Veil: Private Browsing Semantics without Browser-side Assistance

Frank Wang (MIT CSAIL), James Mickens (Harvard), Nickolai Zeldovich (MIT CSAIL)
All popular browsers offer private browsing

In private browsing, web pages shouldn’t leave identifiable, persistent client-side state
Private browser modes are leaky

- DNS cache and database pollution
- Leave RAM artifacts in page swap, hibernation files
- Forensic tools can easily recover this data and fingerprint activity
Problem: Private browsing is hard to implement with only client-side support

• Browsers complex and constantly adding new features

• They lack a priori knowledge of sensitive content
  - Example: prevent RAM from paging to disk, use `mlock()` to pin memory

• Even transmission of web content to a user can pollute in-memory and on-disk regions
What if developers can implement private browsing semantics?

**Goal:** Protect greppable content from post-session attacker

**Insight:** Web services control
1) the content they deliver
2) the servers that deliver this content
Our solution: Veil

<table>
<thead>
<tr>
<th>Developer</th>
<th>Blinding servers</th>
<th>Client browser</th>
</tr>
</thead>
</table>
Our solution: Veil

Developer

Blinding servers

Client browser

HTML

CSS

JS

Image

Veil Compiler

Rewrite webpages
Our solution: Veil

Developer

- HTML
- CSS
- JS
- Image

Blinding servers

Store and mutate content

Client browser

Veil Compiler

Rewrite webpages
Our solution: Veil

Developer

HTML
CSS
JS
Image

Veil Compiler

Rewrite webpages

Blinding servers

Store and mutate content

Client browser

Veil Bootstrap page

Hide page-specific URLs and display pages
Contributions

Veil: first web framework that allows developers to provide private browsing semantics

• Techniques, such as URL blinding, content mutation, and heap walking, to protect privacy

• Two browsing modes to provide different amounts of privacy

• Evaluation on real websites

• No client/browser changes required
Outline

• Veil Architecture
• Implementation
• Evaluation
Overview

Developer

Blinding servers

Client browser

Veil Compiler

URL blinding, Content mutation

Veil Bootstrap page
Veil Compiler

<html>
  <script...>
  <link...>
  <img...>
</html>
Veil Compiler

<html>
<script...>
<link...>
<img...>
</html>

Veil Compiler

JS
CSS
Image
Veil Compiler

```html
<html>
  <script>veilFetch(“b6f…”)</script>
  <script>veilFetch(“94d…”)</script>
  <script>veilFetch(“7cb…”)</script>
</html>
```

Veil Compiler

- JS
- CSS
- Image

- b6f4...
- 94d2...
- 7cb1...
Veil Compiler

<html>
  <script...>veilFetch("b6f...")</script>
  <script...>veilFetch("94d...")</script>
  <script...>veilFetch("7cb...")</script>
</html>
Blinding Servers

Veil Compiler

```html
<html>
  <script>veilFetch("b6f...")</script>
  <script>veilFetch("94d...")</script>
  <script>veilFetch("7cb...")</script>
</html>
```
Blinding Servers

Veil Compiler

<html>
<script>veilFetch("b6f...")</script>
<script>veilFetch("94d...")</script>
<script>veilFetch("7cb...")</script>
</html>

Blinding Servers
veil.io
Blinding Servers

Veil Compiler

<html>
<script>veilFetch("b6f...")</script>
<script>veilFetch("94d...")</script>
<script>veilFetch("7cb...")</script>
</html>

put(b6f..., jsData)
put(94d..., cssData)
put(7cb..., imgData)
put(foo.com, index.html, rewrittenHTML)

Blinding Servers
veil.io
Client Browser: Dynamic Reassembly

Browser
Client Browser: Dynamic Reassembly

Browser

veil.io
Client Browser: Dynamic Reassembly

Browser

GET /index.html

Blinding Servers

veil.io
Client Browser: Dynamic Reassembly

Browser

GET /index.html

Blinding Servers

Page to load
http://foo.com

User Password
***************
Client Browser: Dynamic Reassembly

Browser

Page to load
http://foo.com

User Password
***************
Client Browser: Dynamic Reassembly

GET <foo.com/index.html>_{Kuser}
Veil-Key: <{Kuser}_{Kpub_veil}>

Browser

Page to load
http://foo.com

User Password
***************

Blinding Servers
Client Browser: Dynamic Reassembly

Browser

GET <foo.com/index.html>Kuser

Veil-Key: <K_{user}>_Kpub_veil

<HTML>Kuser

Blinding Servers
Client Browser: Dynamic Reassembly

GET <foo.com/index.html>

Veil-Key: $K_{user}$

Blinding Servers

Browser

```html
<html>
  <script>
    veilFetch("94d2...");
    veilFetch("b6f4...");
    veilFetch("7cb1...");
  </script>
</html>
```
Client Browser: Dynamic Reassembly

Browser

<html>
<script>
    veilFetch("94d2...");
    veilFetch("b6f4...");
    veilFetch("7cb1...");
</script>
</html>

Blinding Servers
Client Browser: Dynamic Reassembly

```html
<html>
  <script>
    veilFetch("94d2...");
    veilFetch("b6f4...");
    veilFetch("7cb1...");
  </script>
</html>
```

GET <foo.com/b6f4…> \( K_{user} \)
GET <foo.com/94d2…> \( K_{user} \)
GET <foo.com/7cb1…> \( K_{user} \)

Veil Key: \( K_{user} K_{pub\_veil} \)

Browser

Blinding Servers
Client Browser: Dynamic Reassembly

```
<html>
    <script>
        veilFetch("94d2...");
        veilFetch("b6f4...");
        veilFetch("7cb1...");
    </script>
</html>
```

Browser

GET <foo.com/b6f4...>\textbf{Kuser}
GET <foo.com/94d2...>\textbf{Kuser}
GET <foo.com/7cb1...>\textbf{Kuser}

Veil Key: \textbf{<Kuser>_{Kpub\_veil}}

<HTTP Objects>\textbf{Kuser}

Blinding Servers
Client Browser: Dynamic Reassembly

GET <foo.com/b6f4…>\textsubscript{Kuser}
GET <foo.com/94d2…>\textsubscript{Kuser}
GET <foo.com/7cb1…>\textsubscript{Kuser}

Veil Key: <\textsubscript{Kuser}\textsubscript{Kpub\_veil}}

<HTTP Objects>\textsubscript{Kuser}

---

Browser

foo.com

page

---

Blinding Servers
Client Browser: Dynamic Reassembly

GET `<foo.com/b6f4…>` → Kuser
GET `<foo.com/94d2…>` → Kuser
GET `<foo.com/7cb1…>` → Kuser

Veil Key: `<Kuser>Kpub_veil`

<HTTP Objects> → Kuser

Browser

Blinding Servers

Key Technique: Blinded URLs
Protecting RAM Artifacts

• Heap Walking: reduce likelihood of swap rooted at markAsSensitive() tree

• Content Mutation: not leak site-specific content
  - Noise to images
  - Add junk code
  - eval()-folding

• More details in the paper
Protecting RAM Artifacts

- Heap Walking: reduce likelihood of swap rooted at `markAsSensitive()` tree
- Content Mutation: not leak site-specific content
  - Noise to images
  - Add junk code
  - `eval()`-folding
- More details in the paper

URL blinding, heap walking, and content mutation are inherently unimplementable by the browser!
Problem: Complex DOM structure still exists
Problem: Complex DOM structure still exists

• Heap walking reduces likelihood of swapping
• Content mutation makes it difficult to recover
• However, some sites might want to minimize client-side DOM state
Problem: Complex DOM structure still exists

- Heap walking reduces likelihood of swapping
- Content mutation makes it difficult to recover
- However, some sites might want to minimize client-side DOM state

Solution: DOM hiding mode
- User’s browser as a thin client
- Remote server loads real page, applies GUI event, and returns screenshot of updated page
DOM Hiding Mode

Browser

Veil remoting JS stub
Page to load
http://foo.com
User Password
**************

Blinding Servers

Content Provider

Headless browser
Veil GUI proxy
DOM Hiding Mode

Browser

veil.io

Veil remoting JS stub

Page to load
http://foo.com

User Password
**************

Blinding Servers

Content Provider

Headless browser

Veil GUI proxy
DOM Hiding Mode

Browser

Veil remoting JS stub
Page to load
http://foo.com
User Password
**************

Blinding Servers

Content Provider

Normal version of page
Headless browser
Veil GUI proxy

veil.io
DOM Hiding Mode

Browser

veil.io

Veil remoting JS stub

Page to load

http://foo.com

User Password

**************

Blinding Servers

Content Provider

Normal version of page

Headless browser

Veil GUI proxy
DOM Hiding Mode

Browser

Screenshot of foo.com

Blinding Servers

Content Provider

Normal version of page

Headless browser

Veil GUI proxy
DOM Hiding Mode

Browser

Screenshot of foo.com

Blinding Servers

Content Provider

Normal version of page

Headless browser

Veil GUI proxy
DOM Hiding Mode

Browser

Screenshot of foo.com

Blinding Servers

Content Provider

Normal version of page

Headless browser

Veil GUI proxy
DOM Hiding Mode

Browser

Screenshot of foo.com

Blinding Servers

Click Event

Content Provider

Page after click event

Headless browser

Veil GUI proxy

Click Event
DOM Hiding Mode

Browser

Screenshot of foo.com after click

Blinding Servers

Content Provider

Page after click event

Headless browser

Veil GUI proxy
Outline

• Veil Architecture
• Implementation
• Evaluation
Implementation

- 4 components
  - Compiler
  - Blinding servers
  - JS library for bootstrap page and DOM hiding mode
  - GUI proxy (for DOM hiding mode)
- Compiler and blinding server written in Python
- GUI proxy uses headless Chrome
- BeautifulSoup to parse and mutate HTML
Page Load Times: Standard Veil Mode

Page load time is 1.25x-3.25x higher with Veil
Page Load Times: Standard Veil Mode

Page load time is 1.25x-3.25x higher with Veil

- Imgur
- Woot
- WaPo
- MIT
- Piechopper
- Google

Bar chart showing page load time for different sites with regular and veil modes.
Page Load Times: Veil in DOM Hiding Mode

Page load time is 1.2x-2.1x higher with Veil in DOM hiding mode

- Regular page load
- DOM hiding mode Veil
Related Work

• CleanOS, Lacuna, PrivExec
  - uses secure deletion to implement privacy
  - require configuration and installation of special runtime
  - cannot protect sensitive data unless abstractions spread across the whole system

• UCognito
  - requires modified client-side stack
  - does not prevent information leakage via non-sandboxed parts
Conclusions

• Traditional private browsing modes still leak information!

• Veil allows developers to improve privacy semantics of their pages

• Veil uses a variety of techniques, which are unimplementable by the browser, to hide sensitive information from post-session attackers

• We evaluated Veil on various real websites and found moderate overhead