const rawText = $input.first().json.output || '';

// Try to find the SIEM Security Use Cases section (optional fallback)

const startIndex = rawText.indexOf('\*\*SIEM Security Use Cases\*\*');

const siemText = startIndex >= 0 ? rawText.slice(startIndex) : rawText;

// Split by categories — look for lines starting with "\*\*N. Title\*\*"

const categorySplitRegex = /^\\*\\*(\d+)\.\s\*(.+?)\\*\\*/gm;

let categories = [];

let lastIndex = 0;

let match;

// Gather categories with their text blocks

while ((match = categorySplitRegex.exec(siemText)) !== null) {

const index = match.index;

if (categories.length > 0) {

categories[categories.length - 1].content = siemText.slice(lastIndex, index).trim();

}

categories.push({

number: match[1],

name: match[2],

content: ''

});

lastIndex = categorySplitRegex.lastIndex;

}

if (categories.length > 0) {

categories[categories.length - 1].content = siemText.slice(lastIndex).trim();

}

// Original extraction function (ID or ID + Desc)

function extractMitreTechniquesOriginal(text) {

const mitreRegex = /\b(T\d{4}(?:\.\d+)?|PRIV\.\d{3,}|ACTIVEDIRECTORY\.\d{3,})(?:\s\*\(([^)]+)\))?/g;

const results = [];

let m;

while ((m = mitreRegex.exec(text)) !== null) {

const id = m[1];

const desc = m[2];

results.push(desc ? `${id} (${desc.trim()})` : id);

}

return results;

}

// Reverse extraction function (Desc + ID)

function extractMitreTechniquesReverse(text) {

const reverseRegex = /([A-Za-z\s\-]+)\s\*\((T\d{4}(?:\.\d+)?|PRIV\.\d{3,}|ACTIVEDIRECTORY\.\d{3,})\)/g;

const results = [];

let m;

while ((m = reverseRegex.exec(text)) !== null) {

const desc = m[1].trim();

const id = m[2];

results.push(`${id} (${desc})`);

}

return results;

}

// Combine and deduplicate ALL found techniques

function extractMitreTechniquesCombined(text) {

const resultsOriginal = extractMitreTechniquesOriginal(text);

const resultsReverse = extractMitreTechniquesReverse(text);

const combined = [...resultsOriginal, ...resultsReverse];

if (combined.length === 0) return null;

// Use a Set to remove duplicates regardless of parentheses

const seen = new Set();

const deduped = [];

for (const item of combined) {

// Normalize for dedupe: lowercase and remove spaces/punctuation differences if needed

const normalized = item.toLowerCase().replace(/\s+/g, '');

if (!seen.has(normalized)) {

seen.add(normalized);

deduped.push(item);

}

}

return deduped;

}

// Build results

const mitreByCategory = categories.map(cat => {

let mitreTechs = null;

const mitreLineMatch = cat.content.match(/MITRE:\s\*([^\n]+)/i);

if (mitreLineMatch) {

mitreTechs = extractMitreTechniquesCombined(mitreLineMatch[1]);

}

if (!mitreTechs) {

mitreTechs = extractMitreTechniquesCombined(cat.content);

}

return {

category: `${cat.number}. ${cat.name}`,

mitre\_techniques: mitreTechs || []

};

});

// Fallback for no categories found

if (mitreByCategory.length === 0) {

const allMitre = extractMitreTechniquesCombined(rawText);

return [{ json: { mitre\_by\_category: [{ category: "All", mitre\_techniques: allMitre || [] }] } }];

}

return [{ json: { mitre\_by\_category: mitreByCategory } }];