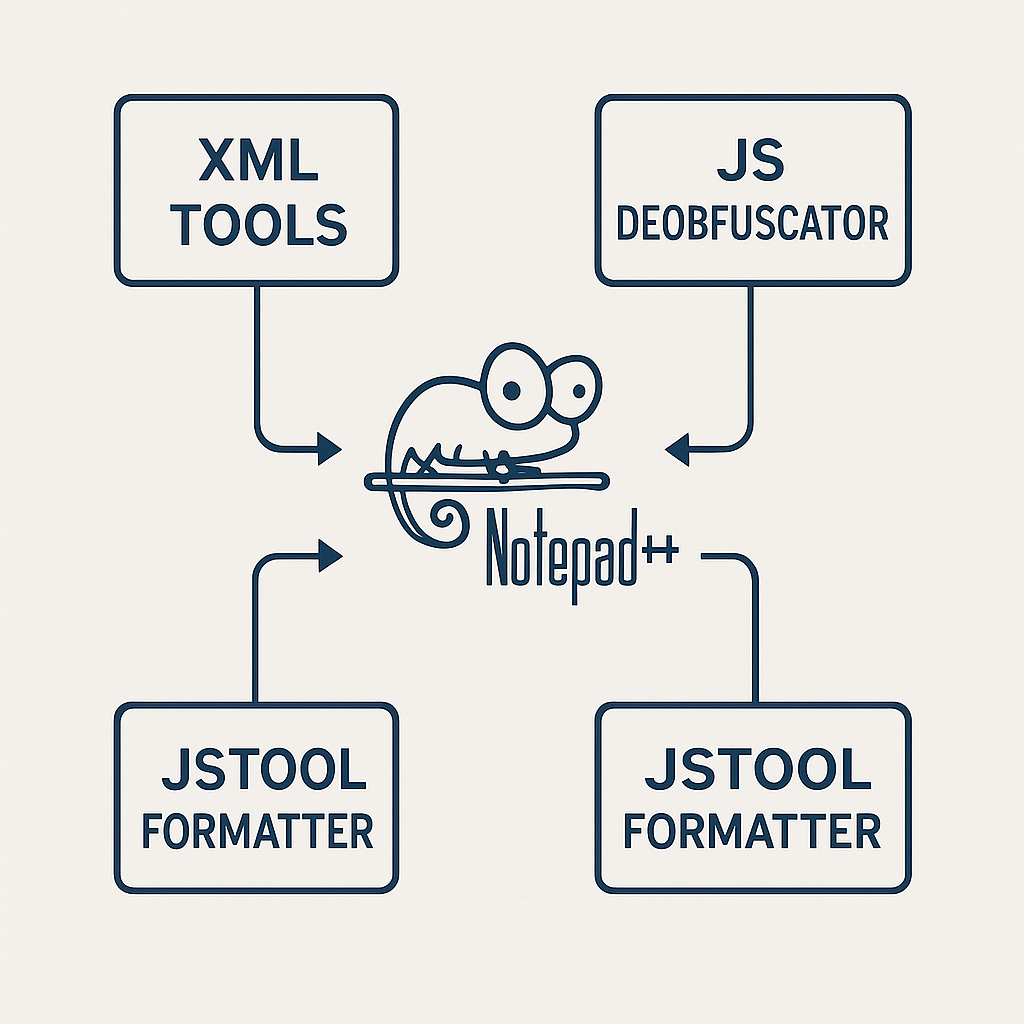
**Proof-of-Concept Suite: Notepad++ XML / JS De-obfuscation Toolkit**

**Name: Chaitanya Mayekar**

**Intern ID: 253**

***A hands-on lab book that shows how three free extensionsXML Tools, JSTool, and the Obfuscator-io-deobfuscator CLI—can be wired together inside Notepad++ to format XML, beautify JavaScript, and reverse Obfuscator.io payloads without ever leaving your favourite editor.***

**Before diving into individual demos, the following diagram summarises the ergonomic stack we will assemble.**

**Figure 1 – Plugin ecosystem overview: Notepad++ and the three key extensions used in this PoC**

**1 Environment Preparation**

**1.1 Base editor**

**Download the current 64-bit installer or portable build of Notepad++ (v8.6 or later) from the official site and launch it once so that the default folder hierarchy—including *plugins* and *plugins\Config*—is created.**

**1.2 XML Tools plugin**

1. **Open Plugins › Plugins Admin…, tick XML Tools, press *Install* and allow Notepad++ to restart.**
2. **If you are behind an air-gapped lab, drop XMLTools.dll plus its four support DLLs in  
   C:\Program Files\Notepad++\plugins\XMLTools\ and restart manually; the folder name must match the DLL name exactly or the loader will silently ignore it.**

**1.3 JSTool plugin (JavaScript beautifier/minifier)**

**Use Plugins Admin in the same way, or unzip the latest \*JSToolNPP.\*zip and copy JSToolNPP.dll into plugins\JSToolNPP\.**

**1.4 NppExec macro engine**

**Install NppExec from Plugins Admin. This adds a console window and a tiny scripting language that can call external programs, capture output, and even chain Notepad++ commands.**

**1.5 Obfuscator-io-deobfuscator CLI**

**Run the global NPM install:**

**bash**

**npm install -g obfuscator-io-deobfuscator**

**The binary obfuscator-io-deobfuscator will be on your system path and can transform an obfuscated file into readable JavaScript with**

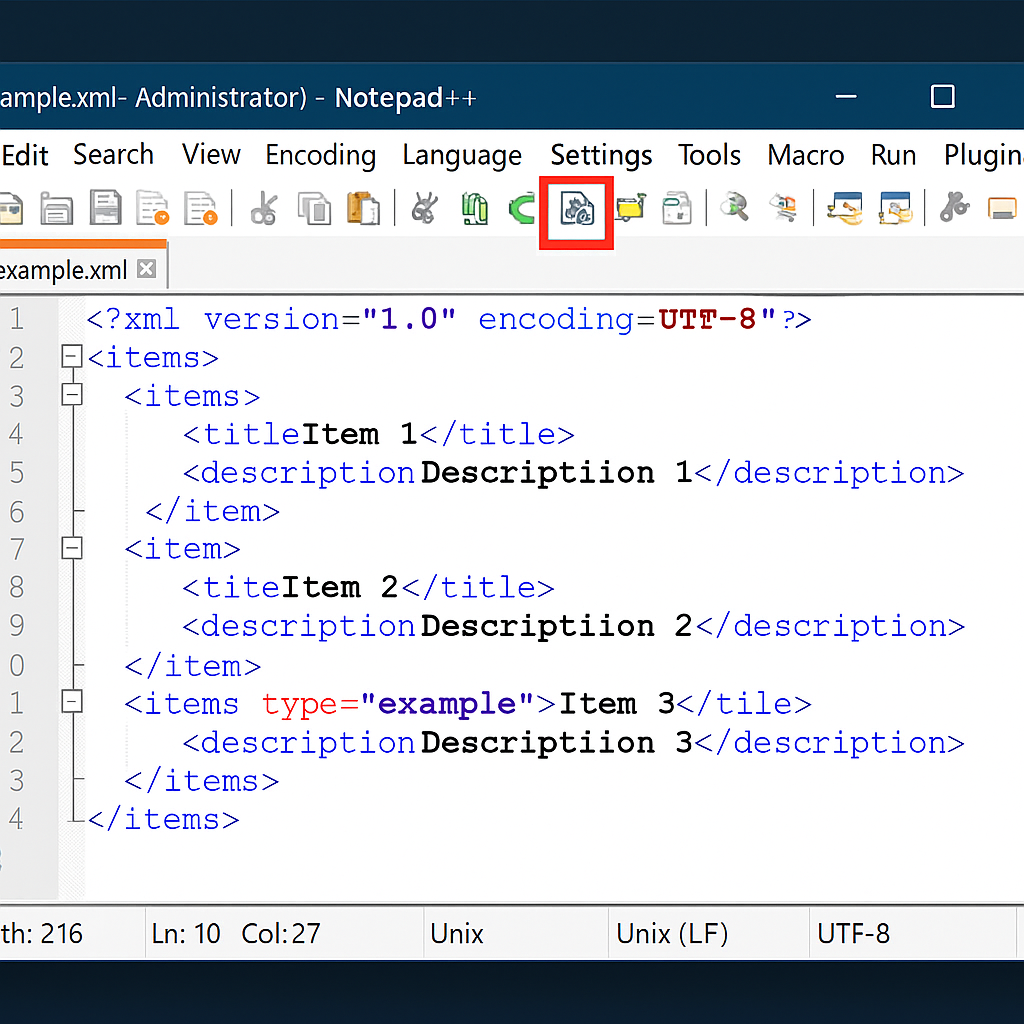
**bash**

**obfuscator-io-deobfuscator input.js -o clean.js**

**2 PoC #1 – Pretty-print any XML file with one keystroke**

**Objective Turn a single-line, hard-to-read XML blob into an indented document.**

1. **Load a messy XML file.**
2. **Press Ctrl + Alt + Shift + B (*Pretty Print – libXML*) or use Plugins › XML Tools › Pretty Print.**
3. **Well-formed markup is instantly re-indented; syntax errors raise a dialog pinpointing the faulty line.**

****

**Figure 2 – XMLTools 'Pretty Print' in action inside Notepad++**

***Troubleshooting* If nothing happens, validate the XML first (Check XML syntax now)—Pretty Print aborts on malformed documents.**

**3 PoC #2 – Quick JavaScript beautification with JSTool**

**Objective Expand minified / lightly obfuscated JavaScript so it is readable before deeper analysis.**

1. **Install JSTool as per §1.3.**
2. **Open a compacted .js file and hit Ctrl + Alt + M (*JSFormat*).**
3. **The plugin restructures the code, adds line breaks and braces, and colour-codes tokens, which makes static inspection or diffing far easier.**

**You can combine JSTool with the built-in View › Fold All to explore high-level functions quickly.**

**4 PoC #3 – Fully automated Obfuscator.io reversal inside Notepad++**

**Objective Hook the npm CLI into Notepad++ so that saving an obfuscated script transparently writes a de-obfuscated copy next to it.**

**4.1 NppExec script**

**Open Plugins › NppExec › Execute… (F6) and paste:**

**text**

**// save current buffer**

**NPP\_SAVE**

**// build paths**

**set local $file = "$(FULL\_CURRENT\_PATH)"**

**set local $clean = "$(CURRENT\_DIRECTORY)\clean\_$(FILE\_NAME)"**

**// run the deobfuscator**

**cmd /c obfuscator-io-deobfuscator "$file" -o "$clean"**

**// open result in a new tab**

**NPP\_OPEN "$clean"**

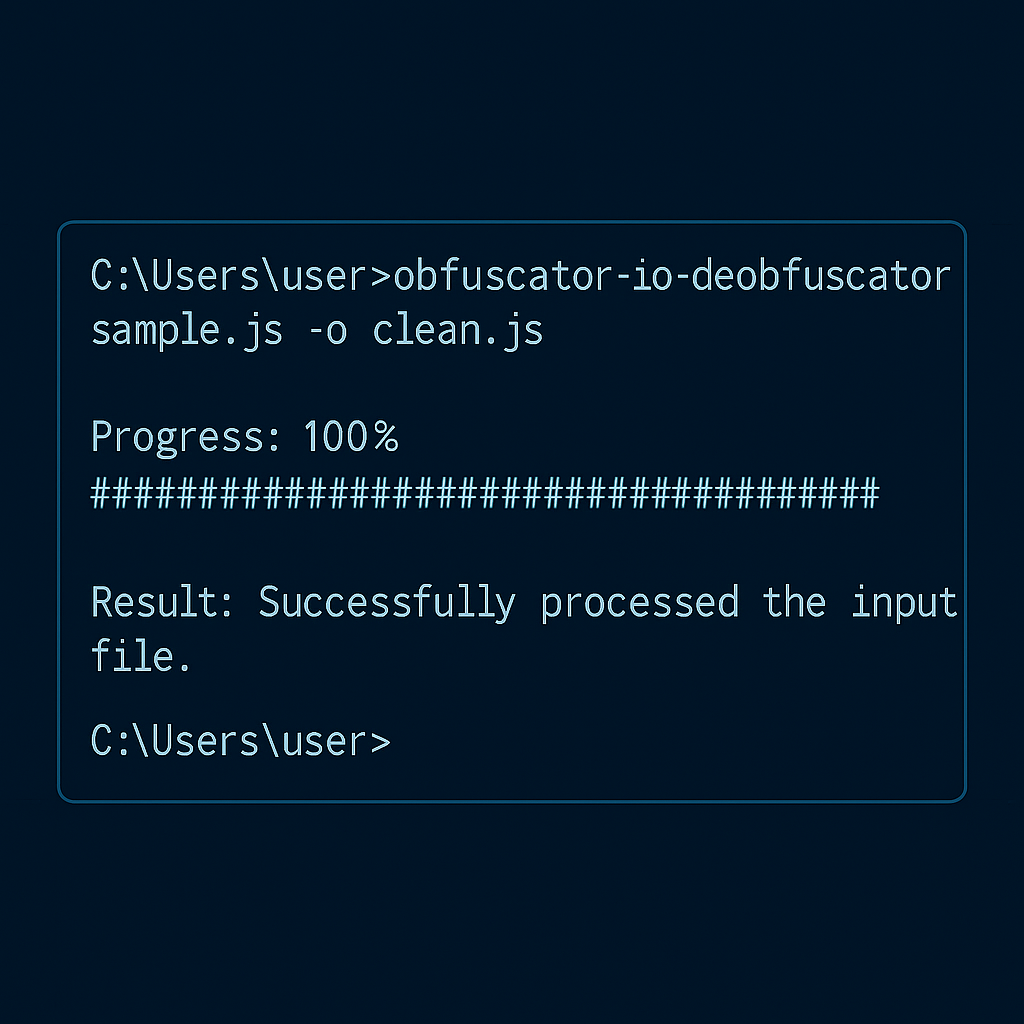
**Save the script as Deobfuscate\_JS.**

**4.2 Add a menu entry & hotkey**

**Plugins › NppExec › Advanced Options… → *Associated script*: Deobfuscate\_JS → Add/Modify → tick *Place to Macros submenu* → OK, then restart Notepad++.  
Assign Ctrl + Shift + D via Settings › Shortcut Mapper › Plugin commands.7**

**4.3 Run the PoC**

1. **Paste any Obfuscator.io sample into a new file and save as sample.js.**
2. **Hit Ctrl + Shift + D.**
3. **The NppExec console shows the CLI progress; a second tab clean\_sample.js appears with readable output.**

****

**Figure 3 – Running the Obfuscator.io CLI deobfuscator directly from NppExec inside Notepad++**

**This round-trip costs under one second for typical payloads, and the original tab remains untouched—ideal for diff-driven malware triage.**

**5 PoC #4 – One-click “beautify + diff” workflow**

**Combine all three plugins:**

**text**

**NPP\_SAVE**

**// step 1 – de-obfuscate**

**set local $in = "$(FULL\_CURRENT\_PATH)"**

**set local $out = "$(CURRENT\_DIRECTORY)\clean\_$(FILE\_NAME)"**

**cmd /c obfuscator-io-deobfuscator "$in" -o "$out"**

**// step 2 – open result**

**NPP\_OPEN "$out"**

**// step 3 – beautify result for diff**

**NPP\_MENUCOMMAND Plugins|JSTool|JSFormat**

**// optional: launch ComparePlus if installed**

**NPP\_MENUCOMMAND Plugins|ComparePlus|Compare**

**Bind this script to *Alt + D* and you have a single shortcut that:**

* **reverses the Obfuscator.io layers,**
* **beautifies the de-obfuscated code, then**
* **shows a side-by-side diff against the original packer output.**

**6 Testing Scenario**

| **Step** | **Action** | **Expected outcome** |
| --- | --- | --- |
| **1** | **Open packed.xml containing one-line XML** | **File loads unformatted** |
| **2** | **Press Ctrl + Alt + Shift + B** | **XML Tools indents tags correctly** |
| **3** | **Open payload.js (Obfuscator.io)** | **Tiny, unreadable source** |
| **4** | **Hit Ctrl + Shift + D** | **New clean\_payload.js tab appears, readable** |
| **5** | **Select Plugins › JS Tool › JS Format** | **Extra spacing & line breaks normalised** |
| **6** | **Run combined macro *Alt + D*** | **diff view opens in Compare Plus (optional)** |

**Total lab time: ≈10 minutes on a fresh VM.**

**7 Security & Ethical Notes**

* **Run tests inside a snapshot-capable VM; scripts may still contain live malware logic.**
* **XML Tools accesses Microsoft’s MSXML libraries; sanitise untrusted XML to avoid external-entity expansion.**
* **Obfuscator-io-deobfuscator executes no payload code—de-obfuscation is performed via AST parsing, but always audit output before running.**
* **Keep the *npm* binary up-to-date to patch any supply-chain vulnerabilities.**

**8 Conclusion**

**By chaining three lightweight extensions, Notepad++ transforms from a mere text editor into a micro-forensics cockpit capable of:**

* **instant XML validation and pretty printing for configuration triage,**
* **single-key JavaScript beautification for static review, and**
* **seamless reversal of Obfuscator.io’s layered packing right inside the editor workflow.**

**Because NppExec can orchestrate any CLI tool, you can extend this playbook to YARA scanning, custom AST analysers, or even invoke *node*-based linters on save—all without leaving Notepad++. The resulting toolkit is transparent, portable, and perfect for incident-response jump boxes where full IDEs are overkill yet quick code insight is critical.**

**With these proofs-of-concept in place, you now have a repeatable recipe for turning raw, obfuscated text artefacts into structured, searchable intelligence in seconds—directly where you edit.**

**POC by Chaitanya P Mayekar ,**

**Intern ID : 253**

**Digi Suraksha Pahari foundation.**