THOMAS MORE

SOC-project

Realization Report

Baisangur Dudayev
Bachelor Electronics-ICT - Cloud & Cybersecurity

2024 - 2025

Table of Contents

TABLES AND FIGURES

1. Introduction

Netcure provides security services to its clients through continuous monitoring and alerting. To support this, it required a Security Operations Center (SOC), which plays a key role in collecting, analysing, and responding to security events from various platforms.

However, alerts and notifications were previously spread across different tools and systems, making it difficult to get a clear overview or generate consistent reports. This created a need for a more centralized and automated approach to improve visibility and efficiency.

The goal of the internship was to help solve this by developing a system that brings together alerts and logs from different sources into one environment. This includes integrating tools such as email, APIs, ELK stack, and visual dashboards like Grafana and Kibana. The SOC not only supports client environments but also helps protect the internal infrastructure of Netcure itself.

By improving how alerts are collected, visualized, and reported, the project directly supports Netcure's core business and strengthens its security operations.

The timeline below reflects the total time I personally spent on each major topic during the internship:

- **Basic SIEM configuration 6 WEEKS**
  - Basic configuration. Collecting logs, filtering and aggregating them.
  - Setting up testing environments for my team using WSL to run Ansible commands needed for configuration and deployment tasks
- **Reporting/Visualization 4 WEEKS**
  - Integrating an automatic Guardian360 report
- **Integrating SOC automations 2,5 WEEKS**
  - Integrating n8n workflows into SIEM for alerting
  - Engineering a time based script to trim excessive documents for indices.
- **TheHive & MISP 0.5 WEEK**
  - Setting up TheHive and Cassandra, and ensuring they work smoothly with the existing containers

This document covers the following parts:

- An analysis of tools and decisions made
- A description of the realisations, covering each implemented stage
- A conclusion reflecting on the progress and what lies ahead

2. Analysis

At the beginning of the project, the employee from the internship company who was involved in the SOC project mentioned that he was very open to suggestions regarding the technologies to be used. The main requirements were that the tools had to be free and preferably popular, as widely used tools usually have better documentation and are generally considered more secure.

The internship team consisted of four interns and the employee. So even though there was a lot of freedom in choosing the tools, the final decisions were not always made by a single intern. Additionally, two Thomas More Mechelen interns started their internship one week earlier, which meant that some of the technologies had already been chosen by the time the rest of the team joined.

2.1. Tools and Technologies Used

WSL (Windows Subsystem for Linux) + Visual Studio Code

- used to create Linux environments directly on Windows machines.
- an easy and fast way approach was needed to run Ansible playbook commands on Windows, which is not possible under normal circumstances, so I chose to learn and install WSL.

Ansible

- A configuration management tool that was used to automate the provisioning and setup of infrastructure components.
- There was a need to automate the infrastructure setup using Infrastructure as Code (IaC) instead of configuring everything manually. This approach allowed me to first test the infrastructure locally and then deploy it to the production server simply by running the Ansible playbook.

Docker

- A tool that that was used to containerize and orchestrate services such as n8n, nginx, etc. …
- The n8n installation documentation used docker to set up n8n and nginx. I followed this flow and did the same instead of running the services directly on the virtual machine of the live server.

Elasticsearch, Logstash, Kibana & (ELK Stack) Grafana

- Elasticsearch stores log data, logstash filters logs before sending them to Elasticsearch, Kibana and Grafana visualize data via dashboards and graphs.
- The ELK stack is a free, open-source, and time-tested solution. It offers extensive technical documentation, community guides, and tutorials, making it an obvious choice.

n8n and AI agents (Hugging Face):

- n8n is used to automate workflows through a visual interface, allowing users to connect different services and define logic without having to write traditional code. It supports custom logic through JavaScript when needed. AI agents can be combined with Large Language Models to perform specific tasks based on the tools and data made available to them.
- An AI specialist introduced various tools relevant to the project. To explore their potential use, a short course on AI agents (Hugging Face) was followed as part of the research phase.

Nginx

- Nginx acts as an intermediate between services like e.g. n8n. This makes certain communications possible and more secure.
- This was needed this to make communication with n8n possible and secure. Before implementing this, I had to read up on some documentation regarding nginx.

OpenSSL

- A command-line tool that was needed to generate self-signed certificates for internal HTTPS usage.
- Self-signed certificates are needed to make HTTPS (secure) connections between certain services possible.

Jinja

- A templating engine that I used to fill in dynamic fields using a script.
- This tool was also used by another script that made the Cyberalarm report. I tried it out and it worked as expected.

Python

- Python is a widely used programming language known for its readability, versatility, and strong support for scripting and automation tasks.
- It was used during the internship for tasks such as automating infrastructure setup, parsing data, and interacting with APIs,

SCP (Secure Copy Protocol)

- A protocol used for transferring files securely between machines.
- SCP provided me with a quick way to copy from the live server to the desktop computer I was working on.

VPN

- A Virtual Private Network is used to securely connect remote systems to the project network.
- The desktop computer I was working on needed a VPN to connect to the network where the live server hosting the SIEM was running.

3. Realisations

3.1. Basic SIEM configuration

During the initial phase of the internship, the SIEM environment was successfully deployed on the production server. Real-time data was collected at five-minute intervals. The other interns successfully visualized the logs using Grafana. The setup process was automated using Infrastructure as Code (IaC) principles through Ansible playbooks, which were developed and configured for services such as Elasticsearch and Kibana. (Elastic, 2025) (elsticsearch, 2025) (Stringer, 2025)

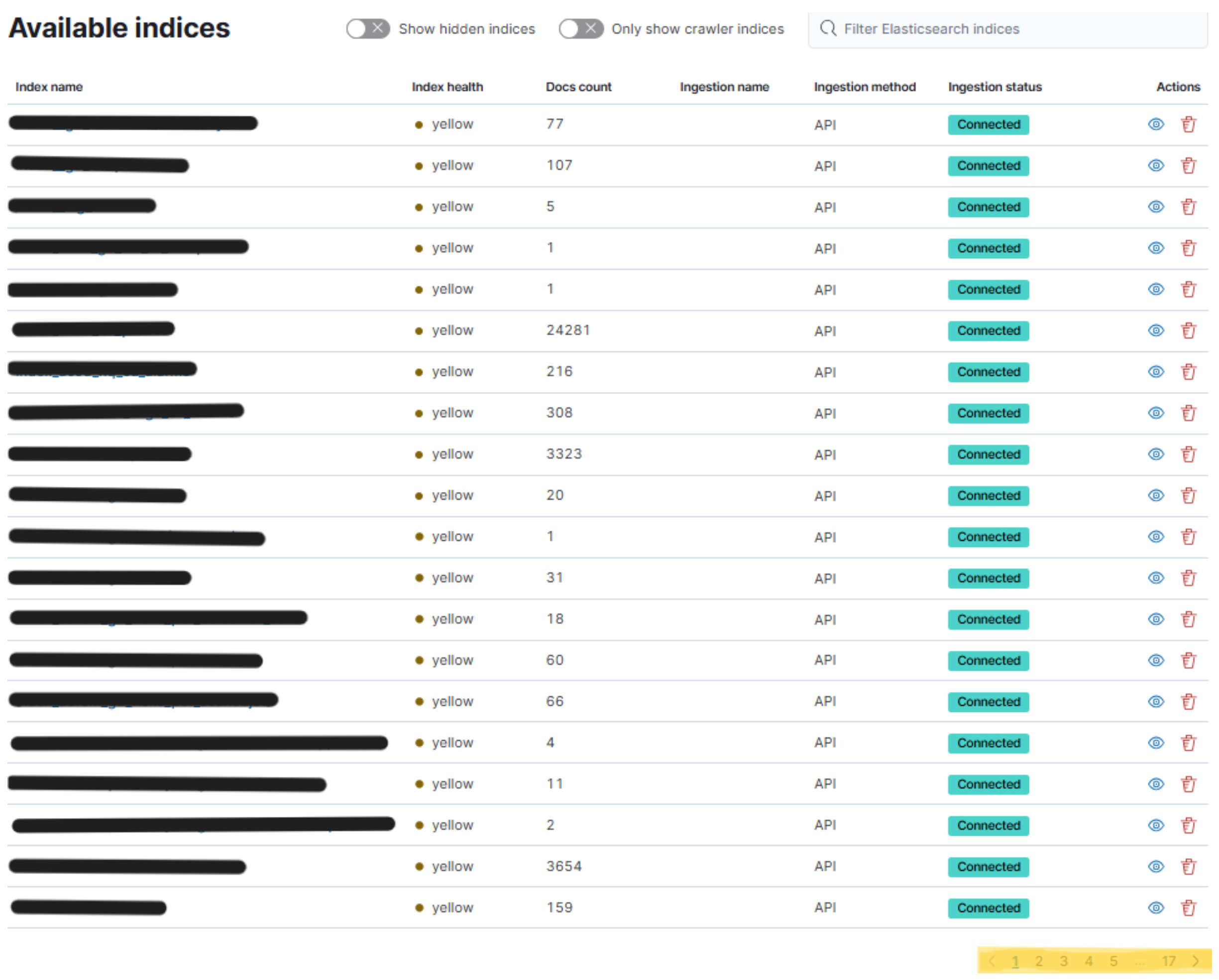
To facilitate development and testing, Windows Subsystem for Linux (WSL) was implemented on all project laptops. This enabled the team to run and test Ansible playbooks directly on Windows systems, allowing for local validation before deployment to the live environment.

Log collection and preprocessing workflows were configured in collaboration with another intern. These workflows ensured that log data was correctly parsed, cleaned, and forwarded to the appropriate services. To support this, approximately thirty scripts were developed by me to retrieve log data from various cybersecurity platforms used by client systems, including Guardian360, Cyberalarm, and WatchGuard. These scripts automated the collection of data through scheduled API calls and were tailored to the specific structure and requirements of each service.

Although some scripts required more complex logic, most shared a reusable structure. This modular approach improved development speed and allowed for easier maintenance and adaptation to changing API endpoints or data formats.

Support was also provided by me to team members from the Thomas More Mechelen campus by assisting with troubleshooting tasks and offering guidance on Linux fundamentals and command-line usage.

Additionally, a solution was developed to manage storage within Elasticsearch by automatically deleting outdated documents based on age and index size. This was achieved through a custom Python script that interfaced with Elasticsearch's REST API. The script was tested using sample indices and dry-run functionality to ensure correct behaviour before production deployment.



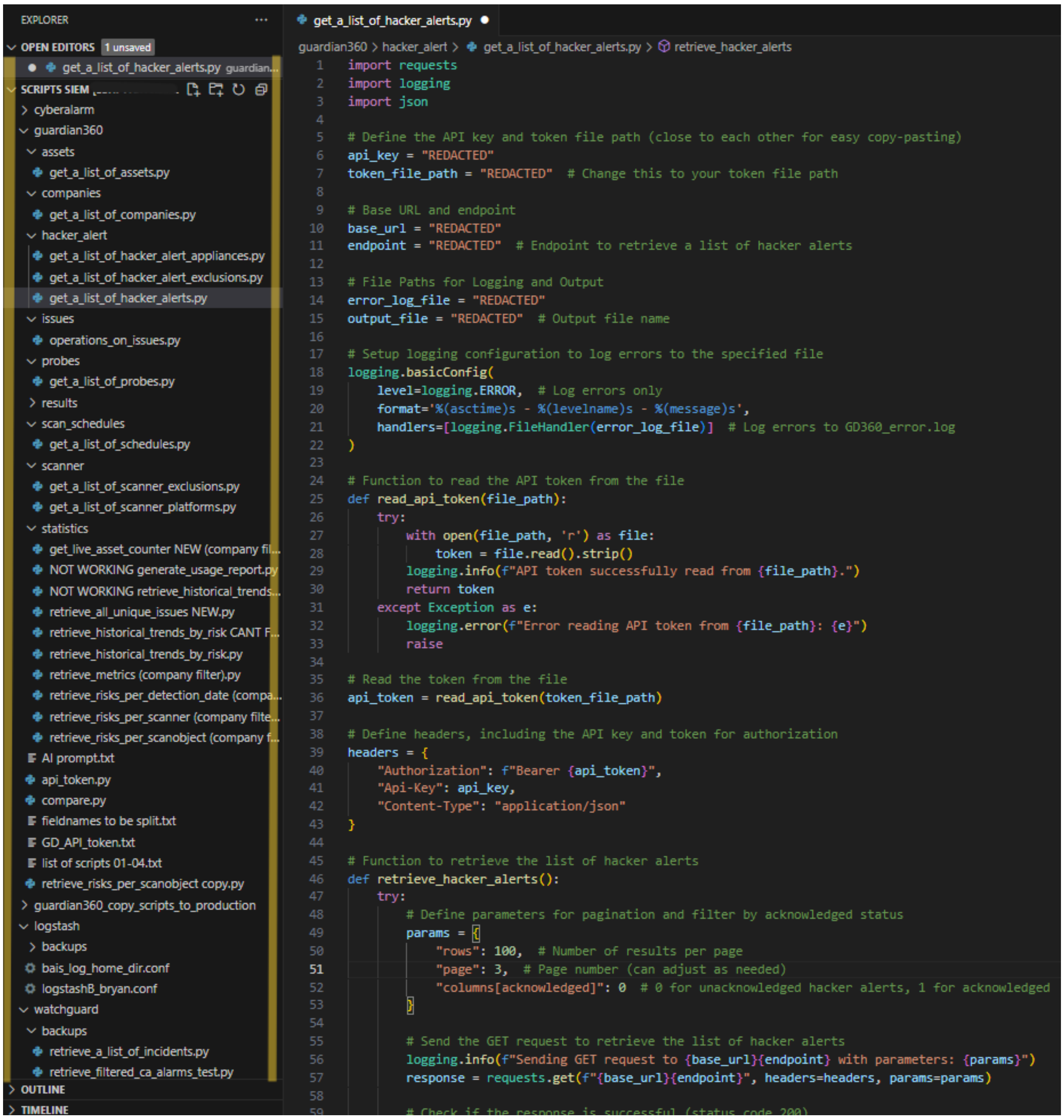
Available indices

Show hidden indices — Only show crawler indices — Filter Elasticsearch indices

Index name	Index health	Docs count	Ingestion name	Ingestion method	Ingestion status	Actions
[redacted]	● yellow	77		API	Connected	👁 🗑
[redacted]	● yellow	107		API	Connected	👁 🗑
[redacted]	● yellow	5		API	Connected	👁 🗑
[redacted]	● yellow	1		API	Connected	👁 🗑
[redacted]	● yellow	1		API	Connected	👁 🗑
[redacted]	● yellow	24281		API	Connected	👁 🗑
[redacted]	● yellow	216		API	Connected	👁 🗑
[redacted]	● yellow	308		API	Connected	👁 🗑
[redacted]	● yellow	3323		API	Connected	👁 🗑
[redacted]	● yellow	20		API	Connected	👁 🗑
[redacted]	● yellow	1		API	Connected	👁 🗑
[redacted]	● yellow	31		API	Connected	👁 🗑
[redacted]	● yellow	18		API	Connected	👁 🗑
[redacted]	● yellow	60		API	Connected	👁 🗑
[redacted]	● yellow	66		API	Connected	👁 🗑
[redacted]	● yellow	4		API	Connected	👁 🗑
[redacted]	● yellow	11		API	Connected	👁 🗑
[redacted]	● yellow	2		API	Connected	👁 🗑
[redacted]	● yellow	3654		API	Connected	👁 🗑
[redacted]	● yellow	159		API	Connected	👁 🗑

< 1 2 3 4 5 ... 17 >

Figure 1. Indices of stored logs



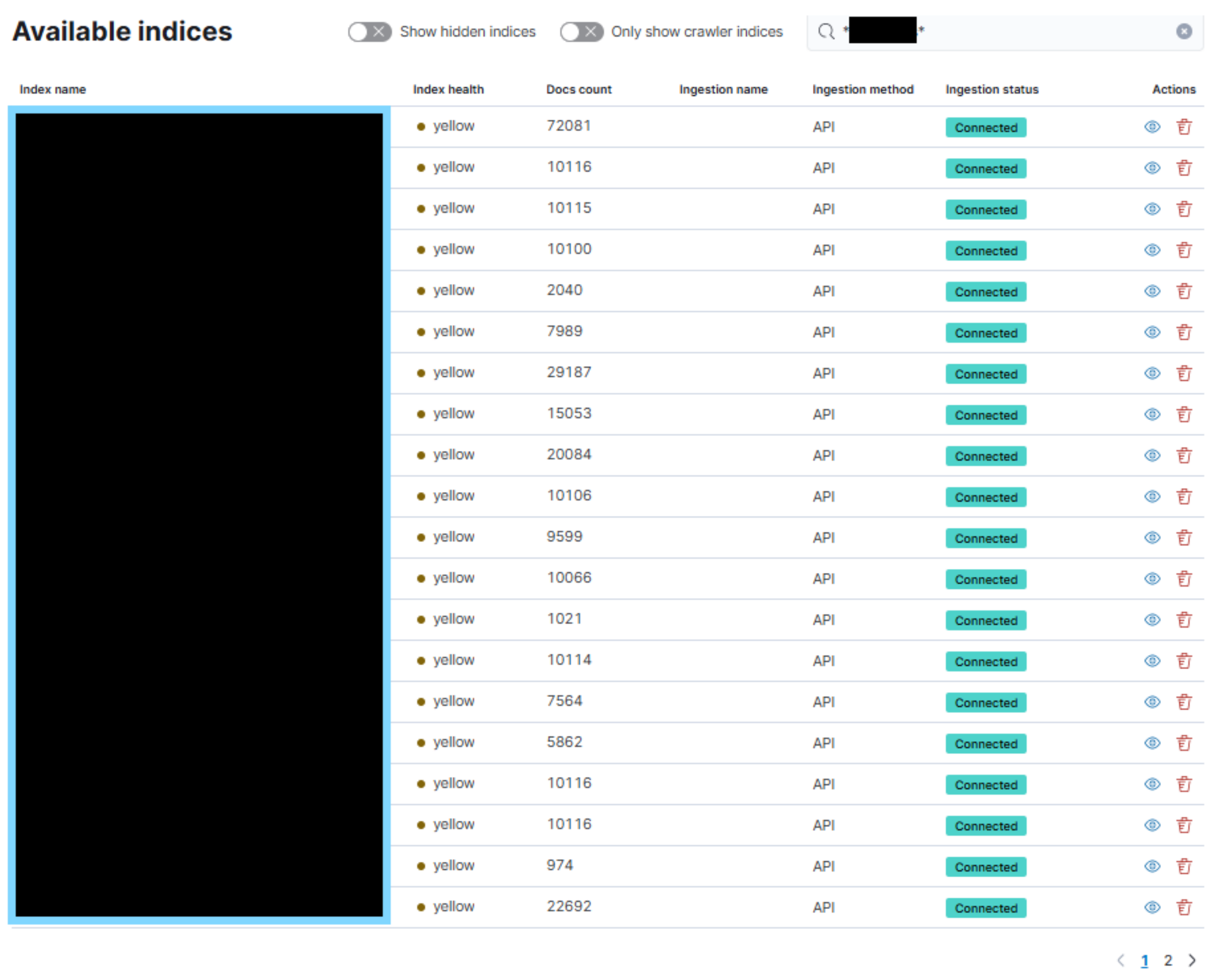
```
EXPLORER                                   ···
∨ OPEN EDITORS  1 unsaved
    ● get_a_list_of_hacker_alerts.py guardian...
∨ SCRIPTS SIEM [redacted]
  > cyberalarm
  ∨ guardian360
    ∨ assets
      get_a_list_of_assets.py
    ∨ companies
      get_a_list_of_companies.py
    ∨ hacker_alert
      get_a_list_of_hacker_alert_appliances.py
      get_a_list_of_hacker_alert_exclusions.py
      get_a_list_of_hacker_alerts.py
    ∨ issues
      operations_on_issues.py
    ∨ probes
      get_a_list_of_probes.py
    > results
    ∨ scan_schedules
      get_a_list_of_schedules.py
    ∨ scanner
      get_a_list_of_scanner_exclusions.py
      get_a_list_of_scanner_platforms.py
    ∨ statistics
      get_live_asset_counter NEW (company fil...
      NOT WORKING generate_usage_report.py
      NOT WORKING retrieve_historical_trends...
      retrieve_all_unique_issues NEW.py
      retrieve_historical_trends_by_risk CANT F...
      retrieve_historical_trends_by_risk.py
      retrieve_metrics (company filter).py
      retrieve_risks_per_detection_date (compa...
      retrieve_risks_per_scanner (company filte...
      retrieve_risks_per_scanobject (company f...
    AI prompt.txt
    api_token.py
    compare.py
    fieldnames to be split.txt
    GD_API_token.txt
    list of scripts 01-04.txt
    retrieve_risks_per_scanobject copy.py
  > guardian360_copy_scripts_to_production
  ∨ logstash
    > backups
    bais_log_home_dir.conf
    logstashB_bryan.conf
  ∨ watchguard
    ∨ backups
      retrieve_a_list_of_incidents.py
      retrieve_filtered_ca_alarms_test.py
> OUTLINE
> TIMELINE
```

get_a_list_of_hacker_alerts.py ●

guardian360 > hacker_alert > get_a_list_of_hacker_alerts.py > retrieve_hacker_alerts

```python
import requests
import logging
import json

# Define the API key and token file path (close to each other for easy copy-pasting)
api_key = "REDACTED"
token_file_path = "REDACTED"  # Change this to your token file path

# Base URL and endpoint
base_url = "REDACTED"
endpoint = "REDACTED"  # Endpoint to retrieve a list of hacker alerts

# File Paths for Logging and Output
error_log_file = "REDACTED"
output_file = "REDACTED"  # Output file name

# Setup logging configuration to log errors to the specified file
logging.basicConfig(
    level=logging.ERROR,  # Log errors only
    format='%(asctime)s - %(levelname)s - %(message)s',
    handlers=[logging.FileHandler(error_log_file)]  # Log errors to GD360_error.log
)

# Function to read the API token from the file
def read_api_token(file_path):
    try:
        with open(file_path, 'r') as file:
            token = file.read().strip()
        logging.info(f"API token successfully read from {file_path}.")
        return token
    except Exception as e:
        logging.error(f"Error reading API token from {file_path}: {e}")
        raise

# Read the token from the file
api_token = read_api_token(token_file_path)

# Define headers, including the API key and token for authorization
headers = {
    "Authorization": f"Bearer {api_token}",
    "Api-Key": api_key,
    "Content-Type": "application/json"
}

# Function to retrieve the list of hacker alerts
def retrieve_hacker_alerts():
    try:
        # Define parameters for pagination and filter by acknowledged status
        params = {
            "rows": 100,  # Number of results per page
            "page": 3,  # Page number (can adjust as needed)
            "columns[acknowledged]": 0  # 0 for unacknowledged hacker alerts, 1 for acknowledged
        }

        # Send the GET request to retrieve the list of hacker alerts
        logging.info(f"Sending GET request to {base_url}{endpoint} with parameters: {params}")
        response = requests.get(f"{base_url}{endpoint}", headers=headers, params=params)

        # Check if the response is successful (status code 200)
```

Figure 2. Extensive list of python scripts



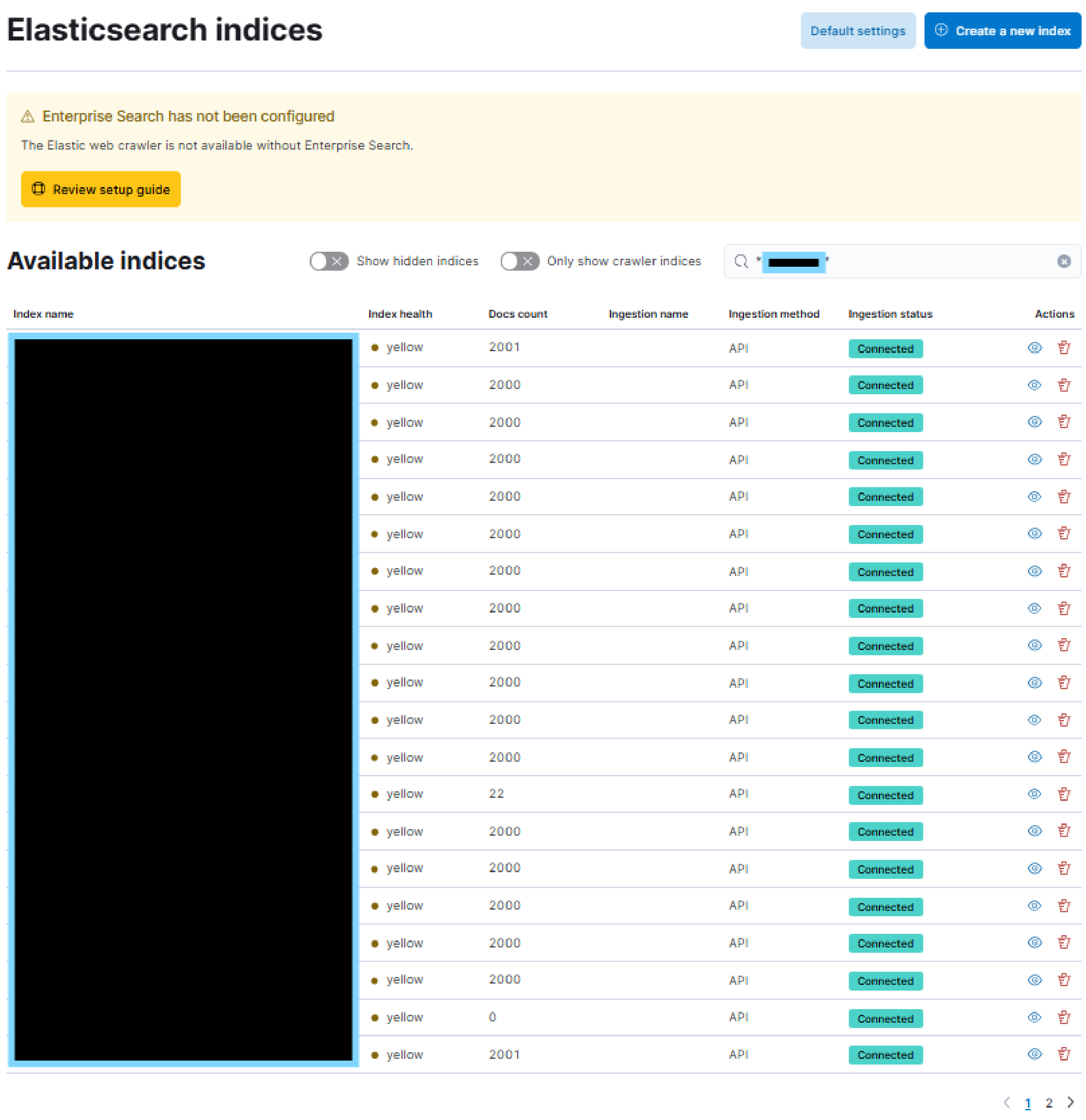
Available indices

Show hidden indices   Only show crawler indices   *[redacted]*

Index name	Index health	Docs count	Ingestion name	Ingestion method	Ingestion status	Actions
[redacted]	● yellow	72081		API	Connected	
[redacted]	● yellow	10116		API	Connected	
[redacted]	● yellow	10115		API	Connected	
[redacted]	● yellow	10100		API	Connected	
[redacted]	● yellow	2040		API	Connected	
[redacted]	● yellow	7989		API	Connected	
[redacted]	● yellow	29187		API	Connected	
[redacted]	● yellow	15053		API	Connected	
[redacted]	● yellow	20084		API	Connected	
[redacted]	● yellow	10106		API	Connected	
[redacted]	● yellow	9599		API	Connected	
[redacted]	● yellow	10066		API	Connected	
[redacted]	● yellow	1021		API	Connected	
[redacted]	● yellow	10114		API	Connected	
[redacted]	● yellow	7564		API	Connected	
[redacted]	● yellow	5862		API	Connected	
[redacted]	● yellow	10116		API	Connected	
[redacted]	● yellow	10116		API	Connected	
[redacted]	● yellow	974		API	Connected	
[redacted]	● yellow	22692		API	Connected	

< 1 2 >

Figure 3. documents count of indices before script deletes documents



Elasticsearch indices

Default settings | Create a new index

⚠ Enterprise Search has not been configured

The Elastic web crawler is not available without Enterprise Search.

Review setup guide

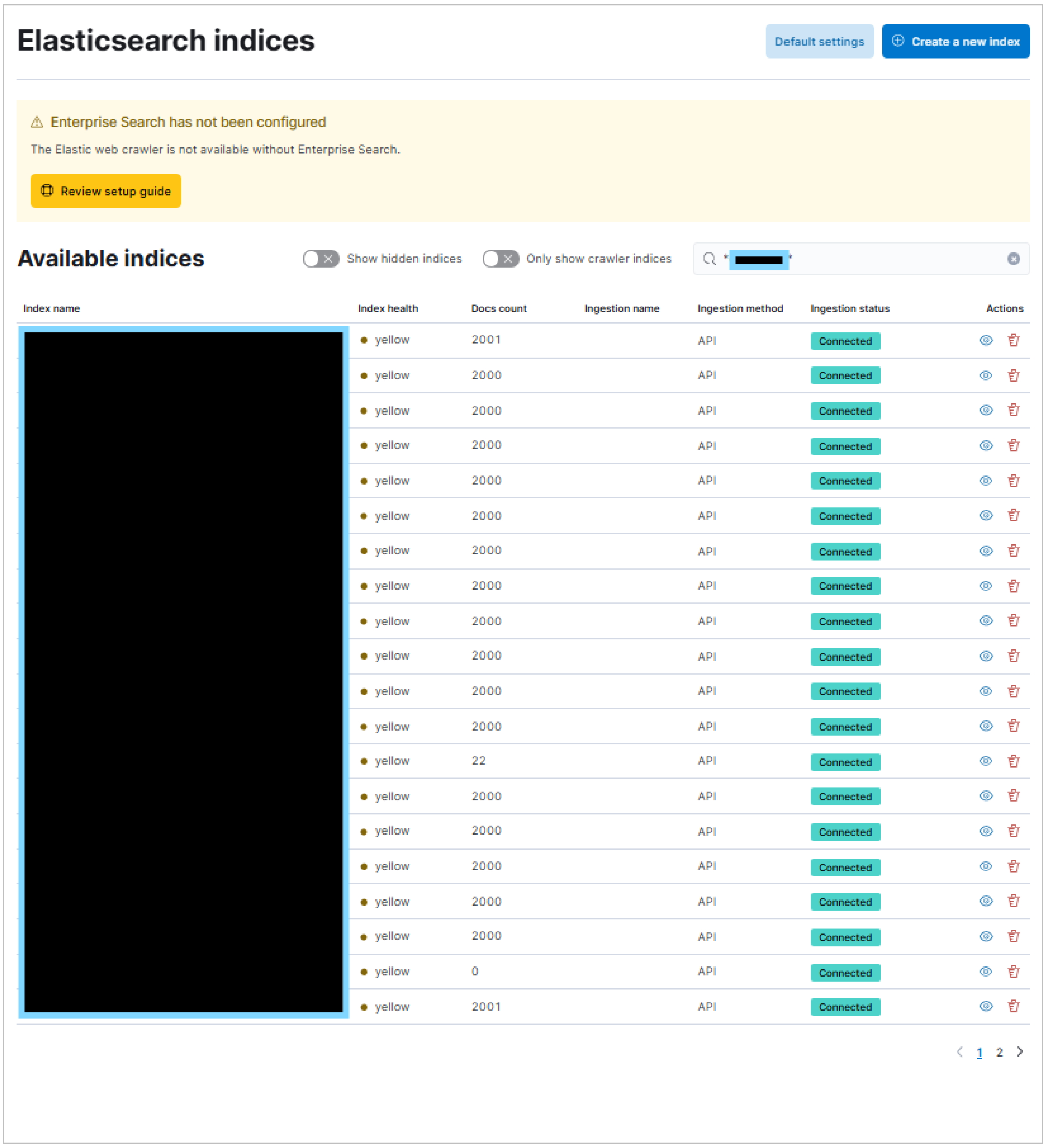
Available indices

Show hidden indices | Only show crawler indices | *[redacted]*

Index name	Index health	Docs count	Ingestion name	Ingestion method	Ingestion status	Actions
[redacted]	● yellow	2001		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	22		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	0		API	Connected	
[redacted]	● yellow	2001		API	Connected	

< 1 2 >

Figure 4. documents count of indices after script deletes documents



⚠ Enterprise Search has not been configured

The Elastic web crawler is not available without Enterprise Search.

[redacted]

Index name	Index health	Docs count	Ingestion name	Ingestion method	Ingestion status	Actions
[redacted]	● yellow	2001		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	22		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	2000		API	Connected	
[redacted]	● yellow	0		API	Connected	
[redacted]	● yellow	2001		API	Connected	

3.2. Visualization & Reporting

During this phase, I focused on automating the generation of a security report based on logs retrieved from Guardian360's vulnerability scans. These logs came in CSV format, consisting of 2 SCV files. I processed them into both interactive and printable reports. I developed an HTML template for the report layout and wrote a Python script that takes data from the log files and fills in the template automatically. In addition to HTML, I incorporated JavaScript for interactivity and CSS for styling. I added charts to show key metrics clearly, making the report easier to read and understand.

in the end, I had a dynamic, interactive HTML report with:

- A table of contents
- Page numbering
- A styled cover page
- Automatically populated issue descriptions, solutions, technical data, and severity levels

A PDF version of the report is also generated besides the HTML version; it kept the look of the HTML version and made it more archive friendly. I tested the reports regularly to make sure all the data was accurate and displayed correctly. Additionally, I implemented multilingual support by adding translation functionality for both static and dynamic content. While predefined translations were used for static sections such as the introduction, the dynamic fields including issue descriptions, solutions, and technical details were translated using the DeepL API. This allows the report to be generated in Dutch, French, or English, depending on the target audience. (Deepl, 2025). The DeepL functionality was later scrapped due to concerns about inaccurate translations of certain technical terms, although the code itself was fully functional.

Two CSV files are used to generate the reports. The first is the main CSV file containing the core scan data, while the second serves as a mapping file to link each company's scanner platform to the correct log entries. Although the main CSV does not directly specify the scanner platform used, it does include the scan object field. This field is also present in the second CSV, which makes it possible to establish the correct connections between platforms and their corresponding logs.

REPORT

Guardian360

[redacted]

vulnerability scans

[redacted]

Netcure
Davincilaan 1
1930 Zaventem
BE 0459.551.653
02 712 00 80

Page 1 of 17

Figure 5. Guardian 360 Report cover page

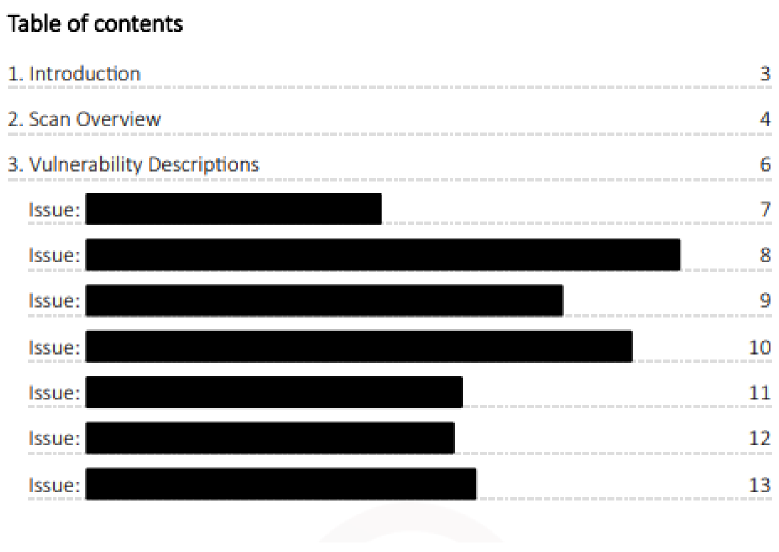
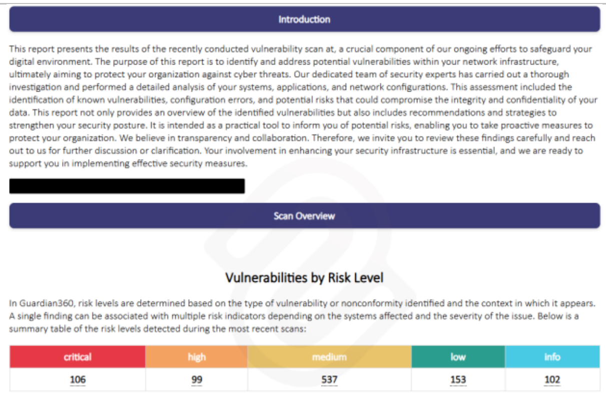


Table of contents

Figure 6. Guardian 360 Report table of contents



Introduction

This report presents the results of the recently conducted vulnerability scan at, a crucial component of our ongoing efforts to safeguard your digital environment. The purpose of this report is to identify and address potential vulnerabilities within your network infrastructure, ultimately aiming to protect your organization against cyber threats. Our dedicated team of security experts has carried out a thorough investigation and performed a detailed analysis of your systems, applications, and network configurations. This assessment included the identification of known vulnerabilities, configuration errors, and potential risks that could compromise the integrity and confidentiality of your data. This report not only provides an overview of the identified vulnerabilities but also includes recommendations and strategies to strengthen your security posture. It is intended as a practical tool to inform you of potential risks, enabling you to take proactive measures to protect your organization. We believe in transparency and collaboration. Therefore, we invite you to review these findings carefully and reach out to us for further discussion or clarification. Your involvement in enhancing your security infrastructure is essential, and we are ready to support you in implementing effective security measures.

[redacted]

Scan Overview

Vulnerabilities by Risk Level

In Guardian360, risk levels are determined based on the type of vulnerability or nonconformity identified and the context in which it appears. A single finding can be associated with multiple risk indicators depending on the systems affected and the severity of the issue. Below is a summary table of the risk levels detected during the most recent scans:

critical	high	medium	low	info
106	99	537	153	102

Figure 7. Guardian 360 Report Intro and risk table

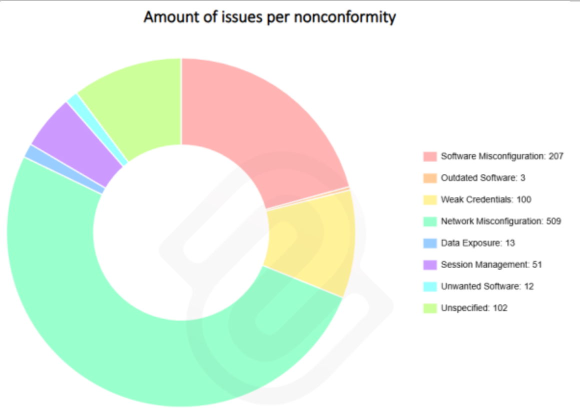


Figure 8. Guardian 360 Report Donut chart

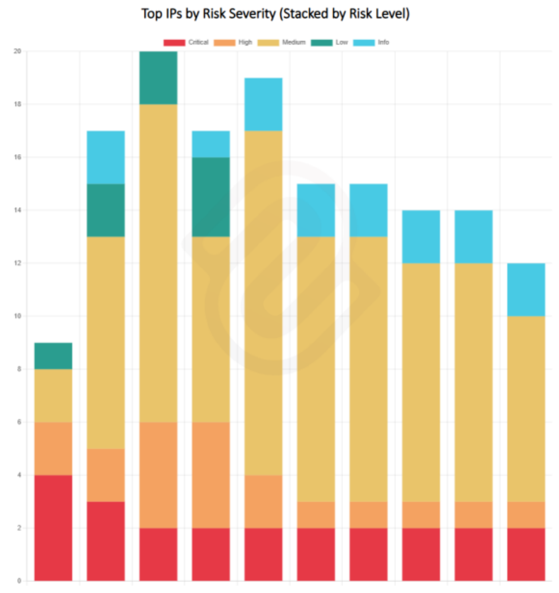
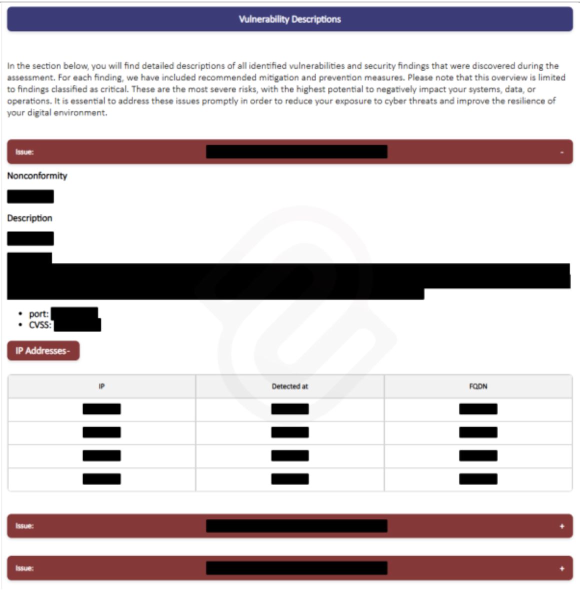


Figure 9. Guardian 360 Report stacked barchart



Vulnerability Descriptions

In the section below, you will find detailed descriptions of all identified vulnerabilities and security findings that were discovered during the assessment. For each finding, we have included recommended mitigation and prevention measures. Please note that this overview is limited to findings classified as critical. These are the most severe risks, with the highest potential to negatively impact your systems, data, or operations. It is essential to address these issues promptly in order to reduce your exposure to cyber threats and improve the resilience of your digital environment.

Issue: [redacted] -

Nonconformity

[redacted]

Description

[redacted]

[redacted]

- port: [redacted]
- CVSS: [redacted]

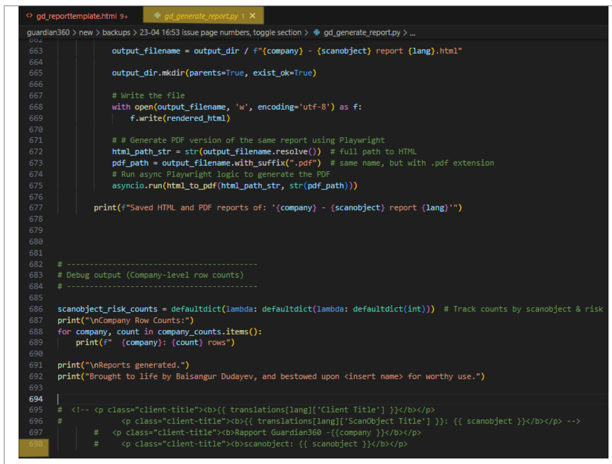
IP Addresses -

IP	Detected at	FQDN
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]
[redacted]	[redacted]	[redacted]

Issue: [redacted] +

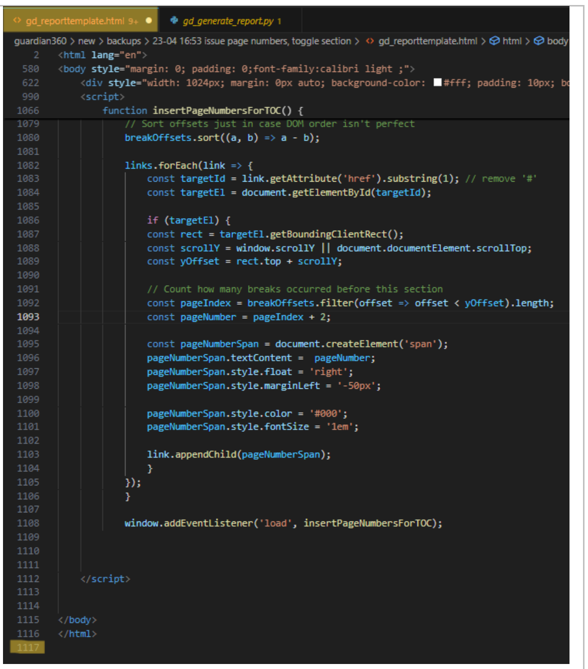
Issue: [redacted] +

Figure 10. Issues Guardian 360 Issues pages


gd_reporttemplate.html 9+
gd_generate_report.py 1
guardian360 > new > backups > 23-04 16:53 issue page numbers, toggle section > gd_generate_report.py > ...

```python
            output_filename = output_dir / f"{company} - {scanobject} report {lang}.html"

            output_dir.mkdir(parents=True, exist_ok=True)

            # Write the file
            with open(output_filename, 'w', encoding='utf-8') as f:
                f.write(rendered_html)

            # # Generate PDF version of the same report using Playwright
            html_path_str = str(output_filename.resolve())  # full path to HTML
            pdf_path = output_filename.with_suffix(".pdf")  # same name, but with .pdf extension
            # Run async Playwright logic to generate the PDF
            asyncio.run(html_to_pdf(html_path_str, str(pdf_path)))

        print(f"Saved HTML and PDF reports of: '{company} - {scanobject} report {lang}'")




# -----------------------------------------
# Debug output (Company-level row counts)
# -----------------------------------------

scanobject_risk_counts = defaultdict(lambda: defaultdict(lambda: defaultdict(int)))  # Track counts by scanobject & risk
print("\nCompany Row Counts:")
for company, count in company_counts.items():
    print(f"  {company}: {count} rows")

print("\nReports generated.")
print("Brought to life by Baisangur Dudayev, and bestowed upon <insert name> for worthy use.")


#  <!-- <p class="client-title"><b>{{ translations[lang]['Client Title'] }}</b></p>
#            <p class="client-title"><b>{{ translations[lang]['ScanObject Title'] }}: {{ scanobject }}</b></p> -->
        #   <p class="client-title"><b>Rapport Guardian360 -{{company }}</b></p>
        #     <p class="client-title"><b>scanobject: {{ scanobject }}</b></p>
```

Figure 11. Script used to generate report, 700 lines of code


gd_reporttemplate.html 9+
gd_generate_report.py 1
guardian360 > new > backups > 23-04 16:53 issue page numbers, toggle section > gd_reporttemplate.html > html > body

```html
<html lang="en">
<body style="margin: 0; padding: 0;font-family:calibri light ;">
    <div style="width: 1024px; margin: 0px auto; background-color: #fff; padding: 10px; bo
    <script>
        function insertPageNumbersForTOC() {
            // Sort offsets just in case DOM order isn't perfect
            breakOffsets.sort((a, b) => a - b);

            links.forEach(link => {
                const targetId = link.getAttribute('href').substring(1); // remove '#'
                const targetEl = document.getElementById(targetId);

                if (targetEl) {
                const rect = targetEl.getBoundingClientRect();
                const scrollY = window.scrollY || document.documentElement.scrollTop;
                const yOffset = rect.top + scrollY;

                // Count how many breaks occurred before this section
                const pageIndex = breakOffsets.filter(offset => offset < yOffset).length;
                const pageNumber = pageIndex + 2;

                const pageNumberSpan = document.createElement('span');
                pageNumberSpan.textContent =  pageNumber;
                pageNumberSpan.style.float = 'right';
                pageNumberSpan.style.marginLeft = '-50px';

                pageNumberSpan.style.color = '#000';
                pageNumberSpan.style.fontSize = '1em';

                link.appendChild(pageNumberSpan);
                }
            });
            }

            window.addEventListener('load', insertPageNumbersForTOC);



    </script>


</body>
</html>
```

Figure 12. Script template used to generate report, 1100 lines of code

3.3. Integrating SIEM automation

I set up an n8n Docker container and configured a Nginx container as a reverse proxy. (Nginx, 2025)

I secured them with HTTPS using a mix of self-made certificates and ones that were already set up from Elasticsearch. I also created an IaC playbook for this setup, so it can be easily replicated on the live server after testing. Ensuring that n8n was up and running was crucial during this phase, as the work of all team members depended on it. (n8n, n.d.)

I built an automated workflow that pulls emails from a Gmail mailbox I set up. This automation looks at the content of the emails to decide whether to stop or continue contacting certain employees in the company through Vocal Notify. Vocal Notify is a tool that can be configured to call employees when specific alerts or events occur. However, if someone wants to stop receiving these calls, they can simply send an email to the mailbox I created.

To prevent the mailbox from requiring manual maintenance, I developed a Google Apps Script that deletes emails that have already been read, scheduled to run monthly. (Apps Script | Google for Developers, 2024)

The n8n workflow is configured to retrieve only unread emails and marks them as read after processing to avoid duplicate handling.

The original workflow using the Google Script was scrapped due to certain requests from upper management. Later, I changed and extended the workflow so that it could send out emails based on both the **type** and **volume** of alerts for Cyberalarm.

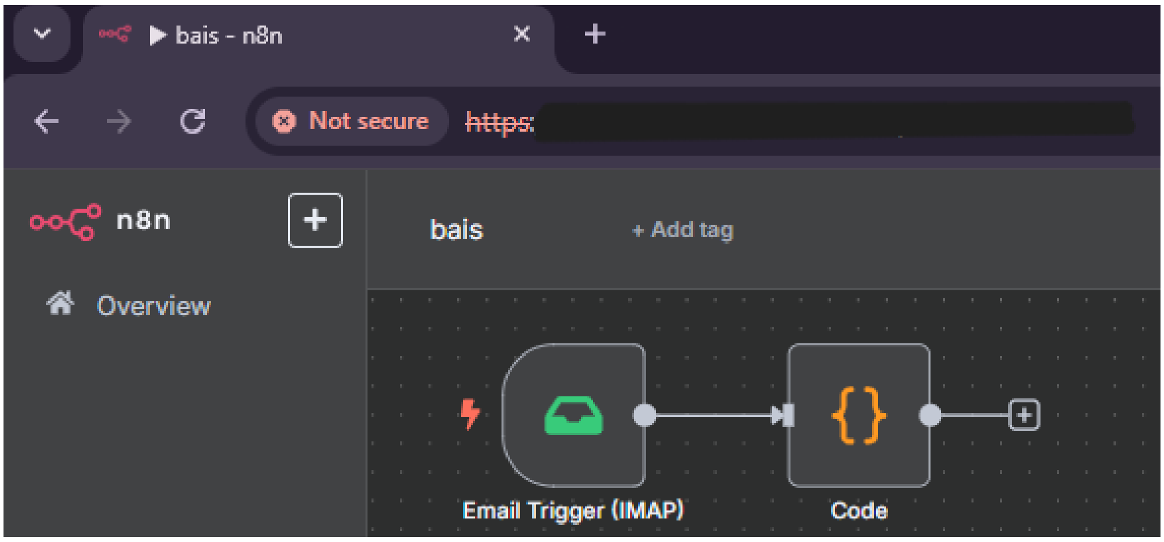
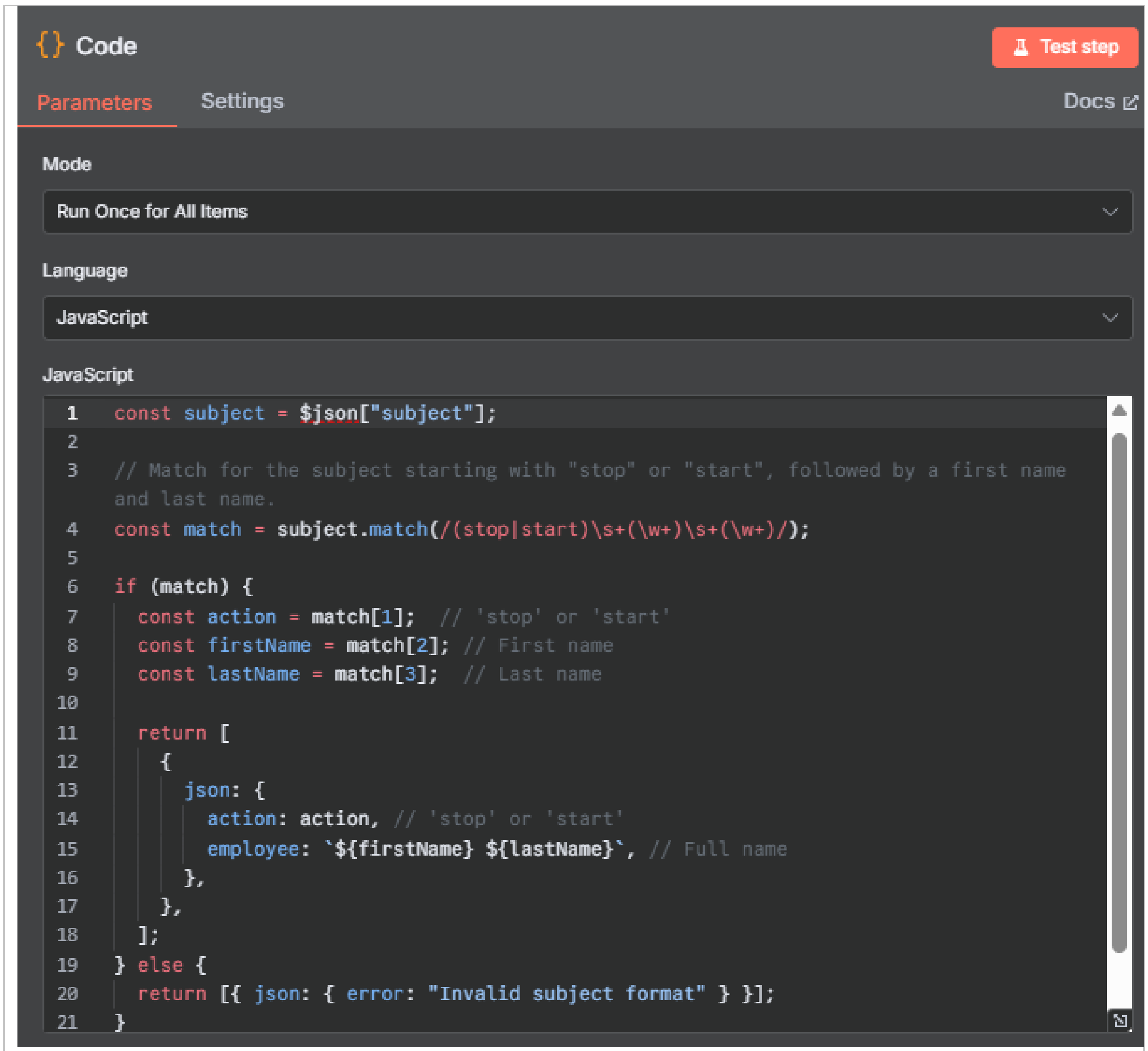


Figure 13. Running n8n container with nodes to and act on retrieve mails



Mode

Run Once for All Items

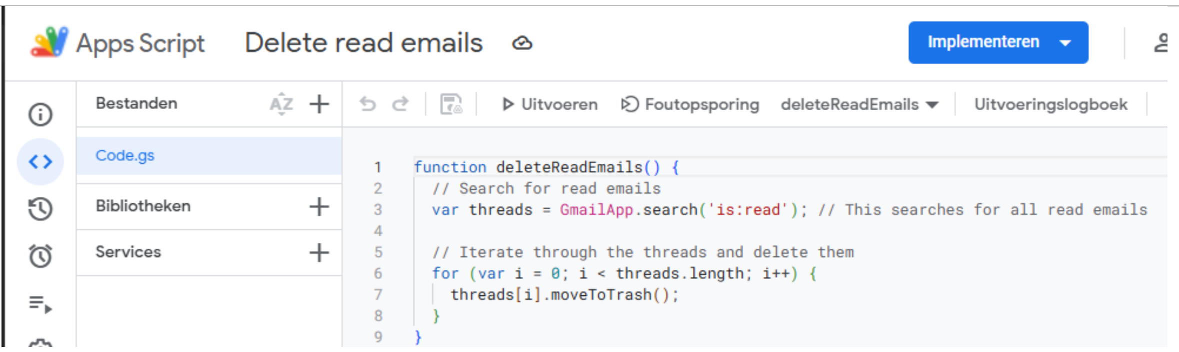
Language

JavaScript

JavaScript

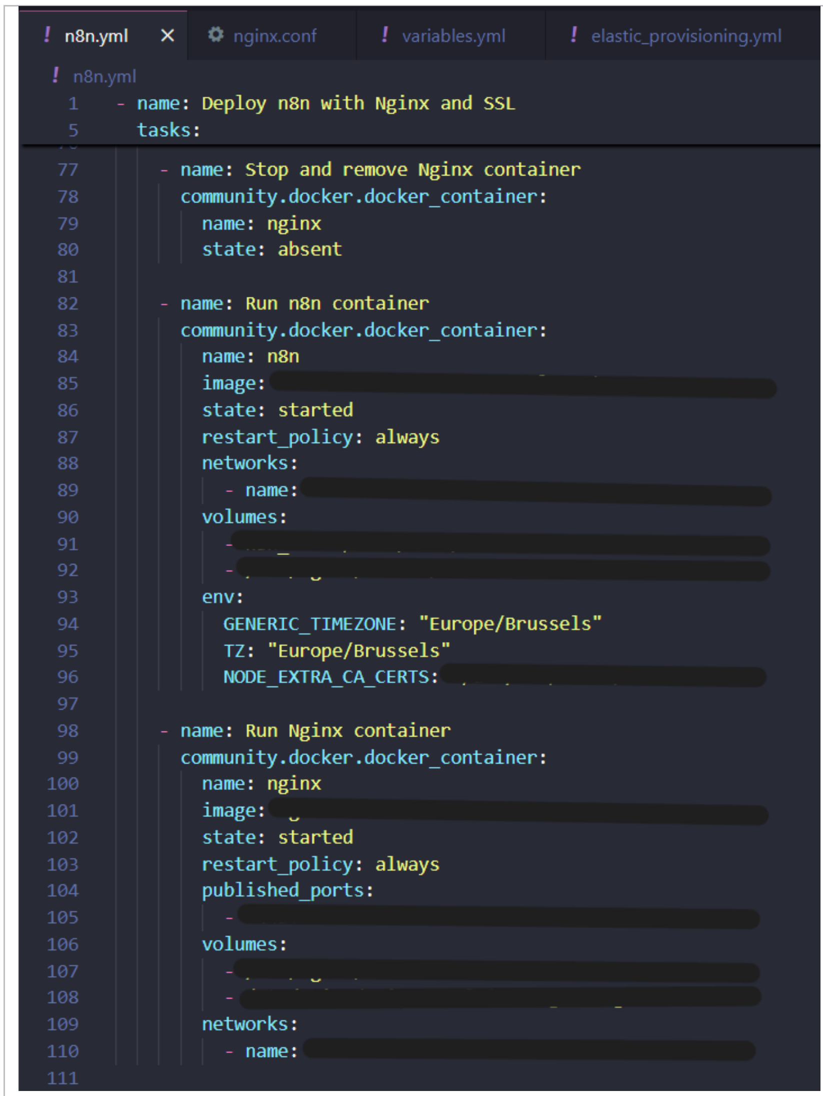
```
const subject = $json["subject"];

// Match for the subject starting with "stop" or "start", followed by a first name
and last name.
const match = subject.match(/(stop|start)\s+(\w+)\s+(\w+)/);

if (match) {
  const action = match[1];  // 'stop' or 'start'
  const firstName = match[2]; // First name
  const lastName = match[3];  // Last name

  return [
    {
      json: {
        action: action, // 'stop' or 'start'
        employee: `${firstName} ${lastName}`, // Full name
      },
    },
  ];
} else {
  return [{ json: { error: "Invalid subject format" } }];
}
```

Figure 14. Nn8n node code to take action on email's contents



```
function deleteReadEmails() {
  // Search for read emails
  var threads = GmailApp.search('is:read'); // This searches for all read emails

  // Iterate through the threads and delete them
  for (var i = 0; i < threads.length; i++) {
    threads[i].moveToTrash();
  }
}
```

Figure 15. Google app script to delete read mails



```yaml
- name: Deploy n8n with Nginx and SSL
  tasks:
    - name: Stop and remove Nginx container
      community.docker.docker_container:
        name: nginx
        state: absent

    - name: Run n8n container
      community.docker.docker_container:
        name: n8n
        image: [redacted]
        state: started
        restart_policy: always
        networks:
          - name: [redacted]
        volumes:
          - [redacted]
        env:
          GENERIC_TIMEZONE: "Europe/Brussels"
          TZ: "Europe/Brussels"
          NODE_EXTRA_CA_CERTS: [redacted]

    - name: Run Nginx container
      community.docker.docker_container:
        name: nginx
        image: [redacted]
        state: started
        restart_policy: always
        published_ports:
          - [redacted]
        volumes:
          - [redacted]
        networks:
          - name: [redacted]
```

Figure 16. Playbook to run n8n and NGINX container

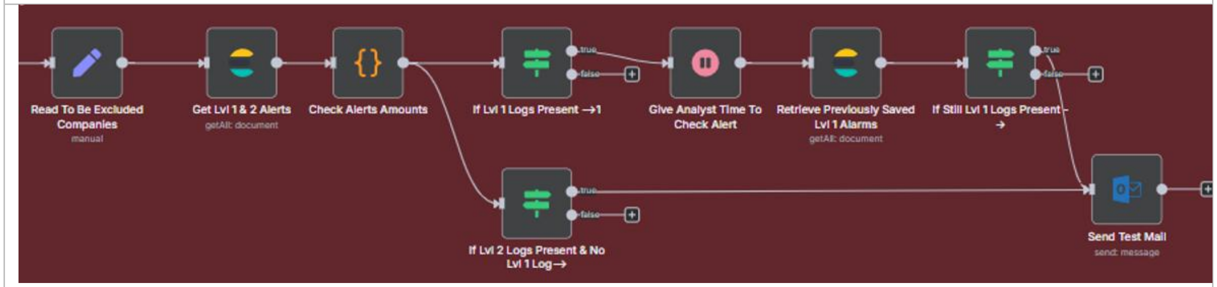

Read To Be Excluded Companies
manual
Get Lvl 1 & 2 Alerts
getAll: document
Check Alerts Amounts
If Lvl 1 Logs Present →1
true
false
Give Analyst Time To Check Alert
Retrieve Previously Saved Lvl 1 Alarms
getAll: document
If Still Lvl 1 Logs Present →
true
false
If Lvl 2 Logs Present & No Lvl 1 Log→
true
false
Send Test Mail
send: message

Figure 17. Updated Vocal Notify workflow to make calls based on logs

3.4. Integrating TheHive

I conducted research on both TheHive and MISP, including completing their respective TryHackMe rooms. Based on this analysis, I chose to proceed with installing TheHive and decided to omit MISP.

During the setup, I deployed the required Cassandra container alongside TheHive. TheHive also requires Nginx and Elasticsearch to function properly. Since we already had existing containers for both Nginx and Elasticsearch in our SOC environment, I configured TheHive to connect to those instead of creating new ones. This approach helped optimize resource usage and improved the overall efficiency of the system.

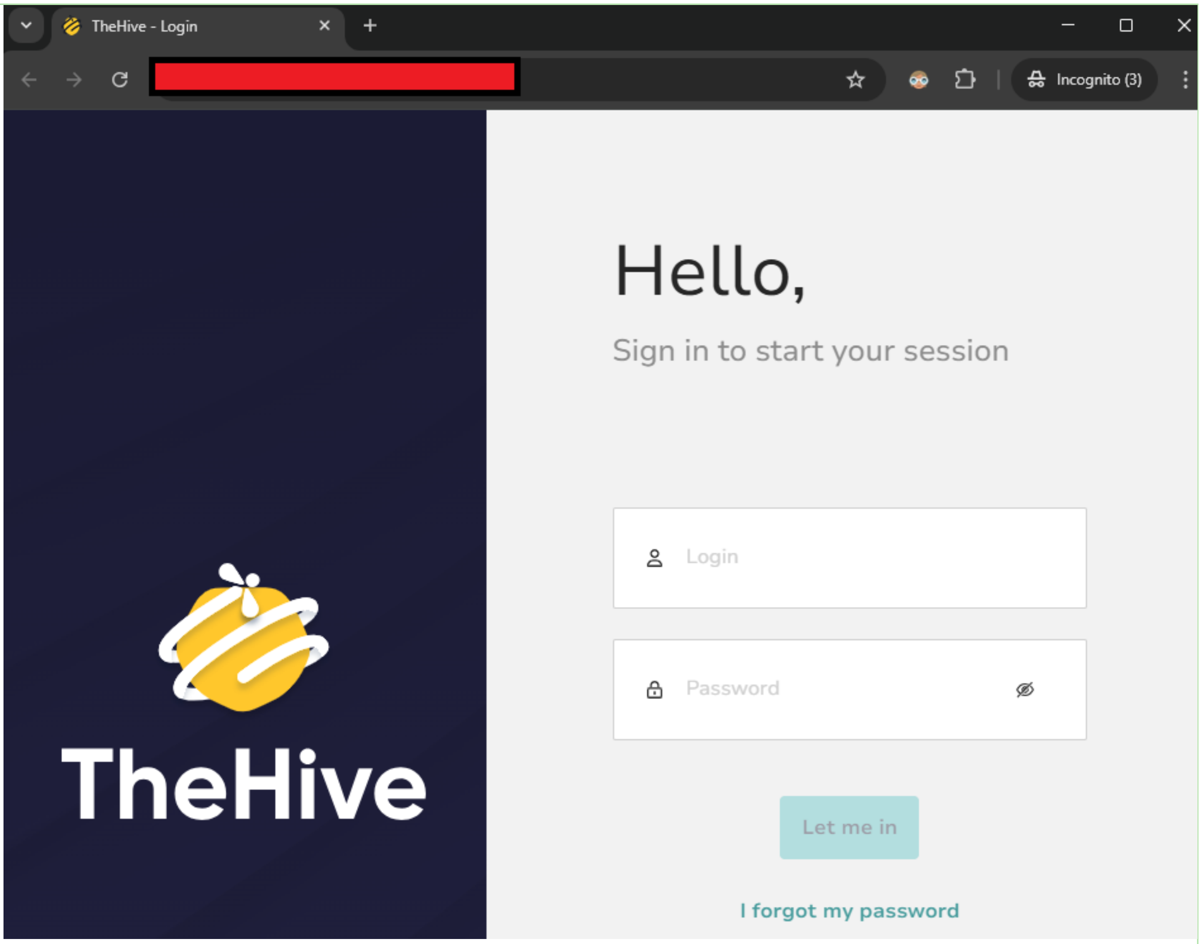


Figure 18. TheHive running

4. Conclusion

This report summarizes the progress made during the internship focused on developing and enhancing a fully functional Security Information and Event Management (SIEM) solution for Netcure. The SIEM environment aims to centralize log collection, alerting, and reporting to improve security monitoring both for the company and its clients.

The project followed a structured timeline, starting with the deployment of the ELK stack components and automated infrastructure using Ansible to enable reliable, real-time log collection and visualization. A multilingual reporting system was developed to transform vulnerability scan data into interactive reports, supporting informed decision-making.

The integration of n8n facilitated workflow automation and improved alert management by connecting with a custom mailbox, causing Vocal Notify alerts to be more efficient.

The SOC environment consists of an intelligent, automated platform that consolidates diverse security data sources, improving situational awareness and operational efficiency.

5. References

Apps Script | Google for Developers. (2024, 31 10). Opgeroepen op 4 2025, van developers.google.com: https://developers.google.com/apps-script

Deepl. (2025, 4 23). *Introduction | DeepL API Documentation*. Opgeroepen op 4 23, 2025, van developers.deepl.com: https://developers.deepl.com/docs

Elastic. (2025, 4 23). *Logstash | Elastic Documentation*. Opgehaald van elastic.co: https://www.elastic.co/docs/reference/logstash

elsticsearch. (2025, 04 23). *Index basics |Elastic Docs*. Opgehaald van www.elastic.co: https://www.elastic.co/docs/manage-data/data-store/index-basics

n8n. (sd). *Docker | n8n Docs*. Opgeroepen op 04 23, 2025, van docs.n8n.io: https://docs.n8n.io/hosting/installation/docker/

Nginx. (2025, 3 23). *NGINX Reverse Proxy | NGINX Documentation*. Opgehaald van https://docs.nginx.com/nginx/: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/

Stringer, T. (2025, 4 3). *Developing modules | Ansible Community Documentation*. Opgeroepen op 4 3, 2025, van docs.ansible.com: https://docs.ansible.com/ansible/latest/dev_guide/developing_modules_general.html#environment-setup