inter-test inconsistently across different clients (two sites) and geolocations (18 sites). For example, one site did not send a CSP controlling framing via frame-ancestors to all iOS clients, leaving those users unprotected against framing-based attacks.

Like for the case of intra-test inconsistent deployment of CSP for TLS enforcement, the inter-test inconsistent deployment of this CSP feature does not have a security impact but a functionality impact. However, while it is a randomly occurring problem for the intra-test inconsistencies, the problem deterministically occurs for parts of the user-base on 17 sites.

X-Frame-Options An inter-test inconsistent deployment of X-Frame-Options exposes a part of the user base to framing-based attacks. In seven out of 37 cases, this type of inconsistency occurred due to specific operating systems or browsers are getting different configurations. Some sites deployed XFO for desktop clients, but mobile browsers got no protection at all, making them vulnerable to framing-based attacks. In other cases specific browsers were excluded from the protection: one site did not deploy XFO for Opera clients, while another excluded Firefox browsers. This behavior was also present against users of a specific operating system, as some sites only gave XFO to non-iOS clients. In addition to that, 13 sites (Onion) and 29 sites (VPN) decided to exclude specific geolocations from the protection against framing based-attacks.

Strict Transport Security In case of inter-test inconsistencies in HSTS it makes no difference if HSTS is disabled (max-age=0) or not present because the affected clients/countries will deterministically get the same insecure configuration. While cross-checking the inconsistent sites with the HSTS preload list, we observed that only five out of the 35 inter-inconsistent sites are actually preloaded.

There are eight Web sites that handle browsers differently. For example, one site only gives enabled HSTS to desktop clients but not to mobile clients, another does not send HSTS to Firefox and Safari-based clients, which exposes parts of the user-base to possible network attacks. In addition to that, 30 sites deploy HSTS inconsistently depending on the geolocation. Another site deploys a proper HSTS for all countries except for clients from India, which do not get an HSTS header. Also, six sites have the inter-test inconsistent deployment of HSTS with/without the includeSubdomains directive. One site, for example, deployed an HSTS header with the directive for clients from some countries such as Hungary or Ireland, but not for others such as Germany or Japan. Here an attacker could abuse subdomains to attack the main domain (e.g., using subdomains to inject cookies into the parent domain).

Reasons for Inter-Test Inconsistencies Inter-test inconsistencies are naturally attributed to deterministic factors, however a few observations are interesting. In many cases flawed User-Agent parsing or wrong handling of the parsed browser information seem to be a problem. Surprisingly many inter-test inconsistencies happened specifically for Firefox on iOS. Therefore we tested this User-Agent in different parsing libraries. All of them showed Firefox with version 40 (released August 11, 2015) as output for our Firefox iOS User-Agent string. In the case of Firefox, the version numbers for the iOS client are different from other operating systems, possibly due to the fact Firefox is based on WebKit instead of Gecko on iOS, so the User-Agent is incorrectly recognized as a legacy client. Notably, the Firefox iOS version number recently jumped from 40.2 to 96.0 on January 18, 2022 [2].

Not only the version number of Firefox, but also the iOS version number present in the User-Agent was the reason for some of the inconsistencies. The User-Agent from an online repository used in our crawler had an old iOS version number (12.1, October 2018). However, with the same Firefox and WebKit version, but a newer iOS version (15.2, December 2021), these inconsistencies were not present, although they are still concerning for a specific user population. Indeed, users may not have control of their OS version due to hardware restrictions.

Also, as mentioned in Section 6.3, some sites deliver a CSP without the 'strict-dynamic' expression to Safari-based clients. During our notification campaign, a videotelephony service confirmed that they are doing this because those clients lack support for this CSP feature. In either case, none of those special handling for browsers is actually necessary; unknown cookie attributes and unknown CSP source-expressions are simply ignored by browsers. Furthermore if certain features are going to be supported in future release (like 'strict-dynamic' in Safari's current Technology Preview), the special handling for certain browsers might cause security issues because the browser switches are not updated or removed. This highlights that having browser switches for security mechanisms is a dangerous practice, at least if the provided level of security differs.

In case of network related inter-test inconsistencies, possible reasons are similar to those from the intra-test inconsistencies. If misconfigured origin servers are only used for requests from specific countries, or if CDNs cache responses for certain countries longer than for others, we can observe inconsistent deployment of security mechanisms depending on the geolocation. We detected three sites with different peer IP adresses that seem to cause the issue, ten sites with different `cache-control` header, and two with different `x-cache`. For example, based on the `x-cache` header sent by one site we hypothesize that for certain countries like France they have a cache in place, because all requests from there produced a cache hit, while other requests for example from Australia only produces cache errors/misses.

6.4 Disclosure

Our findings imply that certain users of the sites under test might be at risk; either because an attacker can target them based on certain properties (e.g., their User-Agent) or can opportunistically exploit the non-determinism of the server. To enable site operators to fix the inconsistencies, but also to gain knowledge about the root cause of the inconsistencies, we attempted to disclose the issues to all sites using *security@* and *webmaster@* aliases. The email that we sent contained information about our institutions and us, as well as a detailed description of the individual inconsistent headers and how they were collected. Also, we informed site operators that we are interested in the reason for the inconsistency such that we can better help others that face similar issues and offered them our assistance and further information.

In total, we sent out 256 emails (see Appendix A for the template). For 197 domains, we received an email delivery failed message. Notably, we sent the email to both aliases (security@ and webmaster@), so we might have received a failure message, although one of the two addresses received our email. Research has shown that scaling up notifications is a known problem [36, 38], also due to the low availability of generic aliases [34]. In addition to that, only 25 out of the 256 domains hosted a `security.txt`, with 7 of those setting their contact email to *security@*. Thus, we only got 21 answers that were more than just an automatic response message. Seven operators asked us to provide more details, like the IP addresses of the servers that we connected to. One of those even asked us to provide a demo video that shows the inconsistency problem. In all cases, we were happy to provide them with more detailed data in order to ease their search for the reason behind the issue. Additional seven claimed that they can confirm the issue and will get right back to us, which nearly none of them have done so far. The other seven answered us that they confirmed and fixed the issue or explained to us that this is out of their control, e.g., because they are not self-hosting their sites in some countries.

Many of those that answered instructed us to contact HackerOne to report vulnerabilities. Notably, our message did not include the word "vulnerability" or similar words like "exploit". Therefore, we answered those emails that we were not interested in any bug bounty, because we only wanted to help and raise attention for the inconsistent behavior such that all clients can be secured consistently. Notably, none of the notified parties answered that this issue is not present in their option, which further strengthens our confidence in the results. The previous *Reasons for* subsections have outlined some of the answers from our disclosure campaign that we used to reason about the inconsistencies in some of the case studies. To increase remediation rates, we tested the problematic sites again in May 2022. Here, we found that 184 still contained the issues we attempted to disclose before. By manually investigating those sites, we were able to find 105 email addresses. In this second round, only four of the manually curated email addresses responded with a failure message.

7 Discussion

Here, we discuss limitations of the work and summarize the security impact of our findings.

7.1 Limitations

Our analysis already shows that client characteristics play a relevant role for Web application security, however it could be improved along different directions. One limitation of our study is the assumption that all the tested browsers implement all the security mechanisms according to their official specifications, which simplified the technical development. This assumption is motivated by our focus on modern clients, yet we are well aware that it is not entirely accurate, e.g., at the time of writing Safari does not support the 'strict-dynamic' source-expression of CSP Level 3 and browsers might suffer from bugs (like all software), especially in corner cases. That said, we manually vetted most of the detected security inconsistencies and we confirm that they are not subtle enough to invalidate the general findings of our study due to our assumption on browser behavior.

Another limitation of our work is the best-effort attribution of the identified security inconsistencies. Discussing correlation rather than causation is a common and accepted limitation of Web measurements. We crawled each page multiple times and formalized different definitions of consistency to mitigate the effects of non-determinism, however we cannot entirely rule out non-determinism, e.g., due to the presence of server-side load balancers. It is possible that we collected five times the same response from a Web page due to non-determinism, rather than due to our testing conditions, however all the cases explicitly named in Section 6 have been manually vetted and confirmed as vulnerable.

7.2 Overall Security Impact

In general, an attacker can abuse the inter-test inconsistent behavior of some sites to attack a certain part of the user base by specifically targeting the less secured clients like specific User-Agents or users from certain geolocations. For the intra-test inconsistent sites, an opportunistic attacker can exploit the non-determinism of the deployed security mechanism by executing the attack multiple times. The individual advantage of the attacker in both cases depend on the mechanism that is deployed inconsistently.

For **Cookies**, a missing security attribute enables an attacker to access the cookie via XSS (*HttpOnly*) or to steal the cookie by downgrading the connection security and eavesdropping on the traffic (*Secure*). Also, missing or inconsistently deployed *SameSite* Attributes allows attackers to successfully execute cross-site attacks such as CSRF. In either case, the difference between inter-test and intra-test inconsistencies in the case of cookies does not change the attack itself but only the way it can be successfully executed, because the attacker either needs to target a certain group of users (*inter*), or perform the attack multiple times (*intra*). Therefore the user-base (or party of it) of more than 172 Web sites can be attacked due to inconsistencies.

In case of an inconsistent **Content Security Policy** header an attacker can perform XSS attacks (inconsistent *XSS mitigation*), framing-based attacks such as Clickjacking (inconsistent *frame-ancestors*), or perform network-based attacks (inconsistent *TLS enforcement*). While the latter is only relevant for functionality rather than security, because Chromium-based browsers nowadays auto-upgrade mixed content [4] and Firefox and Safari block it, the other two cases can indeed be exploited by an attacker. Thus in case of inconsistent XSS mitigation and/or inconsistent framing control, the attacker can exploit a certain group of users (*inter*), or try the attack multiple times (*intra*) on 41 different Web sites.

Similar to the exploitability of inconsistent CSP *frame-ancestors*, inconsistent deployment of the **X-Frame-Options** header can lead to framing-based attacks such as Clickjacking. Notably, however, XFO will be ignored by CSP Level 2 supporting browsers as soon as CSP *frame-ancestors* is present. Still, only ten sites that showed inconsistencies in XFO have deployed a CSP that restricts framing. Thus, the users of 43 sites would still be exploitable by performing the attack multiple times (*intra*), and a specific group of users would be attackable on 17 sites.

For **Strict Transport Security** we have cases that lead to different attacks depending on the type of inconsistency. If, for example, the *preload* directive is deployed intra-test inconsistently, an attacker can remove this site from the HSTS preload list by asking for removal of the site multiple times until the HSTS checker encounters the header without preload. For inconsistencies in the *includeSubDomains* or inconsistent *max-age* duration, an attacker can run network attacks against a certain group of users (*inter*), or perform the attack multiple times (*intra*) on 60 different Web sites. Notably, those sites are only cases where the *page similarity* was considered, so the number of potentially exploitable sites could be higher, as HSTS protects the connection security between client and server, and does not care about the actual content of that is delivered via the server.

8 Conclusion

In this paper we investigated the inconsistent configuration of client-side security mechanisms on top sites across different client characteristics (*inter-test*) or even across multiple communications of the same HTTP request (*intra-test*).

Our measurement has highlighted that client-side security mechanisms are not equally delivered to all clients. Specifically, we found several sites in our dataset that returned different security policies with different semantics in at least some of our tests. Our findings have implications in three dimensions: first, Web users may receive different protection based on subtle differences in their browser or vantage point (*inter*-test inconsistencies). Second, *intra*-test inconsistencies may enable an adversary to launch attacks in an opportunistic fashion, given that the responses for the same request may non-deterministically enforce different security. Acquiring this knowledge is an easy task for the attacker, as they can probe for non-deterministic behavior of the server as we did. Third, our analysis has shown that prior measurements (see related work in Section 2.2) may have inadvertently under- or over-reported findings with respect to the deployment of security mechanisms. Specifically, we identified intra-test security inconsistencies in 127 sites and inter-test security inconsistencies in 194 sites. Our semantics-based analysis gives clear evidence of the potential security implications of the detected inconsistencies, by identifying characteristics which might enable exploitation, while being expressive enough to generalize over previous studies which only focus on missing security headers [29].

To the best of our knowledge, we are the first to systematically study the problem of intra-test inconsistencies. Luckily, dealing with such inconsistencies in Web security measurements appears relatively easy: since most of them (80.4%) are due to unexpectedly missing headers, it suffices to crawl the same page multiple times to detect and fix these omissions. Nevertheless, prior Web measurements on the impact of client characteristics on Web security and privacy might have performed an incorrect attribution of security downgrades, since a single page access does not suffice to assess the impact of non-determinism. Luckily, the number of sites suffering from intra-test inconsistencies is not high enough to invalidate the big picture drawn by prior studies.

Inter-test inconsistencies are likely less surprising to researchers working on Web measurements, due to the publication of papers studying variations of the topic [17, 18]. However, inter-test inconsistencies are particularly concerning to site operators, because they identify weak spots in their security policies reported by our analysis. We observe that inter-test inconsistencies across network access methods might arise due to misconfigured origin server for specific geolocations. Also, User-Agent sniffing leads to security inconsistencies on 177 sites, which can all be attributed to site operators. Notably, due to backwards compatibility of the investigated security mechanisms, none of the individual responses for specific browsers were actually necessary.

Acknowledgments

We would like to thank the reviewers for their advices on how to improve the presentation and reproducibility of our paper. In particular, we thank Yinzhi Cao for his guidance during the shepherding process.

This work was conducted in the scope of a dissertation at the Saarbrücken Graduate School of Computer Science and was partially supported by DAIS - Università Ca' Foscari Venezia within the IRIDE program.

References

[1] Eman Salem Alashwali, Pawel Szalachowski, and Andrew Martin. Does" www." mean better transport layer security? In *ARES*, 2019.

[2] Mozilla Mobile Applications. Releases of Firefox-iOS. GitHub.

[3] Adam Barth. RFC6265: HTTP State Management Mechanism. 2011.

[4] Chromium Blog. No More Mixed Messages About HTTPS. chromium.org.

[5] Michele Bugliesi, Stefano Calzavara, Riccardo Focardi, and Wilayat Khan. Cookiext: Patching the browser against session hijacking attacks. *Journal of Computer Security (IOS Press)*, 2015.

[6] Stefano Calzavara, Alvise Rabitti, and Michele Bugliesi. Semantics-based analysis of content security policy deployment. *ACM TWEB*, 2018.

[7] Stefano Calzavara, Sebastian Roth, Alvise Rabitti, Michael Backes, and Ben Stock. A tale of two headers: A formal analysis of inconsistent click-jacking protection on the web. In *USENIX Security*, 2020.

[8] Stefano Calzavara, Tobias Urban, Dennis Tatang, Marius Steffens, and Ben Stock. Reining in the web's inconsistencies with site policy. In *NDSS*, 2021.

[9] CanIUse.com. headers HTTP header: csp: Content-Security-Policy: strict-dynamic. CanIUse.com.

[10] CISPA. The Security Lottery. GitHub.

[11] Kostas Drakonakis, Sotiris Ioannidis, and Jason Polakis. The cookie hunter: Automated black-box auditing for web authentication and authorization flaws. In *ACM CCS*, 2020.

[12] Zakir Durumeric, Zane Ma, Drew Springall, Richard Barnes, Nick Sullivan, Elie Bursztein, Michael Bailey, J Alex Halderman, and Vern Paxson. The security impact of https interception. In *NDSS*, 2017.

[13] Steven Englehardt, Dillon Reisman, Christian Eubank, Peter Zimmerman, Jonathan Mayer, Arvind Narayanan, and Edward W Felten. Cookies that give you away: The surveillance implications of web tracking. In *WWW*, 2015.

[14] Nathaniel Fruchter, Hsin Miao, Scott Stevenson, and Rebecca Balebako. Variations in tracking in relation to geographic location. *W2SP*, 2015.

[15] Jeff Hodges, Collin Jackson, and Adam Barth. RFC6797: Strict-Transport-Security Response Header Field Processing. 2012.

[16] Shan Huang, Félix Cuadrado, and Steve Uhlig. Middleboxes in the internet: a http perspective. In *TMA*, 2017.

[17] Jordan Jueckstock, Shaown Sarker, Peter Snyder, Panagiotis Papadopoulos, Matteo Varvello, Benjamin Livshits, and Alexandros Kapravelos. The blind men and the internet: Multi-vantage point web measurements. *arXiv preprint arXiv:1905.08767*, 2019.

[18] Jordan Jueckstock, Shaown Sarker, Peter Snyder, Aidan Beggs, Panagiotis Papadopoulos, Matteo Varvello, Benjamin Livshits, and Alexandros Kapravelos. Towards realistic and reproducible web crawl measurements. In *WWW*, 2021.

[19] Mohammad Taha Khan, Joe DeBlasio, Geoffrey M Voelker, Alex C Snoeren, Chris Kanich, and Narseo Vallina-Rodriguez. An empirical analysis of the commercial vpn ecosystem. In *IMC*, 2018.

[20] Akshaya Mani, Tavish Vaidya, David Dworken, and Micah Sherr. An extensive evaluation of the internet's open proxies. In *ACSAC*, 2018.

[21] Abner Mendoza, Phakpoom Chinprutthiwong, and Guofei Gu. Uncovering http header inconsistencies and the impact on desktop/mobile websites. In *WWW*, 2018.

[22] K Michael and B Joseph. Upgrading https in mid-air: an empirical study of strict transport security and key pinning. In *NDSS*, 2015.

[23] Rick Anderson (Microsoft). Work with SameSite cookies in ASP.NET. microsoft.com.

[24] Diego Perino, Matteo Varvello, and Claudio Soriente. Long-term measurement and analysis of the free proxy ecosystem. *ACM TWEB*, 2019.

[25] Victor Le Pochat, Tom Van Goethem, Samaneh Tajalizadehkhoob, Maciej Korczyński, and Wouter Joosen. Tranco: A research-oriented top sites ranking hardened against manipulation. *NDSS*, 2019.

[26] Chromium Project. HSTS preload list. hstspreload.org, .

[27] Chromium Project. HSTS preload list removal. hstspreload.org, .

[28] Sebastian Roth, Timothy Barron, Stefano Calzavara, Nick Nikiforakis, and Ben Stock. Complex security policy? a longitudinal analysis of deployed content security policies. In *NDSS*, 2020.

[29] Eman Salem Alashwali, Pawel Szalachowski, and Andrew Martin. Exploring https security inconsistencies: A cross-regional perspective. *arXiv e-prints*, 2020.

[30] Nayanamana Samarasinghe and Mohammad Mannan. Towards a global perspective on web tracking. *Elsevier: Computers & Security*, 2019.

[31] Jörg Schwenk, Marcus Niemietz, and Christian Mainka. Same-origin policy: Evaluation in modern browsers. In *USENIX Security*, 2017.

[32] Kapil Singh, Alexander Moshchuk, Helen J Wang, and Wenke Lee. On the incoherencies in web browser access control policies. In *IEEE S&P*, 2010.

[33] Suphannee Sivakorn, Angelos D Keromytis, and Jason Polakis. That's the way the cookie crumbles: Evaluating https enforcing mechanisms. In *ACM WPES*, 2016.

[34] Wissem Soussi, Maciej Korczynski, Sourena Maroofi, and Andrzej Duda. Feasibility of large-scale vulnerability notifications after gdpr. In *IEEE EuroS&PW*, 2020.

[35] Sid Stamm, Brandon Sterne, and Gervase Markham. Reining in the web with content security policy. In *WWW*, 2010.

[36] Ben Stock, Giancarlo Pellegrino, Christian Rossow, Martin Johns, and Michael Backes. Hey, you have a problem: On the feasibility of {Large-Scale} web vulnerability notification. In *USENIX Security*, 2016.

[37] Ben Stock, Martin Johns, Marius Steffens, and Michael Backes. How the web tangled itself: Uncovering the history of client-side web (in) security. In *USENIX Security*, 2017.

[38] Ben Stock, Giancarlo Pellegrino, Frank Li, Michael Backes, and Christian Rossow. Didn't you hear me?—towards more successful web vulnerability notifications. 2018.

[39] Shuo Tang, Nathan Dautenhahn, and Samuel T King. Fortifying web-based applications automatically. In *ACM CCS*, 2011.

[40] Martino Trevisan, Stefano Traverso, Eleonora Bassi, and Marco Mellia. 4 years of eu cookie law: Results and lessons learned. *PETS*, 2019.

[41] Giorgos Tsirantonakis, Panagiotis Ilia, Sotiris Ioannidis, Elias Athanasopoulos, and Michalis Polychronakis. A large-scale analysis of content modification by open http proxies. In *NDSS*, 2018.

[42] Gareth Tyson, Shan Huang, Felix Cuadrado, Ignacio Castro, Vasile C Perta, Arjuna Sathiaseelan, and Steve Uhlig. Exploring http header manipulation in-the-wild. In *WWW*, 2017.

[43] Lukas Weichselbaum, Michele Spagnuolo, Sebastian Lekies, and Artur Janc. Csp is dead, long live csp! on the insecurity of whitelists and the future of content security policy. In *ACM CCS*, 2016.

[44] Mike West. Content Security Policy Level 3. w3.org.

[45] WhatIsMyBrowser.com. Latest user agents for web browsers & operating systems. whatismybrowser.com.

A Disclosure Email

```
Hello,

We are a team of security researchers from the CISPA Helmholtz
↪  Center for Information Security located in Saarland, Germany
↪  and Università Ca' Foscari Venezia, Italy. In our current
↪  research project, we investigate inconsistent behavior in the
↪  deployment of security headers for Web applications.

For that, we have visited your site through different vantage
↪  points (VPN and Tor) as well as with different configurations
↪  (User-Agents and Accept-Language request headers).

In our automated tests, we detected both non-deterministic
↪  differences (e.g., we received different levels of security
↪  even with the same user agent) or those differences which
↪  seemed related to the vantage point or configuration.

We would like to raise your attention to one of those
↪  inconsistencies that occurred on <DOMAIN>:
<DETAILS_ABOUT_INCONSISTENCY>

We would appreciate if you can check the reason for the issue,
↪  address it to ensure consistent security, and also let us know
↪  about what such a reason might have been, since this will allow
↪  us to better help others in the future.

If you have any questions or need further information, please do
↪  not hesitate to contact us by answering this email.
```

B Overview of Additional Crawls

Our confirmation crawls offer two additional insights: on the stability of inter-test inconsistent sites and on the *in*stability of the intra-test inconsistent cases. The data, which is shown in the following tables, highlights that even 12 days after our original crawl, we could still detect 194 inter-test inconsistent sites (see Table 6). Intersecting the sites with *intra*-test inconsistencies, however, shows that the numbers seemingly decline (through 100 sites down to 96). This is to be expected, as we are measuring non-deterministic behavior. However, if we take the *union* of all sites which had at least one intra-test inconsistency across any of our crawls, this sums up to 210 (see Table 5) sites instead of only 127. This likely means that the actual dangers of non-deterministic header deployment is more severe than what we are able to measure through our limited number of observations.

Mechanism	**Usage**	**# Sites w/ intra-test inconsistencies**				
		UA	Lang.	VPN	Tor	**Any**
Content Security Policy	2,029	20	16	42	32	50
- for XSS mitigation	364	1	-	3	1	4
- for framing control	1,313	11	10	23	17	28
- for TLS enforcement	673	12	10	23	16	25
X-Frame-Options	5,751	30	30	64	39	74
Strict-Transport-Security	4,607	27	22	49	50	80
*w/o page similarity**	*4,607*	*80*	*57*	*365*	*947*	*1,152*
Cookie Security	3,975	22	16	26	28	33
- Secure attribute	3,009	9	8	13	13	17
- SameSite attribute	812	13	8	13	17	18
- HttpOnly attribute	3,196	2	-	2	2	3
Any	8,237	90	73	163	135	210

Table 5: Union of all intra-test inconsistencies snapshots.

Mechanism	**Usage**	**# Sites w/ intra-test inconsistencies**					**# Sites w/ inter-test inconsistencies**					**# Sites w/ only inter-test inconsistencies**				
		UA	Lang.	VPN	Tor	**Any**	UA	Lang.	VPN	Tor	**Any**	UA	Lang.	VPN	Tor	**Any**
Intersection of January 2 and January 6																
Content Security Policy	1,987	7	4	27	18	29	15	-	25	15	43	15	-	8	4	27
- for XSS mitigation	357	1	-	-	1	2	9	-	1	1	10	9	-	1	-	10
- for framing control	1,281	3	3	14	7	14	2	-	13	5	17	2	-	6	2	10
- for TLS enforcement	659	4	1	16	11	17	4	-	11	9	16	4	-	1	2	7
X-Frame-Options	5,662	15	13	35	17	44	7	-	22	7	30	7	-	7	2	15
Strict-Transport-Security	4,553	13	12	23	17	30	8	-	17	9	28	8	-	9	3	19
w/o page similarity	-	*37*	*23*	*75*	*322*	*394*	*18*	*2*	*515*	*145*	*583*	*17*	*2*	*489*	*27*	*520*
- preload	918	3	3	6	6	10	-	-	8	3	9	-	-	6	-	6
↳ *w/o page similarity*	-	*5*	*4*	*12*	*59*	*67*	*1*	*1*	*115*	*29*	*129*	*1*	*1*	*109*	*4*	*112*
Cookie Security	3,836	9	7	10	11	15	147	1	9	4	158	147	1	8	1	156
- Secure attribute	2,907	4	2	5	6	8	142	-	6	3	148	142	-	6	1	148
- SameSite attribute	777	5	5	5	5	7	5	1	3	1	10	5	1	2	-	8
- HttpOnly attribute	3,069	-	-	1	1	2	2	-	2	1	4	2	-	2	-	4
Any	8,145	39	31	86	59	100	174	1	64	30	244	172	1	26	8	191
Intersection of January 2 and January 10																
Content Security Policy	1,986	9	4	27	18	30	15	-	26	16	43	15	-	10	4	29
- for XSS mitigation	354	-	-	-	1	2	9	-	1	1	10	9	-	1	-	10
- for framing control	1,285	5	3	15	8	15	2	-	14	5	18	2	-	7	1	10
- for TLS enforcement	658	5	1	16	10	18	4	-	11	10	15	4	-	2	3	9
X-Frame-Options	5,654	14	12	35	19	43	7	-	20	12	30	7	-	6	5	17
Strict-Transport-Security	4,549	12	12	21	16	30	8	-	17	9	27	8	-	10	4	20
w/o page similarity	-	*32*	*24*	*77*	*370*	*443*	*18*	*2*	*512*	*139*	*573*	*17*	*2*	*480*	*18*	*503*
- preload	914	2	3	5	5	9	-	-	8	4	9	-	-	6	1	7
↳ *w/o page similarity*	-	*4*	*4*	*11*	*71*	*81*	*1*	*1*	*114*	*28*	*130*	*1*	*1*	*108*	*3*	*112*
Cookie Security	3,841	10	8	11	10	16	147	1	10	4	159	146	1	7	1	154
- Secure attribute	2,914	4	3	5	5	8	141	-	6	3	147	141	-	5	1	146
- SameSite attribute	781	5	5	5	5	7	6	1	4	1	12	6	1	2	-	9
- HttpOnly attribute	3,075	1	-	2	1	3	2	-	2	1	4	2	-	2	-	4
Any	8,142	39	30	86	58	100	174	1	66	35	244	173	1	29	12	194
Intersection of January 2 and January 14																
Content Security Policy	1,985	8	5	26	20	31	15	-	26	16	43	15	-	10	4	29
- for XSS mitigation	359	-	-	-	1	1	9	-	1	1	10	9	-	1	-	10
- for framing control	1,278	5	2	15	8	16	2	-	13	5	17	2	-	6	2	10
- for TLS enforcement	659	4	3	15	12	18	4	-	12	10	16	4	-	3	2	9
X-Frame-Options	5,654	14	8	32	18	38	6	-	18	11	26	6	-	7	5	15
Strict-Transport-Security	4,548	12	10	19	17	26	7	-	15	7	24	7	-	11	2	19
w/o page similarity	-	*33*	*22*	*65*	*369*	*424*	*17*	*2*	*535*	*136*	*595*	*16*	*2*	*512*	*20*	*535*
- preload	913	3	2	5	6	9	-	-	8	4	9	-	-	6	-	6
↳ *w/o page similarity*	-	*5*	*5*	*10*	*66*	*73*	*1*	*1*	*119*	*29*	*131*	*1*	*1*	*114*	*2*	*116*
Cookie Security	3,825	10	9	11	11	16	148	1	11	4	161	147	1	9	1	157
- Secure attribute	2,897	4	4	5	6	8	143	-	8	3	151	143	-	7	1	150
- SameSite attribute	778	5	5	5	5	7	5	1	3	1	10	5	1	2	-	8
- HttpOnly attribute	3,066	1	-	2	1	3	2	-	2	1	4	2	-	2	-	4
Any	8,135	38	27	79	61	96	174	1	61	33	239	173	1	31	10	194

Table 6: Overview of overlap with additional snapshots of our analysis