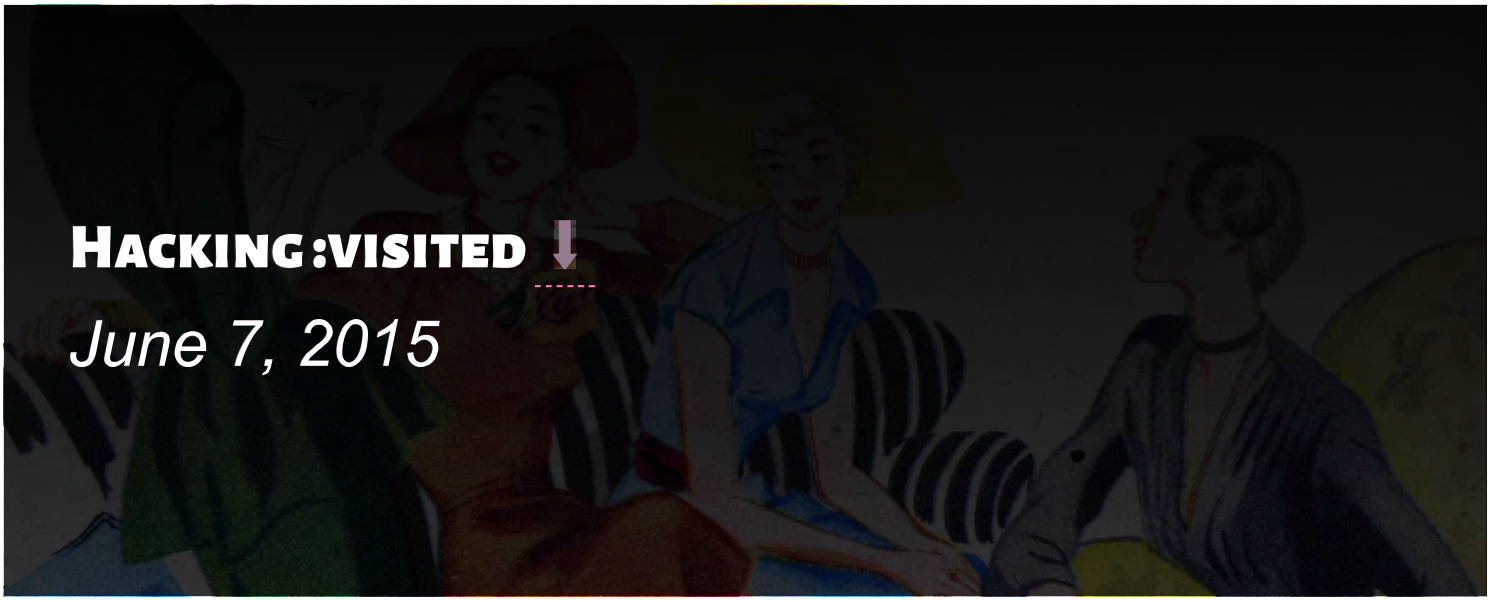

Hacking :visited ⬇

June 7, 2015

Listen to this blog post: ▶ 0:00 / 7:43

> It's been on my to-do list for two weeks now — adding a `:visited` state to my blog posts. I was inspired after reading this blog post by Joel Califa, and thought "Heck yeah. That's a great idea. I can use it on my blog to represent links that haven't been read yet." But like most things on the Internet, it looked a lot cleaner from a distance.

What is :visited?

`:visited` is a pseudo-class that is used to style links which have already been accessed in the browser of a user. Basically when you can tell if you have already been to a link or haven't yet (Joel's post goes over this really nicely – I'd highly recommend checking it out). In the early days of the Internet, stylig visited links was extremely prevalant, and a great way to understand context of place in a digital space. We seem to have lost that

practice, and I rarely see it in action these days.

Limitations

It all started fine and dandy. I applied `:visited` to the links on the home page and then accessed the child `h2` within them to change it's text color. This was a test. To avoid annoying my audience, what I really wanted was `:unvisited`, or to display a strikeout, or checkbox, or an *"Unread!"* badge. I could do any of those with the `:after` psuedo element. Well upon trying it, everything broke. So I did some research and discovered a few fun facts from reading the fine print:

> *"For privacy reasons, browsers strictly limit the styles you can apply using an element selected by this pseudo-class: only color, background-color, border-color (and its sub-properties), outline-color, column-rule-color, and the color parts of fill and stroke."*

So you can only change color? Well, being the stubborn girl I am, I knew I could work around this. I first tried to inject style with JavaScript. That was fine – but it didn't overpower the style logic embedded in browsers. The method `getComputedStyle()` is disabled for this pseudo class. According to the official Mozilla developer documentation: **"the method getComputedStyle will lie."** Nice.

Then, I thought: maybe I can use the `color: transparent` to essentially hide the pseudo element content and only show the color when I want to apply them! I'm changing only the color, right? But when I did this, *all* of the pseudo elements (not just the visited ones) disappeared… *Wat?* … It turns out that the `:visited` tag will always pull its parent's alpha channel. So, even trying `rgba(0,0,0,0)` will make everything disappear.

Why the Limitations?

So what's up with this strict limitation anyway? Well, the way that `:visited` works is by walking through the user's history to figure out what sites the they've visited. This means that a lot of information can be accessed about that user, and their identity could be inferred. In 2010, a lot of

changes were made to limit access to this type of information. Under certain circumstances, the browser is more likely to lie and mark a link as unvisited.

It's Not Over Till I Win

It ended up being a pretty simple solution. To get the effect I wanted, I first, gave every element with a blog title (`h2` in this case) an `:after` pseudo-element and styled it to my preferences:

```
h2 {
  color: blue;

  &:after {
    content: '(unread!)';
    color: hotpink;
    display: inline-block;
    font-size: .6em;
    margin-left: .5em;
    vertical-align: middle;
  }
}
```

Then I had to apply the `:visited` pseudo class to all of the links in the list and target a child element `h2:after` to effectively *"hide"* it. I'm "hiding" it here by giving it the same color as the background (white). If you're not familiar with the power of the Sass ampersand, I'd recommend checking out this post.

```
a {
  display: block;
  text-decoration: none;

  &:visited h2:after {
    color: white;
  }
}
```

Awesome! So we're almost there. `:visited` is working. I've gotten relatively "hidden" badges. But then there's this hover… I'm applying a lightgrey background to each list-item when hovering over them, and I need to make sure I don't give away my pseudo-element secrets when hovering. So I simply just need to compensate:

```
// set global transition:
%transition-duration {
  transition-duration: .5s;
}

li {
  // applying uniform transition:
  @extend %transition-duration;

  // this is where I'm apply hover style:
  &:hover {
    background-color: lightgrey;

    // within the hover state, I'm also transitioning the h2:after color to
match the background transition
    a:visited h2:after {
      color: lightgrey;
    }
  }
}

// return to the header to give it an identical transition:
h2 {
  color: blue;

  &:after {
    // applying uniform transition:
    @extend %transition-duration;
    ...
  }
}
```