CETERUM CENSEO :VISITED ESSE DELENDAM
The two classic web information leaks

### Cache detection

**Timing Attacks on Web Privacy**

Edward W. Felten and Michael A. Schneider  
Secure Internet Programming Laboratory  
Department of Computer Science  
Princeton University  
Princeton, NJ 08544 USA

**ABSTRACT**  
We describe a class of attacks that can compromise the privacy of users’ Web browsing histories. The attacks allow a malicious Web site to determine whether or not the user has recently visited some other, unrelated Web page. The malicious page can determine this information by measuring the time the user’s browser requires to perform certain operations. Since browsers perform various forms of caching, the time required for operations depends on the user’s browsing history; this paper shows that the resulting time variations convey enough information to compromise users’ privacy. This attack method also applies to other types of information gathering by Web sites, such as a more invasive form of Web “cookies”. The attacks we describe can be carried out without the victim’s knowledge, and most “anonymous browsing” tools fail to prevent them. Other simple countermeasures also fail to prevent these attacks. We describe a way of reengineering browsers to prevent most of them.

- Standard Web “anonymization” services do not prevent the attacks; in many cases they actually make the attacks worse.
- Disabling browser features such as Java, JavaScript, and client-side caching do not prevent the attacks.
- The only effective ways we know to prevent the attacks require either an unacceptable slowdown in Web access, or a modification to the design of the browser.
- Even modifying the browser design allows only a partial remedy; several attacks remain possible.

#### 1.1 Why Web Privacy Matters

There is now widespread concern about the privacy of users’ activities on the World Wide Web. The list of Web locations visited by a user often conveys detailed information about the user’s family, financial or health situation. Consequently, users often consider

Felten & Schneider, ACM CCS, 2000

### Browsing history detection

**Browsing history detection**

- CSS on `visited` can load an image and/or reveal if visitor been to a site

- Jesse Ruderman, Mozilla bug #57351, 2000
The (ancient) problems with :visited

Styling visited links differently than unvisited links gives any website 1 bit of information about the user's browsing (whether the user visited a URL or not).

- Detectable with JS (`getComputedStyle()`) or without (`background-image: url(...)`)  

High-speed detection (~30k URLs/s) of any URL visited in a top-level window:

- Search queries, location information, user IDs on social networks, etc.
The fix mitigation (2010)

tl;dr ("Effects on web pages")

- It makes `getComputedStyle` (and similar functions such as `querySelector`) lie by acting as though all links are unvisited.
- It makes certain CSS selectors act as though links are always unvisited, even when they are visited.
- It limits the CSS properties that can be used to style visited links to `color`, `background-color`, `border-*-color`, `outline-color`, `column-rule-color` …

```
visited
unvisited
```

```
> window.getComputedStyle(document.links[1]).color
← "rgb(0, 0, 238)"
```
The (current) problems with :visited

A large and growing number of attacks that bypass existing mitigations.
Including:

1. Attacks based on **user interaction** with the page
2. **Timing attacks**
3. Attacks based on revealing the **color of a single pixel**
4. **Process-level attacks**
Attacks based on user interaction: #1

Weinberg et al, S&P 2011: I still know what you visited last summer

Please click on all of the chess pawns.

Please type the string of characters shown below, then press RETURN. You don’t have to match upper and lower case.

FA4A SABA A-85 A9-5

The large image on the left was assembled from two of the small images on the right: one from the first row and one from the second. Please click on the two small images that make up the large one.

Fig. 3. 7-segment LCD symbols stacked to test three links per composite character. The - at the bottom is always visible, but the 4, 5, and 7 are only visible if a URL was visited.
Attacks based on user interaction: #2

Michal Zalewski, 2013: "Asteroids" game
Attacks based on user interaction: #3

Michal Zalewski, 2016: mix-blend-mode whack-a-mole
Attacks based on user interaction: #4

Ron Masas, 2021: The human side channel

Leaking Your Browser History

Links can have different colors based on whether or not you’ve visited the page they reference. Unfortunately, this behavior can be easily exploited. With a bit of CSS, we can turn the link into a single pixel reflecting this boolean result, and then, we could use one of the techniques I’ve discussed above to trick the user into leaking it.

Yes

URL

https://www.youtube.com/

Did I Visit That Page?
Timing attacks #1: `requestAnimationFrame`

Paul Stone, BlackHat 2013: [Pixel Perfect Timing Attacks with HTML5](#)
NDevTK: [ndev.tk/visted](#)

- `requestAnimationFrame`
  - Can use it to measure frame rate of web page
  - If JS or rendering is too slow, frame rate will drop
  - Can rendering time be used for a timing attack?

- History Sniffing Timing Attack #2
  - Make N link elements with text-shadow
  - For each URL:
    - Update link hrefs to URL
    - Time next frame with `requestAnimationFrame`
    - If frame was slow, link is **visited**
    - Update link hrefs to non-visited URL
Timing attacks #2: High-speed timings in multiple APIs

Michael Smith et al, USENIX WOOT 2018: **Browser history re:visited**
Timing attacks #3: Known WONTFIX'ed bugs

Chromium:

- crbug/252165: Visited links detectable via redraw timing
- crbug/835590: Complicated CSS effects and :visited selector leak browser history through paint timing
  - Duped against 713521: Eliminate :visited privacy issues once and for all

**Bonus #1**: SharedArrayBuffer now allows building high-resolution timers: https://antoinevastel.com/security/privacy/2017/04/09/history-stealing.html

**Bonus #2**: It's not just timings, visitedness can leak in other indirect ways:
- crbug/1205981: Visited links leak via CSS transitions and the transitionrun event
- Manuel Caballero: MS Edge webkitTextFillColor :visited leak
Pixel color attacks #1: Ambient light sensor

Łukasz Olejnik, 2017, Stealing sensitive browser data with the W3C Ambient Light Sensor API
Cross-origin data leaks via the ambient light sensor: arturjanc.com/ls

All the readings below were measured with screen brightness at 50% in a relatively bright room.

- Black: 45 lux. White: 49 lux.
  (Light reflected off the bottom edge of monitor.)

Log
Detecting history: 14 URLs. ETA: 11s.
Detected: https://www.google.com
Detected: https://news.ycombinator.com
Detected: https://www.reddit.com
Detected: https://en.wikipedia.org
Detected: https://en.m.wikipedia.org/wiki/Main_Page
Detected: https://arturjanc.com/ls/demo.html?demo=histor
Pixel color attacks #2: `<input type="color">` eyedropper

1. Use `mix-blend-mode: difference; B(Cb, Cs) = |Cb - Cs|`
2. Set background to `#FFFFFF`, subtract each link's color.
3. Non-visited links have a background-color of `#000000`
4. Visited links have unique colors (a mask with a single set bit).
5. If the final color is:
   a. 255 -> No links visited.
   b. 126 = (255 - 128 - 1) -> First and last links visited.
   c. 0 -> All links visited.
Pixel color attacks #3: Other APIs

Existing APIs that disclose real contents of the user's viewport:

- `<input type="color">`
- **Screen Capture API**: `navigator.mediaDevices.getDisplayMedia()`
- Color picker APIs in browsers' developer tools
- Screenshot functionality in browsers' developer tools

Several proposed new, "more convenient" APIs:

- **getCurrentBrowsingContextMedia**: Casting a video of the current tab.
- **CaptureScreenshot**: Like above, but just a screenshot.
- **EyeDropper API**: Select a color from anywhere on the screen.
- ...?
Process-level attacks

Chromium's Post-Spectre Threat Model Re-Think:

Conclusion

For the reasons above, we now assume any active code can read any data in the same address space. The plan going forward must be to keep sensitive cross-origin data out of address spaces that run untrustworthy code, rather than relying on in-process checks.

Attack #1: Leak the contents of the renderer memory with SpectreJS:

- :visited links still work in cross-origin isolated mode (COOP+COEP)

Attack #2: Renderer compromises without a sandbox escape.
Why should we finally fix :visited?

For **security**: The status quo is that any website can learn the user's browsing history. That's embarrassing.

For **privacy**: Browsing history is global state which allows linking identity across third-party contexts.

For **convenience**: To remove ugly hacks in browsers' CSS implementations and to unblock the shipping of new APIs that would otherwise leak browsing history: screenshots, tab casting, eyedropper color tools, etc.

- These APIs will likely still need to be gated behind cross-origin isolation (COOP+COEP).
How should we fix :visited?

There is a fairly long track record of proposals to address these issues:

- [crbug/713521](https://crbug.com/713521): Eliminate :visited privacy issues once and for all
- [csswg-drafts #3012](https://drafts.csswg.org/cssvisited/): Solve :visited once and for all
- [Google-internal]([Rethinking :visited-ness](https://github.com/sidekick-css/jsel#jsel-carrot))

Practically, we should probably just cut the Gordian knot and store history per-origin, or (for privacy) per-storage partition.