Meddling Middlemen: Empirical Analysis of the Risks of Data-Saving Mobile Browsers

Brian Kondracki[†], Assel Aliyeva[‡], Manuel Egele[‡], Jason Polakis[⋆], Nick Nikiforakis[†]
[†]*Stony Brook University*
[‡]*Boston University*
[⋆]*University of Illinois at Chicago*

***Abstract*—Mobile browsers have become one of the main mediators of our online activities. However, as web pages continue to increase in size and streaming media on-the-go has become commonplace, mobile data plan constraints remain a significant concern for users. As a result, data-saving features can be a differentiating factor when selecting a mobile browser. In this paper, we present a comprehensive exploration of the security and privacy threat that data-saving functionality presents to users. We conduct the first analysis of Android's data-saving browser (DSB) ecosystem across multiple dimensions, including the characteristics of the various browsers' infrastructure, their application and protocol-level behavior, and their effect on users' browsing experience. Our research unequivocally demonstrates that enabling data-saving functionality in major browsers results in significant degradation of the user's security posture by introducing severe vulnerabilities that are not otherwise present in the browser during normal operation. In summary, our experiments show that enabling data savings exposes users to (i) proxy servers running outdated software, (ii) man-in-the-middle attacks due to problematic validation of TLS certificates, (iii) weakened TLS cipher suite selection, (iv) lack of support of security headers like HSTS, and (v) a higher likelihood of being labelled as bots. While the discovered issues can be addressed, we argue that data-saving functionality presents inherent risks in an increasingly-encrypted Web, and users should be alerted of the critical savings-vs-security trade-off that they implicitly accept every time they enable such functionality.**

I. Introduction

As smartphones have reached a near-ubiquitous presence, with widespread adoption even in developing countries [15], mobile devices account for almost half of the global Internet traffic [19]. Furthermore, there is a considerable trend towards the increased consumption of multimedia resources [67], which directly conflicts with mobile data plans that limit the amount of traffic allowed. Prior work has reported that users address such limitations by altering their usage behavior and planning for saving data for the future [47] (albeit not always effectively [67]). This indicates that data-plan limits remain an issue; as such, it comes as no surprise that mobile browsers that purport to "save data" are extremely popular among users. Even Chrome, the browser with the largest market share [58], has recognized the benefits of data savings for offering users a smoother and faster browsing experience [4].

These browsers, hereby referred to as data-saving browsers (*DSBs*), divert users' traffic through proxy servers that mediate web requests, handle application-level logic, and return compressed static pages and resources. This results in reducing the volume of users' network traffic as well as the computational overhead for the client device. Free DSBs in Android are extremely popular, with installations ranging between 1 million (Ninesky) and 5 billion (Chrome). Yet, no study exists on the risks that data-saving practices pose to all these users.

In this paper, we aim to address this gap by conducting a comprehensive empirical analysis of the DSB ecosystem. To that end we first manually analyze 121 browsers from Google's Playstore, and study their behavior to identify browsers that offer data savings, resulting in a dataset of nine data-saving browsers (8 free and 1 paid). These browsers form the basis of our analysis, which aims to quantify the possible degradation in security that occurs when users enable data-saving modes in mobile browsers. We analyze DSBs across five complementary dimensions: infrastructure, encryption, protocol headers, application behavior, and user experience. Across all five dimensions, we search for both malicious and benign behaviors which adversely affect user security and privacy.

We first conduct an exploratory investigation to map out the network infrastructure supporting each DSB, and find multiple cases of gateway proxy servers running severely outdated software. To deliver on the promise of data savings, the network infrastructure of DSBs must transform (e.g., compress) the data that is sent to clients. However, this content modification is diametrically opposed to the design goals of transport-layer-security (TLS) which aims to guarantee the confidentiality and integrity of data in transit. To modify response data, these DSB proxy servers terminate a client's TLS connection and establish a separate TLS session with the end web server. This TLS interception can void many of the security guarantees that an end-to-end deployment of TLS would provide [32].

Intuitively, if the TLS stack of the proxy has a "weaker" security posture than that of the client, a DSB's infrastructure can compromise the user's security and privacy. Our experiments reveal considerable divergence in capabilities and behavior when enabling data saving, which introduces severe vulnerabilities. Four of the DSBs offer more weak cipher suites while significantly reducing the number of strong cipher suites offered; in Opera, the High and Extreme data-saving modes exhibit a 55.5% and 60% reduction in the number of strong cipher suites. To make matters worse, we find that Opera with data-saving enabled fails to detect SSL certificates issued for an incorrect subdomain, which can be exploited for various attacks. Even more alarmingly, we find that Opera in data-

saving mode fails to warn users of web servers providing SSL certificates signed by the infamous SuperFish CA [20]. We demonstrate a practical man-in-the-middle (MITM) attack that exploits this to intercept all traffic bound for a target domain when requested by Opera's data-saving proxy servers, while visual security clues are still shown to the user (e.g., lock).

To assess the impact of TLS interception beyond insecure TLS stacks, we establish a testing pipeline that compares the content served by a web server with the data that arrives at the end user's device after the proxy optimizes it. We discover that Opera injects CSS into HTML responses for blocking advertisements. We also find that, contrary to Google Play Store guidelines, Opera Mini transmits a persistent identifier (OperaID) in conjunction with the device's `advertisingID` and IMEI, which nullifies the privacy a user can gain by resetting the `advertisingID`. Additionally, we check for data-leaking to third parties (accidentally or on purpose) due to TLS interception, but find no evidence that the studied browsers leaked any data for the duration of our experiments.

Furthermore, we highlight how data-saving functionality undermines the security guarantees of web applications by ignoring common security headers. We find that Opera Mini in High data-saving mode, among other DSBs, ignores headers such as X-Frame Options, Content Type Options, and CSRF Tokens. Finally, we leverage Google's reCAPTCHA service to infer how web services perceive traffic originating from DSBs. We find that, in general, the reputation of DSB user traffic is affected by the proxy server they happen to connect to, a decision made without the user's discretion.

Our experiments reveal that enabling the data-saving mode in browsers is equivalent to browsing the web through a broken looking glass, as users' experience is adversely affected across multiple dimensions. The severe vulnerabilities that our research has uncovered in major browsers with hundreds of millions of users demonstrate that data-saving functionality weakens established security practices, in return for minimal traffic savings, according to our measurements of Alexa Top 100 sites and a sample of streamed video content.

In summary, our research contributions include:

- We present the first empirical analysis on the security risks introduced by data-saving functionality in popular mobile browsers. We build an automated testing framework comprised of actual mobile devices.
- We conduct a comprehensive security investigation of the DSBs available for Android and shed light on their inner workings. Our experiments reveal a series of flaws, misconfigurations, and problematic actions that significantly impact the security and privacy of users' communications.
- Due to the severity of our findings, we have started the responsible disclosure process with browser vendors. While HTTPS interception is inherently fraught with risks, we outline guidelines to assist future DSB designs.

II. Background

Data-saving browsers. To conserve precious data volume (e.g., on limited mobile data plans) many popular mobile browsers offer a data-saving mode. Savings are accomplished by routing user traffic through dedicated proxy servers which make requests on behalf of the user and modify/compress the responses before returning them to the user's device.

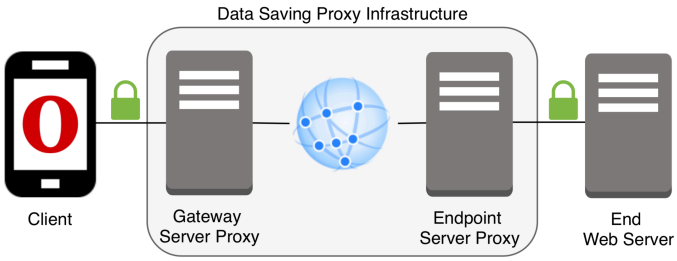

Fig. 1: *HTTPS request interception components of DSBs.*

The core intuition behind DSBs is that fast and powerful proxy servers located in data centers (often physically closer to the web servers) can download the full-sized resources, potentially execute all required logic such as scripts, and eventually return smaller, static pages to users. Through this mode of operation, data-saving browser vendors promise users data savings of up to 90% [6]. In contrast to typical HTTP proxies, or so-called *anonymizing browsers*, whose purpose is to mask the identity or location of a user, DSBs proxy user traffic to effectively reduce the size of requested payloads in-transit. All the browsers included in our study, make the user's IP address readily available to the end web servers through the `X-Forwarded-For` HTTP header, and hence do not advance user privacy.

HTTPS interception. Porter Felt et al. [35] reported that adoption of HTTPS across the Web is constantly growing. Consequently, data-saving browsers need to be able to process HTTPS traffic so as to provide sufficient data savings to users. As such, resources requested over secure HTTPS channels need to be compressed, which requires the capability to intercept and inspect users' encrypted HTTPS traffic. Unlike regular proxy servers, which simply route HTTPS traffic between the source and destination without any knowledge of the encrypted data being transmitted, data-saving browsers split the encrypted end-to-end connection in two parts. The browser on the user's device initiates a TLS handshake with a chosen proxy server upon enabling the data-saving mode. All subsequent HTTPS requests are sent to the proxy server where they are decrypted, read, and re-issued by the proxy server on behalf of the user. Visual cues such as the URL lock icon remain unchanged during this process, which can lead users to assume that the secure communication channel between the device and end web server is never broken. We present an overview of this process in Figure 1.

Like traditional web browsers, data-saving browsers are responsible for establishing an HTTPS connection with sites that support it by completing a TLS handshake with the web server and verifying the integrity of the presented certificate – in cases of errors, users should be explicitly alerted as would be the case during a normal browsing session. However, unlike traditional web browsers, the data-saving pipeline introduces additional entities (i.e., the intermediate

proxy servers) that handle and modify both application and protocol-level information. The combination of pivotal server positioning, additional functionality, and content modifications renders this ecosystem a minefield where flaws could have severe security and privacy implications for millions of users. This necessitates a holistic analysis of data-saving browsers that explores all layers of their operation; from their network infrastructure up to application-level behaviors. We seek to determine the extent of both malicious as well as buggy behaviors present in the ecosystems of each identified data-saving browser. Note that we limit our study to the introduction of such behaviors due to data savings. The security of mobile browsers in general has been explored in past research [22], [44], [45] and is thus outside the scope of this paper.

The ability of users to consent/opt-in to data saving differs across the studied browsers. Most browsers we studied enable data-saving by default, leaving users to manually opt out. Of this group, Yandex, Opera, and Opera Mini, set their data-savings mode to "automatic" by default, leaving the decision of which traffic to proxy, up to the browser. Puffin Free and Puffin Premium allow users to select if data savings should be applied to all traffic, limited to mobile or WiFi, or disabled. However, we discovered that whether data saving is enabled or not, these browsers always proxy user traffic.

III. Experimental Setup and Methodology

In this section we first present our DSB selection process. We then describe the infrastructure we set up for exploring the DSB ecosystem, and then detail the methodology and motivation behind each experiment.

A. Proxy Server Identification and Collection

Prior to studying the DSB ecosystem, it is important to formulate a definition of what a data-saving browser is. For the scope of this paper, we are interested in browsers which use proxy servers to compress, transcode, or filter traffic in-transit for the purpose of saving users' mobile data. This definition removes all browsers which simply proxy traffic (e.g., for anonymization or bypassing geo-blocking).

Using this definition, we conducted a comprehensive search of all data-saving browsers on the Google Play Store. To this end, we downloaded the complete set of 121 apps returned for the search terms: "Proxy Browser", "Data Saving Browser", "Cloud Browser", and "Internet Browser". To limit the set to browsers that proxy network traffic, we manually installed each APK file onto an Android device, and enabled any setting that resembles that of "Data Saving Mode". Subsequently, we visited a web server under our control and considered a browser as a *Proxy Browser* if the IP address connecting to our server was different from the Android device's address.

While proxy browsers all route user data through proxy servers, we are only interested in browsers that do this for the purpose of saving data. To further filter our dataset down to only include browsers that provide data saving functionalities, we manually inspected the Google Play Store descriptions of each proxy browser for any mention of data savings. This left us with our final dataset of 9 *Data-Saving Browser* apps listed in Table I. While certain browsers (e.g., Opera Browser and Opera Mini or UC Browser and UC Mini) are from the same vendor, our experiments reveal differences in their behavior as well as their infrastructure, as detailed in Section IV. As such, we present all DSB apps independently. Furthermore, note that data savings in Opera Mini can be configured in "High" or "Extreme" mode. As this choice has severe implications on how data savings are achieved, we treat these two settings separately. The most pronounced difference between the two modes is that in "High" mode the client and the gateway proxy communicate via regular HTTP, where the proxy provides volume optimized HTML content in its responses. In "Extreme" mode however, the proxy performs all rendering activities, including script execution, and merely sends the resulting rendering tree to the client. We mark Google Chrome with a "*" because although it does proxy HTTPS traffic, it is distinct in that it does not proxy any origin-scoped data (e.g. cookies and local storage). With major browser manufacturers adding data-saving features to their browsers, DSBs serve a significant portion of the overall Android user base.

Browser Name	Downloads	Proxy HTTPS	Data saving by default
Chrome	5,000,000,000+	**Yes***	No
UC	500,000,000+	No	Yes
UC Mini	100,000,000+	No	Yes
Opera	100,000,000+	**Yes**	Yes
Opera Mini	100,000,000+	**Yes**	Yes
Yandex	100,000,000+	No	Yes
Puffin	50,000,000+	**Yes**	Yes
Ninesky	1,000,000+	No	No
Puffin Premium	100,000+	**Yes**	Yes

TABLE I: *Data-saving browsers available on the Google Play Store.*

B. Measurement Infrastructure

To accurately measure the effect of data-saving modes on the browsing experience and security posture of users, we perform all experiments on real devices: Motorola X4, Nexus 6P, and Nokia 6 all running Android 8.1 (Oreo). All devices were updated and running the latest version of each browser before testing. We limit the scope of this work to DSBs present on Android mobile devices and leave exploration of similar browsers on other platforms to future work.

Our devices are orchestrated by an automation framework that uses the Android Debug Bridge (ADB) to handle all user input (touch, swipe, text, etc.). Unless stated otherwise, our system starts each test case with a *clean slate*, by clearing the application data of the DSB under test. The framework sends touch inputs to the device to launch each browser and enable the data-saving mode if required. Lastly, the framework navigates the browser to a list of webpages, each specially designed to explore a specific dimension of the DSB ecosystem and to shed light on the inner workings of each browser (see Section III-H). Figure 2 depicts a high-level view of our framework and experimental pipeline.

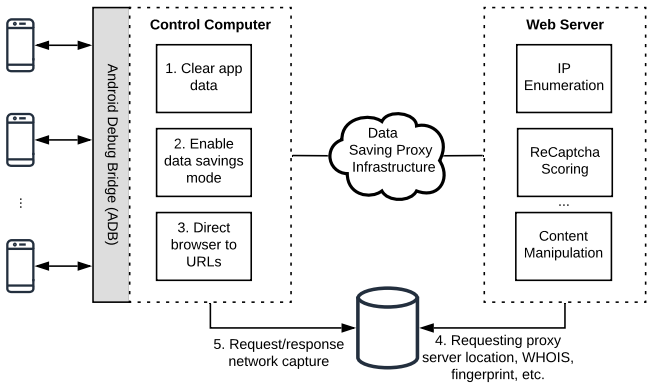

Fig. 2: *Architecture of our framework and infrastructure for auditing data saving browsers.*

Since we follow a black-box testing approach to study the proxy server infrastructure of each DSB, we conduct our data collection from the two vantage points of a web client and end web server. For example, we record the network traffic seen at the user's device using either a decrypting TLS man-in-the-middle proxy or dynamic instrumentation of the browser. At the same time, our end web server records the IP address, geolocation, and Autonomous System of the requesting endpoint proxy server.

C. Quantifying Data Savings

A data-saving feature can be the differentiating factor that tips the scale in favor of a DSB over a traditional web browser. Due to the substantial costs of mobile data plans (e.g., Google's Fi service charges $10 per gigabyte [37]), and the abundance of high-volume media for consumption (e.g., streaming video [34]), users can be enticed by claims from DSBs of reducing users' data consumption by up to 90% [6]. To quantify their effect, we measure the amount of data used by each browser when the data-savings mode is enabled and disabled, while visiting the Top 100 websites according to Alexa (average consumption over five rounds). For browsers with built-in, enabled-by-default ad blockers, we disable the blocker to better understand the data savings potentially gained from other types of resources (e.g., images and videos). Besides measuring data savings during regular web browsing activity, we also quantify the data savings that DSBs provide when streaming video. To this end, we navigate each DSB to a series of ten 3-minute long videos hosted on YouTube and measure the effect on the consumed data volume with data savings enabled and disabled. While we cannot entirely simulate a typical user's browsing patterns, our use of Alexa Top 100 sites and video streaming is a best-effort approach to understand how DSBs handle content typically consumed by the average user.

D. Proxy Server Ecosystem Enumeration

As the quality-of-service that a DSB user can expect depends on the proxy server infrastructure, it is important to quantify the number and location of the involved servers. This allows us to reason both about the magnitude of the DSB ecosystem and its underlying infrastructure, as well as its resilience to Denial of Service attacks and other geography-specific issues (e.g., link-flooding attacks [43]).

To construct a map of the proxy infrastructure, we use each DSB to request a web page from a server under our control, and record information about each request. Our experimental framework combines multiple vantage points to record information from both the client-side device and at the server. Through this two-pronged approach we obtain a view that cross-cuts multiple layers of activity on both ends of the proxy server infrastructure, allowing us to learn what happens to user traffic between the client device and web server. Specifically, we record the following for each request: (i) the IP address of the proxy server making requests to our web server (we refer to this as the *endpoint* server address), (ii) the IP address of the proxy server the browser directly connects to (referred to as the *gateway* server address) – we only record a gateway server if its IP address is different than that of the endpoint server, and (iii) WHOIS and IP location information of the identified servers. Finally, we also leverage NMAP to fingerprint endpoint servers and gateways. This information gives us a better understanding of the architecture illustrated in Figure 1.

To obtain an accurate picture of the DSB ecosystem, and explore proxy-server coverage across different geographic markets, we use a VPN service to change our device's location when connecting to the data-saving browsers. We select 4 countries (United States, Germany, Japan, and Brazil) to obtain diverse and accurate representations of the infrastructure of DSBs in the largest markets that these browsers serve.

Proxy Server Fingerprinting. In conjunction with determining the number and distribution of the proxy servers that support each browser, we also need to understand what types of services are enabled on each proxy server. This allows us to compare the proxy-server configurations of different DSBs, the homogeneity of their supporting infrastructure, and whether these servers are exposing known vulnerable software to the public web. The presence of outdated or vulnerable software on these servers presents a significant threat to the DSB ecosystem, as these servers constitute *single points of failure* for millions of users. Moreover, as many of these proxy servers terminate TLS connections, they have access to plaintext data, such as usernames, passwords, and session cookies. Hence, an attacker who successfully compromises one or more of these proxy servers can effectively MITM all user traffic that flows through them to and from arbitrary end web servers.

Our system fingerprints proxy servers by performing two NMAP [9] scans on the proxy servers it encounters. We refer to them as *basic* and *verbose* scans. The basic scan uses NMAP's TCP SYN scan to infer open ports on a given proxy server and the name of the software listening on that port (if available). We repeat this scan every time we observe an IP address in our proxy connections. To gain an understanding of a proxy's configuration beyond the list of open ports, we further use NMAP's banner-grabbing capabilities as the verbose scan. Since this scan involves more interactions with a

target server than just sending a SYN packet, we only verbose-scan each proxy's IP address the first time it is encountered. We perform these scans from the web server under our control each time the server receives a request for our test web page.

Ethical considerations. From an ethical perspective, both types of scans merely gather information and do *not* attempt to exploit any service in any way; this is in line with widely used prior work, e.g., the ZMap and Censys Internet-wide scans [31], [33]. We opt to collect our own data (instead of relying on third-party services like Censys) as we need to ensure that the host being scanned is the host involved in proxying the DSB's connections. The practices of IP churn and reuse (e.g., in public clouds) suggest that we cannot rely on historical data as published by prior works. We are confident that our fingerprinting approach did not adversely affect the proxy's performance or their users.

E. ReCAPTCHA

Captchas are often the first line of defense against automated malicious activities. Captchas are challenges that are trivial for humans to solve, but difficult for computers [62] (e.g., distorted character recognition), and Google's reCAPTCHA is the most prevalent captcha service [54]. As recent research has demonstrated effective attacks against all traditional forms of captchas (text [66], image [56], audio [25]), Google progressed to a scheme that does not require users to complete a specific task. The release of reCAPTCHA v3 removed all need for explicit user actions, replacing it with a completely transparent, JavaScript "challenge". This new approach compiles user browsing patterns on a given page and returns a score that the site administrator can use to decide how to respond. This score ranges from 0 (very likely a bot) to 1 (very likely a human).

One potential signal employed by reCAPTCHA v3 is the reputation of a user's IP address. This lends the question: if users browse the web from a DSB, will they receive a lower reCAPTCHA v3 score if other users, sharing the same data-saving proxy servers, misbehaved or were deemed to be bots? If the question has an affirmative answer, then users of DSBs will, on average, experience a degraded web compared to using a traditional browser. According to reCAPTCHA's official guidelines, low scores should trigger actions like forcing a two-factor authentication challenge, email verification, throttling user actions, or flagging potentially risky transactions [7].

To understand how the use of a DSB affects reCAPTCHA scores we perform the following experiment, which aims to capture the experience of an average user that starts a browser session using a DSB and accumulates some cookies and browsing history before visiting a reCAPTCHA-protected page. During regular browsing hours, which we define as between 8am-6pm, our system periodically (once every 30 minutes) picks a random browser, clears all previously generated application data and begins a test. The device visits 5-10 random URLs from the Alexa Top 1K and navigates them using a randomly selected set of swipe gestures. The browser is then directed to a page under our control, where we record the public IP address of the endpoint proxy which requests the page, as well as the score returned by reCAPTCHA v3. In order to limit the effect IP reputation has on our base case (data savings mode disabled), we connect the test device directly to our lab network without the use of a VPN service.

F. Proxy Server Security Auditing

The inclusion of data-saving proxy servers between users and web servers presents a new level of complexity. Proxy servers and browsers must both agree on what level of security to provide to end users. A weak proxy server can introduce vulnerabilities that would not exist had the browser directly requested the web pages rather than fetch content through the proxy. While mobile browser security has already been studied extensively [22], [44], [45], we focus on the duality of data-saving enabled vs. data-saving disabled. If data saving is enabled, this means that a number of critical security operations (such as the verification of a TLS certificate) now happen at the proxy servers instead of the user's local device. To determine the implications of these two "competing" TLS stacks, we performed the following experiments:

TLS cipher suite support. TLS is the backbone of confidentiality and integrity for data in transit. During each connection, the client and server negotiate the cipher suite that will be used to facilitate encrypted communication over the course of the session. It is vital that a user's device advertises secure cipher suites as, per the protocol, the end server can only chose from suites offered by the client. When browsing with a DSB that terminates the SSL connection at the proxy server, users' security is limited by the strength of the cipher suites provided by the proxy server to the end web server, instead of the ones present on their device. For this experiment, we determine the number and strength of cipher suites offered by each browser with data savings enabled and disabled.

TLS certificate error handling. The splitting of encrypted HTTPS connections between the mobile browser and the web server forces the user to, usually unknowingly, trust the proxy server to not only properly verify TLS certificates, but also convey accurate information on the state of a domain's certificate back to the device. If the proxy server *mistakenly* reports a certificate as valid, the user would then be vulnerable to many attacks which would not be possible if the data-saving mode was disabled. To test these browsers, we first present them with certificates that have known errors [23], such as, self-signed certificates, expired certificates, and revoked certificates. Next we conduct experiments using certificates issued by authorities which are widely considered untrusted (such as, RapidSSL and GeoTrust [64]).

G. Content Leakage

For DSBs to work as advertised, proxy servers must have the ability to read and modify web pages and resources passing through them. This extends to HTTPS traffic, where sensitive information is more likely to be transmitted. To understand whether private user information is accidentally or intentionally leaked to third parties by these browsers, we perform a series of experiments focusing on three types

of sensitive data: visited URLs, information embedded in web pages, and credentials supplied to login forms. Clearly, evidence of using such sensitive data would be disastrous for a company's reputation on security and, as such, is unlikely to occur. Nonetheless, the incident with Microsoft's Skype service visiting all URLs mentioned in users' encrypted chats [57] prompted us to include such experiments.

URL leakage. The first dimension of leakage we measure is that of URLs being visited *after* being accessed through a DSB. It is important to note that in addition to the tunneling of HTTP requests, DSBs also tunnel DNS requests to their own resolvers. One issue that this introduces is the potential for a third party (either the proxy server or their chosen DNS resolver) to log the URLs and domains requested by users and visit them at a later time. These third-party actors could be malicious (e.g., intercepting DNS requests on a compromised resolver), or curious (e.g., server administrators of a particular browser). To test for this type of leakage, we visit unique URLs on a web server under our control. Each URL starts with a unique sub domain that encodes the browser name as well as the current date, and includes the current timestamp in place of a filename. Given that all URLs were unique and complicated enough so as to not be guessable by a bot, any later revisits to them can be confidently attributed to leakage.

Page content leakage. The second dimension of leakage we measure focuses on the content of HTTP responses, as user traffic includes highly sensitive information and transactions. We create a number of pages which appear to contain sensitive information from both the perspective of an automated program which may be scanning web pages, or a curious human that manually visits pages. These pages leak information such as credentials for our site, credentials for accounts on third party sites, and randomly-generated financial account numbers hidden behind obfuscated links that log all visits. We employ the same URL structure as in the previous experiment to avoid attribution of visits to unrelated bot activity.

Credential leakage. The last dimension of data leakage we examine is that of user credentials supplied to log in forms. As a number of DSBs tunnel all HTTPS content, they have access to the content of any such form. Thus, the existence of a malicious or curious actor located on a proxy server can compromise the accounts of users logging into their accounts through a DSB. To test this, we create a set of unique accounts for each DSB on Yahoo, Dropbox, and Proton Mail. We choose these services as they provide descriptive information on the locations and IP addresses of the devices that log into an account. We then use each browser to log into the accounts assigned to it. To differentiate our testing devices' login attempts from any unauthorized third parties, we record the IP address of the proxy server before logging in. Twice a day, we use Selenium to extract up-to-date login information for each account to detect potential accesses from third parties.

H. Content Manipulation

As DSBs compress and filter traffic flowing through their proxy servers, users are at the mercy of any changes the proxy server makes to the mediated server responses. These changes can be done in malice, such as injecting or modifying advertising IDs for redirecting revenue or tracking [38], or they can occur by mistake, such as a proxy server dropping HTTP headers it does not recognize. Regardless of the underlying reason, these modifications render the user vulnerable to a number of attacks. To determine the extent of such content manipulation, we visit a series of web pages under our control through a DSB while simultaneously directly visiting the same web page using the Python Requests library [52]. We then compare the two responses to detect any content changes.

On the end web server, which is under our control, we create a set of 57 web pages that return different combinations of security-sensitive HTTP headers. The pages also contain multimedia content and fake Google Ads and Amazon Affiliate advertisements. This setup allows us to record any HTTP headers that are added, modified, or deleted by the proxy, as well as any modifications made to the HTML content of each page. To ensure that we observe the response directly from the proxy server rather than what the DSB displays after client-side processing, we capture responses either on the network between the client device and proxy server or directly from the network APIs on the device. When neither is possible, we instrument the DSBs as described below.

Intercepting HTTP traffic. As shown in Table I, UC Browser and UC Mini only proxy HTTP traffic. Hence, we can obtain the proxied content via on-device, packet capturing. However, while the data itself is in plain-text, these browsers implement a custom protocol on top of HTTP to conserve bandwidth. As the protocol merely combines multiple HTTP responses into a single gzip compressed payload, it is straightforward to extract the underlying data. For Yandex and Chrome, the Polar Proxy [16] transparent SSL/TLS proxy is sufficient for capturing responses directly from the network. Note that Yandex does not intercept HTTPS traffic but *does* use an HTTPS connection between the device and the DSB proxies, for transferring modified HTTP traffic. The remaining browsers employ certificate pinning, which prevents this approach. To access their plain-text data, we instrument these browsers and capture the information before it is passed to the TLS library for encryption (when transmitting data), and after the TLS library decrypts cipher-text (when receiving data).

Android relies on the BoringSSL [3] library to provide a system-wide implementation and API for SSL/TLS. To capture the data that the browser receives, we focus on the `SSL_read` function which the browser uses to consume plain-text data from the library after it has been decrypted. Analogously, to transmit information, the browser passes plain-text data to the `SSL_write` function. Thus, by hooking into these functions, we can record sent and received data in plain-text. We insert these hooks into the browser's address space using the AndHook [1] dynamic instrumentation tool. To identify the location of the `SSL_read` and `SSL_write` functions in the system-wide library we examine its symbol table.

Even though the system-wide implementation of BoringSSL is available to all applications on an Android system via the

Browser	Web Browsing [MB]			Video Streaming [MB]		
	Off	On	% Saved	Off	On	% Saved
Ninesky	206	32	84.21%	100	120	-20.14%
Puffin Free	110	25	77.31%	288	336	-16.71%
Opera Mini (Extreme)	174	45	73.81%	108	116	-7.34%
Puffin Premium	91	31	65.4%	298	334	-12.14%
Opera Mini (High)	182	93	48.84%	108	75	30.85%
Opera Browser	200	107	46.53%	111	111	0.08%
Chrome	176	116	33.8%	108	112	-3.95%
Yandex	87	79	8.58%	108	108	-0.05%
UC Mini	154	159	-3.36%	107	107	0.04%
UC Browser	138	154	-12.15%	102	111	-9.17%

TABLE II: *Amount of data used and saved when browsing the top 100 Alexa sites and streaming ten 3-minute long Youtube videos with data savings disabled (Off) and enabled (On).*

`libssl.so` library, we observe that most browsers ship their own, statically-linked copy. Moreover, some browsers contain and use multiple distinct copies of this library. Complicating the matter further is the fact that the statically-linked versions of the libraries do not export symbols which makes identifying the addresses that need to be hooked challenging. To identify the necessary functions, we relied on a two-pronged approach. First, we use dynamic analysis to obtain the user-mode call stack when the `read` system call obtains cipher-text from the underlying socket. Since BoringSSL is implemented as an abstraction layer between the browser and a network socket, `SSL_read` is part of that call stack. Second, when we cannot reliably retrieve a correct call stack (e.g., because libunwind is unreliable on ARM64), we rely on static analysis and manually reverse engineer the libraries used by the browser.

Intercepting Non-HTTP(s) data. One of our interesting discoveries is that Puffin (Free and Premium) and Opera Mini (Extreme), render a web site on the proxy server and merely transmit the rendering tree to the client. A rendering tree cannot be meaningfully compared with the HTTP traffic that generated it. To assess whether the Puffin proxies modify content, we leverage the fact that Puffin and the stock Chrome browser use the same rendering technology. Thus, we manually perform a visual comparison between the same site rendered by Chrome in normal operation mode and Puffin. Furthermore, although there is no HTTP communication between the Opera Mini browser in the Extreme data-saving mode and its supporting proxy server, the proxy server provides an API to query the DOM that was rendered for the client. Hence, we use this information to identify potential modifications that Opera's proxy servers may introduce.

IV. Experimental Evaluation and Results

In this section we present our measurement results and empirical analysis of the data-saving browser ecosystem.

A. Data Savings Mode Quantification

The left part of Table II shows the amount of data used in downloads for each DSB when visiting the Alexa Top 100 web sites. Comparing the data consumption with data savings on vs. off, we observe that DSBs that proxy HTTPS traffic save users a substantial amount of mobile data (up to 84%). Given the recent push to an HTTPS-by-default web, we observe that browsers which do not intercept HTTPS traffic (i.e., Yandex, UC, and Ninesky), perform poorly in terms of data savings.

An unexpected finding is that UC Browser and UC Mini consume *more* data when data savings is enabled. Neither of these browsers tunnel HTTPS traffic so, like Yandex, they cannot offer any data savings for the majority of modern web traffic. We also observed that substantial traffic was sent from our devices and the gateway proxy servers of these browsers when data savings was enabled. This suggests that, due to their inability/unwillingness to handle HTTPS traffic, these two DSBs exchange more metadata for their data-saving operations than the data they actually save on regular user traffic.

Conversely, we recorded the Ninesky browser as saving users the greatest amount of data while also not proxying HTTPS traffic. Over the course of our testing, we noticed 7 of the Alexa Top 100 web sites would fail to load when data-saving mode was enabled using this browser. To prevent bias in our results, we removed these sites from our testing set for this browser, leaving a set of 93 pages to assess Ninesky's data savings capabilities, where Ninesky still saves its users 84.21% of their mobile data on average. These savings appear to be due to client-side blocking of certain resources such as advertisements, images, and videos. As such, we cannot confidently state that the users of this browser would benefit from this aggressive blocking of content or would eventually switch to another mobile browser. Unlike the other browsers in our study, Ninesky does not provide users the option to disable the blocking of advertisements. Thus, we were unable to isolate the source of data savings to its proxy servers.

Even though there clearly exist DSBs that are capable of saving data for users who browse the web, web browsing is not the main culprit of large data consumption. According to recent statistics, YouTube is responsible for 38% of worldwide mobile traffic [28]. Therefore, to assess whether these browsers can save data in a more realistic scenario, we visited ten 3-minute-long YouTube videos with data savings on and off, and measured the data consumption. The results (shown in the right part of Table II) indicate that only Opera Mini in High mode, due to more aggressive video compression, provides substantial data savings when streaming video, while all other browsers either provide negligible data savings or consume more data with data-saving mode enabled than disabled.

B. Proxy Server Ecosystem Enumeration

Proxy endpoint server enumeration. Figure 3 shows the CDF of the total number of proxy endpoint servers encountered by each browser across all VPN endpoint locations, visiting our web server with both HTTP and HTTPS over the course of a 30 day recording period. Opera Mini in extreme data savings mode used a total of 1,481 unique IP addresses to connect to our server, exhibiting one-to-two orders of magnitude more servers than the rest of the browsers. In general we see that most DSBs are banding together in the hundreds range, with Puffin premium being a negative outlier.

An interesting trend in this data involves the number of proxy endpoint servers encountered by browsers of the same

Browser	United States		Germany		Brazil		Japan		Total w/o Duplicates	
	HTTP	HTTPS	HTTP	HTTPS	HTTP	HTTPS	HTTP	HTTPS	HTTP	HTTPS
Opera Mini (Extreme)	312	392	325	366	284	363	334	411	983	1106
UC Mini	251	N/A	239	N/A	229	N/A	253	N/A	414	N/A
UC Browser	236	N/A	252	N/A	220	N/A	246	N/A	403	N/A
Opera Mini (High)	75	75	126	121	78	77	179	206	348	359
Opera Browser	74	76	120	120	71	75	172	201	341	357
Yandex	175	N/A	175	N/A	166	N/A	175	N/A	200	N/A
Puffin Free	100	117	95	103	93	90	90	86	173	178
Chrome	24	48	21	51	24	48	17	52	67	116
Ninesky	32	N/A	7	N/A	23	N/A	39	N/A	63	N/A
Puffin Premium	18	17	17	17	18	17	17	17	19	17

TABLE III: *Number of unique proxy endpoint server IP addresses per request protocol in each region.*

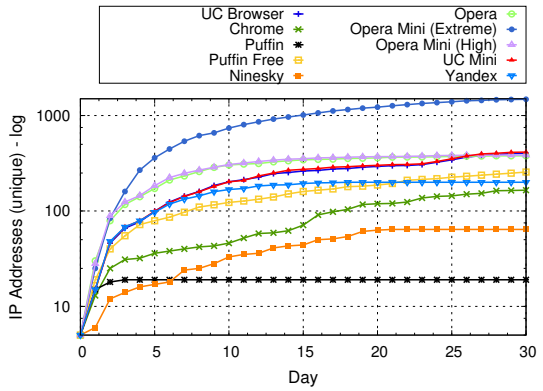

Fig. 3: *Unique proxy endpoint servers encountered with each DSB.*

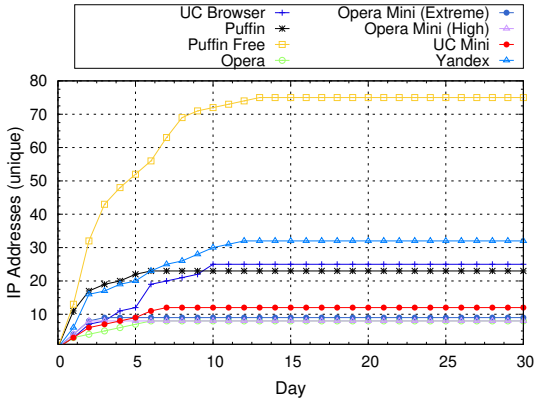

Fig. 4: *Unique proxy gateway servers encountered with each DSB.*

vendor. We can see that Opera Mini in high data savings mode uses the same number of unique endpoint servers as Opera Browser. This relationship also exists with the browser pair of UC Browser and UC Mini. Further, we calculate the Jaccard-Similarity index between the sets of proxy endpoint IP addresses of each browser pair. We find Opera Mini in high data savings mode and Opera Browser have a 93.7% similarity, with UC Browser and UC Mini having an 86.3% similarity. We can use this pattern to deduce a sharing of infrastructure between these pairs of browsers as well as a similar logic incorporated to determine the proper endpoint server to use. However, this relationship disappears when looking at Opera Mini in extreme mode as well as both Puffin Browsers. The extra proxy endpoint servers seen by Opera Mini in extreme mode can be attributed to a greater need for computing power due to the aggressive rewriting of web pages that occurs in extreme mode that is not present in High mode. However, the divide in available proxies available to Puffin Premium compared to Puffin Free requires an alternative explanation. We opine that Puffin Premium users are sent to one of 19 servers that are different than the 256 servers used in Puffin Free to potentially deliver different quality of experience to paying users. The low number of premium servers is likely because there are not enough paying users to justify a greater investment in server-side resources.

To gain geographical insights into the DSB ecosystem, we analyze our dataset with respect to each VPN location we used as well as contrast HTTP and HTTPS requests, in Table III. The sharing of proxy servers between families of browsers becomes more evident, as such browsers encountered the same amount of proxy endpoint servers with each of the request types. The divergence in infrastructure support between the top browsers is also clear. The Opera and UC browser families have strong support in all regions we tested. Conversely, browsers such as Yandex and Puffin have a single pool of proxies that are shared by users across regions. Lastly, for most browsers, we observe more servers handling HTTPS requests, to account for the ever-increasing adoption of HTTPS [35].

Proxy gateway server enumeration. Figure 4 displays the unique proxy gateway servers encountered by each browser. As described in Section III-D, these IP addresses were recorded during requests in which the IP address that the DSB directly connected to was different than the one that made the request to the end web server. Chrome and Ninesky are absent from this figure as their infrastructure uses the same IPs for gateway and endpoint servers. Our results show that the gateway servers are fewer than the endpoint servers suggesting that the former are used to route user traffic to endpoint servers possibly for load balancing or routing efficiency. Moreover, of the 165 distinct gateway servers, 79 are not present in our proxy endpoint dataset, indicating that operators put more resources on endpoint servers than gateways.

Proxy Server Fingerprinting Table IV shows the number of distinct basic configurations observed for all data-saving browsers that reply to port scans on their proxy endpoint servers. We consider the set of open port, listening software tuples as a proxy's configuration. Our findings suggest that Opera Mini in high data savings mode and Opera Browser

Browser	Configurations	Avg. Services
Opera Browser	23	18.7
Opera Mini (High)	28	18.9
Opera Mini (Extreme)	3	8.7
Puffin Free	3	1.3
Puffin Premium	1	2

TABLE IV: *Number of distinct proxy server configurations and average number of listening services for each data saving browser.*

are based on the same underlying technology due to the large overlap in their proxy-endpoint servers' configurations.

The discovered heterogeneity among the proxy-server configurations of Opera Browser and Opera Mini in high data-saving mode was unexpected. One would assume that commercial proxy servers would be created once and cloned to any endpoint machine required. However, our results suggest otherwise. Many of our scans report different Opera proxy endpoint servers hosting the same service on slightly different port numbers. For example, on one recorded proxy server, a specific service was listening on port 9001 while on another proxy server, the same service was listening on 9002. Our results show these differences are not due to setup error. Rather, many of these configurations exist across each of the regions where Opera datacenters are located. There appears to be a deliberate heterogeneity within a particular region, while also a homogeneity is exhibited across all global regions. What remains unclear is what purpose each of these configurations serve and what causes a user's traffic to be routed to a proxy server of a particular configuration.

We found proxy endpoint servers hosting many listening services, instead of limiting the number to the bare minimum to reduce the attack surface. Table IV also presents the average number of open ports on each DSB's endpoint proxy servers. While Puffin's proxy servers listen on the expected ports, Opera's listen on many ports and often have duplicated services listening on 10-15 different port numbers. Even though running multiple instances of the same service on varying ports does not expose servers to new attacks (compared to running different services at different ports), it still complicates the process of securing these servers via network-level defenses, such as, firewalls and intrusion-detection systems. Table V shows the outdated software we discovered listening on the gateway proxy servers of many DSBs. We found that a large percentage of DSB gateway proxy servers have listening services which are almost 6 years old. These outdated services are vulnerable to a wide range of attacks, such as, denial of service, privilege escalation, and directory traversal.

C. ReCAPTCHA

Figure 5 shows the distribution of reCAPTCHA v3 scores for all browsers on each day of our experiment period. We limit our reporting period to 7 days as beyond that time frame we record a convergence of returned scores with data savings enabled and disabled. This convergence informs us that reCAPTCHA has identified our test devices behind DSB proxy servers using methods such as browser fingerprinting. We

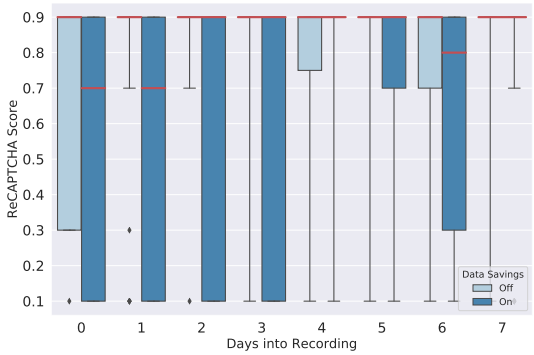

Fig. 5: *Daily reCAPTCHA v3 scores across browsers with data savings enabled and disabled (whiskers extended to 5% and 95% of dataset).*

leave further analysis of the effect DSBs have on reCAPTCHA scores over long time periods to future work.

We find a clear dichotomy in the reCAPTCHA scores returned for browsers with data savings mode enabled and disabled. Our results demonstrate that a user's browsing experience can vary drastically each time they use a DSB. According to Google's reCAPTCHA v3 guidelines [18], web site administrators should use scores below 0.5 as a baseline for bot activity. Assuming most web sites follow this baseline, users of DSBs will experience more captchas, rate-limiting, and even IP bans when data-saving mode is enabled.

As multiple users share the same DSB proxy, the reputation for a proxy depends on the aggregated behavior of all its users. Hence, it is possible that malicious users could negatively impact the web browsing experience of the benign majority. To evaluate this scenario, we conducted a proof-of-concept attack against Puffin. Using one device which we designated as the "attacker", we performed minor bot-like activity on a web site protected by a popular anti-bot service: after requesting the site's home page multiple times in a short time window, the attacker was shown a captcha. When visiting the same site through Puffin on the "victim" device (same smartphone model), it would immediately receive a captcha lasting for a short time frame (approximately 5 minutes). We empirically determined that this effect is only present if the victim and attacker (i) share the same proxy IP address, (ii) have the same device model, and (iii) visit the same web site in close temporal proximity. We determined the probability of a user meeting these criteria during our testing period is small enough to ensure no users were effected by our experiment. However, in the unlikely event that these criteria were met, the resulting effect would be a limited inconvenience to any users of the web site, who would only need to solve a captcha to continue browsing. While in practice attackers could trivially increase the volume of requests and significantly magnify the impact of the attack (e.g., using multiple popular devices), we did not attempt/explore this for ethical reasons.

D. Data Saving Mode Security Degradation

We found that proxy servers supporting DSBs typically do not provide the same level of security as traditional mobile

Browser	Outdated Software	Servers	Months Outdated	CVEs	Maximum CVE Score	Example CVEs
Yandex	nginx v1.8.1, v1.12.2, v1.14.2	14	28.3	3	7.8	CVE-2018-16843, CVE-2018-16844
UC Browser	nginx v1.10.1, v1.12.2, TwistedWeb v10.2.0	5	57.7	2.7	7.8	CVE-2018-16843, CVE-2018-16844
UC Mini	nginx v1.10.1, v1.12.2, TwistedWeb v10.2.0	4	57.7	2.7	7.8	CVE-2018-16843, CVE-2018-16844
Puffin Free	lighttp v1.4.35	4	68	1	5	CVE-2015-3200
Puffin Premium	lighttp v1.4.35	4	68	1	5	CVE-2015-3200
Opera Browser	nginx v1.10.2, v1.12.2	2	32.5	3.5	7.8	CVE-2018-16843, CVE-2018-16844

TABLE V: *Information about outdated software running on DSB gateway proxy servers. Numbers averaged across servers of each DSB.*

Browser	Savings On		Savings Off		Δ	
	Strong	Weak	Strong	Weak	Strong	Weak
Chrome	9	9	9	7	0	+2
Opera Browser	4	9	9	7	-5	+2
Opera Mini (High)	4	9	9	7	-5	+2
Opera Mini (Extreme)	3	10	9	7	-6	+3
Puffin Free	6	7	N/A	N/A	N/A	N/A
Puffin Premium	6	7	N/A	N/A	N/A	N/A

TABLE VI: *Strong and weak cipher suites offered by DSBs.*

browsers. Thus, when users enable the data-saving mode of a browser, they are immediately putting themselves at greater risk for the benefit of mobile data savings. In this section we discuss how the proxies that support DSBs affect two foundational aspects of TLS. First, we observe that proxies support a different and typically less secure set of cipher suites. Second, we highlight grave differences in handling certificate errors with data-saving enabled vs. disabled.

Supported Cipher Suites. Next we determine the extent of security degradation in each browser when data-saving modes are enabled. An interesting finding relates to the number of cipher suites offered by each DSB and its corresponding proxy infrastructure. Table VI lists the number of strong and weak cipher suites (as categorized by SSL Labs) offered by each browser with data-savings enabled and disabled. All browsers that support proxying HTTPS content introduce additional weak cipher suites while the majority also reduce the number of strong cipher suites. This choice of cipher suites puts the user at unnecessary risk as a variety of known attacks (e.g., POODLE [49]) can be launched against the known weak suites. Furthermore, while the Opera and Opera Mini browsers with disabled data savings support the more modern TLS version 1.3, their supporting proxy infrastructure does not.

SSL certificate error handling. The results of our experiments with respect to handling SSL certificate errors are summarized in Table VII. Recall that in this experiment we visited a number of sites where each site features a specific defect in the provided SSL certificate (e.g., expired, self-signed, etc.). We visited each site twice, once with data savings enabled, and once with data savings disabled. If a DSB correctly prevents the user from visiting a site with an erroneous certificate or displays a warning page, we mark the corresponding cell with a ✓ symbol. Further, we represent browsers which allow users to visit web sites with certificate errors but present them with a warning pop-up with a "**!**" symbol. Any browser that does not alert users of certificate errors on a page they are attempting to visit is labeled with a ✗ symbol. Lastly, any behavior that does not fall into the above three categories is labeled with an "N/A". For example, Opera Mini with data savings mode disabled prevents users from visiting sites providing a certificate signed by GeoTrust, but does not alert users of any errors. Although this behavior prevents immediate access to potentially harmful web sites, it leads to user confusion. We mark Puffin browsers with a "*" because they do not allow users to disable proxying.

First, we find that for the majority of errors, Opera and Puffin allow access and only show a warning, as opposed to Chrome that prevents access. As prior work has reported significant click-through rates when users are presented with SSL-related browser warnings [21], this approach is problematic. Opera Browser and Opera Mini in both data savings modes (High and Extreme) allow users to visit pages with certificates signed using the SHA1 hashing algorithm. Additionally, Opera Browser and Opera Mini in High data savings mode allow users to visit sites where the certificate supplied does not match the correct subdomain. We demonstrate this vulnerability in both browsers in demo videos [11], [13]. This could potentially allow *related-domain* attackers [26] to bypass the cookie-integrity checks recently adopted by browsers that rely on cookie prefixes [24]. In both cases, the user sees a lock icon in the URL bar indicating their connection to the web server is secure, when in reality it is not.

One of the most severe revelations resulting from this experiment is that Opera browsers in data-saving mode accept certificates from distrusted certificate authorities (CAs). A CA is considered distrusted if it has its private signing keys disclosed (e.g., SuperFish [20] which was never officially a CA yet signed certificates for all websites for adware-related reasons), or it has a known history of misissuing certificates (e.g., GeoTrust and RapidSSL [64]). Opera Browser and Opera Mini in High data savings mode accept the former, while all variants of Opera accept the latter. We demonstrate this attack against Opera Browser in a demo video [10]. Disabling data savings mode in Opera Browser removes the vulnerability and properly alerts users to the error. However, Opera Mini simply fails to load the potentially dangerous web page and does not alert the user as to the reason why. Accepting these untrusted certificates leaves users vulnerable to severe man-in-the-middle attacks against arbitrary websites.

We successfully executed this attack using a certificate generated for a domain under our control and signed with the SuperFish private key. Although creating the fake certificate is straightforward, to execute this attack, the adversary must also MITM connections in front of the Opera endpoint servers, e.g., by poisoning their DNS cache or performing selective BGP

	expired		wrong.host		self-signed		untrusted-root		sha1-intermediate		SuperFish		GeoTrust	
Data Savings	On	Off	On	Off	On	Off	On	Off	On	Off	On	Off	On	Off
Chrome	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Opera Browser	!	!	✗	!	!	!	!	!	✗	!	✗	!	✗	!
Opera Mini (High)	!	!	✗	!	!	!	!	!	✗	!	✗	!	✗	N/A
Opera Mini (Extreme)	!	!	!	!	!	!	!	!	✗	!	N/A	!	✗	N/A
Puffin Web Browser*	!	!	!	!	!	!	!	!	!	!	!	!	✗	✗
Puffin Premium*	!	!	!	!	!	!	!	!	!	!	!	!	✗	✗

Prevent access: ✓, Allow access: ✗, Warn but allow access: !

TABLE VII: *Differences between how DSBs handle certificate errors depending on whether the data saving mode is enabled or not.*

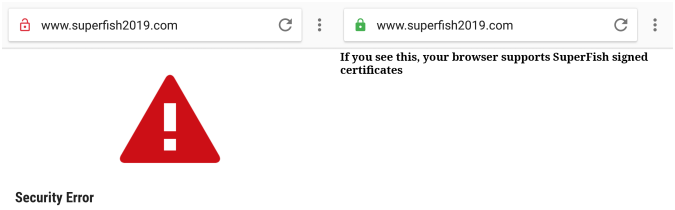

Fig. 6: *Opera behavior when visiting a site with a SuperFish signed certificate with data savings disabled (left) and enabled (right).*

hijacking. Prior work has demonstrated the effectiveness of DNS cache poisoning attacks [40], while security companies have recently issued reports about major DNS hijacking campaigns [2], [5]. We simulate DNS cache poisoning by modifying the authoritative DNS server for our domain to direct traffic to our "attacker" machine. This machine runs MITMProxy [8] and presents the SuperFish signed certificate to Opera's proxy servers. Using Opera Browser and Opera Mini in High data savings mode, we visit our domain and receive the page contents with a lock icon in the URL bar. Figure 6 shows how the same browser detects the bad Superfish-signed certificate when the data-saving mode is disabled yet accepts the certificate when it is enabled. We theorize that, in order to bootstrap the root CA store of their proxies, Opera extracted trusted CAs from real devices including the Lenovo laptops that were infected with Superfish [20]. Even though these root certificates are no longer trusted, this removal decision appears to not be reflected in Opera's root CA store.

E. Content Manipulation and Leakage

Proxy servers are at liberty to modify the actual HTML content of a web site, as well as any meta information in the HTTP responses used to transmit that content.

Modifications to the HTTP transport. Alarmingly, our experiments reveal that DSB proxy servers drop a number of headers in HTTP responses, which renders any security measures that rely on these headers moot. For example, the proxies for both Opera Mini in high data savings as well as Opera Browser do not forward the `X-Frame-Options` HTTP header. As this header is the predominant defense against clickjacking, dropping the header exposes web sites and their users to these attacks. We demonstrate this vulnerability in both browsers with demo videos [12] [14]. Similarly, Opera's proxy servers drop the `X-CSRF-Token` HTTP header, with the same effect of nullifying a class of CSRF defense mechanisms. Opera's proxies furthermore drop the `X-Content-Type-Options`, `X-WebKit-CSP`, and `X-Permitted-Cross-Domain-Policies` headers and custom headers unique to a particular website.

Unfortunately, our experiments also identified a series of modifications in HTTP requests. Particularly, UC Browser, UC Mini, Opera, and Opera Mini in high data savings mode collect a slew of potentially sensitive information (i.e., IMEI, phone number, device serial number, and the Android advertising ID) and transmit that information in HTTP headers to their proxy servers. We note that the Android advertising ID is not considered privacy sensitive per se and users are at liberty to reset this ID at any time. However, Opera additionally includes a so-called OperaID header that includes a persistent identifier in all its requests. This behavior neglects the Google Play Store guidelines, as it allows Opera to re-identify users even after they reset their advertising IDs.

Modifications to HTML content. Manipulating the HTML content would invariably erode users' trust. Hence, as expected, we did not observe any clearly malicious modifications to HTML content. Specifically, we observed modifications to a web site's content for Opera Mini in high data savings mode and Opera Browser, where the proxy server injects CSS content to block advertisements. To this end, the injected CSS styles hide DOM elements originating from well-known advertising exchanges. As hiding advertisements improves users' experience, we ascribe benign intent to these modifications.

Leakage. We performed the URL leakage experiment for 17 weeks, the page content leakage experiment for 8 weeks, and the credential leakage experiment for 5 weeks. During these experiments we did not record any instance of a third party attempting to revisit any page we visited through the browsers or use any of the information we leaked. While this result is unsurprising, we felt it important to include so as to provide a full picture of the DSB ecosystem.

V. Discussion and Future Work

In this section we discuss our main findings, their implications, as well as potential future research directions in the space of data-saving browsers.

Savings vs Security. With DSBs occupying a considerable portion of the market share of mobile browsers, it is critical to bring attention to the security implications of enabling data savings. Our study showed that the majority of data savings only occur if the user is browsing (Section IV-A); when consuming streaming content, most browsers cannot offer any savings and, in fact, seven out of the ten evaluated actually consume more data when streaming videos, rather than less.

At the same time, even though we found no evidence that data-savings browsers act maliciously (Section IV-E), these browsers expose their users to substantial security risks. We observed the usage of outdated software on proxying servers which could potentially lead to attackers achieving full MITM capabilities for these browsers (Section IV-B), as well as the use of weak cryptographic ciphers and issues with the verification process of TLS certificates (Section IV-D). Among others, we discovered that *all* users of the Opera Browser and Opera Mini in High-data-savings mode were vulnerable (prior to our disclosure) to SuperFish attacks where attackers could straightforwardly generate certificates with the leaked SuperFish private key, and successfully MITM all TLS-protected connections of these users. Orthogonally to these issues, we discovered that the "co-location" of traffic from benign and malicious users (i.e., the fact that everyone's traffic flows through the same DSB servers) means that benign users will be shown more CAPTCHAs when using data-savings browsers, compared to browsing the web through traditional means (Section IV-C).

Data-savings design. As HTTPS adoption is steadily increasing across the Web, with recent statistics reporting ∼69% in the top 150K websites [17], this complicates the offering of secure data savings. During our analysis we observed three unique ways of offering data savings. Some browsers (such as Yandex and UC Mini) completely "excuse" themselves from HTTPS connections thereby protecting their end-to-end nature but effectively offering near-zero savings in an HTTPS-by-default web. Other browsers (such as Chrome and Opera Browser) perform TLS termination in order to be able to offer data-savings over TLS-protected connections. This creates the potential for all the issues we discovered in this paper. Offering a data-savings mode through TLS interception means that the browser is, in effect, running *dual TLS stacks*. One at the device itself (when data-savings is turned off) and one at the proxy servers of the browser vendors. Keeping these stacks in sync is clearly complicated and can give rise to subtle bugs (such as forgetting to remove the SuperFish certificate from the proxy-side, root CA store) which can remain hidden for long periods of time, exposing users to MITM attacks. Finally, we observed a class of DSBs (such as Puffin) where the real browser is situated on remote servers and the browser running on the user's device is merely a "terminal" which renders the received server-side content and relays user actions (e.g., clicking a link) to the real server-side browser. This architecture bypasses the issues associated with dual TLS stacks. However, similarly to other TLS-intercepting browsers, user privacy is diminished as these browser vendors get access to users' full browsing history, cookies, and credentials.

We argue that there is a fourth design option available that can result in data savings with reduced privacy concerns. Namely, since multimedia content (such as images and videos) are major culprits of increased data usage, we argue that a mobile browser can selectively use TLS termination and content rewriting just for these types of content. Specifically, the mobile browser can fetch the main HTML content of a webpage without any TLS termination and, at the client side, decide which subsequent resources must be fetched over content-rewriting proxies. This effectively reduces the impact of a misconfigured/compromised content-rewriting server to multimedia content, instead of the entirety of content that is currently exposed. We leave the design and evaluation of such a data-saving browser to future work.

Understanding user perceptions. While the allure of saving data likely plays an important role in the decision-making process of users when choosing a browser, a study that explores how users perceive DSBs could provide important insights. The transparent process of data-saving and the technical complexities of TLS interception, obfuscate the privacy and security implications of sensitive communications being in a readable form on not-explicitly-trusted or potentially-vulnerable proxy servers. It is likely that if end users would understand the methods through which data-savings are offered, that they would stop utilizing these types of browsers.

Similarly, because of the arbitrary geographical location of the content-rewriting servers, plaintext/encrypted data may suddenly flow through unexpected and unfavorable regions with different privacy laws and stances towards user data. In February 2019, US Senators asked the DHS to investigate mobile browsers and VPNs that relay the traffic of government employees to countries of "national security concern" [53]. We, again, expect that if government employees are made aware of the exact methods through which data-savings are achieved, they may very well switch to different browsers.

Responsible disclosure. We have reached out to the affected browsers and responsibly disclosed to them our findings. To date, we have received responses from Opera regarding the SuperFish certificate validation errors and the dropping of security sensitive HTTP headers by proxy servers. Opera has responded to our report and immediately patched the vulnerabilities. We are currently in the process of following up with the remaining vendors to ensure they are aware of our findings and to understand how they intend to address them.

VI. Related Work

To the best of our knowledge, this paper presents the first security analysis of data-saving browsers, highlighting their differences compared to traditional browsers and how these differences can lead to weakened security and degraded user experience. Here, we briefly discuss prior work on TLS interception, mobile browsers, and rogue network relays.

TLS Interception. In 2017, Durumeric et al. proposed a technique for identifying the presence of HTTPS interception by inspecting the TLS handshake of browsers and identifying discrepancies between these handshakes and the declared user agent [32]. Using multiple vantage points (i.e., Cloudflare servers and e-commerce sites) the authors measured the phenomenon of HTTPS interception and traced it back to specific middleboxes and antivirus software. De Carnavalet and Mannan investigated the client-side TLS interception of antivirus and parental-control applications, finding vulnerabilities that included the acceptance of self-signed and revoked certificates,

enabling MITM attackers to hijack connections to this TLS-intercepting software that they could not have hijacked if that software was absent [30]. Waked et al. analyzed the TLS interception of six network appliances finding similar certificate-validation and poor key-storage issues [63].

Given all the issues with TLS interception, the US-Cert has published an alert (TA17-075A) titled "HTTPS Interception Weakens TLS Security" where they encourage organizations to verify that they need HTTPS-interception capabilities and "ensure their HTTPS inspection products are performing correct transport layer security (TLS) certificate validation" [61].

Inspired by the issues introduced by TLS-intercepting software, Lee et al. recently proposed an alternative, middlebox-aware TLS protocol which allows clients to authenticate middleboxes and servers to be aware of the presence of a middlebox between them and clients [41]. Note that, unlike middleboxes, antivirus software, and parental-control applications, data-savings browsers control both the client-side (i.e., the browser) as well as the content-rewriting servers (conceptually similar to a middlebox). Our work therefore confirms the difficulty of securely intercepting TLS connections even with the increased control afforded by data-saving browsers.

Mobile Browser Security. When handheld electronic devices started including browsers in their software, researchers realized that these browsers had to make certain design decisions that made them uniquely vulnerable to specific types of attacks. Niu et al. [50] and Amrutkar et al. [22] showed how the limited screen real-estate of handheld devices could be abused for highly-effective phishing attacks [36]. Luo et al. performed a longitudinal analysis of Android mobile browsers finding that, in terms of UI vulnerabilities, mobile browsers are becoming less secure over time [45] and are slower in supporting standard security mechanisms (such as CSP and HSTS) compared to desktop browsers [44]. Tendulkar et al. investigated how "cloud browsers" (such as Puffin) can be abused for free computation but did not investigate the security and privacy impact of using these browsers [59].

Orthogonal to the ability of mobile browsers to enforce security mechanisms, Mendoza et al. quantified the consistent use of security mechanism across desktop and mobile sites, finding cases where the mobile versions of websites did not employ the same mechanisms as the desktop versions and could therefore be abused to attack users [48]. Previously, Papadopoulos et al. [51] and Leung et al. [42] had compared the privacy loss when accessing a web service over a mobile browser or the corresponding mobile app. Kim et al. demonstrated the privacy risks that users face due to incorrect implementations of the Geolocation API in mobile browsers [39]. Recent studies also measured the extent to which web sites leverage APIs supported by modern browsers for accessing smartphone sensor data, which can be used for a plethora of different attacks [29], [46].

In this paper, we investigate a specific class of mobile browsers that attempt to offer data-savings to their users and present a comprehensive exploration of the security issues that are introduced when users activate this data-savings mode.

Rogue Network Relays. Even though the average user connects to web servers directly, there exist a number of scenarios where users willingly proxy their traffic through various servers, as way of evading censorship, preserving anonymity, and accessing geo-fenced services. Prior work has discovered that a fraction of these proxying systems modify the content that flows through them with security and privacy consequences. In 2014, Winter et al. proposed a system that used honeytokens to discover malicious Tor exit nodes which capitalized on unencrypted connections to collect credentials of Tor users [65]. Sivakorn et al. showed how malicious operators of Tor exit nodes could collect HTTP cookies from unencrypted communications and then later abuse them for session hijacking attacks [55]. Tsirantonakis et al. investigated the content-modification performed by open proxies discovering that 5.15% of them perform some type of malicious modification [60]. Chung et al. utilize an HTTP(S)-proxying service that uses real user devices to measure the level of content-integrity violations across the web finding that up to 4.8% of the proxying nodes in their experiment were subject to some form of content-integrity violation [27].

In contrast to past work, even though we utilized a large number of honeytoken-based experiments, we did not discover any signs of malicious content modification or data leakage. This is an intuitive finding since the browser vendors behind data-saving browsers are legitimate entities which are highly unlikely to voluntarily engage in this type of behavior.

VII. Conclusion

In a world where a large portion of web browsing is conducted on smartphones on-the-go, and data plans present a substantial cost for most users, data-saving browsers pose an alluring option. In our study we set forth to explore, and ultimately demonstrate, that enabling the data-saving mode significantly impacts a user's security posture. Our experimental analysis revealed a series of vulnerabilities that are introduced by data-saving modes in major browsers, and which constitute a significant privacy and security threat to hundreds of millions of users. Our findings highlight the immensity of the trade-off that users are faced with, as the obvious financial benefits of using DSBs are overshadowed by weakened TLS encryption, faulty certificate inspection, lack of support for security mechanisms, traffic flowing through proxy servers running outdated software, and users being labeled as potential bots. Our study sheds light on an important security threat that hundreds of millions of users are currently facing, and we hope that our findings help users make more informed decisions during their mobile browser selection process.

Acknowledgements: We thank our shepherd Emily Stark and the anonymous reviewers for their helpful feedback. This work was supported by the Office of Naval Research (ONR) under grants N00014-17-1-2541 and N00014-19-1-2364, as well as by the National Science Foundation (NSF) under grants CNS-1617593, CNS-1813974, and CNS-1934597.

REFERENCES

[1] Andhook. https://github.com/asLody/AndHook.

[2] Ars technica - dhs: Multiple us gov domains hit in serious dns hijacking wave. https://arstechnica.com/information-technology/2019/01/multiple-us-gov-domains-hit-in-serious-dns-hijacking-wave-dhs-warns/.

[3] Boringssl. https://github.com/google/boringssl.

[4] Chromium blog: Chrome lite pages - for a faster, leaner loading experience. https://blog.chromium.org/2019/03/chrome-lite-pages-for-faster-leaner.html.

[5] Cisco talos - dns hijacking abuses trust in core internet service. https://blog.talosintelligence.com/2019/04/seaturtle.html.

[6] Data savings and turbo mode. https://www.opera.com/turbo.

[7] Google recaptcha v3 - interpreting the score. https://developers.google.com/recaptcha/docs/v3.

[8] Mitmproxy. https://mitmproxy.org/.

[9] Nmap. https://nmap.org.

[10] Opera browser superfish attack demo. https://vimeo.com/376471169/3ce7f26104.

[11] Opera browser wrong host demo. https://vimeo.com/376667209/c38449bb65.

[12] Opera browser x-frame-options demo. https://vimeo.com/376524398/0d36db25cb.

[13] Opera mini wrong host certificate demo. https://vimeo.com/376667274/8dae75351a.

[14] Opera mini x-frame-options demo. https://vimeo.com/376524711/a0c9f78a93.

[15] Pew research center - use of smartphones and social media is common across most emerging economies. https://www.pewresearch.org/internet/2019/03/07/use-of-smartphones-and-social-media-is-common-across-most-emerging-economies/.

[16] Polar proxy. https://www.netresec.com/?page=PolarProxy.

[17] Qualys SSL Labs - SSL Pulse. https://www.ssllabs.com/ssl-pulse/.

[18] recaptcha guidelines. https://developers.google.com/recaptcha/docs/v3.

[19] Statista -share of global mobile website traffic 2015-2019. https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/.

[20] The Guardian - Lenovo accused of compromising user security by installing adware on new PCs. https://www.theguardian.com/technology/2015/feb/19/lenovo-accused-compromising-user-security-installing-adware-pcs-superfish, 2015.

[21] Devdatta Akhawe and Adrienne Porter Felt. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *Proceedings of the 22nd USENIX Security Symposium (USENIX Security)*, 2013.

[22] Chaitrali Amrutkar, Kapil Singh, Arunabh Verma, and Patrick Traynor. VulnerableMe: Measuring systemic weaknesses in mobile browser security. In *Proceedings of the International Conference on Information Systems Security (ICISSP)*, 2012.

[23] Badssl - a memorable site for HTTPS misconfiguration. https://badssl.com.

[24] Adam Barth and Mike West. Cookies: HTTP State Management Mechanism. Internet-draft, Internet Engineering Task Force, 2019. Work in Progress.

[25] Kevin Bock, Daven Patel, George Hughey, and Dave Levin. uncaptcha: a low-resource defeat of recaptcha's audio challenge. In *Proceedings of the 11th USENIX Workshop on Offensive Technologies (WOOT)*, 2017.

[26] Stefano Calzavara, Riccardo Focardi, Matus Nemec, Alvise Rabitti, and Marco Squarcina. Postcards from the post-http world: Amplification of https vulnerabilities in the web ecosystem. In *Proceedings of the 40th IEEE Symposium on Security and Privacy (IEEE S&P)*, 2019.

[27] Taejoong Chung, David Choffnes, and Alan Mislove. Tunneling for transparency: A large-scale analysis of end-to-end violations in the internet. In *Proceedings of the 2016 Internet Measurement Conference (ICM)*, 2016.

[28] Cam Cullen. Sandvine 2019 Mobile Internet Phenomena Report. https://www.sandvine.com/press-releases/sandvine-releases-2019-mobile-internet-phenomena-report, 2019.

[29] Anupam Das, Gunes Acar, Nikita Borisov, and Amogh Pradeep. The web's sixth sense: A study of scripts accessing smartphone sensors. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[30] Xavier de Carné de Carnavalet and Mohammad Mannan. Killed by proxy: Analyzing client-end tls interception software. In *Proceedings of the 23rd Network and Distributed System Security Symposium (NDSS)*, 2016.

[31] Zakir Durumeric, David Adrian, Ariana Mirian, Michael Bailey, and J Alex Halderman. A search engine backed by internet-wide scanning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[32] 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 *Proceedings of the 24th Network and Distributed System Security Symposium (NDSS)*, 2017.

[33] Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Zmap: Fast internet-wide scanning and its security applications. In *Proceedings of the 22nd USENIX Security Symposium (USENIX Security)*, 2013.

[34] Qilin Fan, Hao Yin, Geyong Min, Po Yang, Yan Luo, Yongqiang Lyu, Haojun Huang, and Libo Jiao. Video delivery networks: Challenges, solutions and future directions. *Computers & Electrical Engineering*, 66:332–341, 2018.

[35] Adrienne Porter Felt, Richard Barnes, April King, Chris Palmer, Chris Bentzel, and Parisa Tabriz. Measuring HTTPS adoption on the web. In *Proceedings of the 26th USENIX Security Symposium (USENIX Security)*, 2017.

[36] Adrienne Porter Felt and David Wagner. *Phishing on mobile devices*. na, 2011.

[37] Google fi - plans benefits & details. https://fi.google.com/about/plans/.

[38] Muhammad Ikram, Narseo Vallina-Rodriguez, Suranga Seneviratne, Mohamed Ali Kaafar, and Vern Paxson. An analysis of the privacy and security risks of android vpn permission-enabled apps. In *Proceedings of the Internet Measurement Conference (IMC)*, 2016.

[39] Hyungsub Kim, Sangho Lee, and Jong Kim. Exploring and mitigating privacy threats of html5 geolocation api. In *Proceedings of the 30th Annual Computer Security Applications Conference (ACSAC)*, 2014.

[40] Klein, Amit and Shulman, Haya and Waidner, Michael. Internet-wide study of dns cache injections. In *Proceedings of the IEEE INFOCOM Conference on Computer Communications*, 2017.

[41] Hyunwoo Lee, Zach Smith, Junghwan Lim, Gyeongjae Choi, Selin Chun, Taejoong Chung, and Ted Taekyoung Kwon. matls: How to make tls middlebox-aware? In *Proceedings of the 26th Network and Distributed System Security Symposium (NDSS)*, 2019.

[42] Christophe Leung, Jingjing Ren, David Choffnes, and Christo Wilson. Should you use the app for that?: Comparing the privacy implications of app-and web-based online services. In *Proceedings of the Internet Measurement Conference (IMC)*, 2016.

[43] Christos Liaskos, Vasileios Kotronis, and Xenofontas Dimitropoulos. A novel framework for modeling and mitigating distributed link flooding attacks. In *Proceedings of the 35th Annual IEEE International Conference on Computer Communications (INFOCOM)*, 2016.

[44] Meng Luo, Pierre Laperdrix, Nima Honarmand, and Nick Nikiforakis. Time does not heal all wounds:a longitudinal analysis of security-mechanism support in mobile browsers. In *Proceedings of the 26th Network and Distributed System Security Symposium (NDSS)*, 2019.

[45] Meng Luo, Oleksii Starov, Nima Honarmand, and Nick Nikiforakis. Hindsight: Understanding the Evolution of UI Vulnerabilities in Mobile Browsers. In *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*, 2017.

[46] Francesco Marcantoni, Michalis Diamantaris, Sotiris Ioannidis, and Jason Polakis. A large-scale study on the risks of the html5 webapi for mobile sensor-based attacks. In *Proceedings of the Web Conference (WWW)*, 2019.

[47] Arunesh Mathur, Brent Schlotfeldt, and Marshini Chetty. A mixed-methods study of mobile users' data usage practices in south africa. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 2015.

[48] Abner Mendoza, Phakpoom Chinprutthiwong, and Guofei Gu. Uncovering HTTP Header Inconsistencies and the Impact on Desktop/Mobile Websites. In *Proceedings of the Web Conference (WWW)*, 2018.

[49] Bodo Möller, Thai Duong, and Krzysztof Kotowicz. This poodle bites: exploiting the ssl 3.0 fallback. *Security Advisory*, 2014.

[50] Yuan Niu, Francis Hsu, and Hao Chen. iPhish: Phishing Vulnerabilities on Consumer Electronics. In *Usability, Psychology, and Security (UPSEC)*, 2008.

[51] Elias P Papadopoulos, Michalis Diamantaris, Panagiotis Papadopoulos, Thanasis Petsas, Sotiris Ioannidis, and Evangelos P Markatos. The long-standing privacy debate: Mobile websites vs mobile apps. In *Proceedings of the Web Conference (WWW)*, 2017.

[52] Requests: HTTP for humans. https://2.python-requests.org/en/master/.

[53] Marco Rubio and Ron Wyden. Rubio, Wyden Ask Homeland Security To Investigate National Security Risks Of Foreign VPN Apps. https://www.rubio.senate.gov/public/index.cfm/2019/2/rubio-wyden-ask-homeland-security-to-investigate-national-security-risks-of-foreign-vpn-apps, 2019.

[54] SimilarTech. Captcha Technologies Market Share and Web Usage Statistics. https://www.similartech.com/categories/captcha.

[55] Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis. The cracked cookie jar: Http cookie hijacking and the exposure of private information. In *Proceedings of the 37th IEEE Symposium on Security and Privacy (IEEE S&P)*, 2016.

[56] Suphannee Sivakorn, Iasonas Polakis, and Angelos D Keromytis. I am robot:(deep) learning to break semantic image captchas. In *Proceedings of the 1st IEEE European Symposium on Security and Privacy (IEEE EuroS&P)*, 2016.

[57] Microsoft reads your skype chat messages. https://yro.slashdot.org/story/13/05/14/1516247/microsoft-reads-your-skype-chat-messages.

[58] StatCounter. Browser market share worldwide. http://gs.statcounter.com/browser-market-share, 2019.

[59] Vasant Tendulkar, Ryan Snyder, Joe Pletcher, Kevin Butler, Ashwin Shashidharan, and William Enck. Abusing cloud-based browsers for fun and profit. In *Proceedings of the 28th Annual Computer Security Applications Conference (ACSAC)*, 2012.

[60] Giorgos Tsirantonakis, Panagiotis Ilia, Sotiris Ioannidis, Elias Athanasopoulos, and Michalis Polychronakis. A large-scale analysis of content modification by open http proxies. In *Proceedings of the 25th Network and Distributed System Security Symposium (NDSS)*, 2018.

[61] US CERT. Alert (TA17-075A): HTTPS Interception Weakens TLS Security. https://www.us-cert.gov/ncas/alerts/TA17-075A.

[62] Luis Von Ahn, Manuel Blum, Nicholas J Hopper, and John Langford. Captcha: Using hard ai problems for security. In *International Conference on the Theory and Applications of Cryptographic Techniques*, 2003.

[63] Louis Waked, Mohammad Mannan, and Amr Youssef. To intercept or not to intercept: Analyzing tls interception in network appliances. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security (ASIACCS)*, 2018.

[64] Kathleen Wilson. Distrust of symantec tls certificates. https://blog.mozilla.org/security/2018/03/12/distrust-symantec-tls-certificates/, March 2018.

[65] Philipp Winter, Richard Köwer, Martin Mulazzani, Markus Huber, Sebastian Schrittwieser, Stefan Lindskog, and Edgar Weippl. Spoiled onions: Exposing malicious tor exit relays. In *International Symposium on Privacy Enhancing Technologies Symposium (PETS)*, 2014.

[66] Guixin Ye, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, and Zheng Wang. Yet another text captcha solver: A generative adversarial network based approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.

[67] Huan Zhou, Hui Wang, Xiuhua Li, and Victor CM Leung. A survey on mobile data offloading technologies. *IEEE Access*, 6:5101–5111, 2018.