Internet Censorship:

Great Firewall of China (GFC)

Jong C. Park
joumon@gmail.com
Contents

• Internet Censorship
• Censoring Mechanisms
• How to circumvent
• Censorship Researches
• Our Work @ UNM
Information throttled

- Censorship observed
  - Slashdot
  - Open Net Initiative
  - Reporters Without Borders
  - Human Right Watch
  - UN, etc........
- Censorship in burst
  - 3 out of 41 in ‘02 -> more than 25 in ‘09
  - can view at ONI for more details
  - political, social, security/conflict, Internet tools
Generic Filtering
Maximillian Dornseif, Germany,

from “Government mandated blocking of foreign Web content”

• Packet-level filtering
  • mostly done by firewall (rules), but need up-to-date

• OSI layer 3 filtering
  • inspects header and forwards/drops
  • coarse-grained due to high false positive (overblocking)

• OSI layer 4 filtering: finer-grained than layer 3
Generic Filtering (cont’d)

- Application-level filtering
  - inspects payload and performs most detailed
  - often provides ways of informing users about the filtering
  - hard to be done in real-time due to overhead
  - encryption/compression makes infeasible
Generic Filtering (cont’d)

- Filtering mechanisms
  - proxy firewall (HW requirement, latency)
  - IDS (signature-based and/or heuristic)
  - DNS poisoning
  - help from search engines
  - web-site delisting
  - keyword filtering
  - keyword forgery
Circumvent Censorship

- Discard TCP RST
- Encrypt/Compress
- Use proxy server (web cache)
- Use publicly accessible DNS server (caveat: security)
- Tunneling such as ICMP, SSH, SSL, or VPN
- Add comment in HTML
- Use captchas
- etc
Great Firewall of China

- Why China?
  - Most complicated censorship mechanism
  - China filters to maintain power
  - Cyber-sensors and cyber-police
  - Hierarchical supervisory bodies
  - World’s largest IPv6 backbone
GFC (cont’d)

- A large number of sites block
- Not all about pornography sites
- PA state, USA, blocks child pornography
- Most advanced keyword filtering
- falun gong, human right, Taiwan, Tibet, even foreign cities due to similar pronunciations
- TCP RSTs, specious SYN/ACK
Zittrain & Edelman

- First studied on blocked web sites in China

  *from “Internet filtering in China”, IEEE Internet Computing*

- Studied filtering methods
  - IP address blacklisting
  - DNS IP address blocking
  - DNS redirection
  - URL keyword filtering
  - HTML response keyword filtering
Ignore the GFC

• Seminal work by Richard Clayton et al.
  
  • *from “Ignoring the Great Firewall of China”, 6th workshop on Privacy Enhancing Technologies*

• Filtered at borders

• Triggers 3 consecutive TCP RSTs (seq, seq+1460, seq+4380)

• Stateless inspection
ConceptDoppler

- More work done by Jed Crandall et al.
  - from “ConceptDoppler: a weather tracker for Internet censorship”, ACM CCS, 2007
- Showed GFC as panopticon
- used tcptraceroute with increasing TTL
- Also showed 28.3% requests survived to reach server
- Most filtering happens at 1st hop, but deep thru 13th hops beyond borders
- Provides blacklist words using LSA
TCP RST patterns

- Work done by N. Weaver et al.
  - from “Detecting forged TCP RST Packets”, TRUST, 2008

- Studied on forged TCP RST patterns and fingerprints

- Patterns:
  - D+R, R+D, R+R, S+R, SA+R

- Fingerprints:
  - China has IPID 64, IPID -26, SEQ 1460, etc
Electronic Big Brother

• Work done by J. Karlin et al.
  • from “Nation-state routing: censorship, wiretapping, and BGP”

• Countries with most information flow: USA, England, and Germany
  • China has little effect on interdomain routing

• Power Law can be applied
Our Work being going...

- J. Crandall et al. showed, with a blacklist, that GFCs blocks HTML GET requests substantially.
- Are GFCs symmetric or asymmetric in HTML filtering?
  - If asymmetric, filtering would be less effective.
  - How much blocking on HTML response?
  - What blacklist keywords blocked & why?
  - How do GFCs block & how to escape?
Constraints

• Internet Measurement
  • No fine-grained protocol support
  • Privacy and legal issue
  • and more…

• China claims world’s largest IPv6 backbone
  • Huge latency due to 4over6
  • Little collaboration from IPv6
Keyword Probe

GET server/keyword=*falun* HTTP/1.1

Doesn’t know which direction is blocked…
Number Probe

- Use a proxy server, a.k.a. a web cache
- Use number query instead of keyword
  - Chinese cannot be guaranteed
  - Neither can English since some abbreviations are blocked
- Bogus response page of ~4KB
- Hash will be much better, but file used currently
  - Latency generated due to table iteration
Number Probe

GET server/keyword=1 HTTP/1.1

1 means falun

 Doesn’t matter which direction is blocked…
Setup

- 7 pairs of single server and single client
- Scripting probes with `wget` command + Python code on server
  - `WGETRC`, `.wgetrc`, or `/etc/wgetrc`
  - Scapy used
Setup

- Python code with libpcap record traffics on both server-side & client-side
  - tcptraceroute recording latency every 10 mins
- 3 contiguous probes sent
- Preliminary tests from late Feb to early Mar
- Real experiments from spring recess thru now
FSM for Probe

- **START**
  - 200 OK RCV'D
- **WAIT**
  - TCP RST RCV'D
  - + QUERY SENT(+port/2)
  - + QUERY SENT(-port/2)
  - + QUERY SENT(+256)
  - + QUERY SENT(-256)
- **CLEAR**
  - Exponential backup
  - + HELLO SENT(+1)
  - 200 OK RCV'D

**QUERY SENT**
Proxy Hunt

- Chinese broadcast a list of proxy servers
  - Financial issue
  - Falun movement
- Geographically diverse web caches
- Use Visualroute server 2008 trial
  - Returns *guess locations* from ISP info
- Proxy life time
  - Last up to two weeks or so
  - Some lasts a couple of days
What to test?

- 12/20 diverse locations randomly chosen
  - Most along with east coast (1 partial)
  - 1 inland location included, but partial
  - 1/2 hrs to 2 days each run
- Different locations of a single keyword
  - Beginning/Middle/End
- Different response sizes
  - 600B/4KB/40KB/175KB/350KB
- Static vs dynamic pages
What to test?

- Keyword threshold (*to be tested*)
- How many number of the same/different kind triggers & how?
- Different HTTP protocols (*to be tested*)
- Probably no effect since proxy uses HTTP 1.1
- No STDDEV data available yet (*being tested*)
- TCP RSTs trend: how many & how (*to be tested*)
- Any other ideas?
Map
IPv6 Backbone
TCP RST

- Depending on location & probe, the number of TCP RSTs received vary
  - Sometimes just 1 RST
  - Sometimes more than 10 RSTs
  - No automatism at this time (*future work*)
Odd Responses

- Unlike HTML GET request, HTML response receives odd packets
- 404 Page Not Found (once)
- 502 Error (58 times)
  - Proxy error/Bad gateway
- 503 Service Unavailable (236 times)
- Connection Timed Out (not counted yet): discarded
Odd Responses

- Two kinds
  - Just 502/503 error received without any preceding/following RSTs
  - Some HELLOs get TCP RSTs (delay)
  - RST received first and then 502/503 error received: categorized same as TCP RST

- Who sends these? Let’s look at client-side!
200 OK

When a non-blocked keyword is requested

UNM -> China: [SYN] seq=0, TTL=64
China -> UNM: [SYN+ACK] seq=0, ack=1, TTL=48
UNM -> China: [ACK] seq=1, ack=1, TTL=64
UNM -> China: GET server/search.php?keynum=0
   HTTP/1.0, TTL=64 * 0 means hello
China -> UNM: [ACK] seq=1, ack=138, TTL=48
China -> UNM: [TCP Dup ACK] [ACK] seq=1, ack=138, TTL=48
China -> UNM: [TCP segment of a reassembled PDU] TTL=48
China -> UNM: [TCP segment of a reassembled PDU] TTL=48
China -> UNM: [TCP previous segment lost] [FIN+ACK]
   seq=4262, ack=138, TTL=48
UNM -> China: [ACK] seq=138, ack=2897, TTL=64
UNM -> China: [TCP Dup ACK] [ACK] seq=138, ack=2897
China -> UNM: [TCP out-of-order] HTTP/1.0 200 OK
   (text/html)
TCP RST

When a TCP RST packet is received

UNM → China: [SYN] seq=0, TTL=64
China → UNM: [SYN+ACK] seq=0, ack=1, TTL=48
UNM → China: [ACK] seq=1, ack=1, TTL=64
UNM → China: GET server/search.php?keynum=0
    HTTP/1.0, TTL=64 * 0 means falun
China → UNM: [ACK] seq=1, ack=152, TTL=48
China → UNM: [RST] seq=1, TTL=50
UNM → China: [FIN+ACK] seq=1, ack=1, TTL=64
UNM → China: [FIN+ACK] seq=134, ack=2954, TTL=64
UNM → China: [FIN+ACK] seq=134, ack=2954, TTL=64
UNM → China: [FIN+ACK] seq=134, ack=2954, TTL=64
When a 503 error message is received

UNM -> China: [SYN] seq=0, TTL=64
China -> UNM: [SYN+ACK] seq=0, ack=1, TTL=48
UNM -> China: [ACK] seq=1, ack=1, TTL=64
UNM -> China: GET server/search.php?keynum=0
HTTP/1.0, TTL=64 * 0 means falun
China -> UNM: [ACK] seq=1, ack=134, TTL=48
China -> UNM: [TCP previous segment lost] [FIN+ACK]
    seq=1578, ack=134, TTL=48
UNM -> China: [TCP Dup ACK] [ACK] seq=134, ack=1, TTL=64
China -> UNM: [TCP retransmission] [TCP segment of
    a reassembled PDU] , TTL=48
China -> UNM: [TCP retransmission] HTTP/1.0 503 Service
    Unavailable, TTL=48
502,503 Error

- Based on client-side, GFCs seem to intervene?
  - Probably yes or no depending on preceding/following RSTs
- Let’s look at server-side!
502, 503 Error

client

proxy

server

GET

GET

GET

GET

OK

OK

OK

OK

502/503

GET
502,503 Error

- Probes right before the current probe and right after the current were successful
  - $i$-th keyword (3rd probe): OK
  - $(i+1)$-th keyword (1st probe): 502 Error
  - $(i+1)$-th keyword (2nd probe): OK
502,503 Error

- What caused this kind of 502/503 error?
  - Probably proxy misconfigurations
  - More analysis needed
  - Some HELLOs get delayed RSTs
- 503 same as 502?
  - No concrete evidence
  - Still in analysis
Odd Packets

- Odd packets are generated by a web cache
- Mistakenly thought as TCP RST packets received from GFC on the way from server to a web cache
- Then, self-censorship? Definitely no!
- Probably due to proxy misconfigurations
- Odd packets can also be considered TCP RST? Some are definitely false positive, though.
- Why differs? Can’t explain at this time
Self-censorship?

*When a self-censorship is observed at Beijing*

UNM -> China: [SYN] seq=0, TTL=64
China -> UNM: [SYN+ACK] seq=0, ack=1, TTL=46
UNM -> China: [ACK] seq=1, ack=1, TTL=64
UNM -> China: GET server/search.php?keynum=0 HTTP/1.0, TTL=64 *0 means Falun*
China -> UNM: [ACK] seq=1, ack=152, TTL=46
China -> UNM: [RST] seq=1, TTL=46
UNM -> China: [FIN+ACK] seq=1, ack=1, TTL=64
UNM -> China: [FIN+ACK] seq=134, ack=2954, TTL=64
UNM -> China: [FIN+ACK] seq=134, ack=2954, TTL=64
UNM -> China: [FIN+ACK] seq=134, ack=2954, TTL=64

Proxy ACK’ed server in which TCP RST is not observed at all.
Probing Results

- Odd probing results, but the following should be taken into account:
  - Route flutter
  - HTML request blocking trend varies depending on probing time (diurnal pattern)
  - Might be the most efficient way because of distributed filtering on response?
  - Can’t explain many phenomenon, but can make good guesses at least
#Block on 4KB

#keywords of ~4KB

![Bar chart showing distribution of keywords](chart.png)
#Block on 4KB

Zoom-up of #keywords of ~4KB
#Block on 600B

#keywords of ~600B
#Block on 40KB

#keywords of ~40KB

![Bar Chart]

- #1
- #2
- #4
- #5
- #6
- #7
- #8
- #9
- #10
- #11
- #12

Value:
- #1
- #2
- #4
- #5
- #6
- #7
- #8
- #9
- #10
- #11
- #12
#Block on 175KB

#keywords of ~175KB

![Graph showing the distribution of keywords across different categories.](Image)
#Block on 350KB

#keywords of ~350KB

![Bar chart showing the number of keywords for each block, with Block #11 having the highest number of keywords.](chart.png)
#Block on different size

#keywords by size

![Graph showing the distribution of keywords by size with different block sizes.](image-url)
#Block on different loc

Different Location

![Graph showing different location blocks with various markers and lines for #1 to #11 across different stages s0 to s100.](image)
#Block on different loc

Zoom-up of Different Location

- #1
- #2
- #4
- #5
- #6
- #7
- #8
- #9
- #10
- #11

s0 s1 s10 s50 s100
Nation-wide
STD DEV

STDDEV-AVG of #keywords

#1 #2 #4 #5 #6

#7 #8 #9 #10 #11
STD DEV

Zoom-up of STDDEV-AVG of #keywords by location
Beijing on 4KB

#keywords by date
at Beijing (#9) with keyword at begin

4/6/2008: 6
4/8/2008: 17
False Positive

- Hard to predict, huh?
  - Need more time/tests
- Assume all odd packets false positive
  - Let’s visit all graphs again!
- False positive rate of ~8.5%
  - 2236 times not rec’d in total since Mar
  - 2045 RSTs observed
STD DEV

#keywords of STDDEV at Beijing (#9) with keyword at beginning on Apr 25-26

RST+Others

RST only
#keywords of STDDEV at Beijing (#9)
with keyword at beginning on Apr 26-28
Beijing on 4KB

#keywords by date
at Beijing (#9) with keyword at begin

4/6/2008: 6
4/8/2008: 17
STD DEV

Zoom-up of STDDEV-AVG of #keywords by location
By Location

- #keywords at Jiangsu (#5)
- #keywords at Shanxi (#7)
- #keywords at Shandong (#6)
- #keywords at Tianjin (#8)
By Location

#keywords at Beijing (#9)

#keywords at Harbin (#11)

#keywords at Liaoning (#10)

#keywords at Chengdu (#12)
Dynamic Page (Beijing)

#keywords by dynamic
at Jiangxi (#2)
#keywords by static

Begin
Middle
End
Dynamic vs Static: beginning

#keywords by pages

- s1
- s10
- s50
- s100

- static
- dynamic
Dynamic vs Static: middle

#keywords by pages at Jiangxi (#2)
Dynamic vs Static: end

#keywords by pages
at Jiangxi (#2)
Blacklist updated

#keywords by blacklist

```
<table>
<thead>
<tr>
<th></th>
<th>s0</th>
<th>s1</th>
<th>s10</th>
<th>s50</th>
<th>s100</th>
</tr>
</thead>
<tbody>
<tr>
<td>old</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>new</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Filtering Pattern: probabilistic?

filtering pattern
(91 out of 2045)
Blacklist

- ~95% blacklist keywords of HTML GET request also blocked in HTML response
- ~54% blocked when excluding Harbin
- Less substantial blocking on HTML response
- Probably due to overhead and/or latency
- Unlike request, response blacklist seems to be distributed over the nation
- Some GFCs seem to share
Blacklist (4KB)
Blacklist (600B)

1, 4, 6, 8, 9, 10

5

18 29 30 358

11

more

2, 12 = ∅
3: N/A
Blacklist (all)

- 104, 107, 109, 135, 161, 163, 204, 205, 210, 213, 230, 232, 236, 238, 249, 256, 263, 268, 285, 357, 370
- 40, 55, 56, 73, 74, 79, 84
- 118, 127, 131, 136, 162, 177, 178, 182, 185, 191, 197, 221, 259, 290, 293, 323, 346, 48, 50, 66
- 317, 319, 320, 330, 353

12 = ∅
Blacklist Size

<table>
<thead>
<tr>
<th>Blacklist Size</th>
<th>#keywords blocked in all sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td>300</td>
<td>0</td>
</tr>
<tr>
<td>350</td>
<td>50</td>
</tr>
<tr>
<td>400</td>
<td>100</td>
</tr>
</tbody>
</table>

#keywords blocked
in all sizes

![Bar chart showing the number of keywords blocked across different blacklist sizes](image)
Blacklist Size

Zoom-up of #keywords blocked by location

<table>
<thead>
<tr>
<th>#1</th>
<th>#2</th>
<th>#3</th>
<th>#4</th>
<th>#5</th>
<th>#6</th>
<th>#7</th>
<th>#8</th>
<th>#9</th>
<th>#10</th>
<th>#11</th>
<th>#12</th>
</tr>
</thead>
</table>
Conclusion

- GFC performs symmetric filtering against HTML
  - Less substantial on HTML response probably due to overhead/latency
  - Distributed blacklist words
  - Share a subset of blacklist words
  - Route path, packet size, and keyword location affect censorship efficiency
  - Stateful vs. stateless TCP RSTs
  - Inversion suggests application-layer censorship???
  - Still can’t explain many phenomenon
Thank you

• Any questions or feedbacks?
Lab 4

- We’re not using an actual IDS, but by iptables.
- the server is too simple and is just buffering up to 4 KB
- can beat this by sending a sequence of packets which split keywords
  - need to flush first before sending the next
  - otherwise, TCP RSTs will be triggered
- Lab 4 1/2 will be about TCP fragmentation(?)
import socket  # client

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect(("10.0.0.3", 8080))
s.connect(("10.0.0.3", 8080))
s.send("hello")

d=s.recv(1024)
print repr(d)
s.close()
Scapy

- TCP fragmentation?
  - Not a proper jargon! But, TCP supports byte streams and out-of-order delivery.
    - Let’s call it for the sake of simplicity.
  - Why not IP fragmentation? RFC1858!
    - Tiny fragment attack
      - can use fragroute for smaller IP packets
    - can forge packets with Scapy
Scapy (cont’d)

- Use whatever you like, for example MuxTCP, C code with libpcap, etc. (Visit Milw0rm for code.)

- Scapy is troublesome in initiating TCP
  - need to change firewall rules by iptables
    - iptables -A OUTPUT -p tcp --tcp-flags RST RST -j DROP
  - wouldn’t work for Lab 4 1/2 since the server is not under control
    - initiate TCP handshake with your own socket
Scapy (cont’d)

- To use tcpdump, you need to set PATH
  - use scapy.sh script mailed
  - use sniff.py to sniff the network
- Need to know some TCP/IP stuff
  - IP
    - IPID, IP addresses
  - TCP
    - flags, sequence #, ack #, ports
TCP Handshake

- `fd = open("message.txt", 'r')`
- `lines = fd.readlines()`
- `fd.close()`

- `for l in lines:
  print l.strip() + \n`

- `import time`
- `time.sleep(1)`