Uncovering HTTP Header Inconsistencies and the Impact on Desktop/Mobile Websites

Abner Mendoza
Texas A&M University
abmendoza@tamu.edu

Phakpoom Chinprutthiwong*
Texas A&M University
cpx0rpc@tamu.edu

Guofei Gu
Texas A&M University
guofei@cse.tamu.edu

ABSTRACT

The paradigm shift to a mobile-first economy has seen a drastic increase in mobile-optimized websites that in many cases are derived from their desktop counterparts. Mobile website design is often focused on performance optimization rather than security, and possibly developed by different teams of developers. This has resulted in a number of subtle but critical inconsistencies in terms of security guarantees provided on the web platform, such as protection mechanisms against common web attacks. In this work, we have conducted the first systematic measurement study of inconsistencies between mobile and desktop HTTP security response configuration in the top 70,000 websites. We show that HTTP security configuration inconsistencies between mobile and desktop versions of the same website can lead to vulnerabilities. Our study compares data snapshots collected one year apart to garner insights into the longitudinal trends of mobile versus desktop inconsistencies in websites.

To complement our measurement study, we present a threat analysis that explores some possible attack scenarios that can leverage the inconsistencies found on real websites. We systematically analyze the security impact of the inconsistent implementations between the mobile and desktop versions of a website and show how it can lead to real-world exploits. We present several case studies of popular websites to show real-world impact of how these inconsistencies are leveraged to compromise security and privacy of web users. Our results show little to no improvements across our datasets, which highlight the continued pervasiveness of subtle inconsistencies affecting even some high profile websites.

CCS CONCEPTS

• **Networks → Web protocol security**;

KEYWORDS

Mobile Web Security, HTTP Headers, Inconsistencies

ACM Reference Format:
Abner Mendoza, Phakpoom Chinprutthiwong, and Guofei Gu. 2018. Uncovering HTTP Header Inconsistencies and the Impact on Desktop/Mobile Websites. In *WWW 2018: The 2018 Web Conference, April 23–27, 2018, Lyon, France.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3178876.3186091

*Joint First Author

WWW 2018, April 23–27, 2018, Lyon, France

ACM ISBN 978-1-4503-5639-8/18/04.
https://doi.org/10.1145/3178876.3186091

1 INTRODUCTION

The advent of ubiquitous and powerful smart mobile devices has seen a surge of mobile-optimized websites that complement existing desktop-optimized websites. Many popular web applications offer a mobile optimized version of their website when a mobile client is detected. Mobile websites are typically optimized with usability and performance in mind, and are usually simplified version of their desktop counterparts, which are typically developed and maintained by different teams of developers. While mobile devices become increasingly more powerful, their resources (RAM, CPU, Disk, etc.) are still limited as compared to desktop computers and laptops. This increases a need for optimized web applications served to mobile devices, usually in the form of scaled-down user interface and functionality. In the mobile space, each byte transferred between the server and client carries a cost, both in terms of money and resources required on the device [23]. As a result, developers are especially mindful of minimizing all data sent from the server, and find every opportunity to squeeze extra performance from their web application.

While many previous researchers have explored the problems on the web platform, none have assessed the extent to which mobile conformity affects security and privacy on the web. In this work we explore the pervasiveness and impact of subtle inconsistencies of HTTP security headers between desktop and mobile websites. To that end, we explore and compare inconsistencies between mobile and desktop website artifacts to determine whether they can lead to security vulnerabilities.

While many online services may have corresponding native mobile applications, a recent study showed that many users still access these services through their mobile web browsers, especially in the areas of shopping, reference, news, and education [11]. Moreover, many native apps are simply wrappers around an embedded web browser component that simply loads a mobile optimized website. Web browsers themselves have evolved over the years to become a bona-fide application platform that must now support the interaction of several mutually distrusting principals [28] which may perform critical functionality within a web application. Through this evolution, several security mechanisms have emerged that are aimed at strengthening the security of the browser platform against common attacks [31]. Most of these mechanisms are implemented using HTTP Headers, which serve as directives from the hosting web server to instruct the client browser to enforce certain policies [9, 16, 17, 31, 32]. We refer to such headers as *security headers*.

Web applications are usually designed and deployed using an n-tiered approach that separates the data, business logic, and user interface components. In their pursuit of seamless user experience and optimal performance, web developers use different techniques to serve content according to the user's detected browser.

Many web servers and web applications use simple browser fingerprinting techniques, either through inspection of the HTTP `User-Agent` Header, or JavaScript's `navigator.userAgent` object, to infer the form factor of the user and decide whether to serve a mobile-optimized or full desktop version of a website. These applications can be deployed in a variety of scenarios where mobile and desktop versions share one or more layers. This improves scalability and conveniently allows mobile and desktop-optimized web applications to share the same underlying server resources.

Unfortunately, this design and deployment flexibility can lead to inconsistencies with regard to critical artifacts such as session cookies, and HTTP security headers. For example, consider if the server hosting www.example.com enforces strict HTTPS, but the server hosting m.example.com allows insecure HTTP connections. Assuming a shared back-end server infrastructure and cookies set using the *domain* attribute [28], this trivial inconsistency can enable an attacker to retrieve authentication cookies saved by www.example.com, by forcing the victim to visit m.example.com through a man-in-the-middle scenario.

Regardless of whether a website is served to a desktop browser or a mobile device, we assume some aspect of shared content and resources. This is common in virtual hosting environments where one server is used to host a web application, and a browser fingerprinting component routes requests to a mobile-optimized or desktop-optimized virtual host of the same website. The presence of a common underlying server allows both mobile and desktop websites to conveniently share common resources. For example, while many mobile users are redirected to an 'm.' subdomain, the cookies are set using the second-level domain (SLD) path. This allows the application state to be shared between the mobile and desktop sites. Subtle inconsistencies emerge from this practice, whether due to differences in the configuration of the virtual hosts, or due to performance optimization inconsistently applied between each virtual host on a shared server. In this context, we define the origin as the top level domain. For example, the sites www.example.com and m.example.com are considered as example.com.

In short, our main contributions are as follows:

(1) We conduct the first longitudinal, large-scale, systematic measurement study on the security header inconsistencies of (top 70,000) web server responses requested by desktop and mobile web browsers. We show that such security inconsistency is a pervasive problem in the real world.
(2) We systematically analyze the security impact of these inconsistencies and show how they can cause web applications to become more vulnerable and exposed to exploits with severe consequences. We showcase several examples of real-world top websites such as netflix.com and disneystore.com. We have also worked with a few of them to help fix the inconsistencies.

2 BACKGROUND

The web infrastructure continues to be exploited by attackers, with wide spread implications that affect every facet of society. Over the years, Researchers and Industry have responded in kind by developing browser mechanisms that mitigate popular website attacks such as cross-site scripting, clickjacking, and others. The browser has become the main battleground, and many recent proposals have sought to strengthen the browser by leveraging and extending existing features. In particular, HTTP Headers have become a popular channel for implementing defense mechanisms [9, 17, 25, 31]. Effective website security defense mechanisms have emerged in the form of declarative security in HTTP headers, which provide explicit security parameters that instruct browsers to enforce specific security functionality against common web vulnerabilities. This requires close coordination between the browser and server.

Table 1: List of Security HTTP Headers Analyzed

Headers	Example Attacks Mitigated
Set-Cookie	Session hijacking, Cookie stealing.
X-Frame-Options	Clickjacking
Access-Control-Allow-Origin	Cross-site access
X-XSS-Protection	Cross-site scripting
Strict-Transport-Security	Man-in-the-middle
X-Content-Type-Options	MIME sniffing.
Content-Security-Policy	XSS, CSRF

2.1 HTTP Security Headers

Every HTTP request and response between two communicating parties includes one or more HTTP headers [13]. These headers are an important part of the HTTP protocol, carrying additional information about each request or response that is utilized by the client or server to process the request or response. HTTP headers are formatted as key-value pairs, with the key representing the name of the header, and the value representing the specific configuration for the header. HTTP security headers are declarations by the responding web server instructing the client browser to enforce certain built-in security mechanisms to mitigate common web attacks, such as cross site scripting filtering, prevention of cross site request forgery, and others [4]. Table 1 shows a list of the specific HTTP security headers that we focus on in this work.

HTTP headers can serve as effective and cheap means by which websites can declare to a browser what security principles to enforce. However, the presence of these headers do not guarantee enforcement on the client side.

In addition to specifying the headers, web servers must also correctly configure the header's values and parameters, which serve as the input to the browser regarding the actions it must take to mitigate a specific attack vector. Security headers have no effect or unintended consequences if they are incorrectly or ambiguously configured [12]. For example, a web server must send the header "*X-Frame-Options:DENY*" to disallow the website from being loaded into an iframe. If, for example, the value is incorrectly specified as "*Denied*", then the browser will not be able to understand that directive, and the website may then be vulnerable to clickjacking attacks.

2.1.1 Defense in Depth. Several previous studies have proposed HTTP header-based defenses to mitigate common web attacks [5, 9, 10, 14, 15, 18, 19, 22, 25, 28, 31]. Most of the deployed defenses on major browsers are defense-in-depth strategies and not meant as completed prevention of web attacks. They do not eliminate any

of the targeted attacks, but make it more difficult for attackers to exploit them. We focus only on the HTTP header-specific defenses which we have observed in our analysis of the top 70,000 websites. These Headers are listed in Table 1, and we collectively refer to them in this paper as 'security headers'. We show that inconsistencies across implementations for mobile and desktop rendering reduce the efficacy of these security header defenses.

2.1.2 Cookies. While cookies are HTTP Headers themselves, we treat them differently because cookies have more diverse uses beyond security policy enforcement. Cookies are used as a mechanism for maintaining state in place of HTTP's stateless nature. Cookies have evolved into a major tool in facilitating authorization and authentication for web applications. Additionally, while cookies are also HTTP Headers (Set-Cookie), they differ from traditional HTTP Headers as the only headers for which storage space must be allocated on the client side by the browser for saving and manipulating the stored values. Set-Cookie is also one of the few HTTP headers that can appear multiple times in a request or response [12]. Cookies can also be set using JavaScript, but we focus on it's usage as an HTTP Header in this paper.

2.1.3 Redirection. In the scope of this work, we focus on redirection performed by a web server to upgrade the communication channel from HTTP to HTTPS, which is often done through delivery of a *Location* header along with 3xx return status code. Additionally, we consider redirections due to Strict-Transport-Security (HSTS) or other server configuration that mandates a secure connection channel. We refer to these collectively as Redirection Directives. However, unlike other security headers, the purpose of the *Redirection Directive* headers are not primarily for security, thus we do not list them in Table 1.

3 EXPLOITING INCONSISTENCIES

In this section, we first introduce our threat model, discuss the impact of inconsistent configurations on websites' security, and then show real world examples that contain inconsistencies between their mobile and desktop websites.

3.1 Threat Model

Our threat model assumes an attacker who can act as a user and make requests to remote servers and inspect responses. This attacker can collect, compare, and aggregate web request and response artifacts to understand how to craft attacks that can leverage weaknesses found for each alternative view of a website. We also consider an active network attacker who is able to inspect and modify unencrypted networking traffic between the browser and the server. However, the attacker cannot decrypt or modify traffic over HTTPS except for his own traffic.

3.2 Inconsistency Scenarios

We explore the security impact of inconsistencies based on four broad perspectives that we view as preconditions for enabling common web attacks using inconsistencies, such as man-in-the-middle attacks, cookie stealing or replaying, and others. These inconsistencies have the impact of facilitating attacks by minimizing the attack effort (such as avoiding the need for SSL stripping) or increasing the attack surface (such as a misconfiguration of XSS protection header).

1. Presence Inconsistency. In this scenario, a critical security header is present in one user agent, but missing in the other. An attacker can leverage this knowledge in a chosen user agent attack by redirecting the user to an alternative view of a website based on a security header that is not present. For example, an attacker can force a victim to an alternative view that does not include clickjacking protection, and then embed that view in a hidden iframe to carry out an attack such as illustrated in [7].

As illustrated in Figure 2, an attacker can act as a MITM to rewrite the request's User-Agent header to force a response where a particular security header is known to be missing rather than the response that would include the header. Or, an attacker can use social engineering attack to make a victim access the inconsistent website through a weaker channel.

2. Configuration Inconsistency. In this scenario, an attacker observes and compares the configuration of a particular security header between two user agents. For example, if CSP is configured to allow scripts (script-src) from a source site's subdomains in one user agent, and not in the other, the attacker can leverage such an inconsistency to his advantage. Consider if the domain is used in a shared hosting environment. Then, if the attacker controls a subdomain in that shared hosting environment, he can deploy malicious hosts in his own subdomain, and redirect a victim user to the inconsistent victim website UI where he can then leverage a XSS, MITM, or other attack to inject code from his controlled subdomain that will execute in the victim's browser since its inconsistent CSP configuration allows scripts from subdomains. Although the attacker may be limited by virtue of the subdomain being a different origin, the potential attack is not inconsequential.

3. HTTPS Redirection Inconsistency. In this scenario, an attacker leverages the fact that one response, depending on the user agent, always redirects to HTTPS while another response will not redirect to HTTPS. If an attacker can force a user to visit a site where his request's user agent setting is set to the version that will not redirect to HTTPS, then the attacker can keep the connection unencrypted and leverage a MITM attack, as shown in in Fig. 2. We have found that in some cases, one response is configured with HSTS, while another response does not have HSTS. In such a scenario, the state of the connection is inconsistent and can be leveraged by an attacker.

4. Cookie Flag Inconsistency. In yet another scenario, inconsistency in how a particular cookie's flag is configured between two user agents can enable an attacker to steal HTTP-only cookies that are written using a secure HTTPS channel. In this example scenario, a cookie's Secure flag is set to false when written after a request by a desktop user agent.

It is important to note that in the MITM attacker scenario, an attacker can also launch a full SSLStrip attack instead of just rewriting user agent to lead the victim to a weaker communication channel. These inconsistencies do not necessarily lead to new exploits or stronger attacks, but they give an attacker more options to launch attacks based on different scenarios and circumstances in practice.

3.3 Real World Examples

Here we discuss some real world websites that we found to have inconsistencies in their security headers, HTTPS redirection directives, and cookies. In all these case studies, it is important to note that what we illustrate are not necessarily new attack techniques, but new ways to carry out known attacks by leveraging inconsistencies that can facilitate the targeted attacks. We reported these issues to the respective websites and worked closely with some of them to understand and fix the issue. In all cases, the inconsistency problems have been fixed by these websites.

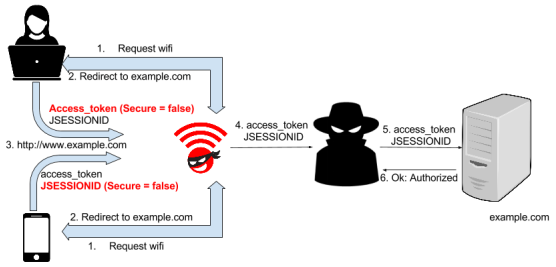

Figure 1: Cross-platform information leakage aggregation attack.

3.3.1 Disneystore.com. This example leverages the cookie flag inconsistency to launch a cookie replay attack by aggregating information from two channels, i.e., desktop and mobile. To the best of our knowledge, aggregating information leaked using both mobile and desktop to collect necessary cookies that would not otherwise be disclosed using one user agent alone has not been discussed before in previous literature. We assume an active network attacker who controls a rogue network access point, such as the evil twin attacker [27], but does not necessarily employ SSL stripping. The goal is to steal the user cookie and launch a cookie replay attack. This considers the classic coffee shop scenario where a victim user accesses free WiFi connectivity.

Our analysis framework (discussed in next section) reported cookie inconsistencies. The cookie 'JSESSIONID', which is used for session management in in J2EE web applications, had its secure flag set to false when browsed by mobile user agents. Additionally, another cookie 'access_token', which can be combined with JSESSIONID to replicate a user's session, had its secure flag set to false when browsed by desktop user agents. This scenario is illustrated in 1.

By aggregating the 'access_token' cookie leaked from the victim's desktop and the 'JSESSIONID' cookie leaked from the victim's mobile, we were able to obtain and replay these cookies to acquire full access to a test victim account. On a real account, this would contain the victim's private information such as mailing address, email, cellphone number, purchase history, and a partial credit card number. For this scenario to be feasible, the victim user has to be logged in to the website both on mobile and desktop devices with persistent authentication, which is not uncommon. Since the Secure flag is not set, the cookies will be sent over an insecure channel, exposing them to an MITM attacker. This scenario is feasible, since it is typical for people to carry their mobile devices everywhere, and use their personal computers from varying locations. A survey by ComScore [2] between 2013-2015 showed that 57 to 84 percent of consumers regularly use both mobile and desktop platforms interchangeably for browsing.

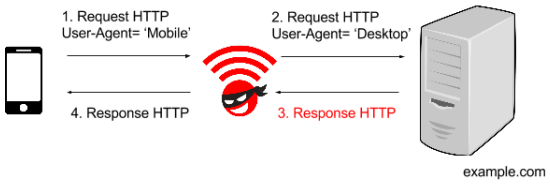

Figure 2: Redirecting client to less secure UI

3.3.2 Bet365.com. This example leverages redirection inconsistency and cookie flag inconsistency by forcing the user to browse through a weaker communication channel. The attacker can force (by altering user agent field on the initial HTTP connection) or coerce (by social engineering attack) the victim to a less secure channel based on the inconsistencies that the attacker has knowledge of. This has a similar result, but is less invasive than an SSL stripping attack. The goal is to redirect the user to a view that meets the preconditions of the attack, such as having an unencrypted channel or a missing XSS protection header.

On mobile agents, bet365.com's server will redirect HTTP request to HTTPS and also set a session cookie, 'pstk', with its Secure flag set to true. However, on desktop agents, bet365.com's server will neither redirect HTTP to HTTPS nor set the 'pstk' Secure flag to true.

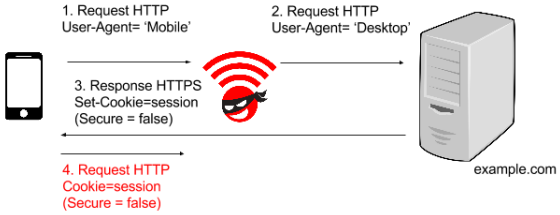

Figure 3: Inconsistency in cookie flag leading to an attack

An attacker can leverage this insight to force a victim into an insecure channel by social engineering or replacing the user agent on the initial HTTP connection and then have full access to his web communication with bet365.com, potentially leaking highly sensitive information, including financial information. This scenario is illustrated in Figure 3

Now let's assume that bet365.com redirects HTTP to HTTPS regardless of the user agent settings. A similar attack can still occur if the 'pstk' cookie flag is still inconsistent. In this case, bet365.com's server will redirect the user to the HTTPS channel, but will not set the 'pstk' Secure flag to true. Any subsequent request from the user through an initial HTTP request will expose the 'pstk' cookie to the MITM attacker. The subsequent request can be voluntarily sent by the user or forcefully using CSRF or XSS, for example. In the

same scenario, the attacker can deploy a rogue WiFi access point and put an invisible iframe over an 'accept terms and conditions' button which will redirect the victim to bet365.com over HTTP causing his session cookie to be leaked.

3.3.3 Netflix.com. This attack leverages the cookie flag inconsistency, and assumes an XSS vulnerability on the website. This website contains a sensitive cookie, *SecureNetflixID*, which is used to maintain the user session. The cookie had its HTTPOnly flag set to true on desktop user agents, but this flag is set to false on mobile user agents. In some cases, a second cookie, *NetflixID*, is also sent with both the 'HttpOnly', and 'Secure' flags set to false. These sensitive cookies are therefore visible to any scripts running in the website. As a result, these can be exposed to malicious scripts that exploit any XSS vulnerability on the site.

We reported this potential data exposure vulnerability to Netflix as part of their responsible disclosure program. They responded quickly and we collaborated with them over a few weeks while their team analyzed the issue. They determined that no mobile use case requires the cookie to be accessed by script and acknowledged that the inconsistency constituted a vulnerability in their deployed cookie configuration for their mobile website. They have since fixed the problem on all their production servers.

They also indicated to us that their web application uses different API endpoints depending on the platform or client being used. This falls in line with our assertion that web application are deployed using shared resources to conform to mobile and other form factors on demand.

4 DATA COLLECTION & ANALYSIS

To understand the pervasiveness and trends of security header inconsistencies in the real world, we performed a large-scale measurement study in two distinct time periods one year apart (2016 to 2017) using the top 70,000 website as ranked by Alexa [1] as of 2016. Here we describe our data collection framework and analysis methodology.

4.1 Data Collection

We developed a custom web crawler that automatically visits each site and saves the full contents of the response for offline analysis. Our crawler simulates a desktop browser using a user agent string as shown in listing 1. These were chosen based on current trends in browser popularity as measured by NetMarketShare.com. These represent the Google Chrome browser for desktop websites and Chrome mobile browser for mobile websites. Additionally, we used two different sets of user agent strings (labeled 'old' and 'new' in Listing 1) to compare whether the user agent string itself makes a difference for the server response with regard to security header inconsistencies.

Additionally, our crawler follows all redirects, including those to secured HTTPS sites, and collects only the data from the final landing pages.

Listing 1: Browser User-Agent Strings

```
1 @old_pc_agent = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
      /537.36 (KHTML, like Gecko) Chrome/51.0.2704.106 Safari
      /537.36"
2 @old_mobile_agent = "Mozilla/5.0 (Linux; Android 4.0.4; Galaxy
      Nexus Build/IMM76B) AppleWebKit/535.19 (KHTML, like Gecko)
       Chrome/18.0.1025.133 Mobile Safari/535.19"
3 @new_pc_agent = "Mozilla/5.0 (X11; Linux x86_64) AppleWebKit
      /537.36 (KHTML, like Gecko) Chrome/60.0.3112.90 Safari
      /537.36"
4 @new_mobile_agent = "Mozilla/5.0 (Linux; Android 8.0; Nexus 6P
      Build/OPP3.170518.006) AppleWebKit/537.36 (KHTML, like
      Gecko) Chrome/58.0.3029.121 Mobile Safari/537.36"
```

We further annotated the site list to include the site categories as defined by Fortinet [3], a web filtering service. The categories allows us to perform further analysis on our results based on trends according to category grouping.

We collected one snapshot in 2016, and one snapshot in 2017. On each occasion, we collected two separate data sets, corresponding to site responses to mobile browsers and site responses to desktop browsers.

4.1.1 Datasets. To compare the longitudinal trends of inconsistencies and the effect of the browser and platform versions of user agent strings, we further categorize the data into three datasets.

Old dataset This dataset represents the HTTP responses data crawled during 2016 using the old_pc_agent and old_mobile_agent presented in Listing 1.

New dataset This dataset represents the HTTP responses data crawled during 2017 using the old_pc_agent and old_mobile_agent.

New+ dataset This dataset represents the HTTP responses data crawled during 2017 using the new_pc_agent and new_mobile_agent.

By comparing the Old dataset and New dataset, we analyze if the situation of inconsistency improved over time. The comparison between the New dataset and New+ dataset will indicate how much the platform and browser version may affect the HTTP header response.

4.2 Analysis Methodology

We analyzed the security header artifacts on mobile and desktop user agents from the two broad perspectives of *Presence*, and *Configuration*. First, we compare and contrast the presence of each security header between mobile and web to identify inconsistent presence between user agents. Similarly, we compare and contrast the Configuration of the same security header between user agents to uncover inconsistencies. We identified instances where configurations are either flawed or conflict with other settings that may weaken the protections offered by HTTP header defenses in browsers. We also analyzed inconsistencies in *HTTP to HTTPS redirection*. Lastly, we looked at subtle but important inconsistencies with *Secure* and *HttpOnly* cookie flags.

For a systematic analysis, we establish the following guidelines for analyzing inconsistency between mobile and desktop versions of a website from the same origin.

(1) If a Security Header is present for one user agent, it must be present for all user agents.
(2) Each Security Header must be consistently delivered with the same effective configuration settings regardless of the user agent.
(3) If cookies are shared between mobile and desktop, they must be consistently delivered with the same settings regardless of the user agent.

(4) If a website redirects to HTTPS, it must do so for all user agents.
(5) The presence and configuration settings within and among different artifacts should not cause conflicting policies. For example, CSP on a website should not allow and deny the same domain [12].

4.2.1 Uncovering Inconsistencies. Using a set of the top 70,000 websites, we collected data using a simulated desktop and mobile browser. For each pair of response from the landing page of the same top-level domain, we checked for violations of our stated guidelines to uncover inconsistencies.

For each pair of responses from each website (mobile vs. desktop), we used our methodology guidelines to find instances that violated our assertions for how security headers should be delivered.

Presence Inconsistency. The most trivial inconsistency to measure is the presence of a header for the same website between two user agents. Although trivial, these subtle inconsistencies can be gateways to serious attacks on a website. For example, if the X-Frame-Options header is present for a website in the desktop agent, but missing in the mobile agent, then it follows that the mobile version can be used in a clickjacking attack.

To measure these inconsistencies, we simply analyze each pair of responses for each website and, for each security header, we note when the header is present in one response and missing in the other response.

Configuration Inconsistency. To evaluate the configuration of each security header, we adopt a notion of '*Strong*' and '*Weak*' configurations. These are based on a set of heuristics as defined in Table 2. This approach is motivated by the methodology use in [33]. We analyzed the configuration options available for each header, and analyzed whether each observed header is deployed with an ineffective setting. For example, if the X-XSS-Protection header is set to any value other than '1' then it is ineffective for that website and we classify that configuration as '*Weak*'. On the other hand, if it is configured with the value '1', then we classify that setting as '*Strong*'. Note that most popular browsers (eg., Chrome) tend to set missing security headers securely by default. For example, X-XSS-Protection is set to 1, and set-cookie headers with no explicit domain will be set to the current domain. Therefore, setting the security headers inconsistently could be worse than not setting them at all.

For analyzing the efficacy of Content Security Policy configurations, we also adopt the methodology used in [33]. It follows that our evaluation analyzes CSP based on its main goal of defending against content injection, mainly through script and object tags.

HTTPS Redirection Inconsistency. We measure redirection inconsistency through our crawler framework by initiating all connections using HTTP and ensuring that all redirects are followed and flagging each connection where the final response uses an HTTPS connection.

Many web attacks, including MITM attacks, can be mitigated if the user agents use secure connections through HTTPS. The current recommended method of mitigating HTTPS downgrade attacks is to use HSTS [17]. However, HSTS is deployed on only a limited number of websites [20]. Websites can also enable redirection using trivial methods such as JavaScript's document.location function, or by using an HTML Meta tag. The reality remains that many websites do not use HTTPS, seldom upgrades connection to HTTPS, and even in the presence of HTTPS redirection, we have found many instances of inconsistent implementation of strict redirection across different user agents for the same website. This inconsistency can be leveraged by a network attacker, by forcing a user to consistently use an insecure connection through which he can most conveniently launch a MITM attack.

Cookie Flag Inconsistency. For instances where a cookie with the same name is present in both responses, we assume that the cookie is shared between the mobile and desktop user agents. We then compare the '*Secure*' and '*HTTPOnly*' flags set for these cookies to find instances where the flags are not set consistently. We check for violations where each flag, exclusive of each other, is true in one user agent instance and false in the other user agent instance.

5 MEASURING INCONSISTENCIES

Using the guiding principles presented in Section 4.2, we evaluate the top 70,000 websites. Table 3 shows the results of our empirical evaluation where we measure and assess the inconsistencies across mobile and desktop user agents. Within each cell, we report the different results from the three datasets discussed in Section 4.1.1 (e.g., Old/New/New+). For each relevant security header, we show the pervasiveness of the inconsistency problem based on presence and configuration perspectives. Overall, over 2,000 websites that responded to our crawler showed one or more inconsistency in one or more security headers in violation of our analysis principles. The inconsistency problem is spread across the entire spectrum of the top 70,000 websites, and even highly ranked and popular web service providers such as Netflix and Google showed inconsistencies.

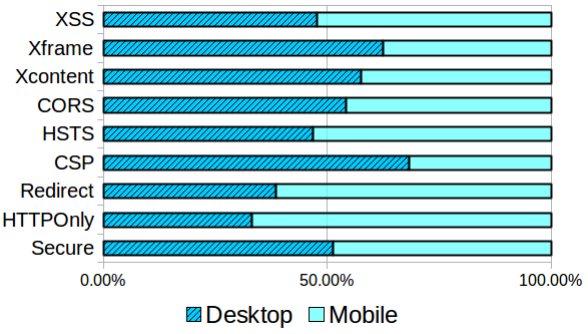

Figure 4: Percentage between desktop and mobile websites with inconsistent presence of HTTP security header

Overall Presence Inconsistency Results. The Desktop column in Table 3 shows the number of websites for which that security header was present only when the desktop user agent was used. Similarly, the Mobile column shows the number of websites for which that security header was present only when the mobile user agent was used. The sum of the Desktop and Mobile columns represents the total number of instances for which that security header was inconsistently delivered.

Figure 4 compares the presence inconsistency for each relevant header in our New+ dataset. We compare the percentage between

Table 2: Heuristics for appraising the Configuration of Security Headers.

Headers	Rules
X-XSS-Protection (XSS)	set to 1, additional parameter is optional
X-Frame-Options (Xframe)	does not use wild card (*) in allow-from
X-Content-Type-Options (Xcontent)	set to nosniff
Access-Control-Allow-Origin (CORS)	does not use wild card (*)
Strict-Transport-Security (HSTS)	max-age is greater than 999, additional parameter is optional
Content-Security-Policy (CSP)	both script-src and object-src must be present or default-src in their absences. does not contain unsafe-inline or unsafe-eval. does not use wild card (*) in script-src, object-src, or default-src whitelists. does not contain unsafe origin in whitelist [16].

Table 3: Inconsistency Analysis of security headers on the top 70,000 websites. (Old/New/New+)

	Presence			Strong Configuration		
Headers	**Desktop**	**Mobile**	**Consistent**	**Desktop**	**Mobile**	**Consistent**
X-XSS-Protection (XSS)	77/103/91	84/113/100	4,322/6,721/6,717	67/94/82	85/115/100	4,208/6,542/6,541
X-Frame-Options (Xframe)	270/285/268	183/185/161	8,600/12,068/12,089	266/279/264	176/179/154	8,425/11,848/11,869
X-Content-Type-Options (Xcontent)	107/132/118	98/99/87	5,462/8,356/8,346	106/132/118	99/99/87	5,444/8,334/8,324
Access-Control-Allow-Origin (CORS)	183/175/163	133/151/138	3,053/3,694/3,726	63/54/51	33/43/47	538/668/671
Strict-Transport-Security (HSTS)	70/99/75	55/86/85	2,957/6,268/6,275	60/93/67	50/77/76	2,566/5,383/5,392
Content-Security-Policy (CSP)	53/96/58	18/20/27	726/1,618/1,654	0/0/0	0/0/0	2/3/3

mobile and desktop user agents. The desktop percentage relates to the number of websites where a security header was present using a desktop user agent, but absent for mobile user agents. Similarly, Mobile relates to the percentage where mobile included a header that was excluded for Desktop on the same site.

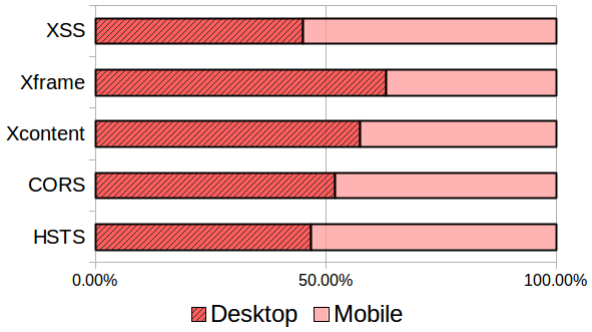

Figure 5: Percentage between desktop and mobile websites with inconsistent strong configuration of HTTP security header

Overall Configuration Inconsistency Results. Figure 5 also shows the results of our evaluation for the each security header's configuration based on our notion of *strong* and *weak* configurations as presented in section 4.2.1. For each header, the Desktop percentage specifies the portion of websites for which that security header was configured as strong for desktop user agents, and weak for mobile user agents. Similarly, the Mobile percentage specifies the portion of websites for which the security header was only configured as 'strong' for the mobile user agent and not for the desktop user agent. This is also shown in Table 3, along with the Consistent column which also indicates the number of websites for which the security header was consistently configured as Strong for responses from both user agents.

Weichselbaum et al. [33] found that 99.34% of hosts with CSP use policies that offer no benefit against XSS. Indeed, we found that there were only two websites with consistently strong CSP configuration between mobile and desktop (hackerone.com and github.com) in both snapshots.

We must also note that we do not include websites that are labeled 'weak' or do not have any security headers present in both user agents as these websites might not be aware of HTTP security headers in the first place.

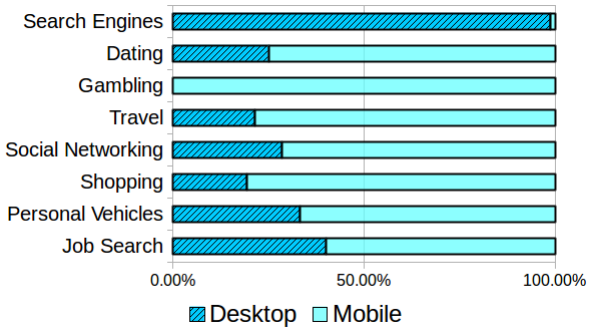

Figure 6: Percentage between desktop and mobile websites of the top 8 website categories with inconsistent redirection to HTTPS

By comparing the presence with the strong/weak notation, we can see one notable observation that the Access-Control-Allow-Origin (CORS) header has the largest gap between presence and strong numbers shown in Table 3. While CORS header is inconsistently present in 163 and 138 websites for the Desktop and Mobile

respectively, only 51 and 47 websites of those are configured correctly. This indicates that the Access-Control-Allow-Origin header configuration is mostly ineffective (eg., using wild card host) in protecting against cross-site access attacks. We note that the primary purpose of CORS is not to mitigate attacks, but to relax the same origin policy (SOP). Nevertheless, by relaxing SOP, CORS opens the doors to a wider cross-site attack surface.

Overall Redirection Inconsistency Results. Table 4 shows the overall results for the number of websites that inconsistently redirected to HTTPS. The Desktop column shows the number of websites that redirect to HTTPS when accessed by desktop user agent, but do not redirect when accessed by mobile user agent. The second column shows the number of websites that redirect to HTTPS when accessed by mobile user agent, but do not redirect when accessed by desktop user agent. The last column shows the number of websites that redirect HTTP to HTTPS request regardless of user agent. These numbers are mutually exclusive, and we do not factor instances where no redirection occurred.

We also found that 36,350 websites in our dataset do not redirect to HTTPS for either user agent. While this is 'consistent' behavior as per our principles, it goes contrary to ubiquitous HTTPS that is advocated by researchers and industry alike. Despite the availability of techniques such as HSTS [17, 20], and support by major browsers, many sites still do not implement HTTPS, posing a serious privacy and security risk on the web. Even sites belonging to Google, such as www.google.cat and www.google.sr, had still not fully implemented ubiquitous HTTPS. We contacted Google about these instances and they acknowledged the issue and noted that updates are forthcoming.

In Fig 6, we show the percentage of inconsistency in HTTPS redirection based on the website categories, showing that Search Engine websites had the most violations. Google was the primary guilty party in these instances, mostly for country-code top level domains for their search engine.

Table 4: Overall Inconsistency Analysis on HTTPS Redirection and Cookie Flags on the top 70,000 websites.

	Desktop	Mobile	Consistent
Redirection	307/392/194	297/307/309	11,967/25,495/25,687
Secure	48/71/78	87/79/74	7,251/14,250/16,695
HTTPOnly	102/70/76	132/154/153	48,187/51,929/60,331

Overall Cookie Flag Inconsistency Results. We also compared the number of inconsistencies in cookie configuration as also shown in Table 4. The numbers for Desktop and Mobile columns indicate the number of cookies which set HTTPOnly or Secure flag to true in desktop but false in the corresponding flag in mobile, and vice versa. On further manual investigation of some of these sites, we found that most have inconsistencies resulting from the default setting of J2EE servlet web server that sets the Secure flag only for HTTPS connections.

Longitudinal Trends of Inconsistency Results. From Table 3, we can see significant changes in every header's Consistent column when comparing between the Old and New datasets both in term of Presence and Strong Configuration. This shows an increasing adoption of HTTP headers over the period of one year. However, the relative numbers of Inconsistencies show very small deviations for almost all of the HTTP headers. Ideally, we would like to see a marked decrease in inconsistencies, but instead we see little to no improvement, even on highly ranked websites between the 2016 and 2017 snapshots.

Effect of Browser and OS Versions. By comparing the results in the New and New+ datasets, we observe that the numbers are almost identical with negligible deviation. This suggests that the Browser and Mobile OS versions in the user agent strings generally play a very small role in HTTP header responses. However, there are notable differences in the HSTS header and the HTTPS Redirection configuration where the newer versions of Chrome and Android yield lesser numbers of inconsistencies on the desktop responses. This implies that the websites that once only provide secure redirection in older Chrome and Android versions now also redirect the mobile user agent to secure channel. This observation strengthens our previous observation that newer mobile devices (possibly with higher computational power) are more secure in term of HTTPS redirection.

6 DISCUSSION & LIMITATIONS

This work is a first step at answering the question of the security impact of mobile web optimization. Our results indicate that in many instances there is a serious disconnect in web deployment with regard to mobile and desktop versions of websites that share common server resources. We cannot attribute these inconsistencies simply to mobile optimization but we believe that the rush to mobile optimization certainly plays a part. We also note the level of complexity involved in deploying websites that must consistently conform to mobile and desktop form factors.

Similar to other works [21, 23, 24] that utilize web crawlers, our approach is limited by the fact that websites are very dynamic, and a one-size-fits-all crawler that can automatically register or log in to all websites is very challenging. As a result, our crawler only visits the first landing page, so our reported evaluation only shows a lower bound of inconsistencies. However, despite this limitation, our study highlights an important problem which we hope will lead to further research in this area at the intersection of mobile and desktop web security.

Some of the attacks we present require forcing the victim to an alternate UI view or insecure channel, which could be very noticeable and raise suspicion. Nevertheless, prior works showed that users can be oblivious to security clues presented in browsers or websites [6, 26]. Additionally, showing a desktop-optimized UI on a mobile device is a very common occurrence since not all websites are mobile-ready.

Another limitation is that we assume that user agent spoofing is enough to make servers distinguish requests from different platforms. Our data collection could be enhanced by using real or emulated mobile devices since some web servers perform more robust fingerprinting. Nevertheless, our method is an accepted approach as used in previous works [21, 23, 24].

Some websites may omit security headers where they may already implement defense mechanism through other means such as built-in code. Since we only consider observed security headers, we cannot make any assertions that a website is more vulnerable

if certain headers are not present. For CORS security headers, we only consider non-credentialed requests.

7 RELATED WORK

We are not the first to find issues with web security headers, but to our knowledge we are the first to analyze the inconsistencies in how security headers are used across different user agent settings for mobile and desktop browsers. Our work is motivated by insights of previous work that show the various issues related to web security and the efforts to mitigate these problems, especially through HTTP headers. We are also motivated by the recent surge of mobile optimized website and seek to fill the gap into understanding the implications of web design and deployment with regard to the security guarantees that should be consistent regardless of the connecting user agent.

Web Security Jim et al. [19] was among the first to propose the use of HTTP Headers as a means of allowing web servers to instruct browsers regarding security policies. Their Browser-Enforced Embedded Policies work, based on writings by Markham [22], later served as motivation for the development of CSP [22]. Content security is primarily aimed at preventing XSS attacks, but can generally be used to restrict the source of content loaded onto a website, thereby enhancing the security of the website. CSP, now in its third iteration, has not gained much traction in adaptation on the web except for more popular and highly ranked website. Weichselbaum et al. [33] found that a large majority of deployments are ineffective. In this work, we use some of their analysis methodology, but we compare the efficacy between two versions of what should be the same configuration between mobile and desktop browsers on the same website.

Man-in-the-middle attacks have long been a nuisance on the web. Many previous works have proposed approaches to mitigate MITM and eavesdropping attacks on the web by enforcing strict HTTPS connectivity. Sivkorn et al. showed that many websites still find it necessary to serve content over HTTP [29] due to compatibility and other reasons. They found a series of implementation flaws and deployment issues in mechanisms used to enforce ubiquitous HTTPS, leading them to conclude that users are exposed to significant threat due to incomplete deployment of HTTPS. We leverage this observation in our work and found that there is also many instances of inconsistent deployment of HTTPS across user agents for the same domain.

Other previous works such as SSLock [14], HTTPSLock [15], and ForceHTTPS [18] all seek to mitigate the data exposure problem of unencrypted HTTP connections. Recently, HSTS [17] has gained much traction as a promising approach to enforce strict HTTPS connectivity. However, as shown by Kranch et al. [20], problems still persist with HSTS, especially in terms of the misunderstanding and inconsistent configuration in deployment. We also found in our study that many web servers inconsistently apply HSTS across different user agent settings, leading to weaker deployments for one user agent over the other.

Chen et al. [12] studied the ambiguities and inconsistencies in implementation and interpretation when multiple Host headers are defined for a web response. The Host header is critical in defining the origin that is used by the client to enforce the web's same origin policy [8]. We similarly focus on headers, but we look at the inconsistencies due to their configuration for different user agents.

Cookies Sivakorn et al. [30] explored the issue of cookie hijacking over HTTP connections. They found that many websites still avoid HTTPS due to performance and compatibility issues, but this results in cookie hijacking vulnerabilities. We also find that cookies are vulnerable to hijacking when HTTPS is not consistently enforced across different user agents. As a result, our approach uncovers issues of privacy leakage even when HTTPS is fully deployed under one view of a website, but not the other.

Zheng et al. [34] explored problems with cookie integrity. Their work is based on the known fact that cookies' secure flag can be set or reset through an insecure connection. Many of their techniques for exploiting cookie integrity can be leveraged to exploit the inconsistencies found in our work. Specifically, cookie overwriting, and cookie shadowing attacks can be carried out by attackers where inconsistencies show that the integrity of cookies can be violated. Singh et al. [28] also measured the use of secure flags, and found that cookies are vulnerable to information leaks. They showed that incoherent SOP policies [8] applied to cookies can cause inconsistent browser states. In this work, we show that inconsistent cookie deployment between user agents can lead to information leaks through cookies.

8 CONCLUSION

We conducted the first measurement study of inconsistencies between alternate mobile and desktop HTTP response configuration of a website. We analyzed the security issues due to inconsistencies in implementation of HTTP security headers, cookies, and HTTPS redirection. We found that subtle, but critical inconsistencies are pervasive among many of the top websites. These inconsistencies can lead to a number of attacks with real world impact to user's private data.

We found that these inconsistencies emerge due to the complexity and flexibility of web deployment to support different browser form factors, including desktop browser and mobile browsers. In some cases, these inconsistencies arise due to simple oversight or mis-communication among different teams developing and deploying complex web applications. Our work exposes the need for careful coordination between mobile and desktop website deployments so that subtle differences in configuration are not neglected to inadvertently expose users to attack. Despite the limitations of our data collection, we show that high numbers of inconsistencies exist that can lead to real world attacks on popular websites using several different attack scenarios. Our work enhances the web community's understanding and appreciation of the need for balancing performance vs. security issues as it relates to mobile web optimization.

ACKNOWLEDGEMENT

This material is based upon work supported in part by the National Science Foundation (NSF) under Grant no. 1314823. Any opinions,findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

REFERENCES

[1] Alexa Top Sites. http://www.alexa.com/top-sites.

[2] ComScore survey. http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/.

[3] Fortinet, Inc. http://www.fortinet.com.

[4] OWASP Secure Headers. https://www.owasp.org/index.php/List_of_useful_HTTP_headers.

[5] Devdatta Akhawe, Adam Barth, Peifung E Lam, John Mitchell, and Dawn Song. 2010. Towards a formal foundation of web security. In *Computer Security Foundations Symposium (CSF), 2010 23rd IEEE*. IEEE, 290–304.

[6] Devdatta Akhawe and Adrienne Porter Felt. 2013. Alice in Warningland: A Large-Scale Field Study of Browser Security Warning Effectiveness. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*. USENIX.

[7] Devdatta Akhawe, Warren He, Zhiwei Li, Reza Moazzezi, and Dawn Song. 2014. Clickjacking Revisited: A Perceptual View of UI Security. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*. USENIX Association, San Diego, CA. https://www.usenix.org/conference/woot14/workshop-program/presentation/akhawe

[8] Adam Barth. 2011. The web origin concept. (2011).

[9] Adam Barth, Juan Caballero, and Dawn Song. 2009. Secure content sniffing for web browsers, or how to stop papers from reviewing themselves. In *Security and Privacy, 2009 30th IEEE Symposium on*. IEEE, 360–371.

[10] Adam Barth, Collin Jackson, and John C Mitchell. 2008. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 75–88.

[11] Kyle Bowen and Matthew D Pistilli. 2012. Student preferences for mobile app usage. *Research Bulletin)(Louisville, CO: EDUCAUSE Center for Applied Research, forthcoming), available from http://www. educause. edu/ecar* (2012).

[12] Jianjun Chen, Jian Jiang, Haixin Duan, Nicholas Weaver, Tao Wan, and Vern Paxson. 2016. Host of Troubles: Multiple Host Ambiguities in HTTP Implementations. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16)*. ACM, New York, NY, USA.

[13] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. 1999. *Hypertext transfer protocol–HTTP/1.1*. Technical Report.

[14] Adonis PH Fung and KW Cheung. 2010. SSLock: sustaining the trust on entities brought by SSL. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ACM.

[15] A. P. H. Fung and K. W. Cheung. 2010. HTTPSLock: Enforcing HTTPS in Unmodified Browsers with Cached Javascript. In *Network and System Security (NSS), 2010 4th International Conference on*.

[16] Robert Hansen and Jeremiah Grossman. 2008. Clickjacking. *Sec Theory, Internet Security* (2008).

[17] Jeff Hodges, Collin Jackson, and Adam Barth. 2012. Http strict transport security (hsts). *URL: http://tools. ietf. org/html/draft-ietf-websec-strict-transport-sec-04* (2012).

[18] Collin Jackson and Adam Barth. 2008. Forcehttps: protecting high-security web sites from network attacks. In *Proceedings of the 17th international conference on World Wide Web*. ACM.

[19] Trevor Jim, Nikhil Swamy, and Michael Hicks. 2007. Defeating script injection attacks with browser-enforced embedded policies. In *Proceedings of the 16th international conference on World Wide Web*. ACM, 601–610.

[20] Michael Kranch and Joseph Bonneau. 2015. Upgrading HTTPS in mid-air: An empirical study of strict transport security and key pinning.. In *NDSS*.

[21] Sebastian Lekies, Ben Stock, Martin Wentzel, and Martin Johns. 2015. The unexpected dangers of dynamic javascript. In *24th USENIX Security Symposium (USENIX Security 15)*. 723–735.

[22] Gervase Markham. 2007. Content restrictions. *Website, version 0.9* (2007).

[23] Abner Mendoza, Kapil Singh, and Guofei Gu. 2015. What is wrecking your data plan? A measurement study of mobile web overhead. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*. IEEE, 2740–2748.

[24] Christopher Olston and Marc Najork. 2010. Web crawling. *Foundations and Trends in Information Retrieval* 4, 3 (2010), 175–246.

[25] Gustav Rydstedt, Elie Bursztein, Dan Boneh, and Collin Jackson. 2010. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. *IEEE Oakland Web* 2 (2010), 6.

[26] Stuart E Schechter, Rachna Dhamija, Andy Ozment, and Ian Fischer. 2007. The emperor's new security indicators. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*. IEEE, 51–65.

[27] David Silver, Suman Jana, Dan Boneh, Eric Chen, and Collin Jackson. 2014. Password Managers: Attacks and Defenses. In *23rd USENIX Security Symposium (USENIX Security 14)*. USENIX Association, San Diego, CA, 449–464. https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/silver

[28] Kapil Singh, Alexander Moshchuk, Helen J Wang, and Wenke Lee. 2010. On the incoherencies in web browser access control policies. In *2010 IEEE Symposium on Security and Privacy*. IEEE, 463–478.

[29] Suphannee Sivakorn, Angelos D Keromytis, and Jason Polakis. 2016. That's the Way the Cookie Crumbles: Evaluating HTTPS Enforcing Mechanisms. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*. ACM, 71–81.

[30] Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis. 2016. The Cracked Cookie Jar: HTTP Cookie Hijacking and the Exposure of Private Information. In *2016 IEEE Symposium on Security and Privacy*. IEEE.

[31] Sid Stamm, Brandon Sterne, and Gervase Markham. 2010. Reining in the web with content security policy. In *Proceedings of the 19th international conference on World wide web*. ACM, 921–930.

[32] Anne Van Kesteren and others. 2010. Cross-origin resource sharing. *W3C Working Draft WD-cors-20100727* (2010).

[33] Lukas Weichselbaum, Michele Spagnuolo, Sebastian Lekies, and Artur Janc. 2016. CSP Is Dead, Long Live CSP! On the Insecurity of Whitelists and the Future of Content Security Policy. In *Proceedings of the 23rd ACM Conference on Computer and Communications Security*. Vienna, Austria.

[34] Xiaofeng Zheng, Jian Jiang, Jinjin Liang, Haixin Duan, Shuo Chen, Tao Wan, and Nicholas Weaver. 2015. Cookies Lack Integrity: Real-World Implications. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, Washington, D.C.