USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Technical Analysis of the Geedge Networks Firewall Source Code Leak

Anna Ablove, *University of Michigan;* Johnnie Walker, *GFW Report;* Ben Wolin, *University of Michigan;* Niklas Niere and Felix Lange, *Paderborn University;* Aaron Ortwein, Armin Huremagic, and Richa Priyanka, *University of Michigan;* Ali Zohaib and Jade Sheffey, *University of Massachusetts Amherst;* Nico Heitmann, *Paderborn University;* J. Alex Halderman, *University of Michigan;* Juraj Somorovsky, *Paderborn University;* Amir Houmansadr, *University of Massachusetts Amherst;* Roya Ensafi, *University of Michigan;* Mingshi Wu, *GFW Report;* Eric Wustrow, *University of Colorado Boulder*

https://www.usenix.org/conference/usenixsecurity26/presentation/ablove

This paper is included in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Proceedings of the 35th USENIX Security Symposium is sponsored by

جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology

Technical Analysis of the Geedge Networks Firewall Source Code Leak

Anna Ablove[1], Johnnie Walker[2], Ben Wolin[1], Niklas Niere[3], Felix Lange[3], Aaron Ortwein[1], Armin Huremagic[1], Richa Priyanka[1], Ali Zohaib[4], Jade Sheffey[4], Nico Heitmann[3], J. Alex Halderman[1], Juraj Somorovsky[3], Amir Houmansadr[4], Roya Ensafi[1], Mingshi Wu[2], Eric Wustrow[5]

[1]*University of Michigan* [2]*GFW Report* [3]*Paderborn University*
[4]*University of Massachusetts Amherst* [5]*University of Colorado Boulder*

Abstract

In September 2025, over 100K internal documents (including code, communications, etc.) from Geedge Networks, a Chinese DPI company with ties to the Great Firewall of China, were leaked to the public. In this paper, we analyze the source code from this leak, focusing on Geedge Networks' flagship product, the Tiangou Secure Gateway (TSG) firewall. Working across multiple repositories, we successfully build and run a local copy of TSG—revealing key aspects of its architecture, including the protocols it is capable of parsing and the format of blocking rules used to censor sites, proxies, and other resources. Finally, we extract several fingerprints from TSG, including custom random number generators and parsing idiosyncrasies that allow us to identify its use and similar deployments in the Great Firewall of China.

This is the *first time* that the source code of a commercial DPI has been leaked, and our work is the first code analysis of a core firewall component used in national censorship infrastructure. This unprecedented investigation offers insights that can assist circumvention developers and Internet security researchers in further understanding the capabilities and limitations of modern censorship technology.

1 Introduction

In recent years, there has been a major increase in the commoditization and utilization of Deep Packet Inspection devices (DPIs). It is easier than ever before for authoritarian nation-states to implement information controls and restrict Internet access. Traditionally, the study of DPI-based censorship systems has relied almost exclusively on black-box analysis, using Internet measurement experiments to infer the behavior of opaque and tightly controlled infrastructures, including those deployed by China [13, 20, 47], Russia [36, 37, 49], and Iran [10, 26, 39]. While this approach has yielded valuable insights, it is inherently limited: measurements are often noisy and incomplete, require substantial resources to deploy at scale, and can expose researchers and volunteers to ethical, legal, and personal safety risks.

The Leak: A significant leak of internal documents from Geedge Networks—a Chinese DPI company affiliated with the Great Firewall of China—was made public in mid-September of 2025 [2, 4, 8, 14, 24]. The leak contained over 100K files from Jira, Confluence, and, critically, Git repositories containing over five years' worth of commit histories, comments, and branches with the latest documents dating to November 2024. Several news organizations and journalists who were granted early access have worked together for the past year to analyze the documents from a non-technical perspective. Reports from The Globe and Mail [2], Amnesty International [3, 8], InterSecLab [5, 24], and Justice for Myanmar [4] document the export of Geedge Networks' censorship systems, including its flagship product Tiangou Secure Gateway (TSG), to Pakistan, Myanmar, Ethiopia, and Kazakhstan, focusing on human-rights impacts, procurement networks, and the spread of Great Firewall-style controls. The current reporting is mostly geared to non-technical audiences—leaving the source code itself largely unexplored.

This is the *first time* that the source code of a commercial DPI has been leaked; and initial examinations show not only code, but also compiled binaries, configuration files, and artifacts. However, with over 500 Git repositories alone, the leak presents a formidable analytical challenge. For instance, compared to Snort [38], one of the few open-source DPIs, TSG's logic is not localized to a single repository, but distributed across many, leaving systematic interpretation difficult. The challenge is compounded by the many different branches and versions of the same components—as well as several cases of fully separate implementations, including at least two distinct IP/UDP/TCP stacks. Challenges aside, this leak represents a remarkable and unprecedented opportunity for the anti-censorship and network security communities to study an actual enterprise DPI system.

In this paper, we aim to fill this reporting gap with the first systematic technical analysis of the leaked source code. Prior to this leak, three decades of remote fingerprinting and reverse engineering efforts have resulted in deep insights and advanced our understanding of censorship systems—yet fun-

damental questions have remained unanswered. For the first time, we can move beyond remote inference and ask: *How do these DPIs parse and analyze traffic through the system? What processes do they implement to block VPNs and circumvention tools?*

In order to fully explore those questions, we set out to systematize the architecture of TSG (shown in Figure 1). This required iteratively building our understanding of the system, spanning raw packet ingress, attribute extraction, blocking enforcement, and application detection. To better understand the system, **we successfully build and run TSG's latest tagged release version. We find significant integration of third-party code**, including two of the three core application detection systems. Also, **we identify client-driven, iterative efforts to block specific circumvention tools**, including Psiphon, Tor, and many popular VPN providers, like ExpressVPN and NordVPN, with a variety of methods ranging from IP-based blocking to matching byte patterns.

Equipped with our build, we conduct measurements to explore potential deployments. Due to the modularity of the system, this is a difficult undertaking. We do targeted tests focusing on the DNS, IP, TCP, TLS, and QUIC protocols.**We find high levels of correspondence with the GFW, uncovering a boundary of 17 compression pointers for GFW's DNS *Injector 2*** as well as a signature `seed_key` embedded in a **custom pseudo-random generator** used to set packet header values in **TCP injectors *GFW II* and *GFW III***. We also find a correlation between *GFW II* and an older version of TSG's SSL module.

Our study is the first source code analysis of a commercial DPI system—Geedge Networks' TSG. We successfully deploy a local build of TSG, allowing us to conduct real-time testing and analysis of the system and its different configuration options. We will make our build available to trusted network security researchers upon request. Overall, we hope this work can assist circumvention tool developers and Internet security researchers in better understanding the capabilities and limitations of modern censorship infrastructure as well as serve as a guide for future research on the leak.

2 Background and Related Work

2.1 The Great Firewall of China

Active since the late 1990s, the Great Firewall of China (GFW) is the most sophisticated Internet censorship apparatus in the world, systematically moderating traffic, restricting access to foreign domains, and limiting free speech [22, 41]. Traditionally, the GFW has been considered highly centralized; however, in recent years, there has been increasing evidence pointing to the deployment of multiple, independently operating deep packet inspection (DPI) devices with distinct enforcement roles [47, 48].

DNS Injections. The GFW tampers with unencrypted DNS queries by racing legitimate DNS resolvers, returning forged responses containing incorrect IP addresses [16, 27]. Anonymous et al. discovered that DNS manipulation is managed by three independent on-path injectors, fingerprintable by idiosyncrasies in response packet headers such as the IP Don't Fragment (DF) bit and Time to Live (TTL) field [9].

TCP RST Injections. The GFW filters HTTP and HTTPS traffic by inspecting unencrypted application-layer fields, such as the HTTP Host header, sensitive keywords in the HTTP request body, and the TLS Server Name Indication (SNI), and then injecting RST packets to tear down disallowed connections [13, 35]. Prior work has identified multiple RST injectors within the GFW, each characterized by distinct injection behavior, including differences in the number of RST packets sent and the TCP flags set [12, 43, 47].

Packet Drops. Beyond the packet injection behavior described above, the GFW can also restrict access to specific IP addresses or the use of certain protocols (e.g., QUIC or circumvention protocols) by dropping packets. Beginning in April 2024, the GFW began blocking QUIC by decrypting Initial packets and dropping connections based on the TLS SNI field [50]. Additionally, packet drops are used to enforce *residual censorship*, in which any traffic matching the 3-tuple `(src IP, dst IP, dst port)` of a censored connection is dropped for a short duration following the block [11, 43].

Circumvention Tool Blocking. The GFW also engages in targeted blocking of specific circumvention tools and protocols. Early work by Winter et al. demonstrated that the GFW fingerprinted Tor by recognizing distinct features in its TLS handshake [44]. Subsequent work by Alice et al. showed that the GFW can flag possible Shadowsocks connections using the payload length and entropy of early packets [7]. More recently, Wu et al. showed that the GFW relies on lightweight statistical heuristics like bit distributions and the position of printable ASCII characters to detect fully encrypted protocols, including Shadowsocks, VMess, and obfs4 [46]. The GFW has also been observed to use active probing to confirm suspected circumvention servers prior to blocking [17, 19].

2.2 Geedge Networks Leak

In September 2025, more than 100K internal documents, source code repositories, and build artifacts from Chinese DPI company Geedge Networks were leaked online. A breakdown of the leak, including relevant code and documents, is shown in Table 1. The company was founded in 2018 by Fang Binxing, who is colloquially known as the "Father of the GFW" for his foundational role in designing the Great Firewall [21]. Additionally, Geedge Networks maintains a close collaborative relationship with the Massive and Effective Stream Analysis (MESA) Lab at the Chinese Academy

Category	Artifact	Size	Key Contents (§)
Source code	`mirror/repo.tar`	463.0 GiB	RPM bundles*: Firewall, libcbd, QDPI and Glimpse detectors (§3, §4)
	`mesalab_git.tar.zst`	59.4 GiB	Git repos: SAPP, Protocol Plugins, Stellar-on-SAPP, Stellar, Maat, tsg-os-buildimage (§3, §5)
Confluence	*2 archives*	46.4 GiB	VPN signature logs, Psiphon collateral-damage analysis (§4)
Jira	`geedge_jira.tar.zst`	2.5 GiB	Deployment tickets (Myanmar, Ethiopia), Bug fixes (§4)
Other	*14 files + filelist*	10.7 MiB	—

Table 1: **Composition of 572 GiB Geedge Networks Data Leak.** We localize relevant repositories and documentation to their positions in the leak. *Note, SAPP and the protocol plugins are additionally packaged in RPM bundles in `repo.tar`.

of Sciences [2], whose academic research on traffic analysis directly informs product development [24, 28].

Geedge Networks' flagship DPI product, the Tiangou Secure Gateway (TSG), is a full-stack censorship platform designed to fully integrate into national telecommunications controls. TSG provides a wide range of traffic control capabilities, including blocking content, detecting VPNs and circumvention tools, and ISP-scale user tracking [24].

Civil society investigations have highlighted evidence of deployments of Geedge Networks' products across multiple countries, including Kazakhstan (code-named K18/K24), Ethiopia (E21), Pakistan (P19), and Myanmar (M22), and describe the company's role as extending beyond software provision to encompass system integration, operator training, and ongoing technical support [14, 24, 28]. In June 2024, Justice for Myanmar reported that Geedge Networks technology was being used to block VPN connections [45]. Despite the importance and unprecedented scale of this leak, no public analysis has yet reconstructed how TSG actually works at the packet-processing level, how it fingerprints circumvention tools, or how it enforces censorship rules. Existing reporting focuses largely on political implications, leaving this crucial technical gap unaddressed.

We highlight that Internet censorship is widely recognized as a violation of Article 19 of the Universal Declaration of Human Rights [42], and circumvention tools have proven essential for journalists and activists in contexts ranging from Iran's Mahsa Amini protests to Myanmar's 2021 coup [23, 32]. Technical transparency into DPI systems, which directly informs circumvention efforts, is therefore a priority for the anti-censorship community.

Critically, this is the first time that the internal design and source code of a commercial DPI system have been leaked. Historically, knowledge about the parsing, reassembly, and classification logic that underpins modern DPI systems has been inferred from measurements [31, 33, 34] or learned from open-source systems such as Snort [38], Suricata, Zeek [30], and nDPI [15, 25, 48]. In this paper, we take advantage of this unique opportunity to conduct the first in-depth technical analysis of Geedge Networks' TSG product.

3 TSG Architecture

In this section, we detail the architecture of Geedge Networks' Tiangou Secure Gateway (TSG) DPI system. Modern DPI systems generally follow a four-stage workflow: (1) *Ingress*, in which the DPI receives inline or mirrored traffic; (2) *Processing*, in which the DPI reassembles flows, parses protocols into their constituent fields, and maintains connection state; (3) *Evaluation*, in which the DPI determines whether a connection matches a censorship policy; and (4) *Interference*, in which the DPI prevents, degrades, or terminates disallowed connections.

Guided by this workflow, we conducted iterative code analysis to identify the internal components of TSG responsible for each of these stages and to establish how these components interact. TSG's complex architecture allows a single system component to provide functionality across different stages. We present the overall workflow of TSG in Figure 1.

3.1 SAPP: Raw Packet Ingress and APIs

On ingress, packets enter SAPP, where they are processed into UDP and TCP streams and sent to plugins via SAPP streams. Stream Analysis Process Platform (SAPP) is the base packet processing platform of TSG. Since its development by MESA Labs in 2005, SAPP has undergone four major iterations—start, papp, SAPPv3, and SAPPv4—with the latest available version of TSG built atop SAPPv4. At a high level, SAPP is a *plugin* platform that separates lower-level packet processing from other system components. It handles parsing of the L2–L4 layers (Ethernet, IP, and UDP/TCP) and of tunneling and encapsulation protocols (e.g., VLAN, PPPoE, VXLAN) before forwarding the reassembled packets to plugins for further parsing and policy evaluation.

Plugin Architecture. SAPP supports three types of plugins, which are specified and separated by type via the `conflist.inf` configuration file. Each plugin has its own `*.inf` file, which registers it to specified stream types. *Platform plugins* provide shared functionality used by other plugins. *Protocol plugins* parse application-layer protocols (e.g., DNS, SSL, and QUIC) and expose extracted fields and events to business plugins. *Business plugins* consume this informa-

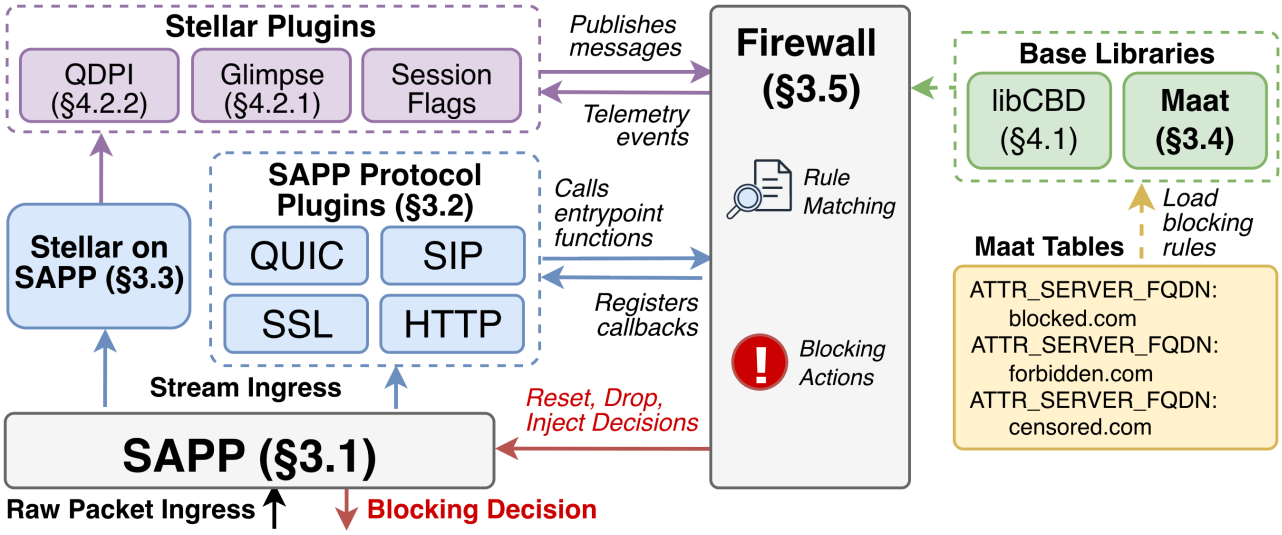

Figure 1: **Tiangou Secure Gateway (TSG) Architecture Overview.** We show the flow of data through the TSG system, spanning raw packet ingress, attribute extraction, application detection, and blocking enforcement.

tion and implement specialized application logic. Note, these plugins are maintained in separate repositories and are distinct from SAPP itself.

Stream Control APIs. Although SAPP itself does not initiate blocking, it provides the underlying functionality required to enforce it. Blocking enforcement actions are mediated through SAPP's `stream.h` header, which defines core stream-manipulation APIs that allow plugins to directly set stream options, send TCP resets, and inject arbitrary packets. Among these, the `MESA_set_stream_opt` API call provides fine-grained control by allowing plugins to modify per-stream options at runtime, with options including `MSO_DROP_STREAM`, which discards all subsequent packets associated with a stream, and the `MSO_DROP_CURRENT_PKT`, which drops only the currently processed packet. Both require TSG to be deployed in-path to stop packets.

3.2 SAPP Protocol Plugins: Stream Ingress

After receiving SAPP streams, protocol plugins perform protocol detection and targeted information extraction. SAPP protocol plugins are responsible for a wide range of attribute extractions. Most protocol plugins are defined in the `MESA_Platform` directory of `mesalab_git`. To better characterize their functionality, we give a high-level overview of some key plugins.

The ***DNS plugin*** registers to UDP SAPP streams. After determining that a stream corresponds to DNS, it extracts information from both queries and responses. In the case of queries, it parses the domain name, type, and class. For responses, it extracts information including all resource and DNSSEC-related records. The ***SSL plugin*** registers to TCP SAPP streams and parses TLS handshakes; specifically, it extracts fields including the SNI, cipher suites, and extension lengths, as well as computes JA3 and JA4 fingerprint hashes. Regarding certificates, it parses fields including the subject, issuer, validity period, and Subject Alternate Names. The ***QUIC plugin*** registers to UDP SAPP streams, parses QUIC handshakes, and extracts data including the SNI, user agent string, and version information.

▶ ***Takeaway:*** *TSG is highly modular. Components are developed and versioned separately—fingerprinting one plugin does not necessarily imply the presence or version of another in the same deployment.*

3.3 Stellar-on-SAPP: Stream Ingress

Stellar-on-SAPP also receives SAPP streams; it adapts them into sessions, which are sent to Stellar plugins for further processing. Stellar-on-SAPP is a SAPP platform plugin that bridges SAPP with Stellar, a separate system developed by Geedge Networks. It is explicitly identified in its README as a transition solution between the two platforms. Stellar-on-SAPP consumes stream data produced directly by SAPP (i.e., *not* by SAPP protocol plugins) and converts it into Stellar sessions, which are abstractions of Layer 4–7 connections.

Compared to SAPP, which is implemented in C, Stellar is written in C++ and offers more stateful stream tracking along with a richer inter-plugin communication model. At the time of the leak, the `stellar` repository appeared to be under active development, although not yet integrated into TSG. It is not referenced in any build repositories and contains no

tagged releases, but it has commits through late 2024, corresponding with the newest files in the leak.

Stellar Plugins. Stellar plugins interact through a session-scoped message bus: plugins can create or locate topics, publish messages to topics, and subscribe to topics via callbacks. This design enables multiple plugins to communicate with each other without additional custom logic, as would be required by SAPP. They perform key functions such as application detection and traffic logging. Key application detection plugins include the QDPI and Glimpse detectors, which are further discussed in Section 4.

3.4 Maat: Rule Setting Engine

Blocking rules are set in Maat, which facilitates defining and scanning against blocking policies. In its README, Maat is described as a "unified description framework for network flow processing configuration." At a high level, Maat is a rule-matching engine: blocking rules are configured and loaded into *Maat tables*, and plugins invoke Maat to scan extracted attributes and obtain the corresponding blocking decision or action. Maat supports three modes for loading rules: Redis, which loads from a Redis server; JSON, which loads from a JSON file; and IRIS, which loads from a combination of a file-based index and data table files. Table 2 shows an example JSON-mode Maat security rule configured to block the domain `blocked.com`.

TSG utilizes a wide range of scan attributes across transport protocols, application protocols, tunnel and encapsulation metadata (e.g., inner/outer IP, GRE endpoint), and endpoint identity fields (e.g., source and destination IPs, ports, FQDN/SNI, ASN, country). Both UDP and TCP have attributes for packet payloads, including specifically the first client-to-server and server-to-client packet payloads.

3.5 Firewall: Administering Blocking

The firewall receives parsed traffic attributes from other plugins and passes blocking decisions. It functions as both a SAPP and Stellar plugin simultaneously. The firewall plugin enforces security policies by scanning packet- and session-level attributes extracted by SAPP and Stellar plugins against Maat rule tables and then applies any resulting blocking decisions using the SAPP stream API.

SAPP Plugin Interface. As a SAPP business plugin, the firewall registers callbacks with each SAPP protocol plugin (e.g., SSL, HTTP, QUIC). When a plugin's callback is invoked, the firewall populates the corresponding scan attributes with the extracted protocol fields and initiates scanning against Maat tables.

Stellar Plugin Interface. The firewall tracks session attributes and subscribes to specific Stellar message topics. For example, at initialization, the firewall subscribes to the application-ID message topic (`TOPIC_APP_ID`). Stellar plugins dedicated to application detection independently publish to this topic; similar to above, upon arrival, the firewall updates its scan attributes and scans against Maat tables.

Field	Value
rule_table_name	`SECURITY_RULE`
uuid	`...00000000A001`
service	`2`
action	`Deny`
blacklist_option	`0`
evaluation_order	`0.0`
action_parameter	
sub_action	`drop`
send_tcp_reset	`1`
after_n_packets	`0`
send_icmp_unreachable	`0`
and_conditions[0]	
negate_option	`false`
attribute_name	`ATTR_SERVER_FQDN`
objects[0]	
expression	`blocked.com`
match_method	`full`
format	`uncase plain`

Table 2: **Maat Security Rule for `blocked.com`.** The bold items are nested objects and indentation reflects JSON nesting depth.

► ***Takeaway:*** *The firewall aggregates attributes from multiple parallel processing paths (SAPP protocol plugins, Stellar detectors) before evaluating blocking decisions.*

3.5.1 Blocking Actions

The firewall supports a wide range of blocking actions that can be specified in Maat rules. In addition to drops and TCP RSTs, it allows for throttling, HTTP and DNS redirects, mail and SIP blocking, tampering, and deferred blocking.

Protocol-Specific. ***HTTP blocks*** can be enacted by specifying a certain code (200, 204, 403, or 404) or a redirect to any site with either a 302 or 303 code. ***DNS redirects*** are also supported. ***SIP blocking*** is achieved by injecting a "SIP/2.0 480 Temporarily Unavailable" or "SIP/2.0 500 Server Internal Error" message. ***Mail blocking*** injects a message with either "550 Mail was identified as spam." or "551 User not local; please try <forward-path>."

Stream-Level. ***Throttling*** is supported at both a session and packet level. The session-level approach initializes a bucket with a set number of bytes per second (`bps`) and then drops packets in accordance with its decision; the packet-level limiter drops packets with a size greater than the configured `bps`. Additionally, ***tampering*** can be done on either every matched packet or variably depending on the provided sampling parameters. The actual tampering action consists

Risk	Applications
5	Atlas VPN; Cyber Ghost VPN; Express VPN; Flash VPN; Hide Me VPN; Hotspot Shield VPN; Ivacy VPN; Nord VPN; Opera VPN; Psiphon-Server; TunnelBear VPN; Turbo VPN; Turbo VPN-Payload; Urban VPN; VPN Unlimited; WARP; Windscribe VPN
4	Gecko VPN; Psiphon-CDN; Psiphon-QUIC; Tor Browser; Ultrasurf VPN
3	Cyber Ghost VPN-Payload; Refraction Networking; WARP-Enhanced

Table 3: **VPN and Circumvention Tool Risk Levels.** We show the risk levels (1–5) for tool signatures included in *VPN Signature File Update Log*.

of swapping the first two and last two bytes of the payload, dropping the original packet, and then crafting and sending another packet with the modified payload. ***Deferred blocking*** allows for firewall rules to be enforced only after the specified number of packets have been processed.

4 Application Detection Systems

In this section, we further examine TSG's application detection systems, as well as some of the associated VPN and circumvention tool fingerprinting efforts.

4.1 Context-Based Detector (CBD)

The Context-Based Detector (CBD), formerly known as AppSketch, is TSG's core application detection system. In earlier releases, AppSketch was implemented as the SAPP plugin `app_sketch_local`, but it has since been fully integrated into the firewall through the CBD library. Distributed via the `libcbd` RPM, this library evaluates application fingerprints, known as *signatures*, loaded from Maat. Built-in signatures are distinguished from user-defined ones by an arbitrary application-ID boundary (default: 10,000), suggesting that TSG ships with a substantial set of pre-installed signatures.

4.1.1 How Does CBD Target Circumvention Tools?

Signature Formats. Throughout the leak, we observe two primary signature formats, ASW (likely AppSketch Works) and TSG2402, which are defined in the `appsketch-works/asw-controller` repository. The ASW format appears to be the storage format for signatures, as `asw-controller` accepts uploads in TSG2402 format, converts them to ASW, and stores them in a Git repository; exports back to TSG2402 are also supported. `appsketch-works/app-test-log` is likely one such storage repository, as it contains signatures in ASW format.

VPN and Circumvention Tool Fingerprints
Express VPN – *expressvpn_ja3* • SSL JA3 fingerprint match
Flash VPN – *flashvpn_cert_issuer* • SSL cert. issuer common name matches `$SUV` • SSL cert. issuer organization name matches `$SUV999`
HideMe VPN – *hidemevpn_openvpn_udp_payload* • UDP first client→server payload length equals 54 bytes • UDP first client→server payload matches `000000016628`
Ultrasurf VPN – *ultrasurfvpn_update_behavior* • Server FQDN matches one of 8 patterns • SSL JA3 fingerprint match
Psiphon – *Psiphon-CDN-SSL* • `common.app_id = 68` or `199` • Server FQDN matches `.com`, `.net`, or `.org` • Matches 1,023 CDN IP ranges • Excludes 56,396 known legitimate FQDNs

Table 4: **Select VPN and Circumvention Tool Signatures.** We highlight a subset of tool signatures included in *VPN Signature File Update Log*. The listed conditions for each are joined with a logical AND.

Correlation with Maat Definitions. CBD loads application signatures from Maat. Although we found no code explicitly converting ASW or TSG2402 formats into Maat signature definitions, structural similarities between the formats strongly suggest such a conversion exists. For instance, the keys `app_id` and `app_name` in TSG2402 exactly match Maat expectations, while the `deny_action` corresponds to Maat's `action_parameter` (see Table 2). Critically, we are able to define and block a custom application through specifications in Maat rules, discussed in Section 5.2.

Select VPN Fingerprints. One of the most detailed bundles of VPN and circumvention tool fingerprints is in the Confluence file "VPN 特征文件更新记录" (or *VPN Signature File Update Log*), which records a log of updates to a signature ZIP from March through November 2024. The signatures are in TSG2402 format and carry risk levels between three and five, as shown in Table 3—the majority (21/29) are ranked five. This stands in sharp contrast to the signatures in `app-test-log`, which target predominantly Chinese applications and all have a risk score of one.

The signatures contain combinations of matchable rule attributes with varying complexities, including server IP addresses, domain names, JA3/JA4 hashes, SSL certificate fields (e.g., issuer name), payload length, and byte patterns. We highlight a subset of the signatures in Table 4. Signatures can also match on the `common.app_id` attribute; we found another JSON, attached to the HTML file "VPN 特征提取记录" (or *VPN Feature Extraction Records*), which includes in its application list many standard network protocols cor-

OMPUB-1372: 【M22 项目】Signal 应用封堵及 LetsVPN/LanternVPN 特征提取需求
[M22 Project] Signal app blocking and LetsVPN/LanternVPN signature extraction requirements

(1) Requirement for blocking the Signal application: when the app enables the "censorship circumvention" option, the blocking is ineffective. Based on experience from the previous Project K, please extract and merge the corresponding signatures. (Higher priority)

(2) LetsVPN signature extraction. (High priority)

+(3) LanternVPN signature extraction: discussed and confirmed with the M client; no specific completion time required (may take a long time). (Low)+

Figure 2: **Iterative Development and Prioritization (Jira Ticket).** This ticket highlights the failure to block Signal traffic when "censorship circumvention" is enabled and the deprioritization of blocking Lantern.

responding to this parameter. For instance, the Psiphon rule in Table 4 specifies values of 68 or 199, which correspond to HTTPS and SSL, respectively.

4.1.2 What Factors Influence Signature Development?

In order to more deeply explore the signature development and support workflow, we investigated related Jira tickets. Specifically, we conducted a case-insensitive search for "vpn", finding 700 tickets across 182 issues. We found major threads of site correspondence with Myanmar (440 tickets) and Ethiopia (53 tickets), extensively detailing continued fingerprinting efforts; frequent topics include reports of server IP extractions, service prioritization, ineffective signatures and iterative development, and instances of over-blocking. We highlight some representative examples below.

Iterative Development and Prioritization. One ticket from the Myanmar site details ineffective blocking of Signal when the "censorship circumvention" option is enabled and highlights previous applicable experience from a "Project K," which likely refers to Kazakhstan, as discussed in Section 2.2. Additionally, the blocking of Signal and LetsVPN is noted as higher priority than Lantern, as per the request of the Myanmar client, as shown in Figure 2.

Over-blocking Concerns. Notably, a ticket describing an incidence of over-blocking in Ethiopia unveils an interesting design for the blocking of Psiphon traffic (see Figure 3). It suggests that Psiphon3 client IPs connecting to a popular SNI should be allowed. This is further corroborated by a Confluence documentation file, "GTN498 The collateral damage analysis of Psiphon3 Blocking." This file establishes the identification of Psiphon3 traffic based on server IP addresses due to the lack of obvious signatures and integration of a "whitelist protection mechanism" to avoid misidentification.

【E21 现场】业主在一个策略里对配置多个 VPN deny 包含 psiphon，针对策略验证效果时多个应用存在也被 deny。业主不满意 VPN deny 策略效果 *[E21 site] Customer configured multiple VPN denys (including Psiphon) in one policy. When validating the policy, several other applications were also denied. Customer is not satisfied with the VPN deny policy behavior.*

In the internal lab env, T9K accessed: tiktok.com/about, bbc.com, edition.cnn.com, nytimes.com.

(1) Test IP not in Psiphon3 Client IP → access OK.

(2) Test IP in Psiphon3 Client IP, server IP not in Top Server IP, SNI not in Top SNI → access denied.

(3) Test IP in Psiphon3 Client IP, SNI in Top SNI → access OK.

Figure 3: **Over-blocking Concerns (Jira Ticket).** Example Jira ticket highlighting concerns of collateral damage from blocking Psiphon by IP addresses and SNI.

► ***Takeaway:*** *Circumvention tool signatures are not developed in isolation. We find evidence of iterative, client-driven processes where customers can report failures, set priorities, and flag collateral damage.*

4.2 Third-Party Detection Systems

For the remaining two application detection systems, the Stellar plugins Glimpse Detector and QDPI Detector, **we find instances of third-party libraries intentionally copied and adapted into the Geedge Networks code base.** We discuss all relevant disclosure procedures and considerations in Ethical Considerations.

4.2.1 Glimpse Detector (libprotoident)

The base protocol classification engine in Glimpse is libprotoident, a free software project from the Libtrace team at the University of Waikato, designed to provide a "very limited form of deep packet inspection," utilizing only the leading four bytes of the application payload in both directions [6]. This is evidenced by the main Stellar session callback subscriber invoking `lpi_guess_protocol`, which is the libprotoident API for protocol identification. All code required for this function contains the libprotoident copyright. Additionally, while libprotoident makes up the majority of application detection capabilities in Glimpse, we highlight additional, more targeted detectors below:

OpenVPN via nDPI. OpenVPN detection is implemented in `openvpn_identify.cpp`, which appears to be a modified copy of `openvpn.c` from a 2022 version of nDPI, a GPL-licensed DPI library. The main function `ndpi_search_openvpn` is renamed to `app_identify_guess_openvpn`, but the logic remains nearly identical.

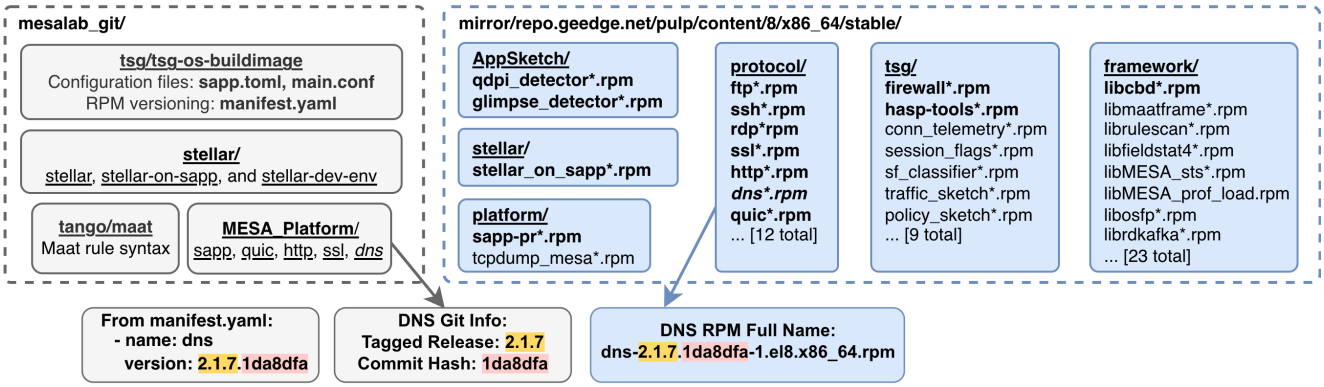

Figure 4: **Build Repositories and RPMs.** We highlight the location of the key repository `tsg-os-buildimage` within `mesalab_git`, as well as the RPMs specified by its `manifest.yaml`. The `mirror/repo.tar` contains static RPM bundle versions including varieties with binary files and source code; there are several RPMs of repositories not included in the repositories in `mesalab_git`. The names of each RPM include a version number and an abbreviated commit hash, which can be correlated across the different locations.

Geedge-Added Functionality. Interestingly, Glimpse implements redundant DNS and QUIC protocol identification in `app_l7_protocol.cpp` and `quic_identify.cpp`, respectively, despite these protocols already being identified and parsed by SAPP protocol plugins. Because Glimpse only publishes an application ID corresponding to the detected protocol to the firewall, it is unlikely that flows are parsed further than strictly necessary for protocol identification. Instead, it can be utilized as part of multi-condition rules that specify protocols, such as those shown in Table 4.

4.2.2 QDPI Detector (Qosmos ixEngine)

Unlike Glimpse, which utilizes free, open-source software, QDPI is built off of Qosmos ixEngine, a proprietary, enterprise DPI library, which reportedly classifies over 4,700 protocols and applications [18]. This is most clearly indicated by the Stellar session callback subscriber function eventually calling `qmdpi_worker_process`, with the inclusion of the associated header `qmdpi.h` appearing below the comment "Qosmos ixEngine header."

The QDPI RPM contains three shared object files: `qdpi_detector.so`, `libqmengine.so`, and `libqmbundle.so`. We find references to the `libqmbundle.so` as a PB ("protocol bundle") file in a Jira ticket, see Figure 8 in the Appendix. This ticket also highlights the strong relationship between QDPI and the AppSketch Database—a resource which is referenced in numerous locations throughout the leak. Interestingly, we find a comment and segment of code in `qdpi_detector_session.cpp` which exits out of detection in the case that the AppSketch database is not loaded. While all available archives of the database in the leak are encrypted, it is still possible to list their contents; notably, we find that the contents of `appsketch_db_24.02.700.0.zip` include `libqmbundle.so.1.700.0-21` and `signatures.pdb.1.700.0-20`.

► ***Takeaway:*** *TSG's reliance on third-party libraries for two of its three detection systems suggests that comprehensive protocol classification is costly to build from scratch, even for a well-resourced, state-linked vendor.*

5 Reconstructing TSG in Practice

In order to more deeply characterize the design of TSG, we prioritized the development of a functional build we could run in an isolated lab environment. In this section, we detail the technical challenges of this effort and highlight key elements of the code base which we identified along the way.

5.1 Building TSG

Building SAPP. To build a functional version of TSG, we look to the manifest file (`manifest.yaml`) in the latest tagged release of the `tsg-os-buildimage` repository (`rel-24.10`). The specified SAPP RPM, `sapp-pr-4.3.67.07feab9`, requires the issuance of a valid license to run. Specifically, we identified a `HASP_ENABLED` flag, enabled at build time, which prompts calls to a wrapper for Sentinel License Development Kit, a commercial software protection solution [40]. This wrapper spawns a monitoring thread that validates the license once per second. We circumvented this check by replacing the original HASP library with a modified wrapper library containing a stub implementation of the validation function.

```
sapp use main config file: ./etc/sapp.toml
[MESA_handle_runtime_log_creation], INIT zlog finish,
Using (etc/sapp_log.conf).
Produced by
    __  _____________ ___
   /  |/  / ____/ ___//   |
  / /|_/ / __/  \__ \/ /| |
 / /  / / /___ ___/ / ___ |
/_/  /_/_____//____/_/  |_|

./plug/protocol/ssl/ssl.so init succ, using [72] us
load SSL success!
./plug/protocol/quic/quic.so init succ, using [855] us
load QUIC success!
./plug/protocol/mail/mail.so init succ, using [1091] us
load MAIL success!
./plug/protocol/http/http.so init succ, using [1329] us
load HTTP success!
./plug/protocol/ftp/ftp.so init succ, using [1840] us
load FTP success!
./plug/protocol/dns/dns.so init succ, using [1320] us
load DNS success!
```

Figure 5: **TSG Startup Logs.** We show the startup logs associated with TSG, including the initialization of select SAPP protocol plugins.

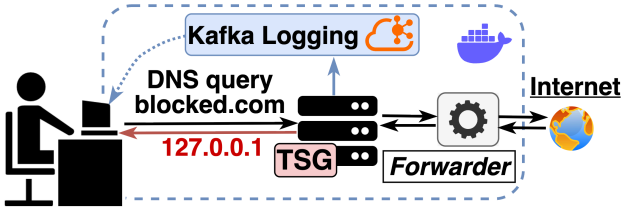

Figure 6: **Docker Test Setup.** Our Docker setup facilitates live testing of TSG in a controlled environment. The forwarder rewrites MAC addresses for routing to the Internet.

Configuring SAPP Plugins. The main configuration file for SAPP is `sapp.toml`, which specifies a variety of parameters, including deployment modes, packet I/O settings, and secondary configuration files. Critically, the latter includes the SAPP plugin configuration file `conflist.inf`, which we use to load SAPP's DNS, SSL, and QUIC plugins.

5.2 Developing a Testbed

We developed a testbed to evaluate TSG's blocking behaviors in a controlled environment. This primarily involved configuring SAPP to support the proper traffic flow setup. SAPP can listen on an interface via the `[packet_io.internal.interface]` table in `sapp.toml`. It also supports both in- and on-path deployment. For the former, it must be run in either `inline` mode, which requires that all packets have a VXLAN overlay or `transparent` mode, which conducts incomplete L2 forwarding such that the MAC addresses are not rewritten.

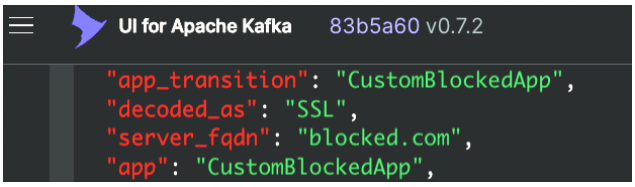

Figure 7: **TSG Apache Kafka UI.** Matches against our custom application as observed in TSG's Kafka UI logs.

Docker and Kafka Setup. Given these constraints, we built a Docker setup consisting of a client container, DPI container, and user-space packet-forwarding container. The client and DPI are both on isolated local (Docker bridge) networks; the packet-forwarding container rewrites source and destination MAC addresses so packets can be routed to the Internet. TSG natively supports integration with Apache Kafka for logging visualizations. We were able to spin up separate Docker containers hosting the necessary Kafka endpoints and configure the firewall to establish a connection.

We were able to successfully test the DNS, HTTP, TLS, and QUIC protocols and trigger injection responses, packet drops, and deferred blocking actions. Through Maat rules, we also defined a custom app specifying an FQDN matching `blocked.com`, as shown in Figure 7.

5.2.1 RST Injector Configurations

We also wanted to further explore the configurability of TSG's RST injector. The firewall also has its own configuration file, `main.conf`.

Flags. Using `main.conf`, TSG can be configured to send one to three RST packets per block decision via the `NUM` option, as well as the flags, either RST or RST+ACK via the `FLAGS` option, where `FLAGS=4` corresponds to a RST and `FLAGS=20` corresponds to a RST+ACK. The default configuration sends one RST+ACK.

IPID, Window, and TTL. TSG has a signature mode, which sets IPID, TCP window, and TTL using two seed parameters. These are set or disabled either through the `sapp.toml`'s `signature_seed1`, `signature_seed2`, and `signature_enabled`, or through `main.conf`'s `SEED1` and `SEED2`. The default configuration is enabled with values of 13 and 65,535.

6 Measurements

Equipped with our functional build of TSG, we explore potential deployments in China, Kazakhstan, Myanmar, Pakistan, and Iran. For China, we separately examine both the GFW and distinct regional censorship in the Henan province, as detailed in [47]. Critically, this is not straightforward, as analysis of measurements must account for the plug-and-play na-

Device	Flags	Compression	17-pointer	18-pointer
TSG	0x8180	✓	✓	✗
Iran	0x81a0	✗	✗	✗
CN Inj. 1	0x8580	✗	✗	✗
CN Inj. 2	0x8180	✓	✓	✗
CN Inj. 3	0x8400	✗	✗	✗

Table 5: **Observed DNS Injector Behavior.** We show the set flags, use of compression pointers in injected responses, and the compression-pointer handling capability boundary for our DNS tests. Note, in our measurements, *Injector 3* has different flags and no longer uses compression pointers, different from the initial observation by Anonymous et al. [9]

Device	Observed Reassembly Boundary
TSG	101
GFW I	500*
GFW II	102
GFW III	0
Henan	0
Kazakhstan	0
Pakistan (RST)	0
Pakistan (DROP)	500*
Myanmar	500*

Table 6: **Observed IP-layer Reassembly Boundaries.** Rows with 500* indicate reassembly was still successful through 500 fragments.

ture of TSG components. For instance, a given deployment may run the same version of SAPP but different versions of the SSL plugin.

In this context, we investigate specific fingerprintable characteristics in the IP, DNS, TCP, TLS, and QUIC protocols. Notably, we find high levels of correspondence with GFW DNS *Injector 2* [9] and TCP *GFW II* and *GFW III* [47], suggesting that the leaked TSG code (or a close derivative) is indeed deployed on part of China's national firewall.

6.1 DNS Injector Behavior

For comparing DNS behavior, we examined how TSG's DNS protocol plugin parses queries and injects responses. We identified a capability boundary in TSG's DNS parsing algorithm, as well as static characteristics of injected response packets that together offer a distinctive fingerprint.

Query Parsing. TSG calls the function `get_decompressed_name` to extract the queried domain name, supporting both uncompressed names (i.e., length-prefixed label sequences) and compression-pointer–based representations as defined in RFC 1035 [1]. Instead of disallowing forward jumps as specified in RFC 1035, the function enforces a fixed limit of 17 pointer jumps per query; exceeding this limit causes domain name extraction to fail. TSG can therefore be fingerprinted by crafting two DNS queries with 17 and 18 pointer jumps: an injection response for the former but not the latter is strongly indicative of TSG.

Response Construction. We also investigated TSG's DNS response injection code contained in the function `build_cheat_pkt`. From this function, we extracted two characteristics that can be used to remotely fingerprint TSG deployments: (1) the injected response has static DNS flags of `0x8180`, and (2) the domain name in the answer section is always encoded using compression pointers.

Measurement. We focus our efforts on evaluating the three well-known DNS injectors of China's GFW, following the naming convention of prior work [9]. We also test against Iran's DNS injector [26, 39]. We first sent normal DNS queries containing censored domains to determine each injector's baseline flags and name-encoding behaviors. We then sent crafted queries containing 17 and 18 successive compression-pointer indirections to test whether their parsers exhibit the same capability boundary observed in TSG.

Our results in Table 5 strongly suggest that TSG provides the GFW Injector 2's DNS injection implementation. Specifically, we show that only China's *Injector 2* exhibits the combination of the flag value `0x8180`, the use of compression pointers in injected responses, and the same pointer-handling capability boundary while parsing DNS queries.

6.2 IP Fragmentation Reassembly

For examining IP-layer behavior, we focus on the IP stack implemented in TSG's SAPP component. Specifically, we focus on the maximum number of IP fragments that can be successfully reassembled and blocked.

IP Reassembly. As shown in Listing 2, TSG reassembles IPv4 fragments using `process_ipv4_frag`, which contains an "attack detection" mechanism that bounds the maximum number of fragments processed per reassembly queue. Due to the ordering of the check and update of the fragment counter, TSG allows up to 101 IPv4 fragments to be processed for a single queue. The first fragment for which the `IPV4_FRAG_NUM_PER_IPQ` threshold is exceeded triggers the `IP_FRAG_IGNORE` state, after which subsequent fragments are ignored.

Measurement. This fragment handling behavior enables remote measurement of IP-layer reassembly limits across different potential deployments. We used Geneva [11] to send fragmented `ClientHello` messages for censored domains with varying numbers of IPv4 fragments. Table 6 summarizes the measured fragment reassembly boundaries.

The observed boundary of 102 fragments for GFW II is very close to TSG's, suggesting the two po-

```
static void trick_algo_getrandval_with_seed(
        int thread_id, unsigned sip, unsigned dip,
        unsigned short *ipid, unsigned short *win,
        u_char *ttl, int seed_key, int seed_max_val) {
    int val = randgroup[thread_id];

    if (val * seed_key > seed_max_val) {
        randgroup[thread_id] = 128;
    } else {
        randgroup[thread_id]++;
    }

    *win = (unsigned short)(val + dip % seed_key);
    if (*win == 0)
        *win = 1;

    *ipid = (unsigned short)(
        seed_max_val - val * seed_key + sip % (*win));
    *ttl  = (u_char)(val % 200 + 48);
}
```

Listing 1: **RST Header Field Generator.** Seed-based generation of IPID, TCP window, and TTL for injected RSTs.

tentially share similar code. Other middleboxes—like *GFW III*, Henan, Kazakhstan, and Pakistan (RST)—have a boundary of zero, indicating IP fragments are simply forwarded to the next hop. This is configurable by the `ipv4_reassembly_enabled` parameter in `sapp.toml`.

6.3 TCP RST Injector Behavior

TSG has code to inject RSTs for blocked connections, for example, when a TLS `ClientHello` contains a blocked SNI. However, TSG contains a custom pseudo-random number generator used for the IPID, TTL, and TCP window size of injected RST packets, enabled by default via the `signature_mode` parameter in TSG's `sapp.toml` (see Section 5.2.1).

Signature Mode. As shown in Listing 1, `trick_algo_getrandval_with_seed` generates the IPID, TTL, and TCP window size of an injected RST packet. The parameters `sip` and `dip` denote the source and destination IP addresses of the TCP stream to be terminated. The parameters `seed_key` and `seed_max_val` are configurable values with defaults of 13 and 65,535.

When `signature_mode` is enabled, the above algorithm couples `IPID`, `win`, and `val` through `seed_key`, making it possible to infer `seed_key` from the IPID, TTL, and TCP window fields of injected RST packets and potentially fingerprint TSG. However, due to the instability of TTL, we focus only on IPID and TCP window.

Ignoring the corner case where `win == 0`, the algorithm reduces to the following, where $k =$ `seed_key` and $M =$ `seed_max_val`:

$$\begin{aligned} \text{win} &= val + (dip \bmod k), \\ \text{ipid} &= M - val \cdot k + (sip \bmod \text{win}), \end{aligned}$$

Device	IP DF	TCP Flags	Count	Stable k
TSG	✓	RST+ACK*	1-3	yes ($k = 13$)*
GFW I	✗	RST	1	no
GFW II	✓	RST+ACK	3	yes ($k = 13$)
GFW III	✓	RST+ACK	1	yes ($k = 13$)
Henan	✗	RST+ACK	1	no
Myanmar	✓	RST+ACK	1	no
Pakistan	✗	RST+AE+RE	1	no

Table 7: **Observed TCP Injection Fingerprints.** "Count" reports the number of forged RST packets emitted per side in our tests. "Stable k" indicates whether pairwise differential estimation yields a unique and consistent integer key; "no" also includes cases where TCP window values remain identical all the time rendering k undefined. *Note, TSG can be configured to send RSTs and not use a seed key.

Given two forged packets i and j with distinct IPID and window values, k can be derived by solving the system of equations:

$$k = \frac{(\text{ipid}_i - \text{ipid}_j) + \big((sip \bmod \text{win}_j) - (sip \bmod \text{win}_i)\big)}{\text{win}_j - \text{win}_i}.$$

Static Characteristics. TSG hardcodes the setting of the IP Don't Fragment (DF) bit in `deal_ipv4.h`. Additionally, TSG can be configured to send 1–3 RST or RST+ACK packets, as confirmed in Section 5.2.1.

Measurement. We evaluated three TCP RST injectors previously attributed to the GFW, following the naming convention of Wu et al. [47], as well as others in Henan, Myanmar, and Pakistan. We did not test in Kazakhstan because its censors do not generate TCP RST packets. We analyzed injected RST packets, recording the IP DF bit, TCP flags, the packet count per injection event, and the stability of the seed key k. To test the stability of k, we elicited many TCP RST packets from each injector, estimated k from every pair of distinct packets, and checked whether k is consistent.

As shown in Table 7, GFW II and GFW III consistently yielded $k = 13$, corresponding to TSG's default `seed_key`, strongly indicating usage of TSG code. However, both *GFW II* and *GFW III* exhibit deviations from TSG. Both fix TTL values at 255, but TSG generates TTL values dependent on an internal state variable `val`, and ranging from 48 to 247. Notably, looking at previous measurements collected in December 2023, *GFW II* did exhibit TTL values consistent with TSG, suggesting that it slightly changed its implementation since then. Additionally, *GFW II* is known to send 3 identical RST+ACK packets [47], while our build of TSG, when configured to also send 3 RST+ACK packets per injection, decreases the IPID by approximately 13. Still, the likelihood

	TSG	GFW I	GFW II	GFW III	Henan	Myanmar	Kazakhstan	Pakistan (RST)	Pakistan (DROP)
Record Length	●	●	●	●	●	●	●	✗	✗
Sess. ID / CS Length	●	●	●	●	●	●	✗	●	✗
Comp. Methods Length	●	●	●	●	●	●	✗	✗	✗
SNI Name Length	●	●	●	✗	●	●	▼	●	✗
SNI Ext. Length	●	✗	●	●	▼	●	✗	✗	✗
Message Length	▼	▼	▼	●	✗	●	✗	✗	✗
Extensions Length	✗* / ▼	▼	✗	▼	✗	▼	✗	▼	✗
SNI List Length	✗	✗	✗	✗	✗	●	✗	✗	✗

● Bypasses Censor ✗ Blocked ▼ Varied

Table 8: **TLS Parser Fingerprints.** We show a condensed view of TLS parsing fingerprints for TSG and several middleboxes potentially running TSG's SSL plugin—extended results in Table 12. *Note, this blocked outcome corresponds to an older version of TSG, which aligns with the observed parsing behavior of GFW II.

of exhibiting a constant seed key by random chance is approximately 10^{-44}, let alone sharing TSG's default value of 13.

In contrast, the other injectors exhibit markedly different behaviors and header values. While we did not observe the seed key correspondence in other regions, TSG can be configured to match the behavior in Myanmar, which sets the IP DF bit, uses supported RST+ACK flags, and sends 1 packet.

6.4 TLS Length Field Manipulation

For TLS, we compare the parsing behavior of TSG's SSL plugin to that of deployed middleboxes. Specifically, we fingerprint how TLS `ClientHello` parsing handles length fields, which are prone to ambiguities because some fields are redundant. Our fingerprinting results are depicted in Table 12.

Length Permutations. In our evaluations, where applicable, we set each `ClientHello` message length field to zero, half its correct value, twice its correct value, and its maximum value. For each of these messages, which do not conform to standards, we tested if TSG would parse the `ClientHello` and respond by blocking the connection. Because the Session ID Length is zero by default, we tested only a non-zero value. For the Message Length and Extensions Length fields, we additionally tested a length that excluded the final extension. We created our test vectors by extending the open-source tool Censor-Scanner [29].

Measurement. We sent `ClientHello` messages containing an SNI extension with a censored domain and a single permuted length field, as described above. Each length permutation was sent 20 times from a vantage point in the respective country to a controlled vantage point in Europe. We recorded censorship behavior, such as injected TCP RST packets or packet dropping, and classified a length permutation as censored only if known middlebox behavior was observed in at least two-thirds of trials; we observed consistent results across two full sets of measurements. To accommodate 3-tuple residual censorship, we waited for the residual censorship duration between tests; to accommodate 4-tuple residual censorship, we used fresh client ports during the residual censorship timeout. Note, we can highlight comparisons with test vectors that are successfully blocked by TSG in cases of non-fingerprintable censorship decisions. This is because, in the case of additional middleboxes besides a TSG device on the path, if a test vector evades TSG, it could still get blocked by another middlebox, but if it is blocked by TSG, it is guaranteed to be blocked.

In Table 8, we show a consolidated breakdown of our TLS fingerprinting test vector results. While no middlebox exhibited exactly the same behavior as the latest leaked TSG, *GFW II* shows behavior equivalent to an older version of TSG's SSL module, which suggests usage of this older version in *GFW II*'s SSL parsing. Additionally, we note that both Kazakhstan and Pakistan (DROP) support potential TSG deployments, as their generic censorship mechanism of packet drops makes bypass failures inconclusive. We show an extended breakdown of these results in Table 12, including the corresponding commits in the source code.

6.5 QUIC Fingerprints

We also identified fingerprintable behaviors in TSG's QUIC plugin, including fragmentation reassembly, 1200-byte minimum size enforcement, and specific version support.

QUIC Parsing. TSG follows the QUIC specification in enforcing the 1200-byte minimum QUIC Initial datagram size (i.e., TSG only parses QUIC Initial packets of at least 1200 bytes). TSG also supports pre-standardization QUIC drafts, including Google QUIC versions. TSG can parse both plaintext and encrypted QUIC Client Initial packets, as well as reassemble fragmented CRYPTO frames, which may span multiple UDP datagrams.

Measurement. We constructed additional test vectors for QUIC, targeting specific version support, QUIC fragmentation reassembly, 1200-byte minimum size enforcement, forced negotiation, and sending the Initial message in multiple packets, as well as utilizing the same set of length per-

Field	TSG	GFW	KZ	MM	PK
QUIC Version					
GQUIC: Q023–40, 42–43	✗	✗	✗	✗	✗
GQUIC: Q041, 44–45	●	●	✗	●	✗
GQUIC: Q046–49	●	●	✗	●	●
GQUIC: T050–51	✗	●	●	✗	✗
IETF: draft-22–28	✗	●	●	✗	✗
IETF: draft-29, v1	✗	✗	✗	✗	✗
IETF: v2	●	●	●	●	✗
QUIC Features					
Fragmentation	✗	✗	✗	✗	✗
Below 1200 Bytes	●	✗	✗	●	●
Forced Negotiation	●	●	●	●	●
Multi-Packet Initial	●	●	●	●	●
TLS Length Fields					
Message Length	✗	✗	✗	●	✗
Extensions Length	✗	✗	✗	✗	✗
SNI Extension Length	✗	✗	✗	●	●
SNI List Length	✗	✗	✗	●	●

● Bypasses Censor ✗ Blocked

Table 9: **QUIC Parser Fingerprints.** We include the blocked permutations for TSG's QUIC module (TSG).

mutations described above—wrapping the `ClientHello` messages in a QUIC CRYPTO Frame and an Initial Packet. We utilize the same procedure detailed in Section 6.4.

For QUIC parsing, we see that both Kazakhstan and the GFW block the same test vectors as TSG's QUIC plugin, though we cannot match them fully due to a difference in supported QUIC versions (see Table 9). Interestingly, the parsing fingerprint exhibited in Kazakhstan fully matches an older version of TSG's QUIC parsing code (see Table 12). We also find the supported QUIC versions in Myanmar align exactly with those supported by TSG's QUIC, though the parsing fingerprint mismatch means we cannot confirm its use.

7 Discussion and Conclusion

In this work, we present the first technical analysis of the Geedge Networks leak. Through this extensive investigation, we characterize the foundational architecture of TSG and uncover the nature of some of its core systems, including SAPP, Maat, and the firewall plugin. We also reconstruct and run a locally deployed build, which substantiates and enriches our knowledge of TSG, especially with regard to its configurability and rule definition syntax. For instance, in the case of RST packets, we are able to configure their number, flags, and the nature of IPID/window/TTL values. We hope this work will serve as a guide for future research on the leak. We highlight several key insights below, which we believe are relevant to the Internet freedom and network security communities.

First, a particularly valuable contribution of this work is the insight it provides into a censor's mental model. We find a common thread of aggregation across multiple pathways. In TSG, packet data takes several parallel flows through the system before aggregating at the firewall plugin. For instance, Glimpse could identify a flow as QUIC, and the firewall could subsequently block the flow due to a rule disallowing QUIC traffic to a certain IP address—all fully separate from the QUIC plugin, which facilitates QUIC SNI parsing. Censorship research has long acknowledged the abstraction of multiple middleboxes on a given path, but this notion of disjoint, co-occurring processes within a single middlebox brings further complexity previously unexplored.

Next, the leak is inextricably linked to the GFW, a three-decades-old censorship apparatus at the forefront of technical innovation. As such, it can serve as a valuable anchor point for long-held assumptions about many aspects of censorship—including previously ignored protocols and adversarial abilities against obfuscation. TSG is capable of identifying circumvention tools at a relatively low cost. Their application detection pipeline affords high flexibility, facilitating matching across a wide range of flow attributes—often very early in the connection. At the same time, we also find evidence of obfuscation raising the cost of detection. As discussed in Section 4.1.2, Psiphon was stated to lack obvious signatures, so fingerprinting efforts were refocused to employ IP-based restrictions in conjunction with a "whitelist protection mechanism" allowing traffic to popular SNIs. Understanding the utilization of these multi-faceted strategies can inform circumvention tool development and broader obfuscation research.

Finally, TSG's complexity and flexibility are a double-edged sword: while they pose a significant obstacle to circumvention, they also undermine code maintainability and correctness. We have witnessed firsthand the ad hoc and non-robust composition of TSG: core components such as SAPP and its protocol plugins are written in memory-unsafe C; the system continues to rely on transitional solutions like Stellar-on-SAPP; and code is copied from several third parties. The Jira tickets corroborate this patchwork development process; we detail an example of an upgrade to the AppSketch database causing SAPP to restart, underscoring a lack of code verification. Together, these reinforce the potential for future work in fuzzing or symbolic execution. Overall, we hope this work can assist circumvention tool developers and Internet security researchers in further understanding the abilities and limitations of modern censorship.

Acknowledgments

The authors are grateful to the anonymous reviewers for their helpful feedback. We thank the Open Technology Fund for their support. Nico Heitmann was supported by the research project "North-Rhine Westphalian Experts in Research on Digitalization (NERD II)", sponsored by the state of North Rhine-Westphalia – NERD II 005-2201-0014. Niklas Niere was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 555828767. Ali Zohaib,

Jade Sheffey, and Amir Houmansadr were supported by the Young Faculty Award program of the Defense Advanced Research Projects Agency (DARPA) under the grant DARPA-RA-21-03-09-YFA9-FP-003. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

Ethical Considerations

There are several components of our research where we considered ethical risks for conducting this work, and in deciding to study this data in the first place. Our goal with this work is to reveal the technical workings of national Internet censorship systems and help uncover new techniques to circumvent them. In an attempt to achieve this goal, we have made careful decisions about what information to publish and what to keep private from this work, which we detail below.

Stakeholders. We identify the following stakeholders impacted by this research: (1) employees and shareholders of Geedge Networks, whose internal data and proprietary code were exposed (*Geedge employees and shareholders*); (2) employees and shareholders of companies whose code is integrated in TSG (*third-party companies*); (3) the government of China and countries purported to purchase TSG devices (*deploying countries*); (4) Internet users in China and countries deploying TSG devices, as well as those in countries that could deploy them in the future (*users under censorship*); and (5) anti-censorship researchers, developers, and activists (*anti-censorship community*).

Use of Leaked Data. We consider the risks and potential harms in studying leaked data from a private company. For *Geedge employees and shareholders,* the main potential risk involves the personal information present in the leak. For instance, the git repositories contain commits and directories that list emails and names of individual developers, and this information could be used to identify them. However, we emphasize that our focus is on the systems and architecture, and not on individual persons. We seek to reduce this risk and do not report on individuals in this paper, and do not include source material in our repository. For *third-party companies,* we recognize that our work may draw additional attention. We disclosed the usage of code by Geedge Networks to both the Libprotoident team and Enea, the parent company of Qosmos. Both have responded to our initial disclosures, and we have scheduled meetings to engage in further discussion.

Firewall Deployment Risk. We also consider that by studying this firewall we could potentially make it easier to use this system. Specifically, our efforts to run a local operational copy of TSG could harm *users experiencing censorship* by enabling *deploying countries* to refine their systems or other censors to more easily deploy it themselves. To mitigate this risk, we choose not to publicly document or provide additional details on how to set up this firewall. We exclude this information from our published repository, instead focusing on information that is useful in fingerprinting these tools rather than direct operation or deployment.

We also considered whether it would be better to not deploy the firewall ourselves at all, as demonstrating feasibility could encourage others to do the same. We concluded that operating a local copy enables the *anti-censorship community* to fingerprint deployments, develop and test circumvention strategies against a working instance, and generalize those strategies across multiple national firewalls—benefits that would not be possible without a working deployment.

Measurement Risks. We designed our measurement methodology to have minimal impact on *users under censorship* in the analyzed countries as well as other third-party servers. We performed all of our scans between two vantage points that we own, not including any other third-party server as the target of our scan. During the renting process of all of our vantage points, we made sure that no entity is present on the current EU sanctions list.

Decision to Publish. Compared to misinformation and other instruments governments use to limit free expression, Internet censorship is an actionable adversary. As discussed in Section 2, censorship restricts fundamental rights and causes concrete harms, including suppressing independent journalism, restricting access to public-health information, and isolating activists and minority communities. Censorship circumvention reduces these harms by facilitating increased access to information. Correspondingly, the overarching benefit of this work falls on *users under censorship* and the *anti-censorship community*. By revealing the technical workings of TSG and developing fingerprinting and circumvention techniques, this work contributes to ongoing efforts to defend the right to information access. We believe these benefits outweigh the potential harms detailed above, which we have sought to mitigate.

Open Science

In the context of the open science policy, we made all of our artifacts available in the following Zenodo link: `https://zenodo.org/records/20274003`. It contains all of our obtained result files, including anonymized PCAPs, as well as all of the scripts and tools that we used to conduct our scans. We have a README at the top-level that details our artifact and folder structure. In the following, we detail all parts of our artifact.

Length Field Scanner. The tool that we used to conduct the length field scan (Table 12) is located in the folder `tls_length_field_manipulations/scanner`. It contains a `README` that details how to setup and use the

tool. To ease reproducibility, we also provide all shell scripts here that we used on our vantage points. Note, we created our test vectors by extending the open-source tool CensorScanner [29].

Length Field Scan Results. The results of our length field scan (Table 12) are contained in the folder `tls_length_field_manipulations/results`. This folder contains subfolders corresponding to our vantage points and scanned protocols. Each country-result folder contains results for two test runs with 20 repetitions each, as well as two ground-truth test runs that were executed in between. Each result consists of all anonymized PCAPs, aggregated results in a JSON file, the output of the tool as a txt file, and the aggregated results as a csv file. A mapping of anonymized IPs and locations can be found in the top-level README file.

QUIC Versions. The folder `quic_fingerprints` contains all of the recorded probes of our QUIC version scan. It covers all QUIC versions from Table 12. We sent each of the probes using netcat and recorded and analyzed each PCAP to determine if there was blocking for a given probe.

IP Fragmentation Reassembly. The folder `ip_fragmentation_reassembly` contains the PCAPs of all of our IP reassembly scans, depicted in Table 6. The folder `ipfragtest` contains Geneva [11], which we used to generate fragmented packets for measurement.

DNS Injection Behavior. The PCAPs corresponding to our results from Table 5 can be found in the folder `dns_injection_behavior`. The tool that we used to create the DNS messages is contained in the underlying `compress-crafter` folder.

TCP RST Injection Behavior. The folder `tcp_rst_injection_behavior` contains all PCAPs of our TCP RST injection fingerprinting scan. These are the results contained in Table 7. The underlying `getkey` folder contains the tool that we used to measure this.

Reconstructing TSG in Practice. We decided against the publication of the leaked source code and our deployment of the Firewall due to ethical concerns. We consider the risk of censors benefiting from such a publication greater than the potential benefit to the community. To ensure that our artifact is only shared with responsible entities not aligned with censor deployment, we do not provide the artifact openly. Instead, we will share it with researchers or other entities that aim to contribute to the community.

References

[1] Domain names - implementation and specification. RFC 1035, 1987.

[2] Leaked files show a chinese company is exporting the great firewall's censorship technology, September 2025.

[3] Pakistan: Shadows of control: Censorship and mass surveillance in pakistan, September 2025.

[4] Silk road of surveillance: The role of china's geedge networks and myanmar telecommunications operators in the junta' s digital terror campaign, September 2025.

[5] Madlink: A taiwanese vestige in the geedge supply chain. Technical report, InterSecLab, April 2026.

[6] Shane Alcock, Donald Neal, Aaron Murrihy, Paweł Foremski, Fabian Weisshaar, Jeroen Roovers, Jiri Havranek, Romain Fontugne, and Jacob van Walraven. Libtraceteam/libprotoident. https://github.com/LibtraceTeam/libprotoident, 2024.

[7] Alice, Bob, Carol, Jan Beznazwy, and Amir Houmansadr. How China detects and blocks Shadowsocks. In *Internet Measurement Conference*. ACM, 2020.

[8] Amnesty International. Pakistan: Mass surveillance and censorship machine is fueled by chinese, european, emirati and north american companies, September 2025.

[9] Anonymous, Arian Akhavan Niaki, Nguyen Phong Hoang, Phillipa Gill, and Amir Houmansadr. Triplet censors: Demystifying Great Firewall's DNS censorship behavior. In *Free and Open Communications on the Internet*. USENIX, 2020.

[10] Simurgh Aryan, Homa Aryan, and J. Alex Halderman. Internet censorship in Iran: A first look. In *Free and Open Communications on the Internet*. USENIX, 2013.

[11] Kevin Bock, George Hughey, Xiao Qiang, and Dave Levin. Geneva: Evolving censorship evasion strategies. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 2199–2214, 2019.

[12] Kevin Bock, Gabriel Naval, Kyle Reese, and Dave Levin. Even censors have a backup: Examining China's double HTTPS censorship middleboxes. In *Free and Open Communications on the Internet*. ACM, 2021.

[13] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. Ignoring the Great Firewall of China. In *Privacy Enhancing Technologies*, pages 20–35. Springer, 2006.

[14] Der Standard. Wie china seine great firewall ins ausland exportiert. *Der Standard*, September 2025.

[15] Luca Deri, Maurizio Martinelli, Tomasz Bujlow, and Alfredo Cardigliano. nDPI: Open-source high-speed deep packet inspection. In *2014 International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 617–622, 2014.

[16] Haixin Duan, Nicholas Weaver, Zongxu Zhao, Meng Hu, Jinjin Liang, Jian Jiang, Kang Li, and Vern Paxson. Hold-On: Protecting against on-path DNS poisoning. In *Securing and Trusting Internet Names*. National Physical Laboratory, 2012.

[17] Arun Dunna, Ciarán O'Brien, and Phillipa Gill. Analyzing China's blocking of unpublished Tor bridges. In *Free and Open Communications on the Internet*. USENIX, 2018.

[18] Enea AB. Qosmos ixEngine: Next-generation DPI engine for traffic visibility. https://www.enea.com/solutions/deep-packet-inspection-traffic-intelligence/dpi-engine/, 2024.

[19] Roya Ensafi, David Fifield, Philipp Winter, Nick Feamster, Nicholas Weaver, and Vern Paxson. Examining how the Great Firewall discovers hidden circumvention servers. In *Internet Measurement Conference*. ACM, 2015.

[20] Roya Ensafi, Philipp Winter, Abdullah Mueen, and Jedidiah R. Crandall. Analyzing the Great Firewall of China over space and time. *Privacy Enhancing Technologies*, 2015(1), 2015.

[21] Jeremy Goldkorn. Fang binxing and the great firewall. In G. R. Barmé and J. Goldkorn, editors, *China Story Yearbook 2013: Civilising China*. Australian Centre on China in the World, 2013.

[22] Nguyen Phong Hoang, Jakub Dalek, Masashi Crete-Nishihata, Nicolas Christin, Vinod Yegneswaran, Michalis Polychronakis, and Nick Feamster. GFWeb: Measuring the great firewall's web censorship at scale. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 2617–2633, Philadelphia, PA, August 2024. USENIX Association.

[23] International Crisis Group. Myanmar's military struggles to control the virtual battlefield, May 2021.

[24] InterSecLab. The internet coup: A technical analysis on how a chinese company is exporting the great firewall to autocratic regimes. Technical report, InterSecLab, sep 2025.

[25] Mohamed Shalman Kursheeth K, Thinnavalli Sree, Sendil Vadivu D, Y Sai Sri Harsha, and Narendran Rajagopalan. Suricata-based intrusion detection and isolation system for local area networks. *2024 International Conference on Signal Processing, Computation, Electronics, Power and Telecommunication (IConSCEPT)*, 2024.

[26] Felix Lange, Niklas Niere, Jonathan von Niessen, Dennis Suermann, Nico Heitmann, and Juraj Somorovsky. I(ra)nconsistencies: Novel insights into Iran's censorship. In *Free and Open Communications on the Internet*, 2025.

[27] Graham Lowe, Patrick Winters, and Michael L. Marcus. The great DNS wall of China. Technical report, New York University, 2007.

[28] Follow The Money. China exports censorship tech to authoritarian regimes –aided by eu firms. *Follow the Money*, September 2025.

[29] Niklas Niere, Felix Lange, Robert Merget, and Juraj Somorovsky. Transport layer obscurity: Circumventing SNI censorship on the TLS layer. In *Symposium on Security & Privacy*. IEEE, 2025.

[30] Vern Paxson. Bro: A system for detecting network intruders in real-time. In *Proceedings of the 7th USENIX Security Symposium*. USENIX Association, 1998.

[31] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, 1998.

[32] Sarvin Qalandar, Philip Engelbutzeder, Dave Randall, and Volker Wulf. From using to infrastructuring: Grassroots vpn-building in iran's women–life–freedom movement. In *Proceedings of the 2026 CHI Conference on Human Factors in Computing Systems*, page 30, New York, NY, USA. ACM.

[33] Ram Sundara Raman, Prerana Shenoy, Katharina Kohls, and Roya Ensafi. Censored Planet: An Internet-wide, longitudinal censorship observatory. In *Computer and Communications Security*. ACM, 2020.

[34] Ram Sundara Raman, Adrian Stoll, Jakub Dalek, Reethika Ramesh, Will Scott, and Roya Ensafi. Measuring the deployment of network censorship filters at global scale. In *Network and Distributed System Security*. The Internet Society, 2020.

[35] Raymond Rambert, Zachary Weinberg, Diogo Barradas, and Nicolas Christin. Chinese wall or Swiss cheese? keyword filtering in the Great Firewall of China. In *WWW*. ACM, 2021.

[36] Reethika Ramesh, Ram Sundara Raman, Matthew Bernhard, Victor Ongkowijaya, Leonid Evdokimov, Anne Edmundson, Steven Sprecher, Muhammad Ikram, and Roya Ensafi. Decentralized control: A case study

of Russia. In *Network and Distributed System Security*. The Internet Society, 2020.

[37] Reethika Ramesh, Ram Sundara Raman, Apurva Virkud, Alexandra Dirksen, Armin Huremagic, David Fifield, Dirk Rodenburg, Rod Hynes, Doug Madory, and Roya Ensafi. Network responses to Russia's invasion of Ukraine in 2022: A cautionary tale for Internet freedom. In *USENIX Security Symposium*. USENIX, 2023.

[38] Martin Roesch. Snort: Lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration (LISA)*. USENIX Association, 1999.

[39] Jonas Tai, Karthik Nishanth Sengottuvelavan, Peter Whiting, and Nguyen Phong Hoang. IRBlock: A large-scale measurement study of the Great Firewall of Iran. In *USENIX Security Symposium*. USENIX, 2025.

[40] Thales Group. Sentinel LDK: Software License Development Kit. https://cpl.thalesgroup.com/software-monetization/license-development-kit-ldk, 2025.

[41] Lokman Tsui. An Inadequate Metaphor: The Great Firewall and Chinese Internet Censorship. *Global Dialogue*, 2007.

[42] United Nations. Universal declaration of human rights. General Assembly Resolution 217 A (III), December 1948.

[43] Zhongjie Wang, Yue Cao, Zhiyun Qian, Chengyu Song, and Srikanth V Krishnamurthy. Your state is not mine: A closer look at evading stateful internet censorship. In *Proceedings of the 2017 Internet Measurement Conference*, 2017.

[44] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is blocking Tor. In *Free and Open Communications on the Internet*. USENIX, 2012.

[45] wkrp. VPN blocking in Myanmar since 2024-05-30 reportedly implemented by a Chinese company, Geedge Networks. https://github.com/net4people/bbs/issues/369, 2024.

[46] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin, and Eric Wustrow. How the Great Firewall of China detects and blocks fully encrypted traffic. In *USENIX Security Symposium*. USENIX, 2023.

[47] Mingshi Wu, Ali Zohaib, Zakir Durumeric, Amir Houmansadr, and Eric Wustrow. A Wall Behind A Wall: Emerging Regional Censorship in China. In *2025 IEEE Symposium on Security and Privacy (SP)*, pages 1363–1380. IEEE, 2025.

[48] Diwen Xue, Armin Huremagic, Wayne Wang, Ram Sundara Raman, and Roya Ensafi. Fingerprinting deep packet inspection devices by their ambiguities. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2025.

[49] Diwen Xue, Benjamin Mixon-Baca, ValdikSS, Anna Ablove, Beau Kujath, Jedidiah R. Crandall, and Roya Ensafi. TSPU: Russia's decentralized censorship system. In *Internet Measurement Conference*. ACM, 2022.

[50] Ali Zohaib, Qiang Zao, Jackson Sippe, Abdulrahman Alaraj, Amir Houmansadr, Zakir Durumeric, and Eric Wustrow. Exposing and circumventing SNI-based QUIC censorship of the Great Firewall of China. In *USENIX Security Symposium*. USENIX, 2025.

A Appendix

P-POC 现场 APP 第三方识别引擎库 (libqmbundle) 出现 segmentfault 导致 SAPP 重启
Segmentation fault in P-POC on-site APP third-party recognition engine library (libqmbundle) caused SAPP to restart

Initial suspicion. The PB version in the `APP_sketch_db` upgrade package has issues.

Update history. The PB updated in `App_sketch_db_23.08` is `1.670-0-21`; the PB in `App_sketch_db_23.09` was not updated.

Solution. Use the latest `App_sketch_db_23.10` upgrade package; the PB version is `1.680.0.20`.

Figure 8: **AppSketch DB (Jira Ticket).** This ticket highlights a relationship between a "third-party recognition engine library" and the AppSketch DB.

ASN	Location
4837	Zhengzhou, Henan, China
45090	Beijing, China
209847	Almaty, Kazakhstan
58061	Almaty, Kazakhstan
150297	Yangon, Myanmar
135307	Yangon, Myanmar
138915	Karachi, Pakistan
150315	Islamabad, Pakistan

Table 10: **ASNs and Locations of Vantage Points.**

Technique	**SAPP TLS** (v3.2.1) MESA_Platform/ssl *src/SSL_Message.c*	**SAPP QUIC** (v2.0.12) MESA_Platform/quic *src/quic_process.cpp*
Record Length	L1566, L1569	—
Message Length	L398, L405[a]	L823
Session ID Length	L431	L446
Cipher Suite Length	L439	L453
Compression Methods Length	L448	L460
Extensions Length	L458, L460[b]	L467
SNI Extension Length	L464	L491
SNI List Length	L315	L404
SNI Name Length	L325, L329	L429

SAPP TLS: version 3.2.1 (c3614b5b) SAPP QUIC: version 2.0.12 (513b9b52)
[a] Changed in commit 21950877 on 2023-07-03.
[b] Changed in commit 482b1ac9 on 2024-09-13.

Table 11: **Code references for TLS parser fingerprints in** Table 12.

```
#define IPV4_FRAG_NUM_PER_IPQ     (100)
static int ipv4_attack_detect(struct frag_ipq *
    ipv4_qp)
{
    ...
  if(ipv4_qp->frags_num > IPV4_FRAG_NUM_PER_IPQ)
  {
    ipv4_qp->attack_flag |= IP_FRAG_IGNORE;
    return IP_FRAG_IGNORE;
  }

  return IP_FRAG_OK;
}

static /*@null@*/ struct mesa_ip4_hdr *
    process_ipv4_frag(
        struct streaminfo_private *pfstream_pr,
        const struct mesa_ip4_hdr *a_packet,
        int thread_num,
        const raw_pkt_t *raw_pkt)
{
    attack_flag = ipv4_attack_detect(ipv4_qp);
    ...
    switch(attack_flag)
    {
        ...
        case IP_FRAG_IGNORE:
        // no break here!
        default:
            return NULL;
    }
    ...
    ipv4_qp->frags_num++;
    ...
}
```

Listing 2: **IP Fragmentation Reassembly Boundary.**

Table 12: TLS and QUIC Parser Fingerprints.

Technique	TLS									QUIC				
	TSG	**GFW II**	KZ	MM	PK DROP	PK RST	GFW I	GFW III	Henan	**TSG**	KZ	MM	PK	GFW
Record Length														
Zero / Half / Double / `0xFFFF`	●	●	●	●	○	○	●	●	●	–	–	–	–	–
Message Length														
Zero	●	●	○	●	○	○	●	●	○	○	○	●	○	○
Double	●	●	○	●	○	○	○	●	○	○	○	●	○	○
Half	○	●[IV]	○	●	○	○	●	●	○	○	○	●	○	○
`0xFFFF`	●	●	○	●	○	○	○	●	○	●	○[I]	●	○	○
Exclude Last Extension	○	○	○	●	○	○	○	●	○	○	○	●	○	○
Session ID Length														
Too Long	●	●	○	●	○	●	●	●	●	●	●	○	○	●
Cipher Suite Length														
Double	●	●	○	●	○	●	●	●	●	●	●	●	○	●
Zero / Half / `0xFFFF`	●	●	○	●	○	●	●	●	●	●	●	○	○	●
Compression Methods Length														
Zero / Half	●	●	○	●	○	○	●	●	●	●	●	●	○	●
Double / `0xFF`	●	●	○	●	○	○	●	●	●	●	●	○	○	●
Extensions Length														
Zero	●	○[V]	○	○	○	●	●	●	○	●	●	○	○	●
Half	●	○[V]	○	○	○	●	○	○	○	○	○	○	○	○
Double / `0xFFFF`	○	○	○	●	○	○	○	○	○	●	○[II]	○	○	○
Exclude Last Extension in Length	○	○	○	○	○	○	○	○	○	○	○	○	○	○
SNI Extension Length														
Zero / Half	●	●	○	●	○	○	○	●	○	●	●	●	●	○
Double	●	●	○	●	○	○	○	●	○	○	○	●	●	○
`0xFFFF`	●	●	○	●	○	○	○	●	●	●	●	●	●	●
SNI List Length														
Zero	○	○	○	●	○	○	○	○	○	●	●	●	●	○
Half	○	○	○	●	○	○	○	○	○	○	○	●	●	○
Double	○	○	○	●	○	○	○	○	○	●	○[III]	●	●	○
`0xFFFF`	○	○	○	●	○	○	○	○	○	●	○[III]	●	●	●
SNI Name Length														
Zero / Half	●	●	●	●	○	●	●	○	●	●	●	●	●	●
Double / `0xFFFF`	●	●	○	●	○	●	●	○	●	●	●	●	●	●
QUIC Version														
GQUIC: Q023–Q040, Q042–Q043	–	–	–	–	–	–	–	–	–	○	○	○	○	○
GQUIC: Q041, Q044–Q045	–	–	–	–	–	–	–	–	–	●	○	●	○	●
GQUIC: Q046–Q049	–	–	–	–	–	–	–	–	–	●	○	●	●	●
GQUIC: T050–T051	–	–	–	–	–	–	–	–	–	○	●	○	○	●
IETF: draft-22–28	–	–	–	–	–	–	–	–	–	○	●	○	○	●
IETF: draft-29, v1	–	–	–	–	–	–	–	–	–	○	○	○	○	○
IETF: v2	–	–	–	–	–	–	–	–	–	●	●	●	○	●
QUIC Features														
Fragmentation	–	–	–	–	–	–	–	–	–	○	○	○	○	○
Below 1200 Bytes	–	–	–	–	–	–	–	–	–	●	○	●	●	○
Forced Negotiation	–	–	–	–	–	–	–	–	–	●	●	●	●	●
Multi-Packet Initial	–	–	–	–	–	–	–	–	–	●	●	●	●	●
# vectors different to TSG	–	**3**	14	8	16	9	7	5	7	–	8	11	12	8

TLS-parsing fingerprints of censors potentially deploying TSG. While none of the evaluated censors deploy the latest TSG version as contained in the leak, some show similar parsing behavior. The second TLS censor of the GFW (GFW II) is particularly similar to the leaked TSG code. We discuss how these differences arise in Section 6.5.

●: Technique circumvents censor, same as TSG, ○: Technique does not circumvent censor, same as TSG,

●: Technique circumvents censor, different from TSG, ○: Technique does not circumvent censor, different from TSG, –: inapplicable, QUIC does not use TLS records

I–V: Git commits in TSG code that altered marked behavior: 513b9b52d9d1acd13d7ff5714c4fa68f38a6dd1c, c67f8195f55fc8872e4708963a94500d571b05d0, e436823d370054508915808d5f26f9665b58ccc0, 21950877e691e1b52038d6cffa3914b944c9dfe9, 93d17f6f5a116854fa05f5ef55b18baea9ca987b