Detecting Probe-resistant Proxies

Sergey Frolov, Jack Wampler, Eric Wustrow
University of Colorado Boulder
Proxies

Censored User

Censor-Controlled Network

obfs3 proxy
Active Probing

Censored User

Censor-Controlled Network

obfs3 proxy

obfs3??
Let’s confirm!
Active Probing

Censored User

Censor-Controlled Network

*speaks obfs3*

obfs3 proxy
Active Probing

Censored User

Censor-Controlled Network

*speaks obfs3 back*
Active Probing

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

- Probe-Resistant proxies
  - Require knowledge of \textit{shared secret} to use
  - Don’t know secret? Server remains \textit{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

Censored User

Censor-Controlled Network

Not sure if I can block this

obfs4 proxy
Probing Probe-Resistant proxies

Are these proxies actually probe-resistant in practice?

- How **common** is the behavior of proxies to never respond to HTTP, TLS, ...any protocol?
  - If not common, censor can block it.
We need a source of TCP endpoints on the internet to compare their responses with Probe-Resistant proxies’ responses. We have 2 datasets:

**ZMap Dataset**
785k endpoints

**Tap Dataset**
433k endpoints
Probing Probe-Resistant proxies

We used the following probes:

1. HTTP
2. TLS ClientHello
3. Modbus
4. S7
5. Random bytes (23B - 17KB)
6. Empty probe
7. DNS zone Transfer
8. STUN
Probing Probe-Resistant proxies

For each probe we record 3-tuple result:

- Time to close
- Type of close (FIN, RST or TIMEOUT)
- Size of response data
  - Probe-resistant proxies never respond!
Endpoints that respond with data

<table>
<thead>
<tr>
<th>Probe</th>
<th>Tap dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS</td>
<td>87.8%</td>
</tr>
<tr>
<td>HTTP</td>
<td>64.6%</td>
</tr>
<tr>
<td>DNS-AXFR</td>
<td>58.8%</td>
</tr>
<tr>
<td>S7</td>
<td>56.9%</td>
</tr>
<tr>
<td>STUN</td>
<td>52.5%</td>
</tr>
<tr>
<td>Modbus</td>
<td>51.4%</td>
</tr>
<tr>
<td>Empty</td>
<td>8.4%</td>
</tr>
<tr>
<td>Any</td>
<td>94.0%</td>
</tr>
</tbody>
</table>

Response *alone* can distinguish 94% of endpoints in the realistic Tap dataset from proxies.
Endpoints that respond with data

<table>
<thead>
<tr>
<th>Probe</th>
<th>Tap dataset</th>
<th>ZMap dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLS</td>
<td>87.8%</td>
<td>0.90%</td>
</tr>
<tr>
<td>HTTP</td>
<td>64.6%</td>
<td>0.95%</td>
</tr>
<tr>
<td>DNS-AXFR</td>
<td>58.8%</td>
<td>0.67%</td>
</tr>
<tr>
<td>S7</td>
<td>56.9%</td>
<td>0.66%</td>
</tr>
<tr>
<td>STUN</td>
<td>52.5%</td>
<td>0.56%</td>
</tr>
<tr>
<td>Modbus</td>
<td>51.4%</td>
<td>0.54%</td>
</tr>
<tr>
<td>Empty</td>
<td>8.4%</td>
<td>0.23%</td>
</tr>
<tr>
<td>Any</td>
<td>94.0%</td>
<td>1.16%</td>
</tr>
</tbody>
</table>

Very few “legitimate” services (lots of firewalls/honeypots)
Probing Probe-Resistant proxies

How do our probe-resistant proxies respond to those probes? We examine:

- obfs4
- ObfuscatedSSH
- Lampshade
- MTProto Proxy
- Shadowsocks-Outline
- Shadowsocks-Python
# Probing ObfuscatedSSH

How else can we distinguish proxies from remaining 6%?

<table>
<thead>
<tr>
<th>Probe</th>
<th>Close Time (s)</th>
<th>Close Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modbus</td>
<td>30.237</td>
<td>FIN</td>
</tr>
<tr>
<td>S7</td>
<td>30.236</td>
<td>FIN</td>
</tr>
<tr>
<td>Random 23</td>
<td>30.238</td>
<td>FIN</td>
</tr>
<tr>
<td>Empty probe</td>
<td>30.238</td>
<td>FIN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Probe</th>
<th>Close Time (s)</th>
<th>Close Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP GET</td>
<td>0.250</td>
<td>RST</td>
</tr>
<tr>
<td>TLS ClientHello</td>
<td>0.240</td>
<td>RST</td>
</tr>
<tr>
<td>Random 25, 47, 51, 7KB, 17KB</td>
<td>0.237 - 0.251</td>
<td>RST</td>
</tr>
<tr>
<td>DNS AXFR</td>
<td>0.242</td>
<td>RST</td>
</tr>
<tr>
<td>STUN</td>
<td>0.236</td>
<td>RST</td>
</tr>
</tbody>
</table>
Proxy server code

clientConn := listener.Accept()
Proxy server code

clientConn := listener.Accept()
clientConn.SetDeadline(in30Seconds)
Proxy server code

clientConn := listener.Accept()
clientConn.SetDeadline(in30Seconds)
buffer := make([]byte, 50)
Proxy server code

clientConn := listener.Accept()
clientConn.SetDeadline(in30Seconds)

buffer := make([]byte, 50)
error := io.ReadFull(clientConn, buffer)
if error != nil { // didn’t get 50 bytes in 30s
    clientConn.Close()
    return
}
Proxy server code

clientConn := listener.Accept()
clientConn.SetDeadline(in30Seconds)
buffer := make([]byte, 50)
error := io.ReadFull(clientConn, buffer)
if error != nil { // didn’t get 50 bytes in 30s
    clientConn.Close()
    return
}

if !checkCredentials(buffer) {
    clientConn.Close()
    return
}

// do the proxying here
Can probe-resistant proxies be distinguished from other servers due to such thresholds?

<table>
<thead>
<tr>
<th>Probe Size</th>
<th>Response Size</th>
<th>Close Time</th>
<th>Close Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>49 bytes or fewer</td>
<td>0</td>
<td>30s</td>
<td>FIN</td>
</tr>
<tr>
<td>50 bytes</td>
<td>0</td>
<td>Right away</td>
<td>FIN</td>
</tr>
<tr>
<td>51 bytes or more</td>
<td>0</td>
<td>Right away</td>
<td>RST</td>
</tr>
</tbody>
</table>
Investigating Close Thresholds

- Built a threshold scanner to **binary search** for close thresholds
  - Send random data of different lengths
  - Scanned Tap/ZMap endpoints to compare with probe-resistant proxies
  - Check for “stability”
## Proxies’ thresholds

<table>
<thead>
<tr>
<th>Proxy</th>
<th>FIN Threshold</th>
<th>RST Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObfuscatedSSH</td>
<td>24 B</td>
<td>25 B</td>
</tr>
<tr>
<td>Shadowsocks-Python</td>
<td>50 B</td>
<td>-</td>
</tr>
<tr>
<td>Shadowsocks-Outline</td>
<td>50 B</td>
<td>51 B</td>
</tr>
<tr>
<td>Lampshade</td>
<td>256 B</td>
<td>257 B</td>
</tr>
<tr>
<td>obfs4</td>
<td>8 KB - 16 KB</td>
<td>next mod 1448</td>
</tr>
<tr>
<td>MTProto</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Investigating Close Thresholds

<table>
<thead>
<tr>
<th></th>
<th>Tap Dataset</th>
<th>ZMap Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td>Endpoints</td>
<td>433k</td>
<td>779k</td>
</tr>
<tr>
<td>“Stable” thresholds</td>
<td>144k (33.5%)</td>
<td>116k (15%)</td>
</tr>
</tbody>
</table>
### Investigating Close Thresholds

<table>
<thead>
<tr>
<th></th>
<th>Tap Dataset</th>
<th>ZMap Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Endpoints</strong></td>
<td>433k</td>
<td>779k</td>
</tr>
<tr>
<td><strong>“Stable” thresholds</strong></td>
<td>144k (33.5%)</td>
<td>116k (15%)</td>
</tr>
<tr>
<td><strong>Sent data response</strong></td>
<td>257k (59.5%)</td>
<td>5k (0.7%)</td>
</tr>
<tr>
<td><strong>Error</strong></td>
<td>3k (0.8%)</td>
<td>568k (73%)</td>
</tr>
<tr>
<td><strong>“Unstable” thresholds</strong></td>
<td>27k (6.2%)</td>
<td>88k (11.3%)</td>
</tr>
</tbody>
</table>

Why so few stable close thresholds?
Tap Endpoints’ Stable Thresholds

5, 11 and no threshold are the most common.
Decision Trees

We built manual decision trees to detect Probe-Resistant proxies based on their responses to our probes.

We also evaluated automatic decision trees, but they seemed less practical (see Appendix).
Manual ObfuscatedSSH decision tree

- **S7, Modbus, rand-23, empty**
  - **FIN and ≥ 30s**
    - **True**
    - **False**
      - **∀ others**
        - **RST and < 2s**
          - **True** → **OSSH**
          - **False** → **not OSSH**
      - **False** → **not OSSH**
Manual Lampshade Decision Tree

- **rand-7k, rand-17k**
  - RST and $< 2s$
    - True: **Lampshade**
    - False: **not Lampshade**
  - others
    - FIN and $\geq 90s$
      - True: **Lampshade**
      - False: **not Lampshade**
## Decision tree results

<table>
<thead>
<tr>
<th>Proxy</th>
<th>Decision Tree Labeled</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tap</td>
</tr>
<tr>
<td>Lampshade</td>
<td>0</td>
</tr>
<tr>
<td>ObfuscatedSSH</td>
<td>8</td>
</tr>
<tr>
<td>obfs4</td>
<td>2</td>
</tr>
<tr>
<td>Shadowsocks-Python</td>
<td>0</td>
</tr>
<tr>
<td>Shadowsocks-Outline</td>
<td>0</td>
</tr>
<tr>
<td>MTProto</td>
<td>3144</td>
</tr>
</tbody>
</table>
Manual MTProto decision tree

∀ probes

TIMEOUT

True

MTProto

False

not MTProto
Defense Strategies

- Recommended: never respond, never close connection
  - 0.56% of Tap dataset
- Randomizing parameters, such as timeout, on a per-server basis increases the overall size of “Anonymity Set” for your transport.
- Stable thresholds are a fingerprint
  - To fix don’t close immediately after handshake fails and keep draining the buffer until the timeout
Responsible Disclosure

We disclosed the presence of unique close thresholds to the devs, and as a result, it was removed from:

- OSSH on May 13, 2019
- obfs4 on June 21, 2019 (version 0.0.11)
- SS-Outline on September 4, 2019 (version 1.0.7)
- Lampshade on October 31, 2019

Timeouts still have to be chosen with care.
Probe-indifferent Server Timeouts (Tap)

But note: popular values might be limited to specific applications
Conclusions

● Probe-resistant proxies aren’t (or weren’t!)
  ○ Never responding with data is uncommon on the Internet
  ○ Connection timeouts and thresholds can be used to fingerprint server applications

● Notified proxy developers
  ○ Removed thresholds
  ○ But choosing timeouts still tricky

● Long-term: investigate alternative proxy protocols
  ○ e.g. Domain Fronting, Refraction, HTTPS-proxy
FIN

Thank you for attention!
Backup
Internet Censorship

Mean percentage of domains from Satellite input list blocked per country.
Source: https://censoredplanet.org/data/visualizations
● “How China Detects and Blocks Shadowsocks” describes evidence of a similar active probing attack occurring in China in 2019.
Removing Close Threshold

How to fix this behavior?

<table>
<thead>
<tr>
<th>Probe Size</th>
<th>Response Size</th>
<th>Close Time</th>
<th>Close Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>49 bytes or fewer</td>
<td>0</td>
<td>30 sec</td>
<td>FIN</td>
</tr>
<tr>
<td>50 bytes</td>
<td>0</td>
<td>Right away</td>
<td>FIN</td>
</tr>
<tr>
<td>51 bytes or more</td>
<td>0</td>
<td>Right away</td>
<td>RST</td>
</tr>
</tbody>
</table>
Removing Close Threshold

clientConn := listener.Accept()

clientConn.SetDeadline(in30Seconds)

buffer := make([]byte, 50)
error := io.ReadFull(clientConn, buffer)
if error != nil { // didn’t get 50 bytes in 30s
    clientConn.Close()
    return
}

if !checkCredentials(buffer) {
    clientConn.Close()
    return
}

// do the proxying here
Removing Close Threshold

clientConn := listener.Accept()
clientConn.SetDeadline(in30Seconds)

buffer := make([]byte, 50)
error := io.ReadFull(clientConn, buffer)
if error != nil { // didn’t get 50 bytes in 30s
    clientConn.Close()
    return
}

if !checkCredentials(buffer) {
    io.Copy(ioutil.Discard, clientConn)
    clientConn.Close()
    return
}
## Removing Close Threshold

<table>
<thead>
<tr>
<th>Probe Size</th>
<th>Response Size</th>
<th>Close Time</th>
<th>Close Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>49 bytes or fewer</td>
<td>0</td>
<td>30 sec</td>
<td>FIN</td>
</tr>
<tr>
<td>50 bytes</td>
<td>0</td>
<td>30 sec</td>
<td>FIN</td>
</tr>
<tr>
<td>51 bytes or more</td>
<td>0</td>
<td>30 sec</td>
<td>FIN</td>
</tr>
</tbody>
</table>