Dangers and Prevalence of Unprotected Web Fonts

Tobias Mueller*, Daniel Klotzsche*, Dominik Herrmann‡ and Hannes Federrath*
*University of Hamburg, Germany
Email: {mueller,3klotzsc,federrath}@informatik.uni-hamburg.de
‡University of Bamberg, Germany
Email: dominik.herrmann@uni-bamberg.de

***Abstract*—Most Web sites rely on resources hosted by third parties such as CDNs. Third parties may be compromised or coerced into misbehaving, e.g. delivering a malicious script or stylesheet. Unexpected changes to resources hosted by third parties can be detected with the Subresource Integrity (SRI) mechanism. The focus of SRI is on scripts and stylesheets. Web fonts cannot be secured with that mechanism under all circumstances. The first contribution of this paper is to evaluates the potential for attacks using malicious fonts. With an instrumented browser we find that (1) more than 95% of the top 50,000 Web sites of the Tranco top list rely on resources hosted by third parties and that (2) only a small fraction employs SRI. Moreover, we find that more than 60% of the sites in our sample use fonts hosted by third parties, most of which are being served by Google. The second contribution of the paper is a proof of concept of a malicious font as well as a tool for automatically generating such a font, which targets security-conscious users who are used to verifying cryptographic fingerprints. Software vendors publish such fingerprints along with their software packages to allow users to verify their integrity. Due to incomplete SRI support for Web fonts, a third party could force a browser to load our malicious font. The font targets a particular cryptographic fingerprint and renders it as a desired different fingerprint. This allows attackers to fool users into believing that they download a genuine software package although they are actually downloading a maliciously modified version. Finally, we propose countermeasures that could be deployed to protect the integrity of Web fonts.**

***Index Terms*—fonts, web, integrity, attack surface**

I. Introduction

The Web is a very popular application on the Internet and part of its success arguably stems from the fact that it allows loading and embedding resources from third parties. In fact, studies have shown that this behaviour is common practise [1], [2] and that a browser needs to establish about 20 separate TLS connections to different host names in order to render a Web page [2], [3].

Loading resources from a third party's host, however, bears risks [4], [5]. Not only can a third party memorise a user's visit and create tracking profiles [3], [6]–[8], it can also discriminate users and alter the content of the requested resource based on the (real-world or pseudonymous) identity of a user. While this may be desirable for certain use cases, such as displaying random images of kittens, it is problematic when it affects security-sensitive content.

In order to protect against the threat of a third party altering content, Web browsers implement Subresource Integrity (SRI) [9] which is a mechanism to load content only if it hashes to an expected value. With this mechanism, Web site providers can instruct browsers to reject the contents of a script (e.g. jQuery or Bootstrap) or stylesheet if it has been altered without notifying the Web site provider.

SRI has been standardised with scripts and stylesheets in mind that are included using the `link` and `script` HTML tags. However, it has been overlooked to protect Web fonts that are included within a stylesheet. Apparently, the attack potential of Web fonts has been deemed insignificant.

With this paper we want to raise attention for this gap. To the best of our knowledge, we are the first to concretely demonstrate and quantify the security implications of malicious fonts. To this end, we make the following contributions:

- We estimate the amount of vulnerable Web sites by measuring the prevalence of SRI on and the use of embedded third-party fonts on popular Web sites,
- we present a proof-of-concept attack that exploits the capabilities malicious Web fonts, and
- we discuss countermeasures that load fonts with integrity protection.

II. Background

This section briefly describes our attack scenario as well as techniques to load a custom font face on a Web site. Finally, we describe the capabilities of Web fonts that will be exploited for our attack.

A. Attacker model

In our scenario, we assume that Web site owners have best intentions. They create a Web site which loads a font from a third party, such as Google Fonts. This connection may or may not be established via TLS [10]. We assume that the third party hosting the font is compromised or coerced into serving a malicious font, i.e. it is able to discriminate requests from the site owner and requests from a targeted user to serve either a benign or a malicious version of the requested font. Such discrimination is commonplace on today's Web [6], [8], [11], [12]. The integrity protection offered by TLS cannot detect such modifications.

Our assumed victim is a security-conscious user who aims to obtain security-sensitive information from the site. Without loss of generality we assume that the user is interested in the value of a cryptographic fingerprint of an OpenPGP certificate or the hash value of a file to be downloaded from the (first-party) Web server. For example, the download page of the

"Signal" messenger app contains the following hint (version 4.39.4, fingerprints shortened):

> You can verify the signing certificate on the APK matches this SHA256 fingerprint:
> `29:F3:4E:5F:27:F2:11:B4:24:BC...`
> `EA:FB:A2:DA:35:AF:35:C1:64:16...`

The attacker's goal is to make the browser render the fingerprint differently, i.e. to show a different fingerprint, without modifying the source code of the Web site the user visits.

B. Styling Web site elements

Styling an element with a custom font has been introduced with HTML 3 via a `font` tag (fig. 1). The `font` tag, which has been deprecated with HTML 5 in 2014 [13], required fonts to be installed on a user's system. Dynamically loading a font from a server on the Internet was introduced with CSS3 in 2013. A contemporary way to use a font from a third party host is provided in fig. 2. This snippet replaces the `font` tag shown in fig. 1. Note, however, that not including a CSS file instead of a font is very common.

C. Subresource Integrity

Web sites commonly rely on third-party services offered by, e.g. Google, Cloudflare, or Akamai, to host and serve content. The prevalence of such Content Delivery Networks (CDNs) is very strong [2]. A Web site developer wanting to protect its users from a compromised CDN can make use of the SRI mechanism when embedding the remote resource (fig. 3).

To this end the developer includes the hash of the intended resource when defining the use of a resource. When the browser has retrieved a linked resource it then hashes its content and checks whether the hash matches the developer's expectation denoted in the `integrity` attribute. SRI's integrity attribute can only be added to `link` and `script` HTML tags but not to CSS definitions. As a result. Web fonts, which are defined in CSS, cannot be protected with SRI.

In experiments, we also tried to trick browsers into performing SRI on fonts by including them with the `link` tag in addition to the CSS definition. This technique can actually be used for pre-loading of fonts. However, browsers ignore the `integrity` attribute in this case.

```
<p><font face="Georgia, Arial, Garamond">
    This paragraph has had its font...
  </font></p>
```

Fig. 1. A HTML2 snippet showing how to use a font (limited to the fonts installed on user's machine)

```
@font-face {
  font-family: Gentium;
  src: url(http://xmpl.com/Gentium.woff);
}

p { font-family: Gentium, serif; }
```

Fig. 2. A CSS file to use a font from a remote host rather than the user's machine

D. Capabilities of Fonts

When a computer renders text on the screen it maps the characters of the text to graphical representations (glyphs) that are defined in the font's file. Before glyphs are rendered on the screen they are "shaped". For example, the three glyphs `ffi` are replaced with a single and more pleasant ligature: ffi.

Arguably the most prevalent standard for fonts is OpenType [14]. It supports various typesetting features which can be expressed via a text file in the corresponding "feature file (fea) format" [15]. Of particular interest for us is the possibility to define ligatures which coalesce two (or more) glyphs into one. Additionally, it allows to expand one glyph into multiple other glyphs. This feature can be used by a font to convert a certain character, e.g. one with tréma (such as ä), into two or more glyphs, e.g. the `a` and the tréma. Other features of fonts, such as kerning, will then move the tréma back over the other glyph. Among the features of OpenType fonts is also the contextual replacement which will only replace glyphs if they appear before or after certain other glyphs. It has been reported that these features make fonts Turing complete [16]. For the Turing completeness to work in practice, however, hard coded limits in the font interpreter need to be lifted.

To the best of our knowledge the closest work to ours is a font named SansBullshitSans[1]. That font coerced several glyphs into a new glyph, e.g. "cloud" into a glyph that appears as "bullshit". Note that SansBullshitSans replaces multiple glyphs with one specifically crafted glyph which has the appearance of several other glyphs rather than actually using several other glyphs of the font.

III. Crafting a Malicious Font

This section describes how to craft a malicious font based on our proof-of-concept font *SansFingerprintSans*.

The objective for creating a fingerprint hiding font is to (1) replace the target fingerprint with a different glyph and then (2) expand that glyph back to a series a characters which represent a different fingerprint.

We created our font as follows: We crafted two ligature tables (cf. section II-D). The first table coalesces the sequence of 64 characters (for SHA256, 40 for SHA1) into one, specially crafted, glyph, e.g. the unicode codepoint U+0600. The second table expands that newly defined glyph into a series of glyphs which represent the desired fingerprint (fig. 4).

[1] http://www.sansbullshitsans.com/

```
<script src="https://example.com/lib.js"
        integrity="sha384-oqd[...]]GYwC"
        crossorigin="anonymous"></script>
```

Fig. 3. A script defined with the SRI hash of the expected content. Clients can use this hash to reject the loaded content if it does not hash to the same value. The value in the `integrity` attribute contains a base64-encoded SHA384 hash (shortened to fit on the page).

```
lookup ligaStandardLigatureslookup3 {
  lookupflag 0;
    sub \zero \five \six \four \four \six
        \F \zero \seven \seven \three \two
        [...] by \uniE600;
} ligaStandardLigatureslookup3;

lookup ligaStandardLigatureslookup4 {
  lookupflag 0;
    sub \uniE600 by \F \two \eight \nine
        \F \seven \B \A \seven [...] ;
} ligaStandardLigatureslookup4;
```

Fig. 4. The behaviour of the malicious font in Feature File Syntax. It replaces a certain series of characters, e.g. a cryptographic fingerprint, with a series of other characters. The fingerprints have been shortened for brevity.

To automate this process we built a tool that generates a malicious font for pairs of fingerprints. It takes the fingerprint to be attacked as well as the fingerprint which should appear instead as arguments and returns a TrueType font file. The runtime is in the order of a few milliseconds, which makes it possible to adapt the attack dynamically in real time.

The resulting font and the generation tool have been published on https://github.com/muelli/SansFingerprintSans.

To assess the relevance of our finding we assess the attack surface in the next section. To this end we analyse the prevalence of SRI and embedded fonts.

IV. Quantifying the attack surface

We start out by describing our approach to finding remote resource usage and how we assessed the use of SRI.

A. Scanning for Remote Resource Usage

We built a scanner to assess whether a Web site includes resources from remote hosts and whether the client made use of SRI for the corresponding requests. Our scanner instruments Chrome using the Devtools protocol[2] via a JavaScript abstraction. We record the HTTP requests while a Web site is retrieved and wait for the page to be loaded completely. Then we traverse the Document Object Model (DOM) and match the observed requests to individual nodes in the DOM. We check the nodes for an integrity attribute and whether the browser reported an error while loading the associated resource. If a resource was loaded successfully and its node had an integrity attribute, we assume that the resource was protected with SRI and its content was genuine. If a node had the integrity attribute and the corresponding resource failed to load, despite the server having returned a response with a status code of 200, we assume a failure in SRI to be the problem.

[2]https://chromedevtools.github.io/devtools-protocol/

B. Prevalence of SRI on the Web

A study published in June 2018 reported that only 0.7% of Web sites used SRI to protect resources [17]. However, that study lacks details such as the type of resource and the identity of the third parties that serve particular files. Moreover, it relied on Alexa's top list, which has been criticised for its high churn and bias [18]. In contrast, we use the Tranco [18] top list[3] to obtain more robust results.

For scanning 50000 Web sites, our scanner performed 5719188 requests for additional resources. Of all the requested resources, only 0.3% (5256) were protected with SRI. Of those SRI protected resources 1.86% (90) failed to load. Interestingly, in 62.24% (61) cases SRI was absent although the Content Security Policy (CSP) of the server required it. In 22.45% (22) cases the hash mismatched and in 3.06% (3) the hash provided by the developer could not be parsed. The remaining 4.08% (4) failed because the request timed out. The SRI protection mechanism was used by 5.71% (2855) of the 50000 Web sites.

The vast majority (69.56%) of the resources loaded via SRI are scripts. The only other protected type of resource is stylesheet (30.44%). Moreover, we have observed one Web site trying to load a font with SRI protection (bugatti.com) by including an `integrity` attribute in the link tag. Since browsers ignore that integrity attribute in this case (cf. section II-C), no protection was in place.

The SRI mechanism was used for external resources hosted on third party hosts in 96.33% (5063) of the cases. Same-domain resources were protected with SRI in 3.67% (193) of the cases.

The most often SRI protected resource is the jQuery library, closely followed by Font Awesome and Bootstrap (table II). Those make up for well over 50% of all SRI-protected resources. The hosts serving those resources belong to the StackPath company group.

Of the top 50000 Web sites, 3.92% made no use of third-party resources at all. 62.22% loaded a font from a third-party host. In total, 156862 fonts were loaded. Google is serving the most fonts with a large margin (table I), serving more than 6.5 times as many fonts as the runner-up.

V. Discussion

In this section we discuss the limitations of our measurement study and propose techniques to protect the integrity of fonts.

A. Limitations

The scanner presented in section IV-A suffers from a limitation in the Devtools protocol which does not directly support providing information about SRI. Because we have to rely on a message related to the logging subsystem, rather than the one responsible for making requests, we cannot see whether the browser has actually performed a verification of the SRI hash for a loaded resource. Chrome only provides a

[3]https://tranco-list.eu/list/J9ZY/50000

TABLE I
TOP 10 HOSTS DELIVERING FONTS

Ratio	Fonts delivered	Hostname
53.21%	83470	fonts.gstatic.com
8.05%	12629	use.typekit.net
5.19%	8141	base64-encoded-string
1.31%	2056	maxcdn.bootstrapcdn.com
1.01%	1588	use.fontawesome.com
0.45%	713	use.typekit.com
0.25%	390	themes.googleusercontent.com
0.21%	330	netdna.bootstrapcdn.com
0.18%	290	js.driftt.com
0.15%	232	pro.fontawesome.com

TABLE II
TOP 10 HOSTS DELIVERING SRI PROTECTED CONTENT

Ratio	Count	Hostname
19.16%	1007	code.jquery.com
14.82%	779	use.fontawesome.com
13.43%	706	cdnjs.cloudflare.com
12.16%	639	maxcdn.bootstrapcdn.com
9.19%	483	cdn.shopify.com
4.72%	248	stackpath.bootstrapcdn.com
4.01%	211	assets.publishing.service.gov.uk
2.47%	130	cdn.vox-cdn.com
2.44%	128	pro.fontawesome.com
1.92%	101	d10lpsik1i8c69.cloudfront.net

rather generic log entry for a SRI failure instead of a structured notification in case SRI verification was successful (or failed). Our scanner would have benefited from the Devtools protocol explicitly supporting SRI by way of extending the requests and response structures.

As described we address this challenge by iterating over the DOM and finding all `script` and `link` elements and check for the existence of an `integrity` attribute (cf. section II-C). This, however, makes the scanner susceptible to time of check, time of use (TOCTOU) deficiencies: A Web site provider could evade detection of loading unprotected content by exploiting our measurement gap. To this end, the server would offer the browser a resource without SRI at first. Once the resource has been loaded, the server would modify the HTML source code with a script that adds a fitting `integrity` attribute to the element. When our scanner iterated over the DOM it would then detect an element with the `integrity` element (fig. 5). We have found no indicators for such modifications during our tests.

A second limitation results from the fact that our findings are based on the front page of Web sites, i.e. we did not perform a deep crawl. It may be possible that some Web sites implement SRI only on a subset of their pages. One could imagine that pages requiring logging in, such as banking or messaging, have a better protected membership area. A more thorough assessment that also includes subpages is left as future work.

B. Solutions

One way of addressing the problem of unprotected fonts is to allow the `integrity` attribute for loading fonts. However, because no dedicated `font` tag exists in HTML and fonts can only be loaded via CSS where no concept akin to SRI exists yet, a similar mechanism could be deployed there. In fact, several approaches exist. Firstly, the `src` property of the `font-face` CSS rule could take a URL which includes integrity protection, e.g. `sha384:ABCD/https://foo`. Secondly, CSS could be extended with a new `integrity` attribute that can be used in conjunction with `src`. Thirdly, HTML could be extended to allow loading fonts via the link tag and respect the integrity tag.

VI. CONCLUSION

We have shown the relevance of integrity protection for fonts that are hosted on third-party servers on the Web. A malicious font can render information differently without breaking existing integrity protection mechanisms. We demonstrated how to build a font that changes the rendering of cryptographic fingerprints that are being used by advanced users to verify the authenticity of software and information. This allows an attacker to trick a victim into downloading compromised software. We have also shown that there are only very few entities that serve the lion's share of the fonts used on the most popular Web sites. Given their reach these entities are enticing targets and it is conceivable that powerful attackers may try to compromise or coerce these entities into misbehaving. Finally, we have shown several straightforward methods to detect malicious fonts. Given the attack surface the affected Web standards should be extended accordingly.

```
<head>
    <link rel="stylesheet"
    href="some.css" /></head>
<body>
<script>
let link = document
    .getElementsByTagName('link')[0];
window.onload = () => {
  link.setAttribute('integrity',
     'sha384-bgOyR[...]Zw1T');
  link.setAttribute('crossorigin',
     'anonymous');
}
</script></body>
```

Fig. 5. Code which evades the detection of using unprotected resource due to a measurement gap in the Devtools protocol which reports only SRI failures but not SRI successes. This leads to a race in correlating successful requests and `integrity` attributes on DOM nodes.

References

[1] N. Nikiforakis, L. Invernizzi, A. Kapravelos, S. Van Acker, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "You Are What You Include: Large-scale Evaluation of Remote Javascript Inclusions," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12, New York, NY, USA: ACM, 2012, pp. 736–747. DOI: 10.1145/2382196.2382274.

[2] M. Ikram, R. Masood, G. Tyson, M. A. Kaafar, N. Loizon, and R. Ensafi, "The Chain of Implicit Trust: An Analysis of the Web Third-party Resources Loading," in *The World Wide Web Conference*, (San Francisco, CA, USA), ser. WWW '19, New York, NY, USA: ACM, 2019, pp. 2851–2857. DOI: 10.1145/3308558.3313521.

[3] E. Sy, C. Burkert, H. Federrath, and M. Fischer, "Tracking Users across the Web via TLS Session Resumption," in *Proceedings of the 34th Annual Computer Security Applications Conference on - ACSAC '18*, San Juan, PR, USA: ACM Press, 2018, pp. 289–299. DOI: 10.1145/3274694.3274708.

[4] B. Stock, S. Lekies, T. Mueller, P. Spiegel, and M. Johns, "Precise Client-side Protection against DOM-based Cross-Site Scripting," in *23rd USENIX Security Symposium (USENIX Security 14)*, San Diego, CA, USA: USENIX Association, Aug. 20, 2014. (visited on 06/25/2014).

[5] D. Kumar, Z. Ma, Z. Durumeric, A. Mirian, J. Mason, J. A. Halderman, and M. Bailey, "Security Challenges in an Increasingly Tangled Web," in *Proceedings of the 26th International Conference on World Wide Web*, (Perth, Australia), ser. WWW '17, Republic and Canton of Geneva, Switzerland: International World Wide Web Conferences Steering Committee, 2017, pp. 677–684. DOI: 10.1145/3038912.3052686.

[6] S. Englehardt and A. Narayanan, "Online Tracking: A 1-million-site Measurement and Analysis," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, Vienna, Austria: ACM Press, 2016, pp. 1388–1401, ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978313. (visited on 06/05/2019).

[7] T. Mueller, M. Marx, H. Pridoehl, P. Wichmann, and D. Herrmann, "Sicherheit und Privatheit auf deutschen Hochschulwebseiten: Eine Analyse mit PrivacyScore," in *Sicherheit in vernetzten Systemen*, Hamburg, Feb. 27, 2018, ISBN: 978-3-7460-8637-8.

[8] J. R. Mayer and J. C. Mitchell, "Third-Party Web Tracking: Policy and Technology," in *2012 IEEE Symposium on Security and Privacy*, May 2012, pp. 413–427. DOI: 10.1109/SP.2012.47.

[9] D. Akhawe, F. Braun, F. Marier, and J. Weinberger. (Jun. 23, 2016). Subresource Integrity, [Online]. Available: https://www.w3.org/TR/SRI/ (visited on 06/05/2019).

[10] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," Request for Comments, no. 8446, p. 160, Aug. 2018. DOI: 10.17487/RFC8446. [Online]. Available: https://rfc-editor.org/rfc/rfc8446.txt.

[11] D. Jang, R. Jhala, S. Lerner, and H. Shacham, "An Empirical Study of Privacy-violating Information Flows in JavaScript Web Applications," in *Proceedings of the 17th ACM Conference on Computer and Communications Security*, ser. CCS '10, New York, NY, USA: ACM, 2010, pp. 270–283, ISBN: 978-1-4503-0245-6. DOI: 10.1145/1866307.1866339. (visited on 11/29/2013).

[12] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna, "Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis," in *In Proceeding of the Network and Distributed System Security Symposium (NDSS'07)*, 2007. DOI: 10.1.1.117.6526.

[13] I. Hickson, R. Berjon, S. Faulkner, T. Leithead, E. D. Navara, T. O'Connor, and S. Pfeiffer. (Oct. 28, 2014). HTML5: A vocabulary and associated APIs for HTML and XHTML, [Online]. Available: https://www.w3.org/TR/2014/REC-html5-20141028/ (visited on 06/05/2019).

[14] P. Constable, M. Jacobs, and R. McKaughan. (Aug. 15, 2018). OpenType specification - Typography, [Online]. Available: https://docs.microsoft.com/en-us/typography/opentype/spec/ (visited on 06/18/2019).

[15] Adobe. (Mar. 21, 2019). OpenType Feature File Specification, [Online]. Available: http://adobe-type-tools.github.io/afdko/OpenTypeFeatureFileSpecification.html (visited on 06/15/2019).

[16] M. C. Maxfield. (Mar. 7, 2019). Litherum: Addition Font, [Online]. Available: https://litherum.blogspot.com/2019/03/addition-font.html (visited on 03/18/2019).

[17] R. Shah and K. Patil, "A measurement study of the subresource integrity mechanism on real-world applications," *International Journal of Security and Networks*, vol. 13, no. 2, p. 129, 2018, ISSN: 1747-8405, 1747-8413. DOI: 10.1504/IJSN.2018.092474. (visited on 06/05/2019).

[18] V. Le Pochat, T. Van Goethem, S. Tajalizadehkhoob, M. Korczynski, and W. Joosen, "Tranco: A Research-Oriented Top Sites Ranking Hardened Against Manipulation," in *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, CA: Internet Society, 2019. DOI: 10.14722/ndss.2019.23386. (visited on 06/05/2019).