SANS ISC: InfoSec Handlers Diary Blog - SANS Internet Storm Center SANS Site Network Current Site SANS Internet Storm Center Other SANS Sites Help Graduate Degree Programs Security Training Security Certification Security Awareness Training Penetration Testing Industrial Control Systems Cyber Defense Foundations DFIR Software Security Government OnSite Training InfoSec Handlers Diary Blog

Published: 2017-08-29
Last Updated: 2017-08-29 14:25:45 UTC
by Renato Marinho (Version: 1)

1. **Introduction**

It seems that Google Chrome extensions have become quite the tool for banking malware fraudsters. Two weeks ago, an offender phoned a victim and asked him to install a supposedly new bank security module that, instead, was a malicious extension hosted at the Google Chrome app store aimed to steal victim's banking credentials [1]. This week I received a report about a targeted email phishing campaign against another company with a suspicious attachment. The attachments, after the analysis detailed in today's diary, revealed itself to be another Google Chrome extension prepared to steal banking credentials, credit card, CVV numbers and fraud "compensation tickets" (a popular and particular Brazilian payment method; we call it "boleto") to divert payments.

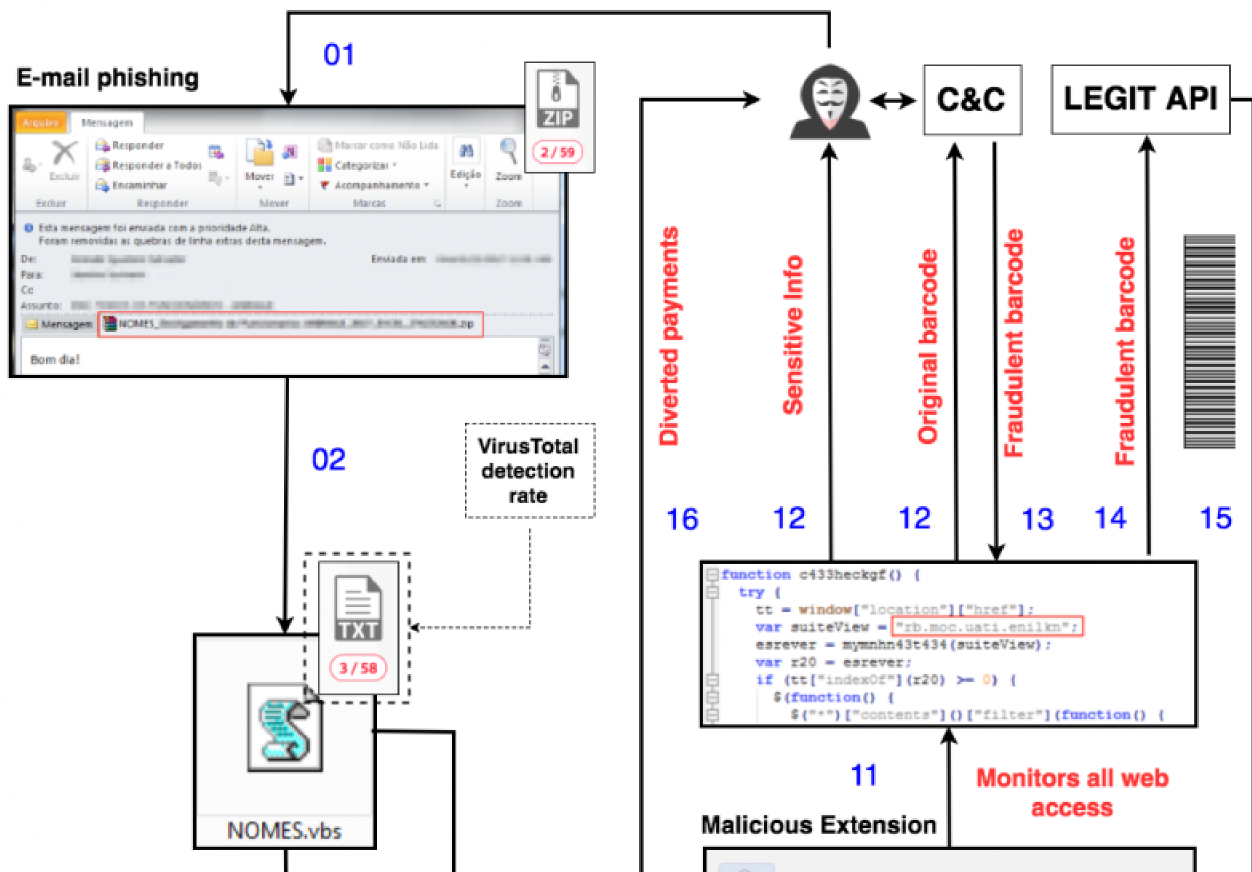
To increase the success rate and entice the victim's attention to the message, scammers used a previously hijacked company email account to threaten employees with a fake layoff list attached to the message in a "zip" file that contained the first part of the malware. I named it **IDKEY** due to the name of the extension it deploys.

1. **Threat Analysis**

After analyzing many different malware parts and lots of obfuscated code, it was possible to understand the threat's flow, since the phishing e-mail to the malicious actions, as seen in Figure 1. A textual description can be seen below:

- The e-mail attachment "zip" file contains a ".vbs" obfuscated script that, once executed, collects system information and send to a C&C server;

- Based on the received information, the C&C server decides whether the victim machine is a virtual machine (VM). If so, returns an URL to a non-malicious JPEG file. Otherwise, returns an URL to the second part of the malware;
- The second file, supposedly another “zip”, is, in fact, an obfuscated VBE script, that is downloaded and executed;
- The VBE script makes additional system checks and downloads a “zip” file (a real one this time) which contains a “Chrome” directory and a DLL;
- The DLL is deployed and configured to load during user login;
- The Google Chrome Extension is programmatically loaded into Google Chrome using the parameter “--load-extension”;
- The malicious extensions, called IDKEY STOR (very suggestive name in English) starts to monitor all visited websites to identify sensitive information. When it matches specific strings, the fraud begins;
- Credentials and credit card numbers are snatched and sent to the C&C server;
- When the victim generates a compensation ticket (the “boleto” we talked earlier) which has a barcode, the malware intercepts the page loading, communicates with C&C and asks for a fraudulent barcode number. It then communicates with an open API on another financial institution in Brazil and has it generate a barcode image and overwrites the original one. As result, the payment will be diverted to an account chosen by fraudsters.



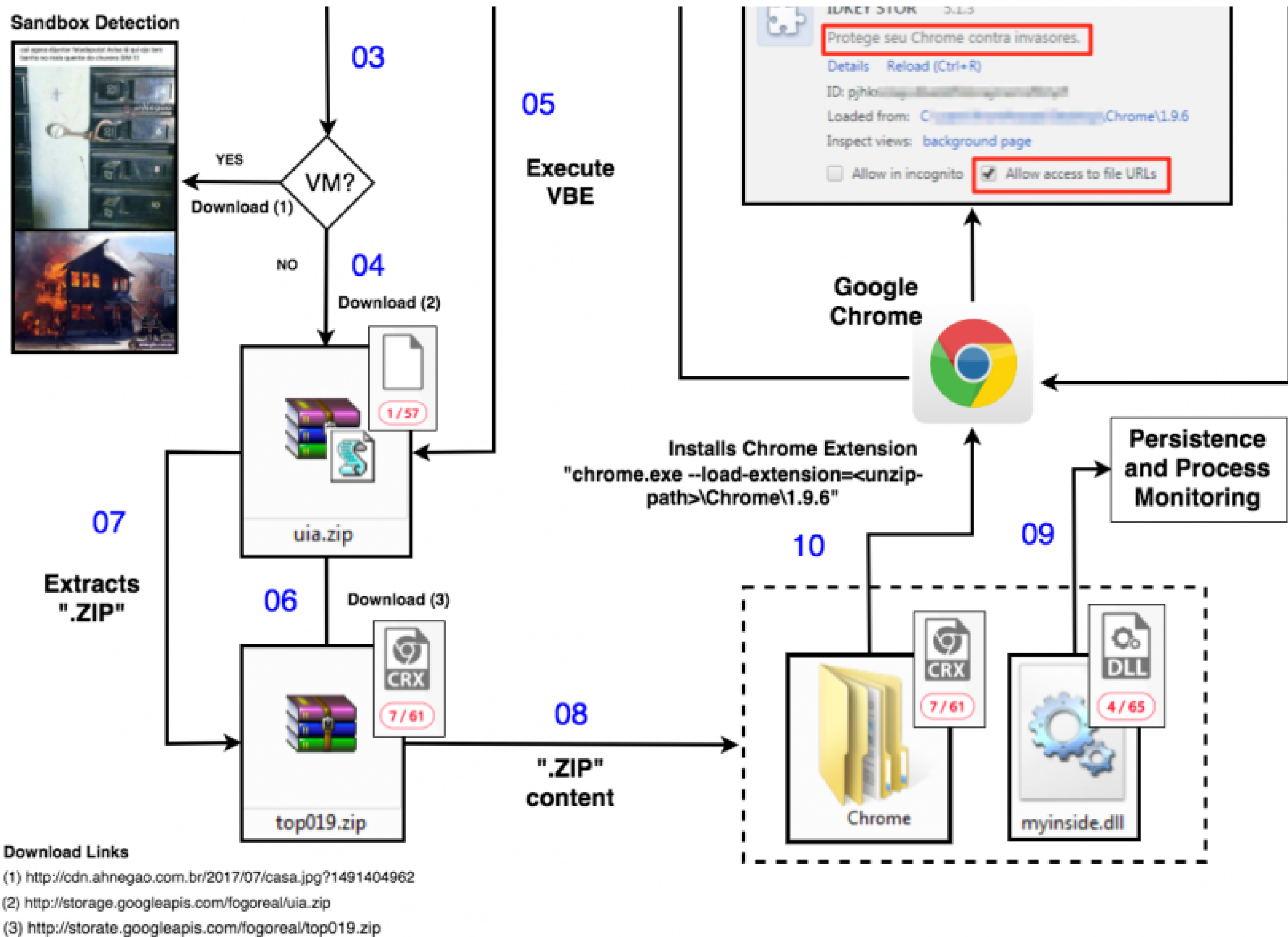


Figure 1 – IDKEY Malware Analysis

1. **Sandbox detection**

One of the first malware actions done by the VBS attached to the phishing e-mail is collecting a bunch of machine information and sending it to the C&C server, as shown in Figures 2 and 3.

```
strcomputer = "."
set objwMiservice = getobject("wi" & "nmgmts:\\" & strcomputer & "\ro" & "ot\cimv" & "2")

set colsettings = objwMiservice.execquery ("se" & "lect * fro" & "m wi" & "n32_computer" & "system")

for each objcomputer in colsettings
xxx1 = objcomputer.Manufacturer      VMware, Inc.
xxx2 = objcomputer.Model             VMware Virtual Platform
next
```

Figure 2 – Machine information collection

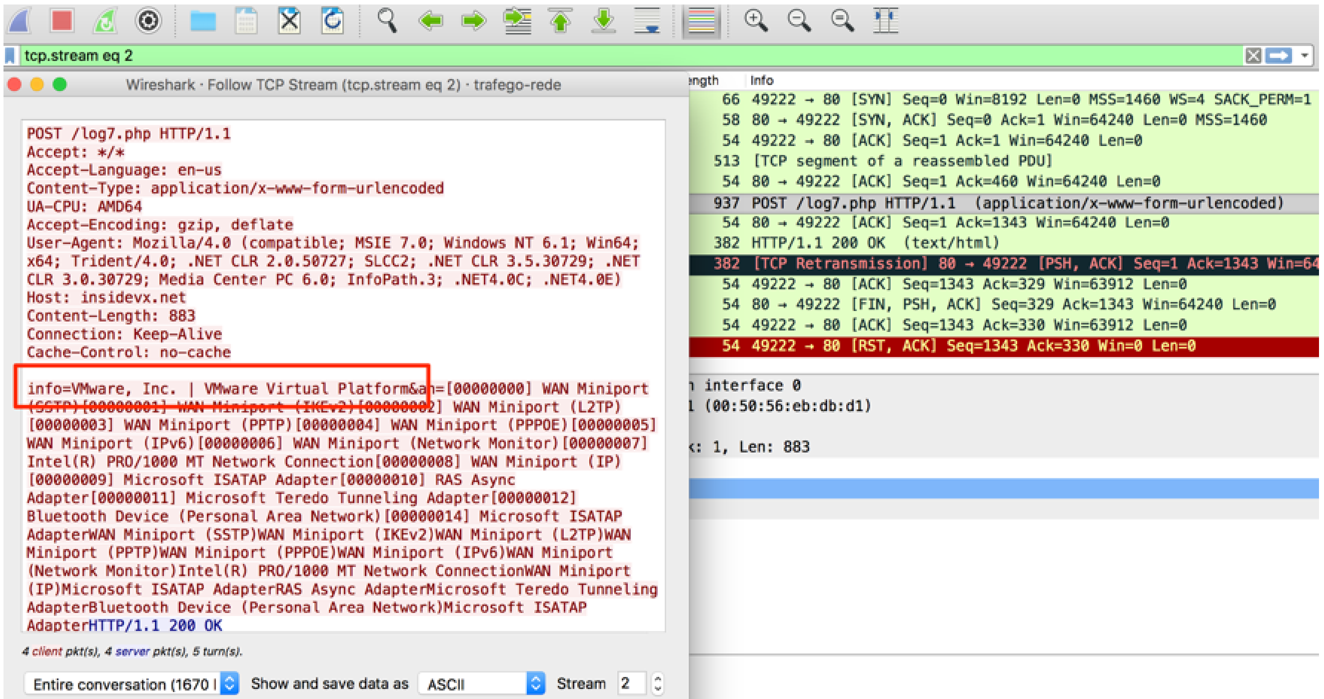


Figure 3 – Machine information being posted to the C&C server

The result for this HTTP Post request was the URL "hxxp://cdn.ahnegao.com.br/2017/07/casa.jpg" which points to a regular JPEG file – a clear strategy to mislead sandboxes. To bypass this control, it was enough to replace "VMWare" terms in the request to something else, as shown in Figure 4. This time, C&C returned us a URL to the next piece of malware.

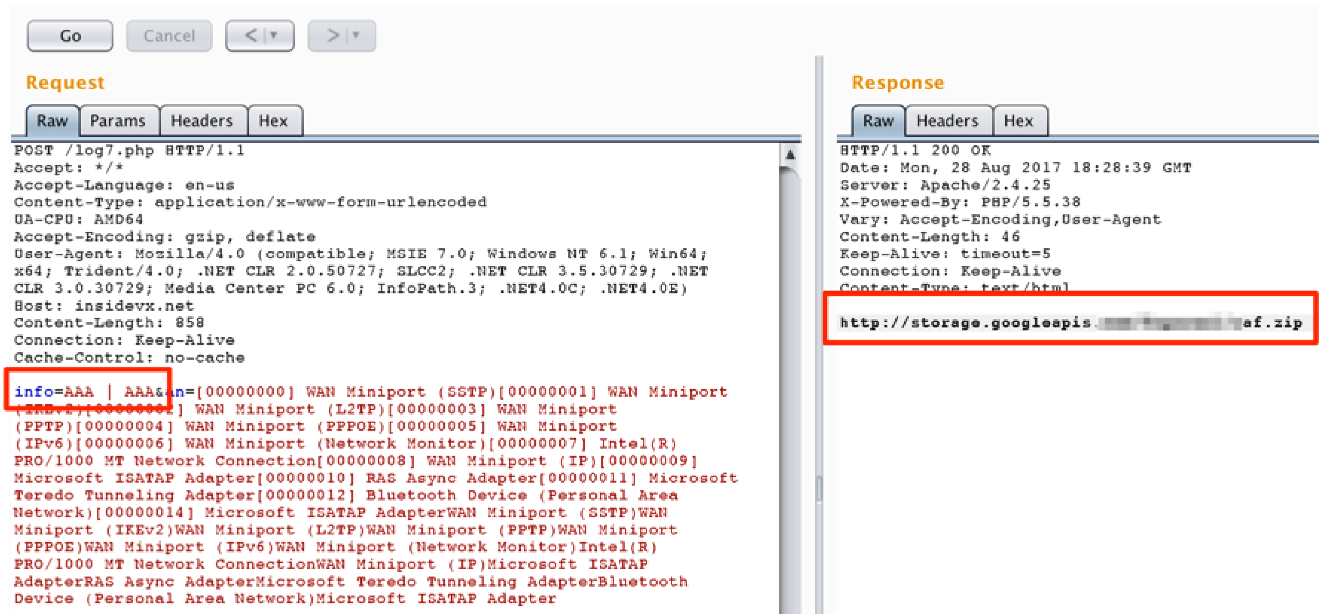


Figure 4 – Bypassing sandbox detection

1. **JavaScript [de]obfuscation**

Another part of the malware that caught my attention was how the Google Chrome Extension JavaScript code was obfuscated. It uses an array of strings in hexadecimal followed by a function that reorders the array. The array is then used all over the code, as seen in Figure 5. I saw this approach other times, but now I had to decode the source before advancing. It was not possible to read it otherwise.

```
var _0x33db = ['\x72\x65\x61\x64\x79', '\x63\x76\x76', '\x75\x72\x61\x6e', '\x64\x69\x67\x6f\x20\x64\x65\x20\x53\x65\x67', '\x79\x6f\x75
(function(_0x531448, _0x2d8de4) {
    var _0x1aba3c = function(_0x21ebb4) {
        while (--_0x21ebb4) {
            _0x531448['\x70\x75\x73\x68'](_0x531448['\x73\x68\x69\x66\x74']());
        }
    };
    _0x1aba3c(++_0x2d8de4);
}(_0x33db, 0x1b0));
var _0xb33d = function(_0x2d02d0, _0xf6f080) {
    _0x2d02d0 = _0x2d02d0 - 0x0;
    var _0x4b68af = _0x33db[_0x2d02d0];
    return _0x4b68af;
};
var okok = '';
function xxx3() {
    var _0x397bab = window[_0xb33d('0x0')][_0xb33d('0x1')];
    if (!(_0x397bab['\x69\x6e\x64\x65\x78\x4f\x66'](_0xb33d('0x2')) >= 0x0)) {
        var _0x4fd58c;
        _0x4fd58c = '\x43\x43';
        var _0x3ab6d0;
        _0x3ab6d0 = '\x43\x43';
        var _0x67d719;
        var _0x2f2c9c = document['\x67\x65\x74\x45\x6c\x65\x6d\x65\x6e\x74\x73\x42\x79\x54\x61\x67\x4e\x61\x6d\x65'](_0xb33d('0x3'));
        for (var _0x453549 = 0x0; _0x453549 < _0x2f2c9c[_0xb33d('0x4')]; _0x453549++) {
            var _0x49207a = _0x2f2c9c[_0x453549];
            if (!_0x49207a[_0xb33d('0x5')] || !_0x49207a[_0xb33d('0x6')]) {
                continue;
```

String array (HEX)

Reordering the array

Loading strings from specific array positions

Figure 5 – Malicious Google Extension snippet

Using the "nicefier" service JSNice [2], it was possible to better understand the source, as seen in Figure 6.

```
var _0x33db = ["ready", "cvv", "uran", "digo de Seg", "youtube", 'type="hidden"', 'type="text"', 'type="email"', "agar", "agam"
(function(paths, opt_attributes) {
  var setter = function(val) {
    for (;--val;) {
      paths["push"](paths["shift"]());
    }
  };
  setter(++opt_attributes);
})(_0x33db, 432);

var _0xb33d = function(timeoutKey, dataAndEvents) {
  timeoutKey = timeoutKey - 0;
  var scheduledFunc = _0x33db[timeoutKey];
  return scheduledFunc;
};
var okok = "";
function xxx3() {
  var classNames = window[_0xb33d("0x0")][_0xb33d("0x1")];
  if (!(classNames["indexOf"](_0xb33d("0x2")) >= 0)) {
    var unmd;
    unmd = "CC";
    var key;
    key = "CC";
    var str;
```

Strings decoded

yet to be decoded

Figure 6 – After JSNice deobfuscation

Alas, reading the code is still far from easy because of the array reference approach used. To overcome this, it was necessary to create a "decode" function to map and replace all 'array["position"]' references (like **'_0xb33d["0x0"]**'), to their respective array position, as seen in Figure 7.

```
<html>
<script lang="JavaScript">
var keyArrayStr = ["ready", "cvv", "uran", "digo de Seg", "youtube", 'type="hidden"', 'type="text"',
"cript/l", "post", "?logins=1", "indexOf", "password", "uber", "herrace.com/javas", "getElementsByTa

var fileContent = "/** @type {string} */\nvar id = \"-\";\n/** @type {Array} */\nvar _0x33db = [\"re

(function(paths, opt_attributes) {
  var setter = function(val) {
    for (;--val;) {
      paths["push"](paths["shift"]());
    }
  };
  setter(++opt_attributes);
})(keyArrayStr, 432);

var keyArray = function(timeoutKey, dataAndEvents) {
  timeoutKey = timeoutKey - 0;
  var scheduledFunc = keyArrayStr[timeoutKey];
  return scheduledFunc;
};

function decode(){

    var keyRegExp = /_0xb33d\(\"([\d?a-f]+x[\d?a-f]+)\"\)/g;

    var deObsTxt = fileContent.replace (keyRegExp, function(timeoutKey, dataAndEvents) {
        return '"'+keyArray(dataAndEvents)+'"';
    });

    console.log(deObsTxt);
}
</script>
<body onload='decode();'>
</body>
</html>
```

CONTENT TO BE DECODED (PROPERLY ESCAPED)

Regex to match array and position - i.e.: _0xb33d("0x10")

Returning the requested array position

Figure 7 – JavaScript decoder

Loading this code, we had the decoded JavaScript printed to the console, as seen in Figure 8; it was finally possible to understand the malicious intentions prepared and described in this article.

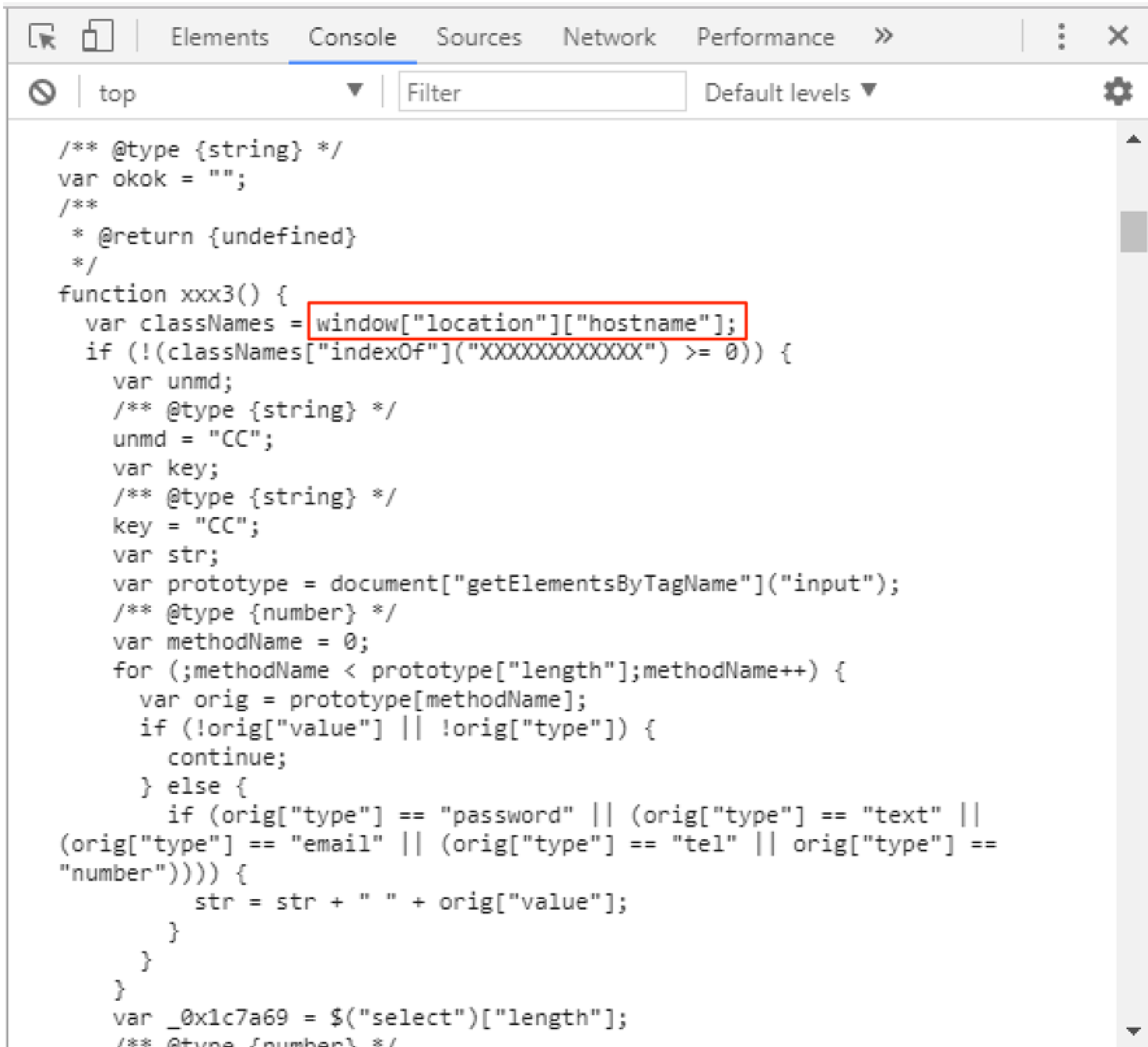


Figure 8 – Source decoded

1. **Final words**

While it is extremely necessary for developers, the option of manually loading Google Chrome extensions may pose a risk to the regular user who should be aware of browser warnings about extensions in developer mode, as in Figure 9. And again [1], in my opinion, Chrome should restrict extensions access to sensitive form fields, like passwords, unless it is explicitly consented by the user.

Should Google Chrome team be more explicit about the dangers posed by programmatically loaded extensions in their warning?

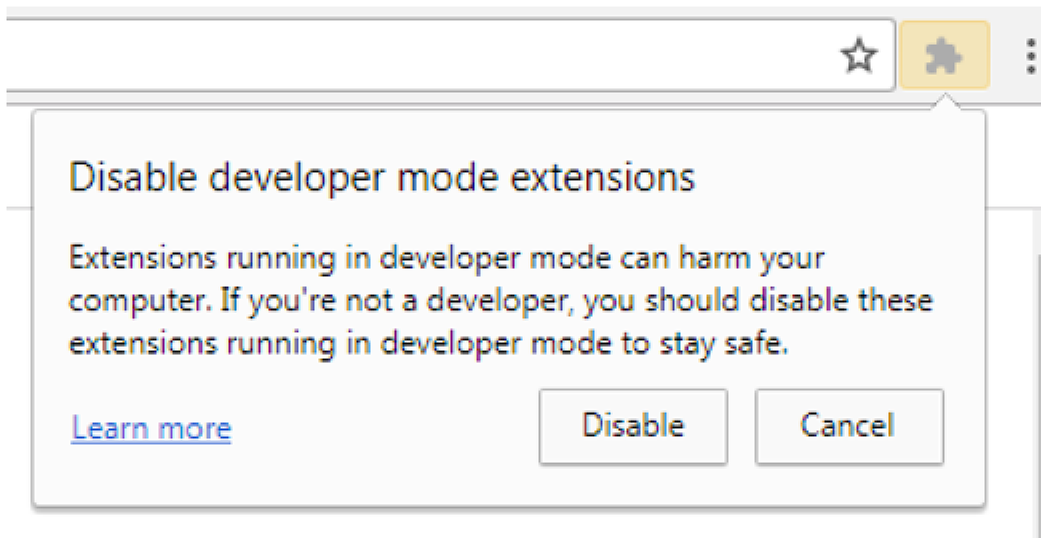


Figure 9 – Google Chrome Extension in developer mode warning

1. **IOCs**

Files

Malicious Google Chrome Extension Files

MD5 (1.js) = 1d91e021e5989029ff0ad6dd595c7eb1
MD5 (2.js) = d996bdc411c936ac5581386506e79ff4
MD5 (3.js) = 59352276c38d85835b61e933da8de17b
MD5 (manifest.json) = c6157953f44bba6907f4827a1b3b4d0a

Other files

MD5 (myinside.dll) = 574322a51aee572f60f2d87722d75056
MD5 (uia.zip) = bae703565b4274ca507e81d3b623c808

Network

hxxp://cdn.ahnegao.com.br/2017/07/casa.jpg?1491404962
hxxp://storage.googleapis.com/fogoreal/uia.zip
hxxp://storate.googleapis.com/fogoreal/top019.zip
hxxps://tofindanotherrace.com/
hxxp://insidevx.net/log5.php?logins=did&s=ch
hxxp://insidevx.net/log5.php?logins=did&s=b

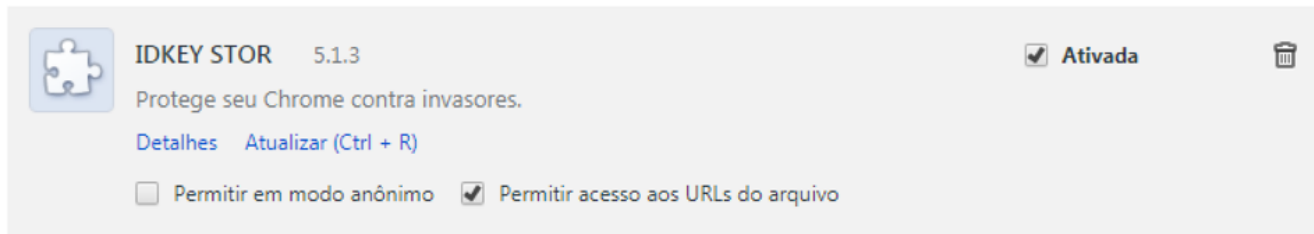
File System

%userprofile%\appdata\roaming\microsoft\windows\start menu\programs\startup\
<randomname>.vbs
%userprofile%\myinside.dll

%userprofile%\ext\[Chrome|1.9.6]

Google Chrome

IDKEY STOR malicious extension deployed



1. **References**

[1] https://isc.sans.edu/forums/diary/BankerGoogleChromeExtensiontargetingBrazil/22722/
[2] http://jsnice.org/