Iran: A New Model for Censorship

POSTS

March 18, 2020

Summary: Ahead of its February 21st elections, Iran subtly deployed a second censorship system: a *protocol whitelister.*

In this article, we will describe the protocol whitelister, how it works, and **how it can be defeated**.

- The protocol whitelister only allows ("whitelists") a small list of protocols to be used, posing a threat to almost all existing censorship-evasion tool deployments (VPNs, Tor, Proxies, etc).
- Due to its design, the whitelister is challenging to detect, measure, or study from outside the country. We present how we performed our measurements, and report on who the whitelister targets and what protocol fingerprints it uses.
- We deployed *Geneva* against the protocol whitelister and have discovered multiple ways to circumvent it. We also identify mistakes Iran made in crafting the fingerprints.

As of time of writing, the whitelister is still in effect from all of our vantage points within Iran.

> All of our experiments were performed across six vantage points in Iran, located in Isfahan, Razavi Khorasan, Fars, Tehran, and Zanjan to machines we controlled in Microsoft Azure, Amazon EC2, and DigitalOcean. While this is representative, it is possible that we are not seeing the full picture; if you would like to help broaden our measurements, please contact us.

A Quiet Deployment

Around the time of Iran's parliamentary elections, reports began surfacing that Iran was degrading internet traffic crossing its borders.

Content of link to tweet is unavailable or may have been removed by the author.

سرعت اینترنت #وی_پی_ان #ایران #اندروید
#سرعت #اینترنت #فیلترشکن #پروکسی
#eclipsevpn #vpn #iran #internet
pic.twitter.com/aplRfTizpb

— Kamran M (@MKami823)
February 18, 2020

However, while internet degradation was taking place, Iran subtly deployed a **secondary censorship system**, a *protocol whitelister*, which garnered much less attention. The protocol whitelister only affects traffic leaving Iran and is minimally invasive to most non-forbidden traffic, making it harder for researchers to detect.

Protocol Whitelisting

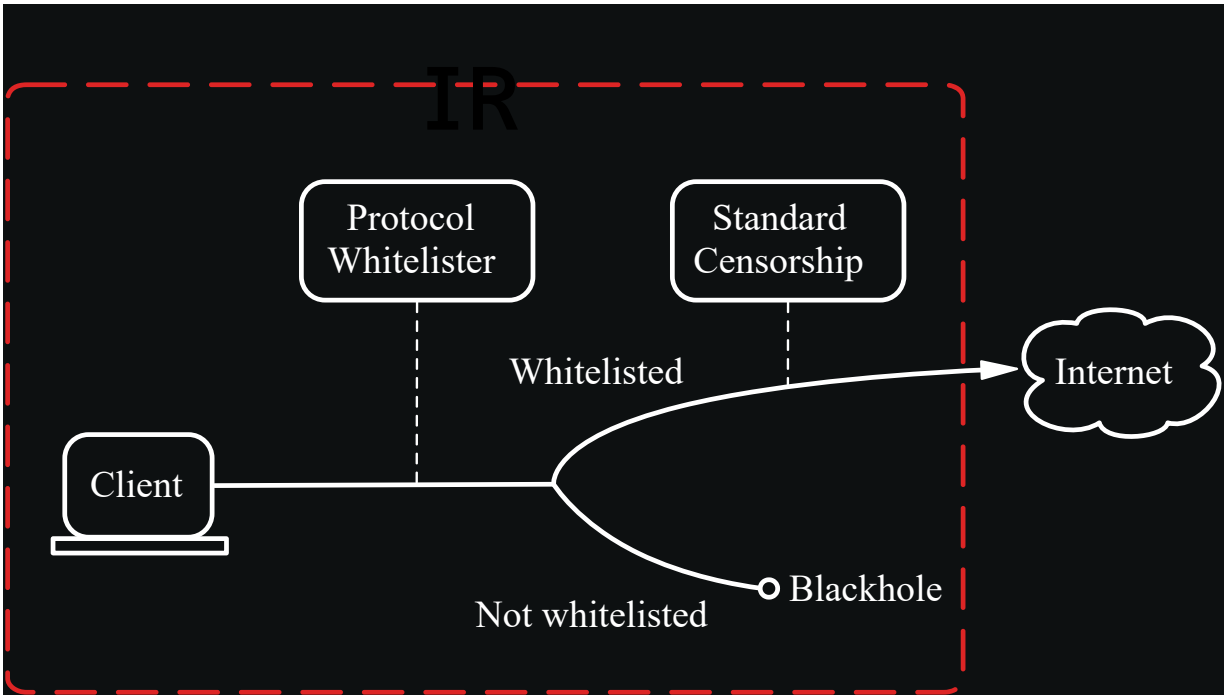
Iran has extended its censorship infrastructure to include a *protocol whitelister* at a high level, a protocol whitelister inspects Internet traffic and drops all connections that do not match an allowed fingerprint.

How Iran's Protocol Whitelister Works

The whitelister performs deep packet inspection (DPI) on the packet payloads sent by clients in Iran; if the data sent at the start of the connection does not match an allowed protocol fingerprint, the rest of the flow from the client is dropped.

Iran's protocol whitelisting system affects only TCP traffic on port 53, 80, 443, and only to specific IP ranges. The whitelister has fingerprints to match for DNS, HTTP, and HTTPS traffic, but each protocol is not bound to is associated port: the whitelister will match all three protocols on any of the three ports.

At the start of a connection, the whitelister **monitors the first two packets** from the client. If *either* of those two packets matches a protocol fingerprint, the flow is unharmed; if no packet does, the second packet and rest of the flow from the client is blackholed (all packets in the flow are dropped). The whitelister will continue to drop the offending flow's network traffic for 60 seconds, but each time an additional packet is sent in a flow, this 60 second timer resets. This means that in practice, because TCP will retransmit packets that are not acknowledged, an offending flow will be affected by the whitelister for much longer than 60 seconds.



The system works in tandem with Iran's standard censorship system: the protocol whitelister ensures communication only occurs over certain protocols that the regular censorship system can verify.

How to Trigger the Whitelister

To trigger the whitelister, one can open port 53, 80, or 443 on a machine outside of Iran (`nc -lvp 80`) and then connect to it from inside of Iran (`nc <ip> 80`). Sending a simple message twice (`test<Enter>test`) is sufficient to trigger the whitelister—if the whitelister is running and affects the destination IP, only the first message will reach the server.

Like Iran's regular censorship systems, the whitelister does not check checksums and is incapable of reassembling TCP segments.

How Clients and Servers are Affected

The whitelister is not bidirectional. This means that it only affects connections where the client is inside Iran, making it more challenging for researchers to probe and study from outside the country.

The server receives almost no indication censorship has taken place. Unlike with the Great Firewall which sends RSTs in both directions, once the whitelister steps in to censor the traffic flow, it only affects the packets leaving the client: packets from the server are unharmed. **Although the client will still get the server's data, the client is unable to ACK or respond to any data.**

Measuring the Protocol Whitelister

These properties make it challenging to even *detect* the whitelister from outside of Iran. Here, we describe additional experiments we performed to learn who the whitelister affects, what its fingerprints are, and how to circumvent it altogether.

Affected IP Space

In order to identify which IPs are affected by the whitelister, we performed an experiment to test the effects of the whitelister on the Alexa top 20,000 list. To avoid the effects of DNS censorship or requesting IPs inside of Iran (thereby not crossing the whitelister), we used `dig` outside of Iran to get IP addresses for all 20,000.

Inside of Iran, we set up an experiment with two conditions. The first condition was a control: we made normal `GET` requests to all 20,000 IP addresses using a python socket, and the success or failure of the request was recorded. The second condition tested for the whitelister. We requested all 20,000 IP addresses again, this time sending `G`, `ET` and `/` in separate messages.

IP addresses that respond in the first condition but time out in the second condition are likely affected by the whitelister. We perform this experiment ten times to validate the results.

Over all 10 experiments, 3595 IPs (15.09%) came up as affected at least 8 times. Of those, 3499 came up as affected all ten times (14.77%). 278 (1.17%) IPs came up as affected between 3 and 7 times, inclusive.

Why are only certain IPs affected?

We explored whether the provider for an IP address was correlated with whether the whitelister affected it. Using whois lookups, we extracted the providers for affected and unaffected IPs.

Top 10 providers for affected IP addresses

Provider	Count
Amazon Technologies Inc.	1453
Cloudflare, Inc.	565
Akamai Technologies, Inc.	229
Amazon.com, Inc.	171
Fastly	167
DigitalOcean, LLC	146
Amazon Data Services Limited	97
RIPE Network Coordination Centre	92
Linode	64
Amazon Data Services	60

Top 10 providers for unaffected IP addresses

Provider	Count
Cloudflare, Inc.	4541
Unknown	1465
Google LLC	657
ALISOFT	657
Amazon Technologies Inc.	580
Asia Pacific NIC	544
RIPE Network Coordination Centre	537
Alibaba.com LLC	287
Amazon.com, Inc.	277
Akamai Technologies, Inc.	253

Overall, IP provider does not seem to be correlated with whether the whitelister affects an IP address.

Case Study: Cloudflare

As Cloudflare appears on both lists, we next explored how much of its infrastructure was affected by the whitelister. Cloudflare's IP prefixes are available publicly on their website. As many of these prefixes are prohibitively large, instead of testing every IP in each prefix, we sampled 256 IP addresses at random for each prefix to test.

We performed a similar experiment to the above: given a Cloudflare IP address, we make two requests to it (first normally, second segmented); IP addresses that respond in the first condition but time out in the second condition are likely affected by the whitelister. We repeated this experiment 5 times for each prefix.

We found that only 2 of Cloudflare's prefixes contained IPs that were affected by the whitelister: `104.18.0.0/16` and `104.31.82.0/24`. Both these entire prefixes appear affected, but none of the other prefixes on Cloudflare's website contained IP addresses in our experiments that appeared affected by the whitelister.

Summary

Frankly, it is not clear why the whitelister affects the IP addresses it does. Anecdotally, all of our vantage points in Amazon EC2 and DigitalOcean around the world were affected by the whitelister, but our Alexa experiment suggests that not all IP addresses managed by Amazon or DigitalOcean are affected. Which IPs are affected may be an artifact of the varied network routes taken from our vantage points, or that Iran selected specific regions of the IPv4 address space as more important to filter. We leave a further investigation into why the whitelister affects what it does to future work.

Fingerprints

We reverse engineered the whitelister's fingerprints for each protocol. Knowing these fingerprints can be a powerful tool for evaders: by injecting a fingerprint at the start of a connection, the whitelister can be bypassed. Since the whitelister will match any of these fingerprints on all three ports, any fingerprint can be used on any whitelisted ports.

DNS Fingerprint

The DNS protocol matcher seems to OK flows if the following conditions are met:

- TCP payload must be ≥ 12 bytes
- Structure of the TCP payload must be a valid DNS-over-UDP header, not a DNS-over-TCP header. Recall that the whitelister is *only active* over TCP - this means their whitelister will *never match a legitimate DNS-over-TCP packet.* The reason this has not caused a significant issue is because the whitelister only affects data from the client *after* the first packet, and there is no second packet from the client in a traditional DNS-over-TCP connection.
- Specific limitations exist for the fields within the DNS header: QR must be 0, QDcount must be less than 15, and ANcount must be 0.

Example: `\x00\x00\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00`

HTTP Fingerprint

The HTTP protocol matcher seems to OK flows if the following conditions are met:

- The TCP payload must be ≥ 8 bytes
- The payload must start with a specific HTTP verb followed by one space; e.g. `"GET "` or `"POST "`
- The whitelister can match GET, POST, HEAD, CONNECT, OPTIONS, DELETE, and PUT. TRACE and PATCH do not work.

Example: `GET testing123`

HTTPS Fingerprint

The HTTPS protocol matcher seems to OK the flow if the following conditions are met:

- The TCP payload must be ≥ 41 bytes: 5 bytes for the TLS header, 36 for the TLS Client Hello
- The length field of the TLS Header equals the length of the Client Hello
- The TLS version header (bytes 2 and 3 of the TCP payload) is TLS 1.0, 1.1, or 1.2, (`\x03\x01`, `\x03\x02`, `\x03\x03`, respectively). Note that real TLS 1.x Client Hellos all have TLS 1.0 in this field, so this criterion has no practical difference.

After the first 5 bytes of the packet (the type, the version, and the length, 1, 2, and 2 bytes respectively), the whitelister does not look at any contents of the Client Hello. Writing garbage bytes to the remaining bytes of the Client Hello does not trip the whitelister.

Example TCP Payload: `\x16\x03\x01\x02\x00...`, where `\x16` is the indication of a handshake, `\x03\x01` is TLS version (1.0), and `\x02\x00` is the length of the Client Hello (512 bytes)

Using Geneva to Bypass the Whitelister

> **Geneva (*Gen*etic *Eva*sion)** is a genetic algorithm we developed that *evolves* censorship evasion strategies against a censor. Unlike most anti-censorship systems, it does not require deployment at both ends of the connection: it runs exclusively at one side (client or server) and defeats censorship by

manipulating the packet stream to confuse the censor without impacting the underlying connection. A *censorship evasion strategy* describes how that traffic should be modified. Since Geneva will be evolving these strategies, they are expressed in a domain-specific language that comprises the DNA of each strategy. (For a full rundown of Geneva's strategy DNA syntax, see our Github page).

Geneva is comprised of two main components. First, the genetic algorithm, which can evolve new ways to defeat a censorship system given an application that experiences censorship and a fitness function. Second, the strategy engine, which applies a given strategy on the fly to modify active network traffic.

We trained Geneva against the protocol whitelister. We wrote a custom fitness function for Geneva which connected to a vantage point outside of Iran and repeatedly send non-whitelisted messages. Using this fitness function, Geneva could test and train strategies directly against the whitelister. *In under two hours*, it discovered two simple strategies that defeat it (beyond simply injecting an innocuous HTTP request). In this section, we will explore these surprisingly simple strategies.

The Geneva strategy engine is open source on our Github, so all of these strategies can be deployed and used by anyone. Since they operate at the TCP layer, they can be applied to any application: with Geneva running, even an unmodified web browser can become a simple censorship evasion tool. To learn more about how Geneva (or the Geneva strategy engine) works under the hood, see our papers or about page.

Note that Geneva is *not* designed as a general purpose evasion tool, and does not provide any additional encryption, privacy, or protection. It is a beta research system and it is not optimized for speed. Use these strategies at your own risk.

Circumvention Strategy 1

The first strategy works by simply sending two additional packets before the 3-way handshake: two empty packets with the `FIN` flag set.

In Geneva's strategy DNA syntax, this strategy looks like this:

```
[TCP:flags:S]-duplicate(tamper{TCP:flags:replace:F}(duplicate,),)-|
```

This triggers on all outbound TCP `SYN` packets and duplicates them; to the first duplicate, the TCP `flags` field is set to `FIN`, and a second copy of this packet is made. The second duplicate of the `SYN` is unchanged. All three packets are sent on the wire. When the packets arrive at the server, the server ignores the `FIN` packets, since they are not associated with any connection yet, but the whitelister processes them, causing it to ignore the rest of our connection. The `SYN` arrives as normal and our connection can start unchanged.

We do not understand why this strategy works, though we hypothesize the `FIN` packets tricks the whitelister into thinking it has already missed the relevant data packets, causing it to ignore the rest of the flow.

Circumvention Strategy 2

The second strategy is stranger than the first. This strategy works by sending *nine copies* of the `ACK` packet during the 3-way handshake.

In Geneva's strategy DNA syntax, this strategy looks like this:

```
[TCP:flags:A]-
duplicate(duplicate(duplicate,duplicate),duplicate(duplicate,duplicate(duplicate(duplicate,),)))-|
```

By sending nine ACKs during the 3-way handshake, the whitelister ignores the rest of our packets. We hypothesize this works because the whitelister has some internal limit on the number of packets it processes for a given flow, and by sending these 9 packets, we can exceed this total, causing the whitelister to ignore the rest of our connection.

Circumvention Strategy 3

Beyond exploiting the whitelister's shortcomings at the TCP layer, we can also use Geneva's strategy engine to bypass the whitelister by simply injecting one of the protocol fingerprints into the start of the connection. Here is one such strategy that injects the HTTP fingerprint into the connection stream before each data packet in the connection:

```
[TCP:flags:PA]-duplicate(tamper{TCP:load:replace:GET%20testing123}(tamper{TCP:chksum:corrupt},),)-| \/
```

Using a Geneva Strategy

To deploy one of these strategies, we can use Geneva's strategy engine (open source on our Github page) to apply these strategies to our network traffic. Since the engine captures network traffic on a specified port, any application sending data on that port will be affected by the strategy. For example, we can test that these strategies work by trying to trigger the whitelister with `nc` as above with the engine running in the background.

```
$ STRATEGY="[TCP:flags:PA]-duplicate(tamper{TCP:load:replace:GET%20testing123}(tamper{TCP:chksum:corrupt},),)-| \/"
$ sudo python3 engine.py --strategy "$STRATEGY" --server-port 80 --log info &
$ nc <ip> 80
test
test
# connection is still successful
```

In this case, we used Geneva to defeat the whitelister for a simple netcat application. We can also use this to defeat whitelisting and run a VPN or similar system on port 80.

Conclusion

Much censorship evasion design today is couched in this idea of maximizing *collateral damage* for the censor. The idea is that if a censor wants to shut down an anti-censorship system or block a resource, we should make it as costly as possible for them to do so. Usually this is done by designing the system in such a way that it forces censors to block much more than they want to, causing collateral damage. For example, running bridges/proxies/VPNs on EC2 has been popular under the assumption that a censor would have to block all of AWS in order to shut them down, which would cause enormous collateral damage due to all the websites that are hosted on AWS.

The whitelister represents an attempt for Iran to sidestep some of this collateral damage. By only allowing content from specific protocols, they can shutdown or degrade most anti-censorship tools without negatively impacting regular websites or services.

Iran has had a greater capacity for censorship than they have exercised in the past, and these systems can pose a threat to existing deployments of censorship-evasion tools (VPNs, Tor, etc). Worse, the unidirectional and non-universal nature of the whitelister makes it more challenging for researchers outside of the country to identify and study. With tools like Geneva, we hope to shorten the response time for the censorship evasion community and alert tool developers how they can make their tools more resilient in the ever-changing censorship landscape.

Citations

[1] Protocol whitelisting was briefly described in 2013: https://censorbib.nymity.ch/pdf/Aryan2013a.pdf, as a capability they turned off after the 2013 elections were over.

Update (3/21/2020): Since this post was written, we were informed the Shatel ISP in Iran has announced the right to restrict access to **any ports or protocols** without informing the network. Combined with the whitelisting system, this poses a potentially threat to anti-censorship tools if only whitelisted ports are available.

Update 3/22/2020): @narimangharib noted that Shatel have been doing this since 2018.