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.

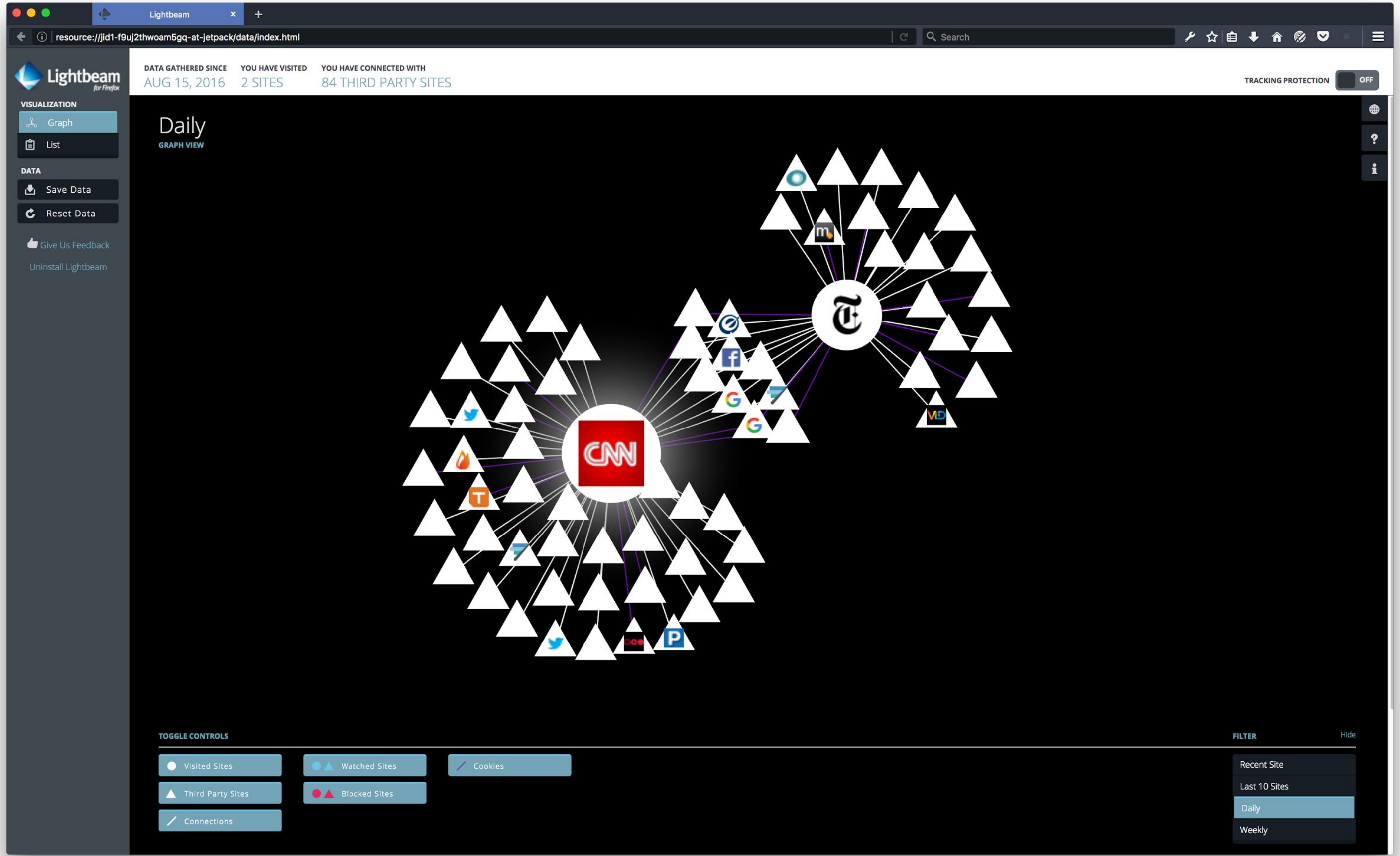
Steven Englehardt
@s_englehardt

Arvind Narayanan
@random_walker

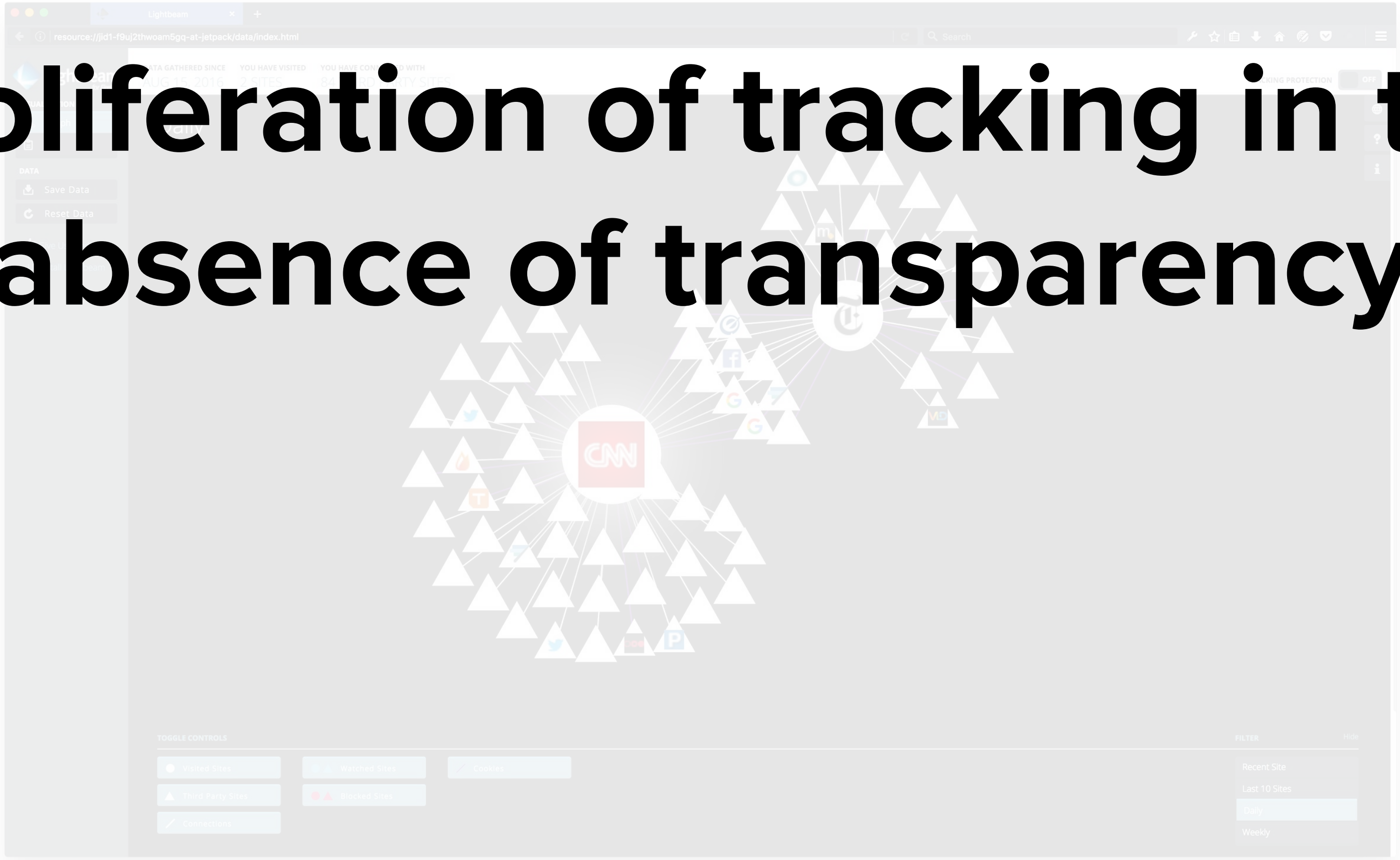
PRINCETON UNIVERSITY

CENTER FOR INFORMATION TECHNOLOGY POLICY
PRINCETON UNIVERSITY

Visiting 2 websites results in 84 third parties contacted

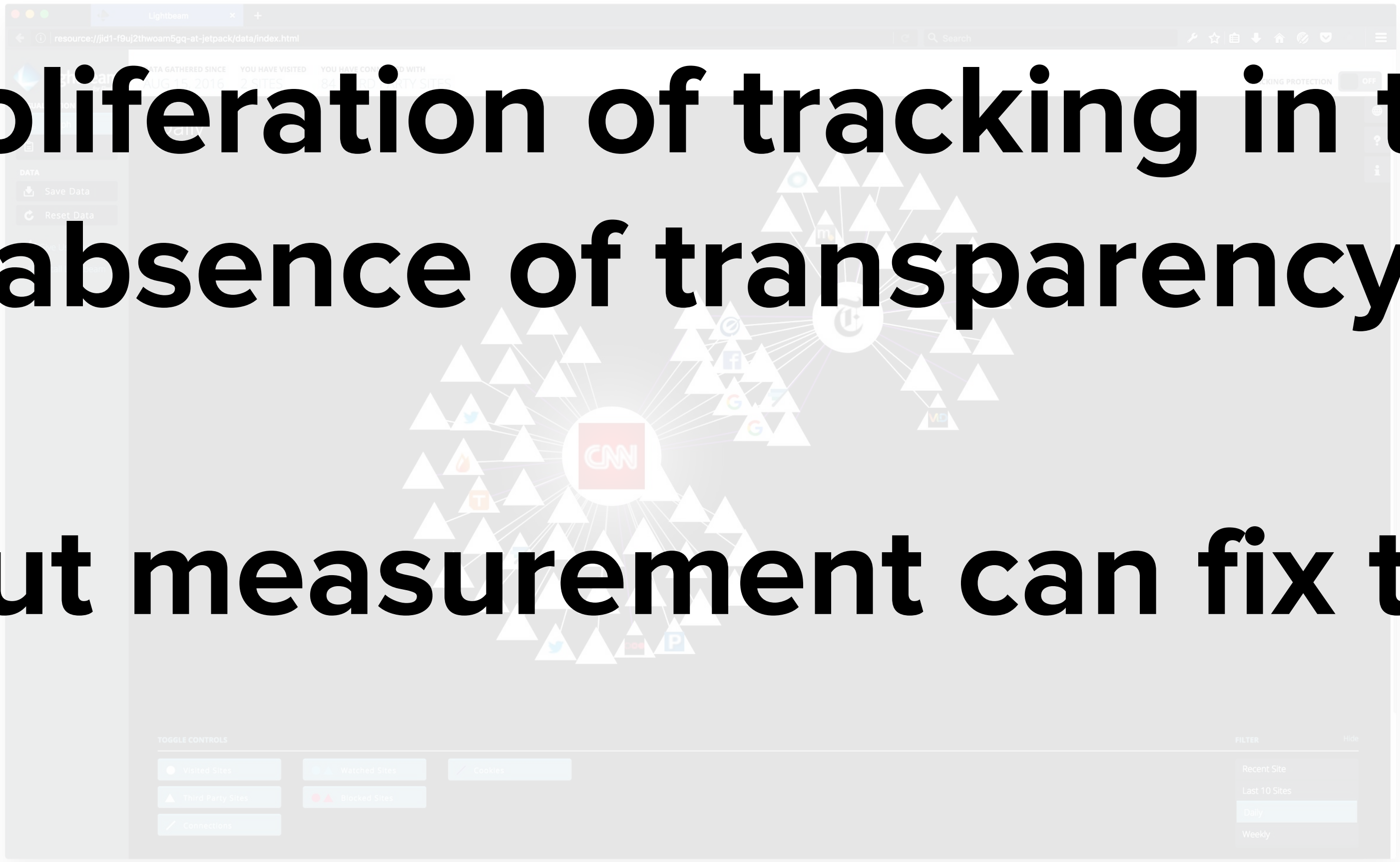


Visiting 2 websites results in 84 third parties contacted



Proliferation of tracking in the absence of transparency

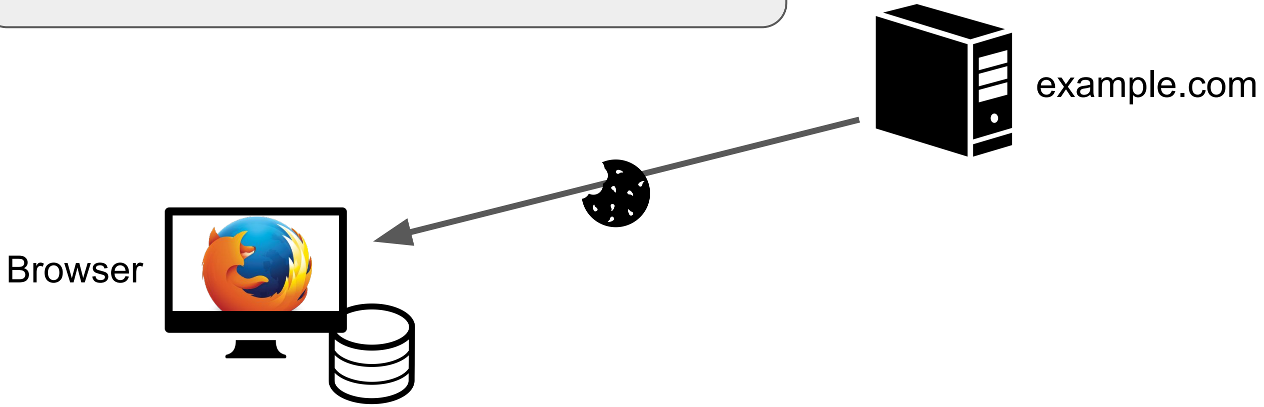
Visiting 2 websites results in 84 third parties contacted



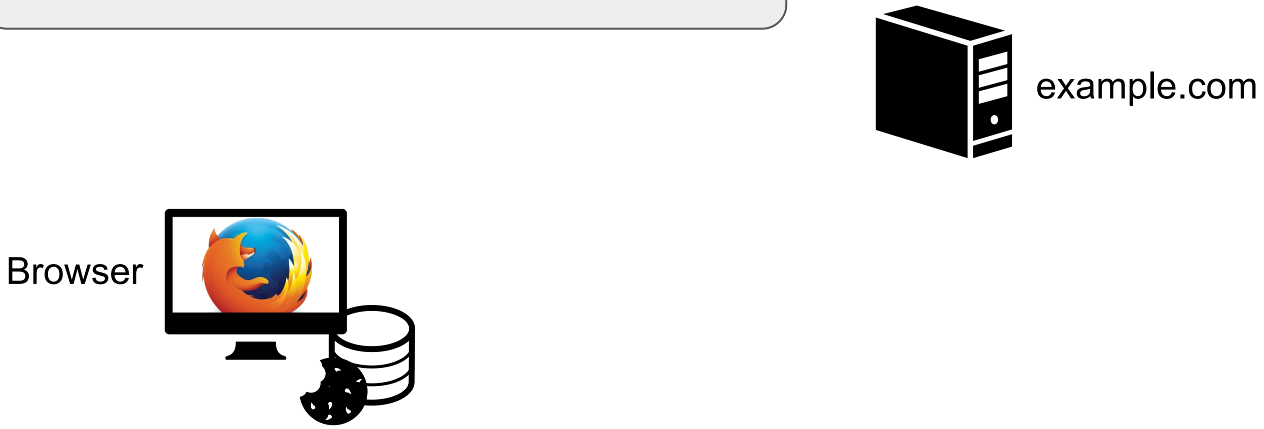
Proliferation of tracking in the absence of transparency

...but measurement can fix that

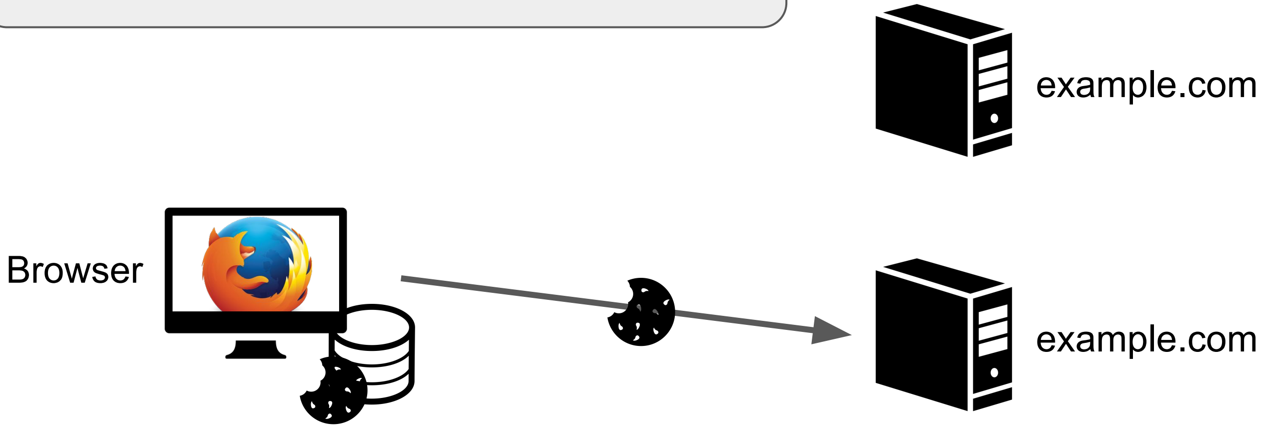
Tracking with browser state



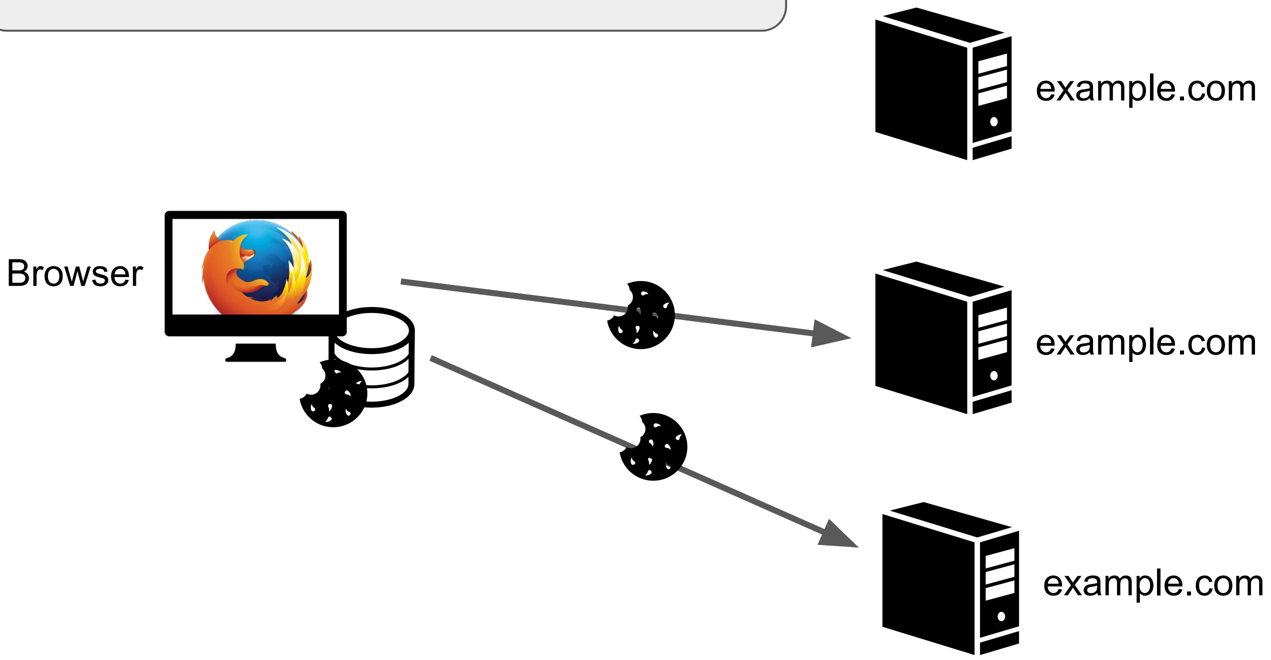
Tracking with browser state



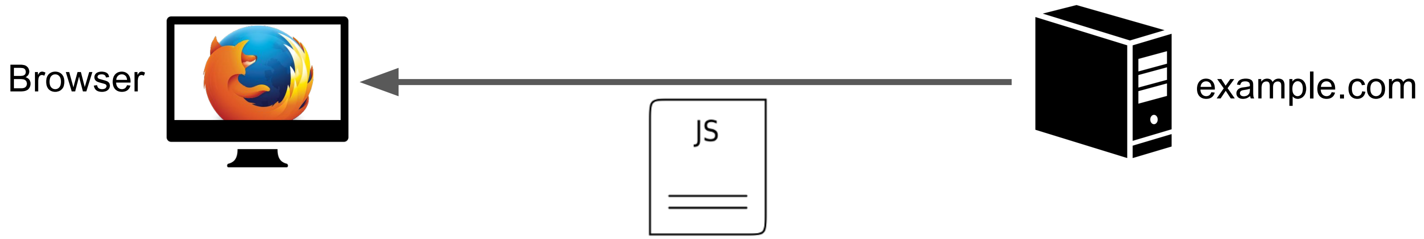
Tracking with browser state



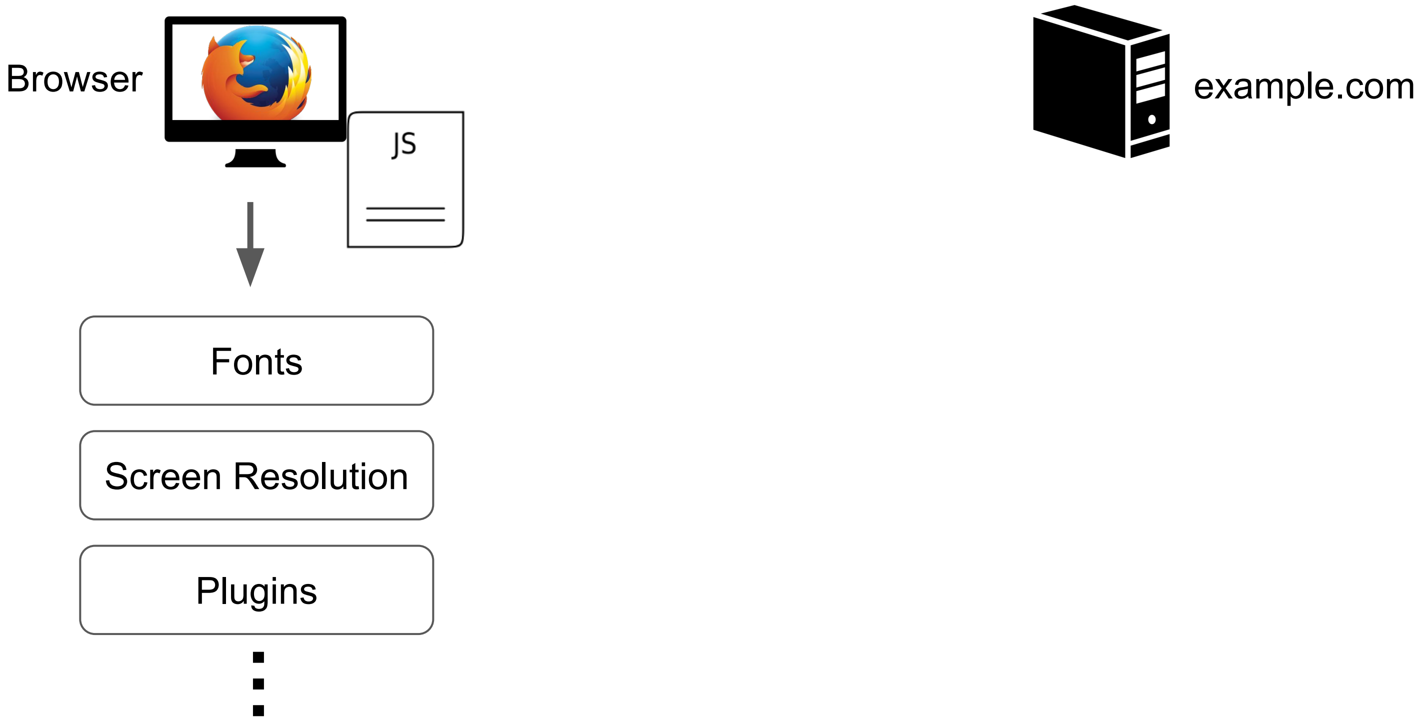
Tracking with browser state



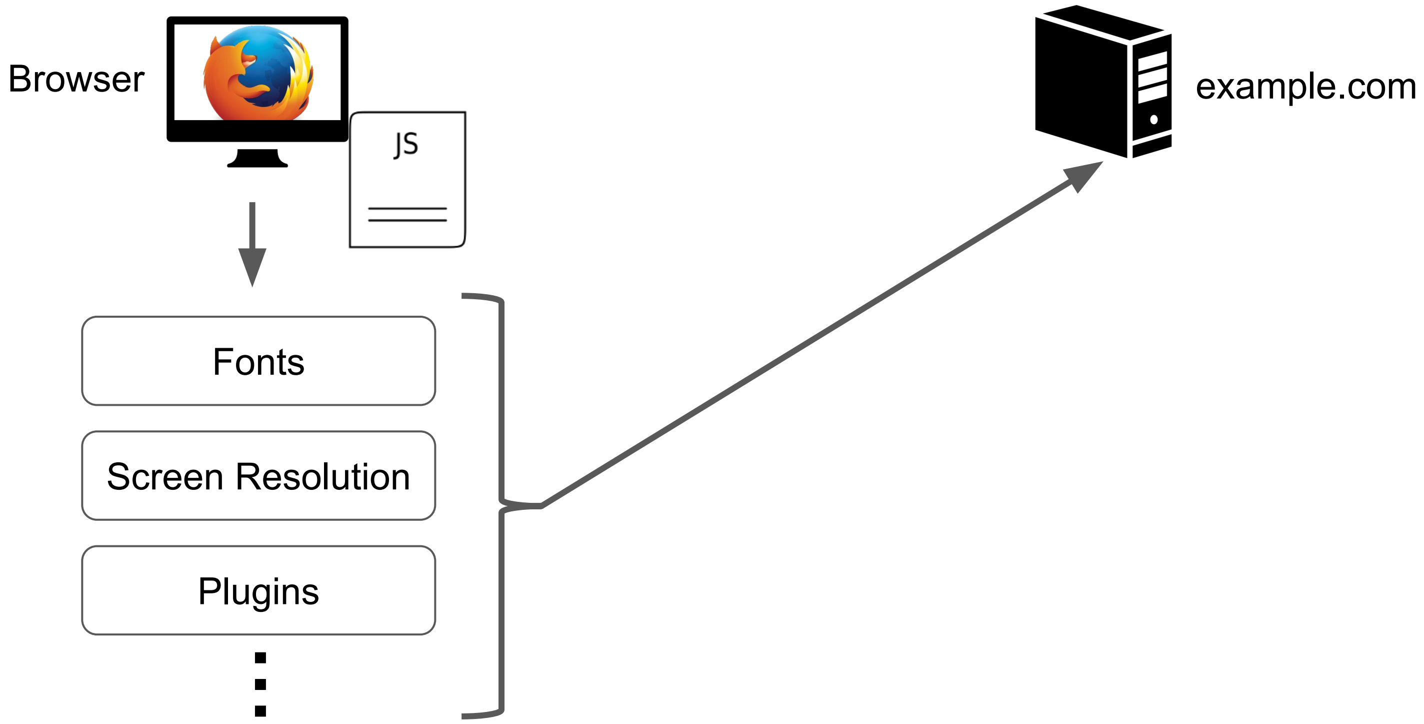
Tracking with fingerprinting



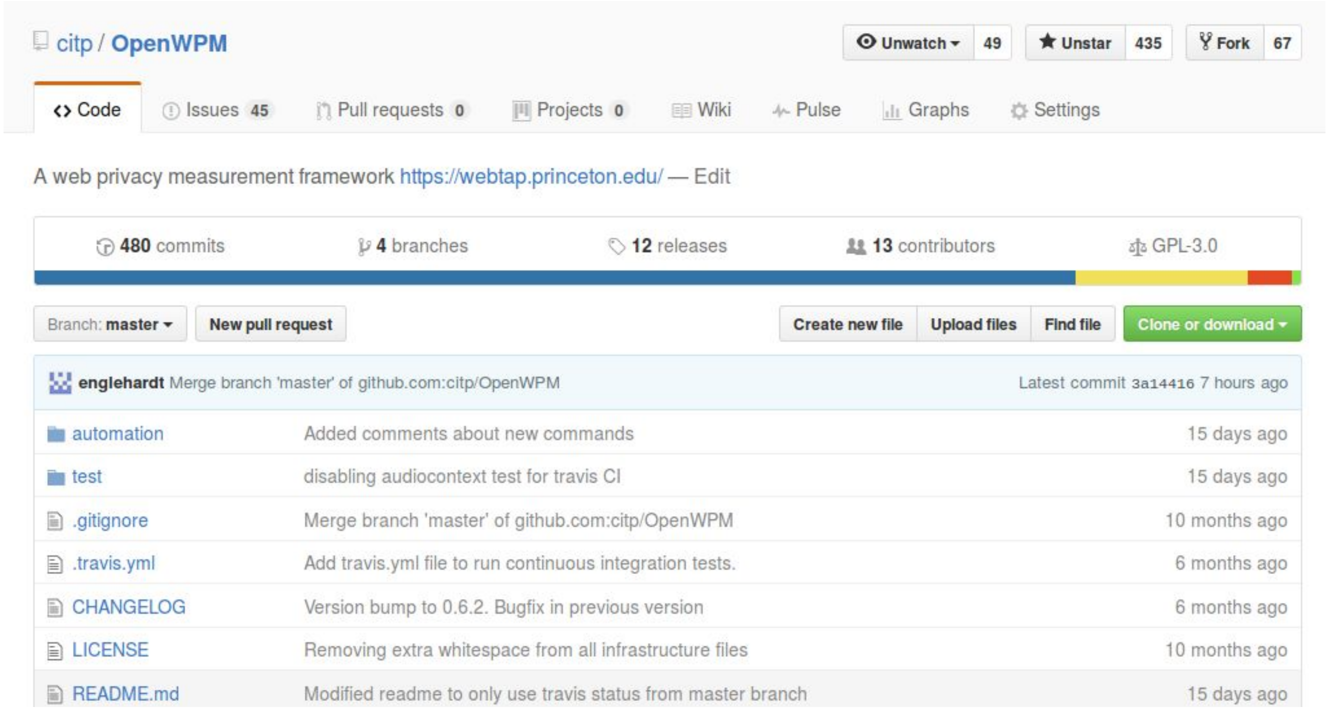
Tracking with fingerprinting



Tracking with fingerprinting



Open Web Privacy Measurement (OpenWPM)



citp / OpenWPM

Unwatch 49 | Unstar 435 | Fork 67

Code | Issues 45 | Pull requests 0 | Projects 0 | Wiki | Pulse | Graphs | Settings

A web privacy measurement framework https://webtap.princeton.edu/ — Edit

480 commits | 4 branches | 12 releases | 13 contributors | GPL-3.0

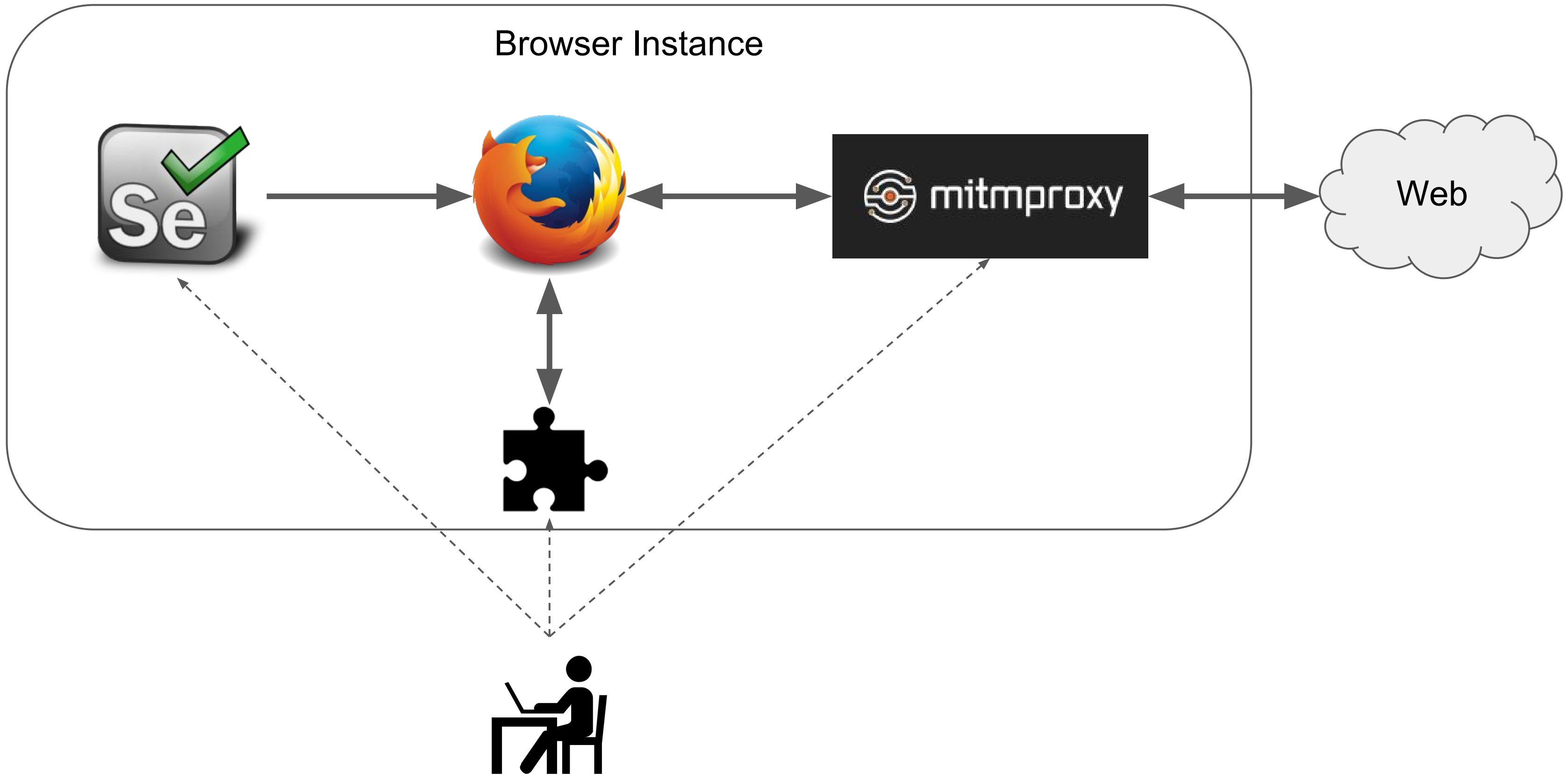
Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

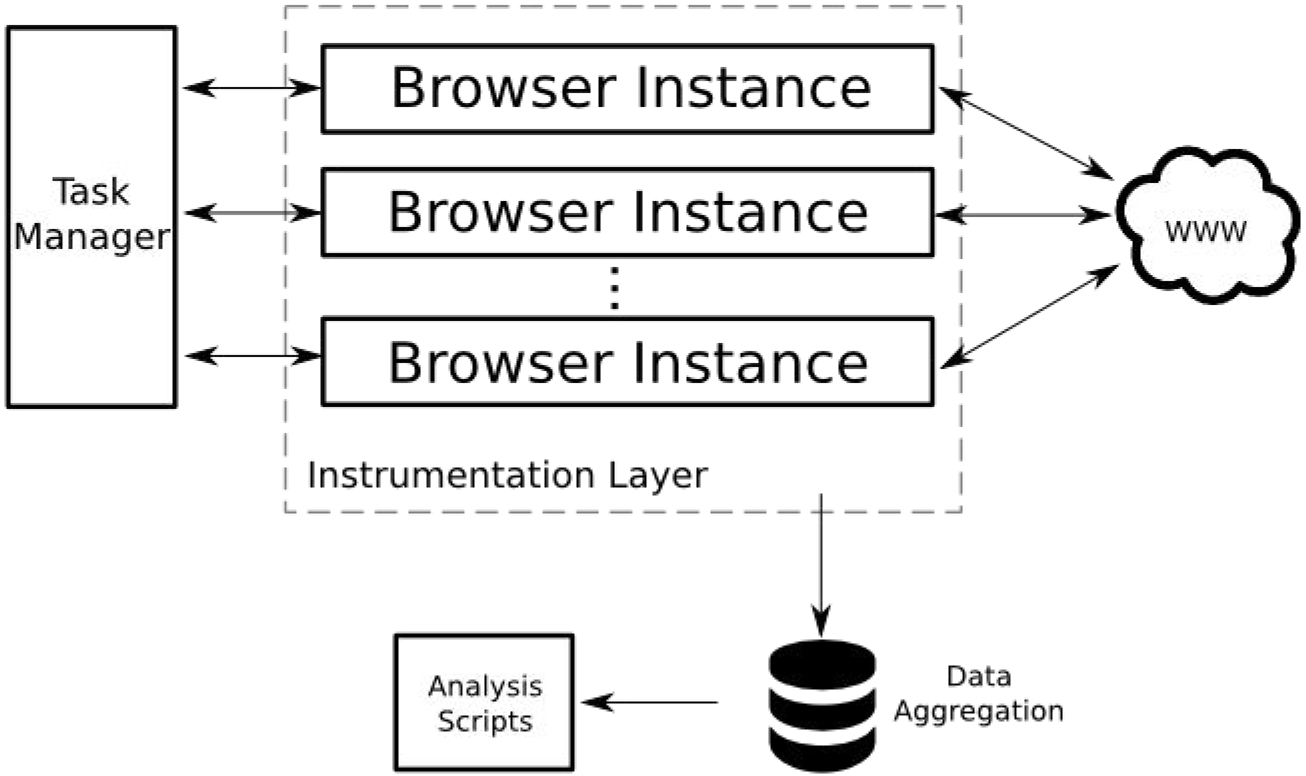
englehardt Merge branch 'master' of github.com:citp/OpenWPM — Latest commit 3a14416 7 hours ago

automation	Added comments about new commands	15 days ago
test	disabling audiocontext test for travis CI	15 days ago
.gitignore	Merge branch 'master' of github.com:citp/OpenWPM	10 months ago
.travis.yml	Add travis.yml file to run continuous integration tests.	6 months ago
CHANGELOG	Version bump to 0.6.2. Bugfix in previous version	6 months ago
LICENSE	Removing extra whitespace from all infrastructure files	10 months ago
README.md	Modified readme to only use travis status from master branch	15 days ago

https://github.com/citp/OpenWPM

OpenWPM




Task Manager
Browser Instance
Browser Instance
Browser Instance
WWW
Instrumentation Layer
Analysis Scripts
Data Aggregation

The Princeton Web Census

Monthly
1 Million Site Crawl

Collecting:

- Javascript Calls
- All javascript files
- HTTP Requests and Responses
- Storage (cookies, Flash, etc)

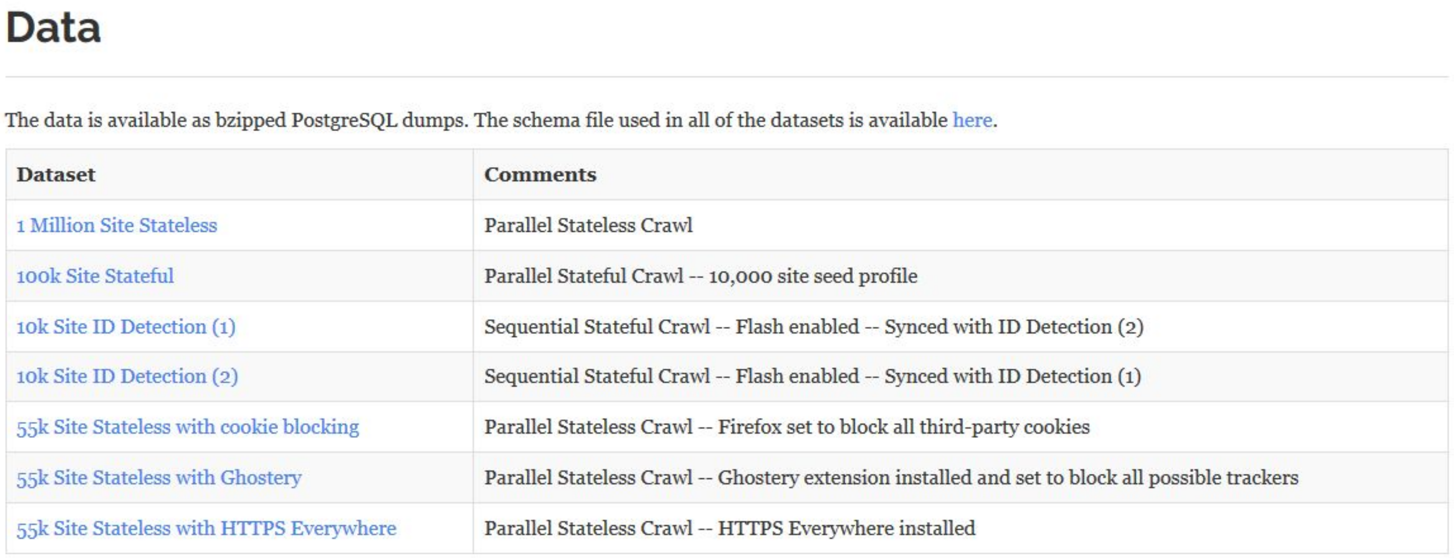
Tackling open questions with OpenWPM

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



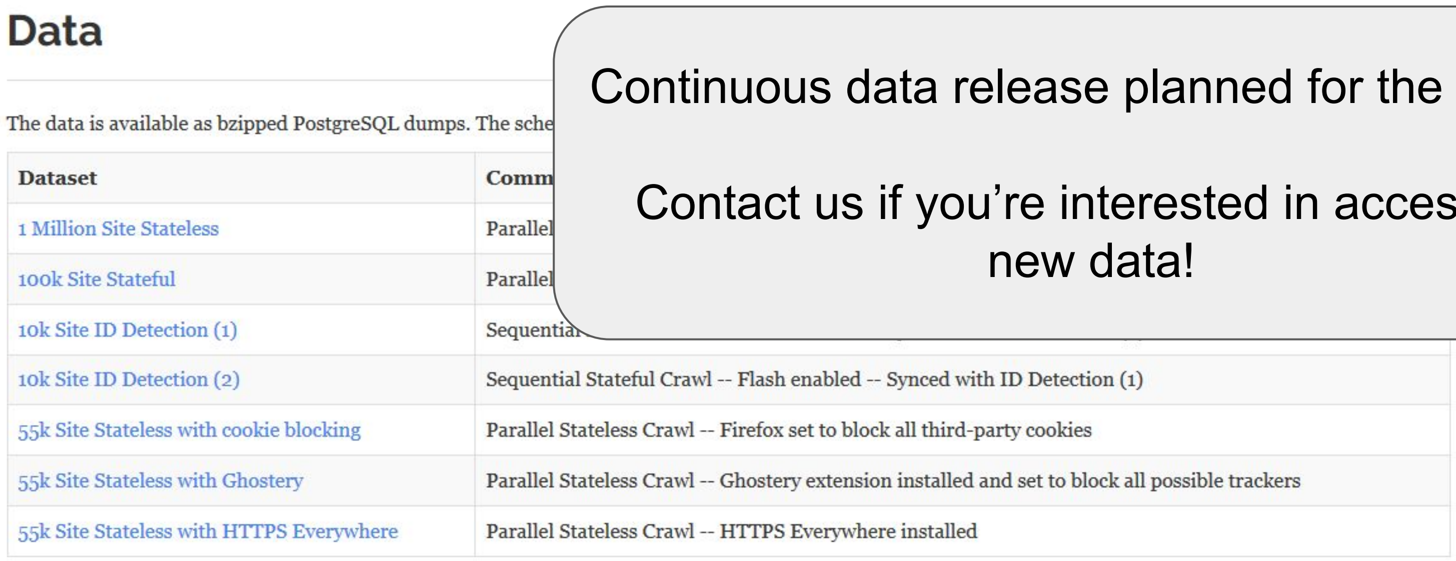
Data

The data is available as bzipped PostgreSQL dumps. The schema file used in all of the datasets is available here.

Dataset	Comments
1 Million Site Stateless	Parallel Stateless Crawl
100k Site Stateful	Parallel Stateful Crawl -- 10,000 site seed profile
10k Site ID Detection (1)	Sequential Stateful Crawl -- Flash enabled -- Synced with ID Detection (2)
10k Site ID Detection (2)	Sequential Stateful Crawl -- Flash enabled -- Synced with ID Detection (1)
55k Site Stateless with cookie blocking	Parallel Stateless Crawl -- Firefox set to block all third-party cookies
55k Site Stateless with Ghostery	Parallel Stateless Crawl -- Ghostery extension installed and set to block all possible trackers
55k Site Stateless with HTTPS Everywhere	Parallel Stateless Crawl -- HTTPS Everywhere installed

https://webtransparency.cs.princeton.edu/webcensus/index.html#data

Download our public postgres dumps



Data

The data is available as bzipped PostgreSQL dumps. The sche

Dataset	Comm
1 Million Site Stateless	Parallel
100k Site Stateful	Parallel
10k Site ID Detection (1)	Sequential
10k Site ID Detection (2)	Sequential Stateful Crawl -- Flash enabled -- Synced with ID Detection (1)
55k Site Stateless with cookie blocking	Parallel Stateless Crawl -- Firefox set to block all third-party cookies
55k Site Stateless with Ghostery	Parallel Stateless Crawl -- Ghostery extension installed and set to block all possible trackers
55k Site Stateless with HTTPS Everywhere	Parallel Stateless Crawl -- HTTPS Everywhere installed

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

Google BigQuery

amazon web services™

DATA TRANSPARENCY LAB

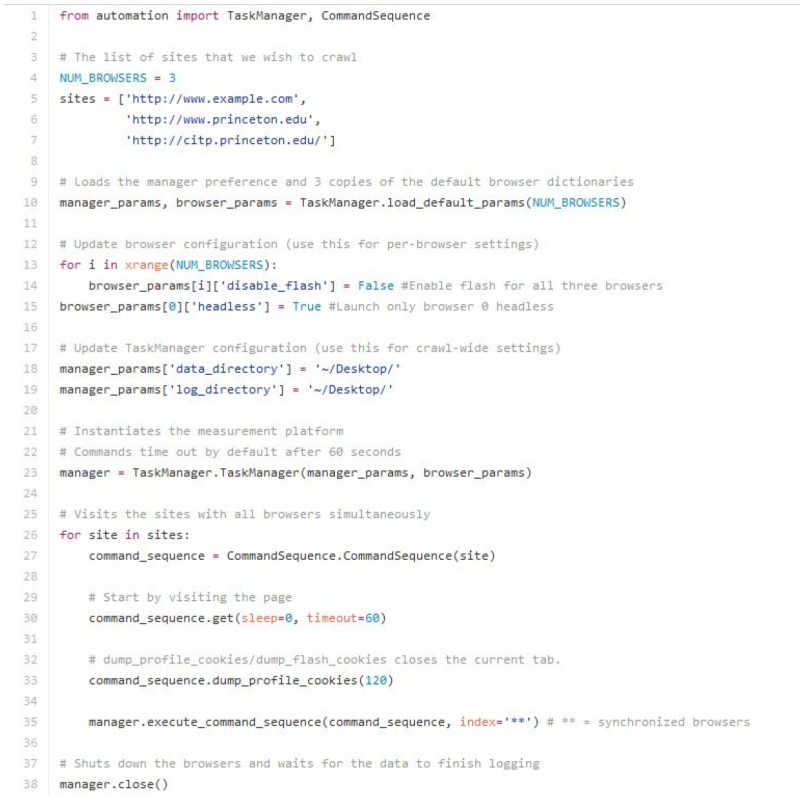
Using OpenWPM and the Princeton Web Census in your research

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

Study using OpenWPM	Conference	Year
The Web Never Forgets: Persistent Tracking Mechanisms in the Wild	**CCS**	**2014**
Cognitive disconnect:Understanding Facebook Connect login permissions	OSN	2014
Cookies that give you away: The surveillance implications of web tracking	**WWW**	**2015**
Upgrading HTTPS in midair: HSTS and key pinning in practice	NDSS	2015
Web Privacy Census	Tech Science	2015
Variations in Tracking in Relation to Geographic Location	W2SP	2015
No Honor Among Thieves: A Large-Scale Analysis of Malicious Web Shells	WWW	2016
Online Tracking: A 1-million-site Measurement and Analysis	**CCS**	**2016**
Dial One for Scam: Analyzing and Detecting Technical Support Scams	NDSS	2017

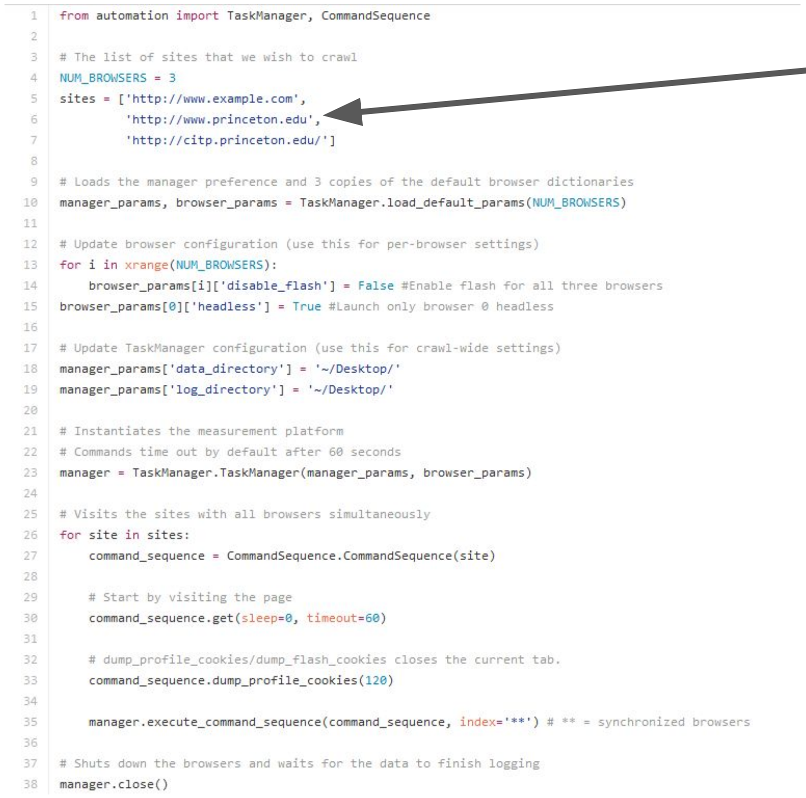
Study using OpenWPM	**Conference**	**Year**
The Web Never Forgets: Persistent Tracking Mechanisms in the Wild	**CCS**	**2014**
Cognitive disconnect:Understanding Facebook Connect login permissions	OSN	2014
Cookies that give you away: The surveillance implications of web tracking	**WWW**	**2015**
Upgrading HTTPS in midair: HSTS and key pinning in practice	NDSS	2015
Web Privacy Census	Tech Science	2015
Variations in Tracking in Relation to Geographic Location	W2SP	2015
No Honor Among Thieves: A Large-Scale Analysis of Malicious Web Shells	WWW	2016
Online Tracking: A 1-million-site Measurement and Analysis	**CCS**	**2016**
Dial One for Scam: Analyzing and Detecting Technical Support Scams	NDSS	2017

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://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 xrange(NUM_BROWSERS):
    browser_params[i]['disable_flash'] = False #Enable flash for all three browsers
browser_params[0]['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=60)

    # 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()
```

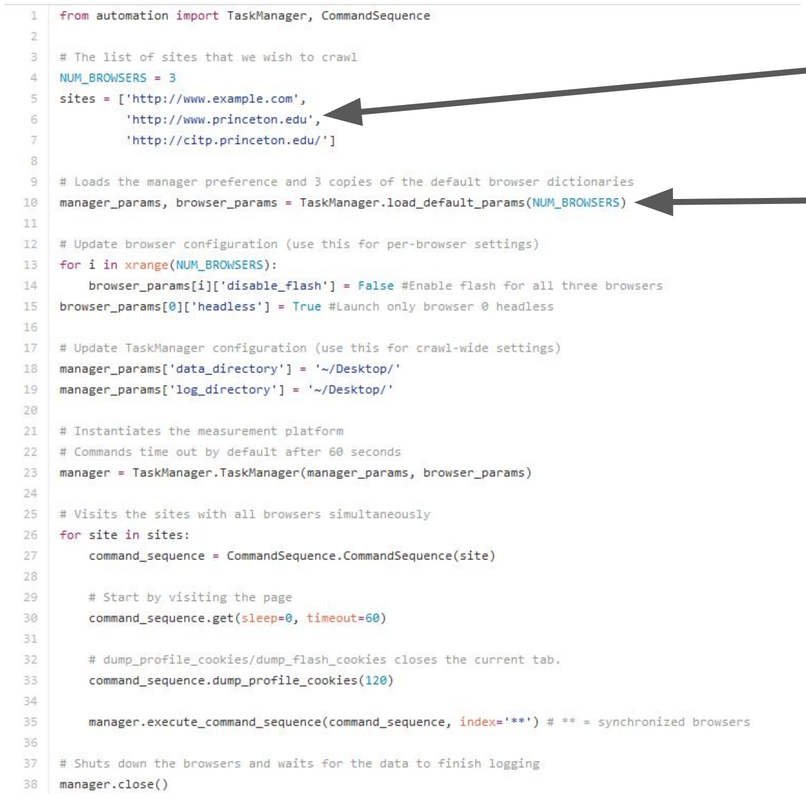
Simple Python interface to run crawls



```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 xrange(NUM_BROWSERS):
    browser_params[i]['disable_flash'] = False #Enable flash for all three browsers
browser_params[0]['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=60)

    # 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

Simple Python interface to run crawls

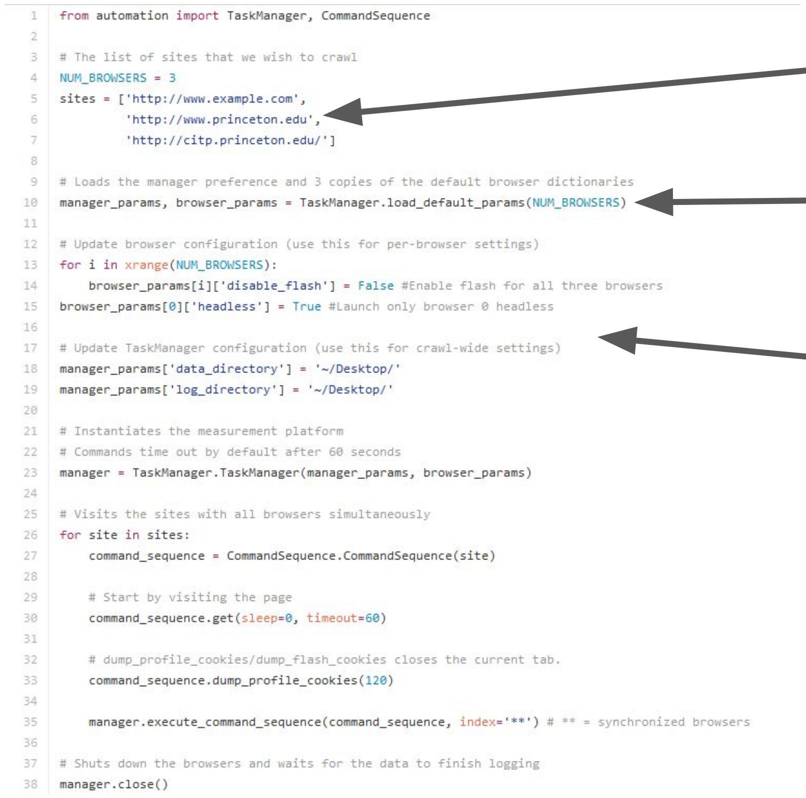


```python
from automation import TaskManager, CommandSequence

# The list of sites that we wish to crawl
NUM_BROWSERS = 3
sites = ['http://www.example.com',
         'http://www.princeton.edu',
         'http://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 xrange(NUM_BROWSERS):
    browser_params[i]['disable_flash'] = False #Enable flash for all three browsers
browser_params[0]['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=60)

    # 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

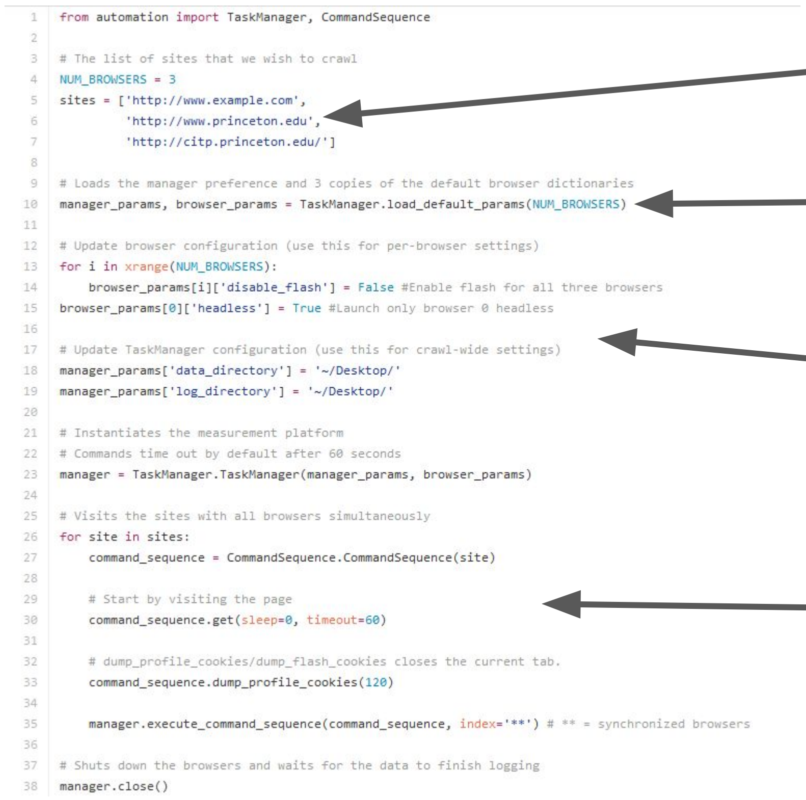
Simple Python interface to run crawls



```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 xrange(NUM_BROWSERS):
    browser_params[i]['disable_flash'] = False #Enable flash for all three browsers
browser_params[0]['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=60)

    # 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
3. Configure instrumentation / platform

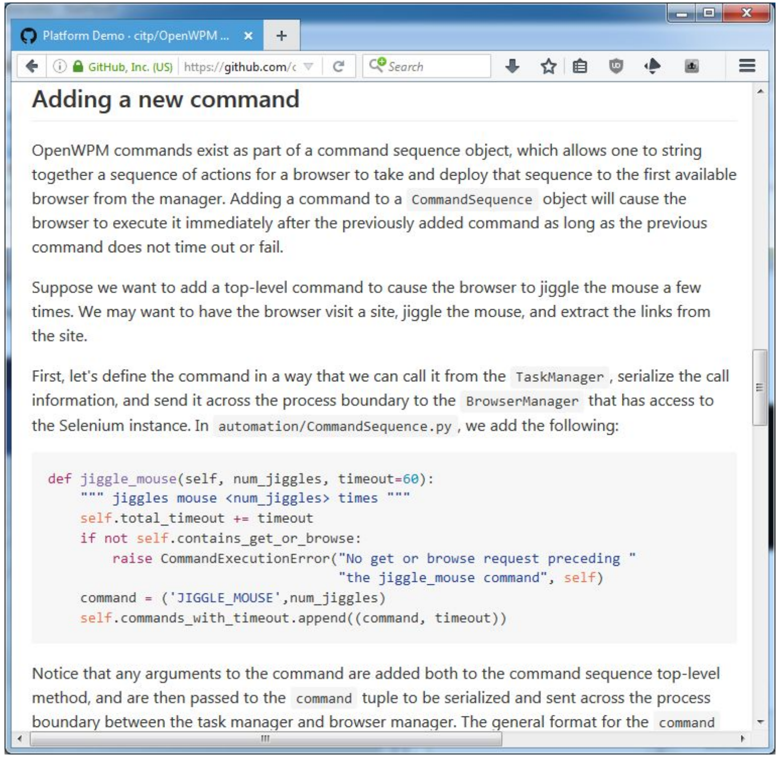
Simple Python interface to run crawls



```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://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 xrange(NUM_BROWSERS):
    browser_params[i]['disable_flash'] = False #Enable flash for all three browsers
browser_params[0]['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=60)

    # 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
3. Configure instrumentation / platform
4. Submit commands during each page visit

Adding new commands is easy


Platform Demo · citp/OpenWPM ...
GitHub, Inc. (US) https://github.com/c
Search

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:

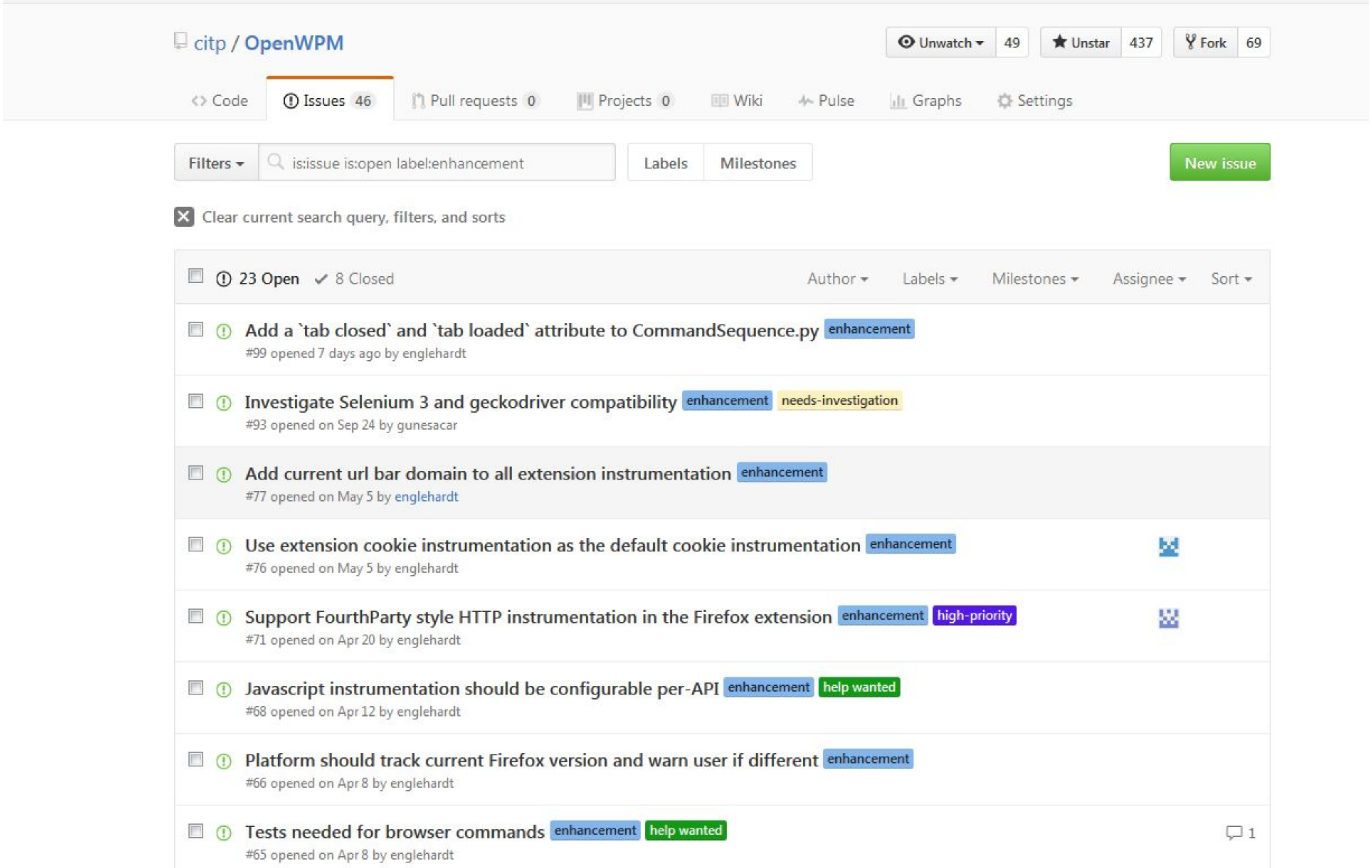
```
def jiggle_mouse(self, num_jiggles, timeout=60):
    """ jiggles mouse <num_jiggles> times """
    self.total_timeout += timeout
    if not self.contains_get_or_browse:
        raise CommandExecutionError("No get or browse request preceding "
                                    "the jiggle_mouse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



is:issue is:open label:enhancement

Clear current search query, filters, and sorts

23 Open ✓ 8 Closed

Add a `tab closed` and `tab loaded` attribute to CommandSequence.py enhancement
#99 opened 7 days ago by englehardt

Investigate Selenium 3 and geckodriver compatibility enhancement needs-investigation
#93 opened on Sep 24 by gunesacar

Add current url bar domain to all extension instrumentation enhancement
#77 opened on May 5 by englehardt

Use extension cookie instrumentation as the default cookie instrumentation enhancement
#76 opened on May 5 by englehardt

Support FourthParty style HTTP instrumentation in the Firefox extension enhancement high-priority
#71 opened on Apr 20 by englehardt

Javascript instrumentation should be configurable per-API enhancement help wanted
#68 opened on Apr 12 by englehardt

Platform should track current Firefox version and warn user if different enhancement
#66 opened on Apr 8 by englehardt

Tests needed for browser commands enhancement help wanted 1
#65 opened on Apr 8 by englehardt

Easy to measure new fingerprinting techniques

Canvas Fingerprinting

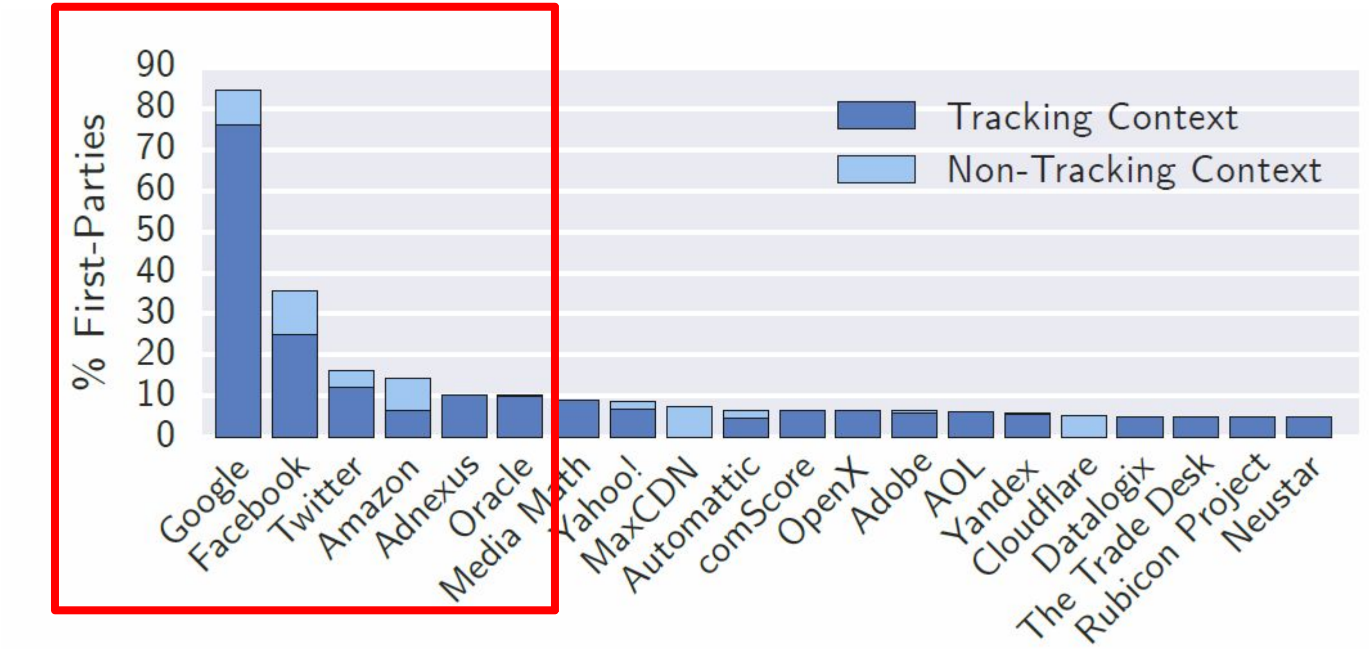
```
// Access to canvas
instrumentObject(
    window.HTMLCanvasElement.prototype,
    "HTMLCanvasElement", true
);

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

```
// Access to webRTC
instrumentObject(
    window.RTCPeerConnection.prototype,
    "RTCPeerConnection", true
);
```

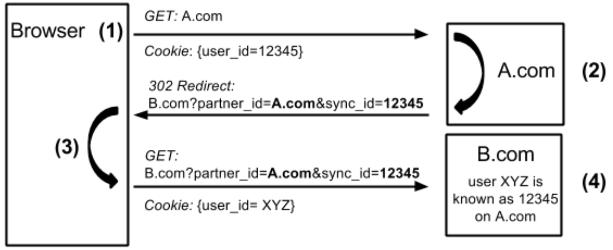
Insights from our own studies using OpenWPM and Princeton Web Census data

Better understand the tracking ecosystem

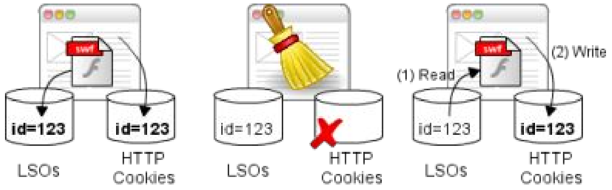


https://webtransparency.cs.princeton.edu/webcensus/

Measure persistent tracking


Browser (1)
GET: A.com
Cookie: {user_id=12345}
A.com (2)
302 Redirect: B.com?partner_id=A.com&sync_id=12345
(3)
GET: B.com?partner_id=A.com&sync_id=12345
Cookie: {user_id= XYZ}
B.com
user XYZ is known as 12345 on A.com
(4)

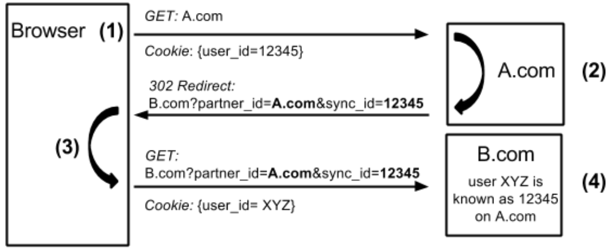
Cookie Syncing



Cookie Respawning

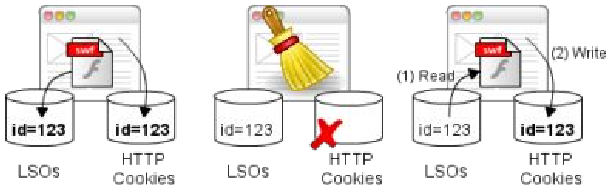
https://webtransparency.cs.princeton.edu/webcensus/

Measure persistent tracking


Browser (1)
GET: A.com
Cookie: {user_id=12345}
A.com
(2)
302 Redirect:
B.com?partner_id=A.com&sync_id=12345
(3)
GET:
B.com?partner_id=A.com&sync_id=12345
Cookie: {user_id= XYZ}
B.com
user XYZ is known as 12345 on A.com
(4)

Cookie Syncing

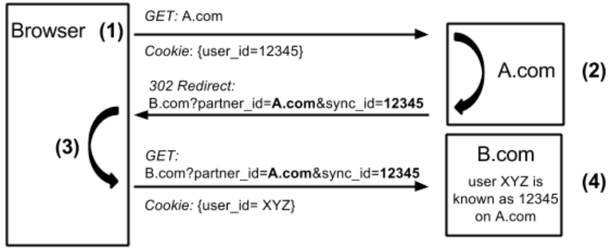
45 of the top 50 third parties



Cookie Respawning

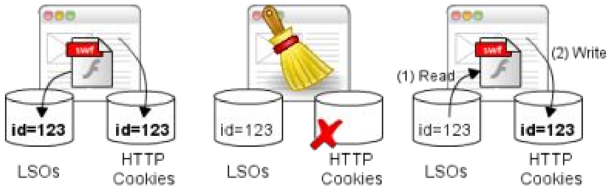
https://webtransparency.cs.princeton.edu/webcensus/

Measure persistent tracking



Browser
(1)
GET: A.com
Cookie: {user_id=12345}
A.com
(2)
302 Redirect:
B.com?partner_id=A.com&sync_id=12345
(3)
GET:
B.com?partner_id=A.com&sync_id=12345
Cookie: {user_id= XYZ}
B.com
user XYZ is known as 12345 on A.com
(4)

Cookie Syncing



Cookie Respawning

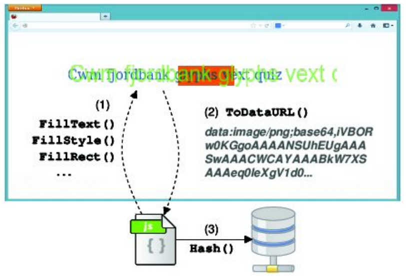
45 of the top 50 third parties

Largely unused by US-based 3rd parties

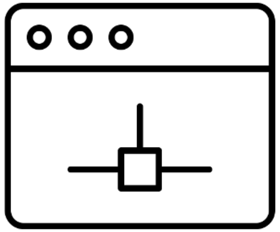
https://webtransparency.cs.princeton.edu/webcensus/

Measure the adoption of fingerprinting techniques

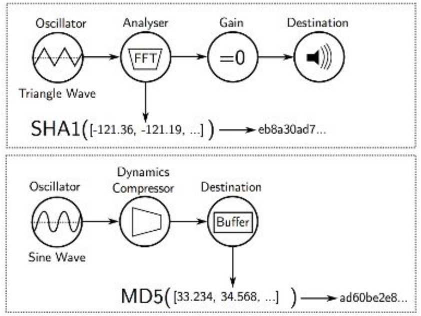
Canvas



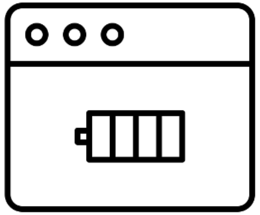
WebRTC



Audio



Battery

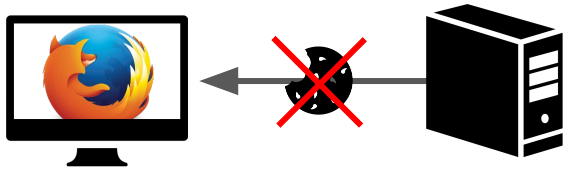


https://webtransparency.cs.princeton.edu/webcensus/

Test the effectiveness of Privacy Tools

Ghostery

AdBlock Plus



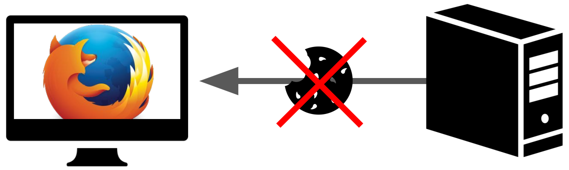
Third-party cookie blocking

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