Notepad++ Plugins and security concerns

Arjun (@247arjun)

Summary

Notepad++ is a very popular open-source text editor for Windows, that also supports syntax highlighting for a wide range of programming languages. This makes it very useful in software engineering organizations of all sizes, as many consider it to be the “VLC Player” equivalent for source code files. The Plugins that it supports can be used to mount a supply chain attack on any organization that uses the tool, primarily due to missing safeguards.

Overview of Plugins

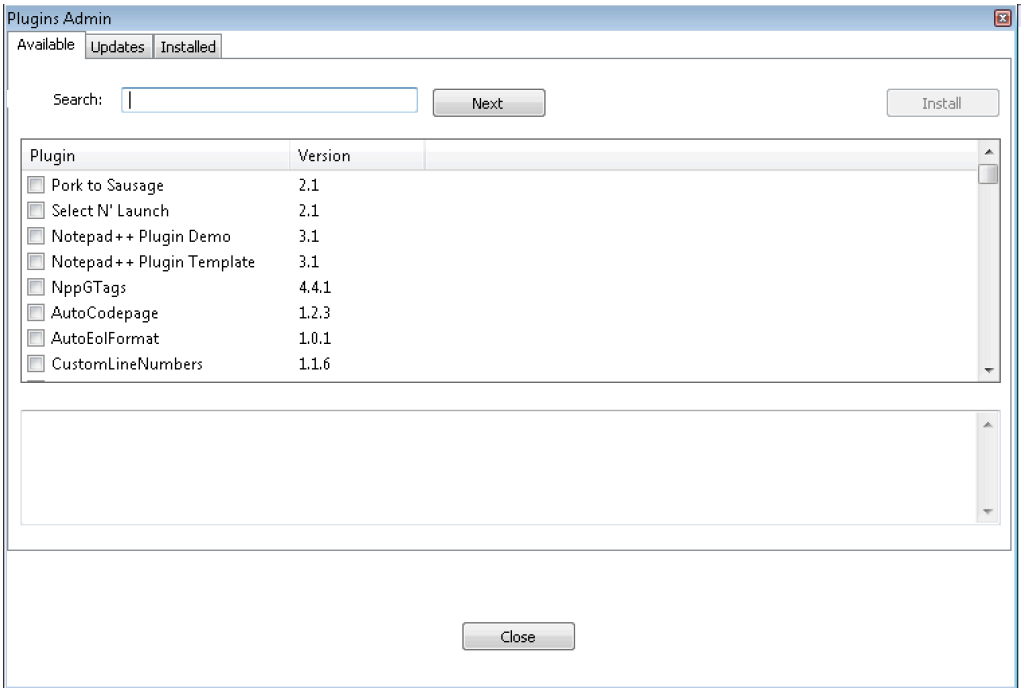
Notepad++ allows for additional functionality to be added to the editor, by means of Plugin support. A plugin is a DLL that is loaded by the Notepad++ process (notepad++.exe) to provide the functionality advertised by the Plugin.

https://npp-user-manual.org/docs/plugins/

Anyone can create a plugin, following the documentation provided by Notepad++. Indeed, at the time of this blog, there were 160 community submitted plugins to the editor.

Where do you get Plugins

Plugins are just DLL files that Notepad++ loads, so there are a few ways a user can get them.



Officially curated Plugins

Notepad++ includes a “Plugin Admin” interface that allows a user to view and install supported plugins to the editor.

Unofficially added Plugins

Users can also manually copy-paste plugin DLLs to paths that Notepad++ will load them from.

Curation of Plugins

Plugins listed in the “Plugin Admin” interface are curated in a separate open-source GitHub nppPluginList repo. This repo contains a list of all plugins for 32-bit and 64-bit Notepad++, and this information is used to populate the list of plugins a user will se in the Plugin Admin UI inside Notepad++.

https://github.com/notepad-plus-plus/nppPluginList

This GitHub repo accepts Pull Requests from Plugin maintainers, who wish to include their plugins in Notepad++. The information necessary to include a Plugin is:

- folder-name (The name of the folder inside which the DLL resides)
- display-name (The name of the Plugin, as seen inside Plugin Admin)
- version (A string representing the versioning of the Plugin)
- id (The SHA-256 hash of the plugin ZIP)
- repository (the URL at which a ZIP file containing the DLL is found)
- author (Plugin author information)
- homepage (A website for the Plugin)

For example, the JSON object for the “JSON Viewer” Plugin looks like:

```
{
        "folder-name": "NPPJSONViewer",
        "display-name": "JSON Viewer",
        "version": "1.40",
        "id": "15feafd549e6f6916af0be81089cd297245c43fff25dab3076d69b41fbc990e2",
        "repository": "https://github.com/kapilratnani/JSON-Viewer/releases/download/v1.40/NPPJSONViewer_Win32.zip",
        "description": "JSON viewer that displays the selected JSON string in a tree view.",
        "author": "Kapil Ratnani",
        "homepage": "https://github.com/kapilratnani/JSON-Viewer"
},
```

When an update to an existing plugin is made, it is expected for the PR to include a new version, id and repository URL. For example, when the JSON Viewer Plugin updated to v1.40 in August 2020, the PR commit looked like this:

```
"version": "1.34",
"id": "725241aef0b2e56e4f2a10da5eb714b6581384e4ae5f26c4d6f35eab49c3a979",
"repository": "https://github.com/kapilratnani/JSON-Viewer/releases/download/1.34/NPPJSONViewer_x64.zip",
"version": "1.40",
"id": "8c1143e13f3d1429b9f625ddbed8fdfbeef86b4d01978e5328f27d9e7ac2662a",
"repository": "https://github.com/kapilratnani/JSON-Viewer/releases/download/v1.40/NPPJSONViewer_x64.zip",
```

Safeguards

When a user chooses to install a Plugin, Notepad++ computes the SHA-256 hash of a Plugin ZIP when it is downloaded, and compares it to the expected hash as submitted by the Plugin maintainer. Any mismatch will reject the installation of the plugin.

Missing Safeguards

Notepad++ leaves Plugin validation up to the Plugin maintainers, and only validates that the file downloaded is the one the maintainers intended. This lack of oversight allows for Plugins with the following issues:

- Unsigned DLLs - some Plugins are unsigned, and these are loaded by Notepad++ without any prompts.
- No validation of Plugin maintainer provided data in Pull Requests - there have been instances when even the incorrect SHA-256 hash has not been validated. Additionally, repository/homepage URLs are not enforced as HTTPS URLs, potentially enabling MITM attacks.
- No validation of functionality - a Plugin could, in theory, have a different functionality than that advertised in the Plugin Admin UI.
- Unsafe option of loading Plugins from %APPDATA% - during install, a user can choose to enable an option to load Plugins from %APPDATA%, which is a Windows folder location that any process can write in to, without needing elevation.

Possible Attack Chain

An attacker could choose to include a new Plugin in the list, by submitting metadata for a malicious DLL (ZIP). Since this information is not vetted strongly, their Plugin could be exposed to users inside the Plugin Admin UI in Notepad++.

Alternatively, they could submit a PR to just change the SHA-256 hash of an existing Plugin, while capturing the repository location of the genuine Plugin - this can be due to account takeover, or an MITM intercepting attack on insecure HTTP URLs.

Given that a plugin has access to all data inside Notepad++ and the (potentially proprietary/confidential) files it has opened, it is possible to exfiltrate sensitive source code when a malicious Plugin is loaded.

In light of the SolarWinds Orion hack, where attackers had previously exfiltrated proprietary source code from the company, it may have been possible to get access to the source code using a malicious Notepad++ Plugin - if the company used Notepad++ as a source code editor in their software supply chain.