HTTPT: A Probe-Resistant Proxy

Sergey Frolov, Eric Wustrow
University of Colorado Boulder
Proxies

Censored User

Censor-Controlled Network

obfs3 proxy
Active Probing

Censored User

Censor-Controlled Network

Let’s confirm!

obfs3 proxy
Active Probing

Censored User

Censor-Controlled Network

*speaks obfs3*
Active Probing

Censored User

Censor-Controlled Network

*speaks obfs3 back*

obfs3 proxy
Active Probing

Okay, now I can safely block this endpoint.
Thwarting Active Probing

- Probe-Resistant proxies
  - Require knowledge of *shared secret* to use
  - Don’t know secret? Server remains *silent*
Thwarting Active Probing

Censored User

Censor-Controlled Network

*Tries to speak obfs4 without knowing server’s password*

obfs4 proxy
Thwarting Active Probing

Censored User

Censor-Controlled Network

*Remains silent*

obfs4 proxy
Thwarting Active Probing

Can’t tell if I can block this simply by speaking obfs4. What else can I do?

Censored User

Censor-Controlled Network

obfs4 proxy
Advanced Active Probing

Replay Attack - censor resends observed client messages

Some probe-resistant proxies implement a nonce cache to prevent replays of previous connections. However,

- GFW thwarts this defense by permuting replays [*]
- The behavior of not responding at all to replays may be unusual

[*] https://gfw.report/blog/gfw_shadowsocks
Advanced Active Probing

Fingerprint the server:

1. Send probes using a few popular protocols: ~94% of servers[*] respond with data to at least one popular protocol
2. Remaining “non-responsive” applications are fingerprintable further using Close Threshold and Close Timeout[*]

[*] Frolov, S., Wampler, J., and Wustrow, E. “Detecting Probe-resistant Proxies”. NDSS 2020
Thwarting Advanced Active Probing

Instead of trying to achieve probing resistance by not responding, we hide our proxy server behind another popular server application, which would

- Tunnel circumvention traffic
- Provide natural responses to the censors’ probes.

We evaluate use of HTTPS with Web Servers
Why HTTPS

HTTPS is common and crucial for the Internet

- Unlikely to be blocked outright

[*] https://letsencrypt.org/stats/
Why HTTPS

- HTTPS is heterogenous
- HTTPS is used for non-circumvention traffic proxying
- TLS handshake includes bidirectional nonces
- May use existing web servers with actual users
Why HTTPS

Reasonable overhead. Per TLS Record:
- 5 bytes for header
- Encryption MAC (usually 16 bytes)
Proving knowledge of the secret

Include the secret in the URL of the initial HTTP request
What content to serve on the index page?

- Existing website
- An error page
  - Only 48.78% of websites scanned by censys respond with 200 OK[*]
- Copying Content
  - `wget` random website
- Restricted Access
- Proxy traffic to another website

Minimizing overhead

One way to get a tunnel with no overhead beyond TLS is to use a WebSockets reverse proxy — feature universally supported by the web servers.

HTTP client starts by sending the HTTP Request with ‘Upgrade: WebSocket’ header.

HTTP server responds with 101 Switching Protocol status code.
HTTP/1.1 101 Switching Protocols
Upgrade: WebSocket
Connection: Upgrade

hi server!

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade

hi client!
Performance Evaluation

Compare HTTPT to Shadowsocks proxy

- Does not use padding
- 0-RTT connection establishment (\(\rightarrow\) no forward secrecy)
Time To First Byte

- shadowsocks-libev 3.1.3
- HTTP(TLS 1.3)
- HTTP(TLS 1.2)
100-Mb file download time

shadowsocks-libev 3.1.3

HTTP (TLS 1.2)
Next Steps

- HTTP/2 support: take advantage of multiplexing
- TurboTunnel
- Optional Padding
Conclusion

HTTPS-based proxy

- Defends against active probing
- Does not need original website content to provide plausible responses to probes
- Performs comparable to lightweight proxies
FIN

Thanks for watching!