Perfect Imitation and Secure Asymmetry for Decoy Routing Systems with Slitheen

Cecylia Bocovich      Ian Goldberg

20 June 2017
EPFL Summer Research Institute
Censorship

Censors may monitor, alter or block traffic that enters or leaves their area of influence.
Censorship measurement studies in Iran [Aryan et al.], Pakistan [Nabi et al.], and China [Winter and Lindskog] show the following techniques:

- Filtering by IP address
- Filtering by hostname
- Protocol-specific throttling
- URL keyword filtering
- Active probing
- Application-layer DPI
Censorship Circumvention

Simple Proxies

- ✔ filtering by host
- ✗ filtering by IP address
Censorship Circumvention

Simple Proxies
✓ filtering by host
✗ filtering by IP address

Tor Bridges
✓ filtering by host
✗ filtering by IP address
✗ active probing
✗ traffic analysis
✗ app-layer DPI
Censorship Circumvention

Simple Proxies

✓ filtering by host
✗ filtering by IP address

Pluggable Transports

SkypeMorph (Mohajeri Moghaddam et al., 2012)
StegoTorus (Weinberg et al., 2012)
Marionette (Dyer et al., 2015)

Tor Bridges

✓ filtering by host
✗ filtering by IP address
✗ active probing
✗ traffic analysis
✗ app-layer DPI

ScrambleSuit (Winter et al., 2012)
Obfsproxy (Dingledine, 2012)
Censorship Circumvention

Simple Proxies
- ✔ filtering by host
- ✗ filtering by IP address

Tor Bridges
- ✔ filtering by host
- ✗ filtering by IP address
- ✗ active probing
- ✗ traffic analysis
- ✗ app-layer DPI

Pluggable Transports

SkypeMorph (Mohajeri Moghaddam et al., 2012)
StegoTorus (Weinberg et al., 2012)
Marionette (Dyer et al., 2015)

ScrambleSuit (Winter et al., 2012)
Obfsproxy (Dingledine, 2012)

(Houmansadr et al., 2013)
Decoy Routing

1. Establish TLS connection with overt site

Friendly ISP

TLS handshake

Uncensored (overt) site

Client
Decoy Routing

1. Establish TLS connection with overt site
2. Steganographically share TLS master secret with friendly ISP
   (Wustrow et al., 2011) (Houmansadr et al., 2011) (Karlin et al., 2011)
   (Wustrow et al., 2014) (Ellard et al., 2015)
Decoy Routing

1. Establish TLS connection with overt site
2. Steganographically share TLS master secret with friendly ISP
   \( (Wustrow \textit{et al.}, 2011) (Houmansadr \textit{et al.}, 2011) (Karlin \textit{et al.}, 2011) \\ (Wustrow \textit{et al.}, 2014) (Ellard \textit{et al.}, 2015) \)
3. Sever or abandon connection to the overt site
1. Establish TLS connection with overt site

2. Steganographically share TLS master secret with friendly ISP (Wustrow et al., 2011) (Houmansadr et al., 2011) (Karlin et al., 2011) (Wustrow et al., 2014) (Ellard et al., 2015)

3. Sever or abandon connection to the overt site

4. Proxy information between client and covert site
Attacks on Decoy Routing

(Wustrow et al., 2011)
(Schuchard et al., 2012)

Active Attacks

- Replay attacks
- Man in the middle
Attacks on Decoy Routing

(Wustrow et al., 2011)
(Schuchard et al., 2012)

Active Attacks

• Replay attacks
• Man in the middle

Routing-Based (RAD) Attacks

• TCP replay
• Crazy Ivan
Attacks on Decoy Routing

(Wustrow et al., 2011)
(Schuchard et al., 2012)

Active Attacks
- Replay attacks
- Man in the middle

Routing-Based (RAD) Attacks
- TCP replay
- Crazy Ivan

Passive Attacks
- Traffic analysis
- Latency analysis
Attacks on Decoy Routing

(Wustrow et al., 2011)
(Schuchard et al., 2012)

Active Attacks
• Replay attacks
• Man in the middle*

Routing-Based (RAD) Attacks
• TCP replay*
• Crazy Ivan

Passive Attacks
• Traffic analysis
• Latency analysis
Traffic Analysis

www.funnycats.com

GET index.html

726 bytes
Traffic Analysis

www.funnycats.com

GET index.html
726 bytes

GET style.css
3487 bytes

GET main.js
490 bytes
Traffic Analysis

www.funnycats.com

GET index.html
726 bytes

www.funnycats.com

GET style.css
GET main.js
3487 bytes
490 bytes

GET mycat.jpg
408 bytes

somecdn.com
Traffic Analysis

GET index.html 726 bytes

GET style.css 3487 bytes
GET main.js 490 bytes

GET mycat.jpg 408 bytes

www.funnycats.com

GET script.js 804 bytes

relay station

censored.com

somecdn.com
Traffic Analysis

GET index.html
726 bytes

GET style.css
3487 bytes
GET main.js
490 bytes

GET mycat.jpg
408 bytes

GET logo.png
850 + 100 bytes
GET pic.jpg

www.funnycats.com

www.funnycats.com

relay station
censored.com

somecdn.com
Latency Analysis

(Schuchard et al., 2012)
Slitheen traffic patterns to overt destinations are identical to a regular access to the overt site.

Covert content is squeezed into “leaf” resources (images, videos, etc.) that do not affect future connections for additional overt resources.
Architecture Overview

Client

SOCKS proxy (frontend)

Overt User Simulator (OUS)

Censor

Slitheen relay station

Uncensored (overt) site

Censored (covert) site

Tagged TLS handshake

HTTP GET notblocked.com

X-Slitheen: SOCKS data

X-Ignore: jkl&jdsa(#@$jkl

Proxy SOCKS data

HTTP 200 OK

Content-Type: image/png

Data from overt site

HTTP 200 OK

Content-Type: text/html

Data from overt site
Tagging Procedure

- Relay station has keypair \((r, g^r)\)

- Client picks \(s\), uses \(g^s \parallel H_1(g^r \parallel \chi)\) as ClientHello random

- Relay station (and only the relay station) can recognize the tag

- Client uses \(H_2(g^r \parallel \chi)\) as (EC)DHE private key

- Relay station can compute the TLS master secret and MITM the connection

- Relay station modifies the server’s Finished message to alert the client that Slitheen is active
Tagging Procedure

- Relay station has keypair \((r, g^r)\)
- Client picks \(s\), uses \(g^s \parallel H_1(g^{rs} || \chi)\) as ClientHello random
  - Relay station (and only the relay station) can recognize the tag
Tagging Procedure

- Relay station has keypair \((r, g^r)\)

- Client picks \(s\), uses \(g^s \parallel H_1(g^{rs} \parallel \chi)\) as ClientHello random
  - Relay station (and only the relay station) can recognize the tag

- Client uses \(H_2(g^{rs} \parallel \chi)\) as (EC)DHE private key
  - Relay station can compute the TLS master secret and MITM the connection
Tagging Procedure

- Relay station has keypair \((r, g^r)\)

- Client picks \(s\), uses \(g^s \parallel H_1(g^{rs} \parallel \chi)\) as ClientHello random
  - Relay station (and only the relay station) can recognize the tag

- Client uses \(H_2(g^{rs} \parallel \chi)\) as (EC)DHE private key
  - Relay station can compute the TLS master secret and MITM the connection

- Relay station modifies the server’s Finished message to alert the client that Slitheen is active
Data Replacement

**SOCKS proxy (frontend)**

HTTP GET resource
X-Slitheen: SlitheenID, streamID,
**SOCKS Connect, data**

**Overt User Simulator**

Assigns streamID
Data Replacement

SOCKS proxy (frontend) -> Overt User Simulator
Assigns streamID

HTTP GET resource X-Slitheen:SlitheenID, streamID, SOCKS Connect, data

TCP handshake
Censored (covert) site

Relay station
Data Replacement

SOCKS proxy (frontend)

HTTP GET resource
X-Slitheen: SlitheenID, streamID,
SOCKS Connect, data

TCP handshake
data

Censored (covert) site

Overt User Simulator
Assigns streamID

Relay station

Downstream queue for SlitheenID
Data Replacement

- **SOCKS proxy (frontend)**
  - HTTP GET resource
  - X-Slitheen: SlitheenID, streamID,
  - SO CKS Connect, data

- **Overt User Simulator**
  - Assigns streamID

- **Relay station**
  - TCP handshake
  - data
  - HTTP GET resource

- **Censored (covert) site**

- **Uncensored (overt) site**

- **Downstream queue for SlitheenID**
Data Replacement

SOCKS proxy (frontend)

Overt User Simulator
Assigns streamID

HTTP GET resource
X-Slitheen: SlitheenID, streamID,
SOCKS Connect, data

Relay station

TCP handshake
data

HTTP GET resource

Censored (covert) site

HTTP 200 OK

Uncensored (overt) site

Downstream queue for SlitheenID

Replace "leaf" (i.e. images, video, plain text) response body with downstream data
Encrypted HTTP responses are sent from the overt site in a series of TLS records.

TLS records can be (and often are) fragmented across packets.

We do not delay packets at the relay station to reconstruct records.
Finding Leaves

We can only decrypt a record after receiving all of it.
Finding Leaves

We can only decrypt a record after receiving all of it.

We only need to decrypt the HTTP response header to find leaves.
We can only decrypt a record after receiving all of it.

We only need to decrypt the HTTP response header to find leaves.

Misordered packets further complicate our decisions.
HTTP States

- START
- BEGIN_HEADER
- MID_HEADER
- BEGIN_CHUNK
- END_CHUNK
- MID_CHUNK
- RESPONSE
- BODY
- UNKNOWN

States:
- Beginning Header
- Mid-Header
- Beginning Chunk
- End Chunk
- Mid Chunk
- Response Body

Transitions:
- Decrypt missing or partial records
- Partial record

States:
- Unknown
Latency Results

Gmail

Wikipedia

Decoy page download time (ms)

CDF

Type
Decoy
Regular

Decoy page download time (ms)

CDF

Type
Decoy
Regular
Bandwidth

Downstream leaf content from the Alexa top 10,000 TLS sites

- Roughly 25% of all sites offer 500 kB or more of potentially replaceable content
- About 40% of traffic across all sites was leaf content
### Realistic Bandwidth

<table>
<thead>
<tr>
<th>Site name</th>
<th>Leaf content (bytes)</th>
<th>% leaf content replaced</th>
<th>% total replaced</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gmail</td>
<td>8800 ± 100</td>
<td>87.7 ± 0.2</td>
<td>23 ± 9</td>
</tr>
<tr>
<td>Wikipedia</td>
<td>24000 ± 2000</td>
<td>100 ± 0</td>
<td>33 ± 4</td>
</tr>
<tr>
<td>Yahoo</td>
<td>400000 ± 100000</td>
<td>100.0 ± 0.2</td>
<td>40 ± 20</td>
</tr>
<tr>
<td>Facebook</td>
<td>40000 ± 10000</td>
<td>0 ± 0</td>
<td>0 ± 0</td>
</tr>
</tbody>
</table>

![CDF Graph](image)
### Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>Telex</th>
<th>Cirripede</th>
<th>Curveball</th>
<th>TapDance</th>
<th>Rebound</th>
<th>Slitheen</th>
</tr>
</thead>
<tbody>
<tr>
<td>No in-line blocking</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Supports asymmetric routes</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
</tr>
<tr>
<td>Defends against TCP replay attacks</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Defends against latency analysis</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Defends against website fingerprinting</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
</tr>
<tr>
<td>RAD-resistant</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
</tbody>
</table>
Supporting Asymmetry and RAD-Resistance
Supporting Asymmetry and RAD-Resistance

- Slitheen station is on **downstream** path
  - Opposite to TapDance, Rebound
Supporting Asymmetry and RAD-Resistance

- Slitheen station is on **downstream** path
  - Opposite to TapDance, Rebound
- How does it identify tagged flows and learn the TLS master secret?
Supporting Asymmetry and RAD-Resistance

- Lightweight **gossip station** on upstream path
  - No flow blocking; just gets a copy of TLS flows
  - When it sees a TLS ClientHello (without having seen a TCP SYN ACK), broadcast it to Slitheen stations
  - If a Slitheen station claims the tag, send upstream TLS data to it
But surely that upstream ClientHello won’t get from the gossip station to the Slitheen station in time?

- The Slitheen station needs it before the TLS handshake completes so that it can read and modify the Finished message
Supporting Asymmetry and RAD-Resistance

- **Key idea**: the client’s Slitheen secret $s$ on its *next* connection to that overt site will be selected as a function of the *previous* client-relay shared secret

  - The first connection acts as a Cirripede-esque *registration*
  - The Slitheen station can then *predict* that client’s future ClientHello messages!
• Gossip stations offer a *two-tiered deployment* strategy
• No need for flow-blocking or traffic replacement routers
  • So easier to deploy
Supporting Asymmetry and RAD-Resistance

- Easier for censor to perform RAD attack on upstream data (change routing for *that one flow*) than downstream (advertise new BGP route to *everyone*)
  - Put lots of cheap gossip stations on possible upstream paths
  - More heavyweight Slithee stations on more stable downstream paths
<table>
<thead>
<tr>
<th>Feature</th>
<th>Telex</th>
<th>Telex+gossip</th>
<th>Cirripede</th>
<th>Curveball</th>
<th>Curveball+gossip</th>
<th>TapDance</th>
<th>Rebound</th>
<th>Slitheen</th>
<th>Slitheen+gossip</th>
</tr>
</thead>
<tbody>
<tr>
<td>No in-line blocking</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
</tr>
<tr>
<td>Supports asymmetric routes</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Defends against TCP replay attacks</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Defends against latency analysis</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>Defends against website fingerprinting</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>RAD-resistant</td>
<td>○</td>
<td>●</td>
<td>○</td>
<td>●</td>
<td>●</td>
<td>○</td>
<td>○</td>
<td>●</td>
<td>●</td>
</tr>
</tbody>
</table>
Summary

- Slitheen is a new proposal for a decoy routing system
- Slitheen addresses previously undefended passive attacks
- Our results show no discernible difference in latency between a “decoy access” to an overt destination and a regular access
- By design, Slitheen defends against website fingerprinting attacks by maintaining packet sizes, timings, and directionality
- The gossip protocol addresses the major challenges to deployability: RAD attacks, asymmetric flows, and concerns over inline blocking
- Implementation and source code of Slitheen (but not yet the gossip protocol) available: https://crysp.uwaterloo.ca/software/slitheen/