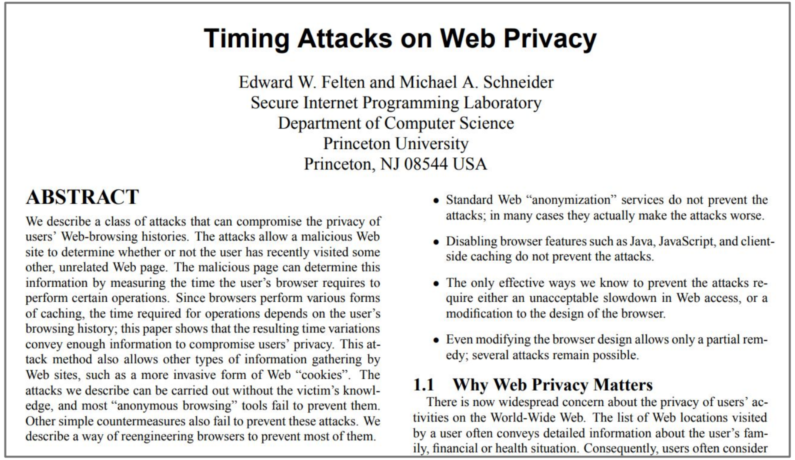
Ceterum censeo :visited esse delendam

Artur Janc, XS-Leaks summit 2021

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 allows 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.

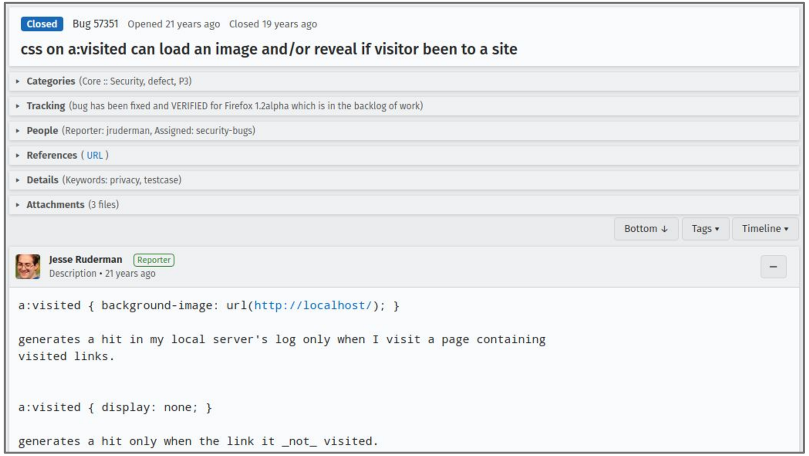
- Standard Web "anonymization" services do not prevent the attacks; in many cases they actually make the attacks worse.
- Disabling browser features such as Java, JavaScript, and client-side caching do not prevent the attacks.
- The only effective ways we know to prevent the attacks require either an 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



Closed Bug 57351 Opened 21 years ago Closed 19 years ago

css on a:visited can load an image and/or reveal if visitor been to a site

- Categories (Core :: Security, defect, P3)
- Tracking (bug has been fixed and VERIFIED for Firefox 1.2alpha which is in the backlog of work)
- People (Reporter: jruderman, Assigned: security-bugs)
- References (URL)
- Details (Keywords: privacy, testcase)
- Attachments (3 files)

Bottom ↓ Tags ▾ Timeline ▾

Jesse Ruderman Reporter
Description • 21 years ago

```
a:visited { background-image: url(http://localhost/); }

generates a hit in my local server's log only when I visit a page containing
visited links.


a:visited { display: none; }

generates a hit only when the link it _not_ visited.
```

Jesse Ruderman, Mozilla bug #57351, 2000

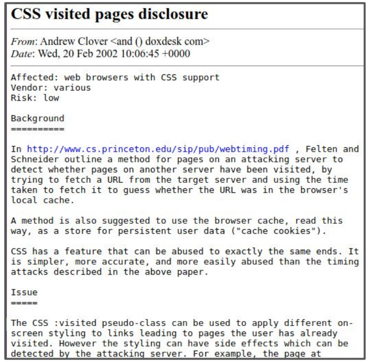
The (ancient) problems with `:visited`

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:

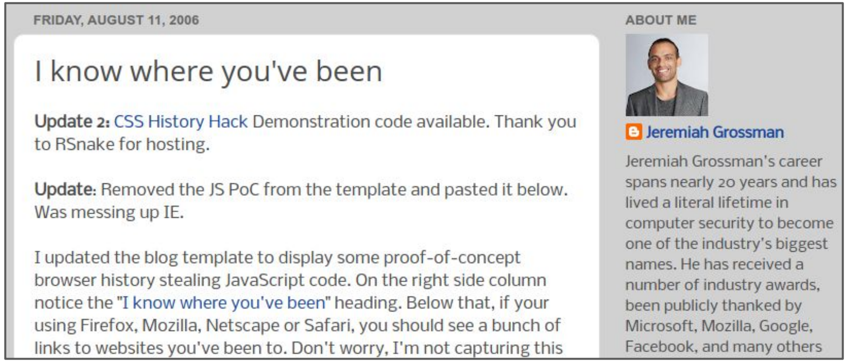
- Search queries, location information, user IDs on social networks, etc.



CSS visited pages disclosure

From: Andrew Clover <and () doxdesk com>
Date: Wed, 20 Feb 2002 10:06:45 +0000

```
Affected: web browsers with CSS support
Vendor: various
Risk: low

Background
==========

In http://www.cs.princeton.edu/sip/pub/webtiming.pdf , Felten and
Schneider outline a method for pages on an attacking server to
detect whether pages on another server have been visited, by
trying to fetch a URL from the target server and using the time
taken to fetch it to guess whether the URL was in the browser's
local cache.

A method is also suggested to use the browser cache, read this
way, as a store for persistent user data ("cache cookies").

CSS has a feature that can be abused to exactly the same ends. It
is simpler, more accurate, and more easily abused than the timing
attacks described in the above paper.

Issue
=====

The CSS :visited pseudo-class can be used to apply different on-
screen styling to links leading to pages the user has already
visited. However the styling can have side effects which can be
detected by the attacking server. For example, the page at
```



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)"
```

Preventing attacks on a user's history through CSS :visited selectors

L. David Baron, Mozilla Corporation

Proposed solution

The approach, in more detail, is as follows: there is at most one element whose presence in the user's history can affect the style of a node: call this the node's *relevant link*. If the node is a link, its relevant link is itself. Otherwise, it is the node's nearest ancestor that is a link, or, if there is no such ancestor, there is no relevant link.

For every node, instead of computing its style by matching selectors against :link and :visited based on whether links are in the user's history, we first compute the style by matching selectors as though all links are unvisited. (This produces an object representing the computed style for the element which, in our code, is called an `nsStyleContext`.) Then, if the node has a relevant link, we compute style a second time on the assumption that the relevant link is visited and all other links are unvisited. This produces a second `nsStyleContext`, which we give the first style context a pointer to (called its *style-if-visited*). We also record in the first style context whether the relevant link was visited. We then handle all dynamic changes to the document or style that would require either of these style contexts to be updated by updating them as needed.

All code except code that is specifically intended to use style based on the history uses the first style context, as all of our existing code does. This causes `getComputedStyle` and other related functions to lie about whether links are in the user's history.

Then, we make the properties that Web pages should be able to style differently for visited links (`color`, `background-color`, `border-*-color`, `outline-color`, `column-rule-color`, `fill`, and `stroke`) opt in to getting the styles for visited links by having the code that implements the drawing for these properties get the color to draw through a function that combines the data from the two style contexts based on whether the relevant link is visited. If the relevant link is not visited, this function returns the color from the first (normal) style context. If the relevant link is visited, it returns a color whose R (red), G (green), and B (blue) components come from the second style context (the style-if-visited) but whose A (alpha) component comes from the first. However, there is one exception to the second rule (to handle the case where the first style context has a usable color and the second style context has a color whose alpha is 0, such as `transparent`, which doesn't have meaningful R, G, and B components): if the color in the style-if-visited has an A component of 0, then the color from the normal style is always used.

It's worth noting that depending on when an implementation starts image loads for images referenced from CSS, the images that are referenced from the if-visited styles for `background-image`, etc., might still be loaded. However, it's important that the implementation ensure that either they're never loaded (preferable), or that they're always loaded at the same time whether or not links are visited.

The (current) problems with :visited

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

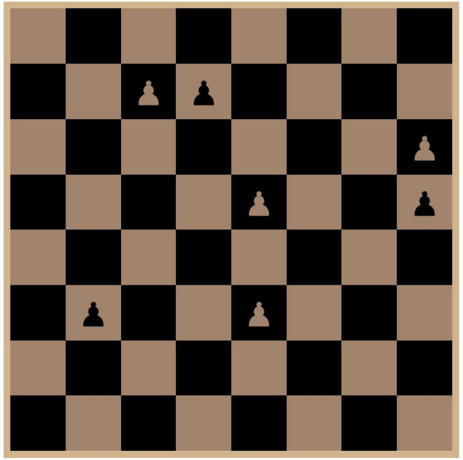
Including:

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]

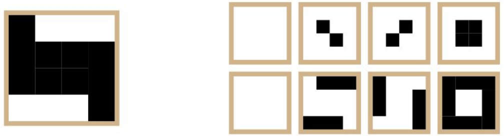
Please click on all of the chess pawns.



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

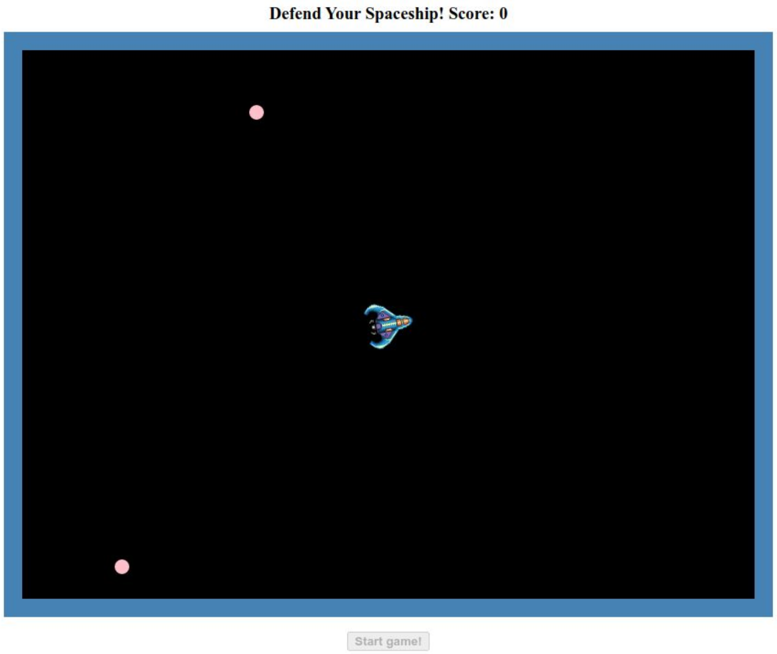
Fig. 3. 7-segment LCD symbols stacked to test three links per composite character. The - at the bottom is always visible, but the 4, 5, and F are only visible if a URL was visited.

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.



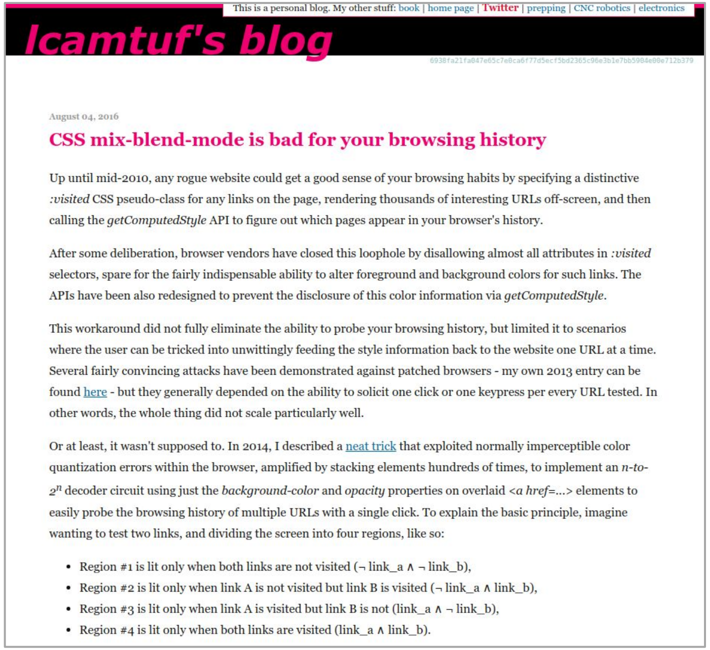
Attacks based on user interaction: #2

Michal Zalewski, 2013: ["Asteroids" game]



Attacks based on user interaction: #3

Michal Zalewski, 2016: [mix-blend-mode whack-a-mole]


This is a personal blog. My other stuff: book | home page | Twitter | prepping | CNC robotics | electronics

lcamtuf's blog

August 04, 2016

CSS mix-blend-mode is bad for your browsing history

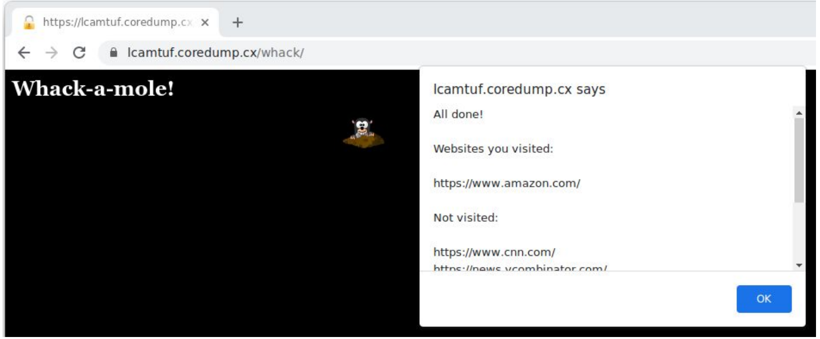
Up until mid-2010, any rogue website could get a good sense of your browsing habits by specifying a distinctive *:visited* CSS pseudo-class for any links on the page, rendering thousands of interesting URLs off-screen, and then calling the *getComputedStyle* API to figure out which pages appear in your browser's history.

After some deliberation, browser vendors have closed this loophole by disallowing almost all attributes in *:visited* selectors, spare for the fairly indispensable ability to alter foreground and background colors for such links. The APIs have been also redesigned to prevent the disclosure of this color information via *getComputedStyle*.

This workaround did not fully eliminate the ability to probe your browsing history, but limited it to scenarios where the user can be tricked into unwittingly feeding the style information back to the website one URL at a time. Several fairly convincing attacks have been demonstrated against patched browsers - my own 2013 entry can be found here - but they generally depended on the ability to solicit one click or one keypress per every URL tested. In other words, the whole thing did not scale particularly well.

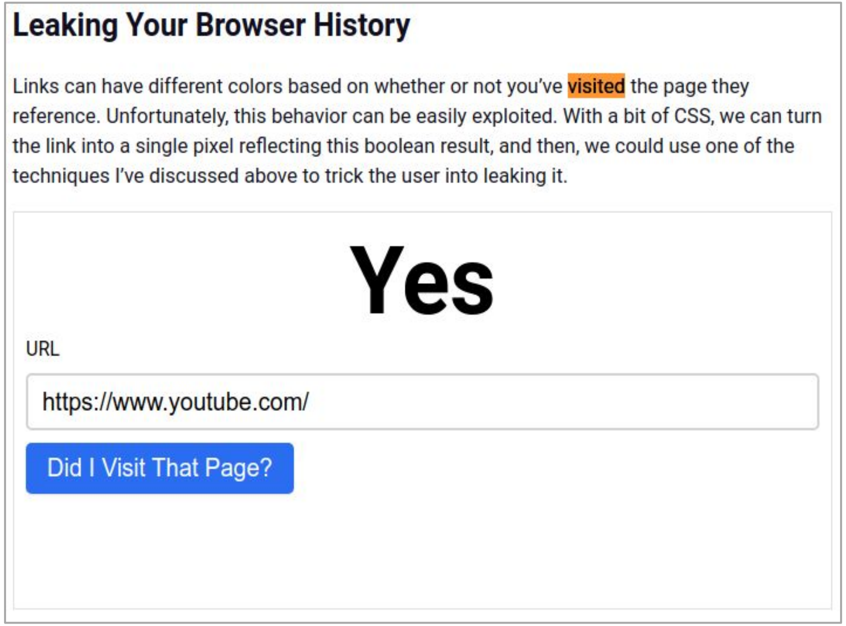
Or at least, it wasn't supposed to. In 2014, I described a neat trick that exploited normally imperceptible color quantization errors within the browser, amplified by stacking elements hundreds of times, to implement an *n-to-2^n* decoder circuit using just the *background-color* and *opacity* properties on overlaid *<a href=...>* elements to easily probe the browsing history of multiple URLs with a single click. To explain the basic principle, imagine wanting to test two links, and dividing the screen into four regions, like so:

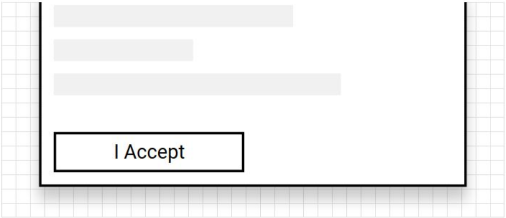
- Region #1 is lit only when both links are not visited (¬ link_a ∧ ¬ link_b),
- Region #2 is lit only when link A is not visited but link B is visited (¬ link_a ∧ link_b),
- Region #3 is lit only when link A is visited but link B is not (link_a ∧ ¬ link_b),
- Region #4 is lit only when both links are visited (link_a ∧ link_b).



Attacks based on user interaction: #4

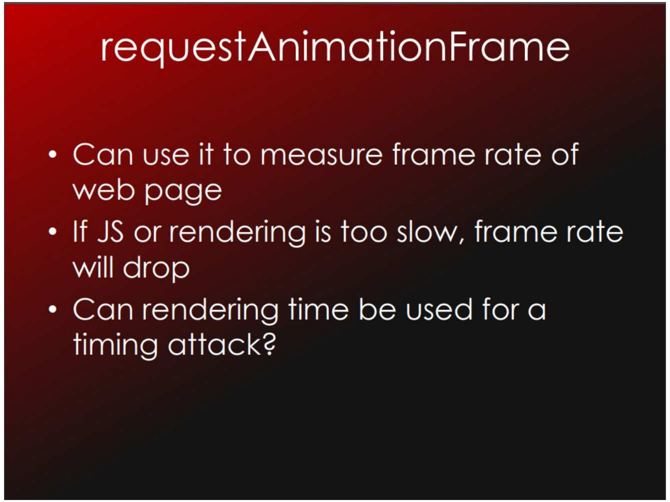
Ron Masas, 2021: The human side channel

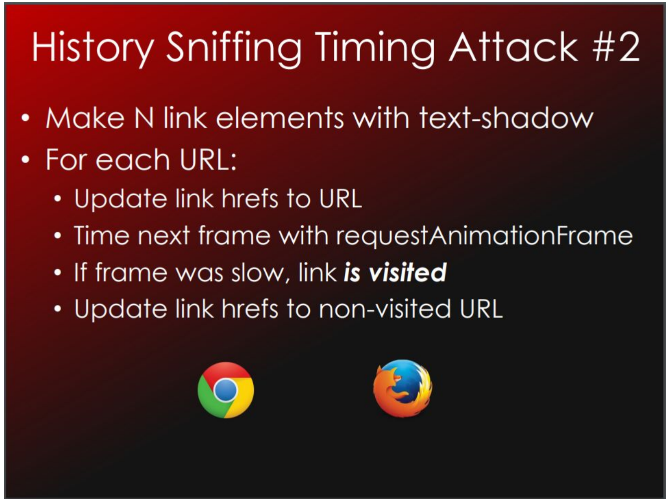




Timing attacks #1: `requestAnimationFrame`

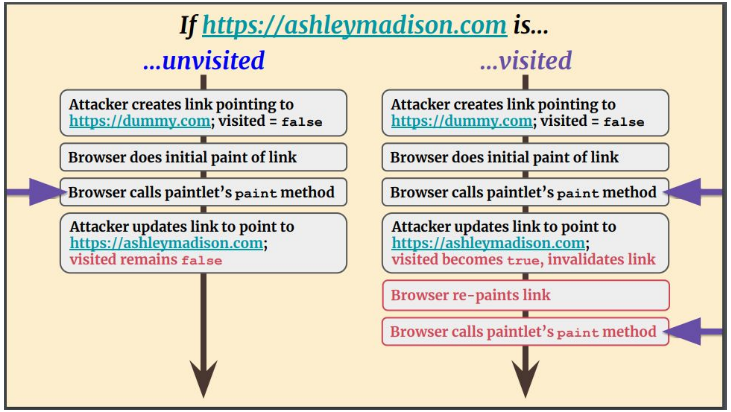
Paul Stone, BlackHat 2013: [Pixel Perfect Timing Attacks with HTML5]()
NDevTK: [ndev.tk/visted]()

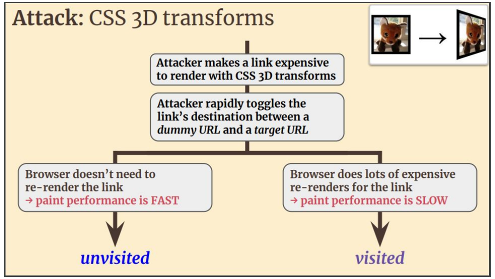


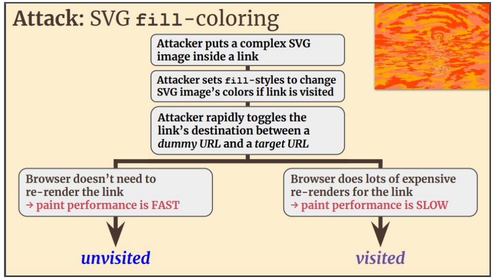


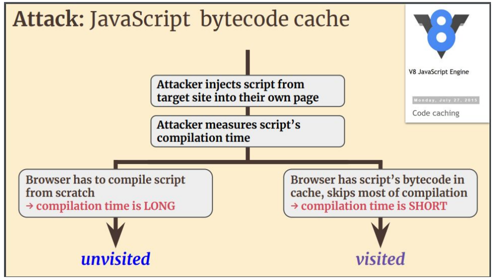
Timing attacks #2: High-speed timings in multiple APIs

Michael Smith et al, USENIX WOOT 2018: Browser history re:visited









4 APIs, 4 attacks

- CSS Paint API
- CSS 3D transforms
- SVG `fill`-coloring
- JavaScript bytecode cache

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*

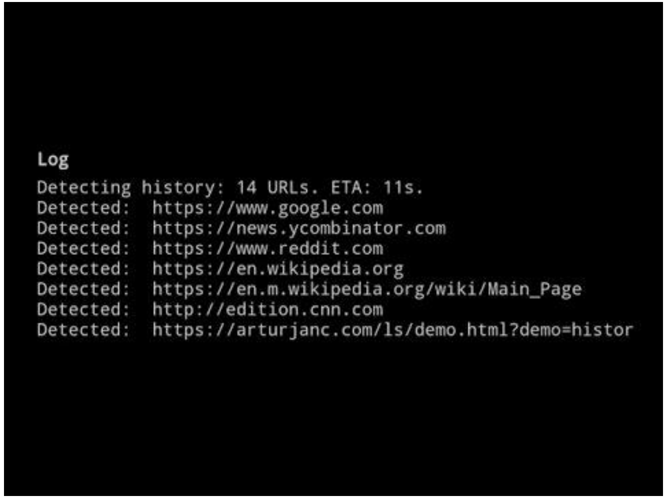
Bonus #1: SharedArrayBuffer now allows building high-resolution timers:
https://antoinevastel.com/security/privacy/2017/04/09/history-stealing.html

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

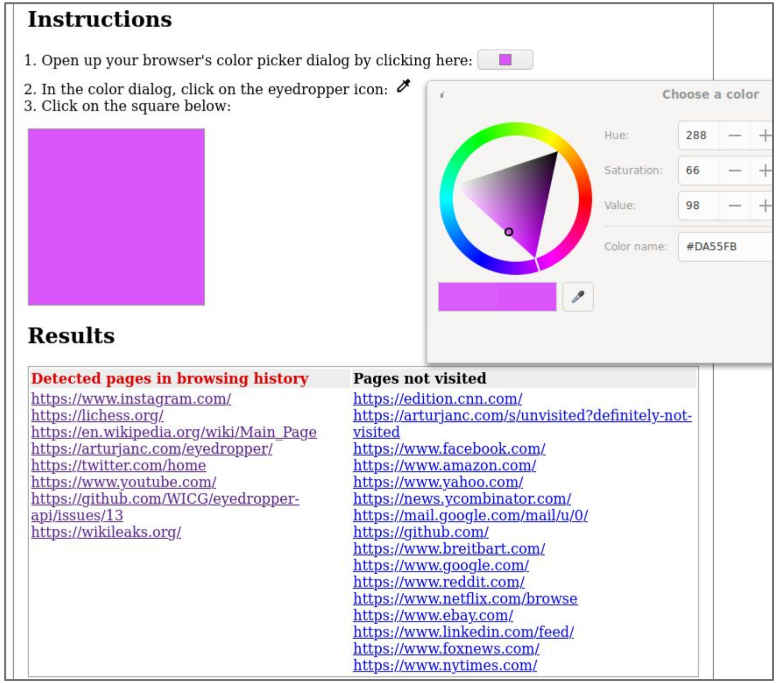
Łukasz Olejnik, 2017, [Stealing sensitive browser data with the W3C Ambient Light Sensor API]

Cross-origin data leaks via the ambient light sensor: [arturjanc.com/ls]



Pixel color attacks #2: `<input type="color">` eyedropper

arturjanc.com/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.
6. Each color selection reveals 24 bits of history.

```
#link-container .first-dummy-link { background-color: #ffffff; }
#link-container a:link            { background-color: #000000; }
#link-container #link1:visited    { background-color: #800000; }
#link-container #link2:visited    { background-color: #400000; }
#link-container #link3:visited    { background-color: #200000; }
#link-container #link4:visited    { background-color: #100000; }
#link-container #link5:visited    { background-color: #080000; }
#link-container #link6:visited    { background-color: #040000; }
#link-container #link7:visited    { background-color: #020000; }
#link-container #link8:visited    { background-color: #010000; }
#link-container #link9:visited    { background-color: #008000; }
```

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: *Eliminate :visited privacy issues once and for all*
- csswg-drafts #3012: *Solve :visited once and for all*
- [Google-internal] *Rethinking :visited-ness*

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