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}imec-COSIC KU Leuven

19th July 2017, PETS’17, Minneapolis, MN, USA
Introduction: Website Fingerprinting (WF)
Tor Hidden Services (HS)

- HS: user visits xyz.onion without resolving it to an IP
- Examples: SecureDrop, Silkroad, DuckDuckGo, Facebook
Website Fingerprinting on Hidden Services (HSes)

- WF adversary can distinguish HSes from regular sites
- Website Fingerprinting in HSes is more threatening:
  - **Smaller world** makes HSes more identifiable
  - HS users vulnerable because content is **sensitive**
Website Fingerprinting defenses

WF Defenses
BuFLO
Tamaraw
CS-BuFLO
WTF-PAD
...

These are TCP packets or Tor messages

User
Entry
Middle
Tor network
Application-layer Defenses

• Existing defenses are designed at the network layer

Key observation: identifying info originates at app layer!

'Latent' features: $F_1, \ldots, F_n$

Web content

Identifying info

Observed features: $O_1, \ldots, O_n$

Last layer of encryption

Adversary

HTTP(S)

Tor

TLS

TCP

...
Pros and Cons of app-layer Defenses

The main advantage is that they are easier to implement:

- do not depend on Tor to be implemented

Cons:

- padding runs end-to-end
- may require server collaboration:

...but HSes have incentives!
LLaMA

- Client-side (FF add-on)
- Applied on HTTP requests
- More latency overhead

ALPaCA

- Server-side (first one)
- Applied on hosted content
- More bandwidth overhead

(two different solutions, not a client-server solution)
ALPaCA

- Abstract web pages as **num objects** and **object sizes**: pad them to match a target page
- Does not impact user experience:
  e.g., comments in HTML/JS, images’ metadata, *hidden* styles
ALPaCA strategies (1)

Example: protect a SecureDrop page

- Strategy 1: target page is Facebook

```
securedrop  index.html  securedrop.png  fake.css
facebook  index.html  facebook.png  style.css
Padding
```
ALPaCA strategies (2)

- Strategy 2: pad to an “anonymity set” target page

```
<table>
<thead>
<tr>
<th>securedrop</th>
<th>index.html</th>
<th>securedrop.png</th>
<th>fake.css</th>
</tr>
</thead>
<tbody>
<tr>
<td>facebook</td>
<td>index.html</td>
<td>facebook.png</td>
<td>style.css</td>
</tr>
<tr>
<td>target</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Define num objects and object sizes by:

- Deterministic: next multiple of $\lambda$, $\delta$
- Probabilistic: sampled from empirical distribution
• Inspired by Randomized Pipelining
  Goal: randomize HTTP requests

• Same goal from a FF add-on:
  - Random delays ($\delta$)
  - Repeat previous requests ($C_1$)
Evaluation: methodology

- Collect **with** and **without** defense: 100 HSes (cached)
  - Security: accuracy of attacks
    - $k$NN, $k$-Fingerprinting ($k$FP), CUMUL
  - Performance: overheads
    - latency (extra delay)
    - bandwidth (extra padding/time)
ALPaCA: results

- From 60% to 40% decrease in accuracy
- 50% latency and 85% bandwidth overheads
LLaMA: results

- Accuracy drops between 20% and 30%
- Less than 10% latency and bandwidth overheads
Take aways

• WF defenses at the app layer are **easier to implement**

• **HSes have incentives** to support server-side defenses:

  SecureDrop has implemented a prototype of ALPaCA

• ALPaCA is running on a HS: [3tmaadslguc72xc2.onion](http://3tmaadslguc72xc2.onion)

• Source code: [github.com/camelids](https://github.com/camelids)