Enabling Reconstruction of Attacks on Users via Efficient Browsing Snapshots

Phani Vadrevu*, Jienan Liu*, Bo Li*, Babak Rahbarinia+, Kyu Hung Lee* and Roberto Perdisci*

* University of Georgia, Athens, USA
+ Auburn University in Montgomery, Alabama, USA
Case Study
Case Study

Social Engineering Attack!
Case Study

Internet

Dark Web

Thu
19

Forensic Analyst
We need ...

Helps in user training to improve awareness
Requirements

A tool that can record and reconstruct user-browser interactions and browser state.

1. **Forensic Rigor**
   - Browser state should be fully captured *synchronously* i.e. before input is processed by the browser

2. **Efficiency (always-on)**
   - HCI research states that a lag < **150 ms** is practically unnoticeable to end users[^1]

3. **Transparency**
   - Should not be easily detected by adversaries

4. **Portability**
   - Should work on all platforms (mobile)

[^1]: Tolia et al. “Quantifying Interactive User Experience on Thin Clients”
http://isr.cmu.edu/doc/tolia06-ieee.pdf
Related Work

• Network:
  – **WebWitness** (USENIX SEC 2014): DPI to reconstruct path to attack pages; **visual reconstruction not possible**

• Browser based record-and-replay:
  – **WebCapsule** (CCS 2015): Instrument Blink to record and replay all browser actions; **not fully deterministic and is complex**

• Whole system record-and-replay:
  – **ReVirt** (OSDI 2002): Record and guest OS’s execution at an instruction level; **heavy-weight and difficult to deploy on mobile devices**
ChromePic

- An always-on, lightweight, efficient and portable forensic engine embedded inside the Chromium browser

- It synchronously records user-browser interactions and the browser state into rich forensic logs called *webshots*
Webshot

User Input, Timestamp

- Esc
- C

Screenshot

DOM Snapshots (for all frames)
Trigger Events
Trigger Events

**Mouse:** left click, right click
**Touch device:** tap
**Keyboard:** return, space, tab, esc, back space, arrows
ChromePic in Action
Building ChromePic

Extensions not viable

Browser Instrumentation
Chromium Architecture

Browser process

Browser UI Thread
Browser IO Thread
Browser File Thread

Renderers

facebook.com
Renderer Main Thread
Renderer Render Thread

google.com
Renderer Main Thread
Renderer Render Thread

yahoo.com
Renderer Main Thread
Renderer Render Thread

Renderers
Chromium Architecture

Browser process

- User
- Browser UI Thread
- Browser IO Thread
- Browser File Thread

- User
  - input
  - Send(input)
  - IPCSend(input)
  - Notify(input)

Renderers

facebook.com
- Renderer Main Thread
- Renderer Render Thread

google.com
- Renderer Main Thread
- Renderer Render Thread

yahoo.com
- Renderer Main Thread
- Renderer Render Thread

Input Processing
Browser Instrumentation

Use as much underlying code as possible:

- `CaptureVisibleTab()`: asynchronous screenshots
- `SaveAsMHTML()`: asynchronous DOM snapshots
Browser Instrumentation

Use as much underlying code as possible:

- `CaptureVisibleTab()`: asynchronous screenshots
- `SaveAsMHTML()`: asynchronous DOM snapshots

both are **asynchronous**
need synchronous and efficient versions
ChromePic Design

ChromePic Trigger Input Processing

- **User** initiates input.
- Input is sent to the **Browser UI Thread**.
- User triggers tab screenshot.
- Browser UI Thread sends screenshot to the **Browser IO Thread**.
- Browser IO Thread sends screenshot to the **Renderer Main Thread**.
- Renderer Main Thread notifies the **Renderer Render Thread**.
- Renderer Render Thread takes DOM snapshot and saves it.
- Regular input processing continues.
- Webshot overhead is considered.

**Key Processes**:
- TakeScreenshot
- Notify
- IPCSend
ChromePic Design

ChromePic Trigger Input Processing

Synchronous by design
ChromePic Design

• Next efficiency needs to be ensured for both:
  – Screenshots
  – DOM Snapshots
Taking screenshots

User input

Browser UI Thread

TakeScreenshot()

Browser IO Thread

IPC message to Renderer

Browser File Thread

Save

Send(screen_taken)
Taking screenshots

- Not very efficient
- Worst case of ~1s on Android
Taking screenshots

User input

Browser UI Thread
- TakeScreenshot()
- CopyFromCompositingSurface()
- RequestCopyOfOutput()
- setNeedsCommit

GL / GPU
- DrawFrame()
- GetFrameBufferPixels()

Browser IO Thread
- IPC message to Renderer

Browser File Thread
- Save
- CopyFromCompositingSurfaceFinished
- CropScaleReadBack
- PrepareTextureCopyOutputResult()

Send(screen_taken)
Taking screenshots

User input

Browser UI Thread

TakeScreenshot()
CopyFromCompositingSurface()
RequestCopyOfOutput()
setNeedsCommmit

DrawFrame()
GetFrameBufferPixels()

Browser FILE Thread

Save

CopyFromCompositingSurfaceFinished

Result

PrepareTextureCopyOutputResult()
CropScaleReadBack

Send(screen_taken)

GL / GPU

Browser IO Thread

IPC message to Renderer
Efficient screenshots

**User input**

- Browser UI Thread
  - TakeScreenshot()
  - CopyFromCompositingSurface()
  - RequestCopyOfOutput()
  - setNeedsCommit

- Browser File Thread
  - Save
  - PrepareTextureCopyOutputResult()
  - CropScaleReadBack
  - CopyFromCompositingSurfaceFinished

- GL / GPU
  - DrawFrame()
  - GetFrameBufferPixels()
  - Result

- Browser IO Thread
  - IPC message to Renderer
    - Send(screen_taken)

**IPC message**
Efficient screenshots

- Target draw rate is about ~16ms!
- 16 ms << 1 s
Chromium MHTML Code

Frame by frame
Chromium MHTML Code

"the render thread is a scary place"
"on ARM, stalls can be seconds long"
- Chromium Docs
Efficient DOM Snapshots

- Process all frame in a single task.
- Piggyback on input processing task.
DOM Snapshots: Comparison

Original MHTML Code

ChromePic DOM Snapshots

< 150 ms
Evaluation

• Reconstructing attacks on users
• Used ChromePic on various UI attack pages
  1. An in-the-wild social engineering attack
  2. Real-world phishing pages
  3. Clickjacking attacks from WOOT ’14 [2]

Social engineering attack

Alert Box (from DOM Snapshot)

WARNING! This Google Pixel C is infected with viruses and your browser is seriously damaged. You need to remove viruses and make corrections immediately. It is necessary to remove and fix now. Don't close this window. ** If you leave, you will be at risk**
**User Study**

- **Measure performance on real user behavior**
- **15 minutes limit for each user/device**

<table>
<thead>
<tr>
<th></th>
<th>ubuntu</th>
<th>ubuntu</th>
<th>Android</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td># Users</td>
<td>11</td>
<td>15</td>
<td>16</td>
<td>22 (unique)</td>
</tr>
<tr>
<td>Browsing Time (minutes)</td>
<td>286</td>
<td>346</td>
<td>363</td>
<td>995</td>
</tr>
<tr>
<td># Domains</td>
<td>65</td>
<td>80</td>
<td>92</td>
<td>204 (unique)</td>
</tr>
<tr>
<td># Webshots</td>
<td>1376</td>
<td>2145</td>
<td>2428</td>
<td>5949</td>
</tr>
</tbody>
</table>
Screenshot Overhead

Original Screenshot Code

ChromePic - Optimized

<table>
<thead>
<tr>
<th>Platform</th>
<th>Median(ms)</th>
<th>98% (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tablet</td>
<td>65.7</td>
<td>110</td>
</tr>
<tr>
<td>Laptop</td>
<td>36.2</td>
<td>71</td>
</tr>
<tr>
<td>Desktop</td>
<td>39</td>
<td>118</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Platform</th>
<th>Median(ms)</th>
<th>98% (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tablet</td>
<td>13</td>
<td>25.9</td>
</tr>
<tr>
<td>Laptop</td>
<td>5.38</td>
<td>27.7</td>
</tr>
<tr>
<td>Desktop</td>
<td>2.7</td>
<td>23.8</td>
</tr>
</tbody>
</table>
Total Webshot Overhead

### Screenshot Overhead

<table>
<thead>
<tr>
<th>Device</th>
<th>Median (ms)</th>
<th>98% (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tablet</td>
<td>13</td>
<td>25.9</td>
</tr>
<tr>
<td>Laptop</td>
<td>5.38</td>
<td>27.7</td>
</tr>
<tr>
<td>Desktop</td>
<td>2.7</td>
<td>23.8</td>
</tr>
</tbody>
</table>

### DOM Snapshot Overhead

<table>
<thead>
<tr>
<th>Device</th>
<th>Median (ms)</th>
<th>98% (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tablet</td>
<td>59.5</td>
<td>203</td>
</tr>
<tr>
<td>Laptop</td>
<td>33.3</td>
<td>109.6</td>
</tr>
<tr>
<td>Desktop</td>
<td>19</td>
<td>76.1</td>
</tr>
</tbody>
</table>
## Total Webshot Overhead

### Screenshot Overhead

<table>
<thead>
<tr>
<th></th>
<th>Median (ms)</th>
<th>98% (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tablet</td>
<td>13</td>
<td>25.9</td>
</tr>
<tr>
<td>Laptop</td>
<td>5.38</td>
<td>27.7</td>
</tr>
<tr>
<td>Desktop</td>
<td>2.7</td>
<td>23.8</td>
</tr>
</tbody>
</table>

### DOM Snapshot Overhead

<table>
<thead>
<tr>
<th></th>
<th>Median (ms)</th>
<th>98% (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tablet</td>
<td>59.5</td>
<td>203</td>
</tr>
<tr>
<td>Laptop</td>
<td>33.3</td>
<td>109.6</td>
</tr>
<tr>
<td>Desktop</td>
<td>19</td>
<td>76.1</td>
</tr>
</tbody>
</table>

89% < 150 ms

< 150 ms
Performance Overhead

Event Duration (ms) - mouse, key press deltas and webshot overheads

- Mouse click deltas (1278)
- Key press deltas (1089)
- Laptop webshots (2117)
- Tablet webshots (2428)
- Desktop webshots (1361)
Storage

Storage requirements in **MB/Minute**

- Maximum requirement of about **1.03 MB/minute** of active browsing
- At this rate, a **1000 employee** corporate network would generate **72 TB** of log data per year
Discussion on Privacy

• **Disable on HTTPS** connections using valid SSL certificates

• **Whitelist sensitive** websites

• Site-based encryption scheme based on a key-escrow agent.
  – Each site’s data is encrypted with a separate key
  – When an incident happens, the investigator gets only keys to the relevant sites.
  – **Forward secure encryption schemes** can be used to extend this for devices that are not always connected to the key escrow agent
Conclusion

- ChromePic is a **lightweight and portable forensic engine**.
- It can accurately log important user inputs and the associated browser states.
- ChromePic can help **reconstruct real world UI attacks**.
- ChromePic has **imperceptible latency** and requires only **moderate disk space** for logs.
Thank You!

Source code to be released soon!
Binaries already available!!

https://github.com/chromepic/chromepic-browser