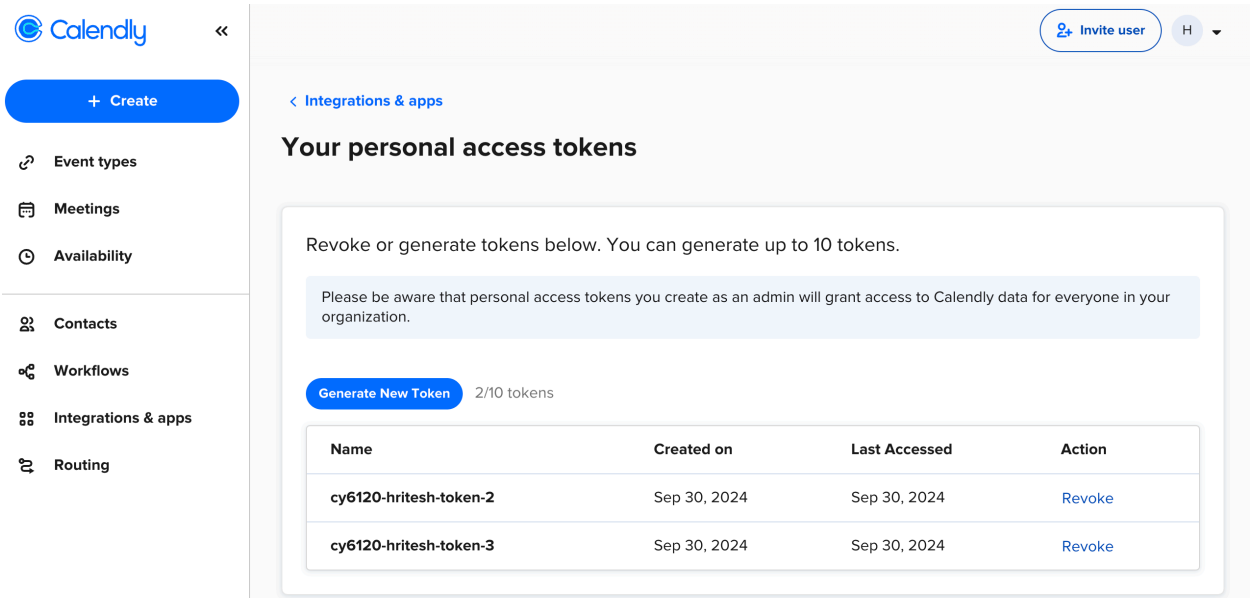
Assignment 1 – Detecting Hardcoded Secrets

Hritesh Sonawane

--

Started with creating a burner account on Calendly and created a couple of API tokens.



--

Version 1: Basic Detection of Hardcoded API Tokens

Objective:
The first version of the tool aims to detect hardcoded Calendly API tokens in files within a given directory using regular expressions. This version identifies tokens based on a simple pattern without any filtering or validation.

Approach:

1. Analyze Token Structure:
    a. I started by analyzing the format of a Calendly API token. It consists of an alphanumeric string and is a Base64-encoded format.
    b. For example, a token may resemble:

```
eyJraWQiOiIxY2UxZTEzNjE3ZGNmNzY2YjNjZWJjY2Y4ZGM1YmFmYThhNjVlNjg0MDIzZjdjMzJiZTgzNDliMjM4MDEzNWI0IiwidHlwIjoiUEFUIiwiYWxnIjoiRVMyNTYifQ.eyJpc3MiOiJodHRwczovL2F1dGguY2FsZW5kbHkuY29tIiwiaWF0IjoxNzI3NzI0MTk5LCJqdGkiOiJiYmMxZTM5MC02NWY4LTRhNTUtODI5MS1mZmM5NDI3MTMwYzgiLCJ1c2VyX3V1aWQiOiJhODk3ZDFjMC00ZjA5LTQ5NTAtOWQyNS1lNGQ1YTBiZjdiODYifQ.s8ormtTmtuPtypOkIK-YDlo-3OcVLEnHepN1lHd7tBdf8F9zoctC_O_1TIcqXFtKg6TlhCUn1h82iHd00Blc0g
```

2. Develop Regex Pattern:
    a. Based on the token format, I created a regex pattern to identify tokens.
    b. This matches JWT-like tokens with 3 parts separated by periods.
3. Scanning Files:
    a. The tool takes a directory as input and recursively scans all files for matching tokens.
    b. It reports the filename, line number, and the matched token.
4. Context Output:
    a. Once a match is found, the tool prints the surrounding lines to provide context for the token usage.

--

Version 2: Improved Filtering of Obvious Invalid Tokens

Objective:
In this version, the tool filters out tokens that clearly cannot be valid. This reduces false positives, like tokens filled with repeating characters or invalid formats.

Approach:

1. Refine Regex for Validity:
    a. In Version 1, the regex was simplistic and could match invalid tokens (e.g., strings filled with 'AAAA').
    b. I improved the regex by adding checks to ensure the middle part of the token doesn't consist entirely of a single repeated character or obvious placeholders.
    c. The regex ensures each part of the token has at least a reasonable length, avoiding false matches.
2. Obvious Invalid Token Filtering:
    a. I also added checks to prevent flagging tokens that are obviously placeholders or invalid, such as:
        i. Repeated patterns like "AAAA..." or "0000..."
        ii. Empty fields where no token is present.
3. Better output:
    a. The context output remains similar, but I added logic to color highlight the detected token more clearly in the output.

--

Version 3: Token Validation

Objective:
The final version adds functionality to validate whether the detected tokens are currently active or expired. This version helps differentiate valid and expired tokens, enhancing security awareness.

Approach:

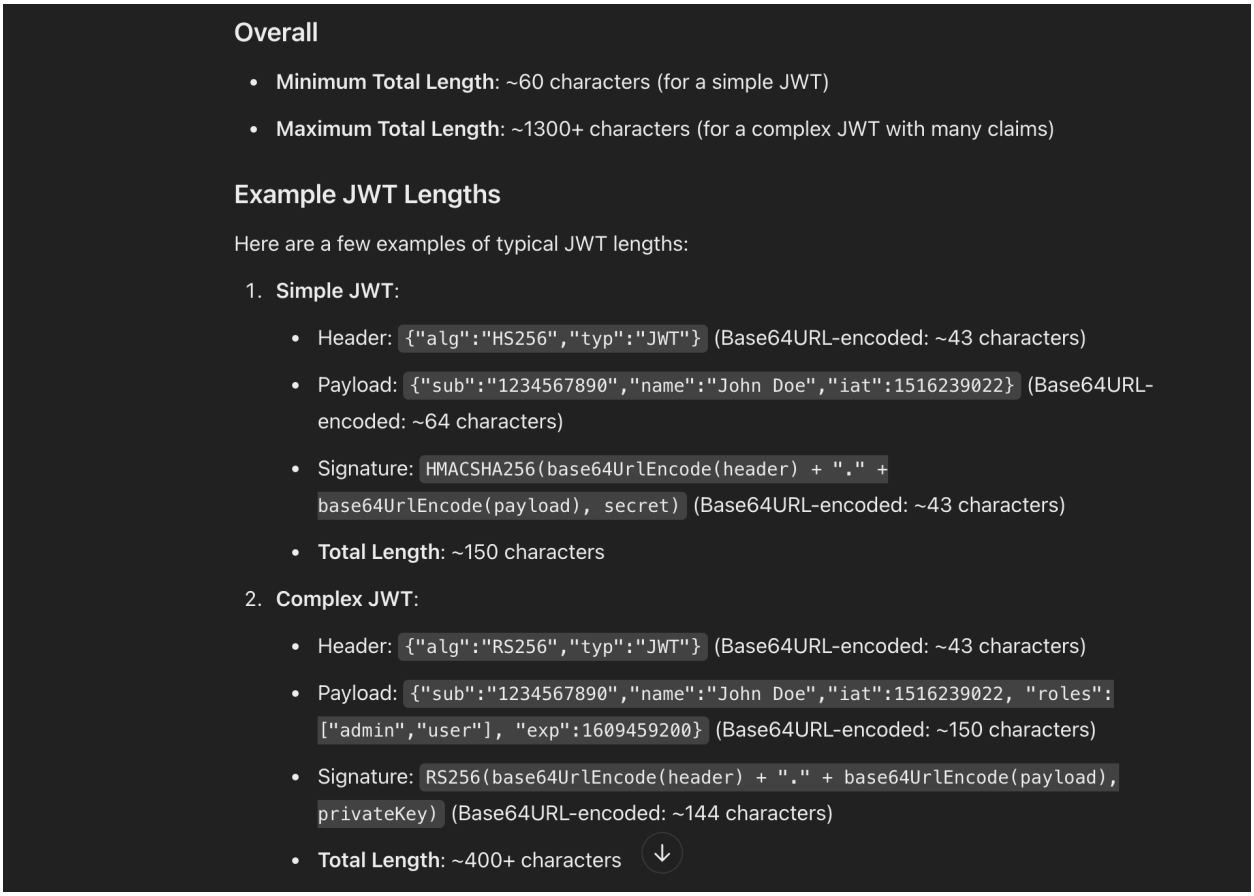
1. Token Validation:
    a. I implemented a validation step to check whether the detected token is still valid or has been revoked.
    b. This involved sending the detected token to Calendly's API endpoint for verification.
2. Validation Workflow:
    a. The detected tokens are sent to an API validation function. If the API response indicates a valid token, the tool prints VALID TOKEN! Otherwise, it flags the token as invalid.
3. Output:
    a. For valid tokens: VALID TOKEN!
    b. For valid but expired tokens: This token is not valid.

--

We have a robust tool that detects hardcoded Calendly API tokens, filters out obviously invalid tokens, and even validates the tokens' status using the Calendly API. Each version of the tool builds upon the previous one, enhancing functionality and accuracy.

Screenshots and reference: (see next page.)

1. Asking ChatGPT for valid JWT lengths for fine-tuning



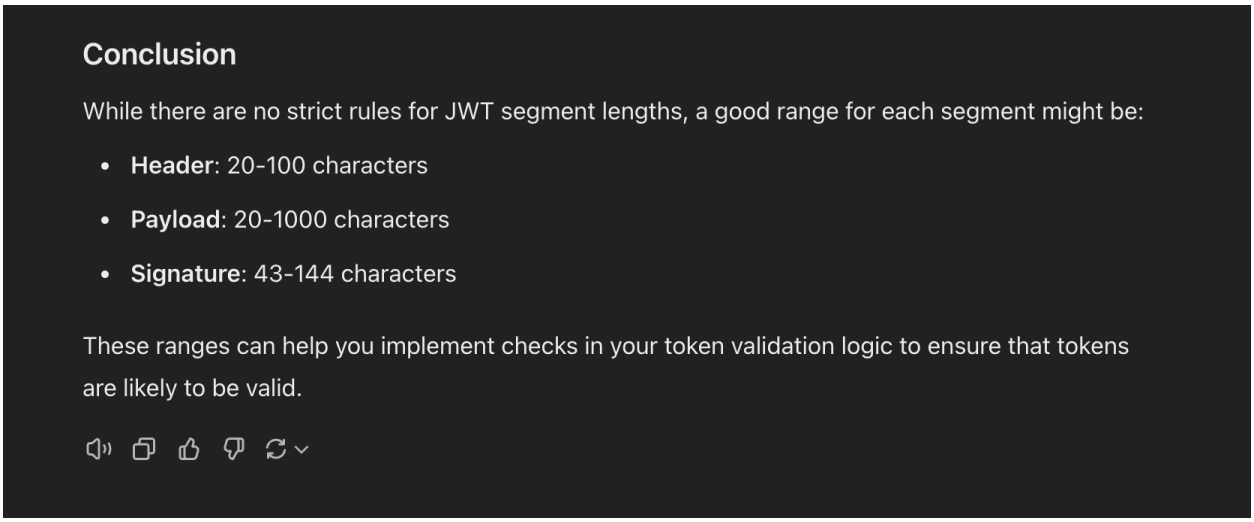
Overall

- **Minimum Total Length**: ~60 characters (for a simple JWT)
- **Maximum Total Length**: ~1300+ characters (for a complex JWT with many claims)

Example JWT Lengths

Here are a few examples of typical JWT lengths:

1. **Simple JWT**:
   - Header: `{"alg":"HS256","typ":"JWT"}` (Base64URL-encoded: ~43 characters)
   - Payload: `{"sub":"1234567890","name":"John Doe","iat":1516239022}` (Base64URL-encoded: ~64 characters)
   - Signature: `HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)` (Base64URL-encoded: ~43 characters)
   - **Total Length**: ~150 characters
2. **Complex JWT**:
   - Header: `{"alg":"RS256","typ":"JWT"}` (Base64URL-encoded: ~43 characters)
   - Payload: `{"sub":"1234567890","name":"John Doe","iat":1516239022, "roles": ["admin","user"], "exp":1609459200}` (Base64URL-encoded: ~150 characters)
   - Signature: `RS256(base64UrlEncode(header) + "." + base64UrlEncode(payload), privateKey)` (Base64URL-encoded: ~144 characters)
   - **Total Length**: ~400+ characters



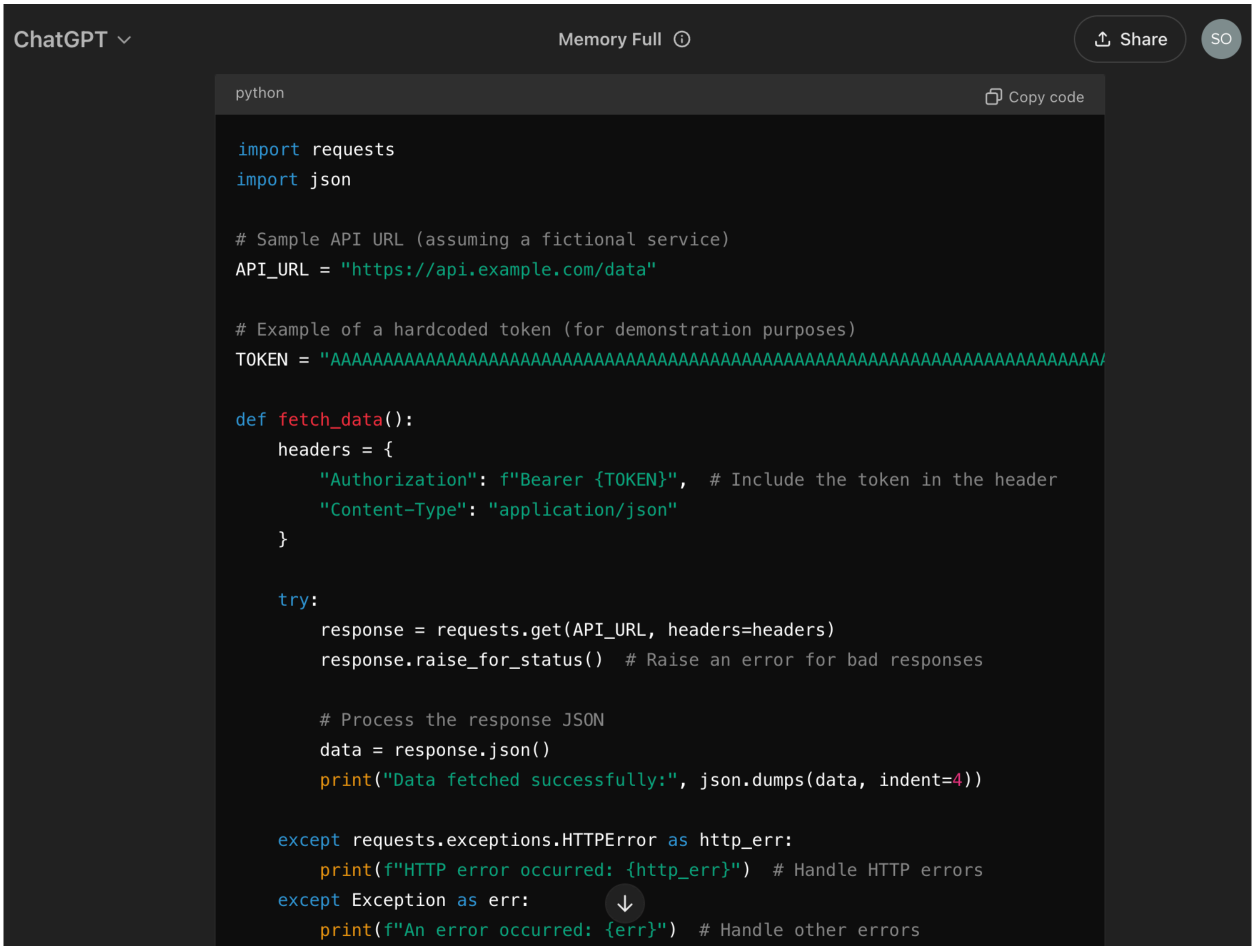
Conclusion

While there are no strict rules for JWT segment lengths, a good range for each segment might be:

- **Header**: 20-100 characters
- **Payload**: 20-1000 characters
- **Signature**: 43-144 characters

These ranges can help you implement checks in your token validation logic to ensure that tokens are likely to be valid.

2. Asking ChatGPT for mock files to embed tokens in




```python
import requests
import json

# Sample API URL (assuming a fictional service)
API_URL = "https://api.example.com/data"

# Example of a hardcoded token (for demonstration purposes)
TOKEN = "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

def fetch_data():
    headers = {
        "Authorization": f"Bearer {TOKEN}",  # Include the token in the header
        "Content-Type": "application/json"
    }

    try:
        response = requests.get(API_URL, headers=headers)
        response.raise_for_status()  # Raise an error for bad responses

        # Process the response JSON
        data = response.json()
        print("Data fetched successfully:", json.dumps(data, indent=4))

    except requests.exceptions.HTTPError as http_err:
        print(f"HTTP error occurred: {http_err}")  # Handle HTTP errors
    except Exception as err:
        print(f"An error occurred: {err}")  # Handle other errors
```

3. Looking around for JWT regex
   https://stackoverflow.com/questions/49517324/why-header-and-payload-in-the-jwt-token-always-starts-with-eyj

4. Calendly API doc
   https://developer.calendly.com/api-docs/005832c83aeae-get-current-user

5. ASCII art
   https://patorjk.com/software/taag/#p=display&f=Graffiti&t=TokenTracker

---
Thank you!