Website Fingerprinting Defenses at the Application Layer

Giovanni Cherubin\textsuperscript{1}  Jamie Hayes\textsuperscript{2}  Marc Juarez\textsuperscript{3}

\textsuperscript{1}Royal Holloway University of London
\textsuperscript{2}University College London
\textsuperscript{3}KU Leuven, ESAT/COSIC and imec

(to appear in PoPETS 2017)
Talk in the CrySyS Lab, Budapest, February 27, 2017
Tor

User

Tor network

Middle

Entry

Exit

Web
Website Fingerprinting (WF)
Open vs Closed World

Closed world

Open world
Tor Hidden Services (HS)

Introduction Point (IP)

Client

Rendezvous Point (RP)

xyz.onion

HS-I

HS-R

Client-R

HSDir
WF on Hidden Services

- Popular examples: SecureDrops, SilkRoad, etc.

- Kwon et al. (USENIX’15): HS circuit fingerprinting
  - The HS world can be considered a closed world

- HS are especially vulnerable to WF:
  - Anonymity makes them suitable to host sensitive content
  - Smaller world makes the attack work better
WF defenses

User → Entry → Adversary

Tor network

x.onion

y.onion

z.onion

Dummy
Real
Network- vs App-layer Defenses

- Existing defenses designed at the network layer. Why?
  - Identifying info originates at the app layer!

- Defences at the application layer:
  - Pros: fine-grained control in padding, no need to deal with the TCP stack.
  - Cons: only client and server can implement them, little incentives for servers (except for HSes!)
The HS world

- Exploratory crawl\(^1\): 5K hidden services (Ahmia.fi)
- Stats for the HS world (from intercepted HTTP)
  - Distrib. of types, sizes and number of resources
  - Most HS are small
- Assumptions: no JS and no 3rd-party content
  - 3rd party content is rare (less than 20%)
  - JS is rare (less than 13%)

\(^1\)https://github.com/webfp/tor-browser-selenium
LLaMA: introduction

- Client-side defense
- Inspired by Randomized Pipelining
- Implemented as a FF add-on
LLaMA: idea

- Add *random* delays to requests (C₂ in fig.)
- Make spurious requests:
  - Dedicated server (not evaluated)
  - Repeating previous requests (C₁’ in fig.)

\[ \delta \]

\[ C_1 \quad C_2 \]

\[ C_1' \quad C_2' \]
Evaluation Methodology

- Collect data with and without the defense: 100 HSes

- Evaluation:
  - Security: Measure accuracy of state-of-the-art WF attacks on the collected data: $k$-NN, $k$-Fingerprinting, CUMUL
  - Performance: measure latency (delay in seconds) and volume (extra padding byes) overheads

\[1\text{https://github.com/webfp/tor-browser-selenium} \]
LLaMA: results

- The accuracy drops 20-30%
- Less than 10% latency and bandwidth overhead

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>k-NN (%)</th>
<th>k-FP (%)</th>
<th>CUMUL (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>JS enabled</td>
<td>64.0</td>
<td>55.8</td>
<td>52.4</td>
</tr>
<tr>
<td>JS disabled</td>
<td>60.8</td>
<td>53.4</td>
<td>52.7</td>
</tr>
<tr>
<td>RP with delays</td>
<td>46.8</td>
<td>47.9</td>
<td>49.6</td>
</tr>
<tr>
<td>Extra requests</td>
<td>31.5</td>
<td>36.0</td>
<td>34.8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Overhead</th>
<th>Latency</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>%</td>
<td>Avg. (s)</td>
</tr>
<tr>
<td>JS disabled</td>
<td>–</td>
<td>5.01</td>
</tr>
<tr>
<td>RP with delays</td>
<td>8.4</td>
<td>5.42</td>
</tr>
<tr>
<td>Extra requests</td>
<td>9.8</td>
<td>5.49</td>
</tr>
</tbody>
</table>
ALPaCA: introduction

- First server-side defense against website fingerprinting
- Based on the idea that all app layer features map to size and timing at the network layer
- Implemented as a cronjob in the server
ALPaCA: idea (1)

- Pads resources (e.g., comments in HTML and adds random strings in the image’s metadata)

- It pads to a match sizes and resources to a target (fake or not) page.
ALPaCA: idea (2)

- Two ways to generate the target page:
  - Probabilistic (P-ALPaCA): sample the number of resources and sizes from the empirical distributions
  - Deterministic (D-ALPaCA): takes params $\delta, \lambda$
    - Pad the page objects to multiples of $\delta$
    - Create a number of fake objects to the next multiple of $\lambda$ objects
ALPaCA: evaluation

- 60-40% decrease in accuracy
- 50% latency and 86% volume overheads

<table>
<thead>
<tr>
<th>Overhead</th>
<th>%</th>
<th>Avg. (s)</th>
<th>%</th>
<th>Avg. (KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefended</td>
<td>–</td>
<td>3.99</td>
<td>–</td>
<td>175</td>
</tr>
<tr>
<td>P-ALPaCA</td>
<td>52.6</td>
<td>6.09</td>
<td>86.2</td>
<td>326</td>
</tr>
<tr>
<td>D-ALPaCA (2, 500, 5000)</td>
<td>66.3</td>
<td>6.63</td>
<td>3.66</td>
<td>182</td>
</tr>
<tr>
<td>D-ALPaCA (2, 5000, 5000)</td>
<td>56.1</td>
<td>6.22</td>
<td>9.84</td>
<td>193</td>
</tr>
<tr>
<td>D-ALPaCA (5, 2500, 5000)</td>
<td>61.7</td>
<td>6.44</td>
<td>15.1</td>
<td>202</td>
</tr>
<tr>
<td>D-ALPaCA (10, 5000, 5000)</td>
<td>41.7</td>
<td>5.65</td>
<td>44</td>
<td>254</td>
</tr>
</tbody>
</table>

Accuracy

<table>
<thead>
<tr>
<th></th>
<th>k-NN (%)</th>
<th>k-FP (%)</th>
<th>CUMUL (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Undefended</td>
<td>45.6</td>
<td>69.6</td>
<td>55.6</td>
</tr>
<tr>
<td>P-ALPaCA</td>
<td>0.2</td>
<td>9.5</td>
<td>15.6</td>
</tr>
<tr>
<td>D-ALPaCA (2, 500, 5000)</td>
<td>9.5</td>
<td>22.7</td>
<td>27.0</td>
</tr>
<tr>
<td>D-ALPaCA (2, 5000, 5000)</td>
<td>12.5</td>
<td>34.4</td>
<td>40.0</td>
</tr>
<tr>
<td>D-ALPaCA (5, 2500, 5000)</td>
<td>5.8</td>
<td>22.3</td>
<td>30</td>
</tr>
<tr>
<td>D-ALPaCA (10, 5000, 5000)</td>
<td>7.2</td>
<td>22.9</td>
<td>33.0</td>
</tr>
<tr>
<td>Decoy [21]</td>
<td>4.9</td>
<td>11.2</td>
<td>X</td>
</tr>
<tr>
<td>BuFLO [9]</td>
<td>5.3</td>
<td>13.3</td>
<td>X</td>
</tr>
</tbody>
</table>
Limitations and Future Work

- ALPaCA can only make sites bigger, but not smaller
- What’s the optimal padding at the app layer? Lack of a thorough feature analysis
- How do distributions change over time? How do we update our defenders accordingly?
  - How does the strategy need be adapted as HSes adopt our defense(s)?
Take aways

- App-layer defenses require a server-side component but are *easier* to implement

- SecureDrop case

- Source code up and running in hidden service: 3tmaadslguc72xc2.onion

- GitHub: [github.com/camelids](https://github.com/camelids)