Kimsuky Deploys TRANSLATEXT Chrome Extension |ThreatLabz

By Seongsu Park

Published: 2024-06-27 · Archived: 2026-04-05 19:14:37 UTC

Technical Analysis

According to a recent publication by a South Korean security vendor, Kimsuky delivered an archive file named “한국군사학논집 심사평서 (1).zip”, which translates to "Review of a Monograph on Korean Military History."

The archive contains two decoy files:

- HWP documents (a popular office file format in South Korea)
- A Windows executable masquerading as related documents

When a user launches the executable, the malware retrieves a PowerShell script from the threat actor’s server. The figure below shows the Kimsuky infection chain.

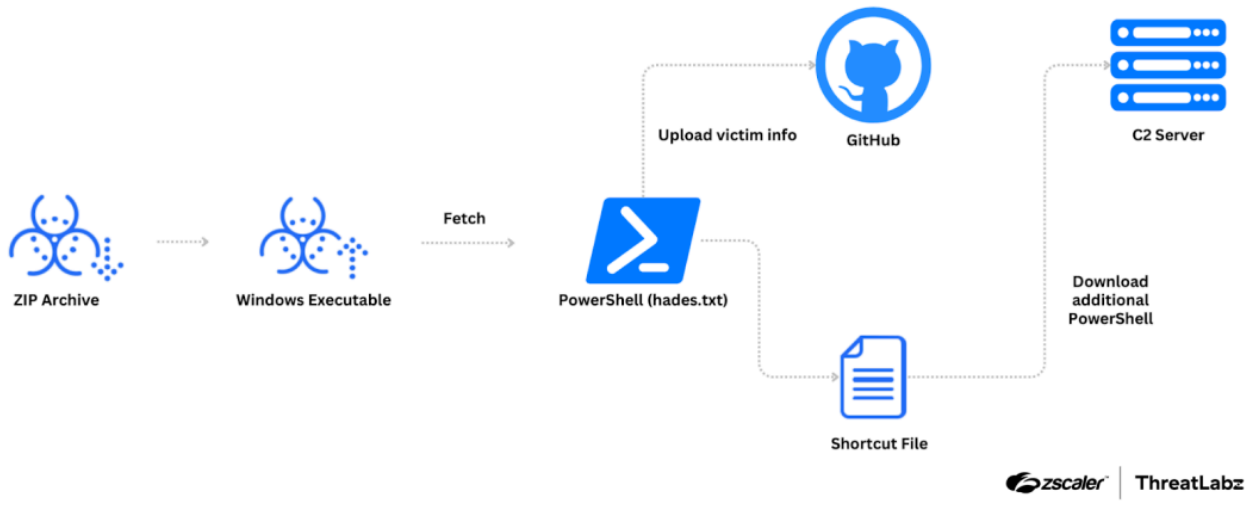

Figure 1: Example Kimsuky infection chain.

The PowerShell script from the remote server is responsible for uploading general information about the victim and creating a Windows shortcut that retrieves an additional PowerShell script from the same server. During our own research into this campaign, we discovered another PowerShell script with the MD5 hash: `bba3b15bad6b5a80ab9fa9a49b643658` and a GitHub account used by the script linked to the same actor. From this newly discovered GitHub account, we observed victim data and a previously deleted Chrome extension utilized by the actor. The delivery method for `TRANSLATEXT` is not currently known.

However, the newly discovered PowerShell script reveals that Kimsuky checked for the presence of installed Chrome extensions using the Windows registry key shown below:

```
HKCU\Software\Policies\Google\Chrome\ExtensionInstallForcelist
```

This registry key is used by Chrome to enforce the installation of specified extensions without user permission or intervention. Therefore, it appears Kimsuky registered `TRANSLATEXT` in this registry key using previous stage methods.

TRANSLATEXT analysis

In the attacker-controlled GitHub account, we observed an XML file in addition to `TRANSLATEXT`. These files were present in the repository on March 7, 2024, and deleted the next day, implying that Kimsuky intended to minimize exposure and use the malware for a short period to target specific individuals.

The figure below shows how Kimsuky uploaded the files on March 7th to one of their GitHub accounts and then deleted them on March 8th.

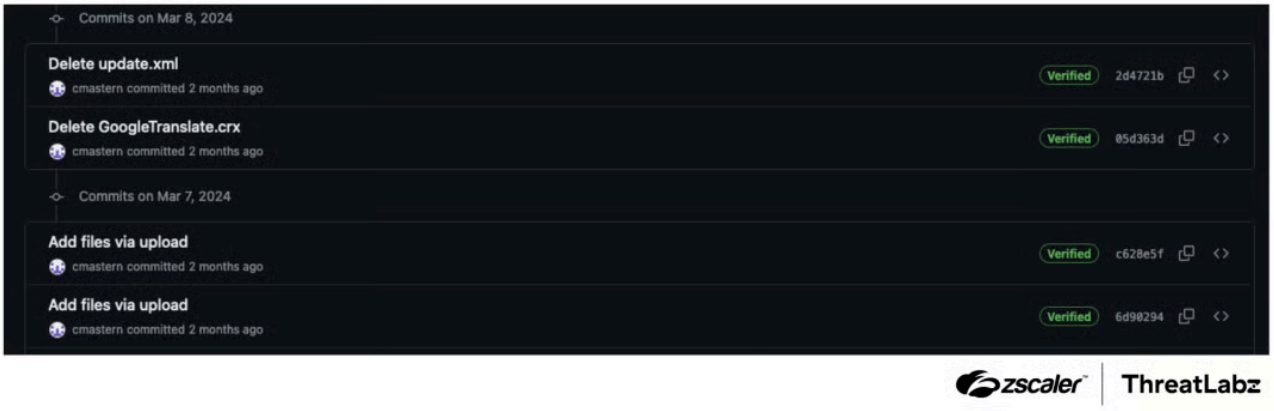

Figure 2: Kimsuky GitHub commit log shows the addition and removal of an XML file and `TRANSLATEXT` after only one day.

A timeline of the GitHub user's activity is listed below:

- **February 13, 2024:** Join GitHub
- **March 7, 2024:** Created first repository named "`motorcycle`"
  - 29 commits including uploads from the victim and subsequent removals.
  - Added `TRANSLATEXT` files: `update.xml`, `GoogleTranslate.crx`
- **Mar 8, 2024:** Removed `update.xml` and `GoogleTranslate.crx`
- **Mar 18, 2024:** Created `motorcycle/calc`
- **Apr 4, 2024:** Created a `motorcycle/laxi/ter.txt` that contains "`sfsadfsadfa`".

As the name suggests, the `update.xml` file contained the parameters necessary for updating `TRANSLATEXT` as shown below.

`TRANSLATEXT` was uploaded to GitHub as "`GoogleTranslate.crx`", and masqueraded as a Google Translate extension. However, `TRANSLATEXT` actually contained four malicious Javascript files for bypassing security measures, stealing email addresses, credentials, cookies, capturing browser screenshots, and exfiltrating stolen data.

The figure below depicts the role of each Javascript file in stealing and sending information to the C2 server.

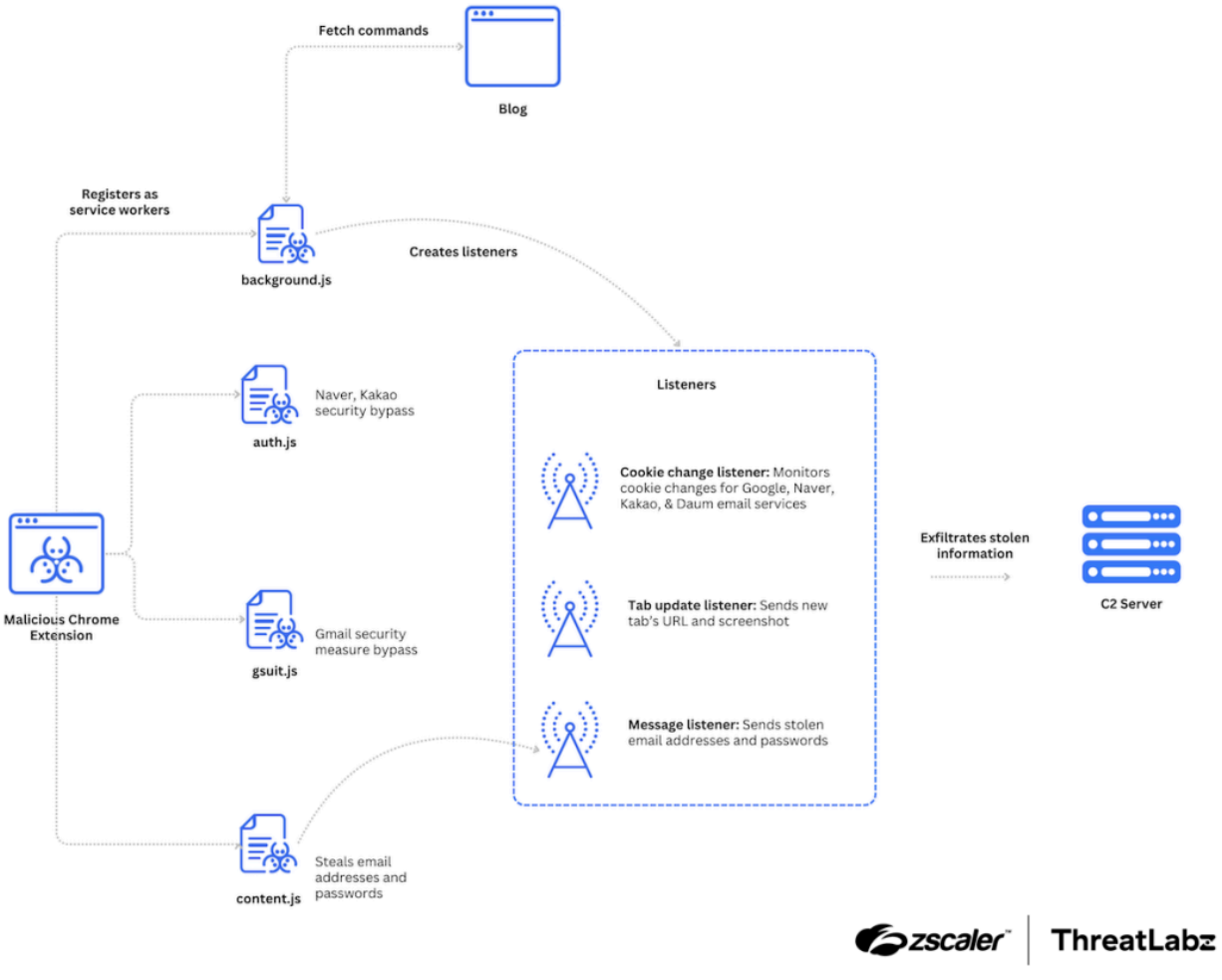

Figure 3: Kimsuky `TRANSLATEXT` architecture.

According to the `manifest.json` file, the author name is listed as "`Piano`", and the `update_url` points to another GitHub address referencing an `update.xml` file that did not exist at the time of our analysis. The description and default title fields contain Korean, which likely indicates that this campaign was specifically targeting South Korea–we discuss this later in the blog.

A part of the `manifest.json` file is shown below.

```
{
  // Required
  "author": "Piano",
  "manifest_version": 3,
  "name": "Google Translate",
  "version": "1.5.2",

  // Recommended
  "action": {
      "default_icon": "icons/16.png",
      "default_title": "번역하려면 마우스 왼쪽 버튼을 클릭하세요."
  },
  "description": "웹을 탐색하면서 편하게 번역을 볼 수 있습니다. 이 기능은 Google 번역팀에서 제공합니다.",
  "icons":{
      "16": "icons/16.png",
      "19": "icons/19.png",
      "32": "icons/32.png",
      "38": "icons/38.png",
      "48": "icons/48.png",
      "128": "icons/128.png"
  },
  "update_url": "https://raw.githubusercontent.com/HelperDav/Web/main/update.xml",

  // Optional
  "background": {
      "service_worker": "background.js"
  },
  "content_security_policy": {
      "extension_page": "script-src 'self' 'wasm-unsafe-eval'; object-src 'self'"
  },
  "permissions": ["tabs", "activeTab", "cookies", "storage", "downloads", "scripting"],
```

The `TRANSLATEXT` manifest requests excessive permissions such as **scripting**. This broad permission allows `TRANSLATEXT` to inject scripts into web pages, enabling it to modify page content, add functionality, and/or interact with the page's elements.

Depending on the URL the victim visits, a corresponding script is launched.

- When the victim visits the Naver login page (nid.naver.com/*) or the Kakao login page (accounts.kakao.com/*), the `auth.js` file is injected into the web page.
- Similarly, when visiting the Gmail login page (mail.google.com/), the `gsuit.js` file is injected into the web page.

The `content.js` script is injected into all web pages using the manifest file as shown below.

```
 "content_scripts": [
       {
           "js": [ "content.js"],
           "matches": [
               "http://*/*", "https://*/*"
           ],
           "run_at": "document_idle",
           "all_frames": false
       },
       {
           "js": [ "auth.js"],
           "matches": [
               "https://nid.naver.com/*",
               "https://accounts.kakao.com/*"
           ],
           "run_at": "document_end",
           "all_frames": false
       },
       {
           "js": [ "gsuit.js"],
           "matches": [
               "https://mail.google.com/*"
           ],
           "run_at": "document_end",
           "all_frames": false
       }
   ]
```

Security bypass

The script injected into the web page is responsible for bypassing security measures on each specific login page.

Note: For security reasons, we've replaced sensitive variable names in the script to prevent unauthorized actors from exploiting these methods.

The `gsuit.js` script searches for all `` `elements with the specific class name in the web page and then removes them from the Document Object Model (DOM) as shown below.`

```
   "use strict";
function NeverNotify()
{
   var x = document.querySelectorAll("[redacted]");
   for(var i=0; i {NeverNotify();}, 50);
```

The `auth.js` script is used for manipulating security measures for Naver and Kakao. To bypass Kakao, the script checks for elements with specific IDs. If these elements exist, the script clicks them. This action typically means opting to remember the browser to avoid repeated security prompts. The script selects all elements and ensures their class names are set correctly, possibly to ensure all checkboxes of this type are checked.

The Naver section of the script, similar to the Kakao section, identifies elements with specific IDs and performs clicks on them. These clicks serve various purposes, such as skipping or acknowledging waiting times and dialogs within Naver's security measure process. For instance, it locates an element with the ID `auto` and sets its value to `init`, potentially as part of a setup or initialization process for the authentication page.

Note: We have notified the Google and Naver security teams about these security bypasses and are closely working with them to mitigate the issue.

Email address stealer - content.js

The main objective of this Javascript file is to collect email address and password data entered into the forms and send the information to a background page. The script performs these actions as follows:

- Hooking into various form elements such as buttons and input fields to capture clicks and keypresses to initiate sending data.
- Collecting all email addresses entered into any input fields (“ `type=email` ”), general text (“ `type=text` ”), or textboxes (“ `role=textbox` ”), and concatenating them into a single string.
- Collecting values from all input fields of the type password, and concatenating the email address and password data collected into a string format suitable for transmission.
- Monitoring user actions, like pressing Enter, by adding event listeners to various button types and input fields. It uses a mutex variable to prevent multiple transmissions at the same time. This monitoring process is repeated every 500 milliseconds, ensuring new elements on the page or dynamically added elements are also monitored.

Service worker - background.js

The Javascript employs the dead drop resolver technique to retrieve configurations and commands from the public blog service:

```
hxxps://onewithshare.blogspot[.]com/2023/04/10.html
```

If the blog URL is active, the Javascript extracts the pattern with the following regular expression:

```

```

This parses the content from the `value` parameter of a hidden input field. When we originally checked the blog, there were no relevant values present in this format. However, on July 1, 2024, another researcher identified an update to the threat actor's blog with the `Capture` command, indicating that the threat actor is actively managing the campaign. The figure below shows the threat actor's blog updated with the `Capture` command.

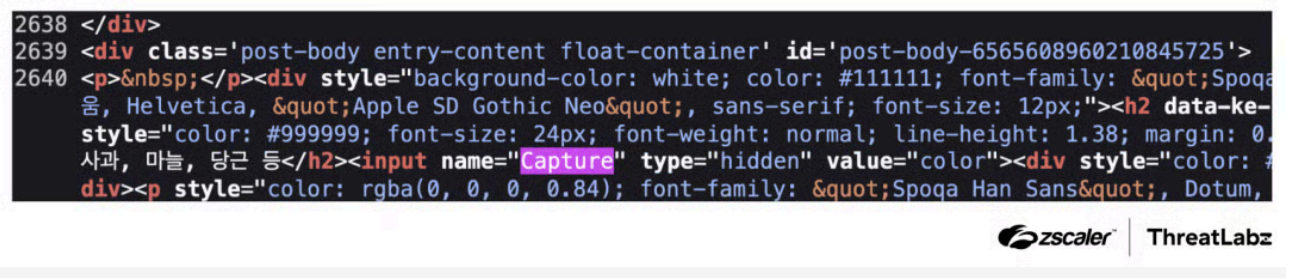

Figure 4: The threat actor's blog's updated with the `Capture` command on July 1, 2024.

There are four types of commands expected by the code, and they are described in the table below:

Command	Description
`URL`	Parses and Base64 decodes the value and appends `/log.php`. This newly formed `URL` is used as a new C2 server.
`Capture`	When a new tab is created, the code sends the current time and URL of the tab, taking a screenshot of the tab with `chrome.tabs.captureVisibleTab` API every 5 seconds.
`delcookie`	Removes all cookies from the browser.
`Run`	Injects a `` tag with the href value `ms-powerpoint://` in all Chrome tabs, invoking the click event every 30 minutes.

Table 1: Commands supported by Kimsuky's `TRANSLATEXT`.

The background script also registers several listeners with specific functionality as described below:

- Send background Javascript listener: This listener is triggered when a new message is created, allowing for appropriate actions to be taken in response.
- Tab update listener: When a tab is updated, this listener sends the URL of the newly created tab along with a screenshot, based on the presence of the `Capture` flag.
- Cookie change listener: Whenever a cookie is modified, this listener checks if the domain includes `google`, `naver`, `kakao`, or `daum`, and if the reason for the change is `expired`, `evicted`, or `explicit`. In such cases, the new cookie value is sent to the remote C2 server.

`TRANSLATEXT` uses HTTP POST requests for C2 communications, with the following hardcoded HTTP headers:

```
 Accept: application/json, application/xml, text/plain, text/html, *.*,
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Access-Control-Allow-Origin: "*"
Access-Control-Allow-Credentials: true
```

`TRANSLATEXT` uses the following HTTP POST fields for sending the stolen information.

Data to Send	POST Data Format

Email/password	`event=[current time]-->`
	`event=[url]`
	`event=email=[email]**pwd=[passwd]`

New tab image	`tab=[current time]-->`
	`tab=[url]`
	`image=[image data]&url=[tab url]`

Cookie (send all cookies)	`cookie=[current time]-->` `cookie=[all cookie value]`
Cookie (cookie changed)	`cookie={expired|evicted|explicit}:[current time]-->` `cookie=[cookie value]`

Table 2: HTTP POST data format for Kimsuky's `TRANSLATEXT`.

Explore more Zscaler blogs

Source: https://www.zscaler.com/blogs/security-research/kimsuky-deploys-translatext-target-south-korean-academia