Using OpenWPM to Measure Tracking on the Web

A demonstration of how OpenWPM and the Princeton 1-million-site Web Census data can help your research.

Steven Englehardt
@s_englehardt

Arvind Narayanan
@random_walker
Visiting 2 websites results in 84 third parties contacted
Visiting 2 websites results in 84 third parties contacted

Proliferation of tracking in the absence of transparency
Visiting 2 websites results in 84 third parties contacted

Proliferation of tracking in the absence of transparency

...but measurement can fix that
Tracking with browser state

Browser example.com
Tracking with browser state

Browser

example.com
Tracking with browser state

Browser

to

e.example.com

e.example.com
Tracking with browser state
Tracking with fingerprinting

Browser

example.com
Tracking with fingerprinting

Browser

Fonts
Screen Resolution
Plugins

example.com
Tracking with fingerprinting

Browser

Fonts
Screen Resolution
Plugins

example.com
Open Web Privacy Measurement (OpenWPM)

https://github.com/citp/OpenWPM
The Princeton Web Census

Monthly
1 Million Site Crawl

Collecting:

- Javascript Calls
- All javascript files
- HTTP Requests and Responses
- Storage (cookies, Flash, etc)
Tackling open questions with OpenWPM

1. Measure new fingerprinting techniques
2. Examine tracking of logged in users
3. Study personalized advertisements
4. Examine the tracking practices of browser extensions
5. Measure price discrimination based on browsing history
Using OpenWPM and the Princeton Web Census in your research

1. Analyze our monthly 1-million-site measurement data
2. Use OpenWPM to run your own measurements
3. Add new features and instrumentation to OpenWPM
Download our public postgres dumps

## Data

The data is available as bzipped PostgreSQL dumps. The schema file used in all of the datasets is available [here](https://webtransparency.cs.princeton.edu/webcensus/index.html#data).

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Million Site Stateless</td>
<td>Parallel Stateless Crawl</td>
</tr>
<tr>
<td>100k Site Stateful</td>
<td>Parallel Stateful Crawl – 10,000 site seed profile</td>
</tr>
<tr>
<td>10k Site ID Detection (1)</td>
<td>Sequential Stateful Crawl -- Flash enabled -- Synced with ID Detection (2)</td>
</tr>
<tr>
<td>10k Site ID Detection (2)</td>
<td>Sequential Stateful Crawl -- Flash enabled -- Synced with ID Detection (1)</td>
</tr>
<tr>
<td>55k Site Stateless with cookie blocking</td>
<td>Parallel Stateless Crawl -- Firefox set to block all third-party cookies</td>
</tr>
<tr>
<td>55k Site Stateless with Ghostery</td>
<td>Parallel Stateless Crawl -- Ghostery extension installed and set to block all possible trackers</td>
</tr>
<tr>
<td>55k Site Stateless with HTTPS Everywhere</td>
<td>Parallel Stateless Crawl -- HTTPS Everywhere installed</td>
</tr>
</tbody>
</table>
Download our public postgres dumps

Continuous data release planned for the future.

Contact us if you’re interested in accessing new data!

https://webtransparency.cs.princeton.edu/webcensus/index.html#data
Future work to provide easy access to data
Using OpenWPM and the Princeton Web Census in your research

1. Analyze our monthly 1-million-site measurement data
2. Use OpenWPM to run your own measurements
3. Add new features and instrumentation to OpenWPM
<table>
<thead>
<tr>
<th>Study using OpenWPM</th>
<th>Conference</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Web Never Forgets: Persistent Tracking Mechanisms in the Wild</td>
<td>CCS</td>
<td>2014</td>
</tr>
<tr>
<td>Cognitive disconnect: Understanding Facebook Connect login permissions</td>
<td>OSN</td>
<td>2014</td>
</tr>
<tr>
<td><strong>Cookies that give you away: The surveillance implications of web tracking</strong></td>
<td>WWW</td>
<td>2015</td>
</tr>
<tr>
<td>Upgrading HTTPS in midair: HSTS and key pinning in practice</td>
<td>NDSS</td>
<td>2015</td>
</tr>
<tr>
<td>Web Privacy Census</td>
<td>Tech Science</td>
<td>2015</td>
</tr>
<tr>
<td>Variations in Tracking in Relation to Geographic Location</td>
<td>W2SP</td>
<td>2015</td>
</tr>
<tr>
<td>No Honor Among Thieves: A Large-Scale Analysis of Malicious Web Shells</td>
<td>WWW</td>
<td>2016</td>
</tr>
<tr>
<td><strong>Online Tracking: A 1-million-site Measurement and Analysis</strong></td>
<td>CCS</td>
<td>2016</td>
</tr>
<tr>
<td>Dial One for Scam: Analyzing and Detecting Technical Support Scams</td>
<td>NDSS</td>
<td>2017</td>
</tr>
<tr>
<td>Study using OpenWPM</td>
<td>Conference</td>
<td>Year</td>
</tr>
<tr>
<td>----------------------------------------------------------</td>
<td>------------</td>
<td>-------</td>
</tr>
<tr>
<td>The Web Never Forgets: Persistent Tracking Mechanisms in the Wild</td>
<td>CCS</td>
<td>2014</td>
</tr>
<tr>
<td>Cognitive disconnect: Understanding Facebook Connect login permissions</td>
<td>OSN</td>
<td>2014</td>
</tr>
<tr>
<td><strong>Cookies that give you away: The surveillance implications of web tracking</strong></td>
<td>WWW</td>
<td>2015</td>
</tr>
<tr>
<td>Upgrading HTTPS in midair: HSTS and key pinning in practice</td>
<td>NDSS</td>
<td>2015</td>
</tr>
<tr>
<td>Web Privacy Census</td>
<td>Tech Science</td>
<td>2015</td>
</tr>
<tr>
<td>Variations in Tracking in Relation to Geographic Location</td>
<td>W2SP</td>
<td>2015</td>
</tr>
<tr>
<td>No Honor Among Thieves: A Large-Scale Analysis of Malicious Web Shells</td>
<td>WWW</td>
<td>2016</td>
</tr>
<tr>
<td><strong>Online Tracking: A 1-million-site Measurement and Analysis</strong></td>
<td>CCS</td>
<td>2016</td>
</tr>
<tr>
<td>Dial One for Scam: Analyzing and Detecting Technical Support Scams</td>
<td>NDSS</td>
<td>2017</td>
</tr>
</tbody>
</table>
Simple Python interface to run crawls

```python
from automation import TaskManager, CommandSequence

# The list of sites that we wish to crawl
NUM_BROWSERS = 3
sites = ['http://www.example.com',
         'http://www.princeton.edu',
         'http://clispr.princeton.edu/]

# Loads the manager preference and 3 copies of the default browser dictionaries
manager_params, browser_params = TaskManager.load_default_params(NUM_BROWSERS)

# Update browser configuration (use this for per-browser settings)
for i in range(NUM_BROWSERS):
    browser_params[i]['disable_flash'] = False  # Enable flash for all three browsers
    browser_params[i]['headless'] = True  # Launch only browser 0 headless

# Update TaskManager configuration (use this for crawl-wide settings)
manager_params['data_directory'] = '~/Desktop/'
manager_params['log_directory'] = '~/Desktop/

# Instantiates the measurement platform
# Commands time out by default after 60 seconds
manager = TaskManager.TaskManager(manager_params, browser_params)

# Visits the sites with all browsers simultaneously
for site in sites:
    command_sequence = CommandSequence.CommandSequence(site)
    # Start by visiting the page
    command_sequence.get(sleep=0, timeout=0)
    # dump_profile_cookies/dump_flash_cookies closes the current tab.
    command_sequence.dump_profile_cookies(120)
    manager.execute_command_sequence(command_sequence, index=0)  # ** = synchronized browsers

# Shuts down the browsers and waits for the data to finish logging
manager.close()
```
Simple Python interface to run crawls

1. Specify sites to crawl

```python
from automation import TaskManager, CommandSequence

# The list of sites that we wish to crawl
NUM_BROWSERS = 3
sites = ['http://www.example.com',
         'http://www.princeton.edu',
         'http://citp.princeton.edu']

# Loads the manager preference and 3 copies of the default browser dictionaries
manager_params, browser_params = TaskManager.load_default_params(NUM_BROWSERS)

# Update browser configuration (use this for per-browser settings)
for i in range(NUM_BROWSERS):
    browser_params[i]['disable_flash'] = False
    browser_params[i]['headless'] = True
    launch only browser 0 headless

# Update TaskManager configuration (use this for crawl-wide settings)
manager_params['data_directory'] = '~/Desktop/'
manager_params['log_directory'] = '~/Desktop/'

# Instantiates the measurement platform
manager = TaskManager.TaskManager(manager_params, browser_params)

# Visits the sites with all browsers simultaneously
for site in sites:
    command_sequence = CommandSequence.CommandSequence(site)

    # Start by visiting the page
    command_sequence.get(sleep=0, timeout=10)

    # dump_profile_cookies/dump_flash_cookies closes the current tab.
    command_sequence.dump_profile_cookies(120)

    manager.execute_command_sequence(command_sequence, index='**') # ** = synchronized browsers

# Shuts down the browsers and waits for the data to finish logging
manager.close()
```
Simple Python interface to run crawls

```
from automation import TaskManager, CommandSequence

# The list of sites that we wish to crawl
NUM_BROWSERS = 3
sites = ['http://www.example.com',
         'http://www.princeton.edu',
         'http://citi.princeton.edu']

# Loads the manager preference and 3 copies of the default browser dictionaries
manager_params, browser_params = TaskManager.load_default_params(NUM_BROWSERS)

# Update browser configuration (use this for per-browser settings)
for i in range(NUM_BROWSERS):
    browser_params[i]['disable_flash'] = False # disable flash for all three browsers
    browser_params[i]['headless'] = True # launch only browser # headless

# Update TaskManager configuration (use this for crawl-wide settings)
manager_params['data_directory'] = '~/Desktop/
manager_params['log_directory'] = '~/Desktop/

# Instantiates the measurement platform
manager = TaskManager.TaskManager(manager_params, browser_params)

# Commands time out by default after 60 seconds
manager = TaskManager.TaskManager(manager_params, browser_params)

# Visits the sites with all browsers simultaneously
for site in sites:
    command_sequence = CommandSequence.CommandSequence(site)

    # Start by visiting the page
    command_sequence.get(sleep=0, timeout=0)

    # dump_profile_cookies/dump_flash_cookies closes the current tab.
    command_sequence.dump_profile_cookies(120)

    manager.execute_command_sequence(command_sequence, index='**') # ** = synchronized browsers

# Shuts down the browsers and waits for the data to finish logging
manager.close()
```

1. Specify sites to crawl
2. Specify the number of browsers to use
Simple Python interface to run crawls

1. Specify sites to crawl
2. Specify the number of browsers to use
3. Configure instrumentation / platform
Simple Python interface to run crawls

1. Specify sites to crawl
2. Specify the number of browsers to use
3. Configure instrumentation / platform
4. Submit commands during each page visit
Adding new commands is easy

Adding a new command

OpenWPM commands exist as part of a command sequence object, which allows one to string together a sequence of actions for a browser to take and deploy that sequence to the first available browser from the manager. Adding a command to a CommandSequence object will cause the browser to execute it immediately after the previously added command as long as the previous command does not time out or fail.

Suppose we want to add a top-level command to cause the browser to jiggle the mouse a few times. We may want to have the browser visit a site, jiggle the mouse, and extract the links from the site.

First, let's define the command in a way that we can call it from the TaskManager, serialize the call information, and send it across the process boundary to the BrowserManager that has access to the Selenium instance. In automation/CommandSequence.py, we add the following:

```python
def jiggle_mouse(self, num_jiggles, timeout=60):
    *** jiggle mouse <num_jiggles> times ***
    self.total_timeout += timeout
    if not self.contains_get_or_browse:
        raise CommandExecutionError("No get or browse request preceding "
                                      "the jiggle mouse command", self)
    command = ("JIGGLE_MOUSE", num_jiggles)
    self.commands_with_timeout.append((command, timeout))
```

Notice that any arguments to the command are added both to the command sequence top-level method, and are then passed to the command tuple to be serialized and sent across the process boundary between the task manager and browser manager. The general format for the command

https://github.com/citp/OpenWPM/wiki/Platform-Demo#adding-a-new-command
Using OpenWPM and the Princeton Web Census in your research

1. Analyze our monthly 1-million-site measurement data
2. Use OpenWPM to run your own measurements
3. Add new features and instrumentation to OpenWPM
<table>
<thead>
<tr>
<th>#</th>
<th>Title</th>
<th>Author</th>
<th>Status</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Add a 'tab closed' and 'tab loaded' attribute to CommandSequence.py</td>
<td>engleradt</td>
<td>Open</td>
<td>enhancement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>#99 opened 7 days ago</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Investigate Selenium 3 and geckodriver compatibility</td>
<td>gunesacar</td>
<td>Open</td>
<td>enhancement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>needs-investigation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#93 opened on Sep 24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Add current url bar domain to all extension instrumentation</td>
<td>engleradt</td>
<td>Open</td>
<td>enhancement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>#77 opened on May 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Use extension cookie instrumentation as the default cookie instrumentation</td>
<td>engleradt</td>
<td>Open</td>
<td>enhancement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>#76 opened on May 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Support FourthParty style HTTP instrumentation in the Firefox extension</td>
<td>engleradt</td>
<td>Open</td>
<td>enhancement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>high-priority</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#71 opened on Apr 20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Javascript instrumentation should be configurable per-API</td>
<td>engleradt</td>
<td>Open</td>
<td>enhancement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>help wanted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#68 opened on Apr 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Platform should track current Firefox version and warn user if different</td>
<td>engleradt</td>
<td>Open</td>
<td>enhancement</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#66 opened on Apr 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Tests needed for browser commands</td>
<td>engleradt</td>
<td>Open</td>
<td>enhancement</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>help wanted</td>
</tr>
<tr>
<td></td>
<td></td>
<td>#65 opened on Apr 8</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Easy to measure new fingerprinting techniques

Canvas Fingerprinting

```javascript
// Access to canvas
instrumentObject(
    window.HTMLCanvasElement.prototype,
    "HTMLCanvasElement", true
);

var excludedProperties = [ "quadraticCurveTo", "lineTo", "transform", "globalAlpha", "moveTo", "drawImage", "setTransform", "clearRect", "closePath", "beginPath", "canvas", "translate" ];

instrumentObject(
    window.CanvasRenderingContext2D.prototype,
    "CanvasRenderingContext2D", true,
    excludedProperties
);
```

WebRTC Local IP Retrieval

```javascript
// Access to webRTC
instrumentObject(
    window.RTCPeerConnection.prototype,
    "RTCPeerConnection", true
);
```
Insights from our own studies using OpenWPM and Princeton Web Census data
Better understand the tracking ecosystem

https://webtransparency.cs.princeton.edu/webcensus/
Measure persistent tracking

Cookie Syncing

Cookie Respawning

https://webtransparency.cs.princeton.edu/webcensus/
Measure persistent tracking

45 of the top 50 third parties

http://webtransparency.cs.princeton.edu/webcensus/
Measure persistent tracking

**Cookie Syncing**

45 of the top 50 third parties

**Cookie Respawning**

Largely unused by US-based 3rd parties

https://webtransparency.cs.princeton.edu/webcensus/
Measure the adoption of fingerprinting techniques

Canvas

WebRTC

Audio

Battery

https://webtransparency.cs.princeton.edu/webcensus/
Test the effectiveness of Privacy Tools

Ghostery

AdBlock
Plus

Third-party
cookie blocking

https://webtransparency.cs.princeton.edu/webcensus/
Test the effectiveness of Privacy Tools

Ghostery

AdBlock Plus

Third-party cookie blocking

Block stateful tracking well, but miss many fingerprinting scripts

https://webtransparency.cs.princeton.edu/webcensus/
Thanks for listening!

Full Paper:  
senglehardt.com/papers/ccs16_online_tracking.pdf

Princeton Web Census Data and Analysis:  
webtransparency.cs.princeton.edu/webcensus/

Collaborate:  
webtap.princeton.edu/research/

Email: ste@cs.princeton.edu  Twitter: @s_englehardt  Web: senglehardt.com