Data Science and Analytics

ORG Data
Thread and Weakness Analysis

Student 1:	Vijay Singh (7025700)
Student 2:	Manoj S (7025649)
Student 3:	Vatsal Mahajan (7025694)
Course of Studies:	Industrial Informatics
First examiner:	Prof. Dr. Elmar Wings
Second examiner:	Prof. Dr. Walter Colombo

Submission date: January 7, 2025

University of Applied Sciences Emden/Leer · Faculty of Technology · Mechanical Engineering Department
Constantiaplatz 4 · 26723 Emden · http://www.hs-emden-leer.de

Contents

1 Introduction

1.1 Introduction

The focus of this analysis is to identify and mitigate potential threats and weaknesses within Orgadata's SimplyTag system. SimplyTag facilitates quick access to construction-related data through a web app, making it essential to safeguard against malicious actors attempting to exploit the system. This involves analyzing logs to detect suspicious activity and ensure data integrity.

1.2 Challenges and Results

Challenges:

- Handling large-scale logs to pinpoint anomalies as shown in the figure 1.1.
- Differentiating legitimate user activity from malicious attempts.
- Establishing efficient protocols to respond to detected threats.

Results Achieved:

- Implementation of enhanced monitoring protocols using trace IDs and HTTP status codes.
- Reduced data breaches by identifying and blocking unauthorized requests.

Figure 1.1: Original Data format

2 Positioning Your Analytics

2.1 Existing Solutions

The analytics developed for Orgadata's SimplyTag system are positioned uniquely when compared to conventional tools. Traditional solutions like Intrusion Detection Systems (IDS) [Bra20] and log monitoring platforms (e.g., Splunk, ELK Stack) [IEE22] offer general frameworks for detecting anomalies and breaches. However, they often lack customization tailored to specific applications. Orgadata's approach is distinguished by its precise utilization of trace IDs, HTTP status codes, user agent patterns, and paths to monitor requests in real-time.

2.2 State-of-the-Art

2.2.1 Overview of Current Solutions

- Existing tools such as Splunk, ELK Stack and Intrusion Detection Systems (IDS) provide robust frameworks for monitoring and analyzing security events. These solutions excel at processing vast amounts of log data and identifying potential threats in general scenarios.
- Security platforms often use rule-based or heuristic approaches for anomaly detection but may struggle with domain-specific customizations.

2.2.2 Capabilities and Limitations

- **Capabilities:** Tools like Splunk and ELK Stack provide scalability, integration options, and comprehensive dashboards for security analytics. IDS focuses on real-time threat detection.
- **Limitations:** Lack of precise tailoring for SimplyTag's requirements, such as analyzing specific HTTP status codes and user agent patterns. High false-positive rates and inability to integrate seamlessly with Orgadata's infrastructure are additional challenges.

2.2.3 Relevance to Analytics

The SimplyTag analytics focus on filling these gaps by leveraging domain-specific insights, such as monitoring trace IDs and HTTP response patterns to identify

anomalies and unauthorized activity.

1. The use of trace IDs enables seamless tracking of individual requests across logs, allowing for detailed insights into potential vulnerabilities.
2. By monitoring HTTP status codes, the system identifies and flags suspicious patterns, such as unexpected 200 codes for unauthorized paths.
3. Integration of user agent analysis ensures that illegitimate devices or configurations can be quickly identified and addressed.

2.2.4 Emerging Trends

- AI-driven anomaly detection is gaining traction, enabling systems to learn and adapt to evolving threats.
- Zero-trust architecture and real-time monitoring advancements align with SimplyTag's goals of enhancing data security and operational reliability.

This focused methodology bridges gaps left by traditional systems, providing a solution that is both targeted and scalable for SimplyTag's operational needs.

3 Application Sector

3.1 Application Sector

This project belongs to the Industry domain, specifically the Construction Software sector, with a focus on::

- **Threat Detection:** Monitoring system logs to identify malicious activity and unauthorized data access.
- **Data Integrity Assurance:** Safeguarding construction-related data from breaches to ensure accurate and reliable information.
- **Operational Security Optimization:** Enabling proactive measures to address potential weaknesses and improve system resilience.

By contributing to the broader domain of secure digital construction tools, the project aligns with emerging trends like data-driven operational efficiency and advanced cybersecurity measures tailored for industry-specific applications.

4 Purpose of Your Analytics

4.1 What It Does

This project belongs to the Industry domain, specifically the Construction Software sector, with a focus on:

- **Condition Monitoring:** Tracks HTTP requests in real-time to identify anomalies. This includes analyzing user agents and status codes to pinpoint irregularities that could indicate potential breaches or vulnerabilities.
- **Prevention:** Identifies and blocks malicious requests before they lead to breaches. Proactive security measures ensure that sensitive data and system integrity remain uncompromised.
- **Diagnosis:** Conducts post-incident analysis to refine future detection capabilities. By learning from past incidents, the analytics evolve to handle emerging threats more effectively.
- **Optimization:** Enhances system performance by ensuring secure operations without disrupting user experience. This involves balancing security measures with system usability, especially in high-demand environments like construction software solutions.

5 Major Requirements

5.1 Requirements

- **Structural:** The analytics must seamlessly integrate with existing SimplyTag and Orgadata infrastructure.
- **Behavioral:** Real-time anomaly detection with high accuracy and minimal false positives.
- **Functional:** Efficient parsing of HTTP status codes and trace IDs to identify suspicious activity.
- **Technological:** Implementation of advanced algorithms for log analysis and anomaly detection.

This structured approach ensures that the Threat and Weakness Analysis for Orgadata's SimplyTag system is robust, effective, and well-positioned within industry standards.

6 Knowledge Discovery in Databases (KDD) Process

KDD plays a pivotal role in helping organizations navigate the over whelming volumes of data generated in today's information-driven world. KDD is a structural approach to data analysis to discover and make explicit knowledge available in extensive data sets [Win24]. This methodology involves a series of well-defined steps, including data selection, preprocessing, transformation and analysis, leading to the generation of actionable knowledge as shown in the figure 6.1. [MFH00]

KDD is particularly relevant for analyzing log file data in the context of threat and weakness analysis. The structured approach ensures that patterns and anomalies are identified efficiently, enabling the identification of targeted solutions for improving system security and operational performance. [FPS97]

In this project, the KDD process will serve as the backbone for organizing and implementing analytics tasks, ensuring a systematic exploration of the data to uncover valuable insights.

6.1 Topic Description

This section of the report outlines the objective to focuses on identifying anomalies, vulnerabilities, and inefficiencies within the dataset. The primary goal is to uncover potential security risks, operational weaknesses, and performance bottlenecks through a comprehensive analysis of system logs. The Knowledge Discovery in Databases (KDD) process was utilized to manage the complexity of the dataset effectively. This structured methodology enabled efficient data handling, cleaning and preparation. By leveraging the KDD process, the analysis distinguished patterns indicative of normal and abnormal behavior, identified clusters of high-risk events based on suspicious trace of useragents and paths. By combining statistical techniques, machine learning algorithms, and domain expertise, the analysis contributed significantly to the overall enhancement of system monitoring and management.

6.1.1 Special Challenges

- The challenges in this dataset includes efficiently handling the high volume of log entries.
- Additionally, the dataset used in this project was not labeled, which posed difficulties for supervised analysis.

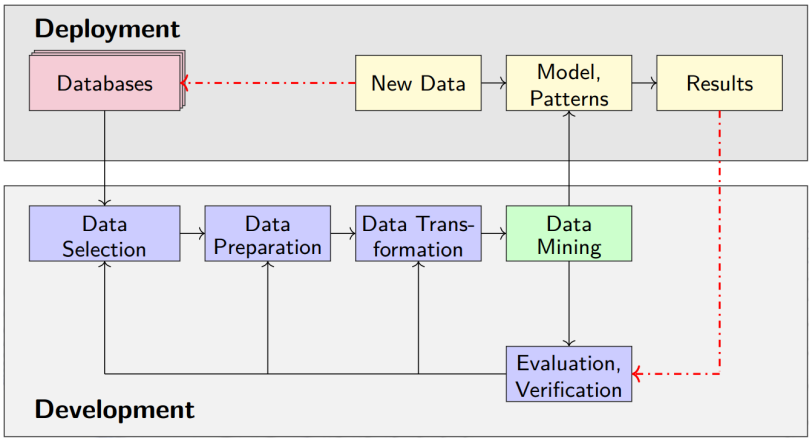


Figure 6.1: KDD Process
[Win24]

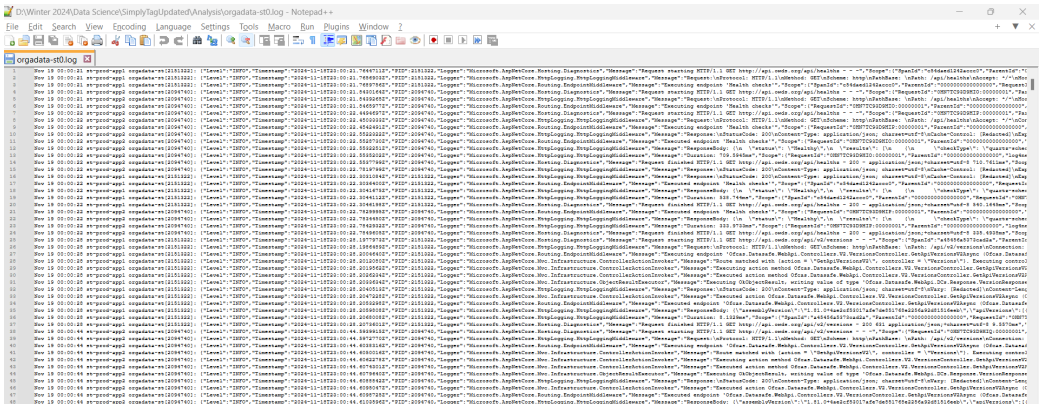


Figure 6.2: Database - Log file

- Another challenge was that log entries were not sequentially arranged by `TraceId`, as logs are ordered by timestamp due to concurrent requests. Grouping and organizing requests using `TraceId` was necessary for accurate analysis, adding complexity to preprocessing.

6.2 Database

The database used in this project comprised of log files collected from the system. The analysis was performed collectively on nine log files dated 20th November, 21st November, 23rd November, 24th November, 26th November, 27th November, 29th November, 30th November, and 1st December, 2024 as shown in the figure 6.2.

6.3 Data Selection

Data selection is a initial phase of the KDD process that directly influences the project's outcomes. It involves examining the input log files to identify relevant portions for analysis, ensuring the focus remains on meaningful and actionable insights.

6.3.1 Origin

Logs generated by monitoring tools and event management systems, specifically collected from the Graylog server.

6.3.2 Data Format

The file format is `.log`.

6.3.3 Features

The dataset includes log entries with the following attributes

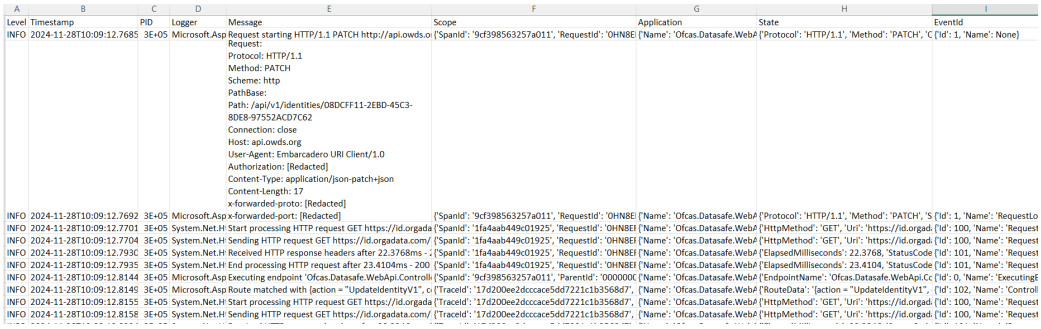
- Timestamps
- PID
- Logger
- Message
- Scope (e.g., TraceId, RequestID)
- Application
- State
- EventID

6.3.4 Size

The dataset is approximately 3.34 GB size.

6.4 Data Preparation

Data preparation in the Knowledge Discovery in Databases (KDD) process, ensures the dataset is clean, consistent and ready for analysis.



Level	Timestamp	PID	Logger	Message	Scope	Application	State	EventId
INFO	2024-11-28T10:09:12.7685	3E+05	Microsoft.Asp	Request starting HTTP/1.1 PATCH http://api.owds.o Request: Protocol: HTTP/1.1 Method: PATCH Scheme: http PathBase: Path: /api/v1/identities/08DCFF11-2EBD-45C3-8DE8-97552ACD7C62 Connection: close Host: api.owds.org User-Agent: Embarcadero URI Client/1.0 Authorization: [Redacted] Content-Type: application/json-patch+json Content-Length: 17 x-forwarded-proto: [Redacted]	{'SpanId': '9cf398563257a011', 'RequestId': '0HN8E	{'Name': 'Ofcas.Datasafe.WebA	{'Protocol': 'HTTP/1.1', 'Method': 'PATCH', 'C	{'Id': 1, 'Name': None}
INFO	2024-11-28T10:09:12.7692	3E+05	Microsoft.Asp	x-forwarded-port: [Redacted]	{'SpanId': '9cf398563257a011', 'RequestId': '0HN8E	{'Name': 'Ofcas.Datasafe.WebA	{'Protocol': 'HTTP/1.1', 'Method': 'PATCH', 'S	{'Id': 1, 'Name': 'RequestLo
INFO	2024-11-28T10:09:12.7701	3E+05	System.Net.H	Start processing HTTP request GET https://id.orgada	{'SpanId': '1fa4aab449c01925', 'RequestId': '0HN8EI	{'Name': 'Ofcas.Datasafe.WebA	{'HttpMethod': 'GET', 'Uri': 'https://id.orgad	{'Id': 100, 'Name': 'Request
INFO	2024-11-28T10:09:12.7704	3E+05	System.Net.H	Sending HTTP request GET https://id.orgadata.com/	{'SpanId': '1fa4aab449c01925', 'RequestId': '0HN8EI	{'Name': 'Ofcas.Datasafe.WebA	{'HttpMethod': 'GET', 'Uri': 'https://id.orgad	{'Id': 100, 'Name': 'Request
INFO	2024-11-28T10:09:12.7930	3E+05	System.Net.H	Received HTTP response headers after 22.3768ms - 2	{'SpanId': '1fa4aab449c01925', 'RequestId': '0HN8EI	{'Name': 'Ofcas.Datasafe.WebA	{'ElapsedMilliseconds': 22.3768, 'StatusCode	{'Id': 101, 'Name': 'Request
INFO	2024-11-28T10:09:12.7935	3E+05	System.Net.H	End processing HTTP request after 23.4104ms - 200	{'SpanId': '1fa4aab449c01925', 'RequestId': '0HN8EI	{'Name': 'Ofcas.Datasafe.WebA	{'ElapsedMilliseconds': 23.4104, 'StatusCode	{'Id': 101, 'Name': 'Request
INFO	2024-11-28T10:09:12.8144	3E+05	Microsoft.Asp	Executing endpoint 'Ofcas.Datasafe.WebApi.Controlle	{'SpanId': '9cf398563257a011', 'ParentId': '000000(	{'Name': 'Ofcas.Datasafe.WebA	{'EndpointName': 'Ofcas.Datasafe.WebApi.Cc	{'Id': 0, 'Name': 'ExecutingE
INFO	2024-11-28T10:09:12.8149	3E+05	Microsoft.Asp	Route matched with {action = "UpdateIdentityV1", c	{'TraceId': '17d200ee2dcccace5dd7221c1b3568d7',	{'Name': 'Ofcas.Datasafe.WebA	{'RouteData': '{action = "UpdateIdentityV1",	{'Id': 102, 'Name': 'Controll
INFO	2024-11-28T10:09:12.8155	3E+05	System.Net.H	Start processing HTTP request GET https://id.orgada	{'TraceId': '17d200ee2dcccace5dd7221c1b3568d7',	{'Name': 'Ofcas.Datasafe.WebA	{'HttpMethod': 'GET', 'Uri': 'https://id.orgad	{'Id': 100, 'Name': 'Request
INFO	2024-11-28T10:09:12.8158	3E+05	System.Net.H	Sending HTTP request GET https://id.orgadata.com/	{'TraceId': '17d200ee2dcccace5dd7221c1b3568d7',	{'Name': 'Ofcas.Datasafe.WebA	{'HttpMethod': 'GET', 'Uri': 'https://id.orgad	{'Id': 100, 'Name': 'Request

Figure 6.3: Converted CSV file

Level	Timestamp	PID	Logger	Message	Scope
ERROR	2024-11-28T18:16:5	668	Microsoft.EntityFramework	An error occurred using the connection to database 'datasafe' on server '192.168.244.135'.	{'ParentId': '0000000000000000', 'C

Figure 6.4: Error log

6.4.1 Merging and Conversion

- The initial step involved merging all nine individual log files into a single consolidated file.
- Then the merged file, originally in format `.log`, were converted to CSV file to facilitate easier manipulation as shown in the figure 6.3.

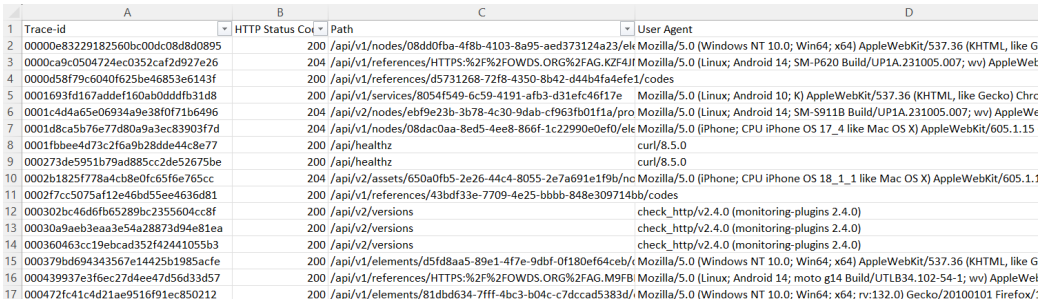
6.4.2 Initial Cleaning

- Excluding entries without a valid TraceId that contain generic error or warning messages as shown in figure 6.4.

6.4.3 Key Attributes

The following were the key parameters that were identified and prioritized for analysis,

- `TraceId`: Used to track individual requests across log entries, enabling the grouping and analysis of entire request flows.
- `HTTP Status Code`: Identifies the result of a request (e.g., success, client errors, or server errors). Codes such as 200, 404 and 500 provide critical insights into system health.
- `Path`: Represents the API endpoint or resource accessed during a request, useful for pinpointing affected components or services.
- `User-Agent`: Provides details about the client or system making the request, helping to identify trends in usage or unusual activity from specific sources.



	A	B	C	D
1	Trace-id	HTTP Status Co	Path	User Agent
2	00000e83229182560bc00dc08d8d0895	200	/api/v1/nodes/08dd0fba-4f8b-4103-8a95-aed373124a23/el	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like G
3	0000ca9c0504724ec0352caf2d927e26	204	/api/v1/references/HTTPS:%2F%2FOWDS.ORG%2FAG.KZF4JI	Mozilla/5.0 (Linux; Android 14; SM-P620 Build/UP1A.231005.007; wv) AppleWeb
4	0000d58f79c6040f625be46853e6143f	200	/api/v1/references/d5731268-72f8-4350-8b42-d44b4fa4efe1/codes	
5	0001693fd167addef160ab0dddfb31d8	200	/api/v1/services/8054f549-6c59-4191-afb3-d31efc46f17e	Mozilla/5.0 (Linux; Android 10; K) AppleWebKit/537.36 (KHTML, like Gecko) Chrc
6	0001c4d4a65e06934a9e38f0f71b6496	204	/api/v2/nodes/ebf9e23b-3b78-4c30-9dab-cf963fb01f1a/pro	Mozilla/5.0 (Linux; Android 14; SM-S911B Build/UP1A.231005.007; wv) AppleWe
7	0001d8ca5b76e77d80a9a3ec83903f7d	204	/api/v1/nodes/08dac0aa-8ed5-4ee8-866f-1c22990e0ef0/ele	Mozilla/5.0 (iPhone; CPU iPhone OS 17_4 like Mac OS X) AppleWebKit/605.1.15
8	0001fbbee4d73c2f6a9b28dde44c8e77	200	/api/healthz	curl/8.5.0
9	000273de5951b79ad885cc2de52675be	200	/api/healthz	curl/8.5.0
10	0002b1825f778a4cb8e0fc65f6e765cc	204	/api/v2/assets/650a0fb5-2e26-44c4-8055-2e7a691e1f9b/no	Mozilla/5.0 (iPhone; CPU iPhone OS 18_1_1 like Mac OS X) AppleWebKit/605.1.1
11	0002f7cc5075af12e46bd55ee4636d81	200	/api/v1/references/43bdf33e-7709-4e25-bbbb-848e309714bb/codes	
12	000302bc46d6fb65289bc2355604cc8f	200	/api/v2/versions	check_http/v2.4.0 (monitoring-plugins 2.4.0)
13	00030a9aeb3eaa3e54a28873d94e81ea	200	/api/v2/versions	check_http/v2.4.0 (monitoring-plugins 2.4.0)
14	000360463cc19ebcad352f42441055b3	200	/api/v2/versions	check_http/v2.4.0 (monitoring-plugins 2.4.0)
15	000379bd694343567e14425b1985acfe	200	/api/v1/elements/d5fd8aa5-89e1-4f7e-9dbf-0f180ef64ceb/	Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like G
16	000439937e3f6ec27d4ee47d56d33d57	200	/api/v1/references/HTTPS:%2F%2FOWDS.ORG%2FAG.M9FB	Mozilla/5.0 (Linux; Android 14; moto g14 Build/UTLB34.102-54-1; wv) AppleWel
17	000472fc41c4d21ae9516f91ec850212	200	/api/v1/elements/81dbd634-7fff-4bc3-b04c-c7dccad5383d/	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:132.0) Gecko/20100101 Firefox/

Figure 6.5: Result after applying data transformation

6.5 Data Transformation

This section focuses on manipulating and reshaping the cleaned data into a structured format ready for analysis. The key transformation includes,

1. Grouping log entries by `TraceId` to reconstruct complete request flows.
2. Parsing relevant fields (`TraceId, HTTP Status Code, Path, User-Agent`) from nested JSON structures.
3. Transforming each row to represent a unique log entry with all associated fields (e.g., aligning `TraceId` with its corresponding attributes) as shown in figure 6.5.

6.6 Data Mining

Data mining involves extracting hidden patterns and anomalies from datasets. In this project, multiple approaches were utilized, combining feature extraction, clustering and anormal detection for potential threat analysis.

6.6.1 Feature Extraction

Feature extraction involves converting raw data into a structured format suitable for machine learning.

- **Path-Based Features (Approach 1 & 2)**
  - Extracted features based on `Path` attribute such as Path length, Presence of special characters, SQL keywords, Path traversal attempts and Suspicious file extensions.
  - Applied TF-IDF vectorization on the `Path` attribute to represent API endpoint access as numerical features.
- **User-Agent Analysis (Approach 3)**

- Analyzed `User-Agent` strings to identify patterns associated with malicious or unusual behaviors.

6.6.2 Clustering and Anomaly Detection

Three complementary approaches were used to detect anomalies:

Approach 1

- This approach combined extracted path features (e.g., path length, special characters) with frequency metrics and numeric attributes like HTTP status codes.
- Anomalies were detected using DBSCAN for clustering and Isolation Forest for outlier detection.

Approach 2

- Used TF-IDF vectorized features and numeric attributes for clustering with DBSCAN.
- Combined DBSCAN with Isolation Forest for robust anomaly detection, flagging requests that deviated from expected patterns.

Approach 3

- Focused on analyzing User-Agent patterns to identify suspicious behaviors, integrating these insights with machine learning models for effective anomaly detection.

6.7 Model

The model selection was driven by the need to perform both clustering and anomaly detection effectively. A combination of unsupervised and ensemble-based learning techniques was employed to analyze request patterns and identify anomalies.

6.7.1 DBSCAN

DBSCAN is a density-based clustering algorithm, groups data points that are closely packed together based on a density threshold. Data points far from any dense cluster are treated as "noise" or anomalies.

Key Parameters

- **eps**: Set to 0.5, defining the maximum distance between points to form a cluster.
- **min_samples**: Set to 5, specifying the minimum number of points required to form a cluster.

Outcome

Requests not assigned to any cluster (cluster = -1) were flagged as anomalies.

6.7.2 Isolation Forest

Isolation Forest is an ensemble learning technique designed to detect outliers by isolating data points. It works on the principle that anomalies are easier to isolate because they have unique features.

Key Parameters

- **n_estimators:** Set to 100, specifying the number of decision trees.
- **contamination:** Set to 0.01, indicating the proportion of expected anomalies in the dataset.
- **random_state:** Ensured reproducibility of results with a fixed seed value (42).

Outcome

Requests identified as anomalies were flagged with a value of -1.

6.8 Validation/Verification

Validation and verification are critical steps to ensure the reliability and accuracy of the data mining and anomaly detection process. This section outlines the methods used to validate the models and verify the results.

6.8.1 DBSCAN Validation

The results of the DBSCAN clustering were validated as follows:

- The clustering results were reviewed to ensure that data points in the same cluster exhibited similar patterns.
- Anomalies (`Cluster = -1`) were manually inspected to confirm that they represented requests deviating from normal patterns.

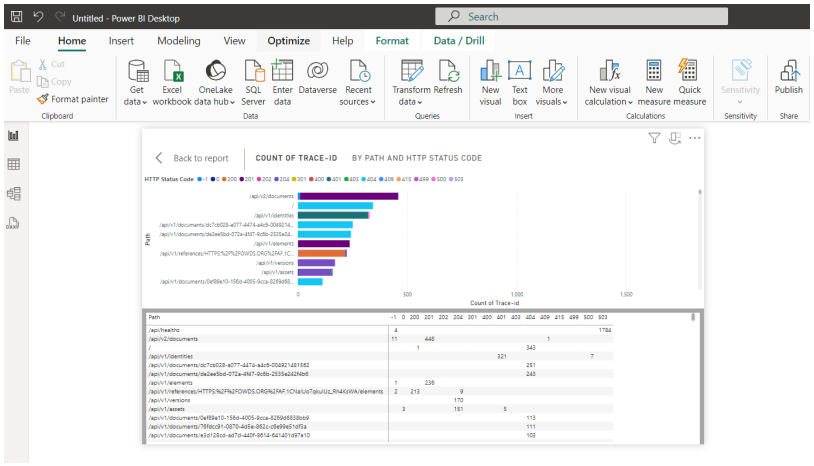


Figure 6.6: Bar chart illustrating the distribution of anomalies based on Path, TraceId, and HTTP Status Code

6.8.2 Isolation Forest Validation

The Isolation Forest model's performance was validated using the following methods:

- **Anomaly Score Distribution**: The distribution of anomaly scores was evaluated to differentiate anomalies from regular requests.
- **Manual Inspection**: A subset of flagged anomalies was reviewed to confirm their unusual characteristics.

6.9 Data Visualization

The final step in the KDD process is representing the acquired knowledge in an understandable form and using it to make decisions. To effectively demonstrate and view the output of our data mining techniques we used Power Bi tool where we created a dashboard for approach 1, approach 2 and approach 3 as shown in the figure 6.6, 6.7 and 6.8 respectively, for visualizing the Paths and the User-agents which we identified as suspicious threat. Power BI's dashboards and reports are excellent for this purpose, as they can effectively communicate insights to all levels of an organization.

6.10 Conclusion

This project successfully utilized data mining and anomaly detection techniques to uncover irregularities within system log files. By utilizing advanced machine learning algorithms such as DBSCAN and Isolation Forest, we were able to cluster request patterns and detect anomalies with precision, resulting in a reliable and efficient detection process. The implementation followed the Knowledge Discovery

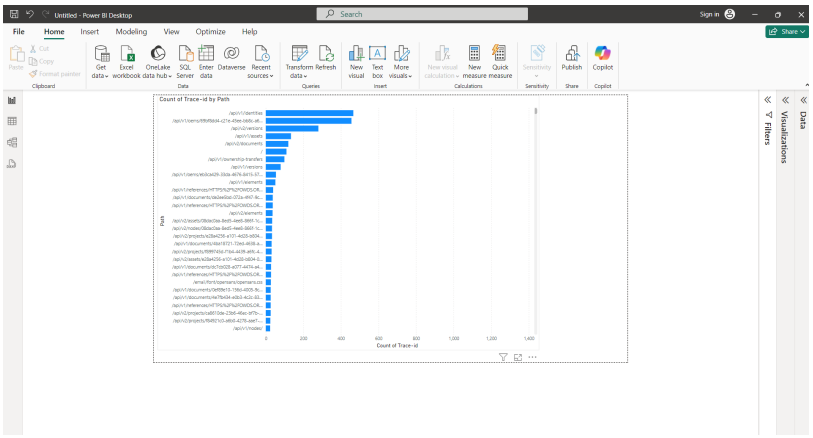


Figure 6.7: Bar chart visualizing anomalies identified using Path and TraceId, based on clustering efficiency and deviations from normal access patterns

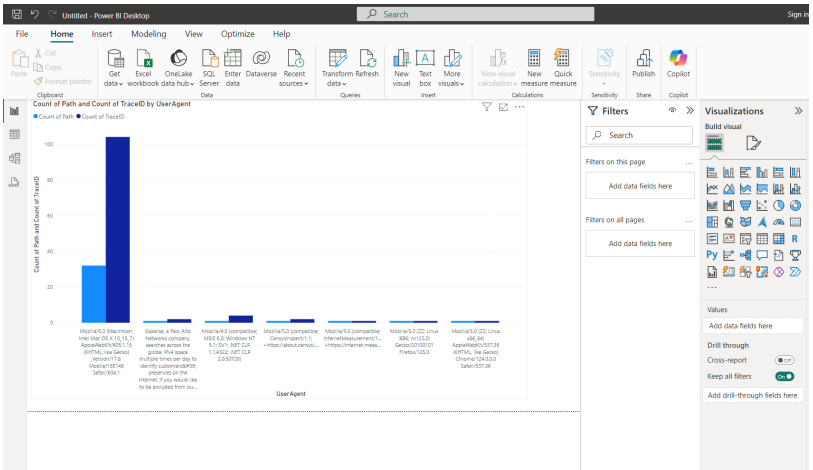


Figure 6.8: Analysis of anomalies detected based on User-Agent behaviors

in Databases (KDD) process, ensuring a systematic approach to data selection, preparation, transformation and analysis, which enhanced the accuracy and relevance of the findings.

Bibliography

[Bra20] A. Brandao, "Log files analysis for network intrusion detection," *IEEE Conference Publication*, 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9199976.

[FPS97] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "Data mining and knowledge discovery in databases: Implications for scientific databases," *9th International Conference on Scientific and Statistical Database Management*, 1997. DOI: 10.1109/SSDM.1997.621141. [Online]. Available: https://ieeexplore.ieee.org/document/621141.

[IEE22] IEEE, "Landscape of automated log analysis: A systematic literature review and mapping study," *IEEE Journals and Magazine*, 2022. [Online]. Available: https://ieeexplore.ieee.org/document/9716129.

[MFH00] A. Mockus, R. T. Fielding, and J. Herbsleb. "The kdd process for extracting useful knowledge from volumes of data." (2000), [Online]. Available: https://dl.acm.org/doi/10.1145/240455.240464.

[Win24] P. E. Wings, *Lecture in data science and analytics: Knowledge discovery in database (kdd) process*, Master in Industrial Informatics, Hochschule Emden/Leer University of Applied Science, 2024.