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

10th July 2017, Cybersecurity Lab, RIT, Rochester, NY, USA
Introduction: Website Fingerprinting (WF)
Website Fingerprinting: deployment
Tor Hidden Services (HS)

- User can visit xyz.onion without resolving to an IP
- Examples: securedrop, silkroad, duckduckgo, facebook
Website Fingerprinting on Hidden Services

- Kwon et al. (USENIX’15): such an adversary can distinguish HSes from regular sites
- Website Fingerprinting is more threatening for HSes:
  - Smaller world makes the sites more identifiable
  - Users are vulnerable because content is likely to be sensitive.
Website Fingerprinting defenses

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

User
Entry
Middle
Tor network

These are TCP/IP packets or Tor messages
Network- vs App-layer Defenses

- Existing defenses are designed at the network layer
  
  **However: identifying info originates at the app layer!**

- Defenses at the application layer:
  
  - Pros: fine-grained and *indistinguishable* padding
  
  - Cons: bandwidth overhead on Exit Nodes, may require server collaboration (but HSees have incentives!)
<table>
<thead>
<tr>
<th>LLaMA</th>
<th>ALPaCA</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Client-side (FF add-on)</td>
<td>• Server-side</td>
</tr>
<tr>
<td>• Applied on HTTP requests</td>
<td>• Applied on hosted content</td>
</tr>
<tr>
<td>• More latency overhead</td>
<td>• More bandwidth overhead</td>
</tr>
</tbody>
</table>

(They run independently from each other)
• Since content is encrypted, abstract web pages as sets of objects:
  pad the objects to match a target page

• Set of primitives to pad resources by type without altering user experience:
  e.g., comments in HTML/JS, random strings in image’ metadata
ALPaCA (2)

- Two ways to generate a target page from the original:
  - **Deterministic** (D-ALPaCA): takes params $\lambda$, $\delta$
    - Number of objects in the page is the next multiple of $\lambda$
    - Sizes of objects are next multiple of $\delta$
  - **Probabilistic** (P-ALPaCA): sample number of objects and object sizes from empirical distributions
LLaMA

- Inspired on Randomized Pipelining: randomize HTTP requests.
- How to re-order them from a FF add-on:
  - Randomly delays HTTP requests to change their order.
  - Repeat previous requests.
Evaluation: methodology

- Collect data with and without the defense: 100 HSes
  - Security: measured by the accuracy of attacks: $kNN$, $k$-Fingerprinting ($kFP$), CUMUL
  - Performance: measured by overhead in:
    - *latency* (extra delay)
    - *volume* (extra padding)
Results

- ALPaCA: comparable to previous defenses
  - 60-40% decrease in accuracy
  - 50% latency and 80% bandwidth overheads
- LLaMA: lightweight but mild protection
  - 20-30% decrease in accuracy
  - Less than 10% latency and bandwidth overheads
Take aways

• It is more natural to design WF defenses at the app layer
  They achieve similar protection and are more practical
• HSes are the perfect scenario for server-side defenses:
  SecureDrop has implemented a prototype of ALPaCA
• ALPaCA running on a HS: 3tmaadslguc72xc2.onion
• Source code: github.com/camelids