Oblique: Accelerating Page Loads Using Symbolic Execution

Ronny Ko, James Mickens
Harvard University

Blake Loring
Royal Holloway
University of London

Ravi Netravali
UCLA
Two Problematic Trends in Web Traffic

● Over half of web traffic is from mobile devices
  ○ Many mobile users (particularly in emerging markets) stuck behind high-latency 3G/4G links
  ○ Even 5G links often suffer from 4G latencies
  ○ Latency, not bandwidth, often determines page load times!

● Over 90% of web traffic is HTTPS, not HTTP
  ○ The crypto is cheap . . .
  ○ . . . but how can we analyze encrypted traffic without breaking confidentiality and integrity?
Existing Web Accelerators (I)

Enables outsourcing of web acceleration

Breaks end-to-end TLS security: cleartext user data (e.g., cookies and User-Agent string) are exposed to third party

Remote Dependency Resolution (e.g., Amazon Silk, Parcel)
Existing Web Accelerators (II)

- Doesn’t expose cleartext HTTPS data to third parties
- Analysis must be run by the first party: outsourcing it would break TLS security

Vroom (SIGCOMM 2017)

First-party offline analysis server

High-latency link

User phone

First-party web server

[Server uses <link> prefetch and HTTP2 push to pre-warm client cache]

[Loads page multiple times, identifies the stable set URLs]
Oblique: Acceleration + Privacy

- Have an offline third-party server load a web page symbolically
  - The symbols are sensitive user values like cookies and User-Agent strings
  - Output of analysis is a list of symbolic URLs fetched by the page
- Have the user’s browser resolve symbolic URLs and prefetch them
- User-specific data is never revealed to the third party!
Example of a Path Constraint (PC) Tree

```
https_req::UserAgent

== “Mobile Chrome”

https_req::cookie[“darkMode”]

== (“no” || “”)

== “yes”

light-mode.css

gui.js

default.html

dark-mode.css

gui.js

{{https_req::cookie[“uid”]}}.html
```
## Client-side Symbols

<table>
<thead>
<tr>
<th>Input name</th>
<th>HTTP header</th>
<th>JavaScript variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Agent</td>
<td>User-Agent</td>
<td>navigator.userAgent</td>
<td>The local browser type, e.g., &quot;Mozilla/5.0 (Windows; U: Win98; en-US; rv:0.9.2) Gecko/20010725 Netscape6/6.1&quot;</td>
</tr>
<tr>
<td>Platform</td>
<td>Included in User-Agent</td>
<td>navigator.platform</td>
<td>The local OS, e.g., &quot;Win64&quot;</td>
</tr>
<tr>
<td>Screen characteristics</td>
<td>N/A</td>
<td>window.screen.*</td>
<td>Information about the local display, e.g., the dimensions and pixel depth</td>
</tr>
<tr>
<td>Host</td>
<td>Host</td>
<td>location.host</td>
<td>Specifies the virtual host and port number to use</td>
</tr>
<tr>
<td>Referer</td>
<td>Referer</td>
<td>document.referrer</td>
<td>The URL of the page whose link was followed to generate a request for the current page</td>
</tr>
<tr>
<td>Origin</td>
<td>Origin</td>
<td>location.origin</td>
<td>Like Referer, but only includes the origin part of the referring URL</td>
</tr>
<tr>
<td>Last modified</td>
<td>Last-Modified (response)</td>
<td>document.lastModified</td>
<td>Set by the server to indicate the last modification date for the returned resource</td>
</tr>
<tr>
<td>Cookie</td>
<td>Cookie (request), Set-Cookie (response)</td>
<td>document.cookie</td>
<td>A text string containing &quot;key=value&quot; pairs</td>
</tr>
</tbody>
</table>
1. Load the page
2. Path constraint tree
3. Page + prefetch library
4. Page request
5. Page with path constraint tree & prefetch library
6. Prefetch URLs

Overview

Prefetch library
Oblique’s Offline Analysis

● High-level goals:
  ○ Explore all of the execution paths that a page’s JavaScript might take, given all possible values for client-specific state like cookies
  ○ For each path, identify the URLs that the page fetches
● To find these paths, Oblique uses concolic execution
Symbolic Analysis (Concolic Execution)

1. Distributor generates initial concrete values for client symbols (e.g., Cookie = "cat=yes", User-Agent = "MobileChrome").
2. Executor launches a web browser.
4. Browser fetches more concrete URLs
   a. CSS and images handled as normal
   b. JavaScript executes code **concolically** (i.e., concretely + symbolically)

As JavaScript executes on concrete data, Oblique tracks symbolic path constraints and symbolic URLs!
Symbolic Analysis (Concolic Execution)

1. Distributor generates initial concrete values for client symbols (e.g., Cookie="cat=yes", User-Agent="MobileChrome").
2. Executor launches a web browser
3. Browser fetches concrete page HTML from the first-party web server
4. Browser fetches more concrete URLs
   a. CSS and images handled as normal
   b. JavaScript evaluated using symbolic execution

As JavaScript executes on concrete data, Oblique tracks symbolic path constraints and symbolic URLs!

[Example of Concolically Executed JavaScript Code]
```javascript
var baseUrl = "foo.com/";
var rndId = Math.random().toString();
if(document.cookie.indexOf("cat")==0){
  fetch(baseUrl + rndId + "/cat.jpg");
} else{
  fetch(baseUrl + "/dog.jpg");
}
```

Concrete URL: foo.com/0.3274/cat.jpg
Symbolic URL: foo.com/{rnd0}/cat.jpg
Symbolic path constraint:
document.cookie == "cat"{.*}
5. Once the page load finishes, the symbolic path constraint, e.g. `document.cookie = "cat"{{.*}}`, is sent to executor

6. Executor asks the SMT solver to invert part of the path constraint, e.g., `document.cookie = ^("cat"{{.*}})``

7. Solver finds a concrete input that satisfies the inversion, e.g., `document.cookie = "x81b5"`

8. Executor returns the new test input to the distributor, who inserts the input into a priority queue

Repeats Steps 1 - 8
```javascript
var baseUrl = "foo.com/";
var rndId = Math.random().toString();
if(document.cookie.indexOf("cat")===0){
    fetch(baseUrl + rndId + "/cat.jpg");
}else{
    fetch(baseUrl + "/dog.jpg");
}
```
Limitations of Oblique

- Only certain native methods are modelled
  - For example, Oblique has a Z3 expression model for `String.charAt()`, but not `Intl.DateTimeFormat()`
  - If the JavaScript string variable `s` contains symbolic data derived from `User-Agent`, then the return value of `String.charAt(s)` will properly capture that symbolic data
  - In contrast, Oblique always treats `Intl.DateTimeFormat(s)` as fully concrete, possibly hurting path coverage
- Oblique’s symbolic analysis may time out, hurting path coverage
- Oblique’s symbolic analysis can’t issue HTTP requests that are nonidempotent
Evaluation Setup

- An HTTP record-and-replay tool recorded content from 200 popular pages
- Digital Ocean VM ran:
  - Oblique web server
  - Vroom server
  - RDR server
- Client device was a Galaxy S10e phone running Chromium v78
  - End-to-end RTT b/w phone and Digital Ocean VM was ~47ms
  - We used `netem` to inject added latency in some experiments
Results: Speed Index

![Speed Index Graph 47 ms RTT](image1)

![Speed Index Graph 150 ms RTT](image2)
Results: Prefetch Hit Rate

(a) Prefetch hit rate (static+dynamic URLs)

(b) Prefetch hit rate (only dynamic URLs)
Conclusion

- Prefetching helps to reduce page load times
- Prior systems generate URL prefetch lists by:
  - Breaking TLS integrity, or
  - Running first-party analyses that cannot be outsourced
- Oblique uses symbolic execution to eliminate the design tension
  - Oblique’s third-party server can model user-specific data as symbols
  - Symbols are only resolved by clients!
- Oblique reduces page loads by up to 31%, outperforming Vroom and RDR by up to 17%
THANK YOU

Ronny Ko
Harvard University
hrko@g.harvard.edu