Turning Attacks into Advantages: Evading HTTP Censorship with HTTP Request Smuggling

Philipp Müller
Paderborn University
mail@pmueller.dev

Niklas Niere
Paderborn University
niklas.niere@upb.de

Felix Lange
Paderborn University
felix.lange@upb.de

Juraj Somorovsky
Paderborn University
juraj.somorovsky@upb.de

ABSTRACT

Many countries limit their residents' access to various websites. As a substantial number of these websites do not support TLS encryption, censorship of unencrypted HTTP requests remains prevalent. Accordingly, circumvention techniques can and have been found for the HTTP protocol. In this paper, we infer novel circumvention techniques on the HTTP layer from a web security vulnerability by utilizing HTTP request smuggling (HRS). To demonstrate the viability of our techniques, we collected various test vectors from previous work about HRS and evaluated them on popular web servers and censors in China, Russia, and Iran. Our findings show that HRS can be successfully employed as a censorship circumvention technique against multiple censors and web servers. We also discover a standard-compliant circumvention technique in Russia, unusually inconsistent censorship in China, and an implementation bug in Iran. The results of this work imply that censorship circumvention techniques can successfully be constructed from existing vulnerabilities. We conjecture that this implication provides insights to the censorship circumvention community beyond the viability of specific techniques presented in this work.

KEYWORDS

http, censorship, request smuggling, censorship circumvention

1 INTRODUCTION

Governments around the world block access to certain websites for their residents [44]. While website encryption with TLS [18, 54] provides some protection against censorship efforts, the SNI extension [20] still leaks the hostname. This allows potential censors to analyze the SNI extension and block the traffic before the TLS-encrypted channel to the website is built [55]. While using TLS is nowadays state-of-the-art from a security perspective, many websites still offer plain HTTP, often in conjunction with TLS [32]. Even when websites support only TLS, they often redirect clients' HTTP traffic to HTTPS websites. This culminates in a considerable number of plain HTTP requests. Cloudflare Radar reports that 16 % of requests originating in China are transmitted over plain HTTP [16]. These HTTP requests are analyzed by censors worldwide [9, 28, 36, 44]. As long as clients send HTTP requests that are intercepted by censors, HTTP censorship remains an important stepping stone toward circumventing censors.

Censors mainly filter HTTP requests by analyzing the domain contained in their `Host` header [28, 44]. Using Deep Packet Inspection (DPI), censors isolate the HTTP layer of received network packets, identify potential `Host` headers, and extract the domain name. Circumventions of this process commonly follow one of two avenues. First, obfuscating the domain or the `Host` header prevents the censor from extracting a domain name [28]. Second, invalidating the TCP state prevents the censor from recognizing or filtering any HTTP packets [10]. Both techniques modify an already present HTTP request and attempt to pass it through the censor.

In this work, we create novel circumvention techniques against `Host` header-based censorship from a web security vulnerability. Specifically, we utilize the knowledge of HTTP request smuggling (HRS) to create ambiguously defined HTTP requests that circumvent censors. Such an ambiguously defined HTTP request is depicted in Figure 1. The request contains a second—smuggled—HTTP request and two differentiating length fields that encompass either the first or both requests, respectively. A smuggled request that is normally censored and missed by the censor but interpreted by the web server constitutes a successful circumvention technique.

```
GET / HTTP/1.1
Host: example.com
Content-Length: 50
Transfer-Encoding: chunked

0

----------------------------------

GET / HTTP/1.1
Host: censored.org
```

Figure 1: Example of a smuggled request. The Content-Length header indicates a single request; the Transfer-Encoding header indicates two requests.

Contributions. In our paper, we construct novel censorship circumvention techniques from HRS vectors based on current work in the field. To determine the viability of our test vectors, our analysis was twofold. First, we evaluated the acceptance rate of our test vectors on popular local web servers, CDNs, and randomly chosen

Free and Open Communications on the Internet 2024 (2), 42–53

domains from the CitizenLab test-lists repository [14]. Second, we analyzed censors' behavior on our test vectors using vantage points in China, Russia, and Iran. Our evaluation of local web servers and CDNs revealed numerous request smuggling opportunities in web servers and CDNs. Our analysis of censors yielded numerous successful circumvention strategies. We combined the results of both analyses and extracted strategies that circumvent censors and are accepted by widely used web servers. Through further analysis, we detected a general standard-compliant circumvention technique for the censor in Russia, inconsistencies in the censorship behavior of China's Great Firewall, and an implementation bug in Iran's censor. Most importantly, we show that a vulnerability can be directly transformed into censorship circumvention techniques.

2 BACKGROUND AND RELATED WORK

In this section, we introduce HTTP request smuggling, outline censorship practices of governments worldwide, and discuss current countermeasures and measurement platforms.

2.1 HTTP Request Smuggling (HRS)

HRS is a web security vulnerability that utilizes discrepancies in the HTTP parsers of at least two systems in a frontend-backend configuration. Specifically, HRS is possible if the frontend and backend servers consider different bounds for the same HTTP message. In this case, an attacker can hide—smuggle—bytes in the bounds of the first HTTP message which are interpreted by one system and missed by the other. To achieve this behavior, the attacker ambiguously defines the bounds of the HTTP message through HTTP headers: usually, the *Transfer-Encoding* (TE) header and the *Content-Length* (CL) header [23].

HTTP Headers. The Content-Length header explicitly defines the number of bytes n contained in the body of the HTTP request through `Content-Length: n`. The Transfer-Encoding header specifies the encoding, which is applied to the message body. For HRS, the chunked encoding (`Transfer-Encoding: chunked`) is relevant. It indicates that the body is streamed in a series of chunks. Each chunk begins with the hexadecimal length of itself, followed by the chunk data. A chunk of length zero terminates the body. Figure 2 depicts the correct usage of both headers for the same HTTP message. Notably, while these headers are used for transferring data in the HTTP body (e.g., in HTTP POST requests), their usage in HTTP GET requests is allowed.

```
POST /path HTTP/1.1
Host: example.com
Content-Length: 15

example-content
```

```
POST /path HTTP/1.1
Host: example.com
Transfer-Encoding: chunked

9
example-c
6
ontent
0
```

Figure 2: POST request with correct usage of `Content-Length` and `Transfer-Encoding` headers.

HTTP Request Smuggling Research. HRS was first presented by Linhart et al. [43] in 2005. Linhart et al. successfully attacked web servers and middleboxes threefold: with two Content-Length (CL) headers, two Transfer-Encoding (TE) headers, and a combination of both. In 2019, Kettle [38] discovered that many servers were resilient against same-header smuggling attacks but introduced new HRS attacks that contain a combination of the CL and TE header. In the following years, new attack vectors have been discovered manually by Klein [40] and through fuzzing by Jabiyev et al. [34]. defparam [17] implemented a tool that gathers known test vectors and automatically detects HRS vulnerabilities in web server implementations. In our work, we gather test vectors from previous work [17, 34, 38, 40] and utilize them to construct novel censorship circumvention techniques.

Kettle [39] and Emil Lerner [21] discovered further HRS attacks against HTTP/2. We did not consider their vectors as browsers only support HTTP/2 in conjunction with TLS [27] and changes to the HTTP headers in this scenario are invisible to the censor.

Attack Example. To execute an HRS attack, an attacker includes both headers in a single HTTP request such that they indicate different message bounds. Figure 1 displays an example of an HRS attack attempt in which the second HTTP request, depicted after the dotted line, is smuggled alongside the first HTTP request. The CL header field of the first request in line 3 defines a body size of 50 bytes: the body contains the highlighted bytes and the second request is not parsed as a second HTTP request. The TE header field in line 4 defines a chunked encoding: the body contains only an empty chunk and the second HTTP request is parsed as such. Therefore, the amount of HTTP requests this message contains depends on which header is accepted by a parser. The standard forbids the presence of both headers and specifies that the TE header takes precedence [23, 46]. The works we discussed above still found HRS vulnerabilities, especially by invalidating either header. In this paper, we draw on test vectors that invalidate either header to circumvent HTTP censorship.

2.2 Censorship

Network censorship is widely used by many countries and has been revealed by numerous analyses [3–6, 26, 45, 50, 52, 60, 64, 67]. Some countries maintain their censorship infrastructure themselves [62] while others delegate this task to local Internet Service Providers (ISPs) [63]. In both cases, middleboxes either block IPs directly or use DPI to scan packets for domain names or other keywords they want to censor. To interrupt a connection, middleboxes drop packets [61, 63] or inject additional malicious packets such as wrong DNS responses [4, 31, 51], or TCP RST packets [15, 42] into the connection. Some middleboxes also employ residual censorship: they block innocuous packets for some time after a previous packet triggered censorship behavior [8]. Various platforms such as OONI [25], CensoredPlanet [56], and GFWatch [30] collect censorship measurements around the world and make them freely accessible. Similarly, various censorship circumvention tools implement a wide range of circumvention techniques [13, 29, 37, 41, 48, 58, 66]. We refer to the comprehensive work of Master and Garman [44] for an overview of countries and techniques involved in global censorship efforts.

Censors Analyzed in this Work. In our work, we analyzed censors in China, Russia, and Iran. China's censor — the Great Firewall of China (GFW) — is arguably the most researched censor [4, 10–12, 15, 22, 28, 31, 44, 61, 62]. The GFW maintains an infrastructure of state-owned middleboxes [62] which analyze the content of HTTP packets [28] and forcibly interrupt connections by injecting three TCP RST packets [10, 12]—immediately and residually [8, 11]. Until 2020, Russia maintained a centralized blocklist but delegated censorship implementation to its ISPs [53]. In 2021, Russia further centralized its censorship by mandating ISPs to build dedicated censorship devices into their networks [63, 64]. These so-called TSPU devices intercept connections by setting the RST flag in packets or dropping them altogether [63]. As of 2021, TSPU devices did not trigger on HTTP packets [65]. HTTP censorship in Iran was first detected by Aryan et al. [5] in 2013. They found that Iran's censor injects an HTTP block page to tear down connections. In 2020, Bock et al. [9] detected an additional Iranian censor which restricts ports 53 (DNS), 80 (HTTP), and 443 (HTTPS) to these protocols and also uses residual censorship [8]. This makes Iran's censor comparably strict in that they only allow traffic on these well-known ports which their censorship system can analyze. We analyzed the censors of China, Russia, and Iran in this work as they censor HTTP, are well-researched, and undergo frequent changes.

Censorship Circumvention. The censors we analyzed and other censors focus on the `Host` header and request path fields of HTTP requests [5, 10, 36, 67]. Accordingly, the obfuscation of these fields to a censor is paramount for a successful circumvention of HTTP censorship and has been successfully achieved by previous work. Bock et al. [10] found numerous HTTP censorship circumventions by manipulating packet fields on the TCP layer; this invalidates the reduced TCP state of the censor such that it lets the packets pass. Manipulations on the TCP layer have the benefit of applying to other application layer protocols as well. On the other hand, they usually require elevated privileges which might not be easily achievable on some devices and it has been shown that the GFW fixed at least one circumvention over time [47]. Jermyn and Weaver [35] and Harrity et al. [28] present many circumvention techniques that confuse censors by modifying the HTTP headers of censored requests. While we considered similar modifications in our evaluations, we did not apply them to censored resources directly. Instead, we modify a crafted request containing the censored request in its body.

In their paper, Harrity et al. [28] relate their methodology to HRS by drawing similarities between their fuzzing approaches. However, they claim that the objective of censorship circumvention—modifying a request such that it passes the censor—differs from the objective of HRS: hiding a second request alongside the first. We argue that censorship circumvention and HRS can follow the same objective by hiding a censored request in a second uncensored request and smuggling it past the censor.

3 TURNING HRS ATTACKS INTO CENSORSHIP CIRCUMVENTIONS

Our major observation is that HRS attacks are conceptually similar to censorship circumvention attacks on censors between a client and a server. Consider the smuggling vector in Figure 1. A censor that interprets the CL header parses the second request as the body of the first request. Subsequently, it would only analyze the harmless `Host` header of the first message and not censor the message. Should the web server interpret the TE header of the first request, it would interpret both requests separately and return the censored resource alongside the uncensored one. If the web server does not host the uncensored resource, it might also return an error alongside the censored request. Figure 3 depicts a successful censorship circumvention using HRS. Note, that a stateful censor could match the number of HTTP requests and HTTP responses and drop additional HTTP responses. We did not identify this behavior from censors in this work and this case could be accommodated by sending an additional dummy request. In summary, we postulate that HRS can be used to circumvent censorship as long as parsing ambiguities between censors and web servers exist. In the following, we present our analyses of various HRS vectors and show that censorship circumvention using HRS is possible.

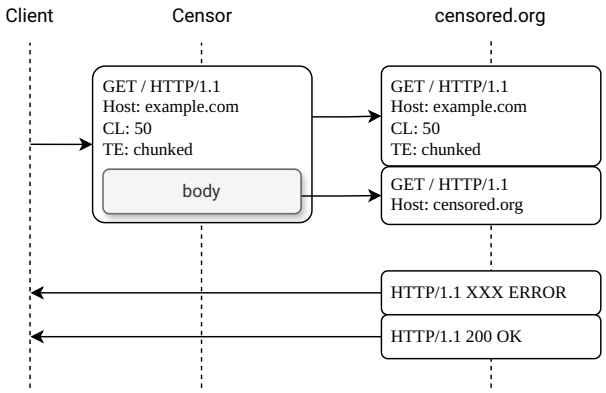

Figure 3: HRS attack on a frontend/censor. The `Content-Length` and `Transfer-Encoding` headers are shortened for readability.

4 METHODOLOGY

In this section, we define the structure of our test vectors and describe our evaluation process. As the censors that we analyze employ different censorship mechanisms, we accommodated our scans accordingly.

4.1 Test Vectors

To circumvent censorship, we attempt to smuggle a censored HTTP request alongside an uncensored one through the censor. Accordingly, each of our test vectors consists of two consequent HTTP requests. The first request always has exactly one CL and one TE header in this order. One header sets the bounds of the first request behind the second request, attempting to confuse the censor; the other header sets the bounds of the first request before the second request, attempting to be accepted by the web server. A test vector is then a modification of either the TE or CL header by strategies such as injecting a new line (\n) in the corresponding header. We split the test vectors into four types: CL*/TE, TE*/CL, CL/TE*, and

TE/CL*. The first value indicates the header that sets the message bounds after the second request, effectively hiding the second request. The second value indicates the header that sets the message bounds after the first request, revealing the second request. Our test vectors modify one of these headers which is indicated by the asterisk (*).

Figure 1 depicts a ground truth test vector without modifications in which the CL header sets the bounds after the second request, confusing the censor. The type of this test vector is CL/TE. Modifying the CL header would result in type CL*/TE while modifying the TE header would result in type CL/TE*. For each modification, we created up to four vectors—one for each type. These test vectors differ in which header the modification is applied to and which bounds each header indicates. Notably, not all modifications can be applied to both headers such as insertions that specifically target the `chunked` keyword of the TE header. Overall, we generated 4,488 unique test vectors from 1,138 modifications. All our test vectors are taken from previous HRS work (cf. Section 2). This way we directly transform a web security vulnerability into a censorship circumvention technique.

4.2 Web Server Acceptance

Successful test vectors bypass the censor and elicit a correct response from the targeted web server (cf. Figure 3). Specifically, the web server has to send two responses of which the last one must indicate a success. To test web servers' acceptance of our test vectors we analyzed which test vectors elicit two responses from common web servers and recorded the HTTP status codes. We consider a test vector to be accepted by a web server if the web server sends two HTTP responses and if the status code of the second response is either 200 Success or 3XX Redirect. Overall, we evaluated 50 different web servers: local web servers, CDNs, and hosts taken from the CitizenLab test lists for China, Iran, Russia, and the global list [14]. This diverse set of web servers aims to find a balance between popular technologies and web servers that are directly impacted by censorship. We performed our evaluation of live web servers on 5th November, 2023. Below, we outline how we decided on web servers for each category.

Local Web Servers. To scan local web servers, we decided to use the most popular web server versions according to W3Tech[59]. As of 26th August, 2023, W3Tech identified Apache and Nginx to have the highest market share followed by Cloudflare Server. As local web servers, we decided to test the four versions with the highest market share of Apache and Nginx as well as the latest version at the point of our evaluations. These were Apache versions 2.2.15, 2.4.6, 2.4.29, 2.4.41, and 2.4.57-latest and Nginx versions 1.14, 1.18, 1.21, 1.22, and 1.25.2-latest respectively.

CDNs. As popular CDNs to scan, we selected Akamai, Amazon, Cloudflare, and Fastly. For each of these CDNs, we selected five popular websites with the help of Hunter Web Services [33]. We verified the hosts with a Who-is-Lookup of their IP address and checked that the Autonomous System Number (ASN) belongs to that CDN before adding them to our targets. We refer to Appendix A for the selection of websites for each CDN.

CitizenLab List Domains. Additionally, we also evaluated our test vectors on websites that are directly impacted by censorship. For this, we randomly selected five hosts from the CitizenLab test lists for China, Iran, Russia, and the global test list, respectively [14]. We list all domains in Appendix A.

4.3 Censor Scans

After evaluating our test vectors on web servers, we determined which test vectors successfully circumvent censorship. For this, we considered every vector that was accepted by at least one web server as described in Section 4.2. This reduces the amount of test vectors we have to evaluate against censors while ensuring that any circumvention technique we find is also accepted by web servers. Additionally, we considered a ground truth vector that consists of a single plain HTTP GET request. We evaluated each accepted test vector against censors in China (Zhengzhou), Russia (Moscow), and Iran (Mashhad). Specifically, we sent our test vectors from a vantage point inside the country to a control server located in Germany and recorded the ensuing network behavior. We sent our messages to a control server rather than an existing web server to gain more control over connection errors, lower the chance of additional IP censorship, and prevent unnecessary load on real-life applications. We also used different authentic `User-Agent` headers during our evaluations to lower the chance of our request being fingerprinted. We provide further details such as the vendor and autonomous system number (ASN) of our vantage points in Appendix B.

For each country, we selected two censored and two uncensored websites. The censored websites were taken from the respective CitizenLab test list [14] while the uncensored websites are either government websites or popular websites such as qq.com in China. We manually verified that HTTP requests including these domains were censored. While some of these domains do not support HTTP—or redirect to HTTPS—we could utilize them to trigger and evaluate censorship in the respective countries. We refer to Section 4.2 for our evaluation of web server support for our test vectors. In our censor evaluation, we applied each of the four domains to each test vector: the `Host` header of the second—smuggled—request in the test vector was set to the censored domain while the `Host` header of the first request was always set to an uncensored domain. Our ground truth vector, which consists of only one HTTP GET request, contained only the censored domain. We randomized the order of our test vectors and sent them 10 times each to our control server which answers every request with static bytes. We saved the possible server responses next to network information such as possible TCP RSTs, timeouts, and block pages. We manually triggered and identified block pages, categorized the server responses, and automatically detected them in our evaluations based on the observed status codes: 403 or 307. We conducted all of the censor scans in March and April 2024. In the following paragraphs, we outline the censorship mechanism of each country and our corresponding evaluation methodology.

China. For China, we selected freetibet.org/ and www.uyghurnet.org/ as censored websites. In a preliminary scan, we discovered that censorship in China happens consistently through injected TCP RST packets after receiving the initial HTTP request. This aligns with previous research [28]. Therefore, we considered a test

vector blocked in China if it triggers a TCP RST. Next to direct censorship through TCP RST injection, we also encountered residual censorship of 60–90 seconds on the triple (source IP, destination IP, destination port). This also aligns with previous research [8]. To avoid classifying uncensored test vectors as censored due to residual censorship, we connected to 1000 different destination ports on our control server and waited at least 120 seconds before reusing a destination port. Specifically, we selected ports 10,000–10,999 for which we ascertained the described HTTP censorship behavior in a previous scan.

Iran. For Iran, we selected gaytoday.com and twitter.com as censored websites. In a preliminary scan, we discovered that censorship in Iran happens very consistently through injected HTTP block pages. Therefore, we considered a test vector blocked in Iran if it triggers an HTTP block page. Since we observed no residual censorship in Iran, we could perform the whole scan on destination port 80, mimicking real HTTP traffic. We want to point out that we could not reproduce the residual censorship on port 80 encountered in previous work [8].

Russia. For Russia, we selected eurasia.amnesty.org and www.mdif.org as censored websites. In a preliminary scan, we discovered inconsistent censorship behavior in Russia. When sending censored requests to our control server, the rate of requests that would trigger a block page varied considerably. We also noticed that the same requests did not trigger a block page but were dropped when we sent them to another server. We attribute this inconsistent censorship behavior to upstream ISPs which are triggered depending on the network path of our packets. This aligns with the results of Bhaskar and Pearce [7]. We also infer that we did not encounter censorship by the newly developed TSPU devices [63]; these devices may still ignore HTTP traffic altogether, as in 2021 [65]. In the end, we considered a test vector to be blocked when it triggered an HTTP block page. Since we observed no residual censorship in Russia, we performed our scan on destination port 80.

5 RESULTS

In this section, we present and interpret the results of our evaluations. We also analyze web servers' behavior regarding our test vectors and show that HRS can be used to circumvent censorship.

5.1 Web Server Evaluation

Of our 4,488 test vectors, 2,015 were accepted by at least one web server. As described in Section 4, we consider a test vector accepted if it elicits two responses from the web server, and if the last response is either a 200 Success or a 3XX Redirect. Table 1 depicts the acceptance rate of web servers for each of our test vector types (cf. Section 4.1). Test vectors of the CL*/TE type are most successful followed by test vectors of the TE*/CL type. The high acceptance rates of the CL*/TE and TE*/CL types in comparison to the CL/TE* and TE/CL* types indicate that web servers largely prefer to interpret the unaltered headers. We did not detect any web server accepting and interpreting an altered CL header. We attribute this to the corresponding RFCs [23, 24, 46] which assign precedence to the TE header. Overall, web servers accepted 44.9% of our test vectors with a profound preference for test vectors for which they could accept the unaltered header. For these vectors to circumvent censorship, the censor has to interpret the altered header. Below, we provide a more detailed overview of the acceptance rates of our test vectors by specific web servers.

Table 1: Number of test vectors which we successfully evaluated on web servers ordered by test vector type.

Vector Type	Evaluated	Accepted
CL*/TE	1,114	1,103 (99.0%)
TE*/CL	1,130	859 (76.0%)
CL/TE*	1,130	53 (4.7%)
TE/CL*	1,114	0 (0.0%)
Total	4,488	2,015 (44.9%)

Local Servers. We locally evaluated our test vectors on five versions of Apache and Nginx; the results are depicted in Table 2 and Table 3, respectively. Both vendors show a decline in accepted test vectors for newer versions. Specifically, no TE/CL* or CL/TE* vectors are accepted by the latest Apache and Nginx versions we evaluated. We attribute this to HRS countermeasures implemented in Apache 2.4.25 [1], Apache 2.4.52 [2], and Nginx 1.21.1 [19]. The few CL*/TE and TE*/CL headers Apache 2.4.57 accepts invalidate the corresponding header heavily through symbol injections such as `\ffContent-Length`. In comparison, Nginx accepted a considerable amount of our test vectors. The latest version we evaluated, Nginx 1.25.2, accepted 1,315 vectors in total. Overall, newer Apache versions are much stricter in their acceptance of HRS vectors than newer Nginx versions. We disclosed our findings to Apache and Nginx. Neither intends to change their implementation's behavior in future versions.

Table 2: Successful test vectors on Apache versions ordered by test vector type.

Vector Type	Accepted by Apache Version				
	2.2.15	2.4.6	2.4.29	2.4.41	2.4.57
CL*/TE	1,101	1,101	156	156	5
TE*/CL	841	841	6	6	6
CL/TE*	41	41	20	20	0
TE/CL*	0	0	0	0	0
Total	1,983	1,983	182	182	11

CDNs and CitizenLab list. During our evaluation of CDNs and domains from the CitizenLab list, we encountered various web server implementations. Many web server implementations were closed-source implementations by the respective CDN while some were an Apache or Nginx version, an Nginx derivative [49, 57], or other closed-source implementations. Table 4 lists all web servers that we could identify with a `Server` header and the number of test vectors they accepted. CDNs largely dismissed our test vectors, with Cloudflare and Amazon's Elastic Load Balancing (awselb) being exceptions. Interestingly, Amazon's awselb accepted some of our test vectors while Amazon's other web server Cloudfront accepted

Table 3: Successful test vectors on Nginx ordered by test vector type.

Vector Type	Accepted by Nginx Version				
	1.14	1.18	1.21	1.22	1.25.2
CL*/TE	859	859	657	657	657
TE*/CL	837	836	658	658	658
CL/TE*	29	28	0	0	0
TE/CL*	0	0	0	0	0
Total	1,725	1,723	1,315	1,315	1,315

none. Most of the non-CDN web servers we discovered accepted an identical or similar set of test vectors as the Nginx versions we evaluated. We suspect that these libraries are built from Nginx; OpenResty and Taobao also state so on their websites [49, 57]. Through similar sets of accepted test vectors, we could also identify web servers with outdated Nginx or Nginx-derivative versions and notified their owners. In summary, the web server implementations of CDNs were stricter than the web server implementations by non-CDNs; mainly because many web servers are using either Nginx or one of its derivatives.

Notably, many websites we evaluated redirect HTTP traffic to an HTTPS port. Acceptance of our test vectors by these web servers still indicates a general acceptance of our test vectors by HTTP implementations used in the wild.

Table 4: Test vectors on CDNs and web servers that resulted in two responses during CitizenLab list evaluation.

Vendor	Web Server	Two Responses
Cloudflare	Cloudflare[a]	1,653–1,677[b]
Amazon	awselb/2.0[a]	470
Amazon	Cloudfront	0
Akamai	GHost	0
Fastly	Varnish	0
Taobao	Tengine	1,725
Open Source	Caddy	1,316
Open Source	Prometheus	1,315
OpenResty	openresty	1,315

[a] Second response is a redirect.
[b] Cloudflare servers accepted between 1,653 and 1,677 test vectors across 5 domains. We attribute this to different deployments or connection issues.

5.2 Censor Evaluation

We evaluated all test vectors that were accepted by at least one web server on censors in China, Iran, and Russia. Of these 2,015 test vectors, 19 circumvented the censor in China and 254 circumvented the censor in Iran. Interestingly, all of our test vectors successfully evaded censorship on our Russian vantage point. We discuss the censorship behavior on our vantage points in Russia and China below. The censorship behavior of Iran is discussed together with the strategies we discovered in Section 5.3.

Table 5 depicts a selection of test vectors that were accepted by at least one web server and circumvented at least one censor. We group these test vectors by their type (cf. Section 4.1) and categories which we describe in Section 5.3. Overall, test vectors of the TE*/CL and CL*/TE types are most successful as web servers accept 0 and 53 test vectors from the TE/CL* and CL/TE* types, respectively. None of the test vectors that were accepted by the latest Nginx and Apache versions circumvented censorship in China or Iran; we discuss this limitation in Section 6. Our test vectors were still evaluated positively on other widely used versions of Apache and Nginx, CDNs, and domains from the CitizenLab list. We consider this positive for the viability of HRS for censorship circumvention and stress the importance of testing live servers in conjunction with local web servers for censorship research.

Russia. As mentioned above, all of our test vectors successfully circumvented the censorship at our vantage point in Russia. We confirmed this behavior through the manual execution of our test vectors. After confirmation of our results, we analyzed the censorship behavior from our vantage point in Russia further. We found that the censor always analyzed only the first HTTP packet of the first TCP segment in a TCP stream. The censor never blocked a second HTTP request either as part of the same TCP packet or in a different TCP packet. Thus, all our test vectors circumvented the censorship at our vantage point in Russia as the censor never analyzed the second—smuggled—HTTP request. We suspect that the censor we encountered is stateful and assumes that the `Host` header does not change during a TCP connection. Thus, the censor is circumventable with a standard-compliant circumvention technique. Additionally, coalescing multiple application layer packets into a single TCP segment is an interesting technique that opposes the well-known fragmentation of a single application layer packet over multiple TCP segments. We see this as an additional hint towards advanced fragmentation techniques that coalesce and fragment network packets over multiple network layers as is possible for TLS or QUIC.

China. In China, most of our test vectors either circumvented the GFW without triggering TCP RST injection or were censored with TCP RSTs such that the answer did not reach our vantage point in any case. For 13 of our test vectors, the behavior was more mixed: we encountered TCP RSTs on some executions and the correct server answer on others. We executed an additional scan with all test vectors and evaluated the behavior of the GFW with 100 executions on each test vector and both censored domains. This scan reproduced the behavior we saw in our first scan on the same test vectors: a subset of our test vectors circumvents the GFW with a rate between 10% and 35%. We suspect that we encountered two parts of the GFW's infrastructure which employ different mechanisms for its HTTP censorship. This behavior is peculiar for the GFW which is usually considered consistent in its censorship behavior. We suggest a future analysis of the consistency of the GFW's censorship.

5.3 Strategies

Below, we describe and discuss the strategies depicted in Table 5.

Double Colon. The double colon strategy consists of test vectors that inject another double colon in the modified header after the existing one. Interestingly, this strategy was successful in Iran for CL*/TE and CL/TE*. We suspect that the censor does not accept this format and that the CL*/TE test vector falls into the same category as Wrapping which we explain below. This strategy also worked for two to four Apache versions and some of the domains on the CitizenLab list for Russia and China.

White-space Injection. This strategy consists of injecting white space in the modified header. The injection of white space is done through injecting line breaks (`\0d` or `\0a`), tabs, or spaces. According to the standard, tabs and spaces are allowed, however, line breaks are not. Iran's censor accepted the modified header with tabs and spaces in the CL*/TE and TE*/CL case, which makes Iran's censor circumventable with a standard-compliant circumvention technique. China's censor was stricter and did not accept tabs and spaces at all. In some cases, the GFW seems to prefer the CL header when tabs or spaces are injected into the TE header. Inserting tabs and spaces was successful for older Apache and Nginx versions, CDNs, and many servers from the CitizenLab lists. Neither Iran nor China accepts headers with injected line feeds (`\0a`) but Iran accepts TE headers with injected carriage returns (`\0d`). Further analysis showed that Iran ignores carriage returns (`\0d`) in header names but fails to parse a header if the header name contains a line feed (`\0a`). Inserted line breaks were mostly accepted by old Apache versions and Cloudflare.

Letter Case. The letter case strategy changes the case of all or some letters to upper case. In some instances, this led the GFW to ignore the TE header and interpret the CL header instead. We suspect that some parts of the GFW rely on the correct case of the headers. Changing the case of letters in the header name is standard-compliant as header names are case-insensitive. Many Apache and Nginx versions, aws, and servers from the CitizenLab lists preferred the other header.

Wrapping. The wrapping strategy wraps the value of the modified header with extra bytes. This strategy was very successful in Iran for the CL*/TE type. At first, we suspected that Iran's censor ignored symbols it could not parse when interpreting the value of the CL header. When we tried to verify this behavior, we recognized that we could not circumvent Iranian censorship by invalidating the CL header if no TE header was present. The CL header was ignored and the second request was blocked. The presence of the TE header was required for the Iranian censor to ignore the second request. We stress that while the presence was required, the Iranian censor did not interpret it as it would have recognized and blocked the second request; it did so when the value of the CL header was a valid integer. Overall, Iran's censor failed to interpret the TE header when a CL header with an invalid value was present. We conclude that the Iranian censor gracefully handles invalid values for the CL header and attempts to parse the following HTTP packets. The same invalid value in the CL headers seems to let the censor fail and pass the following traffic when it interprets the TE header. We suspect an implementation bug in Iran's censor which only becomes apparent through the interplay of its CL and TE header parsing. The GFW sometimes accepted the CL header when it was ended by an additional line feed (`\0a`) or when it was ended by two line feeds and an additional header. The acceptance of test vectors in the Wrapping category by web servers varied, with some being accepted by newer Apache versions.

Invalid Header. This strategy invalidates the header name. For example, this can be done by injecting an additional Unicode character. As a result, the header itself becomes invalid, which leads to many web servers accepting the request because they cannot parse the modified header. All test vectors that were accepted by the newest Apache and Nginx versions are of this type; they treat the modified CL or TE header as an unknown header and no HRS countermeasures are undertaken. This strategy was only successful in Russia where all of our test vectors were successful, as discussed above. The reason for this is that a censor would normally have to parse the modified header correctly, which it is unable to do for an invalid header name.

Double-Header. All censors accepted a test vector with two TE headers instead of one of which only the last indicates a chunked encoding. As the vector has the TE*/CL type censors still interpret the chunked encoding when multiple TE headers are present and do not fall back to the CL header. This test vector was accepted by Nginx 1.14 and some servers from the CitizenLab lists. The standard-compliant way to indicate multiple encodings is in a list of the same TE header.

6 LIMITATIONS

As HRS is a security vulnerability, many web servers have patched their implementations to be less accepting of known HRS vectors. Still, we showed the viability of HRS as a censorship circumvention technique on commonly used web server implementations and public web servers. Furthermore, the test vectors we used in our evaluations were all taken from previous work about HRS. We suggest that a large-scale analysis of web servers and vantage points, using more test vectors, will produce more censorship circumvention techniques that complement the techniques we describe in this work. We consider a fuzz-style approach as in the work of Jabiyev et al. [34] promising. Finally, we evaluated censorship from specific vantage points. Our results might not apply to other vantage points in specific countries.

7 ETHICAL CONSIDERATIONS

We designed our methodology with a minimal impact on web servers, clients, and residents in the countries we evaluated. Like previous work [10, 28, 42], we used vantage points for our censorship scans; we sent all requests from these vantage points to another vantage point in Germany. We did not use or send requests to machines owned by third parties. The evaluations of Apache and Nginx were executed on local machines. Only our evaluations of CDNs and domains on the CitizenLab list imposed traffic on their respective web servers. To each, we sent around 45,000 HTTP requests which we consider a negligible number for a publicly accessible web server.

Table 5: Selection of test vectors that circumvented censorship and were accepted by web servers. ●=successful, ◯=partially successful, −=unsuccessful. Successful test vectors circumvented censorship with both censored domains in the Host header. In China, partially successful test vectors circumvented the GFW in some but not all of its executions. For the CitizenLab lists, a test vector is counted as successful if it is accepted by at least one of the five domains. <len> stands for the correct length value until the end of the second request. Injected ASCII values are represented by their hex values in the format \[value]. Notably, the presented vectors are always applied to the general HRS structure, depicted in Figure 1.

Type	Category	Vector	Censors			Apache 2.X.X					Nginx 1.XX					CDN		CitizenLab			
			CN	IR	RU	2.15	4.6	4.29	4.41	4.57	14	18	21	22	25.2	CF[1]	aws[2]	IR	RU	CN	GL
CL*/TE	Double Colon	`Content-Length:: <len>`	−	●	●	●	●	●	●	−	−	−	−	−	−	−	−	−	●	●	−
	White-Space Injection	`Content-Length\20: <len>\20`	−	●	●	●	●	−	−	−	●	●	−	−	−	●	●	−	●	●	●
		`Content-Length\09: <len>\09`	−	●	●	●	●	−	−	−	●	●	−	−	−	●	●	−	●	●	●
	Wrapping	`Content-Length: '<len>'`	−	●	●	●	●	●	●	−	−	−	−	−	−	−	−	−	●	●	−
		`Content-Length:\20<len>\20`	−	●	●	●	●	●	●	−	●	●	−	−	−	−	●	−	●	●	●
		`Content-Length: <len>\20\0aX: X`	−	●	●	●	●	−	−	−	●	●	−	−	−	−	●	−	●	●	●
		`Content-Length: <len>\u00FF\0aX: X`†	−	●	●	●	●	−	−	−	−	−	−	−	−	−	−	−	−	−	−
		`Content-Length:\0b <len>`	−	●	●	●	●	−	−	−	−	−	−	−	−	−	−	−	−	−	−
		`Content-Length: <len>\0a\0aX: X`	◯	●	●	−	−	−	−	−	−	−	−	−	−	●	●	●	−	●	●
		`Content-Length: <len>\0a`	◯	●	●	−	−	−	−	−	−	−	−	−	−	●	−	−	−	−	−
	Invalid Header	`Content-Encoding: <len>`	−	−	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
		`\u00FFContent-Length: <len>\u00FF`†	−	−	●	●	●	−	−	−	●	●	●	●	●	●	−	●	●	●	●
TE*/CL	Double Header	`Transfer-Encoding: identity\0d\0a Transfer-Encoding: chunked`	●	●	●	−	−	−	−	−	●	−	−	−	−	−	●	−	●	●	−
	White-Space Injection	`\20Transfer-Encoding: chunked\20`	−	●	●	−	−	−	−	−	●	●	−	−	−	●	●	−	●	●	●
		`\09Transfer-Encoding: chunked\09`	−	●	●	−	−	−	−	−	●	●	−	−	−	●	●	−	●	●	●
		`Transfer-Encoding\0d: chunked\0d`	−	●	●	●	●	−	−	−	−	−	−	−	−	−	−	−	−	−	−
	Invalid Header	`Content-Encoding: chunked`	−	−	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
		`\u00FFTransfer-Encoding: chunked\u00FF`†	−	−	●	●	●	−	−	−	●	●	●	●	●	●	−	●	●	●	●
		`Transfer_Encoding: chunked`	−	−	●	●	●	●	●	●	●	●	●	●	●	−	●	●	●	●	●
CL/TE*	Letter Case	`TRANSFER-ENCODING: CHUNKED`	◯	−	●	●	●	●	●	−	●	●	−	−	−	−	●	−	●	●	●
		`TrAnSFer-EnCODinG: cHuNkeD`	◯	−	●	●	●	●	●	−	●	●	−	−	−	−	●	−	●	●	●
	Double Colon	`Transfer-Encoding:: chunked`	−	●	●	−	−	●	●	−	−	−	−	−	−	−	−	−	●	●	−
	White-Space Injection	`Transfer-Encoding\20: chunked\20`	◯	−	●	●	●	−	−	−	−	−	−	−	−	−	−	−	−	−	−
		`Transfer-Encoding\09: chunked\09`	◯	−	●	●	●	−	−	−	−	−	−	−	−	−	−	−	−	−	−
		`Transfer-Encoding:\0a chunked`	◯	−	●	●	●	−	−	−	−	−	−	−	−	−	−	−	−	−	−
		`\0aTransfer-Encoding: chunked\0a`	●	●	●	−	−	−	−	−	−	−	−	−	−	●	−	−	−	−	−

[1] Cloudflare
[2] awselb/2.0
† The extended ASCII character `\u00FF` is UTF-8 encoded to `\xc3\xbf`.

We also believe that our research is more beneficial to the censorship circumvention community than it is to censors. To the best of our knowledge, HRS has not been discovered as a censorship circumvention technique by the community and can complement currently used techniques. While censors might be able to fix their implementations, they would have to allocate resources for that process. We also want to emphasize the general contribution of our work: we showed that a web security vulnerability can be turned into a censorship circumvention technique.

8 CONCLUSIONS

In this paper, we presented novel censorship circumvention techniques, transferred from a web security vulnerability. We successfully evaded censorship in Iran, China, and Russia and showed that web servers accept our requests. For at least one vantage point in Russia, we could also identify an additional standard-compliant censorship circumvention technique. Furthermore, we detected inconsistencies in China's GFW and an implementation bug in Iran's censor. We provide our code and data on GitHub[1] to promote reproducibility. This repository also contains a proof-of-concept that circumvents censorship in Iran with one of our test vectors. We project that HRS can be applied by proxies to smuggle clients' requests and implemented in tools such as Geneva to find advanced circumvention techniques that incorporate HRS. The direct derivation of censorship circumvention techniques from an attack is a novel approach to censorship circumvention and we suggest that it applies to other vulnerabilities. We hope our findings aid affected people and incite new research around the similarities between vulnerabilities and censorship circumvention techniques.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. Niklas Niere and Felix Lange were supported by the German Federal Ministry of Education and Research (BMBF) through the project KoTeBi (16KIS1556K).

[1] https://github.com/UPB-SysSec/SmugglingCircumventionResults

REFERENCES

[1] 2017. CVE-2016-8743. https://nvd.nist.gov/vuln/detail/CVE-2016-8743

[2] 2023. CVE-2022-22720. https://nvd.nist.gov/vuln/detail/CVE-2022-22720

[3] Chaabane Abdelberi, Terence Chen, Mathieu Cunche, Emiliano De Cristofaro, Arik Friedman, and Mohamed Ali Kâafar. 2014. Censorship in the Wild: Analyzing Internet Filtering in Syria. In *Proceedings of the 2014 Internet Measurement Conference, IMC 2014, Vancouver, BC, Canada, November 5-7, 2014*, Carey Williamson, Aditya Akella, and Nina Taft (Eds.). ACM, 285–298. https://doi.org/10.1145/2663716.2663720

[4] Anonymous. 2014. Towards a Comprehensive Picture of the Great Firewall's DNS Censorship. In *4th USENIX Workshop on Free and Open Communications on the Internet, FOCI '14, San Diego, CA, USA, August 18, 2014*, Jedidiah R. Crandall and Vern Paxson (Eds.). USENIX Association. https://www.usenix.org/conference/foci14/workshop-program/presentation/anonymous

[5] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. 2013. Internet Censorship in Iran: A First Look. In *3rd USENIX Workshop on Free and Open Communications on the Internet, FOCI '13, Washington, D.C., USA, August 13, 2013*, Jedidiah R. Crandall and Joss Wright (Eds.). USENIX Association. https://www.usenix.org/conference/foci13/workshop-program/presentation/aryan

[6] Guy Baron and Gareth Hall. 2015. Access online: Internet governance and image in Cuba. *Bulletin of Latin American Research* 34, 3 (2015), 340–355.

[7] Abhishek Bhaskar and Paul Pearce. 2022. Many Roads Lead To Rome: How Packet Headers Influence DNS Censorship Measurement. In *USENIX Security Symposium*. USENIX. https://www.usenix.org/system/files/sec22-bhaskar.pdf

[8] Kevin Bock, Pranav Bharadwaj, Jasraj Singh, and Dave Levin. 2021. Your Censor is My Censor: Weaponizing Censorship Infrastructure for Availability Attacks. In *IEEE Security and Privacy Workshops, SP Workshops 2021, San Francisco, CA, USA, May 27, 2021*. IEEE, 398–409. https://doi.org/10.1109/SPW53761.2021.00059

[9] Kevin Bock, Yair Fax, Kyle Reese, Jasraj Singh, and Dave Levin. 2020. Detecting and Evading Censorship-in-Depth: A Case Study of Iran's Protocol Whitelister. In *10th USENIX Workshop on Free and Open Communications on the Internet (FOCI 20)*. USENIX Association. https://www.usenix.org/conference/foci20/presentation/bock

[10] Kevin Bock, George Hughey, Xiao Qiang, and Dave Levin. 2019. Geneva: Evolving Censorship Evasion Strategies. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz (Eds.). ACM, 2199–2214. https://doi.org/10.1145/3319535.3363189

[11] Kevin Bock, iyouport, Anonymous, Louis-Henri Merino, David Fifield, Amir Houmansadr, and Dave Levin. 2020. Exposing and Circumventing China's Censorship of ESNI. https://gfw.report/blog/gfw_esni_blocking/en/

[12] Kevin Bock, Gabriel Naval, Kyle Reese, and Dave Levin. 2021. Even censors have a backup: Examining china's double https censorship middleboxes. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet*. 1–7.

[13] bol van. 2023. *zapret*. https://github.com/bol-van/zapret

[14] CitizenLab. 2024. CitizenLab Lists. https://github.com/citizenlab/test-lists/tree/master/lists

[15] Richard Clayton, Steven J Murdoch, and Robert NM Watson. 2006. Ignoring the great firewall of china. In *International Workshop on Privacy Enhancing Technologies*. Springer, 20–35.

[16] Cloudflare. 2024. *Cloudflare Radar - Adoption & Usage in China*. https://radar.cloudflare.com/adoption-and-usage/cn

[17] defparam. 2021. *Smuggler*. https://github.com/defparam/smuggler

[18] Tim Dierks and Eric Rescorla. 2008. RFC 5246: The transport layer security (TLS) protocol version 1.2.

[19] Maxim Dounin. 2021. *Nginx, commit a6c109fea5c13b8aa13ed95ca00a64d62601042b*. https://github.com/nginx/nginx/commit/a6c109f

[20] D Eastlake 3rd. 2011. RFC 6066: Transport Layer Security (TLS) Extensions: Extension Definitions.

[21] Emil Lerner. 2023. *http2smugl*. https://github.com/neex/http2smugl

[22] Roya Ensafi, Philipp Winter, Abdullah Mueen, and Jedidiah R. Crandall. 2015. Analyzing the Great Firewall of China Over Space and Time. *Proc. Priv. Enhancing Technol.* 2015, 1 (2015), 61–76. https://doi.org/10.1515/popets-2015-0005

[23] Roy T. Fielding, Mark Nottingham, and Julian Reschke. 2022. HTTP/1.1. RFC 9112. https://doi.org/10.17487/RFC9112

[24] Roy T. Fielding and Julian Reschke. 2014. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230. https://doi.org/10.17487/RFC7230

[25] Arturo Filasto and Jacob Appelbaum. 2012. OONI: open observatory of network interference.. In *FOCI*.

[26] Genevieve Gebhart and Tadayoshi Kohno. 2017. Internet Censorship in Thailand: User Practices and Potential Threats. In *2017 IEEE European Symposium on Security and Privacy, EuroS&P 2017, Paris, France, April 26-28, 2017*. IEEE, 417–432. https://doi.org/10.1109/EuroSP.2017.50

[27] IETF HTTP Working Group. 2015. HTTP/2 Frequently Asked Questions. https://http2.github.io/faq/

[28] Michael Harrity, Kevin Bock, Frederick Sell, and Dave Levin. 2022. GET /out: Automated Discovery of Application-Layer Censorship Evasion Strategies. In *31st USENIX Security Symposium (USENIX Security 22)*. USENIX Association, Boston, MA, 465–483. https://www.usenix.org/conference/usenixsecurity22/presentation/harrity

[29] Sadegh Hayeri. 2022. *{GreenTunnel}*. https://github.com/SadeghHayeri/GreenTunnel

[30] NP. Hoang, AA. Niaki, J. Dalek, J. Knockel, P. Lin, B. Marczak, M. Crete-Nishihata, P. Gill, and M. Polychronakis. 2021. How Great is the Great Firewall? Measuring China's DNS Censorship. In *30th USENIX Security Symposium*. USENIX Association, 3381–3398. https://www.usenix.org/conference/usenixsecurity21/presentation/hoang

[31] Nguyen Phong Hoang, Arian Akhavan Niaki, Jakub Dalek, Jeffrey Knockel, Pellaeon Lin, Bill Marczak, Masashi Crete-Nishihata, Phillipa Gill, and Michalis Polychronakis. 2021. How Great is the Great Firewall? Measuring China's DNS Censorship. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael Bailey and Rachel Greenstadt (Eds.). USENIX Association, 3381–3398. https://www.usenix.org/conference/usenixsecurity21/presentation/hoang

[32] Troy Hunt. 2024. Why Not HTTPS? https://whynohttps.com/

[33] Hunter. 2023. TechLookup - List Websites by Technologies - Hunter. https://hunter.io/techlookup

[34] Bahruz Jabiyev, Steven Sprecher, Kaan Onarlioglu, and Engin Kirda. 2021. T-Reqs: HTTP Request Smuggling with Differential Fuzzing. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 1805–1820.

[35] Jill Jermyn and Nicholas Weaver. 2017. Autosonda: Discovering Rules and Triggers of Censorship Devices. https://www.usenix.org/conference/foci17/workshop-program/presentation/jermyn

[36] Lin Jin, Shuai Hao, Haining Wang, and Chase Cotton. 2021. Understanding the Practices of Global Censorship through Accurate, End-to-End Measurements. *Proc. ACM Meas. Anal. Comput. Syst.* 5, 3, Article 43 (dec 2021), 25 pages. https://doi.org/10.1145/3491055

[37] JonSnowWhite. 2023. *DPYProxy*. https://github.com/UPB-SysSec/DPYProxy

[38] James Kettle. 2019. HTTP Desync Attacks: Request Smuggling Reborn. https://portswigger.net/research/http-desync-attacks-request-smuggling-reborn

[39] James Kettle. 2021. HTTP/2: The Sequel is Always Worse. https://portswigger.net/research/http2

[40] Amit Klein. 2020. Http Request Smuggling in 2020–New Variants, New Defenses and New Challenges. *Black Hat Briefings USA* 8 (2020).

[41] krlvm. 2022. *{PowerTunnel}*. https://github.com/krlvm/PowerTunnel

[42] Fangfan Li, Abbas Razaghpanah, Arash Molavi Kakhki, Arian Akhavan Niaki, David Choffnes, Phillipa Gill, and Alan Mislove. 2017. lib• erate,(n) a library for exposing (traffic-classification) rules and avoiding them efficiently. In *Proceedings of the 2017 Internet Measurement Conference*. 128–141.

[43] Chaim Linhart, Ronen Heled, Amit Klein, and Steve Orrin. 2005. Http Request Smuggling.

[44] Alexander Master and Christina Garman. 2023. A Worldwide View of Nation-state Internet Censorship. *Free and Open Communications on the Internet* (2023).

[45] Zubair Nabi. 2013. The Anatomy of Web Censorship in Pakistan. In *3rd USENIX Workshop on Free and Open Communications on the Internet, FOCI '13, Washington, D.C., USA, August 13, 2013*, Jedidiah R. Crandall and Joss Wright (Eds.). USENIX Association. https://www.usenix.org/conference/foci13/workshop-program/presentation/nabi

[46] Henrik Nielsen, Jeffrey Mogul, Larry M Masinter, Roy T. Fielding, Jim Gettys, Paul J. Leach, and Tim Berners-Lee. 1999. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1.

[47] Niklas Niere, Sven Hebrok, Juraj Somorovsky, and Robert Merget. 2023. Poster: Circumventing the GFW with TLS Record Fragmentation. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 3528–3530.

[48] nomoresat. 2021. *{DPI Tunnel for Android}*. https://github.com/nomoresat/DPITunnel-android

[49] OpenResty. 2024. *OpenResty - Open Source*. https://openresty.org/en/

[50] Ramakrishna Padmanabhan, Arturo Filastò, Maria Xynou, Ram Sundara Raman, Kennedy Middleton, Mingwei Zhang, Doug Madory, Molly Roberts, and Alberto Dainotti. 2021. A multi-perspective view of Internet censorship in Myanmar. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet*. 27–36.

[51] Paul Pearce, Ben Jones, Frank Li, Roya Ensafi, Nick Feamster, Nick Weaver, and Vern Paxson. 2017. Global measurement of {DNS} manipulation. In *26th USENIX Security Symposium (USENIX Security 17)*. 307–323.

[52] Ram Sundara Raman, Leonid Evdokimov, Eric Wustrow, J. Alex Halderman, and Roya Ensafi. 2020. Investigating Large Scale HTTPS Interception in Kazakhstan. In *IMC '20: ACM Internet Measurement Conference, Virtual Event, USA, October 27-29, 2020*. ACM, 125–132. https://doi.org/10.1145/3419394.3423665

[53] Reethika Ramesh, Ram Sundara Raman, Matthew Bernhard, Victor Ongkowijaya, Leonid Evdokimov, Anne Edmundson, Steven Sprecher, Muhammad Ikram, and Roya Ensafi. 2020. Decentralized control: A case study of russia. In *Network and Distributed Systems Security (NDSS) Symposium 2020*.

[54] Eric Rescorla. 2018. Rfc 8446: The transport layer security (tls) protocol version 1.3.
[55] Kushagra Singh, Gurshabad Grover, and Varun Bansal. 2020. How India censors the web. In *Proceedings of the 12th ACM Conference on Web Science*. 21–28.
[56] Ram Sundara Raman, Prerana Shenoy, Katharina Kohls, and Roya Ensafi. 2020. Censored Planet: An Internet-Wide, Longitudinal Censorship Observatory. In *In ACM SIGSAC Conference on Computer and Communications Security (CCS)*.
[57] Taobao. 2024. *The Tengine Web Server*. https://tengine.taobao.org/
[58] ValdikSS. 2022. *GoodbyeDPI — Deep Packet Inspection circumvention utility*. https://github.com/ValdikSS/GoodbyeDPI
[59] W3Techs. 2023. Usage Statistics and Market Share of Web Servers. https://w3techs.com/technologies/overview/web_server
[60] Ben Wagner. 2014. The politics of internet filtering: The United Kingdom and Germany in a comparative perspective. *Politics* 34, 1 (2014), 58–71.
[61] Philipp Winter and Jedidiah R Crandall. 2012. The Great Firewall of China: How it blocks Tor and why it is hard to pinpoint. *Login: The Usenix Magazine* 37, 6 (2012), 42–50.
[62] Xueyang Xu, Zhuoqing Morley Mao, and J. Alex Halderman. 2011. Internet Censorship in China: Where Does the Filtering Occur?. In *Passive and Active Measurement - 12th International Conference, PAM 2011, Atlanta, GA, USA, March 20-22, 2011. Proceedings (Lecture Notes in Computer Science, Vol. 6579)*, Neil Spring and George F. Riley (Eds.). Springer, 133–142. https://doi.org/10.1007/978-3-642-19260-9_14
[63] Diwen Xue, Benjamin Mixon-Baca, ValdikSS, Anna Ablove, Beau Kujath, Jedidiah R. Crandall, and Roya Ensafi. 2022. TSPU: Russia's decentralized censorship system. In *Proceedings of the 22nd ACM Internet Measurement Conference, IMC 2022, Nice, France, October 25-27, 2022*, Chadi Barakat, Cristel Pelsser, Theophilus A. Benson, and David R. Choffnes (Eds.). ACM, 179–194. https://doi.org/10.1145/3517745.3561461
[64] Diwen Xue, Reethika Ramesh, Valdik S. S, Leonid Evdokimov, Andrey Viktorov, Arham Jain, Eric Wustrow, Simone Basso, and Roya Ensafi. 2021. Throttling Twitter: an emerging censorship technique in Russia. In *IMC '21: ACM Internet Measurement Conference, Virtual Event, USA, November 2-4, 2021*, Dave Levin, Alan Mislove, Johanna Amann, and Matthew Luckie (Eds.). ACM, 435–443. https://doi.org/10.1145/3487552.3487858
[65] Diwen Xue, Reethika Ramesh, Valdik S. S, Leonid Evdokimov, Andrey Viktorov, Arham Jain, Eric Wustrow, Simone Basso, and Roya Ensafi. 2021. Throttling Twitter: an emerging censorship technique in Russia. In *IMC '21: ACM Internet Measurement Conference, Virtual Event, USA, November 2-4, 2021*, Dave Levin, Alan Mislove, Johanna Amann, and Matthew Luckie (Eds.). ACM, 435–443. https://doi.org/10.1145/3487552.3487858
[66] xvzc. 2023. *{SpoofDPI}*. https://github.com/xvzc/SpoofDPI
[67] Tarun Kumar Yadav, Akshat Sinha, Devashish Gosain, Piyush Kumar Sharma, and Sambuddho Chakravarty. 2018. Where The Light Gets In: Analyzing Web Censorship Mechanisms in India. In *Proceedings of the Internet Measurement Conference 2018, IMC 2018, Boston, MA, USA, October 31 - November 02, 2018*. ACM, 252–264. https://dl.acm.org/citation.cfm?id=3278555

A VECTOR VIABILITY

Table 6: Targets for our vector viability scan

Category	Targets				
Apache	2.2.15	2.4.6	2.4.29	2.4.41	2.4.57-latest
Nginx	1.14	1.18	1.21	1.22	1.25.2-latest
Akamai	akamai.com	united.com	starwars.com	amd.com	ikea.com
Amazon	redhat.com	sony.com	flickr.com	rakuten.com	eventbrite.com
Cloudflare	tinyurl.com	creativecommons.org	w3.org	cloudflare.com	vimeo.com
Fastly[†]	tomsguide.com	giphy.com	techradar.com	behance.net	usatoday.com
Global CL	data.worldbank.org	www.nmrc.org	www.hackhull.com	www.sbc.net	instinctmagazine.com
China CL	www.redcross.org.cn	clb.org.hk	www.grandlisboahotels.com	www.president.gov.tw	www.sohu.com
Iran CL	www.funpatogh.com	www.dwturkce.com	parsget.com	www.aparat.com	zezito.ir
Russia CL	zezito.ir	nr2.com.ua	www.wonderzine.com	www.sotnik-tv.com	perevedem.ru

[†] We erroneously scanned stackoverflow.com and behance.com for Fastly when they were hosted on Cloudflare and Amazon respectively. We considered the result of these scans for Cloudflare and Amazon instead and evaluated two alternative websites for Fastly instead.

B SERVER SPECIFICATIONS

Table 7: Specification of the server in China.

Country:	Zhengzhou. China
Autonomous System Number:	4837
Vendor:	China VPS Hosting
URL:	https://chinavpshosting.com/
ISP:	CHINA UNICOM (state-owned)

Table 8: Specification of the server in Russia.

Country:	Moscow, Russia
Autonomous System Number:	50867
Vendor:	Server Wala
URL:	https://serverwala.cloud/
ISP:	HOSTKEY B.V. (private)

Table 9: Specification of the server in Iran.

Country:	Mashhad, Iran
Autonomous System Number:	201295
Vendor:	Avanetco
URL:	https://www.avanetco.com/
ISP:	Shabakeh Ertebatat Artak Towseeh PJSC (private)

Table 10: Specification of the server in Germany.

Country:	Berlin, Germany
Autonomous System Number:	201295
Vendor:	IONOS
URL:	https://www.ionos.de/
Internet Service Provider:	IONOS SE (private)