Transport Layer Obscurity: Circumventing SNI Censorship on the TLS-Layer

Niklas Niere*, Felix Lange*, Robert Merget†, and Juraj Somorovsky*
*{niklas.niere, felix.lange, juraj.somorovsky}@upb.de, Paderborn University, Paderborn, DE
†robert.merget@tii.ae, Technology Innovation Institute, Abu Dhabi, UAE

Abstract— **HTTPS composes large parts of today's Internet traffic and has long been subject to censorship efforts in different countries. While censors analyze the Transport Layer Security (TLS) protocol to block encrypted HTTP traffic, censorship circumvention efforts have primarily focused on other protocols such as TCP. In this paper, we hypothesize that the TLS protocol offers previously unseen opportunities for censorship circumvention techniques. We tested our hypothesis by proposing possible censorship circumvention techniques that act on the TLS protocol. To validate the effectiveness of these techniques, we evaluate their acceptance by popular TLS servers and successfully demonstrate that these techniques can circumvent censors in China and Iran. In our evaluations, we discovered 38—partially standard-compliant—distinct censorship circumvention techniques, which we could group into 11 unique categories. Additionally, we provide novel insights into how China censors TLS traffic by presenting evidence of at least three distinct censorship appliances. We suspect that other parts of China's censorship apparatus and other censors exhibit similar structures and advocate future censorship research to anticipate them. With this work, we hope to aid people affected by censorship and stimulate further research into censorship circumvention using cryptographic protocols.**

1. Introduction

Censorship is employed by various countries to restrict network access of their citizens [2], [8], [27], [47], [56], [81], [83]. A prominent example of a censor is the Great Firewall of China (GFW). First analyzed in 2006 by Clayton et al. [18], the GFW has become the most analyzed censor globally [5], [13], [14], [22], [35], [44], [75], [81] while other censors like Iran are also commonly analyzed [4], [8], [11]. These analyses determined general censor behavior and found different censorship evasion techniques to enable a more open Internet.

One of the main goals of censors is to prohibit access to certain websites. To do so, censors are known to perform deep packet inspection (DPI) for different protocols, such as HTTP [8], [32] and DNS [9], [32], [35] and encryption protocols, such as Transport Layer Security (TLS) [49], [81] and VPNs [80]. Depending on the analyzed protocol, blocking can be performed through methods such as TCP RSTs [8], [18], [42], block pages [13], [42], or dropping packets [75], [79]. Censoring plaintext protocols is especially easy for censors. For example, when plaintext HTTP is used, censors can read contained data, such as the server hostname, and decide whether to block the connection.

HTTP can be combined with TLS to encrypt the underlying traffic (HTTPS). HTTPS is widely used across the Internet [19], [29]. Using HTTPS prevents censors from directly analyzing plaintext HTTP. To censor TLS-encrypted content, censors can block specific IP addresses that host the unwanted content. This is possible because TLS does not encrypt IP headers that contain the server's IP address. While used by censors in practice, this strategy has two severe limitations. First, censored servers can change their IPs. Second, in virtual host environments, multiple websites can share the same IP address, forcing the censor to also block benign content. Such scenarios are typical for Content Delivery Networks (CDNs) and cloud providers, making IP blocking impractical for many domains. Instead of blocking IPs, censors rely on the Server Name Indication (SNI) extension when censoring TLS.

Server Name Indication (SNI). Hosting multiple servers on the same IP is realized in TLS with the SNI extension [21]. Using the SNI extension, the client conveys to the server the virtual host it wants to reach. While the extension was optional for older TLS versions, TLS 1.3 requires servers to support the SNI extension [59]. The client sends the SNI extension in the very first TLS message (`ClientHello`), which is sent unencrypted and thus reveals the hostname to a passive observer. Therefore, the SNI extension allows the censors to learn the intended hostname and block the content. Notably, some servers in virtual host environments are reachable without an SNI extension or an incorrect hostname in the SNI extension. This is possible when the virtual host environment identifies the destination domain with the encrypted HTTP Host header or defaults to one of its hosted domains.

Censorship Circumvention. In previous research, censorship has been circumvented on different levels. To prevent censorship analyses on the HTTP level, HTTP-based circumventions usually try to obfuscate censored keywords such as the Host header [32], [83] or hide the request entirely [46]. However, when TLS is used to encrypt HTTP traffic, HTTP-level circumventions are not helpful anymore.

As TCP is application-unspecific and widely used, many strategies on the TCP level have been created that mainly attempt to invalidate the internal TCP state of a censor to prevent censors from blocking content [11], [13]. The most successful strategies are implemented by major censorship evasion tools such as GoodbyeDPI [73] or zapret [16]. Unfortunately, censors are catching up to TCP-level circumvention techniques such as TCP fragmentation [13], necessitating additional avenues for censorship circumvention.

Research Gap. While TCP-based and HTTP-based censorship has been analyzed extensively, there is still unexplored potential in censorship circumvention techniques on the TLS layer. Singular circumvention techniques on the TLS layer have been found such as TLS record fragmentation [49], record injection [81], and hostname alterations [57]. While all three techniques obfuscate the hostname in the SNI extension, they consider only specific parts of the TLS protocol and do not combine different techniques. Raman et al. [57] and Xue et al. [81] have also limited their analysis to censors and make no assertion about the acceptance of their techniques on real-world web servers. Combining these observations, we identify a research gap to thoroughly analyze the TLS handshake for censorship circumvention techniques and their effectiveness against censors and real-world servers.

Methodology. To close this gap and develop TLS-level circumvention techniques that not only circumvent censors but also are accepted by TLS servers, we explore the following research questions:

RQ1: Which censorship circumvention techniques can be designed based on the complexity of the TLS protocol?
RQ2: To what extent are TLS messages that are manipulated through circumvention techniques accepted by real-world servers?
RQ3: Which techniques servers accept are most successful against real-world censors?

To answer these questions, we developed a five-phase methodology. In the first phase, we identify TLS-level censorship circumventions, which we then implement and combine in the second phase. In the following phases, we evaluated our test vector against real-world servers and censors. We concentrated on techniques concerning the `ClientHello` flight, which contains the SNI extension. For example, we inserted a null byte into the SNI extension. In another technique, we fragmented the `ClientHello` message across multiple TLS records, similar to TCP fragmentation. Such deep protocol-level changes may lead to servers rejecting the created messages. Therefore, we evaluated whether combinations of our techniques are accepted by popular web servers and CDNs before testing them on real-world censors.

Findings. As a result of our evaluations, we gained deep insights into the inner workings of the censorship mechanism in China and Iran, allowing us to identify a previously unnoticed third middle box in China's censorship infrastructure. We then present six categories of successful circumvention techniques for the analyzed censors. Our results show that the analyzed censors show diverse behavior when presented with our circumvention techniques (cf. Table 4). The most successful category is TLS record fragmentation. Strategies in this category were widely accepted by all web servers and successfully circumvented censorship in China and Iran. Interestingly, TLS record fragmentation does not cause any protocol violation and can be applied without having root access to the system, making it an ideal circumvention technique for the usage in proxies. Additionally, invalidating version fields and removing the SNI extension entirely were also very successful strategies.

Contributions. We make the following contributions:

- We present a novel five-phase methodology to analyze censorship circumvention techniques systematically. We show how this methodology can be applied to perform the first systematic analysis of the TLS handshake for censorship circumvention possibilities. We identified novel strategies on the TLS level and transformed them into censorship circumvention techniques.
- We developed *Censor Scanner*,[1] an open-source tool that implements and combines the discovered TLS-level techniques and fully automatically evaluates the extent to which they are accepted by a given censor and by servers.
- We demonstrate 24 unique successful circumvention techniques by executing automated scans with *Censor Scanner* against local servers, CDNs, and censors in China and Iran.
- We provide novel insights into Iran's and China's censorship infrastructure and present evidence of three distinct middleboxes that perform China's TLS censorship. We characterize them regarding their censorship behavior and circumventability.

2. Background

Transport Layer Security (TLS) is arguably one of the most important cryptographic protocols in the world. Before it can encrypt and authenticate data, TLS must negotiate the necessary cryptographic parameters such as keys and algorithms. TLS performs this negotiation during a partially unencrypted handshake. Figure 5 in Appendix D depicts the handshake of a TLS 1.2 connection [20]. To stay within the scope of this paper, we focus on which messages are encrypted and which contain the server's domain name in the following. TLS 1.2 only encrypts the `Finished` messages of the handshake. Increasing efficiency and protection, TLS 1.3 removes a round-trip from the handshake and encrypts all handshake messages except the `ClientHello` and the beginning of the `ServerHello` flight [59]. This has certain implications for censors that want to determine the server's domain name. As the server's domain name is contained in the `Certificate` message and the Server

1. https://github.com/tls-attacker/Censor-Scanner/releases/tag/v1.0_sp2025

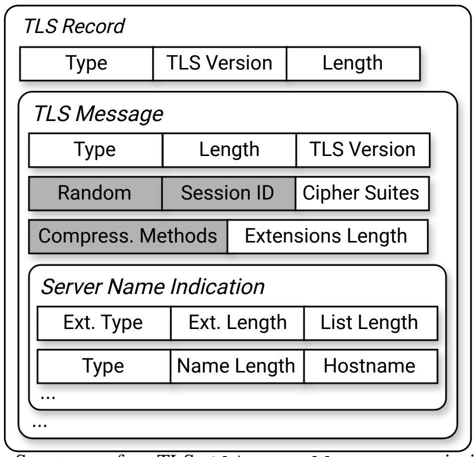

Figure 1. Structure of a TLS `ClientHello` message inside a TLS record with an SNI extension. Greyed-out fields are not used for our manipulations.

Name Indication (SNI) extension in the `ClientHello` message, a censor could analyze either in a TLS 1.2 connection but only the latter in a TLS 1.3 connection. Our work focuses on analyzing SNI-based censorship.

Record and Message Layer. Handshake messages are wrapped in TLS records. The structure of a `ClientHello` message is depicted in Figure 1. The TLS message structure contains the actual message content, such as the proposed cipher suites or the SNI extension, and header information, such as the message's length. The TLS record structure contains the TLS message and its header information, such as the type of the contained message and the record's length. The header bytes of a TLS record of a `ClientHello` message always start with `0x16` for the message type and `0x0303`, `0x0302`, or `0x0301` for the TLS version. The given TLS version indicates the lowest supported version by the client. Notably, the TLS version field of the record header can differ from the TLS version field in the message header.

To transfer messages of arbitrary lengths, TLS supports record fragmentation. This feature allows the TLS peer to fragment the message across several records. Figure 2 depicts a `ClientHello` message that is split over two TLS records. This so-called record fragmentation occurs naturally when a TLS message is too large for a single TLS record. Specifically, TLS messages have a larger maximum size ($2^{24} - 1$ bytes) than TLS records (2^{14} bytes). While the record header's length field would potentially allow for record sizes up to $2^{16} - 1$ bytes, the standard prohibits implementations from sending such large records. TLS record fragmentation can also be enforced by intentionally choosing small TLS records. Furthermore, the `max_fragment_length` extension [21] and the `record_size_limit` extension [69] specify ways to lower the maximum size of TLS records intentionally.

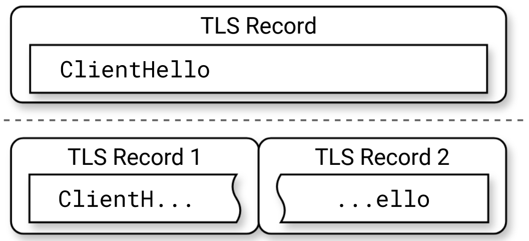

Figure 2. A TLS `ClientHello` message as part of one TLS record and split over two TLS records. The position of the fragmentation can be chosen arbitrarily. For SNI-based censorship circumvention, fragmentations in and around the SNI extension are of special interest.

2.1. Censorship

Internet censorship is widely used by many countries and has been analyzed extensively [2], [4], [27], [44], [47], [51], [53], [56], [62]. While following similar objectives, countries deploy different approaches to limit Internet access for their residents. Some countries, such as China, maintain their own censorship infrastructure [78], while others delegate censorship to Internet Service Providers (ISPs) [79]. Depending on the specific implementation, Internet censorship can affect different services and protocols and exhibit varying complexity. Simple censors can block traffic to certain IPs or ports, risking unintended overblocking of services sharing the same IPs or ports. Advanced censors perform deep packet inspection (DPI) to extract destination information, such as the website's domain name or content from various application-layer protocols. For example, censors analyze HTTP [8], [32], DNS [32], [35], TLS [49], [81] and VPN protocols [80]. To block connections after a successful analysis, censors can drop packets [75], [79] or inject malicious packets such as false DNS responses [62] or TCP RSTs [18]. Censors also employ residual censorship, blocking any connection that follows a censored connection [10] and deploy partially redundant censorship implementations [6], [15], [32]. The diverse landscape of censorship and the closed-source implementations make its analysis challenging.

2.2. TLS Censorship

As the TLS protocol encrypts application data, censors have to rely on the information provided by the TLS protocol when censoring TLS connections. Since the SNI extension contains the server's domain and is unencrypted, censors can extract the domain from a `ClientHello` and determine whether to censor the handshake [10], [62], [79]. To censor TLS handshakes, the censors in China and Iran inject TCP RST packets and have been found to employ residual censorship on subsequent messages [10]. In China, two distinct middleboxes that analyze TLS handshakes have been identified [15]. In TLS handshakes, the `Certificate` message from the server also contains the server's domain. However, there are multiple drawbacks when analyzing the `Certificate` message. For example, it is encrypted in TLS 1.3, making it impossible to extract the server's domain for censors. Additionally, certificates can contain multiple hostnames, making it unclear which hostname the client tries to access. Since censors are not known to consider certificates, we focus on SNI censorship.

2.3. TLS Censorship Circumvention

To circumvent SNI-based censorship, a censor has to be prevented from extracting the server's domain name from the SNI extension. At the same time, the receiving TLS server still has to accept the TLS handshake. Censorship circumvention techniques that circumvent censors and are accepted by TLS servers can be achieved in two common ways. For example, a censor can be tricked into ignoring the TLS handshake altogether by desynchronizing the TCP state of the censor and the server [13]. A different approach is to prevent the censor from extracting the server's domain name from the SNI extension by modifying parts of the TLS handshake to obfuscate or hide the intended hostname. If the implementation of the TLS server is more lenient than that of the censor, these obfuscations can still be accepted by TLS servers. In the following, we review two known obfuscation techniques for TLS that were shown to be successful censorship circumvention techniques.

Domain Fronting. Domain fronting circumvents SNI-based censorship by placing a harmless domain name in the SNI extension. A censor that cannot decrypt the TLS connection sees only the harmless domain in the SNI extension. The server that receives and decrypts the TLS connection can choose the correct destination based on the domain name in the encrypted HTTP `Host` header. To provide the correct resource, the domain indicated in the `Host` header must be hosted on the server that receives the fronted request, making it especially interesting for servers that host a high number of domains, such as Content Delivery Networks (CDNs). This mismatch between the HTTP `Host` header and the SNI extension is not endorsed by the standards but is not explicitly forbidden [21]. The censorship circumvention community recognized domain fronting for censorship evasion [24], [65] and implemented it in various circumvention tools [31], [45], [55], [67], [68]. After initial usage, Amazon [1], [7], Cloudflare [55], and Google [3] disabled domain fronting on their servers, stating unintended usage of their services.

Fragmentation. To prevent censors from extracting the SNI extension from the `ClientHello` message, it can be fragmented over multiple TCP segments or TLS records. Splitting application layer data over multiple TCP segments is known as TCP fragmentation and is a widely successful circumvention technique [13]. More recently, TLS record fragmentation has been introduced as a censorship circumvention technique by Niere et al. [49]. They fragmented the `ClientHello` over multiple TLS records to circumvent censorship in China and determined considerable support by TLS servers. We achieve TLS record fragmentation by manually creating small TLS records and TCP fragmentation by constructing large TLS messages that coerce the underlying TCP socket to fragment the message.

3. Methodology

Previous research only sporadically considered TLS-level censorship circumvention techniques. As some of them were very effective in the past, the question arises whether more effective TLS-level circumvention techniques exist. To evaluate this, we performed the first systematic analysis of TLS-level opportunities to circumvent SNI-based censorship. We developed a five-phase methodology (Figure 3) to find bypass techniques that circumvent censors *and* are accepted by real-world TLS servers. In the first phase (**Design Phase**), we manually analyzed the TLS handshake for meaningful strategies that could potentially circumvent SNI-based censorship. In phase two (**Implementation Phase**), we then implemented our strategies in a tool called *Censor Scanner*. *Censor Scanner* can automatically test these strategies with endpoints to evaluate if the technique is successful with a given censor and accepted by a given TLS server. To this end, *Censor Scanner* exhaustively combines potential strategies up to a configurable degree t. By combining strategies, *Censor Scanner* is able to test advanced circumvention techniques that utilize detailed mismatches between the censor's and server's TLS parsing behavior. We then evaluated which strategies are accepted by TLS servers by running *Censor Scanner* against Apache, Nginx, and websites hosted on various CDNs (**Server Evaluation Phase**). Next, we evaluated the successful strategies from the previous phase against censors in China and Iran (**Censor Evaluation Phase**) by using *Censor Scanner* to initiate TLS connections from China and Iran to a vantage point behind the censor and recorded the censors' behavior. This allows us to develop a final list of strategies for circumventing SNI-based censorship in these countries. Lastly, we evaluate the final list of strategies against the top 10,000 servers of the Tranco list (**Tranco Evaluation Phase**) to assess their usability as censorship circumvention techniques further. Below, we detail each phase of our methodology.

3.1. Phase I: Design Phase

To bypass a censor, one has to find messages that are differently interpreted by the censor and the server that the user tries to reach. Specifically, the censor must think it should not block a given message while the intended server still understands and processes it. This is typically achieved by exploiting discrepancies in message parsing logic or by abusing the fact that censors only partially store the state of a given connection. To explore censors for potential bypasses, we manually created a list of potential TLS-level censorship circumvention strategies, which we present in the following paragraphs, answering **RQ1**.

3.1.1. Circumvention Identification. To generate ambiguous messages in the context of TLS SNI-based censorship, we focus on the first flight of the client, which usually only contains the `ClientHello` message, and systematically apply *manipulations* to it. With *manipulations*, we create uncommon flights that still adhere to the specification or

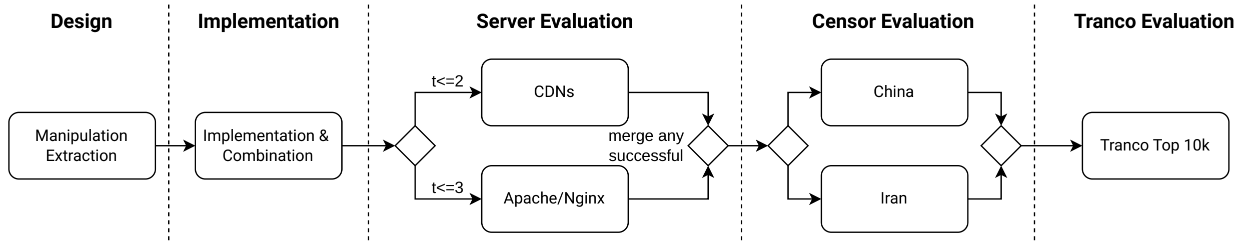

Figure 3. Overview of the methodology of this paper, separated into five phases. In the **Design Phase**, we extract possible manipulations of the TLS handshake. In the **Implementation Phase**, we implemented these manipulations into a tool that automatically combines manipulations exhaustively. Using our tool, we evaluated the acceptance of combined manipulations by local web servers and CDNs (**Server Evaluation Phase**). Next, we attempted to circumvent censorship in China and Iran with every manipulation combination accepted by at least one web server (**Censor Evaluation Phase**). Lastly, we evaluated the most successful circumvention techniques against the top 10,000 servers of the Tranco list (**Tranco Evaluation Phase**).

deviate from the specification slightly, such that they can still be potentially accepted by the server but cannot be correctly parsed by the censor anymore. Each manipulation adjusts a specific field in the message or performs structural changes to the flight, such as record fragmentation. While these manipulations can be potentially arbitrarily chosen, we curated a finite list of those that we deemed especially likely to be useful for censorship circumvention. For example, changing the *client random* in a `ClientHello` is unlikely to evade a censor as it is randomly populated in every benign TLS connection to begin with. Below, we iterate over common ideas that inspired some of our manipulations. We refer to Appendix A for a list of our specific manipulations.

TLS Misidentification. To perform SNI-based censoring, the censor must first identify that a given connection is using the TLS protocol. We presume that censors are doing this by looking specifically at the first few bytes of a TLS connection. By changing the beginning of a TLS connection (e.g., invalidating the type field in the record header, cf. Figure 1), we aim to prevent the censor from identifying our messages as TLS, and therefore from parsing them.

Changing Enums. TLS messages can contain a variety of constants. For instance, the TLS version field in the record header is often set to TLS 1.0 for middlebox compliance. By changing constants to other valid or invalid constants, we hope to surprise the censor with an uncommon choice, coercing it to potentially ignore the message when it does not implement the respective (potentially non-existent) feature.

Length Field Manipulation. Fields that are variable in length are preceded by a field that indicates their length in TLS. By changing the length field to an invalid value (e.g., a shorter length or an overlong length value), we potentially prevent the censor from successfully parsing the message. Intuitively, this should also prevent the intended server from parsing our messages. However, this is not necessarily the case as some length fields are redundant, and TLS servers are known to not always strictly parse these fields [63].

Fragmentation. Most TLS applications do not use TCP or TLS fragmentation, meaning censors can usually function without implementing these features. At the same time, fragmentation can prevent stateless censors from reassembling and analyzing our messages.

SNI Removal. Removing the hostname from the SNI extension or removing the SNI extension completely from the `ClientHello` message can be a powerful circumvention technique. The only possible countermeasure for censors is to block every `ClientHello` message without a SNI extension or to resort to an alternative censorship technique such as IP-based blocking. As censoring every `ClientHello` message without an SNI extension can lead to considerable overblocking, censors are inclined to allow TLS connections without an SNI extension.

Domain Fronting. Placing incorrect hostnames in the SNI extension prevents censors from determining the connection's true destination. While censors possess no countermeasure against this technique, the receiving server must accept TLS connections with a wrong hostname in the SNI extension.

Null Bytes. The SNI value is transmitted as a string. Since many programming languages terminate strings with a null byte internally, we consider the null byte particularly interesting as a malformed value. We use it in various locations to potentially confuse the string-parsing logic of the censor.

When designing our manipulations, we avoided ruling out too many modifications beforehand and let the evaluation decide which manipulations were effective. As we show in phase 2 (**Implementation Phase**), we fully automated our analysis, meaning that a re-execution with a different set of ideas for manipulations in future evaluations is easily possible.

3.2. Phase II: Implementation Phase

To implement our approach, we developed an open-source tool based on TLS-Attacker [70], called *Censor Scanner*. *Censor Scanner* implements all mentioned manipulations and can create TLS connections to a target server. To construct advanced circumvention techniques, *Censor Scanner* combines all implemented manipulations exhaustively up to a configurable strength parameter t. A value of $t = 1$ would mean that *Censor Scanner* would try to create a TLS connection with each possible manipulation once without performing any combinations. With a value of

$t = 2$, we would create a TLS connection for each tuple of possible manipulations. For example, adding a second SNI extension is a manipulation, and manipulating the hostname in either the first or the second SNI extension is a second manipulation. As done in this example, *Censor Scanner* also considers SNI extensions added by one manipulation in all further manipulations. Following these rules, *Censor Scanner* combines all defined manipulations exhaustively up to the given test strength t and applies each combination to a `ClientHello` flight. We call the set of manipulations applied to a given `ClientHello` message a *test vector*. This exhaustive combination of manipulations leads to exponential growth in combinations for increasing test strengths. We want to point out that our approach is deterministic. All combinations we created in our evaluations are reproducible using our tool *Censor Scanner*.

Exhaustively combining all manipulations leads to combinations with inapplicable manipulations. For example, manipulating the hostname in the SNI extension has no impact when combined with a manipulation that removes the SNI extension. *Censor Scanner* also does not combine manipulations of the same type twice. This means that we do not apply two types of record fragmentation simultaneously or manipulate a length field with two different values. To save redundant executions, we explicitly filter out such combinations during the combination phase wherever possible.

3.3. Phase III: Server Evaluation Phase

In this work, we aim to determine *working* censorship circumvention techniques on the TLS layer. To achieve this, it does not suffice to prevent censors from blocking our test vectors: the server the user is trying to reach must still understand and process the sent messages. If a given test vector causes the server to not understand the message anymore, it is unfit for censorship circumvention. To evaluate servers' acceptance of our manipulations and to answer our **RQ2**, we tested each combination of manipulations against popular web servers. Specifically, we used *Censor Scanner* to apply each combination to a TLS handshake with each web server we evaluated. As popular web server targets, we chose a mix of web servers that we can analyze locally and websites served by popular CDNs to estimate real-world acceptance. For the local web servers, we chose the six Apache and Nginx versions that are obtained when running the respective *apt install* command on the Ubuntu versions 18.04, 20.04, 22.04, 23.04, 23.10, and 24.04. As popular CDNs to scan, we chose Akamai, Amazon, Cloudflare, Fastly, and Google. To observe possible differences in SNI parsing, we chose two different public websites that are hosted on the respective CDN. The first website shared its IP address with other websites, while the second website did not. Our motivation behind this setup was that websites sharing their IP addresses with others might enforce stricter SNI parsing, possibly affecting techniques such as domain fronting, which is highly relevant for censorship circumvention [24]. We classified our techniques as successful if the target server responds with the desired HTTP page we requested in the HTTP `Host` header. A complete list of websites we evaluated for each CDN can be found in Section B. We evaluated the local servers with a test strength $t = 3$. To not overload real websites with our scans, we set the test strength of our CDN scans to $t = 2$. We conducted our evaluation of servers from a virtual machine from the DFN[2].

3.4. Phase IV: Censor Evaluation Phase

To evaluate the effectiveness of our manipulations against censors and to answer our **RQ3**, we attempted to circumvent two censors with each manipulation combination that was successful against any web server from phase three. For the evaluation, we chose to test *Censor Scanner* against China and Iran. We analyzed China's Great Firewall of China (GFW) because it is one of the most sophisticated censors, and its use of SNI-based censorship is well-documented. We complemented our analysis of the GFW with an analysis of Iran's censors to collect evidence on the extent to which the discovered techniques transfer to other censors. To trigger the firewalls in China and Iran, we rented vantage points in both countries and sent manipulated `ClientHello` flights to a controlled vantage point in Germany. Specifications of our used vantage points are given in Appendix C. We considered a test vector successful if the `ClientHello` message it was applied to was not intercepted by the censor and a response from the destination server could be received. If the expected answer has not been received, we automatically analyzed all received packets of the connection, such as injected TCP RST packets, timeouts, or TLS alerts, and classify the reason why the connection did not succeed.

To compensate for network irregularities, we tested each test vector three times against each censor and recorded the result of the connection. If the same result was reached in at least $2/3$ of the connections, the connection result was assumed for all connections; otherwise, the test vector was re-executed until the same connection result was reached for at least $2/3$ of all executed connections. We configured *Censor Scanner* to try at most 20 times, after that we considered the test vector to be *undecided*. Notably, we counted timeouts as separate connection results and allowed a maximum of 30 timeouts in addition to the 20 connection attempts. We did so to compensate for network downtimes on our vantage points.

3.5. Phase V: Tranco Evaluation Phase

As a last step, we evaluated successful circumvention techniques from phases three and four on the top 10,000 domains from the Tranco list [54]. After excluding domains that were unresolvable, did not support TLS, or requested exclusion from our scans during previous analyses, we could evaluate 6,739 servers.

2. German National Research and Education Network, https://www.dfn.de/

4. Server Evaluation Results

To answer **RQ2**, we performed acceptance scans with seven distinct web servers (cf. Server Evaluation phase, Section 3) in February and March 2024. In total, we evaluated 2,926,259 test vectors; 10,578 of these test vectors were accepted by at least one web server. We consider a test vector accepted if it connects to the same website as an unmodified request. Table 1 depicts the number of test vectors accepted by different web servers ordered by their test strength. For $t = 1$ and $t = 2$, 53.35% and 48.56% of test vectors were accepted by at least one web server, respectively.

In our local analyses, we evaluated Nginx and Apache default versions on six Ubuntu versions. Regardless of the Ubuntu version, both implementations yielded the same successful test vectors (cf. Table 1). Both servers employ the strictest implementation; they accepted 24 test vectors with test strength $t = 1$. We attribute this to OpenSSL being used by both Apache and Nginx, and security backports applied to Apache, Nginx, and OpenSSL in Ubuntu's LTS versions.

Our CDN evaluations showed that every implementation accepted a unique number of test vectors. This stems from unique parsers and other differences in the CDN's libraries. For instance, Amazon and Akamai's implementations do not validate the maximum TLS record size of 2^{14} bytes [20]. Fastly and Google validate the record size, while Cloudflare accepts only smaller TLS records. Other particularities are the acceptance of a second SNI extension by Akamai's implementation and a wrong certificate returned by Fastly in TLS 1.3 connections with invalid NameType or TLS version fields in the `ClientHello` message. In contrast to Apache and Nginx, CDNs parse the SNI extension less strictly. The most lenient implementation—Amazon's s2n—accepted 85 of these test vectors and over 86% of all successful test vectors.

Our results reveal differences in the leniency of web servers regarding our test vectors. These differences imply fundamentally different behavior of common open-source server implementations and commonly used CDNs, underlining the importance of analyzing live web servers in conjunction with offline implementations. At the same time, the high overlap of successful test vectors between servers indicates possible circumvention techniques that can be used on multiple servers. Overall, our evaluation yielded numerous test vectors accepted by web servers which we could use in our following evaluation of censors.

Placeholder Server Certificates. Some test vectors elicited placeholder certificates from web servers during the TLS handshake. Affected servers would send a placeholder certificate valid for some CDN-specific domain such as `j.sni-644-default.ssl.fastly.net`, finish the TLS handshake successfully, and respond with the HTTP page indicated by the encrypted HTTP `Host` header. Table 2 depicts the test vectors and web server combinations that trigger a placeholder certificate. We mainly triggered placeholder certificates with test vectors that remove or obfuscate the hostname in the SNI extension, such as SNI removal, domain fronting, or byte injections into the SNI hostname. By removing or obfuscating the hostname, these techniques prevent servers that host multiple domains on a shared IP address from providing the correct certificate for the requested hostname. Servers that host only a single domain on an IP address can default to the correct certificate. As such, our results show that Apache, Nginx, Fastly, Google, and Akamai opt to send a placeholder certificate for domains on shared IP addresses. For domains hosted on a single IP address all scanned servers defaulted to the correct certificate in many cases. While placeholder certificates prevent validating the resource's authenticity, the correct HTTP page is accessible. We consider test vectors that trigger placeholder certificates, such as domain fronting, viable censorship circumvention techniques. We also highlight the need for censorship research to consider server deployment when evaluating servers' acceptance of circumvention techniques.

Table 1. VECTOR ACCEPTANCE

Test Strength	$t = 1$	$t = 2$	$t = 3$[a]	$\sum_t$
Vectors	169	18,972	2,907,118	2,926,259
Web Servers				
Apache	24	239	1,275	1,538
Nginx	24	239	1,275	1,538
Cloudflare	28	362	–	390
Fastly	38	606	–	664
Google	42	686	–	728
Akamai	52	865	–	917
Amazon	85	9,037	–	9,122
Any Server	90 (53.35%)	9,213 (48.56%)	1,275	10,578

Number of successful test vectors, ordered by test strength. A test vector counts as successful when the web server returns a correct certificate and the correct website. For CDNs, a test vector counts as successful if it was successful in either deployment (with shared IP or dedicated IP). We attribute the same results of Nginx and Apache to using OpenSSL with backported fixes.
[a] $t = 3$ was only executed on local web servers.

5. Analysis of Middleboxes

After determining the set of test vectors accepted by at least one web server (Censor Evaluation phase, cf. Table 1), we evaluated which test vectors in this set successfully circumvent censorship in China and Iran. We executed our censorship scans in March, April, and May 2024. During our scans, we encountered multiple middleboxes in China and one middlebox in Iran. In this section, we describe the behavior of these middleboxes before showcasing the success of specific test vectors in Section 6.

5.1. China

For our initial evaluations in China, we configured *Censor Scanner* to place wikipedia.org in the SNI extension, triggering censorship. Part of the censorship we encountered in China was residual censorship of up to six minutes on the triple (source IP, destination IP, destination port). Accordingly, we evaluated our test vectors across multiple ports and did not reuse a censored destination port for seven minutes.

Table 2. TEST VECTORS TRIGGERING PLACEHOLDER CERTIFICATES

	TLS 1.2 + TLS 1.3								TLS 1.3	
	RemoveSNI	ExtensionBytes	AdditionalSNI	ListLength	NameType	Domain Fronting	InjectAscii	InjectUnicode	MessageVersion	NameType
Shared IP										
Apache	○	○	–	–	–	–	–	–	–	–
Nginx	○	○	–	–	–	○	○	○	–	–
Fastly	○	○	–	–	–	○	○	○	○	○
Google	○	○	–	–	–	○	○	○	●	–
Akamai	○	○	○	○	○	○	○	○	–	–
Amazon	–	–	–	●	–	–	–	–	●	–
Cloudflare	–	–	–	–	–	–	–	–	●	–
Dedicated IP										
Fastly	●	●	–	–	–	●	●	●	–	–
Google	●	●	–	–	–	●	●	●	●	–
Akamai	●	●	–	–	–	●	●	●	–	–
Amazon	●	●	–	●	●	●	●	●	●	–
Cloudflare	●	●	–	–	–	–	–	●	●	–

Excerpt of resulting minimal test vectors that triggered the correct HTTP page and a correct or incorrect TLS certificate on at least one server hosting either one or multiple domains on an IP address. We determined placeholder certificates in a shared IP setting on Apache, Nginx, Fastly, Google, and Akamai.
●: correct certificate and HTTP page
○: incorrect certificate but correct HTTP page
–: incorrect HTTP page or aborted handshake

When we evaluated how exactly the GFW interfered with our requests, we noticed different blocking behaviors depending on the exact test vector we used. In particular, we would measure distinct censorship behaviors by the GFW for exemplary circumvention strategies A and B with $t = 1$ while their combination with $t = 2$ circumvented the GFW. Through manual analyses, we discovered three distinct ways the GFW was censoring our connection: we saw a single RST, a single RST+ACK, or three RST+ACK packets. The GFW often used multiple blocking techniques simultaneously, resulting in the recorded behavior being a combination of the distinct techniques. The combination of techniques we encountered was different for subsequent executions of the same request, resulting in challenging non-deterministic blocking behavior. We concluded that there is likely not a single censorship device but three different devices that censor connections independently. In our subsequent analyses, we noticed that not all domains were censored by all of the middleboxes. A similar behavior was reported by Bock et al. [15] and Wang et al. [74], who discovered two different censorship behaviors. We could map these behaviors to the first two detected devices (MB-1 and MB-2). However, the last middlebox (we call it MB-New) was not publicly known or documented in the censorship literature.

We asserted that all three middleboxes are part of the GFW and not inserted at the ISP level by sending a censored message with increasing TTL values while waiting for the censorship to trigger. All middleboxes started triggering at the same TTL, indicating that the censoring happens at a single position in the network. Figure 4 depicts how all three middleboxes inject one or multiple RST packets directly after the `ClientHello` message to tear down connections and potentially trigger simultaneously. TCP reassembly and residual censorship are employed differently by all three middleboxes in China, which we believe hindered the analysis of both properties in recent years.

To avoid non-determinism during the evaluation and isolate the middleboxes, we adapted our scanning approach by selecting an initial `ClientHello` to base our manipulations on that only gets censored by one of the middleboxes at a time. For instance, we found that placing wikipedia.org in the SNI extension and increasing the TLS version in the `ClientHello` to `0x0304` isolated MB-1. For MB-2, we placed freetibet.org in the SNI extension and set the `ExtensionsLength` field to 0. For the isolation of MB-New, we used freetibet.org in the SNI extension and set the `NameType` field in the SNI extension to an invalid value. For the full evaluation, we scanned each middlebox individually in a separate scan.

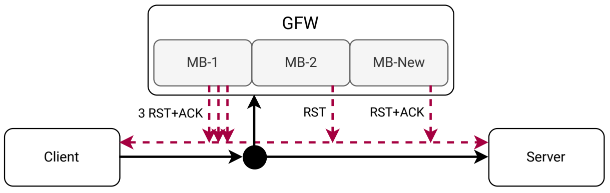

Figure 4. The GFW consists of three distinct middleboxes. Notably, all three middleboxes inject a different pattern of TCP RST packets and can trigger simultaneously.

To characterize the detected middleboxes further, we performed additional manual tests. In the following, we detail specific behaviors for each middlebox.

MB-1. MB-1 injects three indistinguishable RST/ACK packets into the connection which is the behavior of the GFW predominantly mentioned in analyses of the GFW [15], [34], [74]. It also censors residually by injecting a single RST/ACK packet for roughly 90 seconds which has also been reported in previous evaluations for China's SNI and HTTP censorship [10], [12], [17], [34]. MB-1 can reassemble TCP segments up to a coalesced length of 10,240 ($2^{10} * 10$) bytes. However, MB-1's reassembly sporadically fails, which we suspect might be due to limited buffers in the middlebox. When parsing TLS, MB-1 relies on both version fields, the Extension Length, the SNI Name Length, and the SNI NameType; it ignores the Extensions Length and the List Length. Invalidating either of these fields prevents MB-1 from blocking the request. It interprets the last SNI extension when multiple SNI extensions are present and filters out injected records before the `ClientHello` message. As MB-1 most closely resembles the behavior of the GFW discussed in previous works [15], [34], [74], we assume it to be the primary middlebox of the GFW.

MB-2. MB-2 behaves similarly to the middlebox detected by Bock et al. [15] and Wang et al. [74]. It injects a single RST packet with an unset ACK flag and does not employ residual censorship. Like Bock et al., we could only determine domains blocked by both MB-1 and MB-2. In contrast to the analysis of Bock et al., we observed that it injects its RST packet directly after the `ClientHello` packet and not after the `ClientKeyExchange` message. This discrepancy may be due to a software update. As MB-1, MB-2 also reassembles TCP fragments, but it does so differently. It reassembles TCP segments if the segmentation occurs in the TLS record or TLS message header (i.e., in the first 10 bytes) or if the segmentation occurs after byte 512 (2^9). We suspect that MB-2 reassembles the header bytes to classify the protocol of the message and assumes all legitimately fragmented messages to be over 520 bytes long. The TCP reassembly of MB-2 is similarly unreliable to that of MB-1 and fails sporadically. Similarly to MB-1, MB-2 also relies on both version fields and the SNI Name length; it parses the Extensions Length field but ignores the Extension Length field of individual extensions and the List Length field of the SNI extension. It is also the only middlebox that censors invalid SNI extensions without a single list entry—effectively censoring without the presence of a hostname.

MB-New. With MB-New, we present a previously undescribed middlebox of the GFW. This middlebox injects a single RST/ACK packet and employs residual censorship of up to 360 seconds by null routing subsequent packets. In contrast to MB-1 and MB-2, MB-New does not reassemble TCP segments and censors different domains than the other two middleboxes. It parses the Extensions Length field, which summarizes the length of all extensions, and the length field of each extension (cf. Figure 1), but ignores the length fields in the TLS record and TLS message headers. It ignores all values from the SNI extension except for the actual hostname and infers the position of the hostname using the Extension Length field, assuming a single SNI entry. For example, MB-New can successfully block a request with an invalid List Length field but fails to block a request with an invalid Extension Length field. It also does not parse the TLS version of the message header only the first two—major—bytes of the TLS version in the record header. In contrast to MB-1 and MB-2, which parse more header bytes, MB-New relies only on the first two bytes of the TLS connection (`0x1603`) to determine a TLS `ClientHello`. Its permissive parser makes it more resilient against circumvention attempts but might lead it to parse more connections that are not TLS in the first place. As MB-New only triggers on a few domains, it is sufficient to circumvent MB-1 and MB-2 for most domains. We do not know whether MB-New has been installed as a dedicated backup censor for certain domains or resembles internal structures of the GFW's organization. We also do not know if MB-New was only installed recently or has only been overlooked so far. The only reference to the behavior of MB-New we could find was made by Hoang et al. [34] in 2024. While they detected MB-New's unusually long residual censorship, they attributed it to specific domains rather than a separate middlebox.

Table 3. SUCCESSFUL TEST VECTORS

Test Strength	$t = 1$	$t = 2$	$t = 3$	$\sum_t$
Vectors	159	17,382	2,733,298	2,750,839
Iran	61	8,537	1,089	9,687
China	46	7,354	901	8,301
MB-1 [74]	58	8,102	1,097	9,257
MB-2 [15]	58	7,960	1,055	9,073
MB-New	57	8,703	929	9,689
Any Server	67 (42.14%)	9,020 (51.89%)	1,106	10,193

Number of test vectors that circumvented censors and were accepted by at least one web server, ordered by test strength. This table shows that the combination of middleboxes in China is harder to circumvent than the censor in Iran.

5.2. Iran

The middlebox we encountered in Iran censors connections by injecting a single RST/ACK packet into the connection after the `ClientHello` message. It does not reassemble TCP segments. In contrast to previous research, we also did not detect residual censorship [10] in Iran. Iran's middlebox expects two valid TLS version fields and parses the Extensions Length and SNI Name Length fields to determine the location of the SNI. This makes its implementation somewhat similar to that of MB-2 of the GFW.

6. Analysis of Circumvention Techniques

Our evaluation of TLS servers and censors yielded successful strategies for the middleboxes in China and Iran. From the 10,193 test vectors accepted by at least one combination of server and censor (cf. Table 3), we generalized the techniques depicted in Table 4. To reduce the number of depicted vectors, we left out vectors with $t > 1$, yielding the same or worse results as a smaller vector with $t' < t$. For example, we do not display vectors that change the TLS version in the record and message header, as the singular approaches are more successful. Notably, minimal combinations with $t = 2$ exist, such as injecting a CCS message into a TLS 1.3 connection. Besides minimizing test vectors for better result presentation, we merged different test vectors of the same circumvention type that did not show different behavior among each other. For instance, we merged the injection of various ASCII characters into the *ASCII-Letter* technique. The execution of fine-grained test vectors was still beneficial during our evaluations as it provided more detail about the servers' and censors' implementations. Due to space constraints, Table 4 only contains a selection of the most successful circumvention techniques; we plan to publish the extended version of Table 4 after the review process.

6.1. Combining Vectors

Each test vector depicted in Table 4 circumvents censorship of at least one middlebox and is accepted by at least one web server. To successfully circumvent censorship in Iran, it is sufficient to use a technique that circumvents its only censor and is accepted by the required web server; to successfully circumvent the GFW, the process is

Table 4. SELECTION OF SUCCESSFUL TEST VECTORS

Technique	Local Servers		Akamai		Amazon		Cloudflare		Fastly		Google		Censors				Tranco[‡]
	Apache	Nginx	■	■■	■	■■	■	■■	■	■■	■	■■	MB-1	MB-2	MB-New	IR	Top 10k
Version Field																	
Message Header: TLS 1.3 ‖ Invalid Higher	●	●	●	●	●	●	●	●	●	●	●	●	●	●	–	●	97.36%
*Record Header: TLS 1.3	●	●	●	●	●	●	●	●	●	●	●	●	●	●	–	●	99.11%
*Record Header: SSLv2	–	–	–	–	●	●	–	–	–	–	–	–	–	●	●	●	18.50%
*Record Header: Invalid	●	–	–	–	–	●	●	–	–	–	–	–	–	●	●	●	23.45%
Record Fragmentation																	
*In Message Body	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	92.80%
*In Message Header	●	●	●	–	●	●	●	●	●	●	●	●	●	●	●	●	86.66%
Record Injection																	
*In CH: Alert Incomplete[†]	–	–	–	●	●	●	–	–	–	–	–	–	●	●	●	●	21.32%
*In CH: Internal Warn Alert[†]	–	–	–	●	–	–	●	●	●	–	●	●	●	●	●	●	41.04%
*In CH: Heartbeat Any ‖Invalid[†]	–	–	–	–	●	●	–	–	–	–	–	–	●	●	●	●	17.20%
*Before CH: Internal Warn Alert	–	–	–	–	–	–	–	–	●	–	●	●	–	●	●	●	9.42%
*Anywhere: CCS + TLS 1.3	–	–	–	–	–	–	–	–	–	●	–	–	●	●	●	●	4.63%
TCP Fragmentation																	
Additional List Entries	–	–	–	–	●	●	–	–	–	–	–	–	●	●	●	●	16.22%
Additional Cipher Suites Max Possible Record Size	●	●	●	●	●	●	–	–	–	–	–	–	●	●	●	●	47.34%
Additional Cipher Suites Max Defined Record Size	●	●	●	●	●	●	–	–	●	●	●	–	●	●	●	●	71.05%
Padding Extension Max Possible Record Size	●	●	●	●	●	●	–	–	–	–	–	–	●	●	●	●	48.23%
Padding Extension Max Defined Record Size	●	●	●	●	●	●	–	–	●	●	●	●	●	–	●	●	71.87%
Hostname Injection																	
Null Byte after Hostname	–	–	–	–	●	●	–	–	–	–	–	–	–	●	●	●	22.05%
Null Byte in/before Hostname	–	–	–	–	●	–	–	–	–	–	–	–	●	●	●	●	11.28%
ASCII-Letter / Domain Fronting	○	–	●	○	●	–	–	–	●	○	●	○	●	●	●	●	34.96%
Non-ASCII	○	–	●	○	●	–	●	–	●	○	●	○	●	●	●	●	37.08%
SNI Hiding																	
No SNI Extension	○	○	●	○	●	–	●	–	●	○	●	○	●	●	●	●	38.08%
Invalid Extension Bytes	○	○	●	○	●	–	●	–	●	○	●	○	●	●	●	●	38.28%
No SNI List Entry	–	–	–	–	●	–	–	–	–	–	–	–	●	–	●	●	6.63%
SNI Last Extension + Short Extensions Length	–	–	–	–	●	–	–	–	–	–	–	–	–	●	●	●	7.26%

Selection of circumvention techniques that circumvented at least one censor and were accepted by at least one web server. The results are ordered according to the phases presented in Figure 3. Section 6.1 interprets the circumvention techniques presented in this table. This table highlights fragmentation techniques, version field manipulations, and SNI removal as very effective for censorship evasion.
●: correct certificate and HTTP page ○: incorrect certificate but correct HTTP page –: incorrect HTTP page or aborted handshake ■/ ■■: Server hosts one/multiple domains
[*] These techniques can be applied by a MITM tool as they do not invalidate the TLS handshake.
[†] Injected record after fragmentation
[‡] We evaluated 6,739 reachable servers out of 10,000 and counted a vector as successful for a server if any of its instantiations finished the TLS handshake and responded with the same HTTP content as in a default TLS handshake.

more complicated. As all three middleboxes of the GFW can censor a connection, a test vector must circumvent all active middleboxes for a given domain to bypass the GFW as a whole. Since a user typically does not know which middleboxes will be used to censor a given domain, techniques that bypass all three middleboxes simultaneously, such as TLS record fragmentation, are especially desirable. Alternatively, one can combine techniques whose union circumvents all three middleboxes. Notably, web server acceptance for this combination is only the intersection of web server acceptance for the singular techniques. For instance, setting the `Extensions Length` to 0 and invalidating the `NameType` circumvents all middleboxes but is only accepted by the instance of Amazon that hosts one domain.

6.2. Technique Categories

We group successful techniques into categories as visible in Table 4. Each category encompasses techniques that alter similar parts of the `ClientHello` message or are otherwise comparable. We discuss the most successful categories and their effectiveness below.

Version Field. Techniques in this category alter the TLS version in the TLS message header or the TLS record header. Note that these techniques only alter the respective header fields; TLS 1.2 is still used for the handshake. Servers accepted all message header alterations that set the TLS version higher than TLS 1.2. This aligns with the version negotiation mechanism of TLS 1.2 described in Appendix E of RFC 5246 [20] in which servers expect clients to send potentially higher versions than TLS 1.2. This message header alteration is accepted by all censors except MB-New. Similarly, all censors except MB-New can be bypassed by setting the TLS version in the record header to TLS 1.3, which is also accepted by all servers. Setting the TLS version in the record header to SSL 2.0 or invalidating it circumvents MB-New, but only Amazon's web servers accept these changes.

We evaluated the implementations of BoringSSL by Google [28] and OpenSSL [52] used in Apache and Nginx. We discovered that they validate only the first—the major—byte of the TLS version (`0x03`) in the record header. This leads them to allow TLS 1.3 (`0x0304`) but disallow an invalid header such as `0x2020`. As other web servers and MB-New exhibit similar patterns, we suspect them to validate this version field similarly. Overall, changes to the TLS version in the record and message header were successful on all servers and all censors except MB-New. As MB-New operates on only a few domains, we consider alterations of the TLS version fields to be highly successful and usable for censorship circumvention.

Record Fragmentation. Introduced as a circumvention technique by Niere et al. [49], we affirmate TLS record fragmentation as a highly successful circumvention technique. It circumvented all servers in our analyses and was accepted by all TLS servers. Only one of Akamai's web servers did not accept TLS record fragmentation when it occurred in the message header. In contrast to most other techniques presented in this paper, TLS record fragmentation fully complies with the TLS specification [20]. It can also become necessary as TLS handshake messages are potentially larger ($2^{24}-1$ bytes) than TLS records ($2^{16}-1$ bytes). This leads to an overall acceptance by servers. We cannot say why TLS record fragmentation circumvents censors so well as it is conceptually similar to TCP fragmentation. While censors seem to start reassembling TCP fragments (cf. Paragraph TCP Fragmentation), TLS record fragmentation is not yet considered. Therefore, TLS record fragmentation found usage in circumvention tools [38], [77]. Our results confirm the viability of TLS record fragmentation as a censorship circumvention technique.

Record Injection. The techniques in this category inject an additional TLS record into the connection. Some of these techniques inject an additional record between two records of an already fragmented `ClientHello` message. While TLS record fragmentation currently suffices to circumvent censors, we conjecture this combination to complicate future record reassembly by censors further. Every server except the one at Akamai with one hosted domain and the one at Fastly with multiple hosted domains supported at least one combined record fragmentation and injection technique. Besides these combined techniques, we also discovered that injecting an internal warning alert before the `ClientHello` message circumvents censorship and was accepted by some servers. Xue et al. [81] proposed a similar circumvention technique for Russia's TSPU Throttling by injecting a `ChangeCipherSpec` message before the `ClientHello` message. However, contrary to injecting a warning alert, only a single web server accepted TLS 1.3 handshakes starting with injected `ChangeCipherSpec` messages, making it unfit for censorship evasion. Our results show that record injections provide an additional avenue for censorship circumvention, especially in combination with TLS record fragmentation.

TCP Fragmentation. While we do not directly manipulate the TCP-layer in this paper, we could nevertheless trigger TCP fragmentation with alterations on the TLS layer. Specifically, we added SNI list entries, cipher suites, or a padding extension to artificially increase the size of the `ClientHello` message. In Section 5.1, we detailed that MB-1 and MB-2 reassemble TCP fragments. Despite technical progress by censors, we could still circumvent them with TCP fragmentation using large messages indicating limited reassembly capabilities by the middleboxes. While TCP fragmentation reliably circumvented censors, around 47% of servers do not reassemble TLS records that exceeded the maximum defined size of 2^{14} bytes. Cloudflare even rejects large TLS records that still fall in the RFC-compliant size of at most 2^{14} bytes: Cloudflare finishes the TLS handshake but responds with an HTTP error afterward. Most other servers accepted large TLS records smaller than 2^{14} bytes.

Hostname Injection. The techniques in this category inject symbols into the hostname or replace the hostname altogether through domain fronting. Domain fronting or injecting ASCII and non-ASCII symbols circumvents any censor. These techniques were also widely accepted by servers; servers that host multiple domains often responded with an incorrect placeholder certificate (c.f. Section 4).

SNI Hiding. Some techniques remove or hide the SNI extension or a part of it. For instance, one technique removes the SNI extension entirely, while another disguises it as a different extension. Both techniques bypass all evaluated censors and are widely accepted by servers; placeholder certificates were often sent by servers that host multiple domains (c.f. Section 4). We suspect censors ignore these requests as all major browsers send an SNI extension by default; blocking all TLS requests without an SNI extension could lead to overblocking.

6.3. Result Confirmation

Tranco Evaluation. To further determine servers' acceptance of the test vectors depicted in Table 4, we evaluated their acceptance by the top 10,000 servers from the Tranco list. Our results show that general server acceptance of our test vectors aligns with their acceptance by the CDN servers we evaluated in the third phase. This affirms various successful circumvention techniques such as TLS record and TCP fragmentation, increasing the TLS version fields, and domain fronting.

Censorship Circumvention. We manually confirmed the success of two techniques presented in Table 4: TLS record fragmentation and setting the TLS version in the record header to TLS 1.3. Using TLS record fragmentation, we accessed wikipedia.org/wiki/Water censored by all three middleboxes. By setting the TLS version in the record header to TLS 1.3 we accessed en.wikipedia.org/wiki/Water censored by MB-1 and MB-2. We used curl [64] to access the websites and extended DPYProxy [39], an open-source censorship circumvention tool, to alter the TLS record headers. To circumvent the GFW's IP censorship, we routed our connection over a proxy server hosted at a vantage point at our university. We plan to provide the complete setup in our artifact evaluation and create a pull request extending DPYProxy after peer review.

7. Discussion

Successful Censorship Circumvention Techniques. We identify four techniques as most successful: Increasing the TLS version fields, fragmenting TLS records, fragmenting TCP segments, and omitting the SNI extension. We also project that all four techniques will stay viable in the future. The TLS version fields are used by censors to fingerprint the

TLS protocol at the beginning of the handshake. Censors that attempt to reassemble TCP segments or TLS records need complex implementations that hold state. Lastly, omitting the SNI extension from a TLS handshake forces censors to allow the connection, block all TLS connections without an SNI extension, or fallback to IP-based blocking, leading to possible overblocking. Besides these generally successful techniques, we project other techniques to be useful for accessing websites or services with known server implementations.

Integration in Circumvention Tools. In contrast to lower-layer censorship evasion techniques, TLS-layer techniques cannot always be added to an application in hindsight. As soon as a technique modifies the contents of the `ClientHello`, the application has to account for the modification inside the `Finished` messages as the modifications influence the handshake transcript. Therefore, to use a handshake-layer technique, the client's TLS implementation has to be aware of it. In contrast, changes in the record header or the record fragmentation do not influence the cryptographic computations and can, therefore, be added without cooperation from the application. We marked such techniques with a * in Table 4. Alternatively, TLS-based techniques can be used by integrating them as Malicious-in-the-Middle (MitM) between the application and the censor. This necessitates the client to trust a TLS certificate possessed by the circumvention tool to possess a TLS certificate trusted by the client. It is also possible to modify the TLS libraries used by existing software, such as browsers, with `LD_PRELOAD` to apply our techniques [26]. Circumvention tools that already execute their own TLS handshake [68], [77] or browsers can directly implement our discovered techniques.

Benefit of Exhaustive Combinations. The benefit of our combinatorial approach was twofold. We found combined circumvention techniques such as a TLS 1.3 connection with an altered TLS version in the TLS message header. It also helped us to identify the distinct middleboxes of the GFW. When two manipulations triggered different censorship behaviors and their combination circumvented the GFW, we encountered distinct middleboxes. Other frameworks, such as Geneva [13], also yield combinations of manipulations as circumvention techniques. Our approach differs by combining manipulations exhaustively. We argue that this helped us to identify and analyze the three distinct middleboxes of the GFW more easily.

7.1. Limitations and Future Work

Censorship Analysis. We focused our evaluations in Section 5 on the first flight of the TLS handshake, as we only encountered censorship that directly followed the `ClientHello` message in a preliminary scan. Specifically, we did not encounter the deeper TLS-level censorship that was detected by Bock et al. [15] as part of an additional middlebox of the GFW in 2021. While we believe that this middlebox has since changed its behavior (cf. Section 5.1), it is possible that we ignored sporadic TLS censorship that occurs after the servers' `ServerHello` message. Additionally, our scans assume that the behavior of the censor does not change meaningfully *during* our evaluations. We assume that we did not encounter such behavior during our evaluations, as our results were reproducible across two scans. Nevertheless, we recommend future research to use control probes during their measurements to ensure consistent results.

Generalizing Results. We used only one vantage point in Germany to which we sent our TLS `ClientHello` messages: we could have missed SNI censorship that is only triggered in connections to specific IP addresses. We also conducted our evaluations from a single vantage point in China and Iran, which means that the specific techniques presented in this paper might not be generalized to other vantage points in the same or other countries. Similarly, censorship in residential networks might not be equal to the censorship we detected at the rented vantage points. However, we want to emphasize that our methodology—and *Censor Scanner* in particular—can be directly applied to find such techniques from other vantage points. Future work could integrate *Censor Scanner* into evasion tools, such that it automatically finds successful circumvention techniques itself. An application could then dynamically query *Censor Scanner* whenever it needs a new successful circumvention technique.

Censor's Ability to Update Their Implementation. Censors can potentially update their implementations to accommodate the circumvention techniques described in this paper. We consider such amends possible for some techniques: for instance, we believe that detecting hostnames followed by null bytes and considering TLS messages with invalid version fields is feasible for a censor. Other techniques require more complex changes for a censor: reassembling TCP segments or TLS records requires censors to hold state, acquire memory space, and implement mechanisms for freeing memory after analyzing a connection. While we can only make assumptions about the costs for a censor such as the GFW, we argue that they are considerable and potentially deemed too costly. As evidence, we see the GFW's continuing inability to reassemble TCP segments at least 12 years after they first attempted reassembly [40].

Domain Fronting. In this paper, we provide a limited evaluation of domain fronting on CDNs. While we identified effective domain fronting opportunities on singular servers, the same techniques must not necessarily apply to other servers of the same CDN as we did not control server-specific configurations on the CDNs we evaluated. We suggest an extensive analysis of domain fronting opportunities on different CDNs and their usage in censorship circumvention for future work.

TCP Fragmentation. While TCP fragmentation is widely used for censorship circumvention, we saw mixed effectiveness in China. Parts of the GFW can reassemble TCP segments, and while we could overwhelm their reassembly algorithms with large messages, these messages were not accepted by all TLS servers. We conjecture that TCP fragmentation continues to be a vital technique for censorship circumvention in the future but suggest a detailed analysis of censors' TCP reassembly algorithms and servers' acceptance of large messages in future work.

Other protocols. In this work, we consciously manipulated *only* the TLS layer to argue about the effectiveness of this approach meaningfully. Even with this constraint, some TLS layer manipulations forced changes on the TCP layer, such as TCP fragmentation. We presume that censorship circumvention techniques that combine manipulations on multiple network layers, such as TLS and TCP, will be even more effective. Therefore, we propose future work to evaluate the effectiveness of these techniques. Last but not least, similar to TLS, censorship circumvention techniques for TLS-like protocols such as DTLS [58] or QUIC [37] have not been explored yet. We consider QUIC to be especially promising since it combines the features of TLS with its own transport layer mechanisms.

7.2. Ethical Considerations

We considered the impact of our analyses on real-world infrastructure and people. We evaluated Apache and Nginx locally and did not impact the Internet infrastructure. In our evaluations of CDNs, we analyzed two large websites per CDN. Each website was subject to under 20.000 TLS handshakes distributed over multiple hours. We consider that amount of traffic negligible for the large CDN providers we evaluated. After encountering rate limits on Cloudflare, we adjusted our throughput accordingly. During our scan of the Tranco Top 10k server, we sent less than 100 messages to each server and excluded servers that requested exclusion in previous scans. The machines from which we executed our scans were identifiable as such through their DNS records and an HTTP landing page. We rented our vantage points in adherence to the applying sanctions list and export regulations by the European Union [23].

To evaluate censors in China and Iran, we rented a vantage point in each country similar to previous research [13], [34], [42]. From these vantage points, we sent all generated traffic to an additional vantage point we rented in Germany. Thus, our evaluation of censors only inflicted traffic on intermediate network operators used to large amounts of network traffic.

We strongly believe that censorship circumvention research provides more benefits to researchers and affected people than to censors. While censors could adjust their censorship to the techniques we evaluated, this inflicts a resource cost on the censor (cf. Section 7). We also discovered previously unknown techniques that directly benefit the censorship circumvention community. Lastly, structural insights about censors provide avenues for future circumvention research.

8. Related Work

ESNI and ECH. As a countermeasure to SNI inspection, the community is currently working on a new extension called Encrypted Server Name Indication (ESNI)—later redefined as the Encrypted `ClientHello` (ECH) extension [60]. Both extension versions encrypt the SNI extension to prevent analysis by censors or other middleboxes. However, they have not been fully standardized yet [60]. Additionally, with the ECH extension not yet being widely used [17], [71] and focusing on privacy protection rather than censorship circumvention [60], it can be censored easily by blocking it entirely [14], [76]. We consider evaluating the effectiveness of ECH for censorship circumvention as important future work over the next few years as its usage grows.

Locating Censorship Devices. In 2022, Raman et al. [57] determined the network location of various censorship devices and fingerprinted their behavior. As part of their fingerprinting efforts, they recorded the devices' censorship behavior when dealing with different TLS handshakes. For their evaluations, they considered semantically valid changes to TLS handshakes such as different TLS versions, cipher suites, or providing a client certificate. They also invalidated the domain name in the SNI extension. For example, they reversed the domain name and added subdomains. While Raman et al. could circumvent censors with some of their constructed TLS handshakes, they primarily used them to fingerprint censorship devices. Thus, they did not evaluate the acceptance of their constructed TLS handshakes by TLS servers. Additionally, they manipulated only specific parts of the TLS handshake in their analysis and did not combine different manipulations. In our work, we broadened the scope of Raman et al. by thoroughly analyzing the TLS handshake for possible manipulations. Instead of focusing on previously successful techniques, we considered all fields of the TLS handshake and TLS record headers for our manipulations.

Throttling Twitter. In their analysis of Russia's throttling of Twitter in 2021, Xue et al. [81] successfully circumvented the throttling by injecting additional TLS `ChangeCipherSpec` messages before the `ClientHello` message. Xue et al. also circumvented the throttling by inserting a large padding extension into the `ClientHello` message, which splits it over multiple TCP segments that the censor could not reassemble. This technique is particularly interesting as it forces changes on the TCP layer through manipulations on the TLS layer. While the techniques introduced by Xue et al. adhere to the TLS standards [20], [59], Xue et al. did not measure actual support by TLS servers. In our work, we increased TCP segment sizes through the padding extension, as done by Xue et al., and additional cipher suites.

Censorship Measurements. To provide a worldwide view of Internet censorship, OoniProbe [51], ICLab [48], and Censored Planet [66] publish data of repeatedly conducted censorship analyses around the world. Additionally, more specific analyses of censorship have been carried out in various countries worldwide. Examples include Russia [25], [81], Syria [2], Pakistan [47], Thailand [27], Iran [8], [32], and China's GFW [5], [13], [14], [22], [34], [75], [78]. While we analyzed the GFW and Iran's censor in our work, our approach is generic enough that it is also likely to work and circumvent SNI-based censorship in other countries.

Circumvention Tools. To help affected people, many censorship circumvention tools have been developed. They usually fall into one of three categories. Tools in the first category directly manipulate network packets that leave the operating system [16], [43], [73]. These tools usually perform TCP-level manipulations. A second category of tools operates a proxy server on the client's machine [16], [30], [30], [33], [36], [39], [41], [82]. By opening separate sockets, these tools can manipulate HTTP or other application traffic without breaking the client's TCP sequence numbers. Tools in the last category rely on a dedicated proxy behind the censor [61], [68], [72]. While this allows for great flexibility regarding circumvention techniques, clients must set up or be provided with such a dedicated proxy. Some tools in the second and third categories already implement TLS-level circumvention techniques. DPYProxy [39], Intra [38], and V2Ray [77] implement TLS record fragmentation. Tor Snowflake [68] uses domain fronting to connect to servers in CDNs.

Geneva. In our work, we built *Censor Scanner* to automatically detect circumventions for SNI-based censorship by manipulating TLS packets. Similarly, Bock et al. [13] built *Geneva*, a tool that automatically finds censorship circumventions by manipulating TCP packets with a genetic algorithm. *Geneva* has been further extended to the HTTP and DNS layer by Harrity et al. [32]. Since its inception, *Geneva* has successfully evaded censorship in China [13], [14], [32], India [13], [32], Kazakhstan [13], [32], Iran [11], and Turkmenistan [50].

9. Conclusions

In this work, we extend SNI-based censorship circumvention by applying alterations directly to the TLS layer. Our analyses of web servers and censors confirm the effectiveness of our approach, yielding numerous possible censorship circumvention techniques (**RQ1**) many of which are accepted by TLS servers (**RQ2**) and circumvent real-world censors (**RQ3**) at the same time. Circumvention tools can use these circumvention techniques to aid affected people. As we detect substantial differences in the acceptance of test vectors by different web server implementations, we stress the importance of analyzing online web servers in future research. Overall, we recommend applying the methodology developed in our work to the analysis of other protocols. We conjecture that similar circumvention techniques can be identified in cryptographic protocols like DTLS and QUIC, which share similarities with TLS.

Besides specific circumvention techniques, we identified and subsequently isolated three distinct middleboxes of the GFW, one of which was previously unknown. Exhaustively and systematically combining circumvention strategies proved vital for the detection of the three middleboxes, and we suspect that this is the reason why one middlebox stayed undetected for so long. We suggest future research to expect multiple possible middleboxes and to consider augmenting their analyses with the combinatorial approach presented in our work.

Acknowledgments

We want to thank the reviewers and shepherd for their insightful comments and constructive feedback. Niklas Niere and Felix Lange were supported by the German Federal Ministry of Education and Research (BMBF) through the project KoTeBi (16KIS1556K).

References

[1] "Amazon Web Services starts blocking domain-fronting, following Google's lead," https://www.theverge.com/2018/4/30/17304782/amazon-domain-fronting-google-discontinued, Apr. 2018.

[2] C. Abdelberi, T. Chen, M. Cunche, E. D. Cristofaro, A. Friedman, and M. A. Kâafar, "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*, C. Williamson, A. Akella, and N. Taft, Eds. ACM, 2014, pp. 285–298. [Online]. Available: https://doi.org/10.1145/2663716.2663720

[3] accessnow, "Google ends "domain fronting," a crucial way for tools to evade censors," Jan. 2023.

[4] C. Anderson, "Dimming the Internet: Detecting Throttling as a Mechanism of Censorship in Iran," Jun. 2013, arXiv:1306.4361 [cs]. [Online]. Available: http://arxiv.org/abs/1306.4361

[5] Anonymous, "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: USENIX Association, Aug. 2014. [Online]. Available: https://www.usenix.org/conference/foci14/workshop-program/presentation/anonymous

[6] Anonymous, A. A. Niaki, N. P. Hoang, P. Gill, and A. Houmansadr, "Triplet censors: Demystifying great Firewall's DNS censorship behavior," in *10th USENIX workshop on free and open communications on the internet (FOCI 20)*. USENIX Association, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/foci20/presentation/anonymous

[7] W. Arrington, "Continually Enhancing Domain Security on Amazon CloudFront | Networking & Content Delivery," https://aws.amazon.com/blogs/networking-and-content-delivery/continually-enhancing-domain-security-on-amazon-cloudfront/, Apr. 2019.

[8] S. Aryan, H. Aryan, and J. A. Halderman, "Internet Censorship in Iran: A First Look," in *3rd USENIX Workshop on Free and Open Communications on the Internet (FOCI 13)*. Washington, D.C.: USENIX Association, Aug. 2013. [Online]. Available: https://www.usenix.org/conference/foci13/workshop-program/presentation/aryan

[9] A. Bhaskar and P. Pearce, "Many roads lead to rome: How packet headers influence DNS censorship measurement," in *31st USENIX security symposium (USENIX security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 449–464. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/bhaskar

[10] K. Bock, P. Bharadwaj, J. Singh, and D. Levin, "Your Censor is My Censor: Weaponizing Censorship Infrastructure for Availability Attacks," in *2021 IEEE Security and Privacy Workshops (SPW)*, May 2021, pp. 398–409. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9474303

[11] K. Bock, Y. Fax, K. Reese, J. Singh, and D. Levin, "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, Aug. 2020. [Online]. Available: https://www.usenix.org/conference/foci20/presentation/bock

[12] K. Bock, G. Hughey, L.-H. Merino, T. Arya, D. Liscinsky, R. Pogosian, and D. Levin, "Come as You Are: Helping Unmodified Clients Bypass Censorship with Server-side Evasion," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. Virtual Event USA: ACM, Jul. 2020, pp. 586–598. [Online]. Available: https://dl.acm.org/doi/10.1145/3387514.3405889

[13] K. Bock, G. Hughey, X. Qiang, and D. Levin, "Geneva: Evolving Censorship Evasion Strategies," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '19. New York, NY, USA: Association for Computing Machinery, Nov. 2019, pp. 2199–2214. [Online]. Available: https://dl.acm.org/doi/10.1145/3319535.3363189

[14] K. Bock, iyouport, Anonymous, L.-H. Merino, D. Fifield, A. Houmansadr, and D. Levin, "Exposing and Circumventing China's Censorship of ESNI," 2020. [Online]. Available: https://gfw.report/blog/gfw_esni_blocking/en/

[15] K. Bock, G. Naval, K. Reese, and D. Levin, "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*, ser. FOCI '21. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 1–7. [Online]. Available: https://dl.acm.org/doi/10.1145/3473604.3474559

[16] bol van, "zapret," May 2024. [Online]. Available: https://github.com/bol-van/zapret

[17] Z. Chai, A. Ghafari, and A. Houmansadr, "On the importance of Encrypted-SNI (ESNI) to censorship circumvention," in *9th USENIX workshop on free and open communications on the internet (FOCI 19)*. Santa Clara, CA: USENIX Association, Aug. 2019. [Online]. Available: https://www.usenix.org/conference/foci19/presentation/chai

[18] R. Clayton, S. J. Murdoch, and R. N. M. Watson, "Ignoring the Great Firewall of China," in *Privacy Enhancing Technologies*, G. Danezis and P. Golle, Eds. Berlin, Heidelberg: Springer, 2006, pp. 20–35.

[19] Cloudflare, "Cloudflare Radar | Adoption & Usage Worldwide," 2024. [Online]. Available: https://radar.cloudflare.com/adoption-and-usage

[20] T. Dierks and E. Rescorla, "Rfc 5246: The transport layer security (tls) protocol version 1.2," 2008.

[21] D. Eastlake 3rd, "Rfc 6066: Transport layer security (tls) extensions: Extension definitions," 2011.

[22] R. Ensafi, P. Winter, A. Mueen, and J. R. Crandall, "Analyzing the Great Firewall of China Over Space and Time," *Proceedings on Privacy Enhancing Technologies*, 2015. [Online]. Available: https://petsymposium.org/popets/2015/popets-2015-0005.php

[23] European Union. (2025) Eu sanctions map. https://www.sanctionsmap.eu/. [Online]. Available: https://www.sanctionsmap.eu/

[24] D. Fifield, C. Lan, R. Hynes, P. Wegmann, and V. Paxson, "Blocking-resistant communication through domain fronting," *Proceedings on Privacy Enhancing Technologies*, vol. 2015, no. 2, pp. 46–64, Jun. 2015. [Online]. Available: https://petsymposium.org/popets/2015/popets-2015-0009.php

[25] funkerwolf, "Почему Ростелеком блокирует esni трафик?," 2020. [Online]. Available: https://qna.habr.com/q/862669

[26] gaul. (2021) Awesome ld_preload. [Online]. Available: https://github.com/gaul/awesome-ld-preload

[27] G. Gebhart and T. Kohno, "Internet Censorship in Thailand: User Practices and Potential Threats," in *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, Apr. 2017, pp. 417–432. [Online]. Available: https://ieeexplore.ieee.org/document/7961994

[28] Google, "boringssl," 2024. [Online]. Available: https://boringssl.googlesource.com/boringssl/

[29] Google, "HTTPS Encryption in the Web – Google Transparency Report," Jul. 2024. [Online]. Available: https://transparencyreport.google.com/https/overview

[30] Google Jigsaw, "Jigsaw-Code/Intra," Jun. 2024. [Online]. Available: https://github.com/Jigsaw-Code/Intra

[31] greatfire, "Greatfire/freebrowser," GreatFire.org, May 2024.

[32] M. Harrity, K. Bock, F. Sell, and D. Levin, "GET /out: Automated Discovery of Application-Layer Censorship Evasion Strategies," in *31st USENIX Security Symposium (USENIX Security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 465–483. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/harrity

[33] S. Hayeri, "GreenTunnel," May 2024. [Online]. Available: https://github.com/SadeghHayeri/GreenTunnel

[34] N. P. Hoang, J. Dalek, M. Crete-Nishihata, N. Christin, V. Yegneswaran, M. Polychronakis, and N. Feamster, "Gfweb: Measuring the great firewall's web censorship at scale," jun 2024, selected for publication at Usenix 2024: https://www.usenix.org/conference/usenixsecurity24/presentation/hoang.

[35] N. P. Hoang, A. A. Niaki, J. Dalek, J. Knockel, P. Lin, B. Marczak, M. Crete-Nishihata, P. Gill, and M. Polychronakis, "How Great is the Great Firewall? Measuring China's DNS Censorship," in *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, Aug. 2021, pp. 3381–3398. [Online]. Available: https://www.usenix.org/conference/usenixsecurity21/presentation/hoang

[36] hufrea, "hufrea/byedpi: Bypass DPI," 2024. [Online]. Available: https://github.com/hufrea/byedpi

[37] J. Iyengar (Ed.) and M. Thomson (Ed.), "QUIC: a UDP-based multiplexed and secure transport," May 2021, iSSN: 2070-1721 Number: 9000 Place: Fremont, CA, USA Series: Internet request for comments Type: RFC tex.howpublished: RFC 9000 (Proposed Standard) tex.key: RFC 9000. [Online]. Available: https://www.rfc-editor.org/rfc/rfc9000.txt

[38] G. Jigsaw. (2023) Outline SDK, TLS client hello fragmentation by fixed length #134. [Online]. Available: https://github.com/Jigsaw-Code/outline-sdk/pull/134

[39] JonSnowWhite, "DPYProxy," 2024. [Online]. Available: https://github.com/UPB-SysSec/DPYProxy

[40] S. Khattak, M. Javed, P. D. Anderson, and V. Paxson, "Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion," 2013. [Online]. Available: https://www.usenix.org/conference/foci13/workshop-program/presentation/khattak

[41] krlvm, "PowerTunnel," May 2024. [Online]. Available: https://github.com/krlvm/PowerTunnel

[42] F. Li, A. Razaghpanah, A. M. Kakhki, A. A. Niaki, D. Choffnes, P. Gill, and A. Mislove, "lib•erate, (n): a library for exposing (traffic-classification) rules and avoiding them efficiently," in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC '17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 128–141. [Online]. Available: https://dl.acm.org/doi/10.1145/3131365.3131376

[43] macronut, "ghostcp," May 2024. [Online]. Available: https://github.com/macronut/ghostcp

[44] A. Master and C. Garman, “A Worldwide View of Nation-state Internet Censorship,” *Free and Open Communications on the Internet*, 2023. [Online]. Available: https://petsymposium.org/foci/2023/foci-2023-0008.php

[45] moxie0, “A letter from Amazon,” https://signal.org/blog/looking-back-on-the-front/, May 2018.

[46] P. Müller, N. Niere, F. Lange, and J. Somorovsky, “Turning Attacks into Advantages: Evading HTTP Censorship with HTTP Request Smuggling,” in *Free and Open Communications on the Internet*, 2024. [Online]. Available: https://foci.community/foci24.html

[47] Z. Nabi, “The anatomy of web censorship in pakistan,” in *3rd USENIX workshop on free and open communications on the internet (FOCI 13)*. Washington, D.C.: USENIX Association, Aug. 2013. [Online]. Available: https://www.usenix.org/conference/foci13/workshop-program/presentation/nabi

[48] A. A. Niaki, S. Cho, Z. Weinberg, N. P. Hoang, A. Razaghpanah, N. Christin, and P. Gill, “ICLab: A Global, Longitudinal Internet Censorship Measurement Platform,” in *2020 IEEE Symposium on Security and Privacy (SP)*, May 2020, pp. 135–151, iSSN: 2375-1207. [Online]. Available: https://ieeexplore.ieee.org/document/9152784

[49] N. Niere, S. Hebrok, J. Somorovsky, and R. Merget, “Poster: Circumventing the GFW with TLS Record Fragmentation,” *CCS 2023 - Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pp. 3528–3530, Nov. 2023, iSBN: 9798400700507 Publisher: Association for Computing Machinery, Inc. [Online]. Available: https://dl.acm.org/doi/10.1145/3576915.3624372

[50] S. Nourin, V. Tran, X. Jiang, K. Bock, N. Feamster, N. P. Hoang, and D. Levin, “Measuring and Evading Turkmenistan’s Internet Censorship: A Case Study in Large-Scale Measurements of a Low-Penetration Country,” in *Proceedings of the ACM Web Conference 2023*. Austin TX USA: ACM, Apr. 2023, pp. 1969–1979. [Online]. Available: https://dl.acm.org/doi/10.1145/3543507.3583189

[51] Open Observatory of Network Interference, “OONI: Open Observatory of Network Interference,” May 2024. [Online]. Available: https://ooni.org/

[52] OpenSSL, “openssl: TLS/SSL and crypto library,” 2024. [Online]. Available: https://github.com/openssl/openssl

[53] R. Padmanabhan, A. Filastò, M. Xynou, R. S. Raman, K. Middleton, M. Zhang, D. Madory, M. Roberts, and A. Dainotti, “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*, ser. FOCI ’21. New York, NY, USA: Association for Computing Machinery, Aug. 2021, pp. 27–36. [Online]. Available: https://dl.acm.org/doi/10.1145/3473604.3474562

[54] V. L. Pochat, T. van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, “Tranco: A research-oriented top sites ranking hardened against manipulation,” in *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. [Online]. Available: https://www.ndss-symposium.org/ndss-paper/tranco-a-research-oriented-top-sites-ranking-hardened-against-manipulation/

[55] Psiphon, “Why You Don’t Need Google’s Domain Fronting,” Apr. 2018.

[56] R. S. Raman, L. Evdokimov, E. Wurstrow, J. A. Halderman, and R. Ensafi, “Investigating Large Scale HTTPS Interception in Kazakhstan,” in *Proceedings of the ACM Internet Measurement Conference*, ser. IMC ’20. New York, NY, USA: Association for Computing Machinery, Oct. 2020, pp. 125–132. [Online]. Available: https://dl.acm.org/doi/10.1145/3419394.3423665

[57] R. S. Raman, M. Wang, J. Dalek, J. Mayer, and R. Ensafi, “Network measurement methods for locating and examining censorship devices,” in *Proceedings of the 18th International Conference on emerging Networking EXperiments and Technologies*, ser. CoNEXT ’22. New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 18–34. [Online]. Available: https://dl.acm.org/doi/10.1145/3555050.3569133

[58] E. Rescorla and N. Modadugu, “Datagram transport layer security version 1.2,” Jan. 2012, iSSN: 2070-1721 Number: 6347 Place: Fremont, CA, USA Series: Internet request for comments Type: RFC tex.howpublished: RFC 6347 (Proposed Standard) tex.key: RFC 6347. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6347.txt

[59] E. Rescorla, “Rfc 8446: The transport layer security (tls) protocol version 1.3,” 2018.

[60] E. Rescorla, K. Oku, N. Sullivan, and C. A. Wood, “TLS Encrypted Client Hello,” Internet Engineering Task Force, Internet Draft draft-ietf-tls-esni-24, Mar. 2025, num Pages: 53. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-tls-esni

[61] S. Satija and R. Chatterjee, “BlindTLS: Circumventing TLS-based HTTPS censorship,” in *Proceedings of the ACM SIGCOMM 2021 Workshop on Free and Open Communications on the Internet*. Virtual Event USA: ACM, Aug. 2021, pp. 43–49. [Online]. Available: https://dl.acm.org/doi/10.1145/3473604.3474564

[62] K. Singh, G. Grover, and V. Bansal, “How India Censors the Web,” in *Proceedings of the 12th ACM Conference on Web Science*, ser. WebSci ’20. New York, NY, USA: Association for Computing Machinery, Jul. 2020, pp. 21–28. [Online]. Available: https://dl.acm.org/doi/10.1145/3394231.3397891

[63] J. Somorovsky, “Systematic Fuzzing and Testing of TLS Libraries,” in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’16. New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 1492–1504. [Online]. Available: https://doi.org/10.1145/2976749.2978411

[64] D. Stenberg, “curl,” 1996. [Online]. Available: https://curl.se/

[65] K. Subramani, R. Perdisci, P.-C. Skafidas, and M. Antonakakis, “Discovering and Measuring CDNs Prone to Domain Fronting,” in *Proceedings of the ACM on Web Conference 2024*. Singapore Singapore: ACM, May 2024, pp. 1859–1867. [Online]. Available: https://dl.acm.org/doi/10.1145/3589334.3645656

[66] R. Sundara Raman, P. Shenoy, K. Kohls, and R. Ensafi, “Censored Planet: An Internet-wide, Longitudinal Censorship Observatory,” in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’20. New York, NY, USA: Association for Computing Machinery, Nov. 2020, pp. 49–66. [Online]. Available: https://dl.acm.org/doi/10.1145/3372297.3417883

[67] The Tor Project, “Meek,” https://gitlab.torproject.org/tpo/anti-censorship/pluggable-transports/meek, Mar. 2024.

[68] ——, “Snowflake,” 2024. [Online]. Available: https://snowflake.torproject.org/

[69] M. Thomson, “Record Size Limit Extension for TLS,” RFC 8449, Aug. 2018. [Online]. Available: https://www.rfc-editor.org/info/rfc8449

[70] tls-attacker, “TLS-Attacker,” Jun. 2024. [Online]. Available: https://github.com/tls-attacker/TLS-Attacker

[71] Z. Tsiatsikas, G. Karopoulos, and G. Kambourakis, “Measuring the Adoption of TLS Encrypted Client Hello Extension and Its Forebear in the Wild,” in *Computer Security. ESORICS 2022 International Workshops*, S. Katsikas, F. Cuppens, C. Kalloniatis, J. Mylopoulos, F. Pallas, J. Pohle, M. A. Sasse, H. Abie, S. Ranise, L. Verderame, E. Cambiaso, J. Maestre Vidal, M. A. Sotelo Monge, M. Albanese, B. Katt, S. Pirbhulal, and A. Shukla, Eds. Cham: Springer International Publishing, 2023, pp. 177–190.

[72] v2fly, “v2ray: A platform for building proxies to bypass network restrictions.” Jun. 2024. [Online]. Available: https://github.com/v2fly/v2ray-core

[73] ValdikSS, “GoodbyeDPI,” May 2024. [Online]. Available: https://github.com/ValdikSS/GoodbyeDPI

[74] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. V. Krishnamurthy, “Your State is Not Mine: A Closer Look at Evading Stateful Internet Censorship,” in *Proceedings of the 2017 Internet Measurement Conference*, ser. IMC ’17. New York, NY, USA: Association for Computing Machinery, Nov. 2017, pp. 114–127. [Online]. Available: https://dl.acm.org/doi/10.1145/3131365.3131374

[75] P. Winter and S. Lindskog, "How the Great Firewall of China is Blocking Tor," in *2nd USENIX Workshop on Free and Open Communications on the Internet (FOCI 12)*. Bellevue, WA: USENIX Association, Aug. 2012. [Online]. Available: https://www.usenix.org/conference/foci12/workshop-program/presentation/Winter

[76] wkrp, "Blocking of Cloudflare ECH in Russia, 2024-11-05 · Issue #417 · net4people/bbs," Nov. 2024. [Online]. Available: https://github.com/net4people/bbs/issues/417

[77] XTLS. (2023) Xray-core, replace TCP segmentation with TLS hello fragmentation #2131. [Online]. Available: https://github.com/XTLS/Xray-core/pull/2131

[78] X. Xu, Z. M. Mao, and J. A. Halderman, "Internet Censorship in China: Where Does the Filtering Occur?" in *Passive and Active Measurement*, N. Spring and G. F. Riley, Eds. Berlin, Heidelberg: Springer, 2011, pp. 133–142.

[79] D. Xue, B. Mixon-Baca, ValdikSS, A. Ablove, B. Kujath, J. R. Crandall, and R. Ensafi, "TSPU: Russia's decentralized censorship system," in *Proceedings of the 22nd ACM Internet Measurement Conference*, ser. IMC '22. New York, NY, USA: Association for Computing Machinery, Oct. 2022, pp. 179–194. [Online]. Available: https://dl.acm.org/doi/10.1145/3517745.3561461

[80] D. Xue, R. Ramesh, A. Jain, M. Kallitsis, J. A. Halderman, J. R. Crandall, and R. Ensafi, "OpenVPN is open to VPN fingerprinting," in *31st USENIX security symposium (USENIX security 22)*. Boston, MA: USENIX Association, Aug. 2022, pp. 483–500. [Online]. Available: https://www.usenix.org/conference/usenixsecurity22/presentation/xue-diwen

[81] D. Xue, R. Ramesh, V. S. S, L. Evdokimov, A. Viktorov, A. Jain, E. Wustrow, S. Basso, and R. Ensafi, "Throttling Twitter: an emerging censorship technique in Russia," in *Proceedings of the 21st ACM Internet Measurement Conference*, ser. IMC '21. New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 435–443. [Online]. Available: https://dl.acm.org/doi/10.1145/3487552.3487858

[82] xvzc, "SpoofDPI," May 2024. [Online]. Available: https://github.com/xvzc/SpoofDPI

[83] T. K. Yadav, A. Sinha, D. Gosain, P. K. Sharma, and S. Chakravarty, "Where The Light Gets In: Analyzing Web Censorship Mechanisms in India," in *Proceedings of the Internet Measurement Conference 2018*, ser. IMC '18. New York, NY, USA: Association for Computing Machinery, Oct. 2018, pp. 252–264. [Online]. Available: https://doi.org/10.1145/3278532.3278555

Appendix A. Manipulation Details

Figure 1 depicts the structure of a `ClientHello` message with the surrounding TLS record structure and the contained SNI extension. We manipulated all `ClientHello` fields except the `Random`, `Compression Methods`, and `Session ID`. Next to direct manipulation of single fields, we also considered more involved structural manipulations of our flight, such as TLS record fragmentation, the injection of non-handshake records, or the addition of extensions in the `ClientHello`. In the following, we detail our manipulations and how we apply them to a default TLS 1.2 `ClientHello` message.

Base Version. The initial `ClientHello` we manipulate is built for a default TLS 1.2 handshake. As a structural manipulation, we changed the TLS version of the base `ClientHello` to the TLS versions 1.0, 1.1, 1.2, and 1.3. For each version, we sent the corresponding cipher suites, included necessary extensions such as the `Supported Versions` extension, and used version-specific algorithms during the handshake if we tried to test the establishment of the whole connection.

Record Fragmentation. As another structural manipulation, we fragmented the `ClientHello` message across multiple TLS records. We defined four different initial fragmentation points: After two bytes in the message header, before the SNI extension, in the middle of the hostname in the SNI extension, and after the SNI extension. Subsequent records have the same size as the initial record.

Record Injection. Besides the fragmentation of records, we injected additional records into the TLS handshake. Xue et al. [81] discovered `ChangeCipherSpec` message injection as a successful circumvention technique in Russia. We generalized their finding by injecting different records in various locations around the `ClientHello` message. We injected all defined records before the `ClientHello` message, after the `ClientHello` message, and in all positions defined for record fragmentation. For the latter, we fragmented the `ClientHello` message and injected an additional record in between.

Record and Message Types. The record header and message header of a `ClientHello` message contain a type field that indicates the record type and message type, respectively.We manipulated both types with different defined and undefined values. .

Version Fields. The TLS `ClientHello` message contains two TLS version fields. One in the record header and one in the message header. With this manipulation, we only changed the value of the version field and did not attempt to handshake the TLS version we inserted into the version field.

Cipher Suites. To exceed censors' buffer sizes, we artificially increased the size of the `ClientHello`. To this end, we added cipher suites to the `Cipher Suite` field. For this manipulation, we defined two possible values: the maximum allowed size for the record and the maximum possible value for the 2-byte length field that indicates the size of the TLS record. To achieve this, we added multiple instantiations of the same cipher suite `TLS_NULL_WITH_NULL_NULL`. We kept all other cipher suites unchanged.

Length Fields. A `ClientHello` message with a SNI extension contains six length fields: The record length, the message length, the length of all extensions, the length of the SNI extension, the length of the SNI's list structure, and the length of the hostname. We manipulated all six length fields by setting the length to higher values, lower values, or the original value when other manipulations would implicitly increase it through insertions.In another length field manipulation, we decreased the size of the length field so that it excluded the last entry of the structure it refers to. For example, the `List Length` field of the SNI extension can exclude the last list entry. We also manipulated specific length fields by increasing their size by 20 and appending

20 bytes to the end of the structure referred to by the length field.

TLS Extensions. As it is of paramount importance for the censor, we focused on manipulating the SNI extension in our evaluations. We defined manipulations that remove the SNI extension, change its place in the extension list, or add additional SNI extensions with the same hostname as the original. Any added SNI extensions are considered by other manipulations that alter fields in the SNI extension. Specifically, these manipulations can alter the original SNI extension, additional SNI extensions, or all at once. We hoped to evoke mismatches in censors and servers regarding which SNI extensions they prefer to parse. Next to manipulations of the SNI extension, we also injected a padding extension into the extension list to artificially increase the size of the `ClientHello` message, exceeding potentially limited buffer sizes of the censor.

SNI Type Fields. We manipulated the `Extension Type` and `Name Type` fields of the SNI extensions by invalidating them. We set the `Extension Type` value from `0x0000` to `0x9999`. A similar circumvention has been found by Bock et al. [14] for the ESNI extension. We altered the `Name Type` from `0x00` to `0x01`. We anticipated that some censors and servers would ignore the name type because only one name type has ever been defined.

SNI List Entries. Similar to the addition of SNI extensions, we added list entries to the SNI extension or removed the original list entry. We added one list entry, two list entries, or the number of list entries that maximize the SNI extension size. The additionally inserted hostnames are the same as the original, except when the manipulation adds the maximum number of allowed hostnames. There, we added the original hostname in the middle and at the end of the list, while all other hostnames are set to an uncensored hostname. We expected different server and censor behavior for multiple list entries, especially when another manipulation further invalidates a list entry.

SNI Hostname Manipulations. We defined a total of six manipulations for the hostname. The first manipulation changes the casing of the hostname to an alternating case. Notably, this is a standard-conforming manipulation as the hostname is case-insensitive. Another manipulation flips the highest bit of all ASCII characters, turning them into non-ASCII characters. Yet another manipulation injects symbols into the hostname: A null byte, a space, a backspace, subdomains, top-level domains, or possibly incomplete Unicode characters. Possible insertion points for these injections are before, in the middle of, or after the hostname. The fourth manipulation pads the hostname to the maximum allowed length by appending the character *g*. Another possible manipulation replaces the existing hostname with an uncensored hostname. We expected most hostname manipulations to be largely successful against censors. We anticipated some of them to lead to domain fronting opportunities on CDNs.

Appendix B. CDN Websites

Table 5. SELECTED CDN WEBSITES.

	Shared IP	**Unshared IP**
Cloudflare	medium.com	vimeo.com
Fastly	apache.org	nytimes.com
Google	tensorflow.org	boardgamegeek.com
Akamai	europeanbusinessreview.com	walmart.com
Amazon	fxhome.com	flickr.com

We chose one website that is hosted on a server where it shares an IP address with other servers, and one website that is hosted on a dedicated server.

Appendix C. Server Specifications

Table 6. SPECIFICATION OF OUR VANTAGE POINT IN CHINA.

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

Table 7. SPECIFICATION OF OUR VANTAGE POINT IN IRAN.

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

Table 8. SPECIFICATION OF OUR VANTAGE POINT IN GERMANY.

Country:	Berlin, Germany
AS#:	201295
Vendor:	IONOS
URL:	https://www.ionos.de/
ISP:	IONOS SE (private)

Appendix D. TLS Handshake

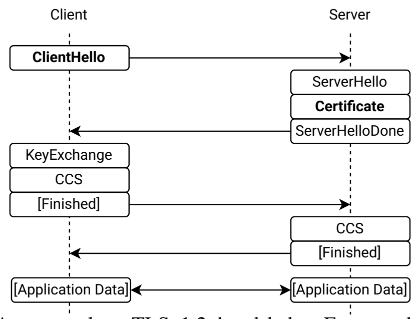

Figure 5. An exemplary TLS 1.2 handshake. Encrypted messages are surrounded by brackets. Messages that contain the hostname of the server are printed in bold. A TLS 1.3 handshake is similar but also encrypts the server's certificate message and the extensions in the `ServerHello` message.