Automating Censorship Evasion

Kevin Bock
In-network censorship by nation-states
In-network censorship by nation-states
In-network censorship by nation-states
In-network censorship by nation-states
In-network censorship by nation-states
In-network censorship by nation-states

Spoofed tear-down packets
In-network censorship by nation-states

Spoofed tear-down packets
In-network censorship by nation-states

Spoofed tear-down packets

Client

The server terminated

Server

The client terminated

Requires *per-flow state*

Censors fighting *end to end principle*
In-network censorship by nation-states

Requires *per-flow state*

Censors fighting *end to end principle*

Evasion can take advantage of these shortcuts
In-network censorship by nation-states

Requires *per-flow state*

Censors fighting *end to end principle*

Evasion can take advantage of these shortcuts
In-network censorship by nation-states

The client terminated

TTL=0

Requires *per-flow state*

Censors fighting *end to end principle*

Evasion can take advantage of these shortcuts
Censorship evasion research

1. **Understand** how censors operate

2. **Apply insight** to create evasion strategies

Largely **manual** efforts give censors the advantage

Our work gives evasion the advantage
Automated censorship evasion research

1. Automate the discovery of new evasion strategies
2. Use the strategies to understand how the censor works
1. Automate the discovery of new evasion strategies

2. Use the strategies to understand how the censor works

Geneva
Genetic Evasion
Geneva
Genetic Evasion

Building Blocks
A
T
C
G

Composition

Mutation

Fitness
Geneva runs strictly at one side.

Manipulates packets to and from the client.
**Geneva**

**Genetic Evasion**

**Building Blocks**

**Manipulates packets** to and from the client

- **Bit manipulation**
  - Versatile but inefficient

- **Known strategies**
  - Efficient but limited
Geneva
Genetic Evasion

Building Blocks

Manipulates packets to and from the client

- Duplicate
- Tamper
- Fragment
- Drop

Fragment (IP) or Segment (TCP)

Alter or corrupt any TCP/IP header field

No semantic understanding of what the fields mean
Geneva
Genetic Evasion

Building Blocks
Actions manipulate individual packets
- Duplicate
- Tamper
- Fragment
- Drop

Composition

Mutation

Fitness

Actions manipulate individual packets
Composition
Mutation
Fitness
Geneva
Genetic Evasion

Building Blocks
Actions manipulate individual packets
- Duplicate
- Tamper
- Fragment
- Drop

Composition

Mutation

Fitness

Geneva
Genetic Evasion

Composition
out:tcp.flags=A

Match
exact

Action
in-order

Duplicate

Tamper
tcp.flags = R

Tamper
ip.ttl = 2
Running a Strategy

Composition

Client

Duplicate

out:tcp.flags=A

Tamper

tcp.flags = R

Tamper

ip.ttl = 2

Server
Running a Strategy

Composition

Client

Duplicate

Tamper
tcp.flags = R

Tamper
ip.ttl = 2

out:tcp.flags=A

Server
Running a Strategy
Running a Strategy

Composition

Client

Duplicate

Tamper
tcp.flags = R

Tamper
ip.ttl = 2

TTL=8

TTL=2

Server
Running a Strategy

Composition

Client ➔ Duplicate ➔ Tamper (tcp.flags = R) ➔ Tamper (ip.ttl = 2) ➔ TTL=2 ➔ Server
Running a Strategy

Client

Duplicate

Tamper (TCP flags = R)
Tamper (ip.ttl = 2)

Server
Geneva
Genetic Evasion

Building Blocks
Actions manipulate individual packets
- Duplicate
- Tamper
- Fragment
- Drop

Composition
Actions compose to form trees
- out:tcp.flags=A
  - Duplicate
    - Tamper tcp.flags = R
    - Tamper ip.ttl = 2

Mutation
Fitness
Geneva
Genetic Evasion

**Building Blocks**
Actions manipulate individual packets
- Duplicate
- Tamper
- Fragment
- Drop

**Composition**
Actions compose to form trees
- out:tcp.flags=A
- $\text{Duplicate}$
- $\text{Tamper}$
  - tcp.flags = R
- $\text{Tamper}$
  - ip.ttl = 2

**Mutation**

**Fitness**
Geneva
Genetic Evasion

Building Blocks
Actions manipulate individual packets

- Duplicate
- Tamper
- Fragment
- Drop

Composition
Actions compose to form trees

- out:tcp.flags=A
- Duplicate
- Tamper tcp.flags = R
- Tamper ip.ttl = 2

Mutation
Randomly alter types, values, and trees

Fitness

Diagram showing the process of building, composing, and mutating packets.
Geneva
Genetic Evasion

Building Blocks
Actions manipulate individual packets

- Duplicate
- Tamper
- Fragment
- Drop

Composition
Actions compose to form trees

- out:tcp.flags = A
- Duplicate
- Tamper tcp.flags = R
- Tamper ip.ttl = 2

Mutation
Randomly alter types, values, and trees

Fitness

Geneva
Genetic Evasion

Fitness
Which individuals should survive to the next generation?
Which individuals should survive to the next generation?
Fitness

Which **individuals** should survive to the next **generation**?

- Not triggering on any packets
- Breaking the TCP connection
- Successfully obtaining forbidden content
- Conciseness
Geneva
Genetic Evasion

Building Blocks
Actions manipulate individual packets
- Duplicate
- Tamper
- Fragment
- Drop

Composition
Actions compose to form trees
- out:tcp.flags=A
  - Duplicate
  - Tamper tcp.flags = R
  - Tamper ip.ttl = 2

Mutation
Randomly alter types, values, and trees

Fitness
Goal: Fewest actions needed to succeed
- No trigger
- Break TCP
- Successful
- Concise
Client-side results

In-lab experiments
Against mock censors

In-lab experiments
Against mock censors

Failed to find the strategies we did not give building blocks for

<table>
<thead>
<tr>
<th>Species</th>
<th>Strategy</th>
<th>Found?</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>[21][33][41]</td>
</tr>
<tr>
<td>TCB Creation</td>
<td>w/ low TTL</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>w/ corrupt checksum</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>( Improved ) and Resync/Desync</td>
<td>✓</td>
</tr>
<tr>
<td>TCB Teardown</td>
<td>w/ RST and low TTL</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>w/ RST and corrupt checksum</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>w/ RST and invalid timestamp</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>w/ RST and invalid MD5 Header</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>w/ RST/ACK and corrupt checksum</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>w/ RST/ACK and low TTL</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>w/ RST/ACK and invalid timestamp</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>w/ RST/ACK and invalid MD5 Header</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>w/ FIN and low TTL</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>w/ FIN and corrupt checksum</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>( Improved ) and TCB Reversal</td>
<td>✓</td>
</tr>
<tr>
<td>Reassembly</td>
<td>TCP Segmentation reassembly out of order data</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Overlapping segments</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>In-order data w/ low TTL</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>In-order data w/ corrupt ACK</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>In-order data w/ corrupt checksum</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>In-order data w/ no TCP flags</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Out-of-order data w/ IP fragments</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Out-of-order data w/ TCP segments</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>( Improved ) In-order data overlapping</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Payload splitting</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Payload reordering</td>
<td>✓</td>
</tr>
<tr>
<td>Traffic Misclassification</td>
<td>Inert Packet Insertion w/ corrupt checksum</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Inert Packet Insertion w/o ACK flag</td>
<td>✓</td>
</tr>
<tr>
<td>State Exhaustion</td>
<td>Send &gt; 1KB of traffic</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Classification Flushing – Delay</td>
<td>✓</td>
</tr>
<tr>
<td>HTTP Incompleteness</td>
<td>GET w/ &gt; 1 space between method and URI</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>GET w/ keyword at location &gt; 2048</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>GET w/ keyword in 2nd or higher of multiple requests in one segment</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>GET w/ URL encoded (except %-encoding)</td>
<td>✓</td>
</tr>
</tbody>
</table>
Client-side results – Real censor experiments

20+ Species — The underlying bug
30+ Sub-species — How Geneva exploits it
80+ Variants — Functionally distinct

45+ China
15+ India
13+ Iran
24+ Kazakhstan
Teardown species

During the TCP handshake, insert a TTL-limited RST
During the TCP handshake, insert a TTL-limited RST

Segment the request
During the TCP handshake, insert a TTL-limited RST

Segment the request

GET /?search=ultrasurf
During the TCP handshake, insert a TTL-limited RST

Segment the request, but *not the keyword*
During the TCP handshake, insert a TTL-limited RST

Segment the request, but *not the keyword*
During the TCP handshake, insert a TTL-limited RST

Segment the request, but *not the keyword*
Censorship evasion has always involved the client

Censoring regime

Client

Software

Poses risks to users

Cannot help those who do not know they are censored
Server-side evasion

Censoring regime

Client

Software

Server
Server-side evasion

Censoring regime

Client

Server

Software
Server-side evasion

Censoring regime

Potentially broadens reachability without any client-side deployment
Server-side evasion “shouldn’t” work
Server-side evasion “shouldn’t” work

Censored keyword
Server-side evasion “shouldn’t” work

Client

SYN
SYN/ACK
ACK
PSH/ACK (query)

Server

ACK
PSH/ACK (response)

All a server does before client is censored

Censored keyword
A successful server-side evasion strategy
A successful server-side evasion strategy
A successful server-side evasion strategy

TCP simultaneous open

Client sends a SYN/ACK

Client

Server

SYN

SYN

SYN

(payload)

SYN/ACK

ACK

ACK

ACK

PSH/ACK

(query)

PSH/ACK

(response)
A successful server-side evasion strategy

TCP simultaneous open

Censor de-synchronizes

Client sends a SYN/ACK

Client

Server
A successful server-side evasion strategy

Success rates:
- DNS: 89%
- FTP: 36%
- HTTP: 54%
- HTTPS: 55%
- SMTP: 70%
Server-side evasion strategies

- **China**: 8 strategies
- **Iran/India**: 1 strategy
- **Kazakhstan**: 3 strategies

None of these require any client-side deployment.
Server-side evasion results

NULL TCP Flags

Success rates

HTTP 100%
Server-side evasion results

**NULL TCP Flags**

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>SYN</td>
<td>ø</td>
</tr>
<tr>
<td>(no flags)</td>
<td></td>
</tr>
<tr>
<td>SYN/ACK</td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td></td>
</tr>
<tr>
<td>PSH/ACK (query)</td>
<td></td>
</tr>
<tr>
<td>ACK</td>
<td></td>
</tr>
<tr>
<td>PSH/ACK (response)</td>
<td></td>
</tr>
</tbody>
</table>

Success rates:

HTTP 100%
Server-side evasion results

NULL TCP Flags

Client  
SYN  
SYN/ACK  
ACK  
PSH/ACK (query)  
PSH/ACK (response)  

Server  
\(\emptyset\) (no flags)  
SYN/ACK  
ACK  

Server sends a packet with no TCP flags set

Success rates

HTTP 100%
Server-side evasion results

NULL TCP Flags

Censor can’t handle unexpected flags

Server sends a packet with no TCP flags set

Success rates

HTTP 100%
Server-side evasion results

Double benign-GETs

Client   Server

SYN
SYN/ACK (benign GET)
SYN/ACK (benign GET)
ACK
ACK
ACK
PSH/ACK (query)
PSH/ACK (response)
Server-side evasion results

Double benign-GETs

Server sends uncensored GETs inside two SYN/ACKs
Server-side evasion results

Double benign-GETs

Client

SYN
SYN/ACK (benign GET)
SYN/ACK (benign GET)
ACK
ACK
PSH/ACK (query)
PSH/ACK (response)

Server

Server sends uncensored GETs inside two SYN/ACKs

Success rates

HTTP 100%

Censor confuses connection direction
Server-side evasion results

Double benign-GETs

Client  Server

- SYN
- SYN/ACK (benign GET)
- SYN/ACK (benign GET)
- ACK
- ACK
- PSH/ACK (query)
- ACK
- PSH/ACK (response)

Success rates
HTTP 100%
Server-side evasion strategies

China
8 strategies

Iran/India
1 strategy

Kazakhstan
3 strategies

None of these require any client-side deployment
Injects TCP RSTs

Injects & blackholes

Injects & blackholes

Injects a block page
Come as you are

Windows XP
Windows 7
Windows 8.1
Windows 10
Server 2003
Server 2008
Server 2013
Server 2018

OS X 10.14
OS X 10.15

iOS 13.3

Android 10

Centos 6
Centos 7

Ubuntu 12.04
Ubuntu 14.04
Ubuntu 16.04
Ubuntu 18.04
What's next?

**New insight into how censors work**
- Success rate changes by protocol
- “Multi-box theory”

**Rapid response to new censorship**
- Iran’s new protocol filter (Feb 2020)
- China’s new ESNI filter (July 2020)
New insights into how censors work

All of the server-side strategies operate strictly during the TCP 3-way handshake

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Success Rates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DNS</td>
</tr>
<tr>
<td><strong>China</strong></td>
<td></td>
</tr>
<tr>
<td>- No evasion</td>
<td>2%</td>
</tr>
<tr>
<td>1 Sim. Open, Injected RST</td>
<td>89%</td>
</tr>
<tr>
<td>2 Sim. Open, Injected Load</td>
<td>83%</td>
</tr>
<tr>
<td>3 Corrupt ACK, Sim. Open</td>
<td>26%</td>
</tr>
<tr>
<td>4 Corrupt ACK Alone</td>
<td>7%</td>
</tr>
<tr>
<td>5 Corrupt ACK, Injected Load</td>
<td>15%</td>
</tr>
<tr>
<td>6 Injected Load, Induced RST</td>
<td>82%</td>
</tr>
<tr>
<td>7 Injected RST, Induced RST</td>
<td>83%</td>
</tr>
<tr>
<td>8 TCP Window Reduction</td>
<td>3%</td>
</tr>
<tr>
<td><strong>India</strong></td>
<td></td>
</tr>
<tr>
<td>- No evasion</td>
<td>100%</td>
</tr>
<tr>
<td>8 TCP Window Reduction</td>
<td>–</td>
</tr>
<tr>
<td><strong>Iran</strong></td>
<td></td>
</tr>
<tr>
<td>- No evasion</td>
<td>100%</td>
</tr>
<tr>
<td>8 TCP Window Reduction</td>
<td>–</td>
</tr>
<tr>
<td><strong>Kazakhstan</strong></td>
<td></td>
</tr>
<tr>
<td>- No evasion</td>
<td>100%</td>
</tr>
<tr>
<td>8 TCP Window Reduction</td>
<td>–</td>
</tr>
<tr>
<td>9 Triple Load</td>
<td>–</td>
</tr>
<tr>
<td>10 Double GET</td>
<td>–</td>
</tr>
<tr>
<td>11 Null Flags</td>
<td>–</td>
</tr>
</tbody>
</table>

So why are different applications affected differently in China?
New Model for Chinese Censorship

Results suggest GFW is running multiple censoring middleboxes in parallel.
Multi-box theory
How does the censor know which one to apply to a connection?
Multi-box theory

GFW

DNS HTTP

FTP HTTPS

Client

Not port number
Censors effectively on any port

Server
Multi-box theory

GFW

DNS HTTP

FTP HTTPS

Client

Server

Not port number
Censors effectively on any port
Multi-box theory

Applies protocol fingerprinting
Multi-box theory

GFW

DNS HTTP

Mine! Forbidden

FTP HTTPS

Not mine

Client

Server

Applies protocol fingerprinting
Where are these middleboxes?

Used TTL-limited probes

Co-located at the network level
1. Automate the discovery of new evasion strategies

2. Use the strategies to understand how the censor works
What’s next?

New insight into how censors work

- Success rate changes by protocol
- “Multi-box theory”

Rapid response to new censorship

- Iran’s new protocol filter (Feb 2020)
- China’s new ESNI filter (July 2020)
Responsive to new censorship events

February 2020: Iran launched a new system: a protocol filter
Responsive to new censorship events

February 2020: Iran launched a new system: a protocol filter

Censors connections that do not match protocol fingerprints

Those that do match are then subjected to standard censorship

Geneva discovered 4 strategies to evade Iran’s filter
Responsive to new censorship events

July 29th 2020: China begins censoring the use of Encrypted SNI

Geneva discovered 6 strategies to evade ESNI censorship
Automating the arms race

AI has the potential to fast-forward the arms race for both sides.

- Bugs in implementation: Easy for censors to fix the low-hanging fruit.
- Gaps in logic: Harder for censors to fix systemic issues.

What is the logical conclusion of the arms race?
Automating Censorship Evasion

Geneva
Genetic Evasion

Discovers strategies quickly
New insights into GFW
Server-side evasion is possible
Code is open source

Geneva code and website [geneva.cs.umd.edu]