Enriching Google CSE Results: Adding Context to CVEs (5 of 5)



By Mario Rojas, Senior Security Researcher

In previous posts, we explored how to build and customise a Google Custom Search Engine (CSE) to streamline vulnerability research. Now, we'll take things further by **enriching CSE results with additional intelligence sources**.

When tracking CVEs, simply finding a mention in a security blog or forum isn't enough. To **assess risk effectively**, you need more details:

- What's the severity and impact of this vulnerability?
- Is there a known exploit or proof-of-concept (PoC) code?
- Is this CVE actively being used in attacks?
- Which Threat Actors are targeting this CVE?

By **automating enrichment**, we can take search results from a **basic list of links** to **actionable intelligence**, helping security teams prioritise threats efficiently.

Step 1: Extracting CVEs from Google CSE Results

Google CSE is an excellent way to discover CVE mentions across security blogs, research reports, and exploit discussions. But first, we need to extract the CVE identifiers from our search results.

Fetching CVE-Related Results:

python Copy code

```python
import requests
import re

GOOGLE_CSE_URL = 'www.googleapis.com/customsearch'
GOOGLE_API_KEY = 'your_google_api_key'
CSE_ID = 'your_google_cse_id'
QUERY = 'CVE-2024 OR CVE-2025 OR vulnerability OR exploit'

url = f"https://{GOOGLE_CSE_URL}/v1?q={QUERY}&key={GOOGLE_API_KEY}&cx={CSE_ID}"
response = requests.get(url)

print("### Collecting CVEs from Google API ###\n")

if response.status_code == 200:
    results = response.json().get("items", [])

    # Extract CVE IDs from search results
    cve_pattern = re.compile(r'CVE-\d{4}-\d+') # Here we use Regex to extract CVE IDs
    extracted_cve_ids = set()

    for item in results:
        found_cve_ids = cve_pattern.findall(item.get("title") + " " + item.get("snippet"))
        extracted_cve_ids.update(found_cve_ids)

    print(f"Found CVEs: {', '.join(extracted_cve_ids)}")
else:
    print("Error:", response.status_code, response.text)
```

You should get something like this after running your script:

```
/Users/myuser/python

### Collecting CVEs from Google API ###

Found CVEs: CVE-2025-1094, CVE-2024-12356, CVE-2024-1234, CVE-2025-0289
```

What This Does:

- **Queries CSE** for vulnerability-related content.
- **Extracts CVE identifiers (e.g., CVE-2025-1094)** from search results.
- **Returns a list of CVEs** found in blog posts, forums, or advisories.

At this point, we have a **list of CVEs mentioned in various sources**. Next, we'll **enrich** them with **official data from the National Vulnerability Database (NVD)**.

Step 2: Enriching CVEs with NVD Data

Once we have a list of CVEs, the next step is **pulling official details** from **the National Vulnerability Database (NVD)**. This allows us to access:

- **Severity ratings (CVSS scores)**
- **Affected vendors and products**
- **Exploitability and patch availability**

To query NVD, we can use their API which allows structured searches for vulnerability details.

Fetching CVE Details from NVD

python Copy code

```python
... previous code ...

NIST_API_KEY = 'your_nvd_api_key'

print("### Checking NIST NVD ###\n")

def get_cve_info(cve_id):
    headers = {'apiKey': NIST_API_KEY}
    url = f"https://services.nvd.nist.gov/rest/json/cves/2.0?cveId={cve_id}"
    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        data = response.json()
        if data.get("vulnerabilities"):
            cve_info = data.get("vulnerabilities")[0].get("cve")
            description = cve_info.get("descriptions")[0].get("value")
            cisa_kev = bool(cve_info.get("cisaExploitAdd", ""))  # Known Exploited Vulnerability
            return description, cisa_kev
    return "CVE not found"


# Example: Enrich each extracted CVE
for cve in extracted_cve_ids:
    details = get_cve_info(str(cve).upper().strip())
    print(f"CVE: {cve}")
    print(f"Description: {details[0]}")
    print(f"CISA KEV: {details[1]}\n")
```

You should get something like this after running your script:

```
/Users/myuser/python

### Checking NIST NVD ###

CVE: CVE-2024-1234
Description: The Exclusive Addons for Elementor plugin for WordPress is vulnerable to Stored
Cross-Site Scripting via data attribute in all versions up to, and including, 2.6.9 due to
insufficient input sanitization and output escaping. This makes it possible for authenticated
attackers, with contributor access or higher, to inject arbitrary web scripts in pages that will
execute whenever a user accesses an injected page.
CISA KEV: False

CVE: CVE-2024-12356
Description: A critical vulnerability has been discovered in Privileged Remote Access (PRA) and
Remote Support (RS) products which can allow an unauthenticated attacker to inject commands that
are run as a site user.
CISA KEV: True
```

Why This Matters:

- **Provides official descriptions** from MITRE/NIST.
- **Confirms if a CVE is valid** (or still in draft status).
- **Includes additional information** like CISA KEV status.

To get higher request limits and additional API features, you can **request an API key by clicking here**. This ensures **faster responses** and **better access control** for large-scale queries.

Step 3: Checking for Public Exploits with Nuclei Templates

Understanding whether a CVE has a **proof-of-concept (PoC) test** or known exploitability is critical for **risk assessment and prioritisation**. While not every vulnerability has a working exploit, security researchers and red teams frequently develop **scanning templates** to verify whether a system is vulnerable.

Nuclei is an open-source vulnerability scanning tool that maintains **CVE-specific templates**. These templates provide a structured way to **test for known vulnerabilities** without relying on unreliable or rate-limited exploit repositories.

Checking Nuclei for CVE Templates

```python
... previous code ...

NUCLEI_CVES = 'https://github.com/projectdiscovery/nuclei-templates/blob/main/cves.json'

print("### Checking Nuclei ###\n")

def check_nuclei(cve_id):

    response = requests.get(NUCLEI_CVES)

    if response.status_code == 200:
        nuclei_data = response.text
        if cve_id in nuclei_data:
            return True
    return False


# Example: Check if CVE has nuclei template
for cve in extracted_cve_ids:
    nuclei_info = check_nuclei(cve)
    print(f"{cve} has nuclei template: {nuclei_info}")
```

You should get something like this after running your script:

```
/Users/myuser/python

### Checking Nuclei ###

CVE-2025-0289 has nuclei template: False
CVE-2024-12356 has nuclei template: False
CVE-2025-1094 has nuclei template: False
CVE-2024-1234 has nuclei template: False
```

Why This Matters:

- Identifies vulnerabilities with publicly available security tests.
- Allows security teams to verify exploitability in real-world environments.
- Supports automation by integrating with vulnerability scanning workflows.

By checking for Nuclei templates, security teams can **quickly determine whether a CVE has known detection or exploitation methods**, making it easier to prioritise remediation or conduct further testing in a controlled environment.

Step 4: Identifying Threat Actor Associations

Not all CVEs are the same: while some remain theoretical risks, others are actively exploited by threat actors in real-world attacks. Identifying **which vulnerabilities are linked to known adversaries** can help security teams prioritise patching efforts and **assess the urgency of a CVE**.

To determine whether a CVE is part of an **active attack campaign**, we can cross-check it against **threat intelligence feeds** such as **AlienVault OTX**, a community-driven threat intelligence platform that tracks IOCs, vulnerabilities, and adversary behavior. You can request an API key here.

Checking AlienVault OTX for Threat Intel

python Copy code

```python
... previous code ...

OTX_API_KEY = 'your_alienvault_key'

print("\n### Checking AlienVault OTX ###\n")

def check_otx(cve_id):

    headers = {"X-OTX-API-KEY": OTX_API_KEY}
    url = f"https://otx.alienvault.com/api/v1/indicators/cve/{cve_id}"
    response = requests.get(url, headers=headers)

    if response.status_code == 200:
        return response.json()
    return "No threat intelligence found"


# Example: Check if CVE has pulses template
for cve in extracted_cve_ids:
    threat_data = check_otx(cve)
    print(f"{cve}: {threat_data.get('pulse_info')}")
```

You should get something like this after running your script:

```
/Users/myuser/python

### Checking AlienVault OTX ###

CVE-2024-12356: {'count': 1, 'pulses': [{'id': '676d59c3bbdd4eec22810de9', 'name': 'CISA Adds
Acclaim Systems and BeyondTrust Vulnerabilities to Exploited List', 'description': "",
'modified': '2024-12-26T13:27:31.199000', 'created': '2024-12-26T13:27:31.199000', 'tags':
['threattype/vulnerability Exploitation', 'kevc/Acclaim Systems USAHERDS CVE-2021-44207', 'kevc/
BeyondTrust Privileged Remote Access (PRA) and Remote Suppo'], 'references': [], 'public': 1,
'adversary': '', 'targeted_countries': [], 'malware_families': [], 'attack_ids': [{'id': 'T1190',
'name': 'Exploit Public-Facing Application', 'display_name': 'T1190 - Exploit Public-Facing
Application'}], 'industries': [], 'TLP': 'green', 'cloned_from': None, 'export_count': 17,
'upvotes_count': 0, 'downvotes_count': 0, 'votes_count': 0, 'locked': False, 'pulse_source':
'web', 'validator_count': 0, 'comment_count': 0, 'follower_count': 0, 'vote': 0, 'author':
{'username': 'eric.ford', 'id': '42510', 'avatar_url': '/otxapi/users/avatar_image/media/avatars/
user_42510/resized/80/avatar_3b9c358f36.png', 'is_subscribed': False, 'is_following': False},
'indicator_type_counts': {'CVE': 2}, 'indicator_count': 2, 'is_author': False, 'is_subscribing':
None, 'subscriber_count': 110, 'modified_text': '68 days ago ', 'is_modified': False, 'groups':
[], 'in_group': False, 'threat_hunter_scannable': False, 'threat_hunter_has_agents': 1,
'related_indicator_type': 'CVE', 'related_indicator_is_active': 1}], 'references': [], 'related':
{'alienvault': {'adversary': [], 'malware_families': [], 'industries': []}, 'other':
{'adversary': [], 'malware_families': [], 'industries': []}}}
```

Why This Matters:

- **Detects if a CVE is part of an active attack campaign.**
- Identifies threat actors and malware using the CVE.

- Helps security teams prioritise patching efforts.

Once you put everything together you should get a nice formatted summary like the one below.

```
/Users/myuser/python

### Collecting CVEs from Google API ###

Found CVEs: CVE-2024-12356

### Checking NIST NVD ###

CVE: CVE-2024-12356
Description: A critical vulnerability has been discovered in Privileged Remote Access (PRA) and
Remote Support (RS) products which can allow an unauthenticated attacker to inject commands that
are run as a site user.
CISA KEV: True

### Checking Nuclei ###

CVE-2024-12356 has nuclei template: False

### Checking AlienVault OTX ###

CVE-2024-12356: {'count': 1, 'pulses': [{'id': '676d59c3bbdd4eec22810de9', 'name': 'CISA Adds
Acclaim Systems and BeyondTrust Vulnerabilities to Exploited List', 'description': "",
'modified': '2024-12-26T13:27:31.199000', 'created': '2024-12-26T13:27:31.199000', 'tags':
['threattype/vulnerability Exploitation', 'kevc/Acclaim Systems USAHERDS CVE-2021-44207', 'kevc/
BeyondTrust Privileged Remote Access (PRA) and Remote Suppo'], 'references': [], 'public': 1,
'adversary': '', 'targeted_countries': [], 'malware_families': [], 'attack_ids': [{'id': 'T1190',
'name': 'Exploit Public-Facing Application', 'display_name': 'T1190 - Exploit Public-Facing
Application'}], 'industries': [], 'TLP': 'green', 'cloned_from': None, 'export_count': 17,
'upvotes_count': 0, 'downvotes_count': 0, 'votes_count': 0, 'locked': False, 'pulse_source':
'web', 'validator_count': 0, 'comment_count': 0, 'follower_count': 0, 'vote': 0, 'author':
{'username': 'eric.ford', 'id': '42510', 'avatar_url': '/otxapi/users/avatar_image/media/avatars/
user_42510/resized/80/avatar_3b9c358f36.png', 'is_subscribed': False, 'is_following': False},
'indicator_type_counts': {'CVE': 2}, 'indicator_count': 2, 'is_author': False, 'is_subscribing':
None, 'subscriber_count': 110, 'modified_text': '68 days ago ', 'is_modified': False, 'groups':
[], 'in_group': False, 'threat_hunter_scannable': False, 'threat_hunter_has_agents': 1,
'related_indicator_type': 'CVE', 'related_indicator_is_active': 1}], 'references': [], 'related':
{'alienvault': {'adversary': [], 'malware_families': [], 'industries': []}, 'other':
{'adversary': [], 'malware_families': [], 'industries': []}}}
```

Conclusion

This post marks the final entry in our **Google Custom Search Engine (CSE) for OSINT** series, and we hope it has provided you with valuable insights into **building, customising, automating, and enriching your CSE** for more effective intelligence gathering.

By applying CVE enrichment techniques, you can **transform basic search results into actionable intelligence**, helping security teams assess risks faster, track exploitation, and prioritise vulnerabilities based on real-world threats.

Throughout this series, we covered:

1. Understanding the potential of CSE for corportate threat intelligence.
2. Creating a custom search engine tailored for OSINT.

3. Using advanced features like refinements and query enhancements.
4. Accessing search results via API and automating queries.
5. Enriching results with external intelligence sources

The goal of this series was to **help OSINT practitioners, threat intelligence teams, and cybersecurity researchers get more from their searches**. By leveraging **Google CSE as a powerful intelligence-gathering tool**, you can streamline your workflow and stay ahead of emerging threats.

Thank You!

We appreciate everyone who followed this series, shared feedback, and explored these techniques alongside us. OSINT is a constantly evolving field, and the best tools are those we refine and improve over time.

This may be the end of this series, but the **OSINT journey never stops**. Keep exploring, refining, and optimising – there’s always more intelligence to uncover.