Overview

This n8n workflow automatically turns new cybersecurity news articles into short videos and posts them to social media. It starts by reading articles (from an RSS feed) and filtering for relevant cybersecurity content (using keyword checks like “attackers”, “data breach”, “ransomware”, “VPN”, etc.). Matching articles are extracted from their HTML pages, cleaned, and saved to Airtable (a cloud “spreadsheet-database”). An AI (OpenAI’s GPT-4) writes a video script from each article, and that script is fed into the Pictory video API to generate a video storyboard and produce a video. Finally, the video is posted to Instagram and Facebook (via the Facebook Graph API) and to YouTube. Along the way, the workflow adds affiliate product links to the content and uses AI to generate platform-specific titles and captions.

Key tools and integrations include:
- **Airtable:** For storing and checking articles (with a Personal Access Token as credentials[1]). The workflow uses “base” and “table” IDs to locate the right tables in your Airtable base.
- **Pictory API:** An AI video generation service. The workflow obtains a Pictory access token using client credentials and then calls Pictory’s storyboard/video endpoints[2].
- **OpenAI (LangChain node):** To generate the video script from the article and to create social-media captions. GPT-4 is used in “chat” mode to craft engaging narratives and post text.
- **Facebook Graph API / Instagram API:** The workflow posts videos to Instagram (business account) and Facebook. This requires a Facebook Graph API OAuth credential (with permissions for Instagram content publishing).
- **YouTube API:** To upload the video to a YouTube channel. The YouTube node in n8n uses Google OAuth credentials.

Throughout the workflow, n8n nodes pass data from step to step. “Set” (Edit Fields) nodes store or rename fields, “If” nodes check conditions (using regex on article text), and “Merge” nodes combine or filter item streams. All steps run automatically once configured, so new articles are processed end-to-end without manual intervention.

Workflow Structure

The workflow is organized into several phases. Each node is labeled by name and type, with arrows showing data flow.

- **Triggers:**
- **RSS Feed Trigger (n8n RSS Trigger):** Starts the flow when new items appear in the Hacker News RSS feed (actually *TheHackerNews.com* feed).

- **Schedule Trigger (n8n Schedule Trigger):** Runs periodically (e.g. once a day) to handle video-posting for articles already processed.

- **Fetching and Filtering Articles:**

- **Hacker News RSS Read (RSS Feed Read node):** Reads the RSS XML from *TheHackerNews.com*. Each item has a title and link.
- **Remove Duplicates (Remove Duplicates node):** Drops any items whose `link` field is identical to a previous one (avoids re-processing the same article).
- **HTTP Request (HTTP Request node):** Downloads the full HTML page of the article using the `link` URL from the RSS item.
- **Extract the Article from the HTML (HTML node):** Uses a CSS selector (`div#articlebody.articlebody.clear.cf p`) to extract all paragraph (`<p>`) text from the article's HTML. It outputs an array of paragraphs under `item.json.articlebody`.
- **Clean and Join the Article (Code node):** Runs custom JavaScript to join the paragraph array into one string (with double newlines between paragraphs). This ensures `item.json.articlebody` is a single clean text field.

- **Categorizing Content:** The workflow applies several **If** and **Set** nodes in sequence to tag each article. At each step, it checks the article text for certain keywords and sets a "Category" field accordingly. If no keyword matches, the article is marked as general or discarded. For example:

- **If(Threats) (If node):** Checks if `articlebody` contains words like "attackers", "threat actor", "malicious", etc. (regex, case-insensitive). If **true**, flow goes to:
    - **Edit Fields (Threats) (Set node):** Sets `Category = "Threats"` and carries forward the article text.
      If **false**, flow goes to **Edit Fields (General)** which sets `Category = "General"`.
- **IF (Data Breach Keywords):** If the article text matches data-breach terms ("data breach", "leak", "database breach", etc.), a Set node tags `Category = "Data Breach"`.
- **IF (VPN Keywords):** Similarly, looks for VPN-related terms ("online privacy", "internet freedom", etc.) and sets `Category = "VPN"`.
- **Edit Fields (Ransomware Category):** (This node is oddly named "Ransomware Category" but actually sets `Category = "Hacked"`, presumably for ransomware/hacking news.)
- **IF (Main Filter - Cybersecurity Keywords) (If node):** A final catch-all filter with general cybersecurity terms (e.g. "cyber", "data", "attack", "phishing", etc.). If **false** here, the article is considered non-cyber and is discarded (via an "Edit Fields (Discard/Log Non-Cyber)" step). If **true**, the article passes to the next phase.

The connections (branching) ensure each article goes through these checks in order. The use of multiple **Set** (Edit Fields) nodes allows tagging the article's `Category` based on which keyword group matched.

- **Storing Articles & Avoiding Duplicates:**
- **Search Existing Article by Link (Airtable Search node):** Before saving a new article, this node queries the Airtable "Articles" table (Base ID `appRrwVQOQiEYIKQc`, Table `tblv6tIfwc2lJE5or`) with a formula like `FIND("{{ $json.link }}", {link})`. It checks if the exact link already exists in the table.
- **Filter the Article Which Does Not Exist (Merge node):** This "Merge" step takes the output of the duplicate check and filters out articles that were found. It's set to "Keep Non-Matches", so only new articles (not already in Airtable) continue.
- **Remove Duplicates (Merge)**: After this check, a final "Merge all articles" node combines all branches of new, filtered articles into one stream.

- **Affiliate Links (Optional Product Promotion):**
  Several **Set** nodes add affiliate links for products (like 1Password, NordVPN, etc.). These nodes copy product info (from fields or static values) into the item, and a **Switch** node picks one. For example:

- **1Password Affiliate (Set node):** Sets fields like `Affiliate Product = "1Password"` and its affiliate URL.
- **NordVPN Affiliate (Set node)** and others do the same for their products.
- **Switch to add affiliate links (Switch node):** Routes the item down one branch depending on some condition (like category or availability of affiliate data) so that only one product is chosen.
  After this, there's **Create Record in Airtable (Airtable node)** with an "upsert" operation: it writes the article data (title, link, category, product links, etc.) into Airtable. Fields in the node are mapped to table columns (e.g. Title → `fields.Title`, Category → `fields.Category`, etc.). *Tip:* Ensure the Airtable node's base and table IDs are correct and that field names match those in your base.

- **Script Generation (OpenAI):**

- **Creating script (LangChain OpenAI node):** This node calls GPT-4 (chat completion) to write a video script from the article. It uses a system prompt like "Act as a cinematic cybersecurity storyteller…" and a user message that includes the article's title, text, and link (using expressions like `{{ $json.fields.link }}`). The output (in `choices[0].message.content`) becomes `item.json.Video Script`. If the script is empty, the workflow will skip video creation later.

- **Pictory Storyboard Creation:**

- **Get Pictory Access Token (HTTP Request node):** Posts to `https://api.pictory.ai/pictoryapis/v1/oauth2/token` with JSON

`{"client_id":"...","client_secret":"..."}` (using n8n generic credentials). Pictory responds with an `access_token` (expires in 1hr)[2].

- **Prepare Pictory Storyboard (Code node):** For each item (article), this custom JavaScript builds the Pictory "storyboard" JSON. It reads fields like `title`, `Video Script`, `Affiliate Product`, etc., from the item. It splits the video script into sentences and makes a "scene" for each sentence. Each scene is an object with properties for Pictory (type: `'blockquote'`, content text, font settings, background settings, etc.). If there's an affiliate product, it inserts that as a final sentence/pitch. The code also handles images: if an `item.json.images` array exists, it uses the first image's URL; otherwise it might insert a placeholder. The result is an object with `title`, `access_token`, `pictoryUserId`, and `storyboard.scenes` ready for Pictory's API. (This code returns an array of items with this JSON.)
- **Create Pictory Storyboard (HTTP Request node):** Sends the JSON from the code node to Pictory's `/v2/video/storyboard/render` endpoint. The request includes `Authorization: Bearer {{ $json.access_token }}` in the headers. Pictory responds with a `job_id`.
- **GET Pictory Video STATUS (HTTP Request node):** Polls Pictory's `/v2/video/{job_id}` endpoint to check if the video is done.
- **Check if Video is In-Progress (If node):** Looks at the status from Pictory. If still "processing", the workflow goes to a **Wait** node (pause, then loop back to GET status). If **done**, it proceeds.
- **Get JobId (Set node):** Extracts `job_id` to a separate field if needed.
- **Updating video link (Airtable node):** Once the Pictory video is complete, this node updates the Airtable record (found via `fields.ID` or link) with the final video URL or ID.

- **Social Media Posting:**

- **Message a model (LangChain OpenAI node):** Uses GPT-4 to generate social media text. It is given the article title, text, and the video script, and a system prompt "You are a social media expert…" instructing it to output a JSON-like structure with captions for TikTok, Instagram, etc. For example, it outputs fields like `core_message`, and under `platform_captions` has `TikTok.title`, `TikTok.caption`, `Instagram.caption`, etc. These captions include hooks, hashtags, emojis, and calls to action as per the system prompt.
- **Search records1 (Airtable Search node):** Triggered by schedule, this finds articles in Airtable that have `videoGenerated != ""` but `Status != 'Post'`. (It limits to 2 records at a time.) These are videos ready to post.
- **Adds link (Set node):** Prepares data for downloading or posting. It may set fields like `upload_url` or other tokens.
- **Download video (HTTP Request node):** Given the Pictory video link or ID, this node downloads the video file. The output is binary video data. Make sure the node is set to return binary (e.g. response as file).

- **Instagram post ready (HTTP Request node):** Uses Facebook Graph API to create an Instagram media container. It POSTs to `https://graph.facebook.com/v{version}/{IG_account_id}/media` with `video_url={{$json.binary.video.data.url}}` or the binary, and message text from the OpenAI output. This step returns an `id` for the IG container.
- **Instagram video posted (Facebook Graph API node):** Calls the Facebook node with **Operation** "Publish Media" (`edge: media_publish`) on the Instagram business account using the container ID. This publishes the video to Instagram.
- **Wait1 (Wait node):** Pauses (e.g. 30 minutes) to let Instagram/Facebook process.
- **YouTube video upload (YouTube node):** Uses the YouTube integration node (requires Google OAuth) to create a video resource with the title and description from the AI captions. It returns an `upload_url` and `videoId`.
- **upload video (HTTP Request node):** Posts the binary video data to the `upload_url` from YouTube. This actually uploads the file.
- **Start the upload process (Facebook Graph API node):** Begins an upload session to a Facebook Page (ID `682680894939182` in this example). It POSTs to `/{page_id}/videos` with `upload_phase=start`. The response has `upload_session_id` and an endpoint to transfer.
- **Wait2 (Wait node):** Pauses again (e.g. a minute).
- **Upload final to facebook (HTTP Request node):** Completes the Facebook video upload by posting chunks. In this example it sends the entire video (using `upload_phase=transfer`) to the session. Once done, it will publish the video as a Facebook Reel (`/video_reels`).

- **Final Update:**

- **Merge1 (Merge node):** Combines the results from Instagram, YouTube, and Facebook branches.
- **Update record (Airtable node):** Updates the Airtable record with the final posted URLs/IDs and sets `Status = "Post"` (or similar).
- **Edit Fields1 (Set node):** May adjust any remaining fields (e.g. ensure `link` is correct).
- **Sticky Note (Note node):** A note left by the workflow author (not used for logic).

Each of these nodes is linked in the n8n editor by wires. Data flows through them as described, with expressions (like `={{ $json.link }}`) mapping outputs to inputs. The structure is meant to be left-to-right in the n8n canvas, but is described here sequentially. By following this flow, a cybersecurity article goes from RSS to a published video post with minimal human intervention.

Setup Instructions

Follow these steps to configure the workflow before running it.

1. Airtable Credentials and Nodes

- **Create an Airtable Personal Access Token (PAT):** In Airtable's "Developer Hub" (see Airtable Support), create a new token. Give it a name (e.g. "n8n Workflow Token") and grant scopes `data.records:read`, `data.records:write`, and `schema.bases:read`. Choose the base(s) you will use. Copy the generated token value[1].
- **Add Airtable Credential in n8n:** In n8n's Credential manager, create new "Airtable API" credentials. Paste the PAT into the **Access Token** field. (Recent n8n versions may ask for just "Airtable Personal Access Token".) Save this credential (e.g. name it "Airtable account").
- **Configure Airtable Nodes:** In each Airtable node (Search, Create, Update), select your Airtable credential and specify the **Base ID** and **Table Name or ID**. The base ID is the string like `appRrwVQOQiEYIKQc` found in your base's URL. The table ID (like `tblv6tIfwc2lJE5or`) is also shown in the URL when viewing that table. For example, the *Search Existing Article by Link* node uses Base `appRrwVQOQiEYIKQc` and Table `tblv6tIfwc2lJE5or` (Articles). Ensure the field names in the node (e.g. Title, link, Category) exactly match the column names in your Airtable table. Common mistakes: forgetting to set the credential or mis-typing a Base ID will cause "unauthorized" or "table not found" errors.

2. Pictory API Credentials

- **Obtain Pictory API Client ID/Secret:** Log into your Pictory account and find the API credentials (client ID and client secret). Pictory supports client-credentials OAuth[2]. (Contact Pictory support if unclear where to find these; they may be in an "Integrations" or "API" section of the dashboard.)
- **Add HTTP Credentials in n8n:** In n8n's credentials, create a **HTTP Request with Custom Auth** credential (sometimes labeled *Generic Credential*). Paste the `client_id` and `client_secret` as shown. In nodes like *Get Pictory Access Token*, select this credential under **HTTP Custom Auth**. Ensure "Send Headers" is enabled and that the token endpoint is correct (`https://api.pictory.ai/pictoryapis/v1/oauth2/token`). This node will use your ID/secret to fetch a bearer token. *Common issue:* If the token request fails, double-check your client ID/secret and that "Content-Type: application/json" header is sent (the node parameters should already include that).

3. OpenAI (ChatGPT) Credentials

- **OpenAI API Key:** Sign up at openai.com, get an API key for GPT-4 or GPT-4o. In n8n, under Credentials create a new **OpenAI** credential and paste your key.

- **Configure LangChain Nodes:** In the *Creating script* and *Message a model* nodes, select your OpenAI credential. These nodes have preset prompts. You usually do not need to modify them, but if you do, double-check the templating (it uses `={{ }}` to insert JSON fields). Common error: leaving the default system/user messages unchanged, or mis-formatting the = in front of prompts.

4. Facebook Graph API (Instagram) Credentials

- **Meta (Facebook/Instagram) OAuth:** You need a Facebook App and a user/system that has an Instagram Business Account connected to a Facebook Page. Follow Meta's docs to create an app, add Instagram Basic Display or Graph API, and get a Page Access Token with the `instagram_content_publish` permission. In n8n, add a **Facebook Graph API** credential using OAuth2, which will generate a token.
- **Configure Facebook Graph Nodes:** In the *Instagram post ready* and *Instagram video posted* nodes, select your Facebook credential. Ensure the **Node ID** is set to your Instagram business account ID (looks like `1784...`) and the **Edge** to `media` or `media_publish`. The *Start the upload process (facebook)* and *Upload final to facebook* nodes are posting to a Facebook Page ID (`682680894939182` in the workflow); update that to your Page ID. Make sure your credential has permissions to post videos to the page. A common pitfall is using a personal Instagram token instead of a business account.

5. YouTube Credentials

- **Google OAuth:** Create credentials in the Google Cloud Console for YouTube Data API. You need OAuth2 Client ID/Secret and enable the YouTube Data API. In n8n, create a **YouTube OAuth2** credential and go through the Google consent to connect.
- **YouTube Node Settings:** In the *YouTube video upload* node, select the YouTube credential. Map fields: set **Title** to `={{ $json.Choices.content.platform_captions.TikTok.title }}` (or whichever caption you want), and **Description** to the corresponding caption content (e.g. TikTok or Instagram text). Uploading the video happens in two steps in n8n: the first (YouTube node) generates an upload URL, and the next HTTP Request node sends the binary data to that URL. Make sure **Binary Data** is mapped correctly (the output from "Download video" is the binary). A typical error: not checking "Binary Mode" or mismatched `{{$json.upload_url}}`.

6. Node-by-Node Tips

- **RSS Feed Read:** Just put the RSS URL (`https://thehackernews.com/feeds/posts/default`). No auth needed.
- **HTTP Request (Article):** Set URL to `={{ $json.link }}` so it fetches the link from the RSS item. No auth needed.

- **If Nodes:** Double-check the regex list under **Right Value**. These must be a valid JavaScript regex (or string with pipe-separated terms). If none match, the flow goes to the "non-cyber" branch. Make sure **Ignore Case** is checked for case-insensitive matching.
- **Set (Edit Fields) Nodes:** These simply assign constant or copied values. For example, *Edit Fields (General)* has assignments `Category = General`, `articlebody = {{ $json.articlebody }}`, `link = {{ $json.link }}`. That preserves the text and link. In each Set node, ensure the field names match your data structure. A common mistake is to accidentally delete an assignment or leave a field blank.
- **Merge Nodes:** Watch the join modes. For example, "Filter the Article which does not exists" uses **keepNonMatches**, meaning it drops anything found in Airtable. Do not accidentally set it to "merge all" or you'll keep duplicates.
- **Airtable Upsert:** The **Create or update** node maps JSON fields (like `{{ $json.title }}`) to Airtable fields. Ensure you choose "Upsert" (if you want to update existing records by unique key). If you only want to insert, use "Create" and skip updating. The "cachedResultName" in some nodes shows the table name ("CyberScripts"), which helps verify you're writing to the correct table.
- **Code Node (Clean and join):** This node's JavaScript concatenates an array into text. If the CSS selector from HTML node returned multiple paragraphs, this joins them. No config needed unless the HTML structure changes.
- **Code Node (Prepare Storyboard):** This must return valid JSON. It uses `return outputItems;`. If something goes wrong (e.g. referencing a missing field like `videoScript`), the output might be empty or throw an error. Ensure the previous node indeed set `Video Script`. You can click "Execute Node" to test just the code output. The styling (font URLs, sizes) in the code can be tweaked for your preferences.
- **If (Check Video Status):** The condition probably checks if `{{$json.status}}` is not `"done"` and repeats. Make sure it reads the correct field (from Pictory's API response).
- **Wait Nodes:** These use milliseconds. For example, Wait1 might be set to 1800000 for 30 minutes. Adjust if needed. The first Wait is for Instagram (give time for processing), the second for Facebook upload (maybe a few seconds/minute).
- **Facebook HTTP Nodes:** The "Instagram post ready" (HTTP Request) must include `Authorization: OAuth <your_token>`. In this workflow, the HTTP Request is set to use "Send Headers" and the Facebook credential. It posts JSON with `video_url` (link to the video file) and `caption` from AI. If you get OAuth errors, ensure your token has the `instagram_content_publish` scope and the page is linked to an IG business account.
- **Facebook Graph API Node:** For "Instagram video posted", select the same Facebook OAuth credential. Choose **Node** = Instagram account ID and **Edge** = `media_publish`. A mistake here is using the wrong node ID. You can usually pick these from a dropdown if the credential is valid.

- **Upload final to Facebook (HTTP):** This uses `video_reels` edge. Make sure the **Page ID** is correct for your target. The body should include `upload_session_id`, `upload_phase=transfer`, and the binary file chunk in the `file` parameter if needed. In the workflow, it seems to do one chunk transfer of the entire file. If it fails, try toggling "JSON/Multipart" options or check the Graph API format.

Throughout setup, frequently test each portion before moving on. n8n lets you run one node at a time. Use the "Execute Node" feature and inspect the output. For example, after the HTTP Request that fetches the article, open its output to verify the HTML was retrieved. After each **Set** or **If** node, you can hover to see the JSON data passing through and confirm fields are as expected.

Data Handling

This workflow transforms raw article text into a formatted Pictory storyboard and eventual video. Here's how the data is processed at key steps:

- **Article Extraction:** The HTML node grabs all `<p>` elements under the article's main content div. The code node then joins these paragraphs. This means `item.json.articlebody` becomes a single string of text (with line breaks between paragraphs). This is the full article text that later nodes use. If the HTML structure changes (thehackernews redesigns, for example), you'd need to update the CSS selector in the HTML node.

- **Script Creation:** The "Creating script" node sends a prompt like: "Here's a cybersecurity article. Turn this into a compelling video script." It provides the article title, category, and content to GPT-4. The output goes into `item.json.Video Script`. (This is **NOT** automatically used by n8n; it's a JSON field we access later in the Pictory preparation code.) If the AI can't generate a script, the code node will skip that item (avoiding empty videos).

- **Prepare Pictory Storyboard (Code node):** This is the heart of data transformation. In simple terms, it:

- Reads the `Video Script` (a long paragraph). Skips the item if it's empty.
- Sanitizes the article title (`articleTitle`) for use as the video title.
- Truncates the script to 3000 characters (Pictory's limit).
- Splits the script into sentences (`split(/[\.!?\n]\s*/)`), filtering out empty strings.
- Builds an array of **scene objects**, one per sentence. Each scene has:
    - `type: 'blockquote'` (meaning it's text on screen).
    - Styling: white Roboto font, center-left, size 30.
    - Text content (`story`) = the sentence.

    - A *background video* query focused on “cybersecurity, hacking, network”, etc. (Pictory will search stock videos matching these terms).
    - If the sentence contains a product name (in code logic), it highlights keywords (the code has a `keywordColor`).
    - If this is the *first* scene (index 0), the scene title is the article title (big text).
- After mapping sentences, it checks if an **Affiliate Product** was set. If so, it inserts an extra scene at the end saying something like “This video brought to you by [Product]” with the product pitch and URL.
- It constructs `outputItems` with JSON: each item has `{ json: { title, access_token, storyboard: { scenes } } }`. These fields match what Pictory’s API expects: a title, and a `storyboard` JSON with an array of scenes.

In sum, the code automatically turns the AI-generated script into a scene-by-scene plan. Each scene’s `background.query` searches Pictory’s stock library for relevant visuals (cybersecurity-themed). If you look closely at the code, you’ll see the `backgroundBrolls.searchFilter.keywords` array with terms like `"cybersecurity"`, `"hacking"`, etc., to guide the search. You can customize these keywords or styles by editing the code node. Just be careful to maintain valid JSON output (see **Debugging** below).

- **Scene Objects:** Each scene object in the storyboard JSON has this structure (simplified example):

```
{
  "type": "blockquote",
  "story": "First sentence of script.",
  "style": {
    "position": "center-left",
    "fontFamily": "Roboto",
    "fontSize": 30,
    "color": "rgba(255,255,255,1)",
    "keywordColor": "rgba(255,204,0,1)"
  },
  "background": {
    "type": "image",
    "settings": { "loop": true, "mute": true, ... }
  },
  "backgroundBrolls": [
    {
      "type": "video",
      "searchFilter": { "keywords": ["cybersecurity", "hacking", ...] }
    }
  ]
}
```

When the code node returns these, the **Create Pictory Storyboard** HTTP node sends them to Pictory, which then generates a video preview.

- **Video Generation Loop:** After sending the storyboard to Pictory, the workflow periodically checks `GET /v2/video/{job_id}`. This returns a JSON with a `status` field. If `"InProgress"`, it waits (via the **Wait** node) and tries again. Once `"Succeeded"`, the video link is available (`result.signedUrl`) and is stored in Airtable. This polling pattern is common for API jobs.

- **Social Captions:** The **Message a model** node uses a clever prompt so that the AI outputs JSON. For example, it might return:

```
{
  "role": "assistant",
  "content": {
    "core_message": "Attackers are exploiting software
vulnerabilities...",
    "platform_captions": {
      "TikTok": {
        "title": "Hacker Alert!",
        "caption": "Cyberattack rising! 🚨💻 Learn how to protect
yourself... #cybersecurity"
      },
      "Instagram": { ... },
      "Twitter": { ... }
    }
  }
}
```

  The node has **JSON Output** enabled, so n8n parses this text as JSON. You can then use expressions like `{{$json.choices[0].message.content.platform_captions.TikTok.caption}}` in later nodes to get the TikTok caption text. (In the YouTube node, the **Title** is set to the TikTok title, for example.) This allows one AI step to craft messages tailored to each platform.

- **Publishing:** After the video is ready, the workflow downloads the video binary and uses Facebook/Instagram API calls and the YouTube node to post it. The video file is passed as binary data through n8n by setting the “binary data” option in nodes and referencing it (e.g. `{{$binary.data}}`).

Customization Tips

You can adapt or extend this workflow in several ways:

- **Keywords and Filters:** To change which articles are caught, edit the **If** nodes’ keyword lists. For example, to add a new category (like “Malware”), add a new

If+Set sequence with your chosen regex (e.g. `/malware|virus/i`). Or expand the existing regex strings in the nodes. Use | to separate alternatives. Always test your regex with some sample text to avoid syntax errors.

- **Input Sources:** Instead of TheHackerNews RSS, you could use a different RSS feed or even an RSS Feed **Trigger** node. Simply replace the URL in the **RSS Feed Read** node, or use a different trigger node. For example, use "RSS Feed Read Trigger" if you want the flow to start on each new item automatically (the provided JSON actually has an RSS Feed *Trigger* node at the end which can be enabled).

- **New Platforms:** To add TikTok, for example, you could insert steps after "Download video" to POST the video to TikTok's API (if a suitable API or n8n node exists). The AI prompts can already generate a TikTok caption (see **platform_captions.TikTok**). You would need your TikTok OAuth credentials and use HTTP Request to the TikTok API. The structure shown for Instagram/Facebook can serve as a template.

- **Affiliate Links:** The workflow has 1Password, NordVPN, Malwarebytes, SurfShark affiliates as examples. To add another product, duplicate one of those **Set** nodes, update it with your product name and affiliate URL, and add it to the **Switch** node logic. Then expand the GPT-4 prompts if needed to include that product's context.

- **AI Prompts:** The prompts in the LangChain nodes are hard-coded but modifiable. You could refine the tone (e.g. ask for humor or more detail). If you add new output fields to the JSON format in **Message a model**, ensure you update how those fields are consumed (e.g. if you add a "Twitter" caption, use `{{$json.platform_captions.Twitter.caption}}`).

- **Error Handling:** You might want to branch flows on failures. n8n lets you add an **Error Trigger** or catch errors in workflows. For example, if Pictory fails to make a video, you could send a notification (Email or Slack) in a separate branch. To do this, add an "Error Trigger" node and connect it.

- **Auto-Upload Scheduling:** Instead of a fixed wait, you could use a Schedule Trigger for posting (e.g. "Run at 9am daily"). The provided "Schedule Trigger" does part of this, pulling ready videos. You could also incorporate calendar checks or posting quotas.

- **Modularity:** For a cleaner workflow, break parts into Sub-Workflows or use **Execute Workflow** nodes. For example, put all "Social Posting" nodes in one sub-workflow that you can reuse for different triggers.

- **Logging:** Enable the **Keep output on error** in settings to help debug by storing data from failed executions. Or insert "Function" nodes with `console.log()` in code for deeper investigation (you can view these logs in n8n's Execution List).

Error Handling and Debugging

A few common issues and tips:

- **Invalid JSON in Code Node:** If the "Prepare Pictory Storyboard" code outputs bad JSON, n8n will show an error like *"The 'JSON Output' does not contain a valid JSON object."* This usually means a typo in the JavaScript or a missing field. To debug, copy your code into a JavaScript JSON validator or add `console.log` statements in the code node (viewable in n8n's execution logs) to inspect intermediate values. Make sure you always `return` an array of items (even if empty). See n8n's docs on JSON output errors[3].

- **Expression Errors:** If an expression can't find data (e.g. `$json.Choices` or `$json.fields.link`), n8n may say "Cannot get value for expression" or "Referenced node is unexecuted." This means the previous node didn't run or didn't produce the field. For example, if **Creating script** did not run (maybe it was skipped due to a condition), but the code node tries to use `$json['Video Script']`, you'll see errors. To fix, test up to that point, and verify that each "parent" node was executed (check the execution list). Also ensure the node order is correct (an expression can only use data from nodes earlier in the flow). The docs suggest checking if the referenced node has executed[4].

- **Authorization Errors:** A "401 Unauthorized" usually means a bad credential. For Pictory, tokens expire every hour[2], so ensure you get a fresh token before each request. The workflow calls `Get Pictory Access Token` each time it needs to talk to Pictory. For Airtable, a **403 Forbidden** often means the PAT lacks the right scope or the base/table ID is wrong. Double-check your token's scopes and that it has access to the base [1]. For Facebook/Instagram, check that your OAuth token has the `instagram_content_publish` and `pages_read_engagement` scopes and that you're using the correct IG Business Account ID. For YouTube, ensure the Google credential has YouTube Data API enabled.

- **Mismatched Data Items:** Some nodes (like **Merge**) combine multiple inputs. If you get a message about "item index mismatch" or similar, it may mean the arrays have different lengths. For example, the *Merge all articles* node expects 5 inputs (keywords branches), so if one branch had no items, ensure the merge is configured to handle missing inputs (`mode: 'append'` or `keepNonMatches`). Also, if a code node expects `item.json.images` but none was provided, it might break. In the Pictory code, the author used a conditional: `let imageUrl = item.json.images ? item.json.images[0].url : placeholderImage;` so it falls back if no images exist. Always use such fallbacks if data might be absent.

- **Debugging Tips:**

- Use **"Split In Batches"** nodes to run a few items at a time and inspect outputs.

- Insert **Debug Note** or **Function** nodes to log data. For example: `console.log(JSON.stringify(item.json, null, 2));` in a code node will print to the workflow execution log.
- Check each node's output panel: n8n displays all fields (JSON and binary). This is invaluable for seeing if the right values (e.g. `$json.Video Script`) are set.

Best Practices

- **Scalability:** For high volume, consider splitting the workflow. For example, use one workflow for "Fetch and store articles" and another for "Generate videos and post." You can chain them with Webhook or sub-workflow calls. This way you can run them on separate schedules or even on different n8n instances. Use **SplitInBatches** if needed to process many items without timing out.
- **Error Tolerance:** Use Try/Catch (Execute Workflow or IF Error) patterns. For instance, surround external API calls (Facebook, Pictory) with branches that catch failures. This prevents the whole workflow from stopping if one article fails.
- **Credential Security:** Store credentials only in the n8n credential store, not in workflow parameters. Use least-privilege scopes (e.g. only give Airtable token access to the needed bases[1]). Rotate tokens regularly. Do **not** hardcode secrets in expression strings.
- **Modularity:** Group related tasks. For example, put the Pictory steps in a Sub-Workflow that accepts the script and returns a video link. Then the main workflow can call it via **Execute Workflow**. This makes testing and reuse easier.
- **Monitoring:** Use n8n's built-in logging and consider integrating a notification on failures (e.g. Slack or email).
- **Documentation:** Keep your field mappings and expressions documented. The workflow itself is complex, so comments (Sticky Notes in n8n) are helpful for future maintainers. For example, note why certain regex patterns were chosen.
- **Version Control:** Export and store your workflow JSON in version control (Git). This JSON (the one provided) is the source file. When you update the workflow in n8n, export it again to keep track of changes.

By carefully setting up each node with the right credentials and mappings, and by following these tips, you can have this n8n workflow automatically fetch cybersecurity news, turn it into engaging videos via AI, and publish across platforms—all with minimal manual effort.

Sources: Official documentation for Airtable PAT setup[1], Pictory API authentication[2], and n8n expression debugging[3][4]. These provide guidance on token setup and common error messages encountered in this flow.

---

[1] Airtable credentials | n8n Docs

https://docs.n8n.io/integrations/builtin/credentials/airtable/

[2] Get Authentication Token

https://docs.pictory.ai/reference/authentication

[3] [4] Expressions common issues | n8n Docs

https://docs.n8n.io/code/cookbook/expressions/common-issues/