ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΑΝΑΛΥΣΗ ΚΑΙ ΟΠΤΙΚΟΠΟΙΗΣΗ ΔΕΔΟΜΕΝΩΝ ΚΑΤΑΓΡΑΦΗΣ (LOG DATA)»

Του φοιτητή
Κόκκινου Ευάγγελου
Αρ. Μητρώου: 144235

Επιβλέπων
Μπράτσας Χαράλαμπος
Επίκουρος Καθηγητής

Ημερομηνία 15-05-2025

Τίτλος Π.Ε. Ανάλυση και Οπτικοποίηση Δεδομένων Καταγραφής (Log Data)
Κωδικός Π.Ε. 24284
Ονοματεπώνυμο φοιτητή Κόκκινος Ευάγγελος
Ονοματεπώνυμο εισηγητή Μπράτσας Χαράλαμπος
Ημερομηνία ανάληψης Π.Ε. 29-01-2025
Ημερομηνία περάτωσης Π.Ε. 15-05-2025

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Η παρούσα εργασία εκπονήθηκε στο πλαίσιο της ολοκλήρωσης των σπουδών μου και αποτελεί το επιστέγασμα της ακαδημαϊκής πορείας μου στο αντικείμενο της Μηχανικής Λογισμικού. Το θέμα της ανάλυσης και οπτικοποίησης δεδομένων καταγραφής (log data) αποτέλεσε για μένα όχι μόνο ένα τεχνικά ενδιαφέρον πεδίο, αλλά και μια ευκαιρία να συνδυάσω τη θεωρητική κατάρτιση με πρακτικές δεξιότητες προγραμματισμού, σχεδιασμού διεπαφής και ανάλυσης δεδομένων, δημιουργώντας ένα real-world application.

Καθ' όλη τη διάρκεια της υλοποίησης, αντιμετωπίστηκαν τεχνικές και εννοιολογικές προκλήσεις που απαιτούσαν επίλυση με τρόπο δημιουργικό και τεκμηριωμένο. Η επιλογή των τεχνολογιών, η μελέτη των parsing αλγορίθμων, η σχεδίαση της αρχιτεκτονικής και η ανάπτυξη του dashboard αποτέλεσαν σταθμούς σε μια πορεία που ήταν όσο επίπονη, άλλο τόσο ικανοποιητική.

Περίληψη

Η εργασία αυτή ασχολείται με την ανάλυση και οπτικοποίηση αρχείων καταγραφής (logs), με σκοπό την υποστήριξη της διαγνωστικής, εποπτικής και αναλυτικής διαδικασίας σε πληροφοριακά συστήματα. Αρχικά παρουσιάζονται τα θεωρητικά θεμέλια του log parsing, οι τύποι δομών και οι βασικές κατηγορίες αλγορίθμων (heuristics, clustering, optimization, ML). Επιπλέον, εξετάζεται η λειτουργία των dashboards και των οπτικοποιήσεων στην εξαγωγή νοήματος από τεράστιες ποσότητες logs.

Στο πρακτικό μέρος αναπτύσσεται μια εφαρμογή σε Python και Streamlit, η οποία επιτρέπει στους χρήστες να φορτώνουν αρχεία log, να επιλέγουν έναν parser και να βλέπουν δομημένες εξόδους, στατιστικά στοιχεία και διαδραστικά γραφήματα. Υποστηρίζονται πολλαπλοί τύποι logs (Windows, Linux, Mac, Suricata), ενώ έχουν ενσωματωθεί parsers από το LogPai repository. Ακόμα, πραγματοποιείται benchmarking για τις παραμέτρους των parsers, εντοπίζονται κρίσιμα ζητήματα αξιοπιστίας και καταγράφονται αρχιτεκτονικές και εργονομικές βελτιώσεις που υλοποιήθηκαν στο σύστημα.

Η εργασία καταλήγει με συμπεράσματα που αφορούν τη σημασία της προσεκτικής παραμετροποίησης, την επιλογή κατάλληλων visual encodings και μερικές προτάσεις για επόμενα βήματα και επεκτάσεις της εφαρμογής.

Log Data Analysis and Visualization

KOKKINOS EVANGELOS

Abstract

This thesis explores the problem of log data analysis and visualization with the goal of supporting diagnostic, monitoring and analytical processes in modern computing environments. It begins by presenting the theoretical background of log parsing, including log structures, parsing techniques, and algorithmic categories such as heuristics, clustering, optimization, and machine learning.

The practical part consists of the development of a Python and Streamlit application that allows users to upload log files, select a parser, and interact with structured data, statistics and dynamic visualizations. The system supports multiple log types (Windows, Linux, Mac, Suricata) and integrates parsers from the LogPai repository. Benchmarking experiments were conducted to fine-tune parsing parameters and identify key stability and accuracy issues. Architectural refinements and usability improvements were also implemented to ensure modularity and extensibility.

The findings emphasize the necessity of reusable, flexible tools for large-scale log analysis, the significance of careful parser configuration, and the function of efficient visual encodings in log interpretation.

Περιεχόμενα

Κατάλογος Σχημάτων

Κεφάλαιο 1ο: Εισαγωγή

Η ραγδαία εξέλιξη των πληροφοριακών συστημάτων σε συνδυασμό με την αυξανόμενη πολυπλοκότητα των υποδομών πληροφορικής έχουν δημιουργήσει νέες προκλήσεις στη διαχείριση, την εποπτεία αλλά και την ανάλυση της λειτουργίας τους. Σε κάθε πληροφοριακό ή υπολογιστικό σύστημα, παράγονται συνεχώς αρχεία καταγραφής (logs) που περιέχουν πληροφορίες για τη λειτουργία του συστήματος, σαν ένα ημερολόγιο με στοιχεία που φανερώνουν τον χρόνο, το περιεχόμενο και την προέλευση ενός συμβάντος, μιας διαδικασίας ή αστοχίας του συστήματος. Τα logs αποτελούν έναν πολύτιμο θησαυρό δεδομένων, ο οποίος, αν αξιοποιηθεί σωστά, μπορεί να βοηθήσει προσφέροντας πολύτιμες γνώσεις για την απόδοση, την ασφάλεια και τη συντήρηση του συστήματος [1].

Η ανάλυση των log data (log analysis) είναι μια κρίσιμη διαδικασία που βοηθάει στον εντοπισμό και την διάγνωση προβλημάτων, την αναγνώριση ανωμαλιών, την παρακολούθηση της απόδοσης και της ασφάλειας, μέχρι τη διασφάλιση της αξιοπιστίας των συστημάτων [2]. Ωστόσο, η φύση των αρχείων καταγραφής, που συνήθως είναι ημι-δομημένα ή αδόμητα, καθιστά δύσκολη την άμεση επεξεργασία τους. Για να μπορέσει κανείς να εφαρμόσει τεχνικές ανάλυσης με αυτοματοποιημένα εργαλεία, απαιτείται πρώτα η μετατροπή των αδόμητων αυτών δεδομένων σε δομημένη μορφή. Αυτή η διαδικασία ονομάζεται log parsing [1].

Η παρούσα εργασία επικεντρώνεται στη μελέτη, την ανάλυση και την οπτικοποίηση δεδομένων καταγραφής, αξιοποιώντας σύγχρονες τεχνικές log parsing και τεχνολογίες παρουσίασης δεδομένων. Στόχος είναι η ανάπτυξη ενός ολοκληρωμένου συστήματος που θα επιτρέπει την αυτόματη επεξεργασία αρχείων log, την εξαγωγή σημαντικών πληροφοριών και την παρουσίαση τους σε φιλικό προς τον χρήστη περιβάλλον, ώστε να μπορεί και να έχει νόημα να χρησιμοποιηθεί στην πράξη, να είναι δηλαδή χρήσιμο και χρησιμοποιήσιμο.

1.1 Σκοπός και στόχοι της εργασίας

Ο κύριος σκοπός της παρούσας εργασίας είναι η ανάπτυξη ενός συστήματος ανάλυσης και οπτικοποίησης log δεδομένων, το οποίο θα διευκολύνει τη διαχείριση και την εποπτεία πληροφοριακών συστημάτων. Ειδικότερα, οι επιμέρους στόχοι περιλαμβάνουν:

- Την εκπόνηση βιβλιογραφικής επισκόπησης για τις σύγχρονες μεθόδους log parsing και τις τεχνολογίες ανάλυσης logs.
- Την ανάπτυξη ενός λογισμικού σε Python, το οποίο θα επιτρέπει στον χρήστη να ανεβάζει αρχεία logs, να επιλέγει τον τύπο τους και τον κατάλληλο αλγόριθμο parsing (π.χ. Drain, Spell, IPLoM).
- Την εξαγωγή δομημένων δεδομένων με πεδία όπως timestamp, log level, source, message.
- Την υλοποίηση ενός διαδραστικού interface με χρήση της πλατφόρμας Streamlit για την προβολή των αποτελεσμάτων της διαδικασίας του parsing, την παρουσίαση στατιστικών αναλύσεων και την οπτικοποίηση δεδομένων.
- Την ενσωμάτωση φίλτρων και αναζήτησης τόσο στους πίνακες των δομημένων πλέον logs όσο και στα διαγράμματα.
- Τη δημιουργία πίνακα parsed logs και πίνακα templates που προκύπτουν από το parsing.
- Την τεκμηρίωση της εφαρμογής με σενάρια χρήσης και αξιολόγηση της λειτουργικότητάς της.

Με αυτόν τον τρόπο, η εργασία επιδιώκει να συνδυάσει τη θεωρητική γνώση με την πρακτική εφαρμογή.

1.2 Η σημασία των logs στη διαχείριση συστημάτων

Τα logs αποτελούν το «μαύρο κουτί» ενός πληροφοριακού συστήματος, μέσα στο οποίο αποτυπώνονται όλες οι ενέργειες, οι αλληλεπιδράσεις και τα σφάλματα που συμβαίνουν κατά τη λειτουργία του. Η αξιοποίησή τους είναι απαραίτητη για πληθώρα εφαρμογών, όπως:

- Ανίχνευση ανωμαλιών (anomaly detection): Μέσω της ανάλυσης των logs, οι διαχειριστές μπορούν να εντοπίσουν ύποπτες ή ασυνήθιστες δραστηριότητες που ενδεχομένως να υποδηλώνουν προβλήματα ασφάλειας ή λειτουργικά σφάλματα [2].
- Διάγνωση προβλημάτων (root cause analysis): Τα logs παρέχουν κρίσιμες ενδείξεις για την αιτία μιας δυσλειτουργίας ή απροσδόκητα χαμηλής απόδοσης, μειώνοντας έτσι τον χρόνο αποκατάστασης και αυξάνοντας την διαθεσιμότητα του συστήματος [3].
- Ασφάλεια συστημάτων (security auditing): Τα αρχεία καταγραφής είναι θεμέλιο για την παρακολούθηση ύποπτων ενεργειών, τον εντοπισμό επιθέσεων και τη συμμόρφωση με πρότυπα κανονισμούς.
- Παρακολούθηση απόδοσης (performance monitoring): Μέσα από τα logs μπορεί να αναλυθεί η συμπεριφορά ενός συστήματος σε πραγματικές συνθήκες λειτουργίας, δίνοντας χώρο για αξιολόγηση και βελτιστοποίηση.

Καθώς τα συστήματα γίνονται ολοένα και πιο πολύπλοκα, με περισσότερες πηγές και τύπους μορφοποίησης των logs και ο όγκος των δεδομένων που παράγονται καθημερινά γιγαντώνετε, υπάρχει ακόμα πιο επιτακτική ανάγκη για αυτόματη επεξεργασία και ανάλυση των logs, καθιστώντας το log parsing βασικό βήμα πριν από οποιαδήποτε άλλη ανάλυση [1].

1.3 Δομή της εργασίας

Η εργασία χωρίζεται σε δύο βασικά μέρη: το θεωρητικό και το πρακτικό.

Στο θεωρητικό μέρος (Κεφάλαια 2-5), γίνεται εισαγωγή στις βασικές έννοιες των logs και του parsing, παρουσιάζονται οι βασικότεροι αλγόριθμοι που έχουν προταθεί στη βιβλιογραφία, και αναλύεται που χρησιμοποιούνται στην πράξη.

Στο πρακτικό μέρος (Κεφάλαιο 6), περιγράφεται η υλοποίηση της εφαρμογής που αναπτύχθηκε, από την αρχιτεκτονική του συστήματος και τον τρόπο που δουλεύει το parsing, μέχρι την διεπαφή με τον χρήστη, τα dashboards και τα τελικά αποτελέσματα.

Κεφάλαιο 2ο: Εισαγωγή στα Log Data

Η ανάλυση των log δεδομένων αποτελεί αναπόσπαστο κομμάτι της διαχείρισης σύγχρονων πληροφοριακών συστημάτων. Καθώς τα υπολογιστικά και επικοινωνιακά συστήματα γίνονται όλο και πιο πολύπλοκα, η ανάγκη για πλήρη και ακριβή παρακολούθηση της λειτουργίας τους γίνεται επιτακτική. Τα log αρχεία παρέχουν ένα λεπτομερές ιστορικό της συμπεριφοράς του συστήματος, καταγράφοντας γεγονότα, ενέργειες, σφάλματα και άλλες κρίσιμες πληροφορίες που παράγονται σε πραγματικό χρόνο κατά τη διάρκεια της εκτέλεσης.

2.1 Τι είναι τα Log Data

Τα log data είναι αρχεία κειμένου τα οποία περιλαμβάνουν μία σειρά από εγγραφές καταγραφής (log entries), κάθε μία από τις οποίες αντιστοιχεί σε ένα συγκεκριμένο γεγονός ή συμβάν στο σύστημα. Τα αρχεία αυτά χρησιμοποιούνται σε ένα ευρύ φάσμα εφαρμογών, από μεγάλες υπολογιστικές συστοιχίες και data centers έως απλές εφαρμογές σε τερματικές συσκευές χρηστών [1]. Ουσιαστικά, αποτελούν το «ημερολόγιο» του συστήματος, καταγράφοντας την ακολουθία γεγονότων με τρόπο που επιτρέπει την εκ των υστέρων ανάλυση, επιθεώρηση και αξιολόγηση της λειτουργίας.

Κάθε καταχώρηση καταγραφής είναι μια εγγραφή που περιλαμβάνει λεπτομέρειες σχετικά με το συμβάν που καταγράφει, συμπεριλαμβανομένης της ώρας που συνέβη, της πηγής ή της υπηρεσίας που το δημιούργησε, του επιπέδου σοβαρότητας και μιας περιγραφής του περιστατικού. Η πλειονότητα των logs τηρεί αυτή τη γενική δομή, παρά την έλλειψη ενός ευρέως αναγνωρισμένου προτύπου μορφοποίησης [1].

2.2 Δομή Log Μηνυμάτων (Timestamp, Log Level, Source, Message)

Η βασική δομή ενός log message περιλαμβάνει τέσσερα κύρια πεδία: τη χρονική σήμανση (timestamp), το επίπεδο σοβαρότητας (log level), την πηγή ή το υποσύστημα (source/component) και το ίδιο το μήνυμα (message). Κάθε ένα από αυτά τα πεδία επιτελεί έναν ξεχωριστό ρόλο στην πληροφόρηση που παρέχει το log.

Το πεδίο timestamp καταγράφει την ακριβή χρονική στιγμή κατά την οποία συνέβη το γεγονός. Η πληροφορία αυτή είναι απαραίτητη για τη χρονολογική ταξινόμηση των logs και τη μελέτη της αλληλουχίας των γεγονότων μέσα στο σύστημα [2].

Το log level προσδιορίζει τη σοβαρότητα ή την προτεραιότητα του γεγονότος. Συνήθως χρησιμοποιούνται επίπεδα όπως INFO, WARNING, ERROR ή DEBUG, που επιτρέπουν τη φιλτράρισή τους ανάλογα με τη σημασία τους για τον διαχειριστή ή τον αναλυτή [2].

Η πηγή (source) αναφέρεται στο υποσύστημα ή την υπηρεσία που παρήγαγε το log. Αυτό το πεδίο βοηθά στην εντοπισμό του σημείου στο σύστημα από όπου προήλθε το γεγονός, διευκολύνοντας τη διάγνωση και την ανάλυση [1].

Τέλος, το message αποτελεί το περιγραφικό μέρος της καταγραφής, όπου παρέχονται λεπτομέρειες για το γεγονός σε μορφή ελεύθερου κειμένου. Το πεδίο αυτό συχνά περιέχει τόσο σταθερά στοιχεία όσο και δυναμικές τιμές που διαφέρουν από καταγραφή σε καταγραφή, όπως IDs, IP διευθύνσεις ή άλλα δεδομένα που σχετίζονται με το συμβάν [4].

Η παρακάτω εικόνα παρουσιάζει ένα παράδειγμα τυπικής δομής log entry:

TIMESTAMP	2015-10-18 18:05:29,570
LEVEL	INFO
COMPONENT	dfs.DataNode$PacketResponder
EVENT TEMPLATE	Received block <*> of size <*> from /<*>
PARAMETERS	["blk_-562725280853087685", "67108864", "10.251.91.84"]

Σχήμα 2.1: Παράδειγμα δομής log [5]

Δεν υπάρχει ένα καθολικό πρότυπο που να καθορίζει τη σύνθεση των πεδίων, παρά το γεγονός ότι υπάρχουν κοινές πρακτικές για τη μορφοποίηση των logs. Όπως αναφέρθηκε, οι προγραμματιστές συχνά επιλέγουν πώς θα ορίσουν τη μορφή του μηνύματος καταγραφής, γεγονός που καθιστά την ανάλυση πιο απρόβλεπτη και πολύπλοκη [1].

Επιπλέον, το μήνυμα μπορεί να διακριθεί σε δύο μέρη: το σταθερό μέρος (event template), που παραμένει ίδιο για όλες τις καταγραφές του ίδιου τύπου, και το μεταβλητό μέρος (parameters), που περιέχει τις δυναμικές πληροφορίες κάθε συγκεκριμένου συμβάντος [4]. Η αναγνώριση αυτής της διάκρισης είναι θεμελιώδης για την περαιτέρω ανάλυση, καθώς επιτρέπει την τυποποίηση των logs μέσω της δημιουργίας προτύπων (templates) πάνω στα οποία βασίζεται η διαδικασία parsing.

2.3 Η σημασία των Log Data στη διαχείριση συστημάτων

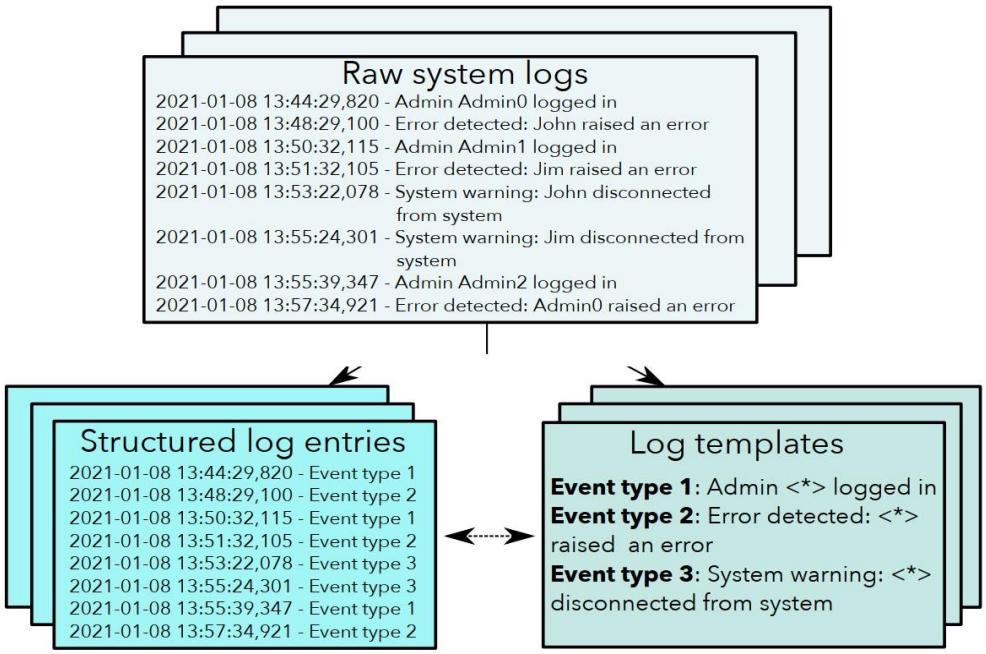
Ένα από τα πιο σημαντικά εργαλεία για την επίβλεψη και διαχείριση σύγχρονων πληροφοριακών συστημάτων είναι τα αρχεία καταγραφής. Οι διαχειριστές μπορούν να εντοπίζουν ανωμαλίες, να παρακολουθούν την απόδοση του συστήματος, να ανιχνεύουν σφάλματα και να ελέγχουν την επιχειρησιακή ασφάλεια μέσω της ανάλυσής τους. Τα δεδομένα καταγραφής λειτουργούν ως ένα λεπτομερές "ημερολόγιο" της δραστηριότητας του συστήματος, καταγράφοντας την τοποθεσία, την ώρα και τις συνθήκες σημαντικών συμβάντων [2].

Οι δηλώσεις καταγραφής (logging statements), οι οποίες καθορίζουν ποια μηνύματα θα καταγραφούν κατά την εκτέλεση, ενσωματώνονται στον πηγαίο κώδικα της εφαρμογής για τη δημιουργία logs. Με τη χρήση αυτής της τεχνικής, οι προγραμματιστές μπορούν να παρακολουθούν τη συμπεριφορά του συστήματος χωρίς να παρεμβαίνουν στη λειτουργικότητά του. Εξόρυξη διεργασιών (process mining), ανίχνευση ανωμαλιών (anomaly detection), αναγνώριση βασικής αιτίας (root cause identification), ανίχνευση εισβολών (intrusion detection), ανάλυση απόδοσης (performance analysis) και επαλήθευση εκτέλεσης (execution verification) είναι μόνο μερικές από τις πολλές χρήσεις των χρήσιμων δεδομένων που συλλέγουν τα logs [4].

Ωστόσο, υπάρχουν πολλές δυσκολίες στην ανάλυση αρχείων καταγραφής. Λόγω του τεράστιου όγκου δεδομένων που παράγεται καθημερινά, οι ουσιαστικές γνώσεις δεν μπορούν να εξαχθούν μέσω χειροκίνητης επιθεώρησης. Επιπλέον, πολλά μηνύματα καταγραφής είναι μη δομημένα, γεγονός που καθιστά δύσκολη την επεξεργασία τους με απλούς επεξεργαστές κειμένου ή συμβατικά εργαλεία αναζήτησης [6].

Για να μετατραπούν τα μη δομημένα δεδομένα καταγραφής σε δομημένες πληροφορίες που είναι έτοιμες για ανάλυση, αναπτύσσονται εξελιγμένες τεχνικές αυτόματης επεξεργασίας. Το Σχήμα 2.2

παρέχει ένα ενδεικτικό παράδειγμα αυτής της μετατροπής, μετατρέποντας ακατέργαστα logs σε πρότυπα και δομημένες καταχωρήσεις καταγραφής.



Σχήμα 2.2: Παράδειγμα μετασχηματισμού αδόμητων log σε δομημένες εγγραφές και πρότυπα γεγονότων [1]

2.4 Επίλογος

Η ανάλυση των log δεδομένων αποτελεί ακρογωνιαίο λίθο για τη διαχείριση, την εποπτεία και τη βελτίωση των πληροφοριακών συστημάτων. Μέσα από την κατάλληλη επεξεργασία τους, οι οργανισμοί αποκτούν πρόσβαση σε πολύτιμες πληροφορίες που επιτρέπουν τον εντοπισμό ανωμαλιών, τη διάγνωση προβλημάτων, τον έλεγχο της απόδοσης και την παρακολούθηση της ασφάλειας. Η διαδικασία μετατροπής των raw logs σε structured logs, μέσω της αναγνώρισης σταθερών και μεταβλητών στοιχείων και της δημιουργίας templates, είναι θεμελιώδης για την περαιτέρω αξιοποίηση των δεδομένων [1].

Η δυνατότητα να συνοψιστούν εκατομμύρια καταγραφές σε λίγα, αναγνωρίσιμα πρότυπα, μειώνει δραστικά την πολυπλοκότητα και επιτρέπει την εφαρμογή πιο προηγμένων τεχνικών ανάλυσης. Ωστόσο, η ποικιλία των μορφών και ο μεγάλος όγκος των logs καθιστούν απαραίτητη την ανάπτυξη αυτόματων, αξιόπιστων μεθόδων επεξεργασίας, ικανών να ανταποκρίνονται στις αυξανόμενες απαιτήσεις των συστημάτων.

Στο επόμενο κεφάλαιο, θα εξετάσουμε διεξοδικά τις τεχνικές και τις μεθόδους parsing, δηλαδή τη διαδικασία εκείνη που επιτρέπει τη μετατροπή των αδόμητων log δεδομένων σε δομημένα γεγονότα, προετοιμάζοντάς τα για περαιτέρω ανάλυση.

Κεφάλαιο 3ο: Log Parsing

Η ανάλυση δεδομένων καταγραφής είναι ένα ουσιαστικό συστατικό της διαχείρισης και ενίσχυσης της λειτουργικότητας των πληροφοριακών συστημάτων. Ωστόσο, επειδή τα logs των συστημάτων είναι συνήθως σε μορφή ελεύθερου κειμένου, η αυτόματη επεξεργασία τους είναι πρόκληση. Για να καταστούν χρήσιμα σε εφαρμογές εξαγωγής γνώσης και ανάλυσης, θα πρέπει πρώτα να μετατραπούν σε δομημένη μορφή. Αυτή η διαδικασία ονομάζεται log parsing και είναι το πρώτο και απαραίτητο βήμα για την περαιτέρω ανάλυση [1], [7].

3.1 Η διαδικασία Log Parsing

Η διαδικασία του log parsing αποτελεί το θεμέλιο για την ανάλυση των δεδομένων καταγραφής, καθώς μετατρέπει τα αδόμητα logs σε δομημένες εγγραφές που μπορούν να αξιοποιηθούν για εξόρυξη γνώσης και άλλες αναλυτικές εργασίες. Η μετατροπή αυτή επιτυγχάνεται με μια σειρά βημάτων, τα οποία λειτουργούν συνεργατικά για να απομονώσουν τα ουσιώδη μοτίβα από την ποικιλία και την πολυπλοκότητα των καταγραφών.

Συγκεκριμένα, το parsing περιλαμβάνει τέσσερα κύρια βήματα [1]. Πρώτον, το preprocessing, όπου αφαιρούνται άσχετες πληροφορίες, όπως IP διευθύνσεις ή μοναδικά αναγνωριστικά IDs, που μπορεί να εμποδίσουν την ομαδοποίηση των logs. Ακολουθεί η ταξινόμηση (data classification), κατά την οποία οι εγγραφές ομαδοποιούνται σε clusters με βάση ομοιότητες στο περιεχόμενό τους. Στη συνέχεια, πραγματοποιείται η εξαγωγή προτύπων (template extraction), δηλαδή η αναγνώριση των σταθερών και των μεταβλητών στοιχείων κάθε ομάδας. Η διαδικασία ολοκληρώνεται με το post-processing, όπου μπορεί να γίνει συγχώνευση ή διόρθωση clusters, με tuning παραμέτρων, για τη βελτιστοποίηση της ακρίβειας.

Ένα βασικό χαρακτηριστικό του log parsing είναι η ικανότητά του να δημιουργεί templates – πρότυπα που συνοψίζουν τη μορφή των γεγονότων καταγραφής. Τα templates αυτά απομονώνουν το σταθερό μέρος των μηνυμάτων από τα δυναμικά τους στοιχεία, επιτρέποντας την κατηγοριοποίηση χιλιάδων μοναδικών καταγραφών σε λίγες δεκάδες ή εκατοντάδες πρότυπα. Με τον τρόπο αυτό, μειώνεται σημαντικά η πολυπλοκότητα της ανάλυσης και διευκολύνεται η εφαρμογή μεθόδων εξόρυξης γνώσης [2].

Στην πράξη, κάθε log message αναλύεται ως προς τη δομή του για να εντοπιστούν ποια μέρη του παραμένουν σταθερά και ποια διαφέρουν σε κάθε εμφάνιση. Για παράδειγμα, ένα μήνυμα όπως:

2008-11-09 20:35:32,146 INFO dfs.DataNode$DataXceiver: Receiving block blk_-1608999687919862906 src: /10.251.31.5:42506 dest: /10.251.31.5:50010

θα άλλαζε για να γίνει το πρότυπο:

*Receiving block * src: * dest: **

όπου οι μεταβλητές (block ID, IP διευθύνσεις, ports) αντικαθίστανται με wildcards (αστερίσκους) [7]. Τα wildcards αποτελούν συμβολικές αναπαραστάσεις που δηλώνουν ότι σε εκείνη τη θέση μπορεί να εμφανιστεί οποιαδήποτε τιμή. Χρησιμοποιούνται για να γενικεύσουν το μήνυμα, ώστε να συμπεριλάβει όλες τις παραλλαγές του ίδιου γεγονότος.

Μαζί με τα wildcards, συχνά αξιοποιούνται και οι κανονικές εκφράσεις (regular expressions ή αλλιώς regex), ιδιαίτερα σε χειροκίνητες ή παραδοσιακές μεθόδους parsing. Οι regular expressions επιτρέπουν τον ορισμό κανόνων για την ανίχνευση ή εξαγωγή patterns μέσα στα logs, όπως IP addresses, timestamps ή IDs. Παρόλο που αποτελούν ισχυρό εργαλείο, οι regex απαιτούν σημαντική προσπάθεια για να γραφούν και να συντηρηθούν, ειδικά σε περιβάλλοντα όπου η μορφή των logs μεταβάλλεται συχνά. Η μετάβαση σε αυτοματοποιημένες, καθοδηγούμενες από δεδομένα τεχνικές parsing ενισχύεται από την παρατήρηση της βιβλιογραφίας ότι τα συστήματα που αλλάζουν γρήγορα καθιστούν τους κανόνες regex ξεπερασμένους [7], [1].

Τελικά, το parsed output αποτελείται από δύο κύρια αρχεία: ένα που περιέχει τα πρότυπα (τους αναγνωρισμένους τύπους συμβάντων) και ένα άλλο που περιέχει τις δομημένες εγγραφές, στις οποίες κάθε ακατέργαστο log αντιστοιχίζεται στο σχετικό πρότυπο. Αυτά τα αρχεία χρησιμοποιούνται ως είσοδος για μεταγενέστερες φάσεις ανάλυσης, συμπεριλαμβανομένης της εξαγωγής κανόνων, της ανίχνευσης ανωμαλιών και της διάγνωσης προβλημάτων [7].

Η διαδικασία του log parsing, είναι επομένως κάτι περισσότερο από ένα βήμα καθαρισμού δεδομένων. Είναι ένα κρίσιμο στάδιο που καθορίζει την ποιότητα και την αξιοπιστία κάθε περαιτέρω αναλυτικής προσπάθειας.

3.2 Σημασία του Log Parsing για την Ανάλυση

Η μετατροπή των logs σε δομημένη μορφή είναι απαραίτητη γιατί επιτρέπει την εφαρμογή μεθόδων εξόρυξης γνώσης, όπως ανίχνευση ανωμαλιών, εντοπισμός αιτιών σφαλμάτων, και ανάλυση απόδοσης. Χωρίς parsing, τα logs παραμένουν ως αδόμητο κείμενο, δύσκολο στη μαζική επεξεργασία. Επιπλέον, ακόμη και μικρά λάθη στο parsing μπορούν να έχουν μεγάλη επίδραση. Για παράδειγμα, ένα ποσοστό σφάλματος 4% στο parsing κρίσιμων συμβάντων θα μπορούσε να οδηγήσει σε μείωση της ακρίβειας ανάλυσης κατά μια τάξη μεγέθους [7].

Σε downstream εφαρμογές όπως η ανίχνευση ανωμαλιών, όπου τα εσφαλμένα ταξινομημένα συμβάντα μπορεί να οδηγήσουν σε λάθος ειδοποιήσεις (false alarms) ή στην παράλειψη πραγματικών προβλημάτων, η διαδικασία parsing έχει αντίκτυπο τόσο στην ακρίβεια όσο και στην αποτελεσματικότητα [7].

3.3 Αλγόριθμοι Log Parsing

Η βιβλιογραφία προτείνει τέσσερις κύριες κατηγορίες αλγορίθμων για log parsing: heuristic-based, clustering-based, frequent pattern mining-based και machine learning-based μεθόδους [1].

3.3.1 Μηχανισμοί βασισμένοι σε Heuristics

Οι heuristic μέθοδοι χρησιμοποιούν κανόνες βασισμένους σε χαρακτηριστικά των logs. Ένα χαρακτηριστικό παράδειγμα είναι ο IPLoM, ο οποίος χρησιμοποιεί μια διαδικασία κατάτμησης τριών βημάτων. Πρώτα με βάση τον αριθμό tokens, μετά με βάση τη θέση token με ελάχιστη παραλλαγή και έπειτα με χαρτογράφηση σχέσεων μεταξύ θέσεων token. Στο τέλος, εξάγει templates από κάθε cluster [7].

Παράλληλα, ο Drain χρησιμοποιεί ένα δέντρο σταθερού βάθους για να ταξινομήσει τα logs σε nodes που αντιστοιχούν σε κοινά prefixes, επιτρέποντας αποτελεσματική ομαδοποίηση και template extraction [6].

Οι heuristic μέθοδοι είναι συνήθως ταχύτερες και πιο σταθερές, αλλά ενδέχεται να μην προσαρμόζονται καλά σε πολύ ετερογενή δεδομένα.

3.3.2 Μέθοδοι βασισμένες σε Clustering

Οι clustering μέθοδοι αντιμετωπίζουν το parsing ως πρόβλημα ομαδοποίησης, βασισμένο σε μέτρα ομοιότητας. Ο LKE εφαρμόζει ιεραρχικό clustering με custom edit distance, ενώ ο LogSig μετατρέπει κάθε log σε σύνολο ζευγών λέξεων (word pairs) για να κωδικοποιήσει πληροφορία λέξης και θέσης, πριν την εφαρμογή clustering. Μετά από αυτό, κάθε cluster μετατρέπεται σε template [7].

Αν και αυτές οι προσεγγίσεις είναι προσαρμόσιμες, έχουν προβλήματα επεκτασιμότητας: Η εφαρμογή του LKE σε μεγάλα σύνολα δεδομένων είναι δύσκολη λόγω της πολυπλοκότητάς του O(n2), και απαιτεί πολύ χρόνο για την προσαρμογή παραμέτρων (όπως ο αριθμός των συστάδων) [7].

3.3.3 Μέθοδοι Frequent Pattern Mining

Η προσέγγιση frequent pattern mining, όπως στον SLCT, στοχεύει στον εντοπισμό λέξεων ή patterns που εμφανίζονται συχνά στα logs, και χρησιμοποιεί αυτά τα patterns για να δημιουργήσει templates. Ο SLCT λειτουργεί σε τρία βήματα: δημιουργία λεξιλογίου, κατασκευή cluster candidates, και εξαγωγή τελικών templates [7].

Παρά το ότι είναι γρήγορη και εύκολη, εάν τα όρια δεν επιλεγούν προσεκτικά, αυτή η μέθοδος μπορεί να παραβλέψει ασυνήθιστα συμβάντα ή να ομαδοποιήσει υπερβολικά τα logs.

3.3.4 Μέθοδοι Machine Learning / Deep Learning

Οι σύγχρονες ML μέθοδοι, όπως ο NuLog, χρησιμοποιούν transformer architectures και self-supervised learning για να μάθουν να προβλέπουν ποια tokens είναι μεταβλητά και ποια είναι στατικά. Έτσι, καταφέρνουν να αυτοματοποιούν πλήρως τη διαδικασία template extraction χωρίς χειροκίνητους κανόνες [8].

Άλλη μέθοδος είναι ο MoLFI, που μοντελοποιεί το parsing ως πρόβλημα πολυαντικειμενικής βελτιστοποίησης, λύνοντάς το με evolutionary algorithms για να βρει ένα σύνολο Pareto-optimal λύσεων [4].

Παρά την υψηλή τους ακρίβεια, οι τεχνικές μηχανικής μάθησης απαιτούν μεγάλα datasets εκπαίδευσης και σημαντική υπολογιστική ισχύ.

3.4 Αξιολόγηση Αλγορίθμων Parsing: Ακρίβεια, Αποδοτικότητα, Ευελιξία

Η αξιολόγηση των αλγορίθμων parsing γίνεται ως προς τρεις άξονες: ακρίβεια, δηλαδή το ποσοστό σωστών templates, αποδοτικότητα, δηλαδή ο χρόνος εκτέλεσης, και ευελιξία, δηλαδή η σταθερότητα της απόδοσης σε διαφορετικά datasets [2], [7].

Σύμφωνα πάντα με τη βιβλιογραφία, οι προσεγγίσεις που βασίζονται σε συσταδοποίηση (clustering-based) δεν έχουν συνεπή ακρίβεια και δυσκολεύονται να χειριστούν μεγάλο όγκο δεδομένων, ενώ οι ευρετικές προσεγγίσεις όπως το IPLoM παρέχουν συνεπή απόδοση και υψηλή ακρίβεια (~99%). Επιπλέον, οι downstream εργασίες εξαρτώνται από το ακριβές parsing, επειδή ακόμη και μικρά λάθη σε σημαντικά συμβάντα μπορούν να προκαλέσουν σοβαρά προβλήματα με την ανίχνευση ανωμαλιών και άλλες εφαρμογές [7].

Για παράδειγμα, σε ανίχνευση ανωμαλιών σε HDFS logs, η χρήση SLCT με 83% parsing accuracy οδήγησε σε 7.515 false alarms, ενώ ο IPLoM με 99% accuracy είχε μόλις 278 false alarms [7].

3.5 Επίλογος

Η ανάλυση που προηγήθηκε ανέδειξε ότι το log parsing αποτελεί μια διαδικασία καθοριστικής σημασίας για την αποτελεσματική αξιοποίηση των δεδομένων καταγραφής. Κάθε στρατηγική έχει πλεονεκτήματα και μειονεκτήματα, ανεξάρτητα από το αν βασίζεται σε μηχανική μάθηση, εξόρυξη συχνών μοτίβων, συσταδοποίηση ή ευρετικές. Το μέγεθος και η ετερογένεια των δεδομένων, η ανάγκη για ακρίβεια και αποτελεσματικότητα, και ο τύπος της εφαρμογής ανάλυσης επηρεάζουν όλα την επιλογή της μεθόδου.

Η διαδικασία parsing δεν είναι απλώς ένα προπαρασκευαστικό βήμα, αλλά καθορίζει σε μεγάλο βαθμό την ποιότητα, την ακρίβεια και την αξιοπιστία των αποτελεσμάτων σε downstream tasks όπως η ανίχνευση ανωμαλιών ή η διάγνωση προβλημάτων. Ο σχεδιασμός ενός αξιόπιστου συστήματος επεξεργασίας logs απαιτεί την κατανόηση των τεχνικών parsing, των περιορισμών τους και του τρόπου που επηρεάζουν την ανάλυση.

Στο επόμενο κεφάλαιο θα επικεντρωθούμε στις τεχνολογίες και τα εργαλεία οπτικοποίησης log δεδομένων, ανακαλύπτοντας πώς τα δομημένα logs που παράγονται μέσω parsing μπορούν να παρουσιαστούν με τρόπο κατανοητό και χρήσιμο για τον τελικό χρήστη.

Κεφάλαιο 4ο: Οπτικοποίηση Log Data

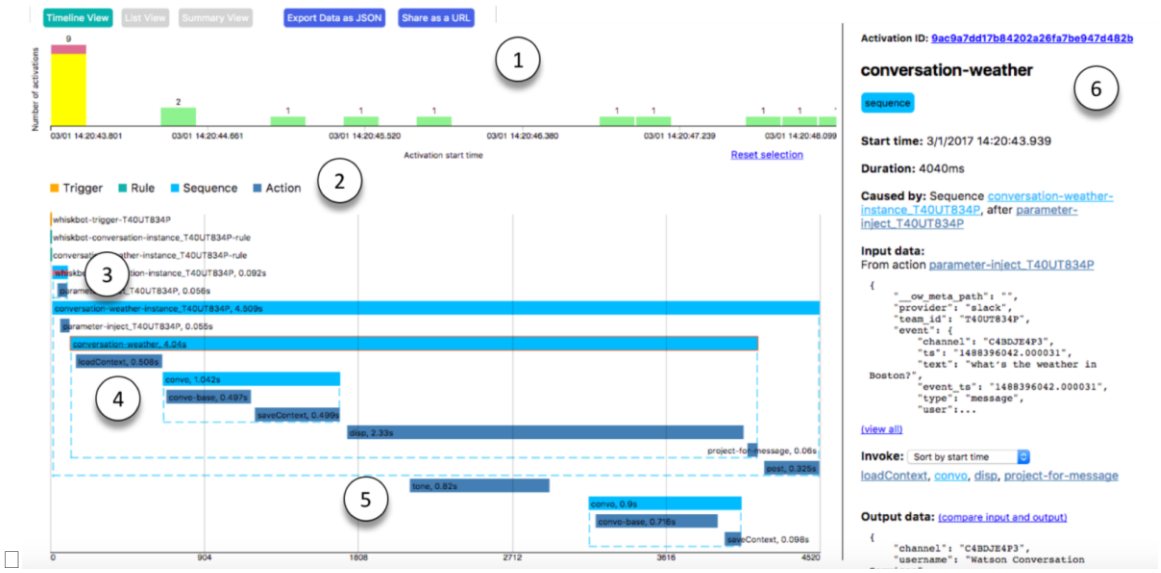
Η οπτικοποίηση δεδομένων καταγραφής (logs) αποτελεί απαραίτητο εργαλείο για την ουσιαστική κατανόηση και την ανάλυση της λειτουργίας πληροφοριακών συστημάτων. Η ικανότητα οπτικής αναπαράστασης δεδομένων βοηθά τους χρήστες στον εντοπισμό τάσεων, ανωμαλιών και σημαντικών σχέσεων, καθώς ο όγκος των δεδομένων αυξάνεται και οι δομές των logs παραμένουν πολύπλοκες και ποικίλες. Πέρα από την αισθητική, η οπτικοποίηση είναι ζωτικής σημασίας για τη μετατροπή ακατέργαστων δεδομένων σε γνώση που μπορεί να χρησιμοποιηθεί [9], [10].

4.1 Τεχνολογίες Οπτικοποίησης Log Data

4.1.1 Εργαλεία και Βιβλιοθήκες

Υπάρχει μεγάλη ποικιλία εργαλείων και βιβλιοθηκών για την οπτικοποίηση log δεδομένων. Το Streamlit, μια σύγχρονη πλατφόρμα για τη δημιουργία διαδραστικών εφαρμογών οπτικοποίησης ιστού, επιτρέπει τη γρήγορη ενσωμάτωση πινάκων, γραφημάτων και widgets αναζήτησης ή φιλτραρίσματος [11]. Ταυτόχρονα, βιβλιοθήκες όπως η Matplotlib και η Plotly παρέχουν τόσο στατικές όσο και διαδραστικές γραφικές δυνατότητες, από γραφήματα δικτύου (network graphs) και heatmaps έως βασικά γραφήματα γραμμών (line plots).

Εργαλεία όπως το Witt, τα οποία προσφέρουν προβολές χρονογραμμής (timeline views), σύνοψη JSON και διαδραστική εξερεύνηση ενεργοποιήσεων προγραμμάτων cloud, επικεντρώνονται σε οπτικοποιήσεις σχεδιασμένες ειδικά για εφαρμογές serverless για πιο εξειδικευμένα σενάρια [9].

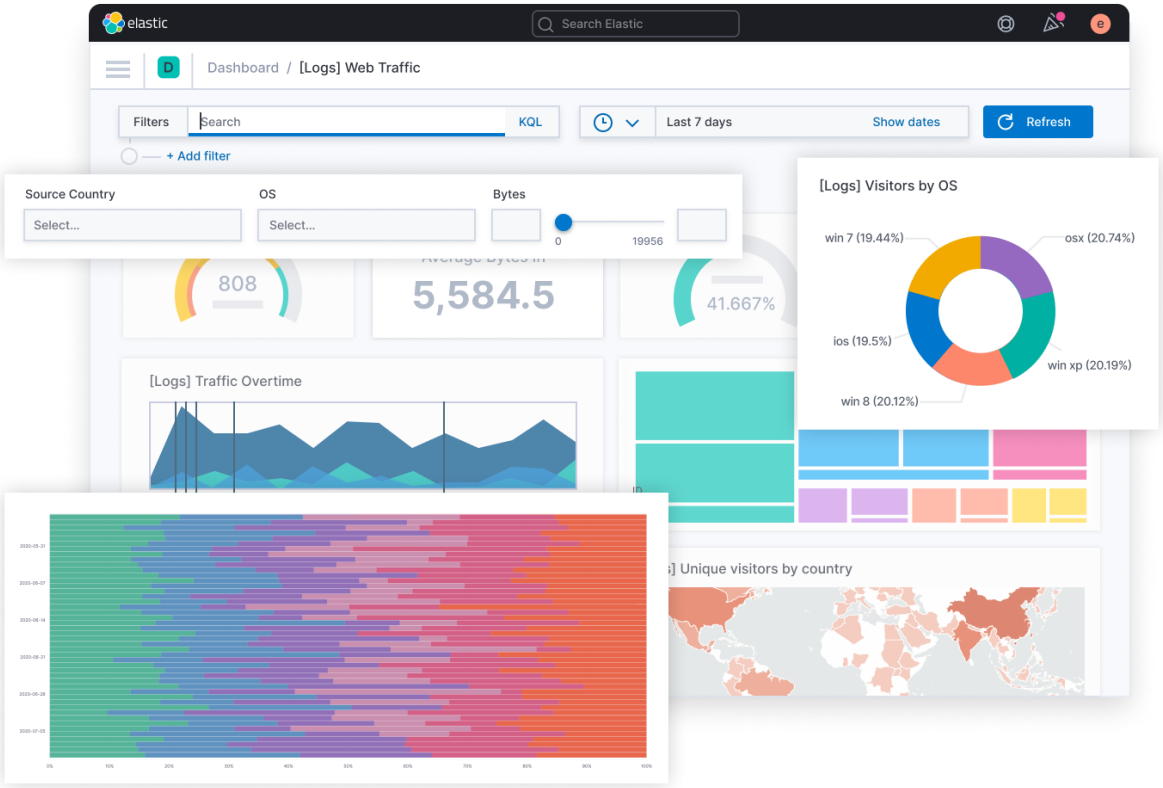


Σχήμα 4.1: Timeline visualization του Witt [9]

4.1.2 Χρήσεις dashboards για ανάλυση logs

Τα dashboards αποτελούν βασικό μηχανισμό για την οπτικοποίηση log δεδομένων, υποστηρίζοντας την επιχειρησιακή παρακολούθηση και την αναλυτική εξερεύνηση. Η βιβλιογραφία διακρίνει διαφορετικά είδη dashboards ανάλογα με το επίπεδο λήψης αποφάσεων: στρατηγικά, τακτικά και επιχειρησιακά [10]. Κάθε τύπος απαιτεί διαφορετική σχεδίαση ως προς το επίπεδο λεπτομέρειας, τον τύπο δεδομένων που προβάλλονται και τον τρόπο που αλληλεπιδράει μαζί τους ο χρήστης.

Τα σύγχρονα dashboards ενσωματώνουν λειτουργίες real-time monitoring, alerts, φιλτραρίσματος και συνδυασμένων views, επιτρέποντας στον χρήστη να εξετάζει διαφορετικές όψεις των δεδομένων. Εργαλεία όπως το ELK (Elasticsearch, Logstash, Kibana) και το Splunk χρησιμοποιούνται ευρέως για τέτοιες εφαρμογές [1].



Σχήμα 4.2: Παράδειγμα dashboard από Kibana [12]

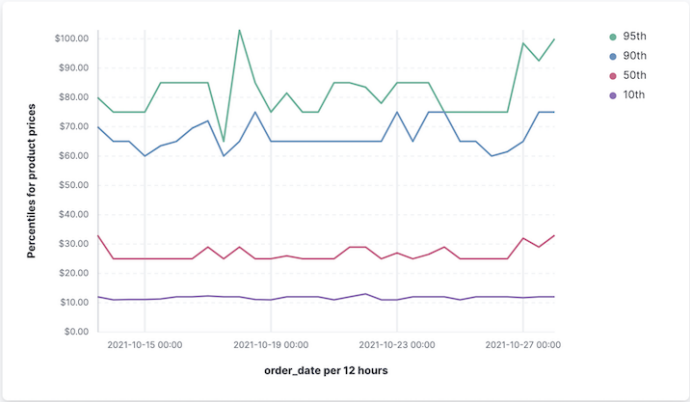
4.1.3 Επιλογή κατάλληλων γραφημάτων για log data visualization

Η επιλογή του κατάλληλου τύπου γραφήματος για την οπτικοποίηση logs εξαρτάται από το είδος της πληροφορίας και το ερώτημα που επιδιώκεται να απαντηθεί κάθε φορά. Για παράδειγμα:

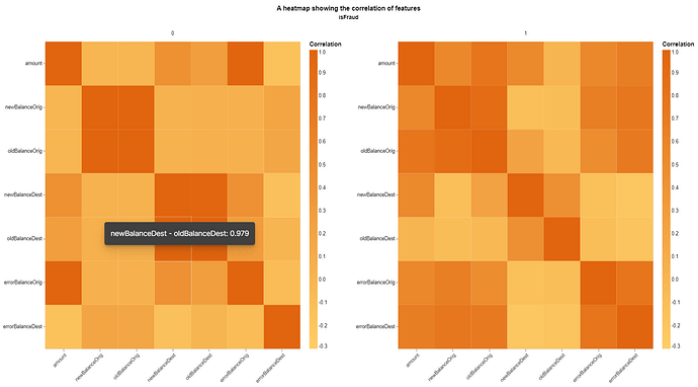
- Τα Timeline visualizations χρησιμοποιούνται όταν ο στόχος είναι η παρακολούθηση της ροής γεγονότων σε χρόνο, βοηθώντας τον χρήστη να εντοπίσει καθυστερήσεις, αλληλουχίες ενεργειών και συγχρονισμένα γεγονότα [9]. (Σχήμα 4.3)
- Τα Heatmaps είναι κατάλληλα για την προβολή συχνοτήτων γεγονότων ανά κατηγορία ή χρονική περίοδο, επιτρέποντας τον εντοπισμό περιοχών υψηλής δραστηριότητας ή ασυνήθιστων patterns [7]. (Σχήμα 4.4)

- Τα Network graphs ενδείκνυνται όταν ζητείται η απεικόνιση σχέσεων ή dependencies μεταξύ συμβάντων ή modules ενός συστήματος, προσφέροντας μια εικόνα της δομής του συστήματος και της διασύνδεσης των λειτουργιών [9]. (Σχήμα 4.5)
- Τα Histograms και bar charts υποστηρίζουν την ανάλυση τάσεων ή την ανίχνευση αλλαγών παρέχοντας μια σύνοψη της συχνότητας των συμβάντων ανά τύπο ή χρονική περίοδο [9]. (Σχήμα 4.6)

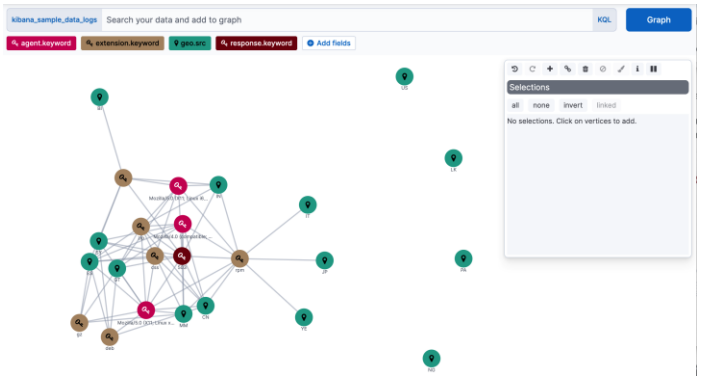
Για να αποφευχθούν υπερβολικά περίπλοκες οπτικοποιήσεις που υπερβαίνουν την ικανότητα κατανοήσης του κοινού, η επιλογή του γραφήματος θα πρέπει να λαμβάνει υπόψη το επίπεδο εμπειρίας του χρήστη [10].



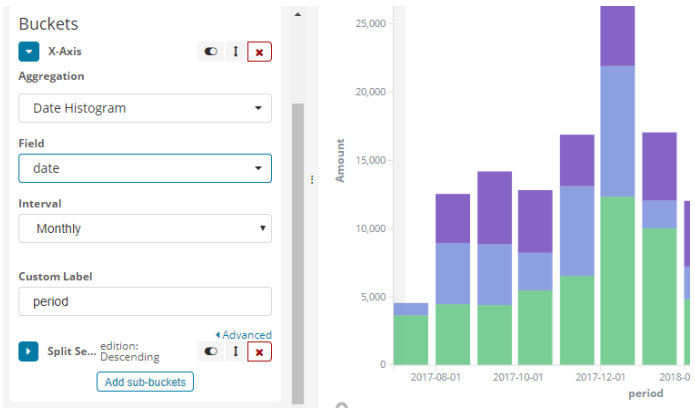
Σχήμα 4.3: Παράδειγμα Line Chart (Timeline) από Kibana



Σχήμα 4.4: Παράδειγμα Heatmap από Kibana



Σχήμα 4.5: Παράδειγμα network diagram με nodes και edges από Kibana



Σχήμα 4.6: Παράδειγμα Date Histogram από Kibana

4.2 Σχεδιαστικές Αρχές και UX στην Οπτικοποίηση Log Data

4.2.1 Διαδραστικότητα, Παραμετροποίηση και Φιλτράρισμα

Η δυνατότητα αλληλεπίδρασης με τα δεδομένα, όπως το φιλτράρισμα, το zoom, η επιλογή περιοχής ενδιαφέροντος και η παραμετροποίηση των views, αποτελεί σημαντικό στοιχείο ενός επιτυχημένου dashboard. Η δυνατότητα ελεύθερης παραμετροποίησης (free customization) ενισχύει την εμπειρία του χρήστη, επιτρέποντας προσαρμογή των visualizations ώστε να ανταποκρίνονται στις ανάγκες του [13].

4.2.2 Visualization literacy & και Εξειδίκευση Χρηστών

Η αποτελεσματικότητα μιας οπτικοποίησης εξαρτάται από την ικανότητα του χρήστη να την ερμηνεύσει σωστά. Σύνθετα γραφήματα, όπως network diagrams ή multi-layer dashboards, μπορεί να απαιτούν υψηλό επίπεδο γνώσης τόσο στο visualization όσο και στο domain, οπότε ένα σύνθετο visualization μπορεί να φανεί άχρηστο σε χρήστες χωρίς την απαραίτητη τεχνογνωσία, οδηγώντας σε εσφαλμένες ερμηνείες ή άγνοια σημαντικών insights. [10]. Η σχεδίαση πρέπει να παρέχει επεξηγηματικά εργαλεία, όπως tooltips, legends και narrative components, για να μειώσει το cognitive load.

4.2.3 Ιδιωτικότητα, Διαφάνεια και Ερμηνευσιμότητα

Η ανάλυση δεδομένων log μέσω dashboards μπορεί να εγείρει ζητήματα ιδιωτικότητας και διαφάνειας αφού μπορεί να περιλαμβάνει προσωπικά ή ευαίσθητα δεδομένα. Ιδιαίτερα σε περιβάλλοντα παρακολούθησης χρηστών ή συστημάτων, οι σχεδιαστές καλούνται να διασφαλίσουν ότι τα δεδομένα απεικονίζονται με τρόπο που σέβεται την ιδιωτικότητα αλλά παρέχει παράλληλα επαρκείς πληροφορίες για λήψη αποφάσεων [10]. Επιπλέον, η ερμηνευσιμότητα (interpretability) είναι κρίσιμη, ώστε ο τελικός χρήστης να κατανοεί πώς προέκυψαν τα αποτελέσματα της ανάλυσης και να μπορεί να τα αξιοποιήσει σωστά.

4.3 Επίλογος

Η οπτικοποίηση των log δεδομένων αποτελεί κρίσιμο εργαλείο για την αξιοποίηση του πλούτου της πληροφορίας που παρέχουν οι καταγραφές συστημάτων. Μέσα από την επιλογή κατάλληλων εργαλείων, τεχνικών και σχεδιαστικών αρχών, οι οπτικοποιήσεις καθιστούν εφικτή την κατανόηση σύνθετων patterns, την έγκαιρη ανίχνευση ανωμαλιών και τη λήψη τεκμηριωμένων αποφάσεων. Η επιτυχία μιας τέτοιας οπτικοποίησης εξαρτάται από τη σωστή ισορροπία μεταξύ πολυπλοκότητας, διαδραστικότητας και χρηστικότητας, ενώ η συμμόρφωση σε αρχές διαφάνειας και ιδιωτικότητας παραμένει απαραίτητη για τη βιωσιμότητα τέτοιων συστημάτων.

Κεφάλαιο 5ο: Εφαρμογές και Χρήσεις Log Analysis

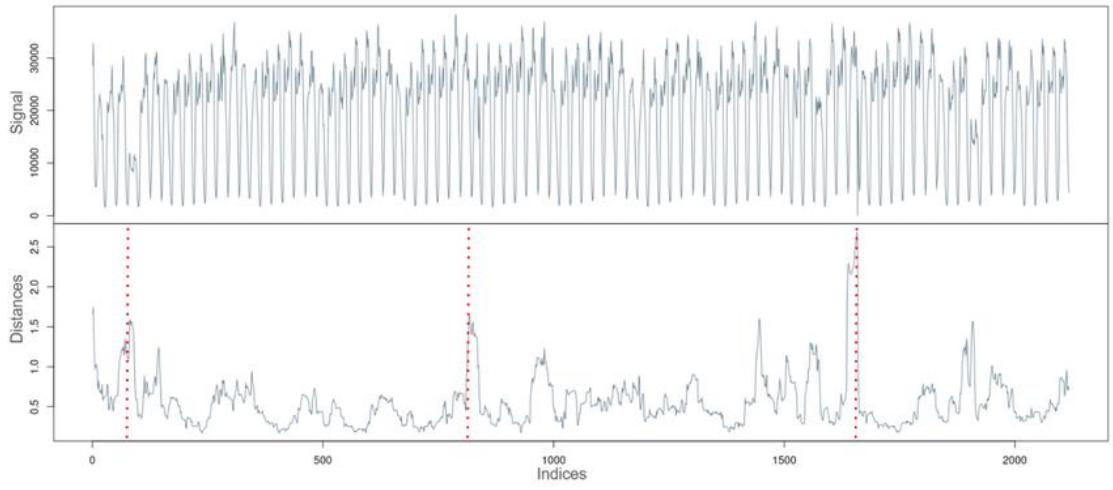
Η ανάλυση δεδομένων καταγραφής (log analysis) έχει εξελιχθεί σε ένα σημαντικό εργαλείο για τη διαχείριση, τη διάγνωση και τη βελτίωση της λειτουργίας σύγχρονων πληροφοριακών συστημάτων. Καθώς τα logs αποτελούν τον «καθρέφτη» της λειτουργίας ενός συστήματος, η αξιοποίησή τους προσφέρει πολύτιμες δυνατότητες σε πολλαπλά επίπεδα: από την άμεση ανίχνευση προβλημάτων έως τη βαθύτερη κατανόηση των διαδικασιών. Η ανίχνευση ανωμαλιών, η διάγνωση αιτίων σφαλμάτων (root cause analysis) και η παρακολούθηση και ανάλυση απόδοσης είναι οι τρεις κύριοι τομείς εφαρμογής της ανάλυσης logs που καλύπτονται σε αυτό το κεφάλαιο.

5.1 Ανίχνευση ανωμαλιών

Μία από τις πιο δημοφιλείς χρήσεις της ανάλυσης logs για την οποία έχουν γίνει αρκετές αναφορές σε προηγούμενα κεφάλαια είναι η ανίχνευση ανωμαλιών. Στόχος της είναι η αναγνώριση μη φυσιολογικών ή ύποπτων προτύπων στα δεδομένα, τα οποία μπορεί να υποδεικνύουν προβλήματα, επιθέσεις ή ασυνήθιστες καταστάσεις στο σύστημα. Η διαδικασία αυτή είναι ιδιαίτερα σημαντική σε μεγάλα και πολύπλοκα συστήματα, όπου ο όγκος των logs καθιστά αδύνατο τον χειροκίνητο έλεγχο [6].

Η βιβλιογραφία περιγράφει διάφορες μεθοδολογίες για την ανίχνευση ανωμαλιών μέσω logs. Ένα βασικό βήμα είναι η μετατροπή των raw logs σε event count matrix, όπου κάθε σειρά αντιστοιχεί σε ένα instance και κάθε στήλη σε ένα event type, με τις τιμές να δηλώνουν τον αριθμό εμφανίσεων. Αυτός ο πίνακας εισάγεται στη συνέχεια σε αλγόριθμους ανίχνευσης ανωμαλιών, όπως PCA ή machine learning μοντέλα, για την ανίχνευση αποκλίσεων [7].

Η επιτυχία της ανίχνευσης ανωμαλιών εξαρτάται σε μεγάλο βαθμό από την ακρίβεια του parsing. Όπως έχει ήδη αναφερθεί, ακόμη και ένα μικρό ποσοστό λαθών μπορεί να οδηγήσει σε υποβάθμιση της απόδοσης της ανάλυσης [7]. Αυτό συμβαίνει επειδή οι ανωμαλίες ενδέχεται να κρύβονται σε σπάνια ή λεπτομερή patterns, τα οποία χάνονται εάν η δομή των logs δεν αποδοθεί σωστά.



Σχήμα 5.1: Παράδειγμα απεικόνισης event count matrix για ανίχνευση ανωμαλιών [14]

Επιπλέον, τα dashboards για οπτικοποίηση ανωμαλιών σε πραγματικό χρόνο περιλαμβάνονται συχνά σε σύγχρονα συστήματα, επιτρέποντας στους διαχειριστές να επικεντρώνονται στα προβλήματα αμέσως. Τα heatmaps, τα timelines και τα alerts με thresholds είναι μερικά παραδείγματα αυτών των οπτικοποιήσεων.

5.2 Root Cause Analysis

Μια άλλη κρίσιμη χρήση της ανάλυσης logs είναι η Ανάλυση Βασικής Αιτίας (Root Cause Analysis, RCA). Στόχος της είναι ο εντοπισμός της βασικής αιτίας ενός προβλήματος, μέσα από τη διερεύνηση των γεγονότων που προηγήθηκαν ενός σφάλματος. Η διαδικασία αυτή στηρίζεται στη συσχέτιση συμβάντων και στην ανάλυση των σχέσεων μεταξύ διαφορετικών log entries [7].

Η RCA χρησιμοποιεί τεχνικές όπως η ακολουθιακή ανάλυση γεγονότων (sequence analysis) και η μοντελοποίηση συστημάτων με finite state machines για να αποδώσει τη ροή των γεγονότων που οδηγούν σε ένα failure [7]. Ένα παράδειγμα είναι το εργαλείο Synoptic, το οποίο εξάγει μοντέλα κατάστασης από logs και επιτρέπει την ανάλυση αποκλίσεων μεταξύ της κανονικής και της προβληματικής λειτουργίας.

Η επιτυχία της RCA προϋποθέτει όχι μόνο την ορθή εξαγωγή των events μέσω parsing, αλλά και την ικανότητα του εργαλείου ανάλυσης να συσχετίζει τα logs από διαφορετικά υποσυστήματα. Αυτό είναι ιδιαίτερα κρίσιμο σε κατανεμημένα συστήματα, επειδή οι περίπλοκες αλληλεπιδράσεις μπορούν να οδηγήσουν σε κάποιο failure.

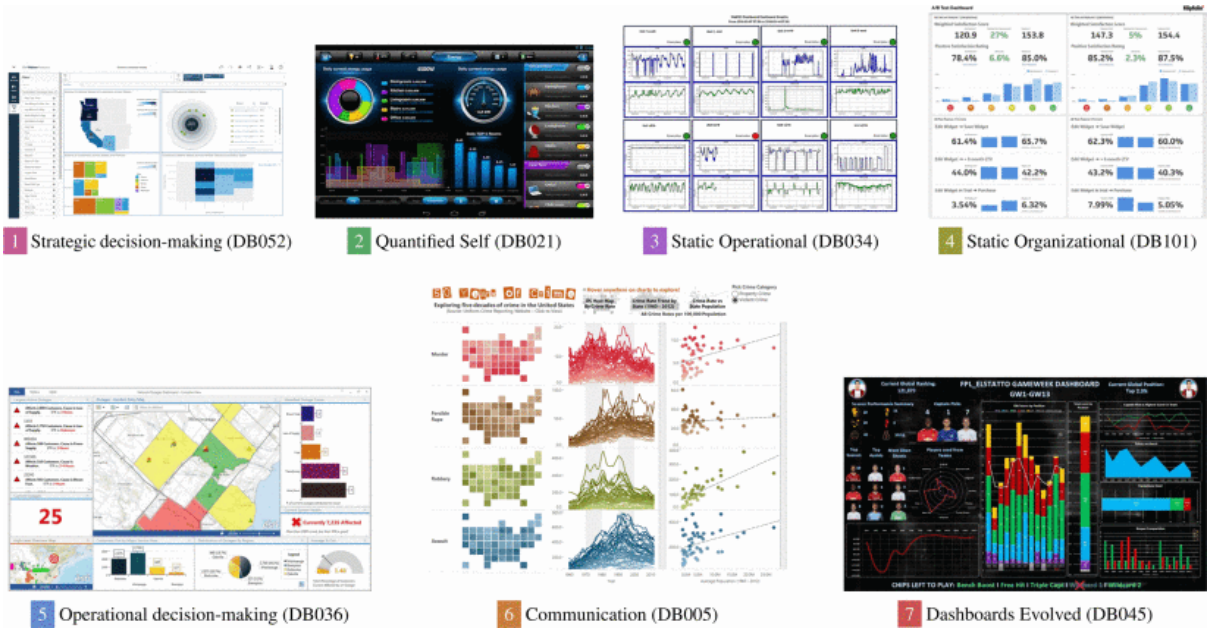
Τα dashboards με δυνατότητες drill-down σε συγκεκριμένες χρονικές περιόδους και linked views που συνδέουν διαφορετικούς τύπους δεδομένων (π.χ. metrics, logs, traces) είναι κοινά εργαλεία για την υποστήριξη της RCA.

5.3 Monitoring και Performance Analysis

Η παρακολούθηση (monitoring) και η ανάλυση απόδοσης (performance analysis) είναι δύο στενά συνδεδεμένες εφαρμογές την ανάλυση logs. Μέσα από τα logs, οι διαχειριστές συστημάτων αποκτούν πληροφόρηση για κρίσιμες παραμέτρους απόδοσης, όπως latency, throughput και resource utilization [6].

Οι μεταβλητές που σχετίζονται με την απόδοση μπορούν να εξαχθούν από τα logs και στη συνέχεια να υποβληθούν σε στατιστικά μοντέλα ή μοντέλα μηχανικής μάθησης για ανάλυση. Σε ένα παράδειγμα του Facebook, οι ακολουθίες συμβάντων πριν και μετά τις αλλαγές αναλύθηκαν χρησιμοποιώντας logs για την επιβεβαίωση της αποτελεσματικότητας των βελτιώσεων απόδοσης [6].

Η απεικόνιση δεδομένων monitoring συχνά περιλαμβάνει time-series γραφήματα, gauge meters και alerting thresholds. Τα dashboards επιτρέπουν στους διαχειριστές να παρακολουθούν σε πραγματικό χρόνο τις βασικές μετρήσεις, ενώ η δυνατότητα ιστορικής ανασκόπησης επιτρέπει την ανάλυση τάσεων.



Σχήμα 5.2: Ενδεικτικά dashboards για στρατηγική, τακτική και επιχειρησιακή χρήση [10]

Η ανάλυση απόδοσης με logs μπορεί επίσης να περιλαμβάνει τη δημιουργία μοντέλων ροής εργασιών (workflow models) ή συστημάτων dependencies που αποτυπώνουν τον τρόπο με τον οποίο οι λειτουργίες του συστήματος συνδέονται μεταξύ τους, επιτρέποντας τον εντοπισμό bottlenecks.

5.4 Επίλογος

Οι εφαρμογές για την ανάλυση δεδομένων καταγραφής είναι πολυάριθμες και υποστηρίζουν τόσο τις επιχειρησιακές ανάγκες (ανίχνευση ανωμαλιών, monitoring) όσο και πιο σύνθετες αναλυτικές διεργασίες (root cause analysis, performance validation). Επιπλέον εφαρμογές επισημαίνονται επίσης στη βιβλιογραφία, συμπεριλαμβανομένου του αυτόματου εντοπισμού επαναλαμβανόμενων ζητημάτων (duplicate issue identification) και της επαλήθευσης deployments μέσω σύγκρισης log sequences από διαφορετικά περιβάλλοντα [6], [7].

Στο επόμενο κεφάλαιο, η ανάλυση που έγινε περνάει από τη θεωρία στην πράξη, μέσω της περιγραφής της υλοποίησης ενός συστήματος επεξεργασίας και οπτικοποίησης log δεδομένων.

Κεφάλαιο 6ο: Σχεδίαση και Ανάπτυξη Συστήματος

Αυτή η ενότητα επικεντρώνεται στην υλοποίηση ενός ολοκληρωμένου συστήματος ανάλυσης αρχείων καταγραφής, σε συνέχεια της ανάλυσης των θεμελιωδών ιδεών που σχετίζονται με τα logs, τις στρατηγικές parsing και τις τεχνικές οπτικοποίησης. Το σύστημα δημιουργήθηκε για να υποστηρίζει διάφορες μορφές log, να επιτρέπει την ευέλικτη επιλογή parser και να παρέχει στους χρήστες μια διαδραστική εμπειρία για την ανάλυση και οπτικοποίηση των αποτελεσμάτων.

Η εφαρμογή υλοποιήθηκε εξολοκλήρου σε Python και βασίστηκε στη βιβλιογραφική έρευνα που παρουσιάστηκε στα προηγούμενα κεφάλαια. Ιδιαίτερη έμφαση δόθηκε στην modular αρχιτεκτονική, στην παραμετροποίηση των parsers και στην προσεκτική σχεδίαση του interface με βάση βέλτιστες πρακτικές UX.

Η αρχιτεκτονική του συστήματος, η τεχνική υλοποίηση του μηχανισμού parsing, η διαδραστική διεπαφή χρήστη και η ενσωμάτωση τεχνικών στατιστικής ανάλυσης και οπτικοποίησης καλύπτονται λεπτομερώς στις ακόλουθες ενότητες. Εξηγείται επίσης η διαδικασία αξιολόγησης (benchmarking) για την επιλογή των καλύτερων παραμέτρων, μαζί με παραδείγματα περιπτώσεων χρήσης και στρατηγικές βελτιστοποίησης κώδικα που χρησιμοποιήθηκαν για την αύξηση της απόδοσης και της επεκτασιμότητας του συστήματος.

6.1 Αρχιτεκτονική του Συστήματος

6.1.1 Γενική περιγραφή της αρχιτεκτονικής και της ροής δεδομένων

Το σύστημα που αναπτύχθηκε αποτελεί μια πλήρως διαδραστική εφαρμογή ανάλυσης log αρχείων, η οποία ενσωματώνει parsing, τυποποίηση και οπτικοποίηση των δεδομένων. Η αρχιτεκτονική του είναι σχεδιασμένη με γνώμονα την ευελιξία στην είσοδο, την επεκτασιμότητα των parsing μηχανισμών και τη modular υλοποίηση για ευκολία συντήρησης και επέκτασης.

Η βασική ροή δεδομένων ξεκινάει από τον χρήστη, ο οποίος ανεβάζει ένα log αρχείο μέσω του Streamlit interface. Στη συνέχεια, ο χρήστης επιλέγει τον parser που θα χρησιμοποιηθεί και τον τύπο αρχείου (π.χ., Windows, Linux, Mac, Suricata). Ανάλογα με τον τύπο του αρχείου, το σύστημα δρομολογεί την είσοδο στο αντίστοιχο module και εκτελεί τον επιλεγμένο parser με συγκεκριμένες ρυθμίσεις. Το αποτέλεσμα είναι μια συλλογή οργανωμένων logs και αντίστοιχων προτύπων που εμφανίζονται σε διαφορετικές καρτέλες μέσω του user interface.

Ο πυρήνας του parsing υλοποιείται μέσω modules που εισάγονται δυναμικά ανάλογα με την επιλογή του χρήστη. Το τελικό αποτέλεσμα εμφανίζεται σε διαδραστικά dashboards και πίνακες, με δυνατότητες φιλτραρίσματος, αναζήτησης και οπτικής ανάλυσης.

Κάθε φορά που ο χρήστης τροποποιεί μια ρύθμιση, το σύστημα επαναφορτώνει τη ροή δεδομένων για να αντικατοπτρίσει τις ενημερωμένες παραμέτρους, επιδεικνύοντας την αρχιτεκτονική "κυκλικής επεξεργασίας" της ροής της εφαρμογής. Αυτή η προσέγγιση ευνοείται ιδιαίτερα από τη χρήση του Streamlit, καθώς κάθε user interaction προκαλεί επανεκτέλεση του script από την αρχή, επιτρέποντας δυναμική ανανέωση της οθόνης με ελάχιστο κώδικα χειρισμού κατάστασης [11].

Σχήμα 6.1: Διάγραμμα ροής για τη βασική λειτουργία της εφαρμογής

Αξίζει να σημειωθεί ότι η υλοποίηση ακολουθεί αρχές modular σχεδίασης, όπου κάθε τύπος log έχει το δικό του script parsing (π.χ. windows_logs.py, linux_logs.py κ.λπ.) και η λογική parsing έχει αφαιρεθεί στο κοινό αρχείο log_utils.py, επιτρέποντας επαναχρησιμοποίηση και μείωση πλεονασμού στον κώδικα.

Αυτός ο διαχωρισμός καθιστά δυνατή τη σταδιακή επέκταση του συστήματος, π.χ. με την προσθήκη νέων parsers ή τη σύνδεση με log pipelines άλλων υπηρεσιών, διατηρώντας το σύστημα συμβατό με τις αρχές modular software architecture, όπως περιγράφονται και στη σχετική βιβλιογραφία για parsing frameworks [7].

6.1.2 Επιλογή γλωσσών, εργαλείων και βιβλιοθηκών

Η γλώσσα προγραμματισμού που επιλέχθηκε για το έργο είναι η Python, λόγω της εκτεταμένης υποστήριξης σε parsing, data manipulation και visualization βιβλιοθήκες, καθώς και της απλότητας ενσωμάτωσης εφαρμογών στον ιστό.

Το βασικό εργαλείο για την υλοποίηση του interface είναι το Streamlit, μια βιβλιοθήκη που επιτρέπει τη δημιουργία web εφαρμογών με Python χωρίς την ανάγκη frontend γνώσεων. Η χρήση του Streamlit εξασφαλίζει γρήγορη ανάπτυξη, εύκολη ενσωμάτωση με Pandas και Altair, καθώς και άμεση ανταπόκριση στις αλλαγές δεδομένων, χάρη στον μηχανισμό ανανέωσης της εφαρμογής με κάθε user interaction [11].

Για την οπτικοποίηση των αποτελεσμάτων, χρησιμοποιήθηκε η Altair, μια declarative βιβλιοθήκη για στατιστική απεικόνιση με βάση την τεχνολογία Vega-Lite. Η Altair επιλέχθηκε λόγω της συμβατότητας με Pandas DataFrames, της υποστήριξης διαδραστικότητας (filters, tooltips, selections) και της δυνατότητας ενσωμάτωσης σε Streamlit με μικρό κώδικα.

Για την υλοποίηση ενός συγκεκριμένου γραφήματος χρησιμοποιήθηκε η βιβλιοθήκη Plotly. Η Plotly είναι ένα ισχυρό εργαλείο για web-based visualizations και επιτρέπει τη δημιουργία διαδραστικών γραφημάτων υψηλής ποιότητας με δυνατότητες όπως zoom, pan, hover και επιλογή περιοχών. Παράγει γραφήματα HTML και ενσωματώνεται ομαλά σε Streamlit μέσω του st.plotly_chart. Το Plotly χρησιμοποιήθηκε για ένα συγκεκριμένο use case που απαιτούσε πιο περίπλοκη οπτικοποίηση και βελτιωμένη διαδραστικότητα, ακόμα κι αν το Altair είναι επαρκές για την πλειονότητα των εφαρμογών [15].

Συνοδευτικές βιβλιοθήκες και εξαρτήσεις

Επιπλέον, το σύστημα χρησιμοποιεί:

- Pandas: για επεξεργασία και διαχείριση πινάκων
- Regex: για ανάλυση logs και αναγνώριση μεταβλητών
- Json / Yaml: για αποθήκευση και φόρτωση ρυθμίσεων παραμέτρων
- os / pathlib: για διαχείριση αρχείων και paths

Περιβάλλον Ανάπτυξης

Η εφαρμογή αναπτύχθηκε στο περιβάλλον Anaconda, μια πλατφόρμα που παρέχει ολοκληρωμένη διαχείριση βιβλιοθηκών και εικονικών περιβαλλόντων. Η ευκολία ρύθμισης παραμέτρων εξαρτήσεων και η αναπαραγωγιμότητα πειραμάτων του Anaconda το καθιστούν δημοφιλή επιλογή για επιστημονικούς υπολογισμούς και μηχανική μάθηση

Οι ακόλουθες εκδόσεις ήταν μέρος του τελικού περιβάλλοντος ανάπτυξης:

- Python 3.12.7
- Streamlit 1.37.1
- Altair 5.0.1
- Plotly 5.24.1
- Pandas 2.2.2
- NumPy 1.26.4
- Regex 2022.3.2

Υποστήριξη Parsers και Περιορισμοί

Για την υλοποίηση του parsing engine χρησιμοποιήθηκαν parsers από το επίσημο αποθετήριο του LogPai [16]. Ορισμένοι από αυτούς, όπως οι Drain και IPLoM, παρουσιάζουν περιορισμούς σχετικά με τη συμβατότητα με νεότερες εκδόσεις Python και απαιτούν τροποποιήσεις ή εκτέλεση μέσω wrappers.

Επιπλέον, αρκετοί parsers δεν είναι πλήρως ανεξάρτητοι από το σύστημα αρχείων ή περιβάλλοντος, γεγονός που περιορίζει τη δυνατότητα χρήσης τους σε πολυνηματικά περιβάλλοντα ή την εύκολη ενσωμάτωσή τους ως modules. Κατά την ενσωμάτωσή τους εφαρμόστηκαν στοχευμένες προσαρμογές ώστε να εξασφαλιστεί η λειτουργικότητα και η σταθερότητα της εφαρμογής στο περιβάλλον Python 3.12.

6.2 Parsing Engine

Η μηχανή parsing αποτελεί τον λειτουργικό πυρήνα του συστήματος, επιφορτισμένη με τη μετατροπή των αδόμητων log δεδομένων σε δομημένες εγγραφές, οι οποίες μπορούν να αναλυθούν περαιτέρω. Η υλοποίηση ακολουθεί τις γενικές αρχές της βιβλιογραφίας περί διαχωρισμού σταθερών και μεταβλητών τμημάτων, δημιουργίας templates και εξαγωγής γεγονότων με σαφή δομή [7].

6.2.1 Υποστήριξη αρχείων πολλαπλών τύπων (Windows, Linux, Mac, Suricata)

Το σύστημα έχει σχεδιαστεί έτσι ώστε να υποστηρίζει ετερογενή είδη log αρχείων, μέσω τεσσάρων επιμέρους parsing modules:

- windows_logs.py
- linux_logs.py
- mac_logs.py
- suricata_logs.py

Κάθε module περιέχει προκαθορισμένες συναρτήσεις parsing για τον αντίστοιχο τύπο αρχείου. Αυτές οι συναρτήσεις υλοποιούν βασικά βήματα preprocessing, όπως κανονικοποίηση πεδίων, απομάκρυνση περιττών χαρακτήρων, τμηματοποίηση του κειμένου, και μετατροπή σε Pandas DataFrame. Ο

διαχωρισμός αυτός καθιστά ευκολότερη την επεκτασιμότητα, καθώς επιτρέπει την προσθήκη νέου τύπου log με ανεξάρτητο parsing αρχείο χωρίς επεμβάσεις στον κύριο κώδικα της εφαρμογής.

Οι δομικές ιδιαιτερότητες κάθε τύπου log καθορίζουν ποια πεδία (π.χ. timestamp, source, message) χρησιμοποιούνται, αλλά το πεδίο "message" επεξεργάζεται χρησιμοποιώντας μια τυπική μέθοδο: data-driven parsing για την προετοιμασία του για εξαγωγή προτύπων.

6.2.2 Δυναμική επιλογή parser από χρήστη (Spell, Drain, LogCluster κ.ά.)

Μετά την εισαγωγή αρχείου, ο χρήστης καλείται να επιλέξει parser μέσα από επιλογή τύπου:

- Spell
- Drain
- LogCluster
- IPLoM
- MoLFI

Η κάθε μέθοδος αντανακλά διαφορετική προσέγγιση parsing: οι Spell [5] και Drain [17] εφαρμόζουν heuristics με βάση σταθερά μοτίβα ή δομές prefix trees· το LogCluster χρησιμοποιεί clustering για την ομαδοποίηση παρόμοιων εγγραφών [18]· το IPLoM εφαρμόζει ιεραρχικό partitioning με βάση τον αριθμό και τη θέση των tokens [19]· ενώ το MoLFI βασίζεται σε πολυαντικειμενική βελτιστοποίηση, με χρήση εξελικτικών αλγορίθμων για την αναζήτηση templates [4].

Η δυνατότητα επιλογής επιτρέπει στον χρήστη να πειραματιστεί με διαφορετικούς parsers, όπως προτείνεται και στη βιβλιογραφία, όπου επισημαίνεται ότι οι χρήστες συχνά επανασχεδιάζουν αλγορίθμους parsing χωρίς να αξιοποιούν υπάρχουσες εναλλακτικές [7].

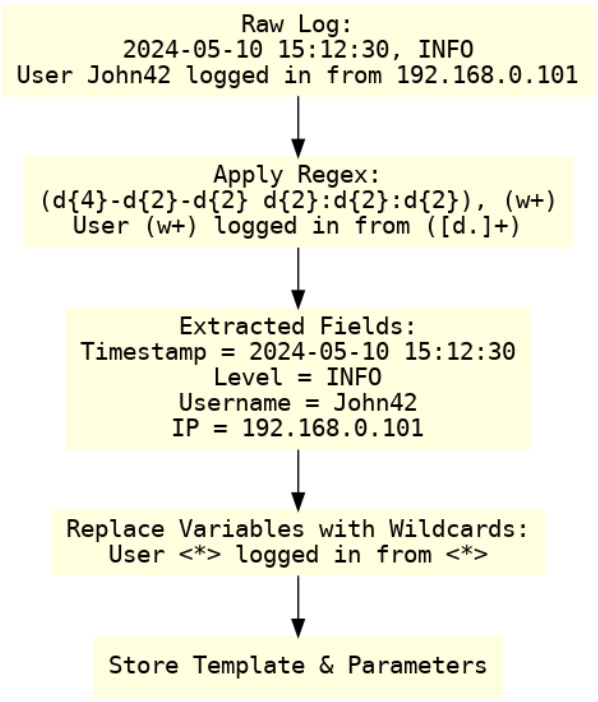
Η εσωτερική υλοποίηση αξιοποιεί mappings (PARSER_DEFAULTS, Παράρτημα Α) για να συνδέσει τον τύπο parser με τις προκαθορισμένες ρυθμίσεις ανάλογα με το format του αρχείου, αποφεύγοντας περιττό configuration.

6.2.3 Αντιστοίχιση κατάλληλων regex & παραμέτρων ανά περίπτωση

Για κάθε τύπο αρχείου έχουν οριστεί κατάλληλες regular expressions (regex), οι οποίες επιτρέπουν:

- τη δομική ανάλυση των log μηνυμάτων,
- τον εντοπισμό μεταβλητών πεδίων όπως IP διευθύνσεις, αναγνωριστικά, ή timestamps,
- την υποβοήθηση της κατηγοριοποίησης των εγγραφών κατά τη φάση template extraction.

Η χρήση regex είναι ιδιαίτερα διαδεδομένη σε parsers όπως το Spell, το οποίο βασίζεται στη σύγκριση λεκτικών μοτίβων και την εξαγωγή κοινού προτύπου μεταξύ πολλαπλών log entries. Όπως περιγράφεται στη βιβλιογραφία, η προκαταρκτική τμηματοποίηση των log μηνυμάτων με βάση regex βελτιώνει τη συνοχή των clusters και την ποιότητα των templates [7].



Σχήμα 6.2: Βήματα χρήσης κανονικών εκφράσεων (regex) για την αναγνώριση μεταβλητών και τη δημιουργία template από log message

Ορισμένοι parsers, όπως το Spell, χρησιμοποιούν παραμέτρους όπως tau ή max_dist, των οποίων οι τιμές καθορίζονται από συντηρητικές προεπιλεγμένες επιλογές ή benchmarking (βλ. Ενότητα 6.5). Το log_utils.py χειρίζεται την αντιστοίχιση παραμέτρων και την εκτέλεση. Η συνάρτηση load_parser() φορτώνει δυναμικά την επιλεγμένη μονάδα parser εφαρμόζοντας είτε παραμέτρους που επιλέγονται από τον χρήστη είτε παραμέτρους που ανατίθενται αυτόματα.

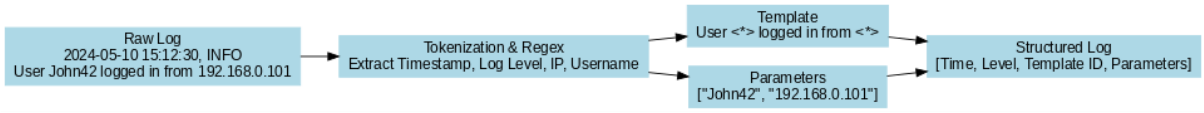
Χρησιμοποιώντας την έννοια της χαλαρής ζεύξης (loose coupling), αυτή η αρχιτεκτονική διαχωρίζει τη λογική parsing από τα επίπεδα παρουσίασης και διεπαφής. Η χαλαρή ζεύξη περιγράφει τη μείωση των αλληλεξαρτήσεων μεταξύ των τμημάτων ενός συστήματος. Αυτό βελτιώνει τη συντηρησιμότητα και την επεκτασιμότητα του έργου, επειδή ο parser μπορεί να αλλάξει, να επεκταθεί ή να αντικατασταθεί χωρίς να επηρεαστεί το υπόλοιπο της εφαρμογής [7].

6.2.4 Δημιουργία structured logs & templates

Το τελικό αποτέλεσμα της διαδικασίας parsing είναι δύο σύνολα εξόδου:

- Parsed Logs: δομημένα logs σε μορφή πίνακα, με πεδία όπως timestamp, level, source, message και event ID.
- Templates: το σύνολο των μοναδικών templates που εξήχθησαν από τα logs.

Σύμφωνα με το θεωρητικό υπόβαθρο, το parsing διαχωρίζει κάθε log εγγραφή σε σταθερό (template) και μεταβλητό (parameters) μέρος, ώστε κάθε καταγραφή να αντιστοιχιστεί σε ένα event type, σύμφωνα με τη βασική αρχή εξαγωγής προτύπων από log δεδομένα [7].



Σχήμα 6.3: Παράδειγμα μετατροπής αδόμητης καταγραφής σε πρότυπο γεγονός και δομημένη εγγραφή μέσω parsing

Τα structured δεδομένα αποθηκεύονται προσωρινά στη μνήμη της εφαρμογής και προβάλλονται στις ενότητες "Parsed Logs" και "Templates" της διεπαφής. Η έξοδος είναι συμβατή με log mining τεχνικές όπως anomaly detection και μπορεί να εξαχθεί για εξωτερική χρήση.

6.3 Διαδραστικό Περιβάλλον Χρήστη

Η σχεδίαση του διαδραστικού περιβάλλοντος ενός συστήματος ανάλυσης log δεδομένων είναι ένα σημαντικό λειτουργικό στοιχείο που επηρεάζει τη χρηστικότητα, την παραγωγικότητα του χρήστη και την αποτελεσματικότητα της ανάλυσης. Το υλοποιημένο σύστημα προσέφερε ένα ευέλικτο και προσαρμόσιμο μοντέλο αλληλεπίδρασης και βασίστηκε σε θεμελιώδεις αρχές σχεδιασμού διεπαφών, όπως το faceted filtering και το progressive disclosure.

Η αρχή του progressive disclosure στοχεύει στο να εμφανίζονται πρώτα οι πιο βασικές ή αναγκαίες επιλογές, και στη συνέχεια, σταδιακά, οι πιο προχωρημένες ή εξειδικευμένες ρυθμίσεις. Έτσι, αποφεύγεται η γνωσιακή υπερφόρτωση και επιτρέπεται στον χρήστη να επικεντρώνεται σε ό,τι είναι πιο κρίσιμο τη δεδομένη στιγμή.

Με παρόμοιο τρόπο, το faceted filtering επιτρέπει στους χρήστες να φιλτράρουν δεδομένα χρησιμοποιώντας διάφορα συνδυασμένα φίλτρα που συνδέονται με διάφορα πεδία. Αυτή η μέθοδος βελτιώνει την ανάκτηση πληροφοριών σε τεράστιους όγκους δεδομένων και χρησιμοποιείται συχνά στον σχεδιασμό dashboards και διαδραστικών εργαλείων [10].

6.3.1 Εισαγωγή και Ροή Χρήσης

Ο χρήστης πρέπει πρώτα να αναλύσει ένα γνωστό αρχείο log, και η διεπαφή της εφαρμογής τον καθοδηγεί βήμα-βήμα μέσω της επεξεργασίας και ερμηνείας των δεδομένων. Η γενική εμπειρία χρήστη έχει ως εξής:

1. Ανέβασμα αρχείου (χρησιμοποιώντας file picker ή drag & drop).
2. Επιλογή του τύπου αρχείου log με βάση τη γνώση που ήδη έχει ο χρήστης (π.χ. «πρόκειται για Windows logs»).
3. Επιλογή parser, όπου η λίστα ενημερώνεται δυναμικά με βάση τον τύπο του αρχείου.
4. Εισαγωγή παραμέτρων parser, με default τιμές που μπορούν να τροποποιηθούν.
5. Πάτημα ενός κουμπιού " Parse" για την έναρξη της ανάλυσης.
6. Χρήση tabs για πλοήγηση στα αποτελέσματα:
    - Dashboard (στατιστικά & γραφήματα)
    - Parsed Logs (δομημένα logs)
    - Templates (πρότυπα συμβάντων)

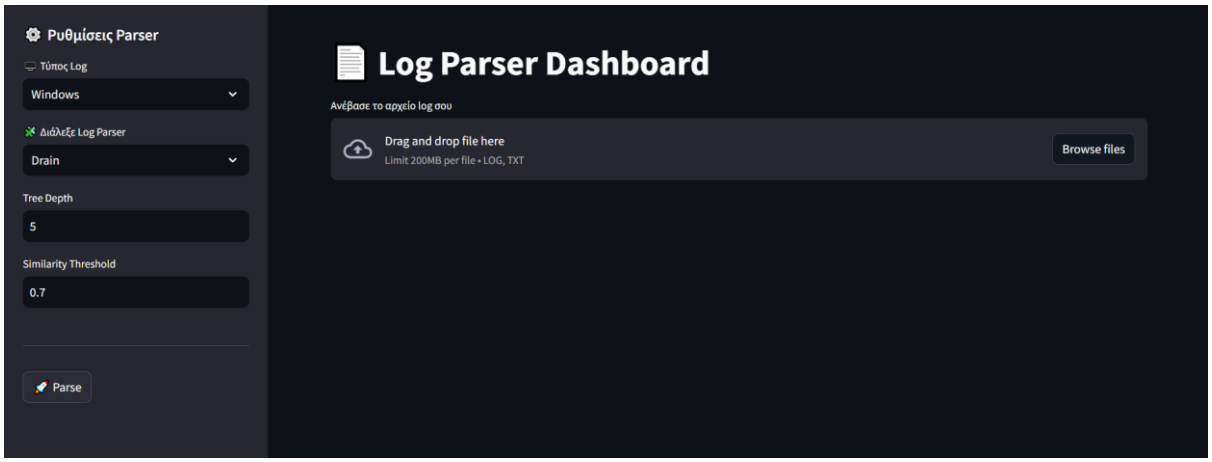
Είναι σκόπιμο η επιλογή τύπου αρχείου να προηγείται της επιλογής parser, επειδή οι περισσότεροι χρήστες γνωρίζουν τον τύπο του αρχείου που έχουν, αλλά δεν είναι σίγουροι για τον καλύτερο parser. Με βάση τον τύπο αρχείου, το σύστημα προβάλλει μόνο συμβατούς parsers, αποφεύγοντας γνωσιακή φόρτιση και μειώνοντας τα λάθη επιλογής.

6.3.2 Επιλογές Εισόδου Χρήστη (Streamlit Sidebar)

Όλες οι ρυθμίσεις και είσοδοι του χρήστη συγκεντρώνονται στο st.sidebar, ακολουθώντας το πρότυπο «μοντέλο εργαλείων στην άκρη» (toolbox on the side). Εκεί οργανώνονται:

- Επιλογή τύπου log (st.selectbox): π.χ. Windows, Linux, Suricata.
- Δυναμική επιλογή parser, η οποία εξαρτάται από το log type.
- Εμφάνιση πεδίων ρυθμίσεων, με χρήση st.text_input για κάθε παράμετρο parser, βασισμένη στα PARSER_DEFAULTS.
- Εκκίνηση parsing με st.button.

Η διεπαφή είναι context-aware: εφόσον ο χρήστης επιλέξει έναν τύπο αρχείου, το interface παρουσιάζει μόνο τις σχετικές επιλογές parser και τις αντίστοιχες παραμέτρους. Η χρήση Python dictionaries και δυναμικής render λογικής επιτρέπει τη δημιουργία της φόρμας σε πραγματικό χρόνο. Η Streamlit διαχειρίζεται την εσωτερική κατάσταση (st.session_state) ώστε να διατηρούνται οι τιμές σε κάθε επανεκτέλεση του script [11].



Σχήμα 6.4: Διεπαφή εισόδου χρήστη με επιλογή τύπου log και parser

6.3.3 Διαχείριση και Εφαρμογή Φίλτρων

Μετά το parsing, ο χρήστης μπορεί να φιλτράρει τα αποτελέσματα με τρόπο εργονομικό και άμεσο. Το filtering εξαρτάται από τα πεδία που περιέχει κάθε log και πραγματοποιείται αποκλειστικά client-side. Χρησιμοποιούνται:

- st.multiselect: για φίλτρα κατηγορικών πεδίων (π.χ. log level, component, protocol).
- st.date_input: για χρονικά διαστήματα.
- st.columns: για οριζόντια ομαδοποίηση φίλτρων στην οθόνη.
- Προαιρετική επιλογή “ALL” για την αποδοχή όλων των τιμών.

Η επιλογή φίλτρων δεν προκαλεί επανεκτέλεση parsing. Το σύστημα εφαρμόζει query masking σε Pandas DataFrames στη μνήμη. Ο μηχανισμός filtering υλοποιείται χωρίς επανεκτέλεση parsing, καθώς οι αλλαγές στους πίνακες και τα γραφήματα βασίζονται σε μεταβολές του session_state ή των widgets, δυνατότητα που ενισχύεται από το εσωτερικό rerun μοντέλο του Streamlit [11].

Η βιβλιογραφία αναφέρεται σε αυτό το είδος αλληλεπίδρασης ως "faceted interaction", η οποία επιτρέπει στους χρήστες να επικεντρώνονται αποκλειστικά στις πληροφορίες που είναι σχετικές με τις απαιτήσεις τους [10].

Σχήμα 6.5: Εφαρμογή φίλτρων ανά πεδίο με χρήση multiselect και χρονικών επιλογών

6.3.4 Παρουσίαση Δομημένων Δεδομένων

Υπάρχουν δύο κύριες προβολές για τα αποτελέσματα του parsing:

- Structured Logs: Ένας πίνακας με τα parsed logs χωρισμένα σε πεδία όπως timestamp, level, log level, message κ.ά. και αντιστοίχιση με event id, event template και parameter list.
- Templates: πίνακας με τα πρότυπα που εξήχθησαν από τα logs, περιλαμβάνοντας το αναγνωριστηκό id, το κείμενο του template και το πλήθος εμφανίσεων.

Η παρουσίαση υλοποιείται με χρήση της συνάρτησης st.dataframe, η οποία προσφέρει ένα σύνολο από αυτόματες λειτουργίες που ενισχύουν τη διαδραστικότητα και την αναλυτική δυνατότητα του χρήστη:

- Ταξινόμηση κατά στήλη (αύξουσα/φθίνουσα)
- Αναζήτηση βάσει περιεχομένου (search)
- Εναλλαγή μεγέθους/προβολής (πλήρης οθόνη)
- Εξαγωγή CSV μέσω του μενού εξαγωγής
- Επικοινωνία με το session state, ώστε να διατηρούνται οι επιλογές του χρήστη κατά την πλοήγηση

Επιπλέον, το Dashboard-Analytics, το οποίο χρησιμοποιεί τα δομημένα δεδομένα για την παραγωγή στατιστικών και οπτικοποιήσεων, είναι χτισμένο πάνω σε αυτή την προβολή. Για παράδειγμα, οι χρήστες μπορούν να δουν την χρονική εξέλιξη των συμβάντων, τη συχνότητα συγκεκριμένων προτύπων ή την κατανομή των log levels.

Η διπλή επίγνωση, τόσο ποσοτική όσο και σε επίπεδο περιεχομένου, καθίσταται δυνατή από την ικανότητα ταυτόχρονης πλοήγησης τόσο σε ακατέργαστα όσο και σε επεξεργασμένα δεδομένα, γεγονός που διευκολύνει τη μετάβαση από την παρατήρηση στην ερμηνεία.

	Date	Time	Level	Component	Content	EventId	EventTemplate	ParameterList
11	2016-09-28	04:30:31	Info	CBS	SQM: Failed to start standard sample upload: [HRESULT = 0x80004005]	e1b6b6df	SQM: Failed to start standard sample upload: [HRESULT = <*> - E_FAIL]	['0x80004005']
12	2016-09-28	04:30:31	Info	CBS	SQM: Queued 0 file(s) for upload with pattern: C:\Windows\servicing\	35fa2e8b	SQM: Queued <*> file(s) for upload with pattern: <*>, flags: <*>	['0', 'C:\\Windows\\servicing\\sq
13	2016-09-28	04:30:31	Info	CBS	SQM: Warning: Failed to upload all unsent reports. [HRESULT = 0x800	545e1d3e	SQM: Warning: Failed to upload all unsent reports. [HRESULT = <*> - E_FAIL]	['0x80004005']
14	2016-09-28	04:30:31	Info	CBS	No startup processing required, TrustedInstaller service was not set a	6342ae85	No startup processing required, TrustedInstaller service was not set as autostart, or e	[]
15	2016-09-28	04:30:31	Info	CBS	NonStart: Checking to ensure startup processing was not required.	b756d24a	NonStart: Checking to ensure startup processing was not required.	[]
16	2016-09-28	04:30:31	Info	CSI	00000004 IAdvancedInstallerAwareStore_ResolvePendingTransactio	cfae8b5f	<*> IAdvancedInstallerAwareStore_ResolvePendingTransactions (call <*>) (flags = <*>	['00000004', '1', '00000004', '0', '0:
17	2016-09-28	04:30:31	Info	CSI	00000005 Creating NT transaction (seq 1), objectname [6]"(null)"	30aca5c5	<*> Creating NT transaction (seq <*>), objectname [<*>]"(null)"	['00000005', '1', '6']
18	2016-09-28	04:30:31	Info	CSI	00000006 Created NT transaction (seq 1) result 0x00000000, handle @	9d46d5b0	<*> Created NT transaction (seq <*>) result <*>, handle @<*>	['00000006', '1', '0x00000000', '0x
19	2016-09-28	04:30:31	Info	CSI	00000007@2016/9/27:20:30:31.462 CSI perf trace:	fec03737	<*>@<*>/<*>/<*>:<*>:<*>:<*>.<*> CSI perf trace:	['00000007@2016', '9/27', '20', '3(
20	2016-09-28	04:30:31	Info	CBS	NonStart: Success, startup processing not required as expected.	c7e0e082	NonStart: Success, startup processing not required as expected.	[]
21	2016-09-28	04:30:31	Info	CBS	Startup processing thread terminated normally	0249909f	Startup processing thread terminated normally	[]
22	2016-09-28	04:30:31	Info	CSI	00000008 CSI Store 4991456 (0x00000000004c29e0) initialized	0a16f435	<*> CSI Store <*> (<*>) initialized	['00000008', '4991456', '0x00000(

Σχήμα 6.6: Προβολή δομημένων εγγραφών μετά το parsing

	EventId	EventTemplate	Occurrences
0	38512358	Loaded Servicing Stack <*> with Core: <*>	2
1	ee1bd557	<*>@<*>/<*>/<*>:<*>:<*>:<*>.<*> WcpInitialize (wcp.dll version <*>.<*>.<*>.<*>) called (stack @<*>	6
2	2d8b5399	Ending TrustedInstaller initialization.	2
3	ddb70245	Starting the TrustedInstaller main loop.	2
4	70236f55	TrustedInstaller service starts successfully.	2
5	492269e4	SQM: Initializing online with Windows opt-in: False	2
6	14f494a2	SQM: Cleaning up report files older than <*> days.	2
7	7e5b1617	SQM: Requesting upload of all unsent reports.	2
8	a60ec242	SQM: Failed to start upload with file pattern: <*>, flags: <*> [HRESULT = <*> - E_FAIL]	2

Σχήμα 6.7: Προβολή των παραγόμενων προτύπων μετά το parsing

6.4 Οπτικοποιήσεις και Αναλύσεις

Η ανάλυση δεδομένων καταγραφής δεν είναι πλήρης χωρίς την κατάλληλη οπτικοποίησή τους. Οι πίνακες structured logs παρέχουν μόνο έναν «επίπεδο» τρόπο αναπαράστασης των δεδομένων, ενώ τα γραφήματα επιτρέπουν την αποκάλυψη μοτίβων, ανωμαλιών, συγκεντρώσεων και χρονικών ακολουθιών που διαφορετικά θα παρέμεναν αθέατα. Η παρούσα ενότητα περιγράφει τη στρατηγική σχεδίασης των γραφημάτων για κάθε τύπο log αρχείου, εξηγεί τις επιλογές charting που έγιναν με βάση τη χρησιμότητα για συγκεκριμένα use cases, και συνδέεται με τις αρχές της αποτελεσματικής παρουσίασης πληροφορίας όπως αναλύονται στη σχετική βιβλιογραφία [10].

6.4.1 Χρήση Altair και Vega-Lite για Οπτικοποίηση Δεδομένων

Η επιλογή της βιβλιοθήκης Altair για την παραγωγή γραφημάτων στην παρούσα εφαρμογή βασίζεται στη στενή της σύνδεση με το Vega-Lite, ένα περιγραφικό μοντέλο διαδραστικών γραφημάτων. Το Vega-Lite παρέχει έναν ισχυρό και ελαφρύ μηχανισμό για την κατασκευή διαδραστικών visualizations, χωρίς την ανάγκη συγγραφής πολύπλοκου imperative κώδικα [20].

Declarative προσέγγιση και Visual Encodings

Σε αντίθεση με άλλες βιβλιοθήκες που απαιτούν χαμηλού επιπέδου έλεγχο των γραφικών στοιχείων, το Vega-Lite (και κατά συνέπεια το Altair) επιτρέπει στον προγραμματιστή να καθορίσει τι πρέπει να εμφανιστεί χωρίς να εισέλθει σε λεπτομέρειες για το πώς. Αυτό γίνεται μέσω ενός συνόλου από visual encodings, όπου τα πεδία δεδομένων αντιστοιχίζονται σε οπτικές μεταβλητές (όπως x-άξονας, y-άξονας, χρώμα, μέγεθος, σχήμα). Για παράδειγμα, η χρονοσφραγίδα μπορεί να αντιστοιχιστεί στον άξονα x και το log_level στο χρώμα.

Η declarative φύση αυτής της προσέγγισης διευκολύνει τον συνδυασμό οπτικών στοιχείων και ερωτημάτων, με αποτέλεσμα κώδικα που είναι σαφής, συνοπτικός και επεκτάσιμος. Αυτή η προσέγγιση ευνοεί την παραγωγή πολλαπλών παραλλαγών του ίδιου τύπου γραφήματος, προσαρμοσμένων στις ανάγκες του κάθε dataset [20].

Υποστήριξη Interactivity

Ένα από τα ισχυρότερα χαρακτηριστικά του Vega-Lite είναι η ενσωμάτωση της υποστήριξης για διαδραστικότητα (interactivity). Το σύστημα επιτρέπει την προσθήκη διαδραστικών λειτουργιών χωρίς την ανάγκη εξωτερικού JavaScript κώδικα, ενισχύοντας σημαντικά τη χρηστικότητα των γραφημάτων. Μεταξύ αυτών περιλαμβάνονται οι επιλογές στοιχείων (selections), όπου ο χρήστης μπορεί να επιλέξει σημεία στο γράφημα μέσω κλικ ή επιλογής περιοχής, το δυναμικό φιλτράρισμα (filters), που εφαρμόζονται αυτόματα με βάση τις επιλογές, τα tooltips, δηλαδή η προβολή λεπτομερειών όταν ο χρήστης αιωρείται πάνω από ένα σημείο, καθώς και λειτουργίες μεγέθυνσης και μετακίνησης (zoom και pan) σε ορισμένους τύπους γραφημάτων.

Αυτές οι λειτουργίες υλοποιούνται με λίγες γραμμές κώδικα και χωρίς εξωτερικά JavaScript components, γεγονός που τις καθιστά ιδιαίτερα εύχρηστες σε περιβάλλοντα Python/Streamlit. Επιπλέον, η δυνατότητα σύνθεσης multiple views (concatenation, layering, faceting) καθιστά εύκολη την δημιουργία dashboards με συντονισμένα γραφήματα, γεγονός που συμβαδίζει με τις αρχές coordinated multiple views [20].

Εφαρμογή στην παρούσα εργασία

Στην παρούσα εφαρμογή, η Altair χρησιμοποιείται σε συνδυασμό με το Streamlit για την απόδοση:

- Γραφημάτων κατανομών (bar, line)
- Συσχετίσεων (scatter)
- Συνδυαστικών προβολών (heatmap)
- Επαναλαμβανόμενων μοτίβων (grouped bar, timeline)

Τα δεδομένα παραδίδονται στην Altair σε μορφή DataFrame, και η κατασκευή κάθε visualization βασίζεται στη ρητή δήλωση των πεδίων και των visual encodings. Η επιλογή αυτή αποδείχθηκε ιδανική για την παρούσα εργασία, καθώς επιτρέπει εύκολη προσαρμογή ανά τύπο αρχείου, υποστηρίζει πλήρως διαδραστικές απεικονίσεις και παράγει καθαρό και επεκτάσιμο κώδικα.

Συνολικά, η υιοθέτηση της Vega-Lite προσέγγισης επέτρεψε τον σχεδιασμό οπτικοποιήσεων που όχι μόνο παρουσιάζουν δεδομένα, αλλά υποστηρίζουν και την αναλυτική σκέψη του χρήστη, όπως περιγράφεται και στη βιβλιογραφία για user-centered dashboards [10], [20].

6.4.2 Windows Logs: Επισκόπηση και Οπτικοποίηση

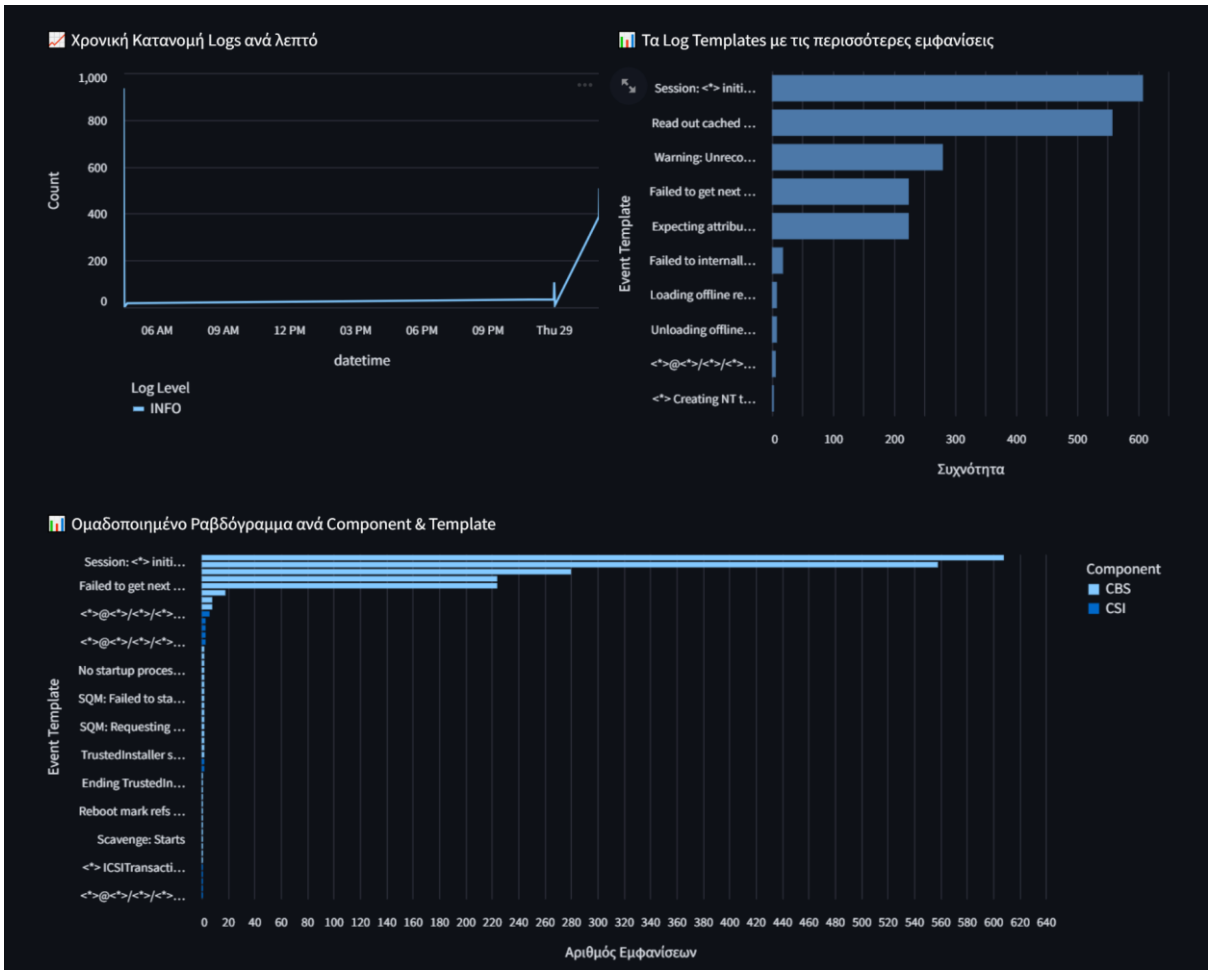
Η ανάλυση των Windows event logs στοχεύει κυρίως στην παρακολούθηση της σταθερότητας του συστήματος και στον εντοπισμό ανεπιθύμητων ή σπάνιων συμβάντων. Η δομή τους περιλαμβάνει συνήθως πεδία όπως datetime, Level, Component, EventTemplate, και Message, που προσφέρονται για grouping και ανάλυση μέσω aggregation. Η στρατηγική της οπτικοποίησης στηρίχθηκε στη χρήση κατανομών (counts) ανά log level, χρονικών εξελίξεων, και ομαδοποίησης templates ανά component.

Line Chart: Χρονική κατανομή συμβάντων ανά log level. Κάθε εγγραφή στα Windows logs συνοδεύεται από επίπεδο σοβαρότητας (INFO, WARNING, ERROR). Οπτικοποιώντας τον όγκο των logs ανά επίπεδο και στιγμή (π.χ. ανά λεπτό), μπορούμε να εντοπίσουμε χρονικά «spikes» ή σιωπές. Η χρήση γραμμικών γραφημάτων (line charts) επιτρέπει την ανάδειξη των διακυμάνσεων στον χρόνο, ενώ η διάκριση χρωμάτων ανά level διευκολύνει την ερμηνεία.

Bar Chart: Τα 10 πιο συχνά templates. Τα πρότυπα (templates) που εξάγονται από τον parser επιτρέπουν ομαδοποίηση παρόμοιων συμβάντων. Εντοπίζοντας τα πιο συχνά templates, μπορούμε να αναγνωρίσουμε λειτουργικά μοτίβα ή επαναλαμβανόμενα σφάλματα. Η χρήση οριζόντιων bar charts επιτρέπει την ανάγνωση μακρών ονομάτων και διευκολύνει τη σύγκριση συχνοτήτων.

Grouped Bar Chart: Εμφανίσεις templates ανά component. Με αυτήν την απεικόνιση γίνεται εφικτός ο εντοπισμός ποιο component προκαλεί ποιο συμβάν. Αυτό το είδος σύνθετου grouping υποστηρίζεται εγγενώς από το Vega-Lite μέσω visual encodings όπως χρώμα και θέσεις στον άξονα [20].

Heatmap: Κατανομή συμβάντων ανά Component και Level ή Template. Το heatmap αξιοποιείται για την αποτύπωση της έντασης εμφάνισης σφαλμάτων ή templates ανά component. Αυτή η matrix-προσέγγιση επιτρέπει την ταχεία επισκόπηση hotspots, δηλαδή modules που παράγουν δυσανάλογο αριθμό logs συγκεκριμένων τύπων. Επιπλέον, διευκολύνει τον εντοπισμό clusters με επαναλαμβανόμενα patterns. Η επιλογή heatmap βασίζεται στη βιβλιογραφική σύσταση ότι τέτοιες απεικονίσεις είναι κατάλληλες όταν συνυπάρχουν δύο κατηγορικά πεδία με πολλές τιμές και στόχος είναι η ανάδειξη σχέσεων συνολικού χαρακτήρα [9], [20].



Σχήμα 6.8: Γραφήματα για Windows Logs

Η επιλογή των παραπάνω γραφημάτων δεν έγινε τυχαία. Οι γραμμές χρόνου υποστηρίζουν την ανίχνευση χρονικών αλληλουχιών, οι ραβδογραμμές διευκολύνουν τη συγκριτική αξιολόγηση συχνοτήτων, και τα grouped bars παρέχουν διπλή ομαδοποίηση (component + template) για root cause analysis. Όπως αναφέρει η βιβλιογραφία του Vega-Lite, κάθε visual encoding πρέπει να καθορίζεται με σκοπό να εξυπηρετεί μία συγκεκριμένη ερώτηση που θέτει ο χρήστης [20].

Αυτά τα γραφήματα λειτουργούν σε επίπεδο στρατηγικής και λειτουργίας: υποστηρίζουν τόσο την υψηλού επιπέδου εποπτεία (monitoring), όσο και τη διερεύνηση ειδικών συμβάντων (troubleshooting) σε επιμέρους services ή subsystems [20].

6.4.3 Linux Logs: Συμπεριφορά και Ερμηνεία

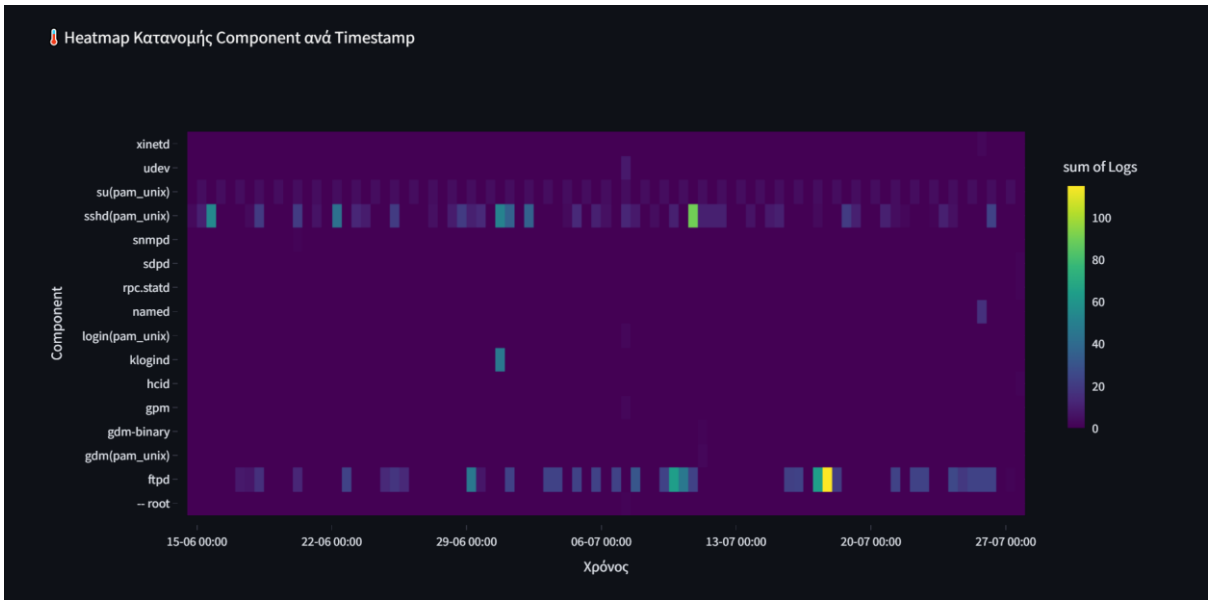
Η ανάλυση των log αρχείων συστημάτων Linux εστιάζει στην παρακολούθηση της συμπεριφοράς υπηρεσιών, διεργασιών και συμβάντων πυρήνα. Τα αρχεία αυτά χαρακτηρίζονται από σταθερή δομή (timestamp, level, component, PID, message) και υψηλή συχνότητα καταγραφής. Η φύση των δεδομένων καθιστά απαραίτητη την χρήση χρονο-εξαρτώμενων και πολυδιάστατων απεικονίσεων για την αναγνώριση λειτουργικών patterns, ανωμαλιών και επαναλαμβανόμενων συμβάντων.

Line Chart: Χρονική κατανομή log events. Χρησιμοποιείται για να εντοπιστούν μοτίβα έντασης logging κατά τη διάρκεια της ημέρας ή λειτουργικών κύκλων. Κάθε γραμμή μπορεί να αντιστοιχεί σε component ή level, αποκαλύπτοντας ρυθμικότητες, spikes ή μη προβλέψιμες μεταβολές.

Grouped Bar Chart: Εμφανίσεις logs ανά Component και Log Level. Η λογική πίσω από αυτό το γράφημα είναι η διπλή ομαδοποίηση: ποια components είναι πιο ενεργά και ποια επίπεδα logs (INFO, ERROR, DEBUG) σχετίζονται με κάθε ένα. Η αντιπαραβολή διαφορετικών bars επιτρέπει συγκριτική αξιολόγηση μεταξύ subsystems, χρήσιμη π.χ. για φορτίο monitoring ή εντοπισμό αστοχιών σε services.

Heatmap: Events *ανά Ώρα* και Component ή PID. Ορίζοντας τη χρονική διάσταση (π.χ. ώρα) στον ένα άξονα και τον Component ή PID στον άλλο, το heatmap επιτρέπει τον εντοπισμό περιοδικών φαινομένων, όπως προγραμματισμένες διεργασίες, watchdogs ή εποχικές υπερφορτώσεις. Τέτοιες οπτικοποιήσεις, σύμφωνα με τη βιβλιογραφία, προσφέρουν καλή αναγνωσιμότητα όταν ο στόχος είναι η αναγνώριση χρονικών clusters ή η προσέγγιση συστημάτων ως οντότητες με χρονική συμπεριφορά [20].

Scatter Plot: Timestamps vs PID ή Component. Στο γράφημα αυτό, κάθε σημείο αντιστοιχεί σε ένα log entry. Ο οριζόντιος άξονας αποδίδει τον χρόνο, ενώ ο κάθετος το PID ή το Component. Η χρωματική διαφοροποίηση ανά log level ή event ID επιτρέπει την άμεση αναγνώριση σφαλμάτων ή σημαντικών γεγονότων. Η χρήση scatter ενδείκνυται όταν υπάρχει ανάγκη ανίχνευσης χρονικών πυκνώσεων, gaps ή επαναλαμβανόμενων ακολουθιών [20].



Σχήμα 6.9: Heatmap για Linux Logs

Η επιλογή των γραφημάτων έγινε με βάση την ανάγκη να υποστηριχθούν τα εξής use cases:

- Παρακολούθηση διεργασιών: μέσω scatter με PID/time, εντοπίζονται gaps, ταυτόχρονες ενεργοποιήσεις, pattern συμπεριφοράς.
- Διάγνωση φορτίου: grouped bar charts αποκαλύπτουν συνεισφορά κάθε component.
- Καταγραφή καθημερινών κύκλων: heatmaps αποκαλύπτουν ωριαία κατανομή συμβάντων ανά υποσύστημα.
- Αξιολόγηση σταθερότητας: line charts υποδεικνύουν ανωμαλίες ή περιοχές υψηλής συχνότητας σφαλμάτων.

Τα παραπάνω ενισχύουν τον ρόλο των οπτικοποιήσεων ως εργαλείο διαγνωστικής κατανόησης και όχι απλής απεικόνισης. Όπως επισημαίνει και η σχετική βιβλιογραφία, οι visual encodings (π.χ. θέση,

χρώμα, συχνότητα) πρέπει να επιλέγονται με τρόπο που να υποστηρίζουν τις ενέργειες του αναλυτή, είτε πρόκειται για high-level εποπτεία είτε για εστιασμένο troubleshooting [20].

6.4.4 Mac Logs: Γραφήματα για Περιβαλλοντική Παρακολούθηση

Τα logs που συλλέγονται σε περιβάλλοντα macOS περιέχουν συνήθως πληροφορίες που σχετίζονται με δραστηριότητες του χρήστη, εγγραφές συστήματος και μηνύματα διεργασιών. Η εστίαση εδώ δεν είναι τόσο στην ανίχνευση απειλών ή στατιστικών εξάρσεων, όσο στην παρακολούθηση και κατανόηση της καθημερινής λειτουργίας των εφαρμογών και του λειτουργικού περιβάλλοντος.

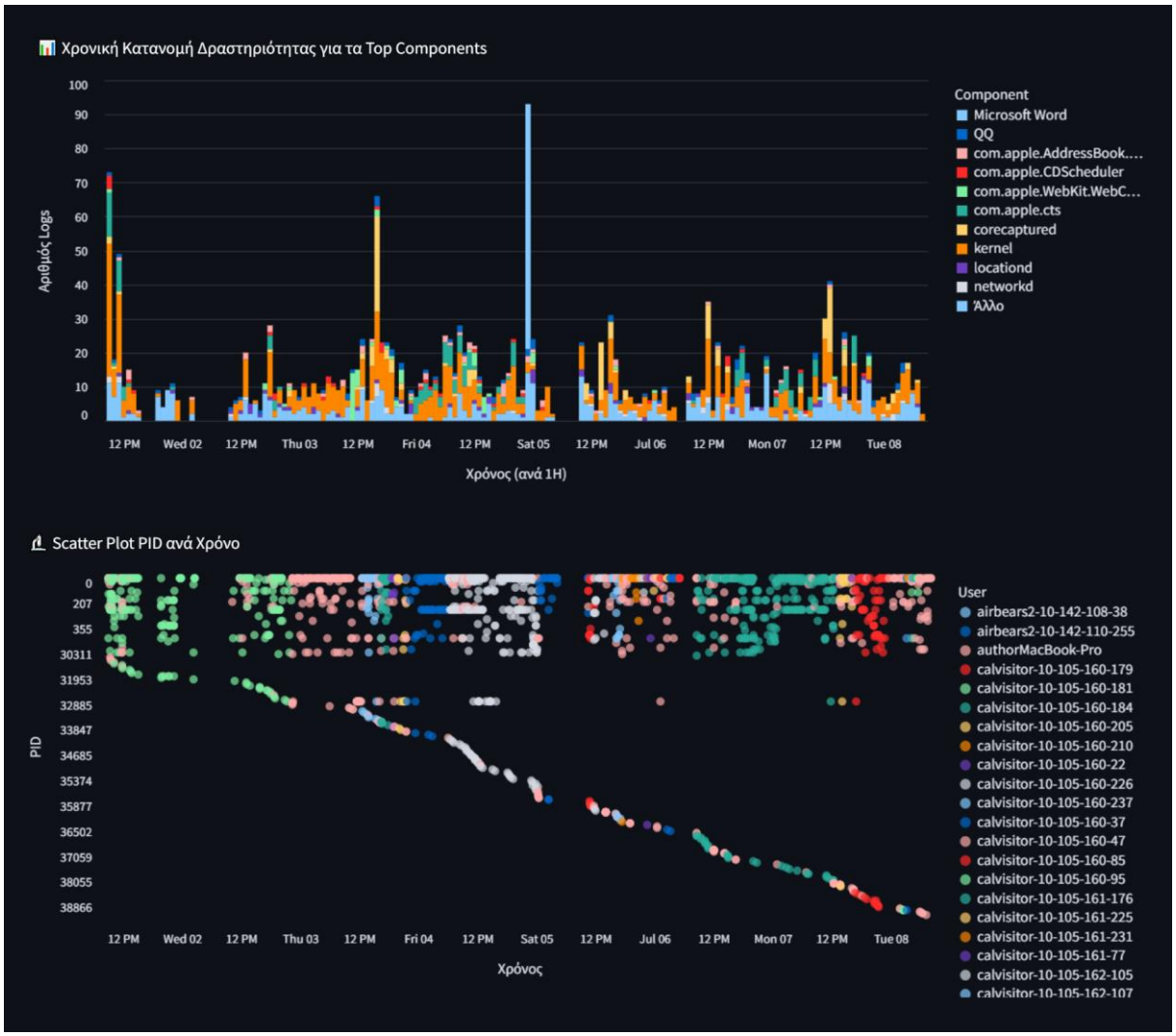
Σύμφωνα με τις απαιτήσεις χρήστη σε τέτοια συστήματα (π.χ. debugging, user profiling, performance context), τα γραφήματα που υλοποιήθηκαν επικεντρώνονται στην παρουσίαση κατανομών, χρονικής εξέλιξης και συσχετισμών μεταξύ PID, component και χρόνου.

Line Chart: Εξέλιξη δραστηριότητας στον χρόνο. Χρησιμοποιείται για την αποτύπωση της έντασης logging (πλήθος εγγραφών ανά χρονική μονάδα), δημιουργώντας μια καμπύλη ρυθμού δραστηριότητας.

Bar Chart: Πιο συχνά components. Το γράφημα αυτό δείχνει ποια components (δηλαδή εφαρμογές ή υποσυστήματα του macOS) παράγουν το μεγαλύτερο πλήθος log entries. Με αυτόν τον τρόπο, εντοπίζονται υπηρεσίες που ενδέχεται να είναι υπερενεργές, να δημιουργούν debugging noise ή να παρουσιάζουν λειτουργικά ζητήματα.

Stacked Bar Chart: Χρονική Κατανομή Δραστηριότητας για τα Top Components. Για την αποτύπωση της συνολικής δραστηριότητας στον χρόνο και την ανάλυση των πιο ενεργών υποσυστημάτων, χρησιμοποιείται stacked bar chart με άξονα x τον χρόνο και άξονα y τον αριθμό logs. Η ομαδοποίηση στον χρονικό άξονα δεν είναι σταθερή αλλά καθορίζεται δυναμικά, με βάση τη χρονική απόσταση μεταξύ της πρώτης και της τελευταίας εγγραφής (datetime), ώστε να διατηρείται καθαρή και ευανάγνωστη οπτικοποίηση τόσο σε μικρά όσο και σε μεγάλα χρονικά παράθυρα. Οι μπάρες χωρίζονται χρωματικά ανά component, αλλά με φιλτράρισμα μόνο στα 10 συχνότερα components, ενώ όλα τα υπόλοιπα συγκεντρώνονται σε μία κοινή κατηγορία "Άλλο". Η προσέγγιση αυτή επιτρέπει την εστίαση στους πιο σημαντικούς contributors χωρίς να θυσιάζεται η γενική εικόνα. Η τεχνική του "Top-N + Others" αποτελεί προτεινόμενη πρακτική για τη διατήρηση της αναγνωσιμότητας σε περιβάλλοντα με πολυκατηγορικά δεδομένα [20] και είναι σύμφωνη με τις αρχές εστιασμένης εποπτείας που προτείνονται στον σχεδιασμό λειτουργικών dashboards [10].

Scatter Plot: Χρήστες και PID στον χρόνο. Κάθε σημείο δείχνει μια εγγραφή με άξονες τον timestamp (x) και τον PID (y), ενώ το user κωδικοποιείται με χρώμα. Το γράφημα αυτό επιτρέπει την οπτική παρακολούθηση της συσχέτισης μεταξύ ενεργειών χρηστών και διεργασιών, χρήσιμο για debugging, ανάλυση απόδοσης ή εντοπισμό μη αναμενόμενων συσχετισμών.



Σχήμα 6.10: Stacked Bar Chart και Scatter Plot για Mac Logs

Η στρατηγική οπτικοποίησης για τα Mac logs βασίστηκε σε ανάγκες που συνδέονται περισσότερο με τη λειτουργική κατανόηση της συμπεριφοράς του χρήστη και του περιβάλλοντος εκτέλεσης, παρά με την ανίχνευση κρίσιμων σφαλμάτων ή απειλών. Ειδικότερα, η καταγραφή του ρυθμού παραγωγής logs από διαφορετικά components επιτρέπει την καθημερινή παρακολούθηση περιβαλλοντικής συμπεριφοράς και την ανίχνευση patterns υψηλής ή χαμηλής δραστηριότητας, κάτι που τεκμηριώνεται ως σημαντικό εργαλείο εργονομικής διάγνωσης σε περιβάλλοντα παρακολούθησης συστημάτων. Παράλληλα, η χαρτογράφηση διεργασιών (μέσω PID) και η σύνδεση με χρονικές διαστάσεις επιτρέπει την εκτίμηση του φορτίου που προκαλούν επιμέρους εφαρμογές. Όταν δε οι χρήστες και οι διαδικασίες κωδικοποιούνται σε scatter plots, ο αναλυτής αποκτά άμεση εικόνα της συσχέτισης συμπεριφοράς χρήστη/χρόνου, κάτι που σχετίζεται με την έννοια του temporal usage profiling και προτείνεται σε dashboards παρακολούθησης χρήσης συστημάτων. Τέλος, η χρήση εργαλείων που αποτυπώνουν τον χρονικό ρυθμό παραγωγής logs και τη συνεισφορά διαφορετικών components μπορεί να οδηγήσει στον εντοπισμό ανεπιθύμητης υπερπαραγωγής (over-logging) ή σε χαμηλού ρυθμού latency patterns, φαινόμενα που αναφέρονται και στη σχετική βιβλιογραφία ως συχνά αντικείμενα διάγνωσης με βάση logs [7].

Η επιλογή των γραφημάτων ενισχύει την κατανόηση της λειτουργίας σε επίπεδο desktop και εστιάζει σε ερωτήματα όπως «ποιος χρήστης κάνει τι;», «πότε ενεργοποιούνται οι περισσότερες διεργασίες;», «ποιες εφαρμογές δημιουργούν τα περισσότερα logs;». Όλα χρήσιμα για system tuning ή troubleshooting.

Τέτοιες μορφές συνδυαστικών encodings (όπως color-by-user σε scatter και layering time) επιτρέπουν στον αναλυτή να βλέπει κατανομή, συσχέτιση και ένταση ταυτόχρονα [20]. Παράλληλα, από τη σκοπιά του dashboard design, αυτά τα γραφήματα συνδυάζουν λειτουργική και τακτική εποπτεία, παρουσιάζουν trends, αποτυπώνουν φορτία, και διευκολύνουν την παρέμβαση σε περίπτωση μη φυσιολογικής συμπεριφοράς [7], [10].

6.4.5 Suricata Logs: Ανάλυση Δικτυακής Κίνησης

Τα αρχεία καταγραφής του συστήματος Suricata έχουν ως στόχο την παρακολούθηση και ανάλυση της δικτυακής δραστηριότητας με σκοπό την αναγνώριση εισβολών, κακόβουλων ενεργειών και άλλων συμβάντων ασφάλειας. Η μορφή τους είναι πιο σύνθετη από άλλους τύπους logs και περιλαμβάνει πεδία όπως EventType, PriorityValue, ClassDescription, Protocol, SrcPort, DstPort, timestamp, SrcIP και DstIP. Αυτή η πληροφοριακή πυκνότητα απαιτεί τη χρήση διαφορετικών τύπων γραφημάτων για την αποκάλυψη συμπεριφορών και συσχετισμών.

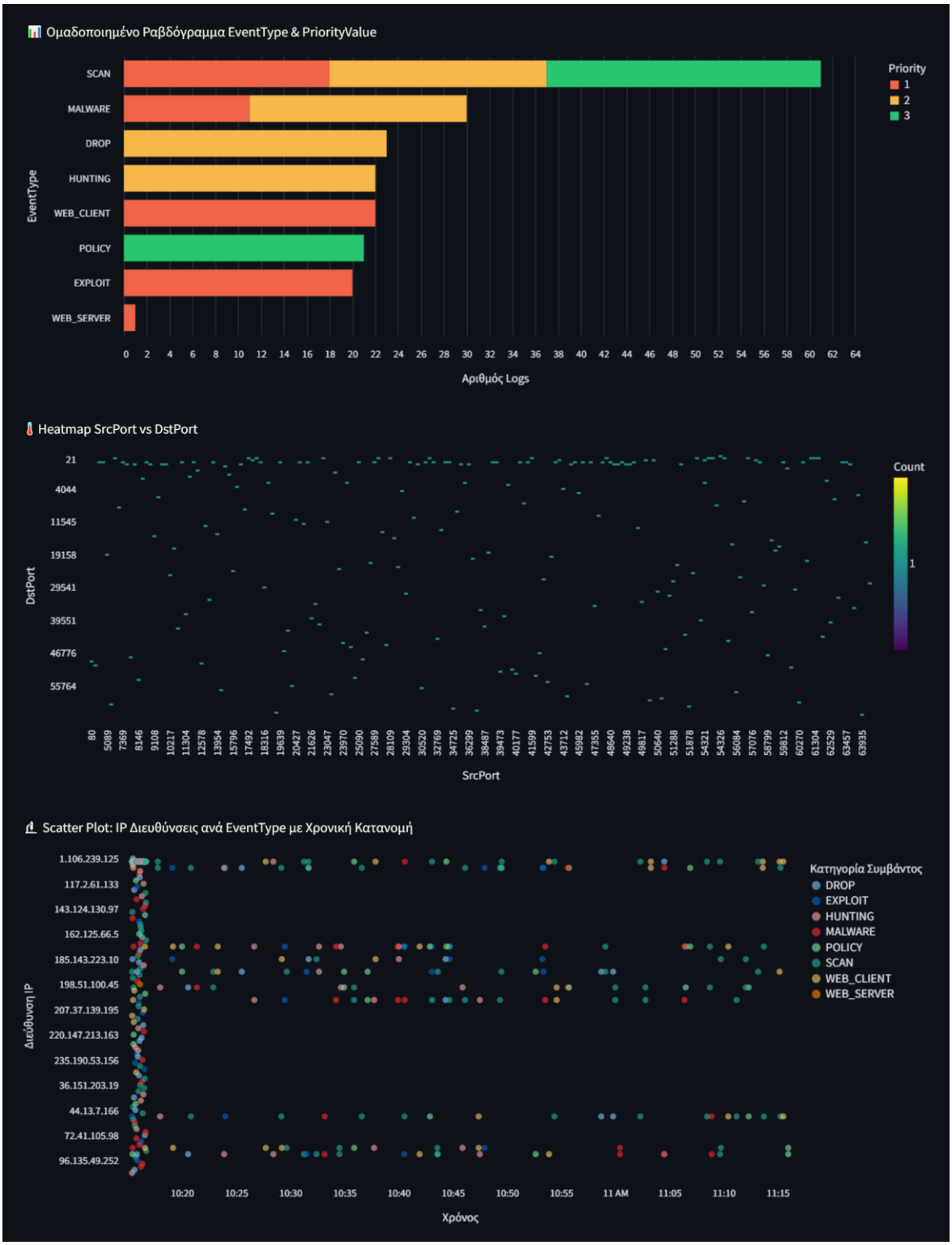
Grouped Bar Chart: Συνδυασμός Event Type και Priority Value. Αυτό το γράφημα συνδυάζει την κατηγορία συμβάντος (EventType) με το πεδίο PriorityValue, το οποίο αποδίδει τη σοβαρότητα του γεγονότος με αριθμητική κλίμακα (π.χ. 1 για υψηλή, 2 για μεσαία, 3 για χαμηλή προτεραιότητα). Κάθε κατηγορία EventType (όπως alert, dns, flow, http, ssh, tls) απεικονίζεται με ένα σύνολο από μπάρες που διαχωρίζονται ανά επίπεδο προτεραιότητας, επιτρέποντας στον αναλυτή όχι μόνο να δει πόσο συχνά εμφανίστηκε κάθε τύπος, αλλά και να αξιολογήσει τη σχετική του σημασία.

Με αυτόν τον τρόπο, το grouped bar chart καλύπτει πλήρως και τη λειτουργία ενός απλού bar chart, το οποίο θα παρουσίαζε μόνο τη συχνότητα εμφάνισης των διαφορετικών EventType, χωρίς να αποκαλύπτει την επικινδυνότητά τους. Αντί να χρησιμοποιούνται δύο ανεξάρτητα γραφήματα για την ίδια μεταβλητή (όπως τα event types), επιλέχθηκε η σύνθεση σε μία μόνο προβολή που συνδυάζει συχνότητα και προτεραιότητα. Αυτή η προσέγγιση ανταποκρίνεται στις αρχές σχεδίασης του Vega-Lite, σύμφωνα με τις οποίες η ενσωμάτωση πολλαπλών χαρακτηριστικών σε ενιαία οπτικοποίηση, μέσω grouping, layering ή encoding, βελτιώνει την αναγνωσιμότητα και αποφεύγει την επανάληψη επικαλυπτόμενης πληροφορίας [20].

Η επιλογή αυτής της μορφής επιτρέπει την ταυτοποίηση συχνών αλλά χαμηλής προτεραιότητας συμβάντων, που μπορεί να αποτελούν ψευδείς συναγερμούς ή άνευ σημασίας δραστηριότητα, αλλά και σπάνιων αλλά σοβαρών περιστατικών που απαιτούν άμεση προσοχή. Το grouped bar chart αξιοποιεί τη δυνατότητα του Vega-Lite να συνδυάζει διαφορετικά κατηγορικά πεδία σε έναν ενιαίο χώρο απεικόνισης, με visual encodings όπως το χρώμα και η ομαδοποίηση για να ενισχύσει την ερμηνευσιμότητα και τη συγκριτική ανάλυση [20].

Heatmap: Source Port vs Destination Port. Χρησιμοποιείται για τον εντοπισμό προτύπων στην κυκλοφορία, μέσω της ανάλυσης των θυρών πηγής και προορισμού. Το heatmap είναι ιδανικό για αυτόν τον σκοπό, καθώς επιτρέπει την αναγνώριση περιοχών με υψηλή πυκνότητα συμβάντων, κάτι που μπορεί να υποδηλώνει σάρωση θυρών, brute force ή κακόβουλη δραστηριότητα. Η χρήση του συγκεκριμένου encoding θεωρείται κατάλληλη για τη σύγκριση πλήθους μεταξύ δύο διακριτών μεταβλητών με υψηλό καρτεσιανό πλήθος [20] και έχει επιβεβαιωθεί ως χρήσιμο εργαλείο σε αναλύσεις δικτυακών logs [9].

Scatter Plot: IP-based Events με Χρονική Κατανομή. Για την κατανόηση της κατανομής δικτυακών συμβάντων στον χρόνο και την αναγνώριση πιθανών ανωμαλιών, χρησιμοποιείται scatter plot όπου ο οριζόντιος άξονας αντιστοιχεί στον timestamp και ο κάθετος σε διευθύνσεις IP (SrcIP ή DstIP). Κάθε σημείο στο γράφημα αντιπροσωπεύει ένα καταγεγραμμένο συμβάν, ενώ με χρήση χρώματος μπορεί να απεικονιστεί και το EventType, προσφέροντας πρόσθετο διαχωρισμό. Η οπτικοποίηση αυτή διευκολύνει τον εντοπισμό χρονικών συγκεντρώσεων συμβάντων, διαδοχικών αιτήσεων από συγκεκριμένες IPs ή απότομων αλλαγών στη συμπεριφορά ενός client ή service. Η αξιοποίηση της χρονικής διάστασης σε συνδυασμό με κατηγορική ή ποσοτική κωδικοποίηση είναι θεμελιώδης αρχή στον σχεδιασμό διαδραστικών γραφημάτων και ενισχύεται από τα visual encodings του Vega-Lite [20].



Σχήμα 6.11: Γραφήματα για Suricata Logs

Η λογική πίσω από τη χρήση των παραπάνω γραφημάτων εστιάζει στην πολυδιάστατη ανάλυση του traffic. Τα grouped bars αναδεικνύουν ποιοτικά χαρακτηριστικά όπως η προτεραιότητα (PriorityValue) μέσω διπλής κατηγοριοποίησης, τα heatmaps χαρτογραφούν επιθέσεις σε επίπεδο θυρών, κάνοντας ορατά τα patterns σε μεγάλους πίνακες τιμών, και τα scatter plots παρέχουν χρονικό context και στόχευση σε IPs, ιδιαίτερα όταν απαιτείται κατανόηση του sequence δραστηριότητας.

Αυτή η προσέγγιση ενισχύεται θεωρητικά από τη χρήση συνδυασμένων visual encodings και layered ή faceted γραφημάτων, χαρακτηριστικά του Vega-Lite [20].

Τα γραφήματα αυτά υποστηρίζουν network monitoring σε βάθος χρόνου, ανάλυση απειλών και απόκριση περιστατικών, εντοπισμό ανωμαλιών και suspicious patterns και διαχωρισμό νόμιμης και κακόβουλης δραστηριότητας

Με βάση τις αρχές του σχεδιασμού διαδραστικών visualizations, κάθε visual πρέπει να απαντά σε μια ερώτηση που θα έθετε ο αναλυτής [20]. Στο συγκεκριμένο dashboard, η οπτικοποίηση έχει σχεδιαστεί ώστε να παρέχει μεγάλη πυκνότητα πληροφορίας με σαφήνεια, επιτρέποντας την άμεση επέμβαση σε περιστατικά ή την επισκόπηση για reporting.

6.4.6 Παρατηρήσεις και Επιλογή Γραφημάτων

Η επιλογή των γραφημάτων στην παρούσα εφαρμογή έγινε με βάση τον συνδυασμό τριών παραγόντων: (α) τον τύπο των δεδομένων και τη δομή των logs, (β) τις αναλυτικές ανάγκες του χρήστη ανά περίπτωση, και (γ) τις συστάσεις της βιβλιογραφίας για κατάλληλα visual encodings. Η υλοποίηση ενσωματώνει διαφορετικά είδη διαγραμμάτων – από απλά bar charts έως heatmaps και scatter plots – ώστε να εξυπηρετεί τόσο την εποπτεία όσο και τη διάγνωση.

Ο πυρήνας μιας αποτελεσματικής οπτικοποίησης είναι η σωστή αντιστοίχιση μεταξύ των χαρακτηριστικών των δεδομένων και των γραφικών παραμέτρων (όπως θέση, χρώμα, μέγεθος, opacity, κ.λπ.). Για παράδειγμα, οι ποσοτικές μεταβλητές αποτυπώνονται καλύτερα μέσω αξόνων θέσης ή περιοχής, ενώ οι κατηγορικές με χρώμα ή φιλτράρισμα. Αυτή η λογική εφαρμόστηκε σε όλα τα dashboards της εφαρμογής, διατηρώντας συνοχή και συνέπεια μεταξύ των διαγραμμάτων [20].

Παράλληλα, οι διαφορετικοί χρήστες έχουν διαφορετικές ανάγκες. Άλλοι ενδιαφέρονται για εποπτική παρακολούθηση (monitoring), άλλοι για γρήγορη αντίδραση σε ανωμαλίες, και άλλοι για in-depth ανάλυση. Κατά συνέπεια, τα γραφήματα πρέπει να επιλέγονται όχι μόνο με βάση τα δεδομένα, αλλά και με βάση τον σκοπό: π.χ. η χρήση timeline και line charts για χρονική εποπτεία, bar charts για κατανομές, scatter για αναγνώριση patterns, και heatmaps για πολλαπλά κατηγορικά πεδία και αναζήτηση clustering [10].

Επιπλέον, η παρατήρηση across dashboards δείχνει ότι ορισμένες επιλογές λειτουργούν ως «πυρήνες σχεδίασης»:

- Η χρονική διάσταση χρησιμοποιείται συστηματικά στον οριζόντιο άξονα για τη διατήρηση του context.
- Το log level ή το Priority χαρτογραφείται σε χρώμα, δίνοντας έμφαση στην κρίσιμη πληροφορία.
- Το component και το template λειτουργούν ως κατηγορίες grouping, προσφέροντας δομή στα δεδομένα.

Τέλος, η εφαρμογή ενσωματώνει πλήρως τη λογική του task-driven visualization: κάθε γράφημα απαντά σε μια συγκεκριμένη ερώτηση ή ανάγκη. Δεν παρουσιάζεται μόνο το «τι έγινε», αλλά και το «πού», «πότε», «με ποιον τρόπο» και «με ποια ένταση». Αυτή η προσέγγιση ενισχύει όχι μόνο την αναγνωσιμότητα αλλά και την ερμηνευσιμότητα, κάτι που υπογραμμίζεται έντονα στη σύγχρονη βιβλιογραφία για την ανάλυση logs και το design analytics συστημάτων [10], [20].

6.5 Benchmarking και Βελτιστοποίηση Ρυθμίσεων

6.5.1 Σκοπός και Μεθοδολογία Αξιολόγησης

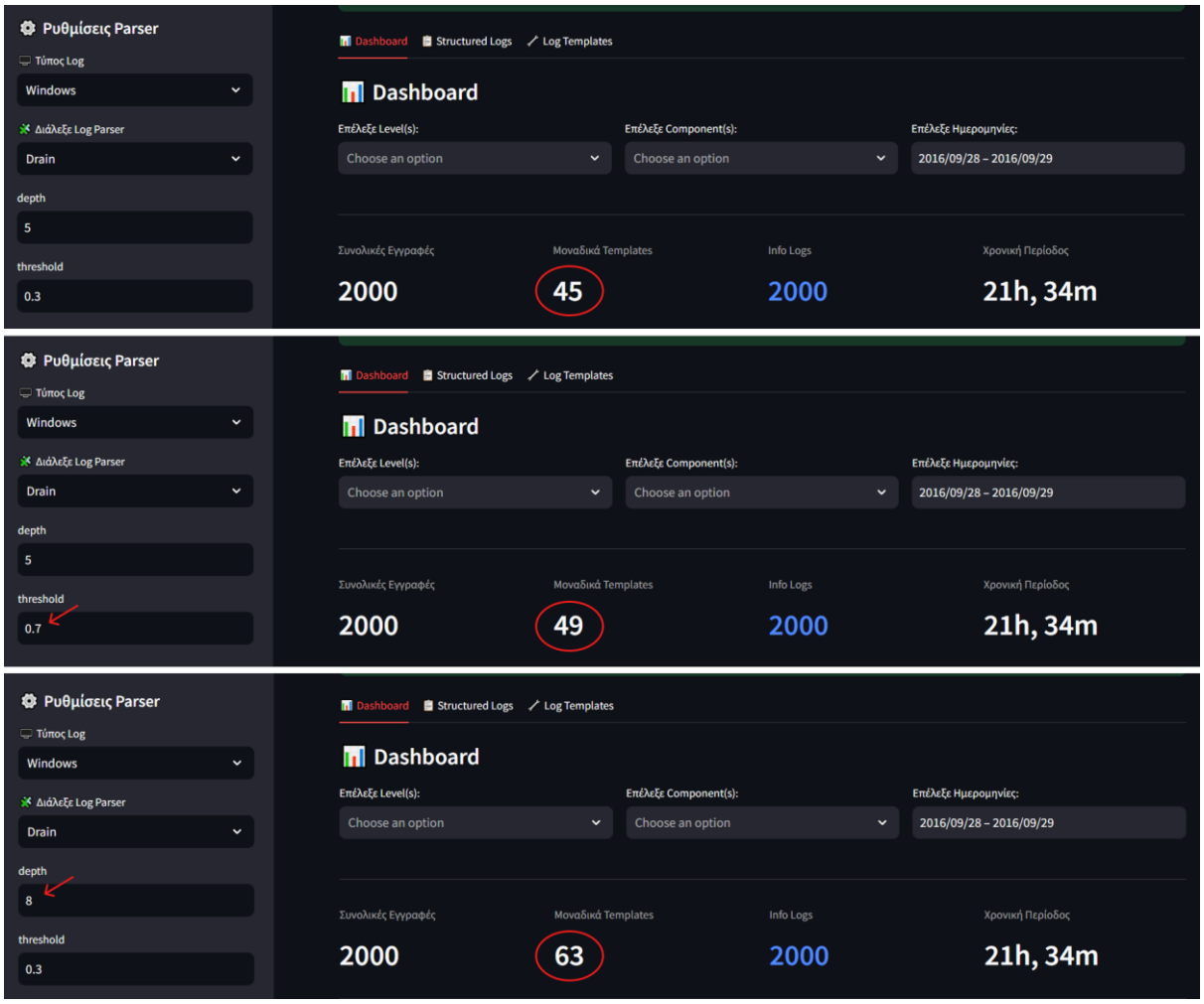
Η διαδικασία benchmarking πραγματοποιήθηκε με σκοπό τη συγκριτική αξιολόγηση της συμπεριφοράς διαφορετικών log parsing αλγορίθμων (Drain, Spell, IPLoM, LogCluster) σε αρχεία καταγραφής τύπου Suricata, χρησιμοποιώντας ground truth templates για τη μέτρηση της ακρίβειας. Βασικός στόχος ήταν η παρατήρηση του πώς μεταβάλλεται η απόδοση κάθε αλγορίθμου όταν τροποποιούνται συγκεκριμένες παράμετροι ρύθμισης, καθώς και η επιλογή των βέλτιστων default ρυθμίσεων για την τελική εφαρμογή.

Η προσέγγιση ευθυγραμμίζεται με πρακτικές αξιολόγησης που συναντώνται στη σχετική βιβλιογραφία, όπου μετρικές όπως template accuracy, χρόνος εκτέλεσης και αριθμός παραχθέντων προτύπων (templates) αποτελούν βασικούς δείκτες απόδοσης [7].

6.5.2 Παρατηρήσεις για τη Σταθερότητα των Αποτελεσμάτων

Ένα κρίσιμο εύρημα, τόσο από τη βιβλιογραφία όσο και από την πειραματική παρατήρηση, είναι ότι η εκτέλεση του ίδιου parser στα ίδια logs με διαφορετικές παραμέτρους μπορεί να οδηγήσει σε δραστικά διαφορετικά αποτελέσματα, επηρεάζοντας τόσο την ακρίβεια όσο και τον αριθμό των παραγόμενων templates. Χαρακτηριστικό παράδειγμα αποτελεί ο αλγόριθμος Drain, στον οποίο η παράμετρος threshold καθορίζει το ελάχιστο ποσοστό ομοιότητας (simSeq) που πρέπει να έχει ένα νέο log message με ένα υπάρχον template για να συσχετιστούν. Όπως αναφέρεται στη βιβλιογραφία, όταν το threshold αυξάνεται, τα κριτήρια γίνονται αυστηρότερα, οδηγώντας σε περισσότερα templates λόγω χαμηλότερης ανοχής ομαδοποίησης (oversegmentation), ενώ σε χαμηλές τιμές αυξάνεται ο κίνδυνος υπερβολικής γενίκευσης (underspecification), καθώς logs που δεν σχετίζονται ουσιαστικά συσχετίζονται στο ίδιο template [17].

Αντίστοιχα, η παράμετρος depth ελέγχει τον αριθμό των θέσεων tokens που λαμβάνονται υπόψη για τη διάκριση διαφορετικών branches στο parsing tree. Μεγαλύτερη τιμή depth επιτρέπει πιο λεπτομερή διάκριση, αυξάνοντας την ακρίβεια και τον αριθμό των παραγόμενων templates, αλλά και επιβαρύνοντας τον χρόνο εκτέλεσης λόγω βαθύτερης traversal λογικής [17]



Σχήμα 6.12: Μεταβολή του πλήθους Templates για διαφορετικά depth και threshold

Η εύρεση σταθερών και αποτελεσματικών παραμέτρων είναι αναγκαία, μάλιστα παρουσιάζονται μεγάλες αποκλίσεις στην απόδοση parsing ανάλογα με την παραμετροποίηση [7]. Αυτό υποστηρίζεται επίσης από την τεκμηρίωση των Drain και Spell [5], [17], και οι προτεινόμενες προεπιλεγμένες τιμές του LogPai είναι το αποτέλεσμα προσεκτικών δοκιμών σε benchmarking datasets [16].

6.5.3 Πειραματική Προσέγγιση για Suricata Logs

Για την προσαρμογή των παραμέτρων parsing στα Suricata logs, αναπτύχθηκε ένα ειδικό script benchmarking (benchmarkSuricata.py), το οποίο εκτελεί επαναληπτικά κάθε parser με διαφορετικούς συνδυασμούς τιμών και συγκρίνει τα παραγόμενα templates με ground truth templates αποθηκευμένα σε CSV. Για κάθε εκτέλεση υπολογίζεται η ακρίβεια (ποσοστό σωστής ταύτισης), ο χρόνος εκτέλεσης και ο αριθμός των παραγόμενων προτύπων.

Το benchmark script υποστηρίζει τις παρακάτω παραμέτρους ανά parser:

- Drain: depth, st (threshold)
- Spell: tau (threshold)
- IPLoM: CT (Coverage Threshold), lowerBound
- LogCluster: rsupport (support ratio για word frequency)

Σύμφωνα με τη δοκιμή, παρατηρήθηκε ότι ο Drain εμφάνισε τη βέλτιστη ακρίβεια με depth=5 και threshold=0.7, με σημαντικές μεταβολές όταν τροποποιούνται και τα δύο. Ενδεικτικό της ευαισθησίας του parser στις ρυθμίσεις δέντρου και σύγκρισης tokens. Ο Spell παρουσίασε σταθερότερη συμπεριφορά ως προς το threshold, αλλά μικρότερη ακρίβεια συνολικά. Ο IPLoM επηρεάστηκε περισσότερο από το CT παρά από το lowerBound, με τις μικρές τιμές να παράγουν μικρότερο αριθμό templates αλλά μεγαλύτερη ακρίβεια. Τέλος, ο LogCluster παρουσίασε σταθερότητα στις μετρήσεις ανεξαρτήτως της τιμής rsupport, με πανομοιότυπα αποτελέσματα ως προς την ακρίβεια και τον αριθμό templates σε όλες τις δοκιμές. Αυτό υποδηλώνει ότι, τουλάχιστον για δεδομένα Suricata, αυτός ο parser είναι λιγότερο ευαίσθητος στην παραμετροποίηση, πράγμα που έρχεται σε αντίθεση με αποτελέσματα από άλλα datasets [18].

6.5.4 Συμπεράσματα και Default Επιλογές

Με βάση την παραπάνω ανάλυση, επιλέχθηκαν για τα Suricata logs οι παρακάτω default τιμές:

- Drain: depth = 5, threshold = 0.7
- Spell: threshold = 0.7
- IPLoM: CT = 0.3, lowerBound = 0.25
- LogCluster: rsupport = 1

Δεδομένου ότι οι προτεινόμενες προεπιλεγμένες τιμές από το LogPai έχουν χρησιμοποιηθεί εκτενώς στη βιβλιογραφία για benchmarking σε συγκρίσιμα σύνολα δεδομένων, υιοθετήθηκαν και για τους άλλους τύπους logs (Windows, Linux και Mac) [16]. Αυτές οι τιμές προσφέρουν μια δοκιμασμένη και αξιόπιστη βάση για σταθερότητα και απόδοση σε διάφορα περιβάλλοντα, και έχουν εξελιχθεί σε ένα baseline για μελλοντικές συγκρίσεις στην έρευνα parsing.

Οι τελικές επιλογές default παραμέτρων αντικατοπτρίζουν μια ισορροπία μεταξύ υψηλής ακρίβειας, αποδεκτών αριθμών προτύπων και ικανοποιητικού χρόνου εκτέλεσης, ακολουθώντας τις αρχές πολυκριτηριακής βελτιστοποίησης που συναντώνται σε parsing frameworks όπως το MoLFI [4].

6.6 Υλοποίηση, Παραδείγματα και Τεχνικές Επισημάνσεις

6.6.1 Οργάνωση και Αρχές Κώδικα

Ο πηγαίος κώδικας της εφαρμογής είναι οργανωμένος σε ξεχωριστά αρχεία ανά τύπο log, με σταθερή και αναγνώσιμη δομή. Το αρχείο main_dashboard.py αποτελεί το κεντρικό σημείο εκκίνησης της εφαρμογής και περιλαμβάνει το κύριο layout του UI, τον μηχανισμό επιλογής αρχείων από τον χρήστη, τα φίλτρα, τα tabs πλοήγησης, καθώς και την εκτέλεση της κατάλληλης συνάρτησης ανά τύπο log μέσω του DASHBOARD_MAP. Επιπλέον, περιέχει τον ορισμό των sidebar widgets (file uploader, επιλογή parser και παραμέτρων) και τη λογική για την εναλλαγή μεταξύ των tabs Dashboard, Parsed Logs και Templates.

Τα αρχεία windows_logs.py, linux_logs.py, mac_logs.py και suricata_logs.py υλοποιούν το dashboard για κάθε αντίστοιχο τύπο αρχείου log. Κάθε αρχείο περιέχει τη συνάρτηση show_dashboard(), τα φίλτρα και widgets που είναι ειδικά για τον εκάστοτε τύπο αρχείου, τη σύνδεση με τα δομημένα δεδομένα που επιστρέφει η συνάρτηση parsing, και την παρουσίαση των γραφημάτων και των σχετικών στατιστικών.

Το βοηθητικό αρχείο log_utils.py συγκεντρώνει την κοινή λογική, τις default παραμέτρους και τους μηχανισμούς parsing. Πιο συγκεκριμένα:

- Το PARSER_FACTORY ενοποιεί την αρχική δημιουργία όλων των parsers, αποφεύγοντας επαναλαμβανόμενα if blocks.
- Οι default παράμετροι για κάθε parser και τύπο log ορίζονται στο PARSER_DEFAULTS και ανακτώνται δυναμικά από το UI.
- Όλα τα regex patterns που χρησιμοποιούνται για το log parsing συγκεντρώνονται στο REGEX_PATTERNS, οργανωμένα ανά τύπο log.
- Η βοηθητική συνάρτηση run_parser() συγκεντρώνει την κοινή λογική parsing (αποθήκευση αρχείων, εκτέλεση parser, ανάγνωση αποτελεσμάτων), καταργώντας blocks 50+ γραμμών σε κάθε αρχείο.
- Η μετατροπή των παραμέτρων σε κατάλληλο τύπο (int, float) γίνεται δυναμικά μέσω της δομής PARSER_PARAM_TYPES, αποφεύγοντας χειροκίνητο try/except σε κάθε module.
- Η προσθήκη του συγκεκριμένου αρχείου οδήγησε σε σημαντική μείωση γραμμών κώδικα, βελτίωσε την αναγνωσιμότητα και την επεκτασιμότητα σε νέους parsers ή log types.

Όταν ο χρήστης επιλέξει αρχείο και πατήσει το κουμπί «Εκκίνηση Ανάλυσης», ενεργοποιείται η λειτουργία parsing. Πιο συγκεκριμένα, ορίζεται η τιμή session_state["parse_button_clicked"] = True και ενεργοποιείται η λογική parsing στον κεντρικό βρόχο του main_dashboard.py. Στη συνέχεια, καλείται η συνάρτηση run_parser() από το log_utils.py (Παράρτημα Β), η οποία αναλαμβάνει τη δημιουργία προσωρινού φακέλου, την αποθήκευση του αρχείου, την επιλογή του κατάλληλου parser βάσει PARSER_FACTORY (Παράρτημα C), και την εκτέλεση parsing με τις τιμές παραμέτρων που επέλεξε ο χρήστης. Το αποτέλεσμα είναι δύο αρχεία CSV (structured logs και templates), τα οποία επιστρέφονται στο dashboard για προβολή, φιλτράρισμα και οπτικοποίηση.

6.6.2 Δημιουργία Συστατικών της Εφαρμογής

Η λειτουργικότητα της εφαρμογής βασίζεται στη δημιουργία επαναχρησιμοποιήσιμων components και δομών, με στόχο τη modular ανάπτυξη και ευκολία επέκτασης. Παρακάτω παρουσιάζονται τα βασικά στοιχεία που κατασκευάστηκαν:

- Sidebar με επιλογές εισόδου: Υλοποιήθηκε στο main_dashboard.py και περιλαμβάνει επιλογή τύπου log, επιλογή parser, δυναμική εμφάνιση των παραμέτρων (Παράρτημα D) βάσει του PARSER_DEFAULTS,. Οι παράμετροι αποθηκεύονται σε λεξικό και μετατρέπονται στους κατάλληλους τύπους μέσω PARSER_PARAM_TYPES.
- Upload αρχείου και μήνυμα επιβεβαίωσης: Ο χρήστης μπορεί να ανεβάσει το αρχείο logs μέσω st.file_uploader στο sidebar, το οποίο υποστηρίζει μόνο .log και .txt τύπους για αποφυγή μη υποστηριζόμενων εισόδων. Μετά το ανέβασμα, εμφανίζεται μήνυμα επιβεβαίωσης με st.success, ενημερώνοντας τον χρήστη ότι το αρχείο ανέβηκε επιτυχώς και μπορεί να προχωρήσει στην επιλογή parser και παραμέτρων. Η υλοποίηση αυτού του feedback είναι σημαντική για την εμπειρία χρήστη, καθώς εξασφαλίζει σαφήνεια και καθοδήγηση στη ροή του interface (Παράρτημα Ε).
- Διαχείριση φίλτρων: Κάθε dashboard υλοποιεί δικά του filters ανά πεδίο, με χρήση st.multiselect, st.date_input κ.λπ., τοποθετημένα σε στήλες μέσω st.columns. Οι επιλογές φιλτραρίσματος εφαρμόζονται στα parsed logs χωρίς να επανεκτελείται το parsing, αξιοποιώντας το state του Streamlit (Παράρτημα F).
- Tabs και προβολή δεδομένων: Το `st.radio` χρησιμοποιείται για την υλοποίηση της πλοήγησης, η οποία καθορίζει αν ο χρήστης βλέπει το dashboard με γραφήματα, τον πίνακα προτύπων ή τον πίνακα logs. Οι πίνακες χρησιμοποιούν st.dataframe και υποστηρίζουν native λειτουργίες όπως search, full-screen και export σε CSV.
- Dashboards ανά τύπο αρχείου: Καθένα περιλαμβάνει κατάλληλα γραφήματα με Altair ή Plotly, τα οποία επιλέχθηκαν βάσει του είδους των δεδομένων και των use cases. Προκειμένου να διασφαλιστεί μια συνεπής εμπειρία χρήστη σε όλους τους τύπους log, η συνάρτηση `show_dashboard()` επιστρέφει πάντα μια modular δομή (header, filters, charts).

6.6.3 Προβλήματα και Τρόποι Επίλυσης

Πολλά τεχνικά προβλήματα αναδύθηκαν κατά την ανάπτυξη, επηρεάζοντας την απόδοση και τη χρηστικότητα του συστήματος ή την ακρίβεια των αποτελεσμάτων.

Σε περιπτώσεις όπου οι εγγραφές log δεν ταίριαζαν ακριβώς με το log_format, αυτές αγνοούνταν αθόρυβα από τον parser. Η αιτία ήταν ότι ορισμένα πεδία του regex format ήταν υποχρεωτικά. Η λύση δόθηκε με τη μετατροπή των πεδίων αυτών σε προαιρετικά, μέσω της προσθήκης "?" στο regex pattern. Αυτό επέτρεψε στο σύστημα να διατηρεί τις εγγραφές και να επιστρέφει None ή κενή τιμή για όσα πεδία έλειπαν.

Κατά την εμφάνιση heatmap στο dashboard των Linux logs, παρατηρήθηκε ότι η βιβλιοθήκη Altair εμφάνιζε μόνο δύο τιμές στον άξονα χρόνου, ανεξαρτήτως πλήθους δεδομένων. Αυτό οφειλόταν σε αυτόματη ομαδοποίηση του πεδίου Time κατά binning, που αλλοίωνε την ανάλυση. Η επίλυση δόθηκε με την αντικατάσταση του συγκεκριμένου chart από Plotly heatmap, που απέδωσε ορθά όλες τις διακριτές χρονικές τιμές και βελτίωσε τη διαδραστικότητα του χρήστη (Παράρτημα G).

Στην πρώτη φόρτωση της σελίδας μετά το parsing, όλα τα διαθέσιμα φίλτρα εμφανίζονταν επιλεγμένα εξ ορισμού, προκειμένου να εμφανιστούν όλα τα δεδομένα χωρίς περιορισμούς. Όμως, επειδή το Streamlit εμφανίζει κάθε επιλεγμένη τιμή ως label μέσα στο input box του st.multiselect, σε περιπτώσεις με μεγάλο αριθμό διαθέσιμων τιμών, το widget αποκτούσε υπερβολικό ύψος, καταστρέφοντας την ευθυγράμμιση των στοιχείων στο UI και την εργονομία των st.columns. Για την αντιμετώπιση του προβλήματος, υλοποιήθηκε μηχανισμός όπου κάθε φίλτρο περιλαμβάνει μια επιλογή "ALL" μαζί με τις υπόλοιπες τιμές, και ο έλεγχος γίνεται εκ των υστέρων. Αν είναι επιλεγμένο το "ALL" ή δεν έχει επιλεγεί τίποτα, εφαρμόζονται όλα τα διαθέσιμα values, χωρίς να εμφανίζονται όλα τα labels. Έτσι, διατηρείται το πλήρες αποτέλεσμα στην έξοδο χωρίς να επηρεάζεται το layout του interface (Παράρτημα F).

6.7 Επίλογος

Στο έκτο κεφάλαιο παρουσιάστηκε η πλήρης διαδικασία υλοποίησης του συστήματος, καλύπτοντας την αρχιτεκτονική σχεδίαση, τη ροή δεδομένων, τη λειτουργικότητα parsing και οπτικοποίησης, καθώς και τις τεχνικές λεπτομέρειες που υποστηρίζουν την ευχρηστία και την επεκτασιμότητα της εφαρμογής.

Ιδιαίτερη έμφαση δόθηκε στην οργάνωση και επαναχρησιμοποίηση του κώδικα, μέσω κοινών βοηθητικών δομών, καθώς και στη δυνατότητα δυναμικής προσθήκης νέων parsers ή τύπων logs με ελάχιστες αλλαγές. Η προσέγγιση αυτή καθιστά το σύστημα παραμετροποιήσιμο, modular και testable, σύμφωνα με βασικές αρχές καλής σχεδίασης λογισμικού.

Μέσα από συγκεκριμένα παραδείγματα παρουσιάστηκε η συνολική εμπειρία χρήστη, η ροή από την εισαγωγή του αρχείου έως την προβολή των δεδομένων και των αναλύσεων, καθώς και η ενσωμάτωση στοιχείων διαδραστικότητας που βασίζονται σε τεκμηριωμένες πρακτικές σχεδίασης UI.

Παράλληλα, τεκμηριώθηκαν προβλήματα που προέκυψαν κατά την ανάπτυξη και παρουσιάστηκαν οι λύσεις που εφαρμόστηκαν, είτε με τροποποίηση του κώδικα είτε με αντικατάσταση εργαλείων.

Τέλος, αποτυπώθηκαν οι βελτιώσεις στην αρχιτεκτονική του συστήματος στην πορεία της υλοποίησης, με σκοπό την εξάλειψη επαναλήψεων, τη βελτίωση της συντηρησιμότητας και τη διευκόλυνση μελλοντικών επεκτάσεων. Η εφαρμογή που προέκυψε είναι σταθερή, προσαρμόσιμη σε διαφορετικά

log formats, και ικανή να προσφέρει ουσιαστική υποστήριξη στην ανάλυση καταγραφών μέσω parsing και οπτικοποίησης.

Για τη διευκόλυνση της πρόσβασης και χρήσης, η εφαρμογή έχει αναπτυχθεί και είναι διαθέσιμη online στη διεύθυνση https://log-parser.streamlit.app. Διατέθηκε μέσω του Streamlit Cloud και του GitHub, τα οποία παρέχουν απλή ανάπτυξη για εφαρμογές Python χωρίς να απαιτούν περίπλοκες ρυθμίσεις διακομιστή ή υποδομής. Για την αυτόματη μεταφορά και φιλοξενία της εφαρμογής σε ένα περιβάλλον web, ένα αποθετήριο GitHub με τα αρχεία Python (.py) και ένα αρχείο requirements.txt είναι επαρκές. Αυτό καθιστά δυνατή τη χρήση της εφαρμογής από τους τελικούς χρήστες μέσω browser, χωρίς εγκατάσταση λογισμικού.

Κεφάλαιο 7ο: Συμπεράσματα

Η παρούσα εργασία πραγματεύτηκε το ζήτημα της ανάλυσης και οπτικοποίησης δεδομένων καταγραφής (log data), προσεγγίζοντάς το τόσο θεωρητικά όσο και πρακτικά. Μέσα από την επισκόπηση της βιβλιογραφίας και την υλοποίηση ενός ολοκληρωμένου συστήματος parsing και visualization, αναδείχθηκαν οι προκλήσεις, οι τεχνικές λύσεις και οι δυνατότητες αξιοποίησης των logs για σκοπούς διαχείρισης, παρακολούθησης και διάγνωσης πληροφοριακών συστημάτων.

Σε θεωρητικό επίπεδο, καταδείχθηκε η σημασία των logs ως κρίσιμη πηγή πληροφορίας για εφαρμογές όπως anomaly detection, root cause analysis και performance monitoring. Παρουσιάστηκαν οι κύριες τεχνικές log parsing, από παραδοσιακές μεθόδους που βασίζονται σε heuristics και regex, έως εξελιγμένες ML και optimization προσεγγίσεις, καθώς και τα πλεονεκτήματα και οι περιορισμοί καθεμιάς. Εξετάστηκε επίσης η σημασία της ακρίβειας του parsing, η οποία επηρεάζει καταλυτικά την αξιοπιστία των μεταγενέστερων αναλύσεων.

Στο πρακτικό σκέλος, αναπτύχθηκε ένα ευέλικτο και επεκτάσιμο σύστημα ανάλυσης logs, βασισμένο σε Python και Streamlit, με υποστήριξη parsing για διαφορετικούς τύπους logs (Windows, Linux, Mac, Suricata) και χρήση επιλεγμένων αλγορίθμων (Drain, Spell, IPLoM, LogCluster, MoLFI). Ιδιαίτερη έμφαση δόθηκε στον modular σχεδιασμό, στην παραμετροποίηση των parsers και στη φιλικότητα του interface. Η διαδικασία parsing υποστηρίζεται από τεχνικές benchmarking και μηχανισμούς αυτόματης επιλογής παραμέτρων, ενώ η απεικόνιση των δεδομένων υλοποιήθηκε με χρήση Altair και Plotly, ακολουθώντας αρχές task-driven και user-centered visualization.

Η εργασία ανέδειξε πρακτικά ζητήματα που προέκυψαν κατά την ανάπτυξη του συστήματος, όπως η απώλεια εγγραφών λόγω ακατάλληλων regex, προβλήματα αποτύπωσης σε γραφήματα υψηλής πυκνότητας και ασυμβατότητες στην αποθήκευση αρχείων από parsers. Για καθένα εφαρμόστηκαν στοχευμένες λύσεις που βελτίωσαν τη σταθερότητα και την εργονομία της εφαρμογής.

Η προσέγγιση που ακολουθήθηκε ενσωμάτωσε τεκμηριωμένες αρχές σχεδίασης, τόσο στην αρχιτεκτονική όσο και στη διεπαφή χρήστη, καθιστώντας το σύστημα ευπροσάρμοστο και συντηρήσιμο. Η χρήση πραγματικών logs και η προσεκτική επιλογή visual encodings βοήθησαν στη δημιουργία dashboards που προσφέρουν ουσιαστική υποστήριξη σε διαχειριστές και αναλυτές.

Συνολικά, η εργασία πέτυχε τους στόχους της, αποδεικνύοντας ότι ο συνδυασμός parsing και visualization μπορεί να προσφέρει αξιόπιστα, χρήσιμα και επεκτάσιμα εργαλεία για την ανάλυση logs.

Ως επέκταση της παρούσας εργασίας στο μελλον, θα μπορούσαν να ενσωματωθούν τεχνικές ανίχνευσης ανωμαλιών (anomaly detection) πάνω στα δομημένα δεδομένα, με χρήση αλγορίθμων μηχανικής μάθησης, ώστε να αναδεικνύονται αυτόματα ύποπτα patterns. Θα μπορούσε ακόμα, να προστεθεί δυνατότητα εισαγωγής logs από απομακρυσμένες πηγές (όπως servers ή cloud buckets) με υποστήριξη ανάλυσης σε πραγματικό χρόνο. Μια τελευταία αλλά εξίσου σημαντική επέκταση θα μπορούσε μα είναι η σύνδεση με εργαλεία SIEM και η αποθήκευση των αποτελεσμάτων σε βάσεις δεδομένων, κάτι που θα επέτρεπε τη διατήρηση ιστορικών αναλύσεων και τη δημιουργία διαχρονικών dashboards.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] T. Zhang, H. Qiu, G. Castellano, M. Rifai, C. S. Chen, and F. Pianese, "System Log Parsing: A Survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 8, pp. 8596–8614, Aug. 2023.

[2] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards Automated Log Parsing for Large-Scale Log Data Analysis," IEEE Trans. Dependable and Secure Computing, vol. 15, no. 6, pp. 931–944, Nov.–Dec. 2018.

[3] Y. Liu, S. Tao, W. Meng, J. Wang, W. Ma, Y. Chen, Y. Zhao, H. Yang, and Y. Jiang, "Interpretable Online Log Analysis Using Large Language Models with Prompt Strategies," in *Proc. 32nd ACM Int. Conf. on Program Comprehension (ICPC)*, 2024, pp. 35–46.

[4] S. Messaoudi et al., "A Search-based Approach For Accurate Identification of Log Message Formats," in Proc. Int. Conf. on Program Comprehension (ICPC), Gothenburg, Sweden, 2018, pp. 61–71.

[5] M. Du and F. Li, "Spell: Streaming Parsing of System Event Logs," in Proc. IEEE Int. Conf. on Data Mining (ICDM), 2016, pp. 859–864.

[6] J. Zhu et al., "Tools and Benchmarks for Automated Log Parsing," in Proc. IEEE/ACM Int. Conf. on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2019, pp. 121–130.

[7] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "An Evaluation Study on Log Parsing and Its Use in Log Mining," in Proc. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN), 2016, pp. 654–661.

[8] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-Supervised Log Parsing," in Proc. IEEE Int. Conf. on Cloud Computing Technology and Science (CloudCom), 2020, pp. 110–115.

[9] K. S.-P. Chang and S. J. Fink, "Visualizing Serverless Cloud Application Logs for Program Understanding," in Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2017, pp. 261–269.

[10] A. Sarikaya, M. Correll, L. Bartram, M. Tory, and D. Fisher, "What Do We Talk About When We Talk About Dashboards?," IEEE Trans. Visualization and Computer Graphics, vol. 25, no. 1, pp. 682–692, Jan. 2019.

[11] Streamlit, "Basic Concepts of Streamlit," Streamlit Docs, [Online]. Available: https://docs.streamlit.io.

[12] Elastic N.V., "Kibana Dashboards," Elastic, [Online]. Available: https://www.elastic.co/fr/kibana/kibana-dashboard.

[13] M. Sun, H. Ee, Z. Ou, and Y. Chen, "More Flexible: A Free-Customization Technique Based on Log Visualization," in *Proc. 2020 12th Int. Conf. on Measuring Technology and Mechatronics Automation (ICMTMA)*, 2020, pp. 375–381.

[14] H. Borges, R. Akbarinia, and F. Masseglia, “Anomaly Detection in Time Series,” in *Transactions on Large-Scale Data- and Knowledge-Centered Systems*, vol. L, pp. 46–62, 2021.

[15] Plotly Technologies Inc., “Plotly Python Open Source Graphing Library,” [Online]. Available: https://plotly.com/python/

[16] LogPAI Team, “LogPAI: Log Parsing and Analysis Infrastructure,” [Online]. Available: https://github.com/logpai

[17] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, “Drain: An Online Log Parsing Approach with Fixed Depth Tree,” in Proc. IEEE Int. Conf. on Web Services (ICWS), 2017, pp. 33–40.

[18] R. Vaarandi and M. Pihelgas, “LogCluster – A Data Clustering and Pattern Mining Algorithm for Event Logs,” in Proc. Int. Conf. on Network and Service Management (CNSM), 2015.

[19] A. Makanju, A. N. Zincir-Heywood, and E. E. Milios, “Clustering Event Logs Using Iterative Partitioning,” in Proc. ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD), Paris, France, 2009, pp. 1255–1264.

[20] A. Satyanarayan, D. Moritz, K. Wongsuphasawat, and J. Heer, “Vega-Lite: A Grammar of Interactive Graphics,” IEEE Trans. Visualization and Computer Graphics, vol. 23, no. 1, pp. 341–350, Jan. 2017.

ΠΑΡΑΡΤΗΜΑ Α : PARSER_DEFAULTS

```
PARSER_DEFAULTS = {
    "Drain": {
        "Windows": {"depth": "5", "threshold": "0.7"},
        "Linux": {"depth": "4", "threshold": "0.39"},
        "Mac": {"depth": "6", "threshold": "0.7"},
        "Suricata": {"depth": "4", "threshold": "0.7"}
    },
    "Spell": {
        "Windows": {"threshold": "0.7"},
        "Linux": {"threshold": "0.55"},
        "Mac": {"threshold": "0.6"},
        "Suricata": {"threshold": "0.7"}
    },
    "LogCluster": {
        "Windows": {"rsupport": "1"},
        "Linux": {"rsupport": "40"},
        "Mac": {"rsupport": "1"},
        "Suricata": {"rsupport": "1"}
    },
    "IPLoM": {
        "Windows": {"CT": "0.3", "lowerBound": "0.25"},
        "Linux": {"CT": "0.3", "lowerBound": "0.3"},
        "Mac": {"CT": "0.3", "lowerBound": "0.25"},
        "Suricata": {"CT": "0.3", "lowerBound": "0.25"}
    },
    "MoLFI": {
        "Windows": {},
        "Linux": {},
        "Mac": {},
        "Suricata": {}
    }
```

}

ΠΑΡΑΡΤΗΜΑ Β : run_parser

```
def run_parser(uploaded_file, parser_choice, log_format, regex, **kwargs):
    try:
        with tempfile.TemporaryDirectory() as tmp_input_dir, tempfile.TemporaryDirectory() as tmp_output_dir:
            temp_log_path = os.path.join(tmp_input_dir, uploaded_file.name)
            with open(temp_log_path, "wb") as f:
                f.write(uploaded_file.read())

            parser_args = {
                "log_format": log_format,
                "indir": tmp_input_dir,
                "outdir": tmp_output_dir,
                "rex": regex
            }
            parser_args.update(kwargs)

            param_types = PARSER_PARAM_TYPES.get(parser_choice, {})
            for key, expected_type in param_types.items():
                try:
                    if key in kwargs:
                        kwargs[key] = expected_type(kwargs[key])
                except ValueError:
                    st.error(f" Η παράμετρος '{key}' πρέπει να είναι τύπου {expected_type.__name__}.")
                    st.stop()

            parser_factory = PARSER_FACTORY.get(parser_choice)
            if not parser_factory:
                st.warning("Άγνωστος parser.")
                return pd.DataFrame(), pd.DataFrame()
```

```
            parser = parser_factory(parser_args)
            parser.parse(uploaded_file.name)

            structured_path = os.path.join(tmp_output_dir, uploaded_file.name + "_structured.csv")
            templates_path = os.path.join(tmp_output_dir, uploaded_file.name + "_templates.csv")

            df_structured = pd.read_csv(structured_path) if os.path.exists(structured_path) else pd.DataFrame()
            df_templates = pd.read_csv(templates_path) if os.path.exists(templates_path) else pd.DataFrame()

            return df_structured, df_templates
    except Exception as e:
        st.error(f" Σφάλμα κατά την εκτέλεση του parser: {e}")
        return pd.DataFrame(), pd.DataFrame()
```

ΠΑΡΑΡΤΗΜΑ C : PARSER_FACTORY

```
PARSER_FACTORY = {
    "Drain": lambda args: DrainParser(
        args["log_format"],
        indir=args["indir"],
        outdir=args["outdir"],
        depth=int(args["depth"]),
        st=float(args["threshold"]),
        rex=args.get("rex", [])
    ),
    "Spell": lambda args: SpellParser(
        indir=args["indir"],
        outdir=args["outdir"],
        log_format=args["log_format"],
        tau=float(args["threshold"]),
        rex=args.get("rex", [])
    ),
```

```
    "LogCluster": lambda args: LogClusterParser(
        args["indir"],
        args["log_format"],
        args["outdir"],
        rsupport=int(args["rsupport"])
    ),
    "IPLoM": lambda args: IPLoMParser(
        log_format=args["log_format"],
        indir=args["indir"],
        outdir=args["outdir"],
        CT=float(args["CT"]),
        lowerBound=float(args["lowerBound"]),
        rex=args.get("rex", [])
    ),
    "MoLFI": lambda args: MoLFIParser(
        indir=args["indir"],
        outdir=args["outdir"],
        log_format=args["log_format"],
        rex=args.get("rex", [])
    )
}
```

ΠΑΡΑΡΤΗΜΑ D : Sidebar

```
with st.sidebar:
    st.header("Ρυθμίσεις Parser")
    log_type = st.selectbox("Τύπος Log", options=["Windows", "Linux", "Mac", "Suricata"])
    parser_choice = st.selectbox("Διάλεξε Log Parser", options=["Drain", "Spell", "LogCluster", "IPLoM", "MoLFI"]
)
    parser_params = {}

    default_params = PARSER_DEFAULTS.get(parser_choice, {}).get(log_type, {})
```

```
    for param_key, param_val in default_params.items():
        parser_params[param_key] = st.text_input(param_key, value=param_val)

    st.markdown("---")
    run_parse = st.button("Parse")
```

ΠΑΡΑΡΤΗΜΑ Ε : file_uploader

```
uploaded_file = st.file_uploader("Ανέβασε το αρχείο log σου", type=["log", "txt"])
if uploaded_file is not None:
    st.success(f" Το αρχείο ανέβηκε: `{uploaded_file.name}`")
```

ΠΑΡΑΡΤΗΜΑ F : Filters

```
    # Filters for Windows Logs
    levels_available = df_structured["Level"].dropna().unique().tolist()
    components_available = df_structured["Component"].dropna().unique().tolist()

    min_date = pd.to_datetime(df_structured["Date"]).min().date()
    max_date = pd.to_datetime(df_structured["Date"]).max().date()

    all_levels_option = "ALL"
    all_components_option = "ALL"

    level_options = [all_levels_option] + levels_available
    component_options = [all_components_option] + components_available

    col1, col2, col3 = st.columns(3)
    with col1:
        selected = st.multiselect("Επέλεξε Level(s):", level_options)
        if all_levels_option in selected or not selected:
            selected_levels = levels_available
        else:
            selected_levels = selected
    with col2:
```

```
        selected = st.multiselect("Επέλεξε Component(s):", component_options)
        if all_components_option in selected or not selected:
            selected_components = components_available
        else:
            selected_components = selected
    with col3:
        selected_date_range = st.date_input("Επέλεξε Ημερομηνίες:", value=[min_date, max_date])
    st.markdown("---")

    filtered_df = df_structured[
        df_structured["Level"].isin(selected_levels) &
        df_structured["Component"].isin(selected_components) &
        (pd.to_datetime(df_structured["Date"]) >= pd.to_datetime(selected_date_range[0])) &
        (pd.to_datetime(df_structured["Date"]) <= pd.to_datetime(selected_date_range[1]))
    ]
    if filtered_df.empty:
        st.warning("Δεν υπάρχουν δεδομένα για τα επιλεγμένα φίλτρα. Παρακαλώ δοκιμάστε άλλες επιλογές.")
        return
```

ΠΑΡΑΡΤΗΜΑ G : Heatmap

```
    st.markdown("Heatmap Κατανομής Component ανά Timestamp")
    df_heatmap = filtered_df.groupby(["Component", "datetime"]).size().reset_index(name="Count")

    fig = px.density_heatmap(
        df_heatmap,
        x="datetime",
        y="Component",
        z="Count",
        nbinsx=100,
        color_continuous_scale="Viridis",
        labels={"datetime": "Χρόνος", "Component": "Component", "Count": "Logs"},
    )
```

```
fig.update_layout(
    xaxis_title="Χρόνος",
    yaxis_title="Component",
    xaxis_tickformat="%d-%m %H:%M",
    autosize=True,
    height=500
)

st.plotly_chart(fig, use_container_width=True)
```