Towards Illuminating a Censorship Monitor's Model to Facilitate Evasion

Sheharbano Khattak[*] *Mobin Javed*[†] *Philip D. Anderson*[*] *Vern Paxson*[†◇]
[*]*Independent researcher* [†]*U.C. Berkeley* [◇]*International Computer Science Institute*

Abstract

Censorship systems that make dynamic blocking decisions must inspect network activity on-the-fly to identify content to filter. By inferring the analysis models of such monitors we can identify their vulnerabilities to different forms of evasions that we can then exploit for circumvention. We leverage the observation that censorship monitors essentially work on the same principles as Network Intrusion Detection Systems (NIDS) and therefore inherit the same evasion vulnerabilities already discussed in the NIDS context for years. Using this past work as a guide, we illustrate the power of illuminating a monitor's analysis model by conducting extensive probing to test for vulnerabilities in the Great Firewall of China. We find exploitable flaws in its TCB creation and destruction, fragment and segment reassembly, packet validation, (in)completeness of HTTP analysis, and state management.

1. Introduction

Any censorship system that makes dynamic blocking decisions will employ a particular *analysis model* for dynamically inspecting activity. More powerful models enable more accurate decisions while also resisting attempts by users to evade detection; but such models present the censor with costly tradeoffs in terms of ease of implementation and scalability of performance versus the richness of analysis that a given model provides. In general, users can attempt to shape their network traffic in various ways so that the censor's system fails to accurately identify activity the censor wishes to suppress. Such anti-censorship *evasions* can then spur an "arms race" in which the censor modifies their system to plug holes in its coverage.

We premise our work on the observation that if we can illuminate the model used by a censorship system, we can identify *classes* of evasions that can elude its control: ways that users can alter network traffic that will both avoid detection and, crucially, require the censor to make *expensive changes to the system's basic model* to remedy.

We focus our effort on one widely used technology for censorship systems, an *on-path network monitor* that passively observes network traffic and effects censorship disruption by packet injection (e.g., forged RSTs or DNS replies). These systems, which we will refer to simply as *monitors*, afford more scalability than *in-path* elements that forward traffic (and can directly manipulate it), and this technology underlies much of the functionality used by the Great Firewall of China (GFW).

Particularly significant for our purposes, from a technical perspective such monitors function in the same manner as Network Intrusion Detection Systems (NIDS)—and for well over a decade researchers have studied the problem of the many ways that the analysis model of a NIDS renders it vulnerable to evasions, some very difficult to remedy [15] without requiring some form of in-line operation [11]. Potential monitor models span a wide range of basic architectures: simple stateless packet inspection; stateful but incomplete (lacking full protocol coverage); complete for a single set of implementation choices (such as how to resolve ambiguities); or assessing multiple possible implementation choices in parallel. In addition, stateful approaches must make decisions regarding *state management* that can render them vulnerable to either state exhaustion attacks or manipulation of state-holding to subvert the monitor's analysis capabilities. All of these present evasion opportunities.

To explore this space, we perform extensive probing of the GFW monitor. We treat it as a black-box that given HTTP/TCP/IP traffic provides indications of positive censorship decisions in terms of producing spoofed TCP RST packets[1] (we leave analysis of its other components, such as poisoning of DNS/UDP/IP traffic, for future work). In particular, we employ HTTP requests

[1] As noted in prior work [20], we confirmed that in particular it sends 3 spoofed RSTs with varying gaps between their sequence numbers. After this disruption, a $\approx$ 90 sec blocking period bars any communication from the client to server on the given server port [2].

that selectively contain banned keywords or hostnames along with varying forms of traffic permutation in order to map out a model of how GFW's analysis works, including its state management. As we make observations of its structural vulnerabilities, we confirm the viability of the corresponding evasions by employing them to retrieve banned content.

We find that GFW performs stateful analysis of the client side of HTTP streams, but not the server side. It lacks certain low-level protocol coverage such as validating checksums that renders it vulnerable to misdirection, but which the implementors can easily fix. It also suffers from deeper problems, including vulnerabilities to ambiguities due to overlapping IP fragments and TCP segments; TCB manipulation via TTL-limited SYN, FIN or RST packets; limited analysis of pipelined HTTP requests and URL encodings; and state management that will retain state for extensive periods of time (hours) but will discard state in lieu of very modest TCP bytestream buffering (1KB). We confirmed numerous evasions due to monitor's limited semantic model, illustrating the power of conducting such an exploration. We aim in follow-on work to then develop anti-censorship tools based on these findings.

2. Related Work

Numerous efforts have examined the operation of GFW [6, 8, 9, 13], finding it uses NIDS-like functionality to match sensitive keywords and terminates TCP connections by injecting RST packets. The authors of [13] find that GFW rarely filters HTTP responses, and apparently ceased so entirely between Aug. 2008 and Jan. 2009.

Evasion attacks on NIDS [11, 15, 17] have been studied for more than a decade, and a number of tools (`fragroute` [18], `sniffjoke` [1] and `idsprobe` [12]) implement these techniques. However, evasion attacks have only recently received attention by anti-censorship efforts. Clayton et al.'s study found that discarding the RST packets a censor injects protects effectively against keyword-based censorship [8]. This circumvention technique, however, does not provide full "evasion" because the censor has already noticed the activity and may employ other steps (e.g., IP blocking) to suppress further activity. It also requires both client-side and server-side changes, and impairs the intended use of RSTs for terminating connections for benign reasons.

WestChamber is a community research effort by a group of Chinese netizens [3–5]. It uses two approaches to evade GFW injection: (a) each end host sends a RST packet to fool GFW into viewing the connection as terminated, and (b) the client drops DNS replies injected by GFW based on fingerprint signatures. The authors caution that their techniques can fail in the future due to GFW updates. `brdgrd` [21] leverages GFW's lack of TCP segment reassembly for TLS negotiations to evade the detection of Tor Bridges via active scanning.

Effort (a) of *WestChamber* has the most relevance for our work, which concerns crafting traffic in such a fashion that GFW will not be triggered at all. In general, our efforts aim to foster the development of similar evasion tools in the specific context of on-path monitoring.

3. Methodology

We deduce GFW's analysis model by treating it as a black box and using `scapy` [7] to actively probe it with specially crafted packets, some of which include a triggering keyword. Observing GFW responding to such packets indicates its processing enabled it to observe the keywords in context; a lack of response indicates a failure of its analysis model to make such an observation.

Accurate design of our probe tests involves determining whether GFW is stateful (confirmed in § 5) and at which stage of packet/stream processing it triggers. For the latter, we probed it placing a censored keyword in an IP fragment, full IP packet, TCP segment, incomplete HTTP request, and full HTTP request (with and without accompanying HTTP response). We find that GFW only censors HTTP requests containing a sensitive keyword. This result confirms that filtering on HTTP response has likely discontinued [13]. Consequently, we always send probes with at least one complete HTTP request.

Experimental Setup. We use two machines for our evaluation, a client in China and an Apache webserver in USA. In reality a nation-wide censorship system like GFW surely comprises multiple heterogeneous monitors, but our use of a single network path means we should generally encounter the same particular components. Using the traceroute-based technique described by Xu et al. [23] we confirmed that we always encountered the same GFW IP address, though our activity may have triggered additional filtering devices along the path.

Soundness. To ascertain the integrity of our tests and the insights consequently obtained, we must ensure that (i) our tests work as intended, and (ii) we accurately identify RST packets injected by GFW. For the former, we first carry out all the tests locally without GFW in the picture, giving us a baseline against which to compare and contrast GFW's inferred analysis model. For the latter, we capture traces at both client and server, which enables us to accurately detect GFW-injected RST packets by observing RSTs at the receiver not sent by the sender.

4. Model Elements to Probe

In this section we frame design considerations for stateful monitors and the corresponding probe tests we can use to infer their analysis models. We categorize the

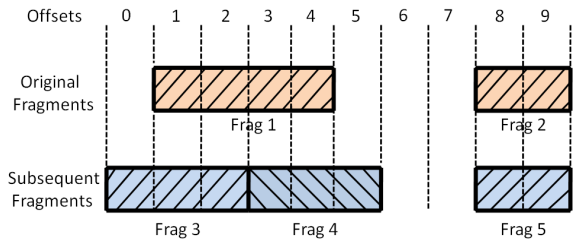

Figure 1: Terminology for overlapping fragments: (a) Frag 1 and Frag 2 are *original* fragments, while the others are *subsequent* fragments w.r.t. arrival time. (b) Frag 3 is *left-long* to Frag 1, Frag 4 is *left-short*, while Frag 2 and 5 are *left-equal*. (c) Frag 3 is *right-short* to Frag 1, Frag 4 is *right-long*, and Frag 2 and Frag 5 are *right-equal*. (d) Frag 3 and Frag 4 *partially overlap* Frag 1, while Frag 5 *fully overlaps* Frag 2. (e) Offsets 6 and 7 represent a *hole*.

design decisions into TCB creation, reassembly (packets/bytestream), TCB teardown, state management, and protocol message interpretation. We base our probe tests largely upon prior techniques [11, 15].

TCB Creation. Stateful analysis of TCP requires that a monitor uses some criteria to determine when to instantiate state, i.e., a Transmission Control Block (TCB) for TCP connections. The monitor can do so upon observation of a proper three-way handshake, but can also do so based on more partial observation.

To infer a stateful monitor's TCB creation policy, we can test (i) an initial SYN but no responding SYN-ACK, (ii) a SYN-ACK but no initial SYN, or (iii) both SYNs. After each we send a packet with banned content (its ACK completes the handshake) and see whether we observe disruption, indicating that the monitor has tracked the connection and thus presumably instantiated a TCB.

Reassembly. To fully recover application data, a monitor must reassemble fragmented packets as well as recover the TCP bytestream. In addition, overlapping fragments or TCP segments that differ in their contents render the correct reassembly ambiguous. If the monitor uses a different interpretation than the receiver, opportunities for evasion arise.

Figure 1 shows our terminology for discussing reassembly overlap. We test each of the 18 possible overlap cases that can arise to see how the monitor acts to resolve the ambiguous overlap, i.e., whether it reacts to a censored keyword that appears only in one version of the overlap. The same approach applies for reassembly of both IP fragments and overlapping TCP segments.

State Management. Due to resource constraints, any stateful monitor must define a policy for ultimately discarding state (for long-running connections), typically framed in terms of how long and/or how much state to keep. Inducing the monitor to discard state can then open up an evasion opportunity. We can infer a monitor's state-holding capabilities and policies using persistent HTTP connections over which we introduce increasing amounts of time and volume of non-sensitive data prior to sending a sensitive request. In addition, a monitor may time out connections with holes (pending undelivered data) using a different policy, which we can probe in an analogous manner.

TCB Teardown. Similarly, a monitor must decide when to stop maintaining *any* information about a connection. Normally it makes sense to do so upon observing one side transmit a RST, or both engage in a FIN handshake. However, connections can also cease due to other reasons, such as a loss of network connectivity, for which the monitor must ultimately recover using a timeout, the choice of which will necessarily trade off in-depth analysis versus available resources.

We can infer a monitor's teardown policy by establishing a connection, transmitting a termination message crafted so that the intended receiver will ignore them, and then subsequently attempting to access censored content. Success for the latter indicates the monitor tore down the TCB in response to the termination message.

Protocol Message Interpretation. In addition to connection management and recovery of data, a basic monitor analysis issue concerns correct and full interpretation of the meaning of various messages. Here, implementation shortcomings can result in failures to validate values for different header fields (particularly true of edge cases), but more fundamentally the monitor's job becomes difficult in the face of messages that diverge from the protocol specification. For lower-layer protocols exploiting such deficiencies can prove difficult (e.g., a packet with bad IP header checksum will be dropped before even reaching the monitor), but more opportunities arise at higher layers—particularly, for our purposes, when considering HTTP messages.

Even given full, correct implementation, monitors face difficult analysis decisions due to incomplete information. For example, by itself a monitor will have significant problems determining whether a given packet's IP TTL suffices for delivery of the packet to the receiver. Prior work has shown that a monitor can resolve many protocol-based ambiguities by observing an end machine's response to special probes that comprehensively explore a protocol in all perceivable contexts [17]. Given limited space we do not explore this approach comprehensively, but in the next section present a subset that reflects the most significant aspects.

5. Illuminating GFW's Analysis Model

We now turn to using the probe tests identified in the last section to deduce GFW's analysis model. While we conducted 40+ such tests, due to limited space here we only report tests leading to interesting observations.

Notation. For unambiguous presentation of our results, we introduce notation (inspired by [20]) as follows. We denote the temporal sequence of packets sent by the client and the web sever as (p_i, p_{i+1}, …, p_m) and ($\overline{p_j}$, $\overline{p_{j+1}}$, …, $\overline{p_n}$), respectively. We represent packet(s) injected by GFW to the client and server by g and $\overline{g}$ (we omit the number of such packets when repeated), and the shorthand *reset* refers to (g,$\overline{g}$). We capture a packet's TCP flags using superscripts—for example, p^{FA} means with FIN and ACK set—and optional flags with parentheses, such as $p^{R(F)}$ for a RST with or without FIN. We prefix a flag with negation $\neg$ to indicate that it is not set.

$p(D)$ refers to a data packet, for $D \in \{Bad, Good\}$, meaning carrying a censored keyword or not. We represent back-to-back data packets collectively carrying data D as $p(D)^+$ and D's total size by $\delta(D)$. $seq(p)$ and $ack(p)$ represent p's TCP sequence and ack. numbers. $\tau(t)$ refers to the passage of t units of time, and $tuple(p)$ represents the connection 5-tuple of a packet p. $Hole$ refers to one or more missing data packets for which packets later in the sequence space have arrived. The predicate *abovehole(p)* expresses a packet p for which $seq(p)$ falls within the advertised window but above a gap in the received bytestream. Finally, we include any additional information using a leading subscript. For example, $_{IP(TTL=<low>)}p$ indicates a packet with an IP TTL low enough to not make it to the receiver (but high enough to be seen by the monitor).

Given this notation, Table 1 summarizes our findings. In the table, *insertion* refers to inducing the monitor to process data that the receiver will not, while *evasion* refers to undermining the monitor's processing so that it fails to recognize data that the receiver will process. In some cases, insertion of data might lead to evasion of subsequent data, which we represent by *insertion-evasion*. All the evasion opportunities listed in the table involve client-side traffic manipulation, however some cases presume specific processing style on the server end. We indicate this under "**Receiver Dependent?**".

TCB Creation. We find that GFW creates a TCB directly upon observing p^S, enabling an evasion attack (TCP1): transmitting $_{IP(TTL=<low>)}p^S$ past GFW and then establishing a connection with the same $tuple$ but different initial sequence numbers enables successful retrieval of a censored web page. We note that this TCB creation behavior renders GFW vulnerable to SYN-flooding-style attacks that exhaust its available memory.

Reassembly. Different from [22] and [13], we find GFW can assemble both IP fragments and TCP segments for HTTP connections, suggesting that GFW employs different policies for different protocol (or has subsequently changed its operation). However, it prefers the original IP fragment for all cases except where F_{sub} is left-long and right-long to F_{orig}, per the terminology in Figure 1 (IP2). For reassembly of overlapping TCP segments, we find that if a subsequent left-equal segment arrives, GFW prefers it (TCP5). For all other cases, GFW does not react to any keyword placement, enabling us to successfully retrieve the censored content.

State management. We find that GFW maintains state for a connection for as long as 10 hours (TCP7), and for as much as 1 GB data transferred from client to server (TCP8). (Note, these values correspond to the maxima we tested, rather than precise thresholds. We obtained consistent results over repeated measurements for the values listed, but the boundaries used at a given time could depend on the monitor's operational load.)

We repeated these tests but with a preceding hole, finding that GFW abandons state more readily, doing so after having to buffer 1KB of data above the hole (TCP9). Likewise, GFW purges state after 60–90 minutes elapses for a connection that includes a hole (TCP10).

TCB Teardown. To identify GFW's TCB teardown policy (TCP6^a), we send $p^{R(A)}$ and (separately) $p^{F(A)}$ from client to server. GFW's acceptance of these allows us to force TCB teardown (for example, using insufficient TTL for the termination packet to make it to the receiver), after which we verified we can successfully access banned content. Moreover, simply sending p^F suffices to induce GFW to tear down connection state (TCP6^b), even though the receiver will drop the packet due to the lack of an ACK.

Protocol Message Interpretation. We test GFW for resilience against a number of issues regarding protocol validation, completeness, and ambiguous messages. One telling observation is that GFW accepts a packet with wrong ack. number (TCP2). RFC [14] states that hosts should drop incoming segments that acknowledge data the host has not sent (and send an ACK in response). GFW also processes data packets that lack ACK (TCP3) or have incorrect checksums (TCP4). All three test cases allow us to insert packets into the monitor's processing.

Regarding HTTP processing, we find that GFW inspects both GET and POST requests, but not others such as HEAD or DELETE. In addition, GFW only analyzes the URI and headers of POST requests, but not the following message body. These tests establish that GFW does not perform blind regular-expression matching on complete HTTP packets, but instead selectively inspects only the header information.

We performed two tests to inspect GFW's behavior when receiving requests that deviate from the specification [10], due to either an extra space between the Method and the Request-URI, or between the `Host:` header name and its value. The first successfully evades GFW, but the monitor does process the second. We also find that the monitor does not look beyond 2,048 bytes into the Request-URI (HTTP2).

Test	Evasion Class	Description	Circumvention Opportunities	Fixing Cost	Receiver Dependent?
IP1	Ambiguity	$_{IP(TTL=<low>)}p(Bad) \implies reset$	Insertion	High	
IP2	Reassembly	Overlapping fragment processing	Insertion	High	✓
TCP1	TCB creation	$_{IP(TTL=<low>)}p_i^S,\ p_{i+1}^S,\ p_{i+2}(Bad) \wedge (tuple(p_i) = tuple(p_{i+1})) \wedge (seq(p_i) \neq seq(p_{i+1})) \implies \neg\, reset$	Insertion-Evasion	Low	
TCP2	Incompleteness	$_{IP(ack=<bad>)}p(Bad) \implies reset$	Insertion	Low	
TCP3	Incompleteness	$_{IP(chksum=<bad>)}p(Bad) \implies reset$	Insertion	Low	
TCP4	Incompleteness	$p^{\neg A}(Bad) \implies reset$	Insertion	Low	
TCP5	Reassembly	Overlapping segment processing	Insertion	High	✓
TCP6[a]	TCB Teardown	$_{IP(TTL=<low>)}p_i^{R(A)},\ p_{i+1}(Bad) \implies \neg\, reset$	Insertion-Evasion	High	
TCP6[b]	TCB Teardown	$_{IP(TTL=<low>)}\ p_i^F,\ p_{i+1}(Bad) \implies \neg\, reset$	Insertion-Evasion	Low	
TCP7	State Management	$\tau(\leq\ \approx 10$ hr), $p_i(Bad) \implies reset$	State exhaust.	High	
TCP8	State Management	$(p_i(Good)^+ \wedge \delta(Good) \leq\ \approx 1$ GB), $p_{i+1}(Bad) \implies reset$	State exhaust.	High	
TCP9	State Management	$hole,\ (p_i(Good)^+ \wedge \delta(Good) \geq 1$ KB $\wedge\ abovehole(p_i))$, $p_{i+1}(Bad) \implies \neg reset$	State exhaust.	High	✓
TCP10	State Management	$hole, \tau(y) \geq 60$ min, $(p_i(Bad) \wedge abovehole(p_i)) \implies \neg reset$	State exhaust.	High	✓
HTTP1	Ambiguity	GET with >1 space between method and URI $\implies \neg\, reset$	Evasion	Low	
HTTP2	Incompleteness	GET with keyword at location $>2048 \implies \neg\, reset$	Evasion	Low	
HTTP3	Incompleteness	GET with keyword in $\geq$ 2nd of multiple requests in single segment $\implies \neg\, reset$	Evasion	Low	
HTTP4	Incompleteness	GET with URL encoded (except %-encoding) $\implies \neg\, reset$	Evasion	Low	✓

Table 1: Evasion opportunities in GFW's analysis of network traffic.

Next, we aim to establish how comprehensively GFW will look into an HTTP packet to extract these fields, which has relevance because HTTP allows multiple outstanding requests ("pipelining"), and TCP segmentation can cause these to appear in a single packet. Thus, correct monitor processing requires extracting all requests present in a packet, not just the first. However, we find that GFW inspects only the first Request-URI and `Host:` value (HTTP3). But for *some* `Host:` values, GFW detects them even if placed in subsequent pipelined requests, indicating it must contain two separate blacklists evaluated by separate code paths (or two separate devices operate along the same path). One of these appears to have the form "Host: *.baddomain.com", for which it scans the entire bytestream, while the other contains keywords for which it only scans the first instance of a URI and a `Host:` header in a packet.

Finally, we test GFW's decoding capabilities by encoding sensitive URIs using different encoding techniques. The tests reveal that GFW only processes standard URI percent-encoding, but not numerous others [16]. Many of the latter have only partial support amongst web servers, but for those that do support them, they provide quite convenient evasion.

6. The Cost of Fixing Evasion Bugs

In the last section we identified a number of evasion opportunities to counter GFW censorship. Here we discuss the potential level of difficulty that GFW implementors likely face to address issues such as these, basing our rationale on our own experiences with implementing network monitors.

High Difficulty. We can readily cause GFW to process packets that receivers will not by using a low IP TTL value (IP1). This vulnerability appears hard to fix, as monitor has little ability to validate the IP TTL through passive monitoring, given a discrepancy of just one hop suffices for the evasion. Hop counts can vary organically due to routing changes, making an accompanying decision process difficult to determine.

Removing ambiguities arising from reassembly of partially overlapping fragments (IP2) and TCP streams (TCP5) appears expensive for monitor. It requires either in-line normalization [11,19], imposing performance and robustness concerns, or prior knowledge of end host implementation behavior [17], unscalable for a nation-wide censorship system.

TCB teardown (TCP6[a]) marks the point at which a monitor decides to stop maintaining information about

a connection.[2] This is a critical decision because if a monitor deletes the TCB of a connection that has not actually terminated, it loses any further monitoring opportunities. In principle, the monitor could kill connections for which it discards state, but doing so would incur large collateral damage if either the monitor has state-holding problems, or if routing changes bring new active flows by one of its monitors. A similar problem arises for the state management issues sketched in TCP7–TCP10, i.e., distinguishing timed-out connections from those that simply lie dormant. GFW's current design choice of keeping state for lengthy periods of time (except for connections with holes) renders it vulnerable to resource-exhaustion attacks. However, the alternate option of timing out inactive connections risks evasion possibilities.

One other approach GFW could implement would be to use "cold start" to re-instantiate state for connections that continue to be active after the monitor has either discarded state for them or observed them to apparently terminate [11]. This would enable it to recover memory for connections that have genuinely terminated, but still monitor those that somehow continue. However, this at best provides a partial solution. The cold-start approach of [11] relied on the fact that the local side of each connection could be deemed trustworthy. In our context, this is not the case, and can be leveraged for evasion by, for example, splitting a banned keyword across the bytestream as seen prior to a connection's state apparent termination versus after its reappearance.

Low Difficulty. A number of the evasions we list (TCP1–TCP4, HTTP1–HTTP4) are simple to fix, likely requiring only modest modifications to the existing code base. For example, the censor can address TCB creation issues (TCP1) by creating a TCB only upon seeing both a SYN and the corresponding SYN-ACK, since the latter is difficult to spoof.

However, some of these will reflect design decisions to trade off performance for completeness. Thus, even fixing these simple problems could represent significant burdens for the implementors.

7. Conclusion and Future Directions

Our work focuses on the observation that monitors that dynamically inspect and censor network traffic fundamentally must employ some sort of *analysis model* to determine whether the traffic includes prohibited content. Inference of this model can then enable systematic identification of evasion opportunities that users can exploit to access restricted content by permuting their network traffic accordingly.

[2] TCP6^b reflects a similar state-holding issue as TCP6^a, but one easy to fix as it is based on an incorrect implementation of the TCP specification.

We explore this problem space inspired by prior work analyzing the vulnerabilities of NIDS to forms of evasion. We consider general classes of evasions to which monitors can be vulnerable and develop an extensive set of probe tests for each category, using the tests to illuminate the analysis model of China's GFW. Based on GFW's responses (or lack thereof) to our probes, we identified a number of facets of how it performs its monitoring:

- GFW operates in a stateful manner, but limits its analysis to one direction only (client-to-server) .
- We can manipulate GFW's connection management state (TCB) both in terms of inducing state for connections that do not actually exist (creating SYN-flooding and unsynchronized-monitoring vulnerabilities) and triggering tear-down of state for connections that still exist (spoofed termination packets, timeouts, accelerated timeouts for connections with sequence holes).
- GFW is prone to state exhaustion attacks, as it maintains state for ≥ 10 hours of inactivity.
- GFW employs specific fragment and TCP segment reassembly policies, enabling evasions by overlapping fragments or segments for receivers whose policies differ.
- GFW's HTTP request processing has numerous omissions and limitations (syntax tweaks can undermine blacklist matching; elements of pipelined requests go unanalyzed).

Our preliminary results suggest numerous directions for further exploration: (1) *automated model extraction* (turn-key generation of full analysis profiles of monitor); (2) *evasion tools* (using the output of such extraction to automatically configure tools for permuting a system's traffic to facilitate circumvention); (3) *assessment of analysis incongruity* (e.g., tracking how GFW's analysis components both differ across various points in the network, and evolve over time); (4) *policy discrepancies* (e.g., whether a censor employs different policies for different protocols or web sites).

Acknowledgments

This work was supported by the National Science Foundation under grants 1223717 and 1237265, Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

8. References

[1] sniffjoke. http://www.delirandom.net/sniffjoke/.

[2] HTTP URL/keyword detection in depth. http://gfwrev.blogspot.jp/2010/03/http-url.html, 2010.

[3] Scholar Zhang: Intrusion detection evasion and black box mechanism research of the Great Firewall of China. https://code.google.com/p/scholarzhang/, 2010.

[4] west-chamber-season-2. https://code.google.com/p/west-chamber-season-2/, 2010.

[5] west-chamber-season-3. https://github.com/liruqi/west-chamber-season-3/, 2011.

[6] D. Anderson. Splinternet Behind the Great Firewall of China. *Queue*, 10(11):40:40–40:49, Nov. 2012.

[7] P. Biondi. Scapy. http://www.secdev.org/projects/scapy/.

[8] R. Clayton, S. J. Murdoch, and R. N. M. Watson. Ignoring the Great Firewall of China. In G. Danezis and P. Golle, editors, *Proceedings of the Sixth Workshop on Privacy Enhancing Technologies (PET 2006)*, pages 20–35, Cambridge, UK, June 2006. Springer.

[9] J. R. Crandall, D. Zinn, M. Byrd, E. Barr, and R. East. Conceptdoppler: a weather tracker for Internet censorship. In *Proceedings of the 14th ACM conference on Computer and communications security*, CCS '07, pages 352–365, New York, NY, USA, 2007. ACM.

[10] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, 1999.

[11] M. Handley, V. Paxson, and C. Kreibich. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-end Protocol Semantics. In *Proceedings of the 10th conference on USENIX Security Symposium - Volume 10*, SSYM'01, pages 9–9, Berkeley, CA, USA, 2001. USENIX Association.

[12] L. Juan, C. Kreibich, C.-H. Lin, and V. Paxson. A tool for offline and live testing of evasion resilience in network intrusion detection systems. In *Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, DIMVA '08, pages 267–278, Berlin, Heidelberg, 2008. Springer-Verlag.

[13] J. C. Park and J. R. Crandall. Empirical Study of a National-Scale Distributed Intrusion Detection System: Backbone-Level Filtering of HTML Responses in China. In *Proceedings of the 2010 IEEE 30th International Conference on Distributed Computing Systems*, ICDCS '10, pages 315–326, Washington, DC, USA, 2010. IEEE Computer Society.

[14] J. Postel. RFC 793: Transmission Control Protocol, Sept. 1981.

[15] T. H. Ptacek and T. N. Newsham. Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection. Technical report, Secure Networks, Inc., Suite 330, 1201 5th Street S.W, Calgary, Alberta, Canada, T2R-0Y6, 1998.

[16] D. Roelker. HTTP IDS Evasion Revisited. Technical report, Aug. 2003.

[17] U. Shankar and V. Paxson. Active Mapping: Resisting NIDS Evasion without Altering Traffic. In *Proceedings of the 2003 IEEE Symposium on Security and Privacy*, SP '03, pages 44–, Washington, DC, USA, 2003. IEEE Computer Society.

[18] D. Song. fragroute. http://www.monkey.org/~dugsong/fragroute/.

[19] M. Vutukuru, H. Balakrishnan, and V. Paxson. Efficient and Robust TCP Stream Normalization. In *Proceedings of the 2008 IEEE Symposium on Security and Privacy*, SP '08, pages 96–110, Washington, DC, USA, 2008. IEEE Computer Society.

[20] N. Weaver, R. Sommer, and V. Paxson. Detecting Forged TCP Reset Packets. In *NDSS*, 2009.

[21] P. Winter. brdgrd. https://git.torproject.org/brdgrd.git.

[22] P. Winter and S. Lindskog. How the Great Firewall of China is blocking Tor. In *Proceedings of the USENIX Workshop on Free and Open Communications on the Internet (FOCI 2012)*, August 2012.

[23] X. Xu, Z. M. Mao, and J. A. Halderman. Internet censorship in China: where does the filtering occur? In *Proceedings of the 12th international conference on Passive and active measurement*, PAM'11, pages 133–142, Berlin, Heidelberg, 2011. Springer-Verlag.