Your Censor is My Censor: Weaponizing Censorship Infrastructure for Availability Attacks

Kevin Bock Pranav Bharadwaj Jasraj Singh Dave Levin

University of Maryland

***Abstract*—Nationwide Internet censorship threatens free and open access to communication and information for millions of users living inside of censoring regimes. In this paper, we show that this poses an even greater threat to the Internet than previously understood. We demonstrate an off-path attack that exploits a little-studied but widespread feature of many censoring infrastructures: *residual censorship*, in which a censor continues blocking traffic between two end-hosts for some time after a censorship event. Our attack sends spoofed packets with censored content, keeping two victim end-hosts separated by a censor from being able to communicate with one another. Although conceptually simple, this attack has several challenges, which we address. We demonstrate the feasibility of the attack through two studies: one to capture the current state of residual censorship, and another to actually launch the attack (against machines we control). We show that the attack can be launched despite stateful TCP tracking used by many censors, and that it also works against those who censor by null-routing. We will be making our code publicly available.**

I. Introduction

Many nation-states around the world perform automated, in-network censorship of the traffic traversing their borders. Although the precise mechanisms vary from one censoring regime to another, there are some similarities in how they operate. Most of them look for specific keywords, domain names, or protocols to censor, and they halt communication by injecting connection tear-down packets (TCP `RST`s), sending block-pages, or simply dropping traffic. Collectively, Internet censorship affects billions of people living within these countries' borders, and all users seeking to communicate with them—ultimately threatening free and open communication on the Internet.

In this paper, we show that censoring regimes pose an even greater threat to the Internet than previously understood. In particular, we show that attackers can *weaponize* censoring infrastructure to keep two end-hosts separated by that country's borders from being able to communicate with one another. The attacker need not be within the censoring regime; it merely needs the ability to source-spoof packets.

The attack makes use of a little-studied but widespread feature of many censoring infrastructures: *residual censorship*. After a given TCP connection triggers a censor (e.g., by including a forbidden keyword in a plaintext HTTP GET request), some censors not only tear down the connection, but "residually censor" all *future* communication between the two endhosts (on particular ports) for some period of time—even if the subsequent traffic is completely innocuous.

Armed with this insight, our attack is relatively straightforward: the adversary spoofs the victim endhosts, sending packets with censored content across the censor's border, thereby triggering censorship and blocking the victims from communicating for some time.

Although conceptually simple, there are several challenging aspects of this attack in practice. In particular, most censoring middleboxes are stateful (they track connections across packets), and so it would seem that the attacker would have to fake a TCP three-way handshake in order to be able to send a valid censored packet in the first place. We show that, surprisingly, the attack is indeed possible, even with a completely off-path attacker.

Our central contributions of the paper are not just in demonstrating the possibility of weaponizing residual censorship, but also in performing two comprehensive feasibility studies for the attack:

First, we perform active measurements to analyze the current state of residual censorship around the world today: what countries employ it, how it operates, how long it lasts, and so on. Our results demonstrate a wide variety in the implementation of residual censorship systems—even within a given country, residual censorship can operate very differently from one protocol to another.

Second, we analyze our attack's success and feasibility by launching it using (and targeting) hosts we control in three censoring nation-states—China, Iran, and Kazakhstan—across four protocols (HTTP, HTTPS+SNI, HTTPS+ESNI, and Iran's protocol filter [1]). This study sheds light on the limitations of the attack—for instance, we find that the attacker generally needs to be on the same side of the censor as the victim client. It also shows several surprising strengths of the attack. For example, Iran and Kazakhstan extend the duration of residual censorship whenever the censor sees a matching packet—as a result, once the attack is started, the victim's own packets help sustain the attack on themselves.

Our results show that even a low-resource attacker can weaponize censoring nation-states to launch an effective availability attack. In China, a source-spoofing attacker needs to send only four packets every three *minutes* to indefinitely sustain blocking between a given pair of end-hosts on a given destination port. An attacker that can sustain 1,093 packets per second (about 600 kbps) can weaponize Kazakhstan's censor, or 728 packets per second (422 kbps) to weaponize Iran's. Collectively, our results show that censorship infrastructures

as they are deployed today have the potential to cause even more harm to the Internet at large than previously understood.

The rest of this paper is organized as follows. In Section II, we review related work and provide a background on nation-state censorship, residual censorship, and availability attacks. We describe our experiment methodology in Section III. Section IV presents our study of the current state of residual censorship, and Section V presents our feasibility study from launching the attack against hosts under our control. We speculate about the breadth of the attack and discuss limitations in Section VI, explore potential mitigations in VII, and present ethical considerations in Section VIII. Finally, we conclude in Section IX.

II. Background & Related Work

How censors operate There have been many measurement studies to understand how various censoring infrastructures work—far too many and varied to do full justice here. Instead, we highlight several key properties that are critical to understanding our results.

In-network censors generally have two broad components: a mechanism for determining whether to censor, and a set of mechanisms for actually tearing down the offensive connection. Determining whether to censor a connection has been shown to depend on keywords (e.g., in HTTP GET requests [2], [3]), domain names (e.g., in the Server Name Indication (SNI) field during an HTTPS connection [4], [5], [6]), or the very protocol being used [7], [1]. Our evaluation spans different types of these.

To actually tear down a connection, censors often employ one of two tactics: Some simply drop the offending user's (or connection's) traffic. This is referred to as *null routing*, and is obviously a very effective way of terminating a connection. However, it is also costly for the censor, as it requires them to have a box on the path between source and destination at which they can drop the traffic. More commonly, censors are deployed not as man-in-the-middle adversaries, but as man-on-the-side: they sit just off of the path, and the ISPs send copies of packets (in both directions) to the censor for processing. For such deployments, the censor tears down the connection not by dropping the offending traffic, but by injecting spoofed TCP `RST`s (or lemon DNS responses [8]) to both client and server, causing them both to believe the other had terminated the connection. In our experiments, we study both null-routing and tear-down censors.

Residual censorship Residual censorship is a feature observed in some censorship systems in which the censor continues to block innocuous requests for a short period of time *after* censoring a forbidden request. We are not the first to observe this behavior; the Censored Planet datasets [9] report on instances where innocuous queries are blocked shortly after sending a censored query. It has also been noted in the context of studying censorship in China [4], Iran [1], and others [2] that, for some countries and some protocols, once a connection triggers censorship, subsequent connections can also be censored. However, to the best of our knowledge, we are the first to systematically study residual censorship—what precise protocols and ports it targets, for how long, and whether innocuous traffic can keep residual censorship in place—and how attackers can weaponize it.

An important facet of residual censorship is precisely what the censor blocks after censorship is initially triggered. There are three basic options available to an adversary: *2-tuple* (client IP, server IP), *3-tuple* (client IP, server IP+port), or *4-tuple* (client IP+port, server IP+port)[1]. We are not aware of any censors who use 2-tuple residual censorship. All prior work of which we are aware that had identified some form of residual censorship focused only on 3-tuple. To our knowledge, we are the first to identify 4-tuple censorship, and yet, as we will show, it is one of the most widespread forms of residual censorship.

Weaponizing censors We are aware of only one instance of coercing a censor into blocking someone else. In 2014, the developers of VPN Gate realized that the Great Firewall of China (GFW) had developed an active system for scraping the IP addresses of their VPNs and automatically blocking them without validating that these IP addresses were actually VPNs. The researchers began to mix innocent IP addresses into their published list of VPN servers and were able to control which IP addresses were globally blocked by the GFW for two days until the GFW added verification checks [10]. Our approach differs considerably; in our setting, an attacker can trigger the censorship, without needing the GFW to actively scan them. Moreover, our attack appears to be more difficult for the GFW to mitigate.

Recently, Bock et al. [11] demonstrated another way in which censors can be weaponized. They showed that censoring middleboxes can be exploited for TCP-based reflected amplification attacks, with surprisingly high amplification factors—a potentially powerful tool for volume-based denial of service attacks. In this paper, we show that censors can also be weaponized to launch availability attacks.

Off-path attacks Our work fits into a much broader space of off-path attacks. Prior work has explored how to adversely affect TCP connections between two end-hosts in myriad ways, including TCP side channels [12] and data injection [13]. Other work has shown that an off-path attacker can weaponize network infrastructure to launch amplification attacks [14], [15], [16]. Each of these prior attacks manipulate the state at the end-hosts it targets. Our work broadens this space by showing that attackers can manipulate the state of *middleboxes in the network itself* to adversely affect end-hosts' ability to communicate.

III. Measurement Methodology

As with all censorship measurement research, we are limited by the vantage points we can access and the censorship we can experience. For our experiments, we obtained four vantage

[1]It is also conceivable that a censor could block *multiple* IP addresses at a time, such as a /24, but we did not study this.

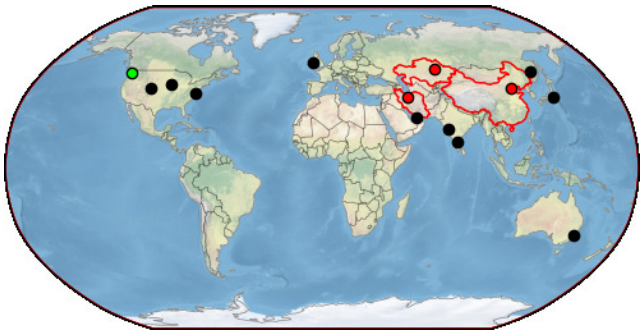

Fig. 1: Vantage points in our experiments. The green dot is our attacker running SP[3] [17]; black dots represent victim vantage points; and the red dots denote the location of the servers inside the censoring regimes we studied: China, Iran, and Kazakhstan (outlined in red). Note that some dots overlap.

points within censoring countries: two in China (Beijing), one in Iran (Tehran), and one in Kazakhstan (Qaraghandy). We also performed experiments from two vantage points we obtained in India (Bangalore) and one vantage point we obtained in Russia (Khabarovsk), but as we will see in the next section, we were unable to identify residual censorship in either location. We also obtained vantage points located in geographically disparate locations around the world that do not experience censorship: Australia (Sydney), India (Mumbai), Ireland (Dublin), Japan (Tokyo), United Arab Emirates (Dubai), and the United States (Iowa, Colorado, and Virginia). Figure 1 shows the locations of each of these vantage points, along with the censoring regimes in which we validated our attack.

To test for residual censorship, we issued queries that trigger censorship followed by queries that do not trigger censorship on their own and observed if the censor interferes. The specific queries we issued for each protocol are as follows (for ease of exposition, we will refer to HTTPS with SNI as simply "SNI", and HTTPS with ESNI as simply "ESNI"):

- **SMTP**: Sent an SMTP request with a forbidden email address (such as "xiazai@upup.info" in China [6]) in the `MAIL FROM:` field.
- **DNS**: Issued a DNS query (over both UDP and TCP) with a forbidden question record (such as "facebook.com" in China) both to real DNS resolvers and to resolvers we controlled.
- **HTTP**: Issued a HTTP GET request with a forbidden URL in the host header (such as `Host: youporn.com`), or with a forbidden keyword as an HTTP parameter (such as `?q=ultrasurf`).
- **HTTPS (SNI)**: Initiated a TLS handshake with a forbidden domain in the SNI field to servers we controlled.
- **HTTPS (ESNI)**: Initiated a TLS handshake configured with ESNI to servers we controlled.
- **Protocol Filter (Iran)**[2]: Sent two messages back to back containing the message "test". As this trivially does not match any approved protocol, it triggers censorship [1].

We also tested different patterns of follow-up requests and packets. To identify 3-tuple residual censorship, we issued follow-up queries with the same protocol to the same destination, containing an innocuous payload (such as "example.com"). We also tested making innocuous queries of different protocols and malformed payloads that do not resemble any protocol (such as just the string "test"). To identify 4-tuple residual censorship, we sent follow-up packets with the same source port to the same destination IP address and port (but with an out-of-window TCP sequence and acknowledgment number) and confirmed that our packets arrived at the destination correctly and without interference. We performed this check with `SYN` packets, `PSH` packets, `PSH+ACK` packets, and `RST` packets. We then repeated these experiments across many ports to identify which ports were affected.

IV. State of Residual Censorship

In this section, we present the results from our comprehensive study of the current state of residual censorship in China, Iran, and Kazakhstan. Table I provides a breakdown of all of our results in this section.

Which countries employ residual censorship? We found some form of residual censorship (3-tuple or 4-tuple) for multiple protocols in China (SNI, ESNI, and HTTP), Iran (HTTP, SNI, and its protocol filter), and Kazakhstan (HTTP and SNI).

China and Iran in particular employ residual censorship for only *some* of the protocols they censor. Neither have residual censorship for any of their DNS censorship (DNS-over-UDP or DNS-over-TCP)[3]. Further, China does not employ residual censorship for their SMTP censorship.

Some countries we tested do not employ residual censorship at all against our vantage points. Both of our vantage points within the Airtel ISP in India experienced HTTP and SNI censorship, but neither experienced residual censorship. We were also unable to trigger censorship from our vantage point in Russia to any of our destination vantage points, so we exclude both of these from our analysis.

What types of residual censorship do censors employ? We find that censors vary between 3-tuple and 4-tuple residual censorship, depending on the protocol being censored.

China uses 3-tuple residual censorship for HTTP traffic and censors by injecting TCP `RST` packets. This has been observed in the past [19], [2]. Prior work has reported residual censorship in China for SNI [4] by injecting `RST`s, but neither of our two vantage points experienced any SNI residual censorship to any of our vantage destinations.

[2]In addition to its standard content filter, Iran uses a protocol filter, which censors unrecognized protocols on monitored ports [1].

[3]In Iran, although some prior work has reported DNS-over-TCP censorship [18], we are unable to trigger any DNS-over-TCP censorship at this time (similar to what was reported in [6]).

Country	Protocol	Ports	Type	Duration	Bidirectional	Timer Reset	Mechanism
China	HTTP	Any	3-tuple	90s	✓	✘	Injected RST
	SNI	Any	3-tuple	60s	✓	Unknown	Injected RST
	ESNI	Any	3 and 4-tuple	120-180s	✓	✘	Null Routing
Kazakhstan	HTTP	Any	4-tuple	120s	✓	✓	Null Routing
	SNI	Any	4-tuple	120s	✓	✓	Null Routing
Iran	HTTP	53, 80, 443	4-tuple	180s	✓	✓	Null Routing
	SNI	53, 80, 443	4-tuple	180s*	✓	✓	Null Routing
	Protocol Filter	53, 80, 443	4-tuple	60s	✘	✓	Null Routing

TABLE I: The current state of residual censorship, among the countries and protocols we tested (those that we tested but are not in the table did not residually censor in our tests). We were unable to reproduce SNI censorship in China; in that row, we report prior results [4]. *: Iran's SNI residual censorship sometimes lasts longer than 180s; in a small number of our experiments, we found it to last upwards of 5 minutes.

ESNI censorship in China presents a more complicated picture. Less than 1 second *after* the GFW sees a TLS ClientHello containing the ESNI extension, it begins dropping all traffic that matches the connection's 4-tuple (note that the ESNI packet itself reaches the server unaffected). This is 4-tuple residual censorship. For approximately five seconds, the GFW also drops all traffic that matches the connection's 3-tuple: a short window of 3-tuple residual censorship. But if the client sends a second ESNI request with the same 3-tuple within the next three minutes, the GFW will begin dropping all traffic that matches the 3-tuple for three minutes: a long window of 3-tuple residual censorship. Unlike for HTTP and SNI, ESNI's residual censorship does not operate equally in both directions. Researchers have hypothesized in the past that China censors each protocol using a different set of middleboxes; the vast disparity between residual censorship implementation across our vantage points supports this hypothesis [6], [20].

In Iran and Kazakhstan, we find that the mechanism used for residual censorship (null-routing) and type of residual censorship (4-tuple) is consistent between protocols. As we will see later in this section, however, there are other inconsistencies in the implementations of the residual censorship for each censored protocol within Iran and Kazakhstan, such as the duration of censorship.

Does residual censorship use the same mechanisms as the initial censorship? We find that residual censorship is generally enforced using the same mechanism as the initial censorship. For example, China injects RST packets to censor HTTP normally, and injects RST packets for its residual censorship (the same is also reported for China's SNI censorship [4]). China's ESNI censorship operates with null-routing, as does its residual censorship. The censorship mechanisms are also consistent in Iran and Kazakhstan, with one exception.

We find that Iran censors HTTP using multiple methods simultaneously: injecting a block page with a packet that has the RST flag set while simultaneously null routing the connection. Despite using three censorship mechanisms for regular censorship, only 4-tuple null-routing continues for residual censorship.

What ports are affected by residual censorship? We tested this by issuing censored requests to vantage points we controlled destined to all 65,535 ports and confirmed that all were affected. We find that the ports affected by residual censorship match the ports affected by the regular censorship in each country we studied, but each country monitors a different set of ports. In China (with HTTP and ESNI) and Kazakhstan (with HTTP and SNI), we find that we can trigger residual censorship on any arbitrary port, including ephemeral ports. In Iran, however, both the protocol filter and the standard censorship system only monitor ports 53, 80, and 443, and therefore we can only trigger residual censorship to these ports. Note that in Iran, residual censorship can be triggered for any protocol on any of those three ports: for example, we can trigger HTTP residual censorship to port 53.

Is residual censorship applied bidirectionally? Even within the same country, residual censorship is not always applied equally to connections entering the country as to those exiting the country. Although we find that Iran's standard censorship system can be triggered bidirectionally, we confirm the findings of [1] that the protocol filter (and by extension, its residual censorship) only operates on flows leaving Iran. China's ESNI censorship operates bidirectionally, but it operates differently (and more aggressively) against traffic entering the country than exiting the country.

For every other censorship system we tested, we were able to trigger censorship (and residual censorship) equally from outside the country. Like all censorship research, our study is limited by the vantage points we can access; it is possible that there are other censorship systems that only employ residual censorship on connections leaving the country that we cannot study.

We find that the direction of subsequent traffic is important in whether it is affected by residual censorship. If a client within a censored regime makes a forbidden request to a server outside, we find that only traffic sent by the client is affected by residual censorship. This makes sense: traffic direction is encoded in both 3-tuple and 4-tuple flow tracking. However, this does impose an important limitation on attackers: an attacker generally must be on the same side of the censor as their victim.

What packets are affected by residual censorship? Which packets are impacted by residual censorship changes de-

pending on the censorship mechanism used. China's HTTP residual censorship mechanism of injecting RST packets does not initiate until *after* the client has sent a new request in a PSH+ACK packet. None of the 3-way handshake is impacted; it reaches the server without interference. However, China's ESNI residual censorship (both 3-tuple or 4-tuple) null-routes: all packets leaving the client, including SYN packets are affected by the residual censorship.

We find the same effect for the null-routing residual censorship in Kazakhstan and Iran. Note that the direction of traffic matters for every censor we studied: only packets from the client are impacted. If a server sends packets in a connection being null-routed, the packets will reach the client unaffected.

How long does residual censorship last? To determine the duration of residual censorship, we performed an experiment in which we varied the duration of time between triggering censorship and making a follow-up request, and recorded whether residual censorship took place.

We find the duration of residual censorship also varies between countries and protocols, but is generally less than three minutes in every country we studied. HTTP residual censorship in China lasts approximately 90 seconds (as observed in [19], [2]) and ESNI is residually censored for 120 seconds (as observed in [5]). We note that for ESNI censorship in China, other researchers have reported both 120 and 180 seconds of residual censorship [5]. In Iran, while its protocol filter residually censors for 60 seconds, its HTTP and SNI censorship systems residually censor for 180 seconds (and in a small number of our experiments, the SNI system *continued* to residually censor requests up to approximately 5 minutes). In Kazakhstan, both HTTP and SNI residual censorship systems operate for 120 seconds.

We find that both Iran and Kazakhstan *restarts* their residual censorship timer if the client sends a matching packet, thereby extending the duration of time that the client is affected. Due to TCP retransmissions, in practice this means that Iran and Kazakhstan will drop traffic for much longer than their original time. This is presumably done to make their censorship systems more robust against TCP retransmissions. As we will see in the next section, however, this timer reset makes our attack easier to launch.

Does residual censorship require a full 3-way handshake? No! We were able to trigger residual censorship without a proper 3-way handshake for every censor we studied. To discover this, we followed the methodology of Bock et al. [11] to attempt subsets of the TCP 3-way handshake before sending a PSH+ACK with a censored keyword.

The Airtel ISP in India enacted residual censorship without *any* of the 3-way handshake (one needs only send the PSH+ACK). Censorship of clients within this ISP appears to maintain no TCP state for their censored system.

Other countries required a subset, but not the entirety, of the TCP 3-way handshake. We sent a single SYN packet with a decremented sequence number, followed by a PSH+ACK containing the forbidden payload (we will refer to these two packets as the "censorship trigger"). This successfully triggered censorship (and residual censorship) for every censorship system we studied.

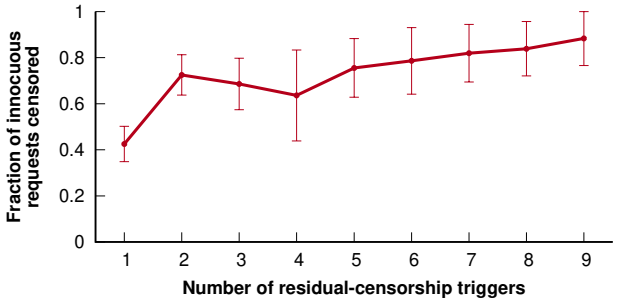

Fig. 2: The relationship between the number of times censorship is triggered and the reliability of HTTP residual censorship, as measured from our Beijing 2 vantage point. As the number of times residual censorship is triggered increases, the reliability improves. (Error bars represent 95% confidence.)

How reliable is residual censorship? We define the "reliability" of residual censorship as the fraction of follow-up innocuous requests made within the residual censorship window that are successfully censored. Note that this is distinct from the reliability of censorship itself, which traditionally refers to the fraction of forbidden requests a censor successfully censors [6].

We performed an experiment to measure residual censorship reliability from each of our censored vantage points. We triggered censorship and then made one innocuous request per second and recorded how many requests were impacted; this experiment was repeated 10 times, spaced evenly throughout a 24 hour period. For every protocol in Iran and Kazakhstan and for ESNI censorship in China, we find that 100% of our requests were residually censored as expected. For HTTP residual censorship in China, however, we find that only approximately 50% of our requests are correctly residually censored. We find this pattern holds bidirectionally.

We next explored if we could improve the reliability of HTTP residual censorship. We performed an experiment in which we varied the number of forbidden requests we made before starting our test innocuous queries. From our Beijing 1 vantage point, we varied the number times we issued forbidden requests between 1 and 9 times, and then made one innocuous request per second for one minute. We randomized the order of the trials, implemented 5 minutes of sleep between each, and issued innocuous test queries before starting each experiment to ensure that the experiments did not interfere with each other. We repeated this experiment 6 times.

Figure 2 shows the average fraction of innocuous queries that were censored as a function of the number of residual-censorship triggers we send ahead of time. We find that as we increase the number of forbidden queries, we improve the reliability of residual censorship and after seven retries, the success rate levels out.

We hypothesize that the GFW is internally load balancing queries from this vantage point and that different middleboxes within the GFW do not communicate with one another when residual censorship starts. As we add additional queries, we are more likely to trigger residual censorship with multiple middleboxes, thereby increasing the likelihood that as future requests are made, they will get routed through a middlebox with active residual censorship.

V. Residual Censorship Attack

The results from our measurement of residual censorship indicate that it would be possible for an off-path attacker to get a victim's connections residually censored. Because censors do not look for the entire 3-way handshake, an attacker could simply source-spoof the victim, send a censored request, thereby residually censoring communication between the victim and server.

In this section, we empirically evaluate the feasibility of this attack by launching it against ourselves.

A. Launching the Attack

Since all of our vantage points employed egress filtering, we cannot launch the attack directly from our censored vantage points within China, Iran, or Kazakhstan.

Instead, we leverage a public deployment of SP^3 (A Simple Practical & Safe Packet Spoofing Protocol) [17] deployed at the University of Washington, to ethically send source-spoofed packets and thus act as our attacker. SP^3 is a web server that offers the ability to send spoofed packets, but mandates that a client *consent* to receiving source-spoofed packets. A client gives this consent by creating and holding open a websocket connection to SP^3. When the client connects, SP^3 returns a UUID16 challenge string. As long as the websocket connection is held open, other servers can connect to SP^3 with a websocket, supply the challenge code, and can give SP^3 packets through binary frames to send to that client.

We launched the attack on ourselves as follows. We used SP^3 to send a sequence of packets to trigger residual censorship to a server that crosses the censor, with the source addresses spoofed to be a test victim under our control. Recall that traffic direction matters to residual censorship in each of these three countries: the attacker must be on the same side of the censor as the victim. Since SP^3 is located in the United States, this means we are launching the attack from outside-in for each censoring country. Fortunately, as we saw in Section IV, residual censorship is bidirectional for most of the protocols we study. Our vantage points within each country acted as the server; we launched the attack against all of our geographically disparate vantage points around the world as victims.

Then, we used our "victim" to make requests to the server, and recorded if the connection succeeded or if it was impacted by residual censorship. We varied our test request based on the protocol and type of residual censorship. For 3-tuple residual censorship, the client makes an innocuous request with a different source port to the same server IP address and port. For 4-tuple residual censorship, we ensure the client uses the same source port as the attacker. Of course, in a real attack scenario, the attacker cannot know the source port a victim will use *a priori*. Therefore, to weaponize 4-tuple residual censorship systems, the attacker would re-trigger censorship for all 65,535 possible source ports. We investigate the limitations imposed by this later in this section; for now to demonstrate the attack, we allow the attacker to access the source port.

We launched this attack against every uncensored vantage point for every bidirectional, residually censored protocol to each of our vantage points in China, with HTTP and ESNI, in Kazakhstan, with HTTP and SNI, and in Iran, with HTTP and SNI. Recall that Iran's protocol filter censorship cannot be triggered from outside the country, and therefore we omit it from these experiments. To determine attack reliability, we repeated each attack 20 times.

Before we launched each attack, we also record two traceroutes. First, we performed a regular traceroute between the victim and the destination. Second, we performed a source-spoofed traceroute using SP^3. Our server (inside the censored regime) connects to SP^3 and consents to receive TCP `SYN` packets with the `TTL` ranging from 1 to 30, with the source address of the packets spoofed to be the victim. While SP^3 sends these packets, the victim (a vantage outside of the censored country) records `TTL` "Time Exceeded" messages. This allows us to reconstruct the network path taken by the packets spoofed by SP^3, and compare it to the network path taken by the victim's test request.

B. Results

In every country we tested, we could successfully weaponize the censorship infrastructure against every victim vantage point at least once around the world. We find that the attack is sensitive to the chosen protocol (for example, HTTPS offers better results in Kazakhstan than HTTP). Table II presents an overview of our results.

Collectively, our results suggest that there are many shared paths through the censorship infrastructure of each country, and an attacker that can access just one source spoofed capable machine is capable of launching highly effective availability attacks. A more well resourced attacker could likely get even better results by choosing vantage points with even more similar paths as their victims.

In the remainder of this section, we detail the results in each of the countries we tested.

Kazakhstan In Kazakhstan, 100% of the attacks succeeded if the attacker triggered residual censorship with SNI payloads. However, we find that if a forbidden HTTP payload is used instead, the success varies depending on the victim vantage point, and this pattern persists irrespective of the port the attacker uses.

First, we explored why the success of the HTTP attack changes depending on the victim location. We hypothesize the reason for this is that the network path of the packets

Victim Location		Destination Location: Kazakhstan HTTP	Kazakhstan HTTPS	Iran HTTP	Iran HTTPS	Beijing 1 HTTP	Beijing 1 ESNI	Beijing 2 HTTP	Beijing 2 ESNI
Australia	**Sydney**	✓	✓	✓	✓	50%	10%	55%	✓
China	**Beijing 1**	✘	✓	✓	✓	N/A	N/A	N/A	N/A
	Beijing 2	✘	✓	✓	✓	N/A	N/A	N/A	N/A
India	**Mumbai**	✘	✓	✓	✓	✘	✘	✘	30%
	Bangalore 1	✓	✓	✓	✓	50%	10%	✓	✓
	Bangalore 2	✓	✓	✓	✓	25%	10%	✓	✓
Iran	**Tehran**	✓	✓	N/A	N/A	✘	50%	75%	✓
Ireland	**Dublin 1**	✘	✓	✓	✓	✘	✘	✘	5%
	Dublin 2	✘	✓	✓	✓	50%	✘	✘	✘
Japan	**Tokyo**	✓	✓	✓	✓	25%	✘	✘	✓
Kazakhstan	**Qaraghandy**	N/A	N/A	✓	✓	50%	✘	20%	✘
Russia	**Khabarovsk**	✓	✓	✓	✓	✓	✘	✓	✘
UAE	**Dubai 1**	✘	✓	✓	✓	85%	✘	95%	✘
	Dubai 2	✘	✓	✓	✓	✘	10%	✘	50%
USA	**Colorado**	✓	✓	✓	✓	✘	✘	✓	✘
	Iowa	✘	✓	✓	✓	✘	✘	✘	60%
	Virginia	✓	✓	✓	✓	50%	✓	55%	✘

TABLE II: Success rates in weaponizing each country's censorship infrastructure against each victim vantage point from our attacker in Seattle, WA. (✓ denotes 100%, ✘ denotes 0%, and N/A denotes a location that does not cross the border of the censor.) Note that the success rates are not always consistent, even to victims in the same country, or between censored protocols in each censored regime. Iran is consistent and reliable; Kazakhstan is consistently unreliable for HTTP, but consistently reliable for HTTPS. In China, however, the attack was not always consistent by protocol, victim location, or server location.

sent by the attacker and sent by the victim enter at different ingress points within the censor's infrastructure, and triggering censorship at one ingress does not initiate residual censorship at the other. To gain insight into this, we can compare the two traceroutes taken before the attack is launched: one from the attacker and one from the victim. Although both traceroutes are performed with the same source IP address, since they start from different geographic locations, the packets will necessarily take (at least partially) different paths to reach the server. By comparing the paths taken for each traceroute, we can try to determine if the paths converged *before* the packets reached the censor, or afterwards. If the paths converge after the packets reach the censor, it is possible that the attacker's traffic and victim's traffic will take different ingress points, and therefore be processed by different censoring middleboxes. To determine how many hops away the censor is from the server inside the censoring regime, we send `TTL`-limited forbidden queries until we initiate censorship. We find that our vantage point inside Kazakhstan is 5 hops away from the censor. Necessarily, this analysis will not be perfect; many routers and middleboxes can simply choose not to send a `TTL` Time Exceeded message and hide themselves from this analysis.

Nevertheless, for all victims for which the attack failed, we find that paths do not converge until less than 5 hops away from reaching the server.

Why then, even for victims with paths that do not converge, does the attack succeed when HTTPS is used, even when the same destination ports are used as in HTTP? Frankly, we do not know. We hypothesize this could be due to Kazakhstan having physically fewer HTTPS censoring middleboxes, and therefore fewer internal paths for the attacker and victim's traffic to be split between.

What sending rate is required for an attacker to weaponize Kazakhstan's censor to block a 3-tuple (source IP address, destination IP address, destination port)? Since both HTTP and SNI residual censorship can be triggered on any port, the attacker can choose to use whichever is more convenient. Both are 4-tuple residual censorship systems, which means the attacker must trigger censorship with the same source port that the victim will use. Since the attacker cannot know the victim's source ports ahead of time, instead the attacker will trigger censorship for all 65,535 possible source ports. It requires 2 packets to trigger censorship (a `SYN`, followed by a `PSH+ACK` with the forbidden payload), and once triggered, residual censorship will last for 120 seconds. Therefore, an attacker needs to send $2 \times \frac{65{,}535}{120}$ = 1,093 packets per second to sustain the attack indefinitely. The `SYN` packet is 54 bytes long (including the Ethernet header), but the length of the `PSH+ACK` will change depending on the protocol. Our HTTP trigger payload is 91 bytes long (54 bytes of headers and 37 bytes for the HTTP request), and our HTTPS trigger payload is 379 bytes long (54 bytes of headers and 325 bytes of TLS ClientHello). To sustain the HTTP attack, an attacker must be able to send (54 + 91) $\times \frac{65{,}535}{120}$ = 79,188 bytes per second, or 634 kbps. For HTTPS: (54 + 379) $\times \frac{65{,}535}{120}$ = 236,473 bytes per second, or 1,892 kbps.

Recall that we found no difference in reliability between HTTP and SNI, and therefore an attacker could opt to use the smaller HTTP triggers and reduce the amount of required bandwidth unless their victim was located in a geographically disadvantageous location.

Would it be advantageous for an attacker to try to trigger residual censorship with both protocols? We cannot be sure, but an attacker likely does not need to. Since both censorship systems reset the duration of their residual censorship anytime a matching packet is encountered, once the attacker triggers one censorship system, any packets sent to trigger the other will reset the timer for the first. We also note that the effects of censorship for HTTP and SNI are identical: for this reason, we cannot be certain whether packets being residually censored by one censorship system reach the other.

China The attack was inconsistent to both of our vantage points in China. The success rate of the attack varied based on multiple factors: the victim location, server location, and the chosen residually censored protocol.

As in Kazakhstan, we consulted the traceroutes to examine if the network paths could explain the lack of success for the attack. We repeatedly sent `TTL`-limited forbidden requests to determine how many hops both of our machines are away from the GFW (6 hops and 9 hops respectively). We hypothesized that the attack should succeed greater than 0% of the time if the paths converge before it reaches the censor. Recall from Section IV that in China, triggering HTTP residual censorship once does not guarantee that all future requests that match the 3-tuple will be censored; therefore, even if the attacker's and victim's paths converge before packets reach the GFW, we cannot guarantee success. Nevertheless, the traceroutes do not contradict our hypothesis: we find almost no path convergence for every victim against which the attack frequently failed (such as Ireland 1& 2).

Why are these success rates not either 100% or 0%, as in Iran and Kazakhstan? Bock et al. observed a similar phenomenon in [6] and posited that the GFW is a heterogeneous deployment of many different middleboxes, all running in parallel. We hypothesize that fractional success rates are caused by geographic variation in deployments of the GFW itself, and load balancing between multiple middleboxes running in parallel.

For an attacker, weaponizing the GFW poses an interesting opportunity, as it offers both types of residual censorship (3-tuple or 4-tuple) and multiple different censorship mechanisms (null routing or injected `RST`s). Attackers within the country can choose to trigger ESNI residual censorship at either the 3-tuple or 4-tuple with null routing, or trigger 3-tuple HTTP residual censorship to get injected `RST`s. Outside the country, ESNI censorship is limited to 4-tuple residual censorship, so the attacker can choose whether to launch one or the other depending on the location of their victim.

With 3-tuple censorship systems at an attackers disposal, weaponizing the GFW to prevent a victim from communicating with a given destination IP address and port is trivial. An attacker needs to trigger censorship only once to initiate the residual censorship, and can trivially re-send the censorship triggers to improve the reliability if needed. If 3-tuple residual censorship is unavailable, the attacker can fall back to leveraging 4-tuple residual censorship, as we demonstrated in Iran and Kazakhstan, which also lasts for 120 seconds. To trigger ESNI's 4-tuple residual censorship, the attacker must send a `SYN` (54 bytes), followed by the `PSH+ACK` containing the ESNI trigger (54 bytes for headers and 65 bytes of payload). An attacker needs to send $2 \times \frac{65,535}{120}$ = 1,093 packets per second, equivalent to (54 + 119) $\times \frac{65,535}{120}$ = 94,480 bytes per second, or 756 kbps to sustain the attack indefinitely.

Could an attacker simply try to invoke both censorship systems simultaneously in an attempt to improve the reliability of this attack? We find the answer is yes: the attacker can send multiple back-to-back packet sequences to trigger censorship using different protocols, as long as each source port is different. For example, the attacker can trigger 3-tuple HTTP residual censorship, followed by a trigger for 4-tuple ESNI censorship with a different source port. We find that if both triggers are sent with the same source port, only the first trigger will be successful. The reason for this was posited by [6]: once the HTTP censorship system sees the ESNI payload, it stops paying attention to the connection. However, since the HTTP residual censorship is 3-tuple, the attacker can use one source port to trigger the HTTP residual censorship system and still trigger 4-tuple residual censorship on all of the other source ports.

With both censorship systems performing residual censorship in parallel, which one affects a victim? We find the answer is the ESNI censorship system: this is because the ESNI residual censorship affects all packets, but the HTTP residual censorship system does not teardown a connection until after the 3-way handshake has completed. In our testing, we did not see an improvement in reliability when combining censorship triggers, but its utility may increase for victims in other geographic locations.

Iran Our attack was most successful in Iran. Here, 100% of the attacks succeeded using both forbidden HTTP and HTTPS (SNI) against every victim we tested. Both of these protocols are 4-tuple censored for a full 180 seconds, and both timers reset in the presence of any matching packet.

What is required for an attacker to effectively block a victim from communicating with a destination IP address and port across the censor? The attacker requires 2 packets to trigger censorship (a `SYN`, followed by a `PSH+ACK` with the forbidden payload), and once triggered, residual censorship will last for 180 seconds. Therefore, an attacker needs to send $2 \times \frac{65,535}{180}$ = 729 packets per second to sustain the attack indefinitely. The triggers are the same for Iran as for Kazakhstan: the `SYN` packet is 54 bytes long (including the Ethernet header), our HTTP trigger payload is 91 bytes long (54 bytes of headers and 37 bytes for the HTTP request), and our HTTPS trigger payload is 379 bytes long (54 bytes of headers and 325 bytes of TLS ClientHello). To sustain the HTTP attack, an attacker must be able to send (54 + 91) $\times \frac{65,535}{180}$ = 52,792 bytes per second, or 422 kbps. For HTTPS: (54 + 325) $\times \frac{65,535}{180}$ = 137,987 bytes per second, or 1.1 Mbps—a modest amount.

The length of the payload required to trigger SNI censorship is significantly larger than the payload required to trigger

HTTP censorship, and since each protocol worked equally well for our attacker, there is no incentive to use the longer SNI trigger. Of course, like in Kazakhstan, if the HTTP trigger fails for a given victim location, A bandwidth constrained attacker could opt to start with HTTP triggers and only switch to SNI triggers if their victim is in a disadvantageous geographic area.

VI. Attack Impact

Here, we reason about the potential impact of this attack by considering the potential breadth and limitations.

Breadth What is the true breadth of this attack? Unfortunately, we are limited by our vantage points to answer this definitively. Nevertheless, we can speculate about what other systems could potentially be weaponized.

We restricted our analysis only to censoring countries in which we could obtain vantage points that experienced residual censorship. Although we were unable to test this attack in India or Russia, prior work has found that other ISPs in India (Vodafone and Idea [21]) and Russia [22] employ null routing for censorship. Depending on how the null routing is implemented, these ISPs may be vulnerable to this attack, but we were unable to obtain vantage points within these systems to confirm this.

Our analysis assumed that either the server or victim is located physically inside a censoring regime. However, researchers in the past have observed that traffic that simply traverses the Internet borders of a censored regime can trigger censorship, even if neither the client nor server are located within the country [8]. Performing this attack against traversing traffic is an interesting area of future work.

We can also speculate about the breadth of this attack by examining the results of Quack, a powerful censorship scanning tool from Censored Planet [23]. Every day, Quack sends well-formed HTTP GET requests with potentially forbidden domains in the `Host:` header to echo servers around the world to identify interference. Quack records the cause of censorship and also monitors for 3-tuple residual censorship (called "stateful disruption"). In the December 27th, 2020 dataset, Quack had identified censoring middleboxes in 33 countries where 3-tuple stateful disruption was present and in 18 countries where null routing was used to censor. These results suggest that this attack may be significantly more broadly applicable.

Limitations Despite the potential breadth, there are limitations to this attack. An attacker must be able to obtain a vantage point (1) without egress filtering that (2) shares a similar enough path with their victim and (3) the traffic crosses a censor (4) with residual censorship (5) that can be triggered statelessly.

Our experiments suggest that there are a surprisingly high number of joint network paths, even for geographically disparate victims (such as Australia and USA). Still, not every attacking vantage point will be able to affect every victim, and the attacker has no mechanism to confirm whether their attack successfully blocked the victim.

Another potential limitation is that this attack may not work for every IP address. Researchers have observed in the past that some censorship systems vary their response based on the destination [1]. We were unaffected by this for all of our victim locations, but an interesting area of future work would be to repeat this study across a very broad range of IP addresses.

Lastly, there are some limitations to how completely an attacker could cut off two hosts. Could an attacker weaponize these censorship systems to *completely* cut two hosts from communicating? It depends on the type of residual censorship. We believe it is infeasible for an attacker to use a 4-tuple censorship system to completely prevent two IP addresses from communicating, as this would require triggering censorship for all 2^{32} possible combinations of source and destination ports. However, for a 3-tuple residual censorship system, the attacker could trigger residual censorship 65,535 times to all possible destination ports and accomplish this.

Does this attack become infeasible if middleboxes start properly tracking the 3-way handshake? Yes, but we believe it would be difficult for censors to do so. Particularly at the scale at which nation-state censors must operate, censors must content with path asymmetry: the network path used by traffic exiting the country may be different than the path used by traffic entering the country, even for the same connection. This makes properly tracking the 3-way handshake difficult: different middleboxes may see the `SYN` packet from the client than those that see the `SYN+ACK` packet from the server.

Can the attacker trigger residual censorship for UDP-based protocols as well? In our experiments, we only identified residual censorship for TCP-based protocols. However, this is only a partial limitation, since all of the null-routing residual censorship we studied affected both TCP and UDP traffic. If an attacker wishes to interfere with UDP traffic, she can simply trigger null-routing residual censorship over TCP and the victim's UDP traffic will be censored.

VII. Mitigations

In this section, we discuss our recommendations to potential victims and censoring regimes to mitigate this attack.

A. Censors

Null-routing should track sequence numbers, or should not be used. All of the null-routing censorship systems we study (Iran, Kazakhstan, and China's ESNI censorship) operate only at the 4-tuple, and do not do any validation of the sequence or acknowledgment numbers of the packets they drop. Unfortunately, this implementation of censorship with null-routing is inherently flawed. TCP is designed to be tolerant to packet loss, so most end-hosts will continue to retry sending packets when confronted with null-routing. This forces censors to maintain the flow's null-routing for a long enough period of time to exceed the duration of time that network stacks will retransmit (or further reset their internal timer when an offending packet is sent). Unfortunately, the longer this window of time is, the easier it is for an attacker to abuse null-routing to perform this attack. Therefore, to

eliminate 4-tuple residual censorship, we recommend that middleboxes who use null-routing only drop packets with the correct sequence and acknowledgment numbers, or to avoid using null-routing entirely.

Eliminate (or modify) 3-tuple residual censorship. Presumably, 3-tuple residual censorship is designed as a deterrent system: users who search for a forbidden term are "punished" and forbidden from trying to communicate with the same server again for a small period of time. Unlike 4-tuple residual censorship, the effect of 3-tuple residual censorship is salient to the user. However, we question the efficacy of this feature as a deterrent, since there is no communication or information to the end-user to alert them *why* they are continually being censored in all countries we tested in (China, Iran, Kazakhstan). Consider a user in China that searches for a long string of text containing a single verboten word. The GFW only sends `RST` packets: it does not inform the user the cause of censorship, and an uneducated user may be unaware that censorship is the reason their subsequent connections continue to fail. Worse, as we showed in Section IV, residual censorship is not even always be effective, and can fail depending on the users network route. For these reasons, we recommend that middleboxes—particularly the GFW—remove their residual censorship components altogether or modify their response from null routing to sending a block page or some response that indicates to the user who is being censored that they are being "punished" for their search.

We also echo many of the suggestions made by Bock et al. [11], as the root of our attack also stems from the ability to trigger censorship systems without a proper 3-way handshake.

B. Potential Victims

Unfortunately, once the attack is initiated, there is very little a victim can do to stop it. Nevertheless, we make recommendations here to mitigate or work around this attack.

Use a proxy. Since our availability attack is generally limited by the 3-tuple or 4-tuple, changing the source IP address that the censor sees is an effective way to bypass the attack. Therefore, we recommend that an affected user switch to use some proxying system, such as VPN, Tor, or an HTTP proxy. Further, a victim can rapidly rotate between proxies in an effort to stay ahead of an attacker. Unfortunately, this is only a stopgap solution; if the path from the victim to the proxy's entry nodes also crosses the censor, an attacker can simply switch to attacking the proxy itself.

Do not immediately try to reconnect. In some censorship systems, the presence of additional matching traffic causes the residual censorship timer to reset, thereby prolonging the attack. Therefore, if a user is affected, they should not continue trying to reconnect; instead, they should stop sending network traffic and wait a few minutes.

VIII. Ethical Considerations

Experiment Design We took care in designing our experiment to ensure that it would not involve or cause harm to any other users. Our experiments do not induce any in-country clients outside of our control to send forbidden requests; all communication was strictly between hosts we fully controlled. To the best of our knowledge, none of our vantage points in-country were NATted with other hosts, making it unlikely other users were affected.

Responsible Disclosure It is difficult to responsibly disclose our findings, as the affected censorship systems have historically been unresponsive to similar issues [11] or unwilling to intentionally weaken their censorship systems. Nevertheless, we are in the process of contacting several country-level Computer Emergency Readiness Teams (CERT) that coordinate disclosure for their respective countries.

IX. Conclusion

In this work, we demonstrated that it is possible to weaponize the censorship infrastructure in Iran, Kazakhstan, and China to perform availability attacks. We launch this attack against 17 different geographically disparate victims and show that even a weak attacker (with access to a single low-bandwidth source spoofer) can launch effective availability attacks. Collectively, these results show that the negative impact of censorship extends well beyond the censor's borders, and that they pose an even larger threat to the Internet writ large.

Acknowledgments

We thank our shepherd Kevin Borgolte and the anonymous reviewers for their helpful feedback; Will Scott for his support with SP^3; and our collaborators from the OTF and OONI communities for contributing insights and resources that made this work possible. Also, we thank the anonymous Artifact Evaluators for their diligent efforts. This research was supported in part by the Open Technology Fund and NSF grants CNS-1816802 and CNS-1943240.

References

[1] K. Bock, Y. Fax, K. Reese, J. Singh, and D. Levin, "Detecting and evading censorship-in-depth: A case study of irans protocol whitelister," in *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2020.

[2] K. Bock, G. Hughey, X. Qiang, and D. Levin, "Geneva: Evolving Censorship Evasion," in *ACM Conference on Computer and Communications Security (CCS)*, 2019.

[3] R. Ensafi, P. Winter, A. Mueen, and J. R. Crandall, "Analyzing the Great Firewall of China Over Space and Time," in *Privacy Enhancing Technologies Symposium (PETS)*, 2015.

[4] Z. Chai, A. Ghafari, and A. Houmansadr, "On the importance of encrypted-sni (esni) to censorship circumvention," in *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2019.

[5] K. Bock, iyouport, Anonymous, L.-H. Merino, D. Fifield, A. Houmansadr, and D. Levin, "Exposing and circumventing china's censorship of esni," 2020.

[6] K. Bock, G. Hughey, L.-H. Merino, T. Arya, D. Liscinsky, R. Pogosian, and D. Levin, "Come as you are: Helping unmodified clients bypass censorship with server-side evasion," in *ACM SIGCOMM*, 2020.

[7] R. Ensafi, D. Fifield, P. Winter, N. Feamster, N. Weaver, and V. Paxson, "Examining How the Great Firewall Discovers Hidden Circumvention Servers," in *ACM Internet Measurement Conference (IMC)*, 2015.
[8] Anonymous, "The Collateral Damage of Internet Censorship," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 42, no. 3, pp. 21–27, 2012.
[9] R. Ensafi, "CensoredPlanet Raw Data," https://censoredplanet.org/data/raw.
[10] D. Nobori and Y. Shinjo, "VPN Gate: A Volunteer-Organized Public VPN Relay System with Blocking Resistance for Bypassing Government Censorship Firewalls," in *Symposium on Networked Systems Design and Implementation (NSDI)*, 2014.
[11] K. Bock, A. Alaraj, Y. Fax, K. Hurley, E. Wustrow, and D. Levin, "Co-opting Middleboxes for TCP Reflected Amplification," in *USENIX Security Symposium*, 2021.
[12] Y. Cao, Z. Qian, Z. Wang, T. Dao, S. V. Krishnamurthy, and L. M. Marvel, "Off-Path TCP Exploits: Global Rate Limit Considered Dangerous," in *USENIX Security Symposium*, 2016.
[13] Y. Gilad and A. Herzberg, "Off-Path Attacking the Web," in *USENIX Workshop on Offensive Technologies (WOOT)*, 2012.
[14] M. Kührer, T. Hupperich, C. Rossow, and T. Holz, "Hell of a Handshake: Abusing TCP for Reflective Amplification DDoS Attacks," in *USENIX Security Symposium*, 2014.
[15] F. Adamsky, S. A. Khayam, R. Jäger, and M. Rajarajan, "P2P File-Sharing in Hell: Exploiting BitTorrent Vulnerabilities to Launch Distributed Reflective DoS Attacks," in *USENIX Workshop on Offensive Technologies (WOOT)*, 2015.
[16] J. Bushart, "Optimizing Recurrent Pulsing Attacks using Application-Layer Amplification of Open DNS Resolvers," in *USENIX Workshop on Offensive Technologies (WOOT)*, 2018.
[17] W. Scott, "A Secure, Practical & Safe Packet Spoofing Service," in *ACM ASIA Conference on Computer and Communications Security (ASIA CCS)*, 2017.
[18] S. Aryan, H. Aryan, and J. A. Halderman, "Internet Censorship in Iran: A First Look," in *USENIX Workshop on Free and Open Communications on the Internet (FOCI)*, 2013.
[19] Z. Wang, Y. Cao, Z. Qian, C. Song, and S. V. Krishnamurthy, "Your State is Not Mine: A Closer Look at Evading Stateful Internet Censorship," in *ACM Internet Measurement Conference (IMC)*, 2017.
[20] J. Beznazwy and A. Houmansadr, "How china detects and blocks shadowsocks," in *ACM Internet Measurement Conference (IMC)*, 2020.
[21] T. K. Yadav, A. Sinha, D. Gosain, P. K. Sharma, and S. Chakravarty, "Where The Light Gets In: Analyzing Web Censorship Mechanisms in India," in *ACM Internet Measurement Conference (IMC)*, 2018.
[22] R. Ramesh, R. S. Raman, M. Bernhard, V. Ongkowijaya, L. Evdokimov, A. Edmundson, S. Sprecher, M. Ikram, and R. Ensafi, "Decentralized control: A case study of russia," in *Network and Distributed System Security Symposium (NDSS)*, 2020.
[23] B. VanderSloot, A. McDonald, W. Scott, J. A. Halderman, and R. Ensafi, "Quack: Scalable Remote Measurement of Application-Layer Censorship," in *USENIX Security Symposium*, 2018.

APPENDIX

This appendix contains this work's submission for artifact evaluation, which received the Open Research Objects (ORO), Research Objects Reviewed (ROR), and Results Reproduced (ROR-R) badges[4].

A. Artifact Abstract

Nationwide Internet censorship threatens free and open access to communication and information for millions of users living inside of censoring regimes. In this paper, we show that this poses an even greater threat to the Internet than previously understood. We demonstrate an attack that exploits a little-studied but widespread feature of many censoring infrastructures: what we call residual censorship, in which a censor continues blocking traffic between two end-hosts for some time after a censorship event. Our attack sends spoofed packets with censored content, keeping two victim end-hosts separated by a censor from being able to communicate with one another. Although conceptually simple, this attack has several challenges, which we address. We demonstrate the feasibility of the attack through two studies: one to capture the current state of residual censorship, and another to actually launch the attack (against machines we control). We show that the attack can be launched despite stateful TCP tracking used by many censors, and that it also works against those who censor by null-routing. Our code is publicly available.

B. Disclaimer

These scripts will intentionally trigger censorship responses with malformed or non-protocol-compliant packet sequences that are detectable on the network. These scripts do not provide any anonymity. Understand the risks of testing them before doing so.

C. Try it

There are two scripts in this repository: one for triggering and identifying residual censorship and a second to launch this attack from a source spoofed attacker.

Note that to prevent abuse, this code is not useful to launch this attack at scale: these scripts are for testing small-scale attacks and to reproduce the results in our paper.

D. Set up

Just install the dependencies and you're good to go.

```
python3 -m pip install -r requirements.txt
```

Before using the residual censorship scanner script, ensure that outbound RST packets are being dropped. You can use the `drop_outbound_rsts.sh` script for this.

E. Identifying Abuseable Residual Censorship

The residual_censorship_scanner.py script is used to identify abusable four-tuple residual censorship (null-routing). The script is designed to be run from a client (located either inside or outside a censored regime) to an echo server located on the other side of the censor.

The script attempts to trigger censorship statelessly (without properly completing a 3-way handshake), and then sends various follow-up test packets and checks if the server gets them. In total, it performs 6 tests; if all six pass, the censorship system is likely abusable. If some tests fail, it is possible the censorship system is not abuseable, or that the script is simply running in an unexpected environment and needs modification.

The script is designed specifically for an echo server on one side, and expects a censor that performs four-tuple null-routing censorship.

Before running, ensure outbound RST packets are being dropped, as they can interfere with the script. You can use the provided `drop_outbound_rsts.sh` script for this.

`Example:`

[4]Note that the ROR-R badge implies ROR.

```
$ sudo python3 residual_censorship_scanner.py
<ip address> 7 --host avaaz.org
...
Summary:
- 3-way handshake, tested with a well-formed
  innocuous query on PSH+ACK packet, with
  a good seqno/ackno: censored.
- 3-way handshake, tested with a malformed
  innocuous query on PSH+ACK packet,
  with a good seqno/ackno: censored.
- No 3-way handshake, tested with a well-formed
  innocuous query on PSH+ACK packet, with
  a good seqno/ackno: censored.
- 3-way handshake, tested with a SYN packet,
  with a good seqno/ackno: censored.
- No 3-way handshake, tested with a SYN packet
  with a good seqno/ackno: censored.
- No 3-way handshake, tested with a SYN packet,
  with a different seqno/ackno: censored.
Abusable residual censorship detected.
```

This script can also be used to determine the duration of time residual censorship lasts with `--determine-duration`.

F. Source IP Address Spoofing: Launching the Attack

To test launching the attack against yourself, you can use `sp3_send.py`. This relies on a public instance of the amazing SP^3 project (see `https://github.com/willscott/sp3`). SP3 is a service that allows a client to consent to receiving spoofed traffic. If you set up a different SP3 server yourself, you can override the default and use that; otherwise, you can use the default public instance of SP3 located in the University of Washington.

To use this script to test the attack, you must use it from a machine located inside a censored regime (this is because this attack relies on the attacker and victim being on the same side of the censor, and SP3 is located in the United States). When you run the script, that machine will connect to SP3 with a websocket connection to consent to receiving spoofed traffic and then give SP3 packets to send to it. By controlling the source address of those packets to a machine you control, you can ethically test if this attack could feasibly block communication between your two IP addresses.

Note that to prevent abuse, this script will only trigger censorship for a single given source port. To test if the resulting four-tuple residual censorship is affecting you, you can give SP3 a specific source port and then use curl with `--local-port`, such as: `curl <ip-address-of-machine-in-censored-regime>:<port> --local-port 22222`.

The script supports four different payloads to send with: a forbidden HTTP request, HTTPS request (youporn in the SNI field), an ESNI request, and a series of malformed bytes.

```
$ python3 sp3_send.py --help
SP3 Spoofing Script

optional arguments:
  --public-ip PUBLIC_IP
      IP address of this computer
  --victim-ip VICTIM_IP
      IP address of victim computer
      (who traffic should be spoofed as)
  --protocol {http,https,malformed,esni}
      payload protocol to send with.
  --sport SPORT
      source port to use
  --perform-sp3-traceroute
      instead of launching the attack,
      perform an sp3 traceroute
  --sp3 SP3
      The URI IP:port of the sp3 server
```