



ΠΑΝΕΠΙΣΤΗΜΙΟ  
ΠΑΤΡΩΝ  
UNIVERSITY OF PATRAS

Πολυτεχνική Σχολή  
Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Διπλωματική Εργασία

---

# Αλγόριθμοι Τεχνητής Νοημοσύνης για την ανίχνευση malware σε TLS κίνηση

---

Μπουρλάκης Γεώργιος  
1054321

Επιβλέπων  
Κυριάκος Βλάχος, Καθηγητής

Μέλος Επιτροπής Αξιολόγησης  
Κωνσταντίνος Τσίχλας, Επίκουρος καθηγητής

Μέλος Επιτροπής Αξιολόγησης  
Αριστείδης Ηλίας, ΕΔΙΠ

Πάτρα, Νοέμβριος 2021





UNIVERSITY OF  
**PATRAS**  
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

Πανεπιστήμιο Πατρών

Πολυτεχνική Σχολή

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής

© Copyright συγγραφής Μπουρλάκης Γεώργιος, 2021

© Copyright θέματος Βλάχος Κυριάκος, Καθηγητής

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Η έγκριση τις διπλωματικής εργασίας από το Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών & Πληροφορικής του Πανεπιστημίου Πατρών δεν υποδηλώνει απαραίτητως και αποδοχή των απόψεων του συγγραφέα εκ μέρους του Τμήματος.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή τις παρούσας εργασίας, εξ ολοκλήρου ή τμήματος τις, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα.

Το περιεχόμενο αυτής της εργασίας δεν απηχεί τις απόψεις του Τμήματος, του Επιβλέποντα, ή της επιτροπής που την ενέκρινε.

## Υπεύθυνη Δήλωση

Βεβαιώνω ότι είμαι συγγραφέας αυτής της διπλωματικής εργασίας, και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην διπλωματική εργασία. Επίσης έχω αναφέρει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επίσης, βεβαιώνω ότι αυτή η διπλωματική εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για τις απαιτήσεις του προγράμματος σπουδών του Τμήματος Μηχανικών Ηλεκτρονικών Υπολογιστών και Πληροφορικής του Πανεπιστημίου Πατρών.

(Υπογραφή)

.....  
Μπουρλάκης Γεώργιος

# Ευχαριστίες

*Θα ήθελα να ευχαριστήσω θερμά για τη στήριξη και τη βοήθεια τον επιβλέποντα Καθηγητή κ. Βλάχο Κυριάκο, καθώς και τους Τσίχλα Κωνσταντίνο, Επίκουρο καθηγητή, και Ηλία Αριστείδη, ΕΔΙΠ, για την καθοδήγηση σε όλη τη διάρκεια εκπόνησης της διπλωματικής εργασίας. Επίσης, θα ήθελα να ευχαριστήσω την οικογένειά μου για την ψυχολογική και οικονομική στήριξη κατά τη διάρκεια των σπουδών μου.*



# Πρόλογος

Τα τελευταία χρόνια η ασφάλεια ψηφιακών συστημάτων και δικτύων αποτελεί σημείο ενδιαφέροντος στον κόσμο της τεχνολογίας, κυρίως για οικονομικούς, πολιτικούς και κοινωνικούς λόγους. Η ασφάλεια έχει 3 κύριους στόχους, οι οποίοι είναι εμπιστευτικότητα, δηλαδή απόκρυψη πληροφορίας από τρίτους, ακεραιότητα, δηλαδή να είναι αξιόπιστη η πληροφορία, και διαθεσιμότητα, δηλαδή να υπάρχει πρόσβαση στην πληροφορία. Γι' αυτό και προσπαθούμε σε κάθε σύστημα να συνδυάζουμε αυτούς τους στόχους, προκειμένου να πετύχουμε τη μέγιστη δυνατή ασφάλεια.

Το πρόβλημα είναι ότι ποτέ ένα σύστημα δεν είναι απολύτως ασφαλές, καθώς με την εξέλιξη της τεχνολογίας εξελίσσονται και οι τεχνικές των κακόβουλων επιτιθέμενων, με αποτέλεσμα να προκύπτουν συνεχώς νέες απειλές και ευπάθειες. Επομένως, δεν πρέπει να υπάρχει στασιμότητα στο χώρο της ασφάλειας, αλλά συνεχής ενημέρωση και συγχρονισμός με τα νέα δεδομένα.

Η καλύτερη αντιμετώπιση για μια επίθεση είναι η πρόληψη και όχι η αναζήτηση λύσεων μετά την επίθεση για την ελαχιστοποίηση των συνεπειών. Έτσι, πρέπει να εφαρμόζονται μηχανισμοί που θα αποτρέπουν την αποτελεσματικότητα μίας επίθεσης, δηλαδή θα την κάνουν να αποτύχει. Ούτε η πρόληψη είναι απολύτως αποτελεσματική, γι' αυτό στη συνέχεια υπάρχει και ο εντοπισμός της επίθεσης, που πρέπει να είναι άμεσος για να μη δημιουργηθούν μεγάλα προβλήματα. Τέλος, υπάρχει και η επαναφορά της επίθεσης που οδηγεί σε αποκατάσταση του συστήματος στην προηγούμενη φυσιολογική του κατάσταση.

# Περίληψη

Ο στόχος της παρούσας διπλωματικής εργασίας είναι η εφαρμογή αλγορίθμων τεχνητής νοημοσύνης, οι οποίοι ανιχνεύουν κακόβουλο λογισμικό σε κίνηση του πρωτοκόλλου TLS. Το πρωτόκολλο TLS είναι αυτό που χρησιμοποιείται κατά κόρον για την επικοινωνία πελάτη και διακομιστή στο Διαδίκτυο. Προσφέρει κρυπτογράφηση πακέτων για προστασία μηνυμάτων, συναλλαγών και άλλων δραστηριοτήτων. Η καινοτομία αυτής της εργασίας είναι ότι τα πακέτα δεν αποκρυπτογραφούνται για να ελέγξουμε αν περιέχουν κακόβουλο λογισμικό, αλλά εκπαιδεύονται αλγόριθμοι τεχνητής νοημοσύνης, με βάση χαρακτηριστικά, πιστοποιητικά και μεταδεδομένα που επιλέγουμε εμείς, να προβλέπουν με μεγάλη ακρίβεια αν πρόκειται για κακόβουλο λογισμικό ή όχι. Αυτό έχει σαν αποτέλεσμα να μην παραβιάζουμε τα προσωπικά δεδομένα των χρηστών, όπως κωδικούς πρόσβασης, αφού δεν αποκρυπτογραφούνται τα πακέτα και επίσης να ελαχιστοποιούμε την καθυστέρηση στην επικοινωνία πελάτη και διακομιστή, κάτι που δεν θα συνέβαινε αν αποκρυπτογραφούσαμε κάθε ένα πακέτο ξεχωριστά.

# **Abstract**

The goal of this diploma thesis is the application of artificial intelligence algorithms, which detect malware in the traffic of TLS protocol. The TLS protocol is the one used extensively for client and server communication over the Web. It provides packet encryption in order to protect messages, transactions and other activities. The novelty of this work is that the packages are not decrypted to check if they contain malware, but artificial intelligence algorithms are trained, based on features, certificates and metadata we choose, to predict with great accuracy whether it is malware or not. As a result, we do not violate users' personal data, such as passwords, since packets are not decrypted and we also minimize delays in the communication of client and server, which would not happen if we decrypted each packet separately.



# Περιεχόμενα

Ευχαριστίες.....	5
Πρόλογος.....	7
Περίληψη.....	8
Abstract.....	9
1 Εισαγωγή.....	10
1.1 Στόχοι διπλωματικής εργασίας.....	10
1.2 Δομή διπλωματικής εργασίας.....	10
2 TLS πρωτόκολλο, επιλογή δεδομένων και χαρακτηριστικών και ταξινόμηση... 12	
2.1 SSL, TLS.....	12
2.1.1 SSL.....	12
2.1.2 TLSv1.....	13
2.1.3 TLSv2.....	14
2.1.4 TLSv3.....	14
2.2 Συλλογή δεδομένων.....	16
2.3 Επιλογή χαρακτηριστικών.....	17
2.3.1 Επιλογή υποσυνόλου χαρακτηριστικών.....	17
2.3.2 Διαδικασία επιλογής χαρακτηριστικών.....	23
2.3.3 Στιβαρότητα χαρακτηριστικών.....	23
2.4 Ταξινόμηση ροών δεδομένων.....	25
2.4.1 Επιβλεπόμενη-μη επιβλεπόμενη εκπαίδευση.....	25
2.4.2 Μοντέλα ταξινόμησης.....	26
2.4.3 Μετρικές απόδοσης μοντέλων.....	27
2.4.4 Αποτελέσματα.....	29
3 Ανάλυση πειράματος.....	32
3.1 Διαδικασία επιλογής δεδομένων.....	32
3.2 Διαδικασία εξαγωγής χαρακτηριστικών.....	33
3.3 Υλοποίηση αλγορίθμων και ταξινόμηση.....	36
4 Συμπεράσματα.....	37
4.1 Σύνοψη.....	37
4.2 Μελλοντική εργασία.....	38

Παράρτημα Α – κώδικας.....	39
Παράρτημα Β – πίνακας ciphersuites.....	53
Βιβλιογραφία.....	57

# 1 Εισαγωγή

## 1.1 Στόχοι διπλωματικής εργασίας

Η διπλωματική εργασία έχει ως στόχο τη δημιουργία ενός συστήματος, το οποίο δίνοντας του κατάλληλα δεδομένα, κακόβουλο λογισμικό και μη, να τα επεξεργάζεται με βάση τα χαρακτηριστικά που εμείς θεωρούμε σημαντικά και να καταλήγει στην ταξινόμηση δεδομένων που δεν έχει ξαναδεί σε κακόβουλο λογισμικό ή όχι. Αυτό γίνεται με το διαχωρισμό των αρχικών δεδομένων σε ένα κομμάτι για εκπαίδευση και στο υπόλοιπο για αξιολόγηση του συστήματος, δηλαδή το τμήμα των δεδομένων που δεν έχει ξαναδεί το σύστημα. Με αυτόν τον τρόπο γίνεται η εκπαίδευση των αλγορίθμων τεχνητής νοημοσύνης, ώστε σε νέα δεδομένα να επιστρέφει ικανοποιητικά αποτελέσματα. Κάποιες από τις βασικές προϋποθέσεις για την επιτυχία του συστήματος είναι η κατάλληλη επιλογή δεδομένων, δηλαδή να γνωρίζουμε με σιγουριά ότι κάποια δεδομένα περιέχουν κακόβουλο λογισμικό και κάποια άλλα όχι, καθώς και να γνωρίζουμε ότι αυτά τα δεδομένα είναι σύγχρονα, δηλαδή να έχουν εμφανιστεί το πολύ 5 χρόνια πριν σε σχέση με τη χρονολογία χρήσης του συστήματος. Αυτό θα οδηγήσει σε πολύ καλύτερα αποτελέσματα, αφού το νέο κακόβουλο λογισμικό συνήθως θα έχει κοινά χαρακτηριστικά με κάποιο προηγούμενο οπότε θα είναι ευκολότερο για εμάς να το ανιχνεύσουμε. Ένας ακόμα στόχος είναι η σύγκριση μεταξύ διάφορων αλγορίθμων τεχνητής νοημοσύνης, δηλαδή να δούμε μέσα από πειράματα ποιος επιστρέφει τα καλύτερα αποτελέσματα στο λιγότερο χρόνο.

## 1.2 Δομή διπλωματικής εργασίας

Η διπλωματική εργασία ξεκινάει με το Κεφάλαιο 1, όπου γίνεται μια εισαγωγή στο πρόβλημα και αναγνωρίζονται οι στόχοι που επιθυμούμε. Στη συνέχεια ακολουθεί το Κεφάλαιο 2, στο οποίο γίνεται μια αναδρομή στο πρωτόκολλο TLS και αναφέρουμε τις πηγές των δεδομένων που χρησιμοποιήσαμε. Επίσης, αναφέρουμε και τα χαρακτηριστικά που επιλέξαμε, στα οποία στηριζόμαστε για την ταξινόμηση των δεδομένων, και τους λόγους για τους οποίους έγιναν αυτές οι επιλογές. Επιπλέον, γίνεται ανάλυση των αλγορίθμων τεχνητής νοημοσύνης που χρησιμοποιήθηκαν, τονίζονται τα πλεονεκτήματα και τα μειονεκτήματα του καθενός και περιγράφουμε τα αποτελέσματα της ταξινόμησης. Στο Κεφάλαιο 3 γίνεται η ανάλυση της διαδικασίας του πειράματος που κάναμε. Τέλος, στο Κεφάλαιο 4 παρουσιάζονται συνοπτικά τα αποτελέσματα των αλγορίθμων και γίνεται αναφορά για μελλοντική εργασία.

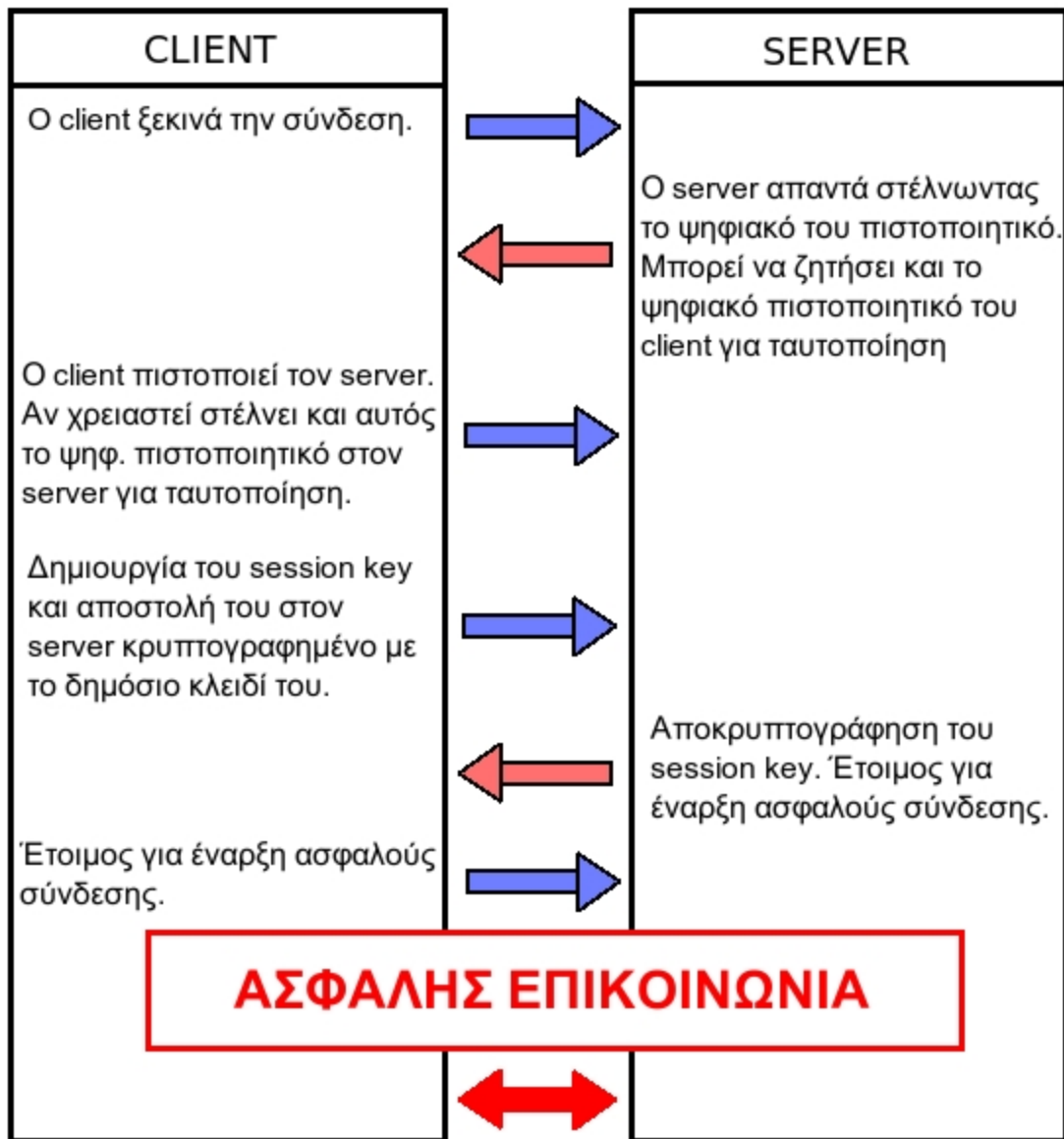
## **2 TLS πρωτόκολλο, επιλογή δεδομένων και χαρακτηριστικών και ταξινόμηση**

### **2.1 SSL, TLS**

#### **2.1.1 SSL**

Καθώς το Ίντερνετ μεγάλωνε, εμφανίστηκε η ανάγκη για ένα πρωτόκολλο κρυπτογράφησης, που θα παρείχε ασφάλεια στην επικοινωνία μεταξύ πελάτη και διακομιστή. Έτσι, οδηγηθήκαμε στη δημιουργία του πρωτοκόλλου SSL (Secure Sockets Layer) [1] από την Netscape Communications, η οποία το 1996 κυκλοφόρησε την πιο σταθερή έκδοση μέχρι τότε, την SSL 3.0. Το SSL βρίσκεται ανάμεσα στο επίπεδο δικτύου και στο επίπεδο εφαρμογών ανάλογα τη λειτουργία του και προσφέρει ένα ασφαλές κρυπτογραφημένο κανάλι επικοινωνίας ανάμεσα σε πελάτη και διακομιστή, καθώς και πιστοποίηση του πελάτη από τον διακομιστή και αντίστροφα. Αυτά επιτυγχάνονται με κάποιους αλγόριθμους κρυπτογράφησης όπως είναι οι DES (Data Encryption Standard), DSA (Digital Signature Algorithm), RSA (Rivest Shamir Adleman) κ.α.

Υπάρχουν 2 μορφές κρυπτογραφίας που χρησιμοποιούνται σήμερα, η συμμετρικού κλειδιού και η δημοσίου κλειδιού. Η πρώτη χρησιμοποιεί ένα μόνο κλειδί και για κρυπτογράφηση και για αποκρυπτογράφηση των μηνυμάτων, με αποτέλεσμα να είναι πιο γρήγορη και αποδοτική σε σχέση με την δημοσίου κλειδιού, που χρησιμοποιεί ένα ιδιωτικό κλειδί για αποκρυπτογράφηση του μηνύματος και ένα δημόσιο για κρυπτογράφηση του μηνύματος. Το πλεονέκτημα της δεύτερης είναι ότι παρέχει καλύτερες τεχνικές πιστοποίησης αυτών που επικοινωνούν. Το πρωτόκολλο SSL χρησιμοποιεί ένα συνδυασμό των 2 μορφών κρυπτογραφίας και ξεκινάει με τη χειραψία, μια διαδικασία για να πιστοποιηθούν πελάτης και διακομιστής. Η αναλυτική λειτουργία της χειραψίας φαίνεται στην παρακάτω εικόνα [2].

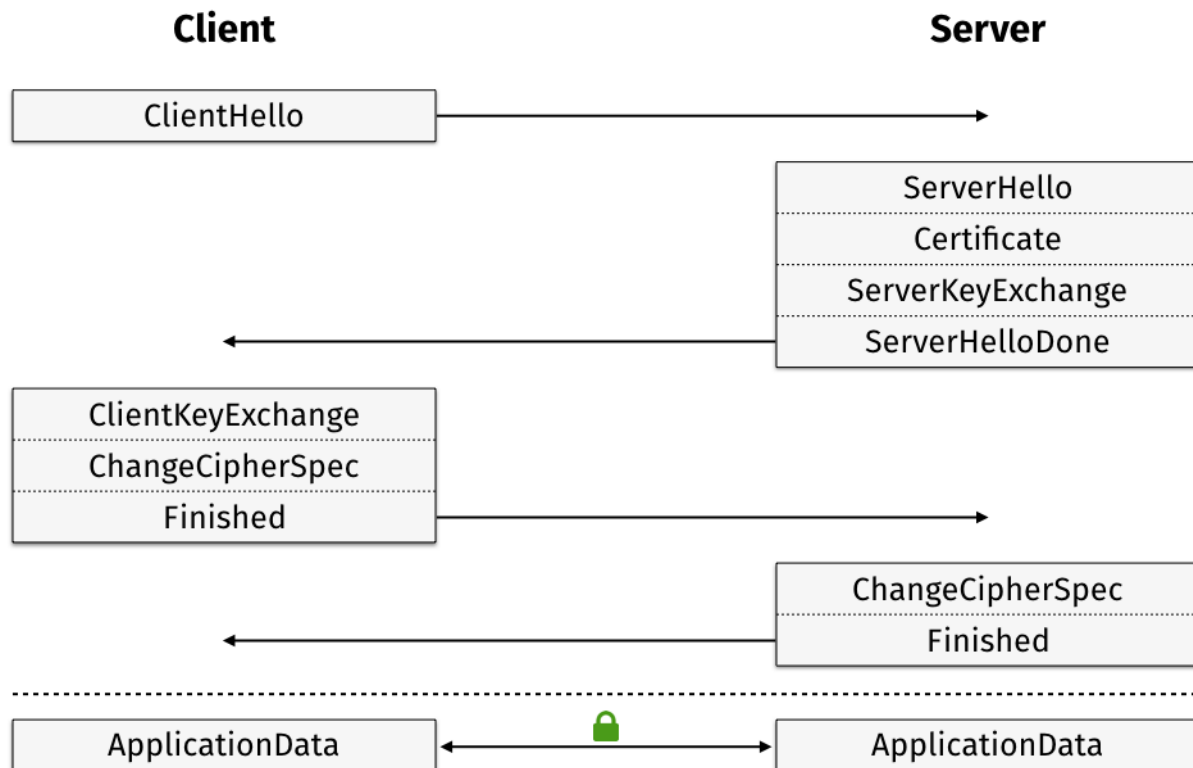


### 2.1.2 TLSv1

Το TLS (Transport Layer Security) πρωτόκολλο εμφανίστηκε μετά το SSL με την έκδοση TLSv1.0 να ανακοινώνεται το 1999 έχοντας μικρές διαφορές σε σχέση με το SSL 3.0. Στη συνέχεια, δημιουργήθηκε η αναβάθμιση του TLSv1.0, το TLSv1.1 τη χρονιά 2006, με σημαντικές διαφορές στην προστασία από επιθέσεις CBC (Cipher Block Chaining) και δυνατότητα υποστήριξης παραμέτρων IANA (Internet Assigned Numbers Authority).

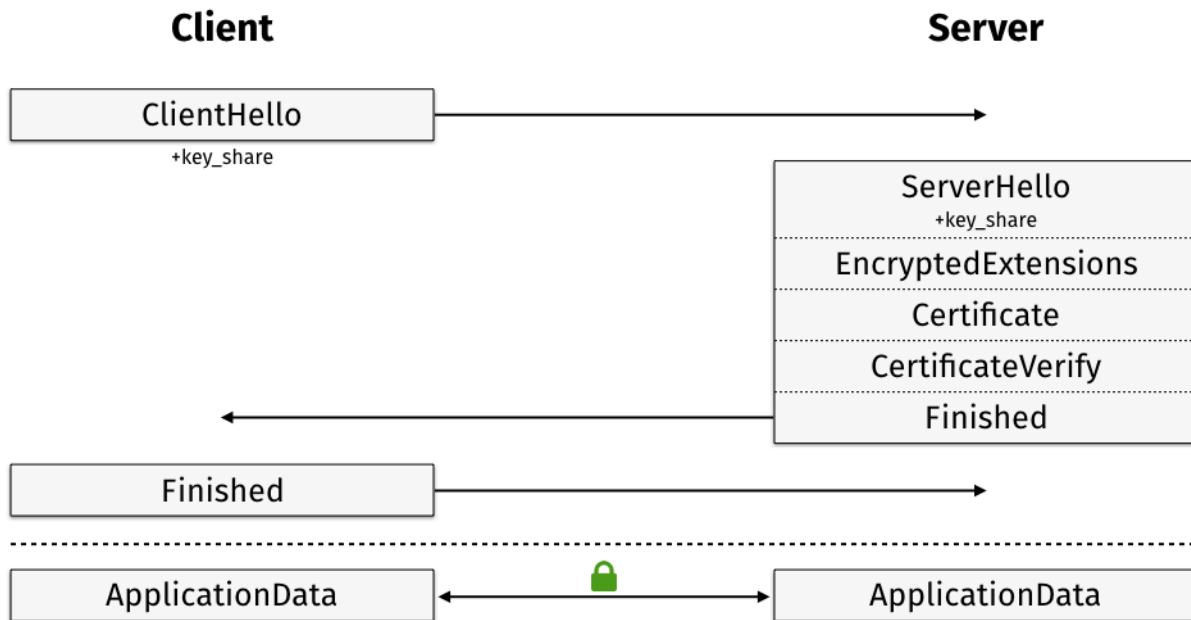
### 2.1.3 TLSv2

Δεν άργησε να έρθει και η αναβάθμιση του TLSv1.1, το TLSv1.2 τη χρονιά 2008, με διαφορές στην αντικατάσταση του MD5-SHA-1 από τον αλγόριθμο SHA-256, δυνατότητα και στον πελάτη και στον διακομιστή επιλογής των αλγορίθμων που αποδέχονται και δυνατότητα επιλογής του αλγορίθμου AES (Advanced Encryption Standard). Η διαδικασία της χειραψίας συνεχίζεται και στο TLSv1.2, είναι βελτιωμένη σε σχέση με τη χειραψία του SSL και στην παρακάτω εικόνα φαίνεται η αναλυτική λειτουργία της [3].



### 2.1.4 TLSv3

Το TLSv1.3 εμφανίζεται το 2018 έχοντας πολλές και σημαντικές διαφορές σε σχέση με τις προηγούμενες εκδόσεις [4] [5]. Οι κυριότερες είναι ότι υποστηρίζει νέους αλγόριθμους κρυπτογράφησης που μπορεί να μην είναι συμβατές με τις προηγούμενες εκδόσεις, αφαιρεί αδύναμους αλγορίθμους και συναρτήσεις κατακερματισμού, όλα τα μηνύματα μετά το **ServerHello** της χειραψίας είναι κρυπτογραφημένα κ.α. Η χειραψία του TLSv1.3 που περιέχει ακόμα λιγότερα μηνύματα συγκριτικά με τις προηγούμενες εκδόσεις φαίνεται στην παρακάτω εικόνα [6].



Επίσης, μπορούμε να δούμε τα ποσοστά χρήσης του κάθε πρωτοκόλλου από τις ιστοσελίδες μέχρι τον Αύγουστο του 2021 [7] και διάφορα στατιστικά για το πόσες ιστοσελίδες είναι ασφαλείς, τι ποσοστό ιστοσελίδων υποστηρίζει το κάθε πρωτόκολλο κλπ [8].

**Website protocol support (Aug. 2021)**

Protocol version	Website support <sup>[70]</sup>	Security <sup>[70][71]</sup>
SSL 2.0	0.4%	Insecure
SSL 3.0	3.2%	Insecure <sup>[72]</sup>
TLS 1.0	44.6%	Deprecated <sup>[8][9][10]</sup>
TLS 1.1	48.9%	Deprecated <sup>[8][9][10]</sup>
TLS 1.2	99.5%	Depends on cipher <sup>[n 1]</sup> and client mitigations <sup>[n 2]</sup>
TLS 1.3	47.8%	Secure

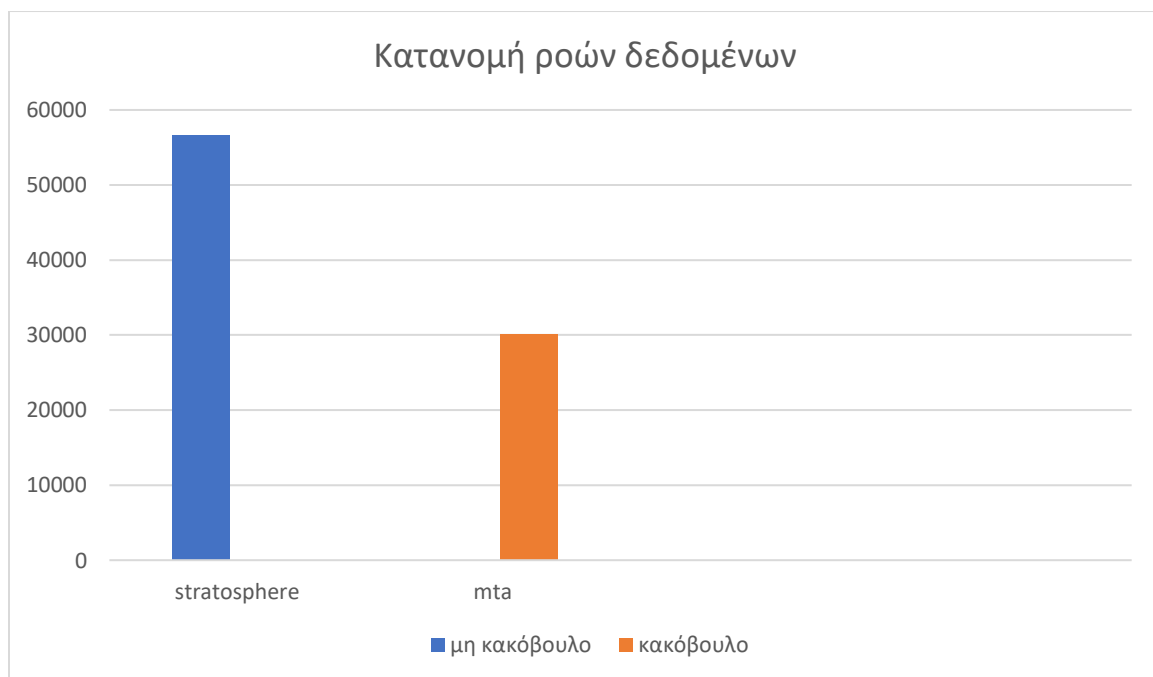
## 2.2 Συλλογή δεδομένων

Η συλλογή των δεδομένων είναι ένα πολύ σημαντικό κομμάτι για την καλή απόδοση των αλγορίθμων τεχνητής νοημοσύνης, αφού πρέπει να γνωρίζουμε ποια δεδομένα περιέχουν σίγουρα κακόβουλο λογισμικό και ποια όχι ώστε να γίνεται σωστά η ταξινόμηση στη συνέχεια. Επίσης, είναι σημαντικό τα δεδομένα να προέρχονται από κοντινές χρονικές περιόδους για να γίνεται καλύτερα η σύγκριση ανάμεσα στη συμπεριφορά του κακόβουλου και μη λογισμικού, αν και αυτό και δεν είναι πάντοτε εφικτό μιας και το μη κακόβουλο λογισμικό δεν καταγράφεται τόσο συχνά όσο το κακόβουλο.

Για το μη κακόβουλο λογισμικό συλλέχθηκαν δεδομένα από την πηγή [9] με μέγεθος σε αρχεία .pcap ίσο με 2.3G, τα οποία δημιούργησαν 56691 ροές μη κακόβουλων δεδομένων και αναφέρονται όλα στη χρονιά 2017, μιας και όπως αναφέρθηκε και προηγουμένως δεν υπάρχουν αρκετές καταγραφές μη κακόβουλου λογισμικού για δεδομένα TLS πρωτοκόλλου.

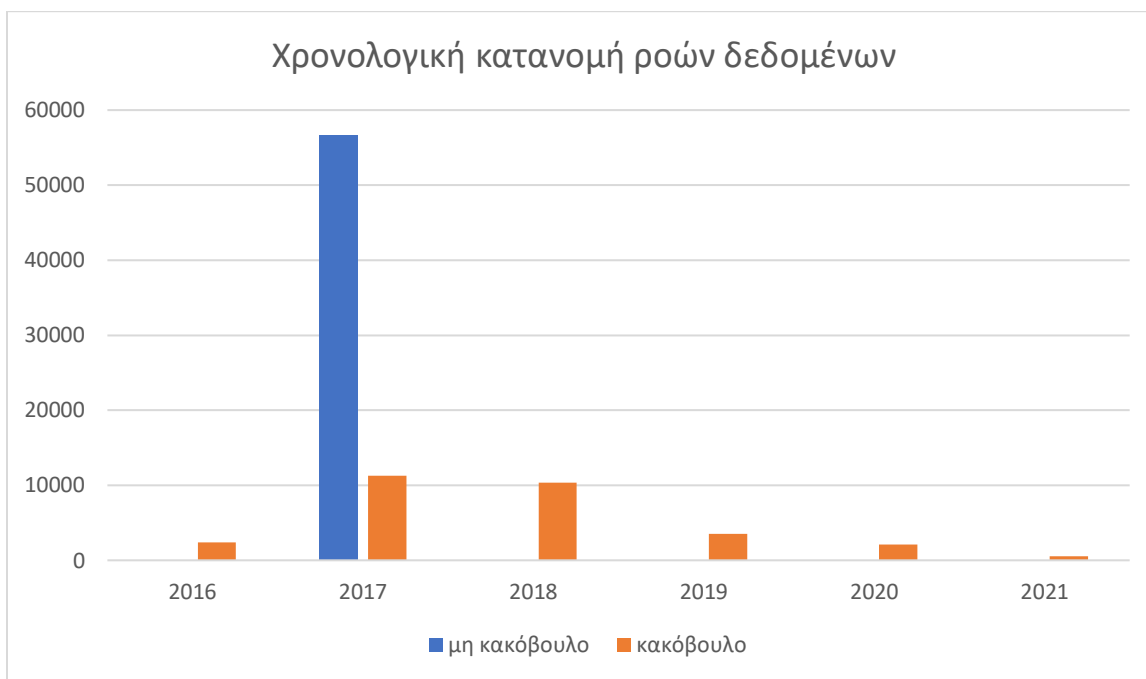
Όσον αφορά το κακόβουλο λογισμικό, η εύρεση τέτοιων δεδομένων είναι πιο εύκολη καθώς καταγράφονται συχνότερα. Συλλέχθηκαν δεδομένα από την πηγή [10] με μέγεθος σε αρχεία .pcap ίσο με 2.5G και δημιούργησαν 30122 ροές κακόβουλων δεδομένων. Αυτά τα δεδομένα αναφέρονται στις χρονιές από το 2016 μέχρι και το 2021 για μεγαλύτερη ποικιλία κακόβουλων οικογενειών δεδομένων.

Στο παρακάτω σχήμα παρουσιάζεται η κατανομή των ροών δεδομένων για κακόβουλο λογισμικό και μη.





Ακόμα, στο παρακάτω σχήμα παρουσιάζεται η χρονολογική κατανομή των ροών δεδομένων.



Επίσης, είναι αξιοσημείωτο να αναφερθεί ότι οι ροές που περιέχουν το πρωτόκολλο TLSv1.1 είναι 10672, αυτές που περιέχουν το TLSv1.2 είναι μόλις 7 και αυτές που περιέχουν το TLSv1.3 είναι 19412.

## 2.3 Επιλογή χαρακτηριστικών

### 2.3.1 Επιλογή υποσυνόλου χαρακτηριστικών

Η επιλογή των χαρακτηριστικών είναι το πιο σημαντικό κομμάτι για τη βέλτιστη ταξινόμηση των ροών δεδομένων, διότι με βάση αυτά γίνεται η ταξινόμηση, η οποία επηρεάζεται και από το βαθμό τους. Ο βαθμός σημαίνει ότι κάποια χαρακτηριστικά έχουν πιο ουσιαστικό ρόλο από κάποια άλλα γι' αυτό και πρέπει να γίνει κατάλληλη επιλογή τους.

Το σύνολο των χαρακτηριστικών παρουσιάζεται σε μία έρευνα που έκανε η Cisco [11] και φαίνεται στον παρακάτω πίνακα.

Category	Feature	Type
Flow Metadata	Source port	Integer
	Destination port	Integer
	Number of inbound bytes	Integer
	Number of outbound bytes	Integer
	Number of inbound packets	Integer
	Number of outbound packets	Integer
	Duration of the flow	Integer
Distribution	Sequence of packet lengths	Stochastic matrix
	Sequence of packet times	Stochastic matrix
	Byte distribution	Length-256 Array
TLS Metadata	List of ciphersuites	Binary vector
	List of TLS extensions	Binary vector
	Client's public key length	Integer
	Selected cipher suite	Integer
	Selected extensions	Binary vector
	Number of SAN (Subject Alternative Names)	Integer
	Validity (in days)	Integer
	Certificate self-signed or not	Boolean

Όπως φαίνεται και στον παραπάνω πίνακα τα χαρακτηριστικά χωρίζονται σε 3 μεγάλες κατηγορίες: τα μεταδεδομένα των ροών, την κατανομή και τα μεταδεδομένα του πρωτοκόλλου TLS. Στην πρώτη κατηγορία περιλαμβάνονται η θύρα προέλευσης, η θύρα προορισμού, ο αριθμός των εισερχόμενων και εξερχόμενων bytes, ο αριθμός εισερχόμενων και εξερχόμενων πακέτων και η διάρκεια της ροής. Στη δεύτερη κατηγορία περιλαμβάνονται η ακολουθία του μήκους των πακέτων, η ακολουθία του χρόνου των πακέτων και η κατανομή των bytes. Στην τελευταία κατηγορία ανήκουν η λίστα των αλγόριθμων κρυπτογράφησης, η λίστα των επεκτάσεων του TLS πρωτοκόλλου, το μήκος του δημοσίου κλειδιού του πελάτη, ο επιλεγμένος αλγόριθμος κρυπτογράφησης, οι επιλεγμένες επεκτάσεις, ο αριθμός των ιστοσελίδων για τις οποίες είναι έγκυρο το πιστοποιητικό, η εγκυρότητα του πιστοποιητικού σε διάρκεια ημερών και αν το πιστοποιητικό είναι αυτο-υπογεγραμμένο ή όχι. Κάθε ένα από τα χαρακτηριστικά έχει και έναν τύπο που μπορεί να είναι ακέραιος αριθμός, στοχαστικό μητρώο, πίνακας ή δυαδικό διάνυσμα. Επίσης, πρέπει να αναφέρουμε ότι τα μεταδεδομένα ροών αναφέρονται σε χαρακτηριστικά που δεν έχουν απόλυτη σχέση με το πρωτόκολλο TLS, η κατανομή αναφέρεται σε χαρακτηριστικά που προκύπτουν από ανάλυση συχνότητας στα πακέτα και τα μεταδεδομένα του TLS πρωτοκόλλου αναφέρονται σε χαρακτηριστικά που υπάρχουν στη διάρκεια της χειραψίας. Από το σύνολο αυτών των χαρακτηριστικών επιλέξαμε κάποια τα οποία θεωρούμε ότι έχουν κυρίαρχο ρόλο για την αποδοτική ταξινόμηση. Αυτά παρουσιάζονται στον παρακάτω πίνακα.

Category	Feature
Flow Metadata	Source port Destination port Source IP Destination IP
TLS Metadata	List of ciphersuites List of extensions Public key encryption algorithm Validity (not before) Server name

Θεωρούμε σημαντικά χαρακτηριστικά τις θύρες προέλευσης και θύρες προορισμού, καθώς αν η θύρα προορισμού από τη μεριά του πελάτη, δηλαδή η θύρα που ακούει ο διακομιστής, δεν βρίσκεται σε μία συγκεκριμένη λίστα που φαίνεται εδώ [12], τότε είναι ένδειξη ότι υπάρχει κακόβουλο λογισμικό. Κάποιες από τις κυριότερες θύρες που παρουσιάζονται στην προηγούμενη λίστα είναι οι εξής: 443 (HTTPS), 465 (SMTP over TLS), 563 (NNTP over TLS), 636 (LDAPS), 853 (DNS over TLS) κ.α. Επίσης, κάτι αντίστοιχο ισχύει και για τη θύρα προέλευσης του πελάτη, αφού αυτή επιλέγεται αυτόματα από το λειτουργικό σύστημα στο εύρος 49152-65535, όπως φαίνεται και εδώ [13]. Επομένως αν ο πελάτης επιλέξει σκόπιμα κάποια θύρα που δεν βρίσκεται στο συγκεκριμένο εύρος, τότε είναι και αυτό ένδειξη παρουσίας κακόβουλου λογισμικού.

Επιπλέον, οι IP προέλευσης και προορισμού έχουν σπουδαίο ρόλο, αφού υπάρχουν διάφορες λίστες στο διαδίκτυο, οι οποίες περιλαμβάνουν IP με υψηλό ρίσκο. Αυτό σημαίνει ότι έχουν αναφερθεί ύποπτες ενέργειες από χρήστες για τις συγκεκριμένες IP, οπότε αν δούμε κάποια IP από τέτοιου είδους λίστες όπως [14] και [15] μπορούμε να υποθέσουμε κακόβουλη ενέργεια.

Τα προαναφερθέντα χαρακτηριστικά αναφέρονται στα μεταδεδομένα των ροών δεδομένων, οπότε τώρα θα δούμε χαρακτηριστικά από την κατηγορία μεταδεδομένα του TLS πρωτοκόλλου. Ένα από τα πιο σημαντικά χαρακτηριστικά είναι η λίστα των αλγορίθμων κρυπτογράφησης, η οποία παρουσιάζεται αναλυτικά εδώ [16]. Σε αυτή τη λίστα περιλαμβάνονται και αλγόριθμοι που είναι αδύναμοι και άλλοι που είναι ισχυροί, οπότε αν ένας αλγόριθμος είναι αδύναμος, κάτι που μπορούμε να ελέγξουμε και εδώ [17], υποθέτουμε ότι μπορεί να χρησιμοποιείται από κάποιον κακόβουλο. Αυτό οφείλεται είτε στο ότι οι κακόβουλοι δεν γνωρίζουν ποιοι είναι οι ισχυροί αλγόριθμοι είτε στο ότι δεν τους ενδιαφέρει να χρησιμοποιήσουν ισχυρό αλγόριθμο αρκεί τα πακέτα να είναι κρυπτογραφημένα.

Παρατηρήσαμε ότι για τα μη κακόβουλα δεδομένα που έχουμε συλλέξει υπάρχουν 3 διαφορετικοί αλγόριθμοι, ενώ για τα κακόβουλα 38 διαφορετικοί. Επίσης να αναφέρουμε ότι οι αλγόριθμοι έχουν κατακερματισμένη τιμή, οπότε μπορούμε να κάνουμε την αντιστοίχιση αριθμών και κατακερματισμένων τιμών στο Παράρτημα Β – πίνακας ciphersuites. Παρακάτω παρουσιάζουμε κάποιες φωτογραφίες από το script [classify.py](#) με την αναγνώριση των κατακερματισμένων τιμών των αλγορίθμων και τον αριθμό εμφάνισής τους.

- Μη κακόβουλα ciphersuites:

#### Benign ciphersuites:

```
1) 002f00350005000ac013c014c009c00a0032003800130004: 19
2) 003c002f003d00350005000ac027c013c014c02bc023c02cc024c009c00a00400032006a003800130004: 28
3) c02bc02fcc9cca8c02cc030c00ac009c013c01400330039002f0035000a: 26510
```

- Κακόβουλα ciphersuites:

#### Malware ciphersuites:

```
1) 003c002f003d00350005000ac027c013c014c02bc023c02cc024c009c00a00400032006a003800130004: 4766
2) 002f00350005000ac013c014c009c00a0032003800130004: 3890
3) c011c007c00cc0020005c014c00a0039003800880087c00fc00500350084c013c0090033003200450044c00ec004002f0041c012c00800160013c00dc003000a00ff: 8
4) c028c027c014c013009f009e009d009c003d003c0035002fc02cc02bc024c023c00ac009006a004000380032000a001300050004: 81
5) 000400050009002f0035003c003d: 1
6) c014c013c00ac0090035002f00380032000a001300050004: 70
7) c00ac0140088008700390038c00fc00500840035c007c009c011c0130045004400330032c00cc00ec002c0040096004100040005002fc008c01200160013c00dc003feff000a00ff: 814
8) c00ac0140088008700390038c00fc00500840035c007c009c011c01300450044006600330032c00cc00ec002c0040096004100050004002fc008c01200160013c00dc003feff000a: 1
9) cacac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 3
10) 4a4ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 1
11) 7a7ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 4
12) 6a6ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 1
13) dadac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 5
14) c028c027c014c013009f009e00390033009d009c003d003c0035002fc02cc02bc024c023c00ac009006a004000380032000a0013: 2
15) 1a1ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 2
16) 2a2ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 2
17) c02bc02fc00ac009c013c014c012c007c0110033003200450039003800880016002f004100350084000a0005000400ff: 150
18) c02bc02cc02fc030009e009fc009c00ac013c01400330039c007c011009c009d002f0035000500ff: 159
19) c02bc02fc00ac009c013c014c01200330032003900380016002f0035000a00ff: 10
20) c030c02cc028c024c014c00a00a500a300a1009f006b006a006900680039003800370036c032c02
```

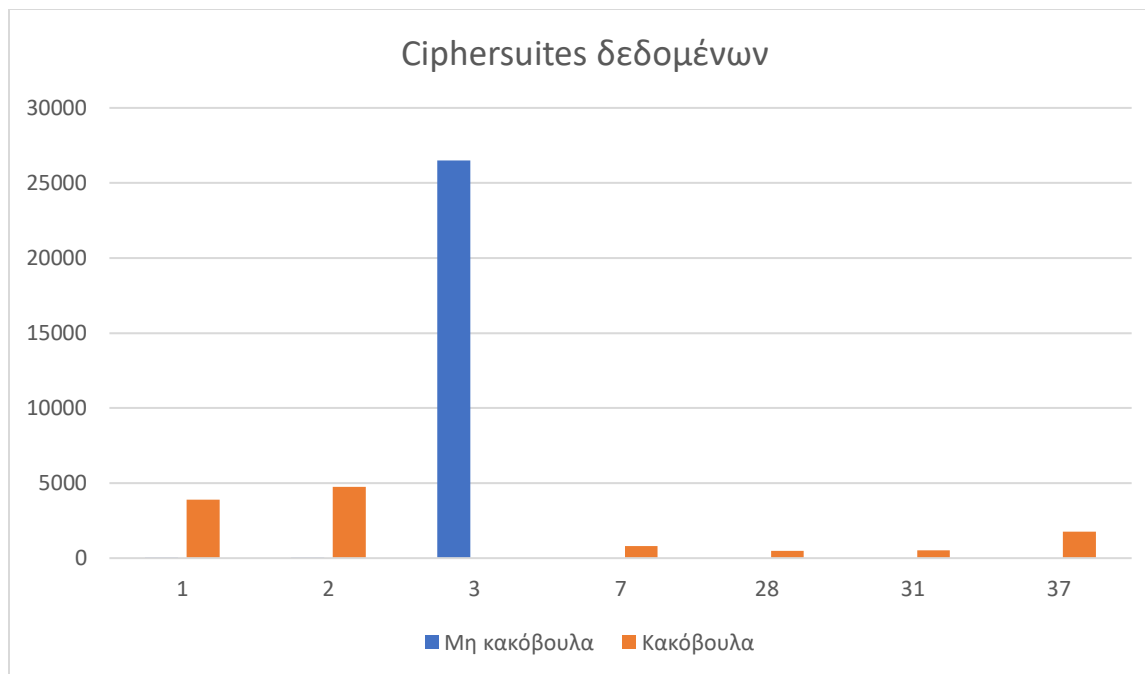


```

21) aaaac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 2
22) eaeac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 2
23) 5a5ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 1
24) 3a3ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: 1
25) cca9cca8cc14cc13c02bc02fc02cc030c009c013c00ac014009c009d002f0035000a: 211
26) 8a8ac02bc02fc02cc030cca9cca8cc13c014009c009d002f0035000a: 12
27) 0a0ac02bc02fc02cc030cca9cca8cc13c014009c009d002f0035000a: 21
28) 0033002f0035000a0005000400ff: 476
29) c0270067009cc011c007c00cc0020005c030c02cc028c024c014c00a00a500a300a1009f006b006
a0069006800390038003700360088008700860085c032c02ec02ac026c00fc005009d003d00350084c0
2fc02bc023c013c00900a400a200a0009e0040003f003e00330032003100300045004400430042c031c
02dc029c025c00ec004003c002f004100ff: 19
30) cca9cca8cc14cc13c02bc02fc00ac014c009c013009c0035002f000a: 138
31) c02cc030009fcca9cca8ccaac02bc02f009ec024c028006bc023c0270067c00ac0140039c009c01
30033009d009c003d003c0035002f00ff: 524
32) 130213031301c02fc02bc030c02c009ec0270067c028006b00a3009fcca9cca8ccaac0afc0adc0a
3c09fc05dc061c057c05300a2c0aec0acc0a2c09ec05cc060c056c052c024006ac0230040c00ac01400
390038c009c01300330032009dc0a1c09dc051009cc0a0c09cc050003d003c0035002f00ff: 44
33) 130113031302c02bc02fcca9cca8c02cc030c00ac009c013c01400330039002f0035000a: 4
34) c030c02cc028c024c014c00a00a500a300a1009f006b006a0069006800390038003700360088008
700860085c032c02ec02ac026c00fc005009d003d00350084c02fc02bc027c023c013c00900a400a200
a0009e00670040003f003e0033003200310030009a0099009800970045004400430042c031c02dc029c
025c00ec004009c003c002f00960041c012c008001600130010000dc00dc003000a0007c011c007c00c
c0020005000400ff: 92
35) c030c02fc028c027c014c013009f009e009d009c003d003c0035002fc02cc02bc024c023c00ac00
9006a004000380032000a0013: 157
36) fafa130113021303c02bc02fc02cc030cca9cca8cc13c014009c009d002f0035: 71
37) c02cc02bc030c02fc024c023c028c027c00ac009c014c013009d009c003d003c0035002f000a:
1749
38) c02fc030c02bc02cca8cca9c013c009c014c00a009c009d002f0035c012000a130113031302:
164

```

Επίσης, παρακάτω παρουσιάζουμε ένα διάγραμμα με τους αλγορίθμους που έχουν χρησιμοποιηθεί περισσότερο. Να σημειωθεί ότι οι κατακερματισμένες τιμές 1 με 5 και 2 με 4 στο Παράρτημα Β – πίνακας ciphersuites είναι ίδιες, οπότε στο παρακάτω διάγραμμα στις θέσεις 1 και 2 απεικονίζεται και ο αριθμός ciphersuites για μη κακόβουλα και για κακόβουλα δεδομένα.



Η λίστα των επεκτάσεων είναι επίσης σημαντική, καθώς παρατηρείται επιλογή λίγων και συγκεκριμένων επεκτάσεων από τους κακόβουλους συγκριτικά με την επιλογή πολλών επεκτάσεων από τους μη κακόβουλους. Η πλήρης λίστα παρουσιάζεται εδώ [18]. Από τα extensions επιλέξαμε τα subject key identifier και authority key identifier. Το subject key identifier, όπως παρουσιάζεται αναλυτικά εδώ [19], παρέχει ένα μέσο αναγνώρισης για πιστοποιητικά που περιέχουν ένα συγκεκριμένο δημόσιο κλειδί και έχει κατακερματισμένη τιμή. Συνδυάζεται με το επόμενο χαρακτηριστικό που επιλέξαμε από τα extensions, το authority key identifier [20], που παρέχει ένα μέσο αναγνώρισης του δημοσίου κλειδιού που αντιστοιχεί στο ιδιωτικό κλειδί που χρησιμοποιείται για την υπογραφή ενός πιστοποιητικού και έχει και αυτό κατακερματισμένη τιμή.

Το επόμενο χαρακτηριστικό είναι το public key encryption algorithm, που παρουσιάζεται αναλυτικά εδώ [21], και παρέχει έναν ασφαλή τρόπο μεταφοράς ενός δημοσίου κλειδιού που θα χρησιμοποιηθεί σε κρυπτογραφία δημοσίου κλειδιού. Πρέπει να ελέγχουμε αν αυτός ο αλγόριθμος που χρησιμοποιείται είναι ισχυρός, αλλιώς αν είναι αδύναμος μπορεί να δημιουργηθεί από κάποιον κακόβουλο δημόσιο και ιδιωτικό κλειδί παριστάνοντας άλλο πρόσωπο.

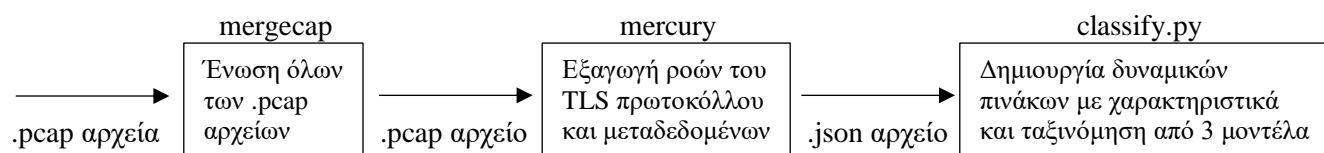
Επιπλέον, αρκετά σημαντικό χαρακτηριστικό είναι το validity, το οποίο ορίζει για πόσο καιρό είναι έγκυρο ένα πιστοποιητικό. Επιλέξαμε για αυτό το χαρακτηριστικό να εστιάζουμε στο να μην το δεχόμαστε αν είναι πριν από κάποια συγκεκριμένη ημερομηνία. Πλέον τα πιστοποιητικά είναι έγκυρα για 397 μέρες, όπως αναφέρεται εδώ [22] και εδώ [23], οπότε όταν περάσει αυτή η περίοδος θεωρούμε μη έγκυρο το πιστοποιητικό και ότι παραπέμπει σε κακόβουλη ενέργεια.

Τέλος, το τελευταίο χαρακτηριστικό που επιλέξαμε είναι το server name, το οποίο φανερώνει από τη μεριά του χρήστη αν η ιστοσελίδα που θέλει να επισκεφτεί είναι έμπιστη. Πιο αναλυτικά για αυτό το χαρακτηριστικό μπορούμε να διαβάσουμε εδώ [24], ενώ έχουμε τη δυνατότητα να παρακολουθούμε έμπιστες ιστοσελίδες εδώ [25].

Αυτά είναι τα χαρακτηριστικά που επιλέξαμε για την αρκετά ικανοποιητική ταξινόμηση, τα αποτελέσματα της οποίας παρουσιάζουμε στο Κεφάλαιο 2.4.4.

### 2.3.2 Διαδικασία επιλογής χαρακτηριστικών

Η διαδικασία που ακολουθήσαμε για την εξαγωγή των επιθυμητών χαρακτηριστικών παρουσιάζεται συνοπτικά στο παρακάτω σχήμα και θα εξηγηθεί αναλυτικά στο Κεφάλαιο 3.2.



### 2.3.3 Στιβαρότητα χαρακτηριστικών

Αναφερόμενοι στη στιβαρότητα των χαρακτηριστικών, εννοούμε κατά πόσο είναι εύκολο ένας επιτιθέμενος να τροποποιήσει κάποιο χαρακτηριστικό για να φαίνεται σαν μη κακόβουλο. Στον παρακάτω πίνακα, που αναφέρεται σε μία σχετική έρευνα [26], φαίνονται τα διάφορα χαρακτηριστικά, τα οποία συνοδεύονται από ένα βαθμό που αντιπροσωπεύει το πόσο εύκολο είναι να τροποποιηθούν για να φαίνεται ότι δεν περιέχουν κακόβουλο λογισμικό. Η βαθμολογία είναι κλιμακωτή από το 1 μέχρι το 3, με το 1 να σημαίνει ότι είναι αρκετά εύκολη η τροποποίηση και το 3 ότι είναι αρκετά δύσκολη και απαιτεί αρκετές γνώσεις.

Feature	Score
Source and Destination ports	1
Bytes in and out	3
Packets in and out	3
Duration	3
Sequence of packet lengths	3
Sequence of packet times	3
Byte distribution	3
Entropy	1
Ciphersuites	2
Extensions	2
Number of extensions	2
Elliptic curve groups	2
Elliptic curve point formats	2
Client's key length	1
Certificate's validity	1
Certificate's number of SAN	1
Self-signed certificate	1

Ένα παράδειγμα για τη βαθμολογία 1 είναι το self-signed certificate, δηλαδή αυτό-υπογεγραμμένο πιστοποιητικό, το οποίο είναι πολύ εύκολο να αποφύγει κάποιος με χρήση πιστοποιητικών από τον οργανισμό Let's Encrypt [27], που παρέχει έγκυρα πιστοποιητικά, τα οποία δεν υποδεικνύουν κακόβουλο λογισμικό. Επομένως, ακόμα και κακόβουλο λογισμικό μπορεί να έχει έγκυρο πιστοποιητικό για 90 μέρες χρησιμοποιώντας δωρεάν τις υπηρεσίες αυτού του οργανισμού.

Ένα παράδειγμα για τη βαθμολογία 3 ανήκει στην κατηγορία των μεταδεδομένων ροής, που περιλαμβάνει bytes in and out και packets in and out, τα οποία δεν ελέγχονται από τους επιτιθέμενους αλλά εξαρτώνται από τα δεδομένα που ανταλλάσσονται, οπότε καταλαβαίνουμε ότι είναι εξαιρετικά δύσκολο να τροποποιήσει κάποιος αυτά τα χαρακτηριστικά.



## 2.4 Ταξινόμηση ροών δεδομένων

### 2.4.1 Επιβλεπόμενη-μη επιβλεπόμενη εκπαίδευση

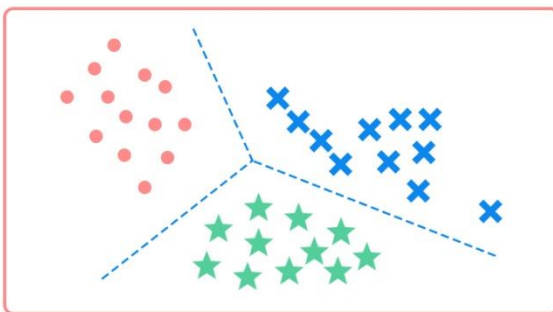
Υπάρχουν 2 είδη εκπαίδευσης στη μηχανική μάθηση ανάλογα με το σκοπό που θέλουμε να πετύχουμε, όπως αναφέρεται και εδώ [28]. Το πρώτο είναι η επιβλεπόμενη εκπαίδευση (supervised learning), στην οποία έχουμε δεδομένα με ετικέτες, ένα μέρος των οποίων εκπαιδεύεται ώστε να γίνει η σωστή ταξινόμηση των υπόλοιπων δεδομένων. Η επιβλεπόμενη εκπαίδευση χωρίζεται σε 2 κατηγορίες ανάλογα το πρόβλημα που θέλουμε να επιλύσουμε. Το πρώτο είναι πρόβλημα ταξινόμησης (classification), δηλαδή προσπαθούμε να ταξινομήσουμε τα δεδομένα στη σωστή κατηγορία με όσο το δυνατόν μεγαλύτερη ακρίβεια και το δεύτερο είναι πρόβλημα παλινδρόμησης (regression), δηλαδή ο αλγόριθμος προσπαθεί να βρει σχέσεις μεταξύ ανεξάρτητων και εξαρτημένων μεταβλητών. Το άλλο είδος εκπαίδευσης είναι η μη επιβλεπόμενη εκπαίδευση (unsupervised learning), στην οποία αναλύονται και ομαδοποιούνται δεδομένα που δεν έχουν ετικέτες. Αυτό το είδος εκπαίδευσης λύνει προβλήματα όπως ομαδοποίηση (clustering) δεδομένων που εμφανίζουν ομοιότητες, συσχέτιση (association) για εύρεση σχέσεων μεταξύ των μεταβλητών στα δεδομένα και μείωση της διαστατικότητας όταν έχουμε τεράστιο αριθμό χαρακτηριστικών.

Τις διαφορές στα δύο είδη εκπαίδευσης μπορούμε να τις αντιληφθούμε και από την παρακάτω εικόνα [29].



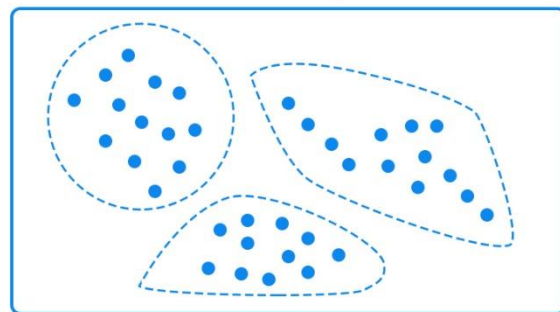
#### Supervised vs. Unsupervised Learning

Classification



Supervised learning

Clustering



Unsupervised learning

Με βάση τα παραπάνω εμείς ενδιαφερόμαστε για επιβλεπόμενη εκπαίδευση, καθώς θέλουμε οι αλγόριθμοι να εκπαιδεύονται κάνοντας επαναλήψεις και να προβλέπουν τη σωστή απάντηση με μεγάλη ακρίβεια. Επιπλέον, το πρόβλημα που έχουμε είναι πρόβλημα ταξινόμησης, αφού θέλουμε να κατηγοριοποιήσουμε τα δεδομένα σε κακόβουλο λογισμικό ή όχι και συγκεκριμένα για ανίχνευση ανωμαλιών μπορούμε να δούμε ότι έχει ξαναχρησιμοποιηθεί επιβλεπόμενη εκπαίδευση όπως σε αυτή την έρευνα [30].

## 2.4.2 Μοντέλα ταξινόμησης

Όπως θα δούμε και παρακάτω δοκιμάσαμε 3 μοντέλα ταξινόμησης, δηλαδή 3 αλγορίθμους τεχνητής νοημοσύνης, οι οποίοι υλοποιήθηκαν σε γλώσσα Python με έκδοση 3.8 και βασίζονται στην βιβλιοθήκη scikit-learn, η οποία είναι ανοιχτού λογισμικού και μπορούμε να βρούμε τον κώδικα της εδώ [31]. Τα 3 μοντέλα είναι τα εξής: Random Forests, KNN (K-Nearest Neighbors) και SVM (Support Vector Machine).

Ένας από τους πιο γνωστούς αλγορίθμους για ταξινόμηση στο data science είναι ο Random Forests, ο οποίος περιλαμβάνει δέντρα αποφάσεων που παράγουν προβλέψεις για την ταξινόμηση των δεδομένων. Όσο αυξάνουμε τον αριθμό των δέντρων αποφάσεων τόσο αυξάνεται η ακρίβεια στα αποτελέσματα, εφόσον βρισκόμαστε μέσα σε επιτρεπτά όρια και δεν οδηγούμαστε σε overfitting των δεδομένων. Overfitting [32] είναι μία προβληματική κατάσταση όπου τα δεδομένα προς εκπαίδευση έχουν εκπαιδευτεί τέλεια, αλλά τα υπόλοιπα που είναι προς ταξινόμηση δεν ταξινομούνται σωστά. Πιο αναλυτικά μπορούμε να μελετήσουμε τη λειτουργία αυτού του αλγορίθμου εδώ [33] και εδώ [34].

Ο επόμενος αλγόριθμος είναι ο K-Nearest Neighbors (KNN), οποίος ταξινομεί τα δεδομένα με βάση την ομοιότητα που εμφανίζουν. Επιλέγουμε εμείς τον αριθμό των κοντινότερων γειτόνων που επιθυμούμε και ο αλγόριθμος με κάποιες μαθηματικές πράξεις υπολογίζει την απόσταση μεταξύ των σημείων και στη συνέχεια κάνει την ταξινόμηση. Για περισσότερες λεπτομέρειες μπορούμε να διαβάσουμε εδώ [35].

Ο τελευταίος αλγόριθμος που δοκιμάσαμε είναι ο Support Vector Machine (SVM), ο οποίος αναζητάει το υπερεπίπεδο σε έναν N-διάστατο χώρο (το N είναι ο αριθμός των χαρακτηριστικών που έχουμε επιλέξει) όπου τα σημεία είναι πλήρως διαχωρισμένα και ταξινομημένα. Οι μαθηματικές πράξεις αυτού του αλγορίθμου είναι ο λεγόμενος πυρήνας, που μπορεί να είναι linear, nonlinear, radial basis function (RBF) ή sigmoid. Περισσότερες πληροφορίες για τον πυρήνα μπορούμε να βρούμε εδώ [36] και γενικότερα για το SVM εδώ [37].

### 2.4.3 Μετρικές απόδοσης μοντέλων

Ο στόχος μας είναι, μετά την εξαγωγή χαρακτηριστικών και είσοδό τους στους αλγορίθμους τεχνητής νοημοσύνης, η αξιολόγηση των αποτελεσμάτων. Αυτό μπορεί να γίνει με διάφορες μετρικές για μια σφαιρική αξιολόγηση. Οι μετρικές που θα χρησιμοποιήσουμε είναι accuracy, precision, recall και f1-score. Επίσης, τονίζουμε ότι η ταξινόμηση που κάνουμε είναι δυαδική, δηλαδή υπάρχουν 2 δυνατές τιμές, το 0 και το 1. Το 1 στην περίπτωση μας σημαίνει παρουσία κακόβουλο λογισμικού και το 0 απουσία. Το 1 αναφέρεται στο Positive, δηλαδή έχουμε κακόβουλο λογισμικό, και το 0 στο Negative, δηλαδή δεν έχουμε κακόβουλο λογισμικό.

Πριν αναλύσουμε αυτές τις μετρικές πρέπει να ορίσουμε κάποιες ακόμα έννοιες, που βρίσκουμε σε αυτό το άρθρο [38]. Αυτές είναι: True Positives (TP), True Negatives (TN), False Positives (FP) και False Negatives (FN). Τα TP είναι όταν το δεδομένο μας είναι κακόβουλο και η πρόβλεψη που κάνουμε είναι επίσης κακόβουλο λογισμικό. Τα TN είναι όταν το δεδομένο μας είναι μη κακόβουλο και η πρόβλεψη που κάνουμε είναι επίσης μη κακόβουλο. Τα FP είναι όταν το δεδομένο μας είναι μη κακόβουλο αλλά η πρόβλεψή μας είναι κακόβουλο. Τα FN είναι όταν το δεδομένο μας είναι κακόβουλο αλλά η πρόβλεψή μας είναι μη κακόβουλο. Παρακάτω παραθέτουμε μία φωτογραφία, όπου βλέπουμε σε έναν πίνακα πως χωρίζονται οι προηγούμενες έννοιες.

		Actual	
		Positives(1)	Negatives(0)
Predicted	Positives(1)	TP	FP
	Negatives(0)	FN	TN

Γνωρίζοντας πλέον τις παραπάνω έννοιες θα ορίσουμε τις μετρικές, όπως περιγράφονται και στο επόμενο άρθρο [39].

- Accuracy είναι ο αριθμός των σωστών προβλέψεων στο σύνολο των προβλέψεων.
- Precision είναι ο αριθμός των σωστών θετικών προβλέψεων στο σύνολο των θετικών προβλέψεων.
- Recall είναι ο αριθμός των σωστών θετικών προβλέψεων στο σύνολο των θετικών δεδομένων.
- F1-score είναι το ημίθροισμα των Precision και Recall, δηλαδή ένας αριθμός που εκπροσωπεί τις 2 αυτές μετρικές και είναι ο μέσος όρος τους.

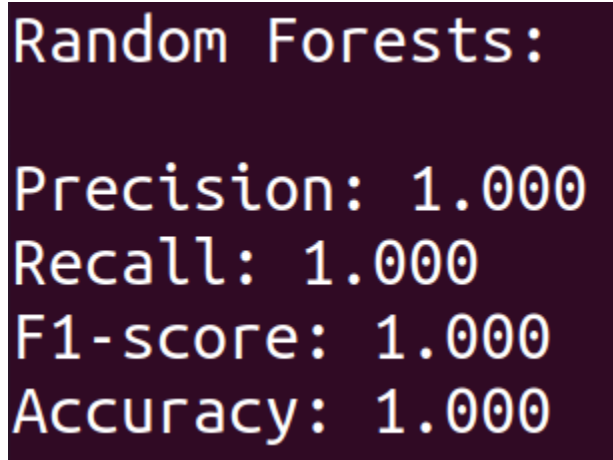
Παρακάτω βλέπουμε πως περιγράφονται μαθηματικά αυτές οι μετρικές βασισμένες στις έννοιες που περιγράψαμε προηγουμένως [40].

Metric	Formula
True positive rate, recall	$\frac{TP}{TP+FN}$
False positive rate	$\frac{FP}{FP+TN}$
Precision	$\frac{TP}{TP+FP}$
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$
F-measure	$\frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$

## 2.4.4 Αποτελέσματα

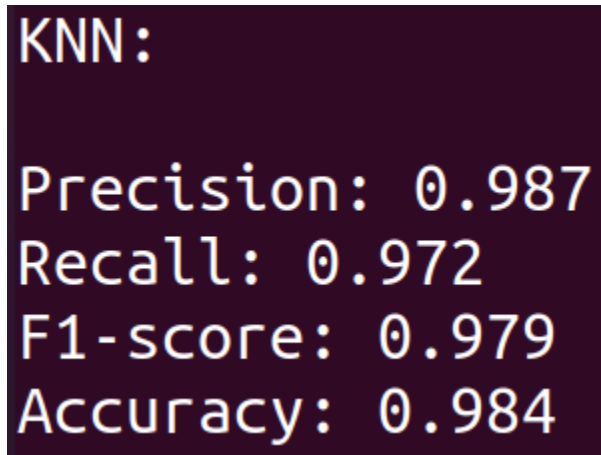
Μέχρι στιγμής έχουμε δει τη συλλογή δεδομένων στο Κεφάλαιο 2.2, στη συνέχεια την επιλογή των χαρακτηριστικών που μας ενδιαφέρουν Κεφάλαιο 2.3.1, τους αλγόριθμους τεχνητής νοημοσύνης που θα χρησιμοποιήσουμε στο Κεφάλαιο 2.4.2 για την ταξινόμηση και τέλος τις μετρικές με τις οποίες θα τους αξιολογήσουμε στο Κεφάλαιο 2.4.3. Επομένως, σε αυτό το Κεφάλαιο θα παρουσιάσουμε τα αποτελέσματα για κάθε έναν αλγόριθμο και θα τους συγκρίνουμε ως προς την καλύτερη επίδοση.

Τα καλύτερα αποτελέσματα παρουσίασε ο αλγόριθμος Random Forests, ο οποίος με 200 δέντρα αποφάσεων έδωσε τα αποτελέσματα της παρακάτω εικόνας. Τα αποτελέσματα είναι τέλεια, αφού έχουμε σε όλες τις μετρικές την τιμή 1, οπότε δεν έγινε ούτε ένα λάθος στην ταξινόμηση των δεδομένων.



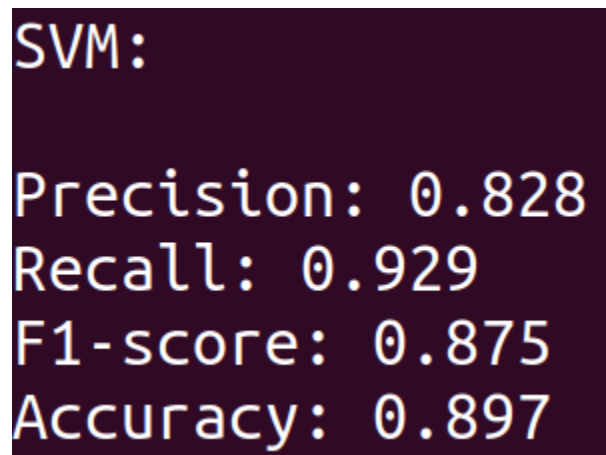
```
Random Forests:
Precision: 1.000
Recall: 1.000
F1-score: 1.000
Accuracy: 1.000
```

Αμέσως επόμενος είναι ο αλγόριθμος K-Nearest Neighbors (KNN) χρησιμοποιώντας σαν παράμετρο 3 κοντινότερους γείτονες, τα αποτελέσματα του οποίου παρουσιάζονται στην παρακάτω εικόνα και είναι πολύ ικανοποιητικά επίσης.



```
KNN:  
  
Precision: 0.987  
Recall: 0.972  
F1-score: 0.979  
Accuracy: 0.984
```

Τέλος, ο αλγόριθμος Support Vector Machine έδωσε τα «χειρότερα» αποτελέσματα από τους 3, τα οποία παρ' όλα αυτά είναι ικανοποιητικά, και αυτό συμβαίνει διότι παρατηρούμε μία μικρή πτώση στο precision. Αυτός ο αλγόριθμος παίρνει σαν παράμετρο τον πυρήνα όπως είδαμε στο Κεφάλαιο 2.4.2, που είναι στην περίπτωση μας ο rbf (Radial Basis Function). Αξίζει να αναφέρουμε ότι δοκιμάστηκε και ο linear πυρήνας, ο οποίος δεν σύγκλινε καθόλου και δεν παίρναμε αποτελέσματα ακόμα και μετά από πολλές ώρες.



```
SVM:  
  
Precision: 0.828  
Recall: 0.929  
F1-score: 0.875  
Accuracy: 0.897
```

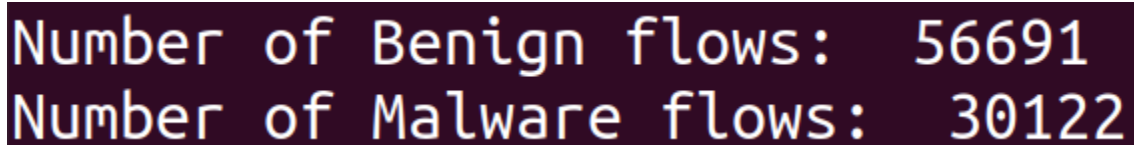
Τα αποτελέσματα και από τους 3 αλγορίθμους είναι αρκετά ικανοποιητικά όπως είδαμε παραπάνω, οπότε αυτό σημαίνει ότι τα χαρακτηριστικά που επιλέξαμε για να ταξινομούνται τα δεδομένα είναι βασικά και μπορούν να οδηγήσουν σε καλή ταξινόμηση. Επίσης, θεωρούμε δεδομένη την αξιοπιστία των δεδομένων από τις πηγές που επιλέχθηκαν, δηλαδή ότι τα δεδομένα που αναφέρεται ότι περιέχουν κακόβουλο λογισμικό τα θεωρούμε σίγουρα κακόβουλα και αντίστοιχα για τα μη κακόβουλα.

Στο Κεφάλαιο 3 ακολουθεί η αναλυτική διαδικασία για μετατροπή .pcap αρχείων σε .json, εξαγωγή χαρακτηριστικών και υλοποίηση αλγορίθμων και μετρικών αξιολόγησης.

## 3 Ανάλυση πειράματος

### 3.1 Διαδικασία επιλογής δεδομένων

Η επιλογή δεδομένων περιγράφηκε και στο Κεφάλαιο 2.2 Συλλογή Δεδομένων, όπου αναφέρθηκε η πηγή των κακόβουλων .pcap αρχείων [10] και η πηγή των μη κακόβουλων .pcap αρχείων [9]. Επιλέξαμε με τέτοιο τρόπο τα αρχεία, ώστε το μέγεθος του συνόλου των κακόβουλων (2.5G) να είναι περίπου ίσο με το μέγεθος του συνόλου των μη κακόβουλων (2.3G) για να υπάρχει ισορροπία. Παρ' όλα αυτά υπάρχει μια διαφορά στην εξαγωγή των ροών δεδομένων, αφού τα κακόβουλα δημιούργησαν 30122 ροές (34,7%), ενώ τα μη κακόβουλα 56691 (65,3%). Αυτό επιβεβαιώνεται από την παρακάτω φωτογραφία.

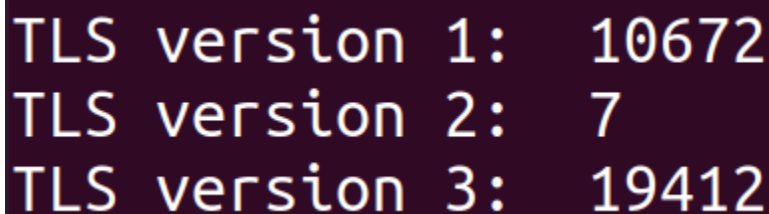


```
Number of Benign flows: 56691
Number of Malware flows: 30122
```

Η διαφορά αυτή οφείλεται στο ότι στη φύση γενικά δημιουργείται περισσότερο μη κακόβουλο λογισμικό σε σχέση με το κακόβουλο, αν και το κακόβουλο λογισμικό το καταγράφουμε συχνότερα για να ερμηνεύουμε τη συμπεριφορά του.

Μία ακόμα διαφορά βρίσκεται στη χρονολογία εμφάνισης κακόβουλων και μη κακόβουλων δεδομένων, αφού όπως έχουμε αναφέρει το μη κακόβουλο λογισμικό δεν καταγράφεται συχνά, οπότε τα μη κακόβουλα δεδομένα μας σχετικά με το TLS πρωτόκολλο είναι από τη χρονιά 2017. Αντίθετα, τα κακόβουλα δεδομένα μας είναι μοιρασμένα στις χρονιές από το 2016 μέχρι και το 2021.

Τέλος, είναι σημαντικό να αναφέρουμε το ποσοστό των ροών δεδομένων σχετικά με τις διάφορες εκδόσεις του πρωτοκόλλου TLS. Ο αριθμός των ροών για το κάθε πρωτόκολλο παρουσιάζεται στην παρακάτω εικόνα και τα ποσοστά είναι αντίστοιχα 35,47%, 0,02% και 64,51%.



```
TLS version 1: 10672
TLS version 2: 7
TLS version 3: 19412
```



## 3.2 Διαδικασία εξαγωγής χαρακτηριστικών

Αρχικά, έχουμε χωρίσει σε 2 ξεχωριστούς φακέλους όλα τα κακόβουλα .pcap αρχεία από τα μη κακόβουλα .pcap αρχεία. Τα βήματα που ακολουθούν έχουν γίνει 2 φορές, μία για τον κάθε φάκελο, αλλά είναι τα ίδια ακριβώς. Στη συνέχεια χρησιμοποιήσαμε το εργαλείο merg pcap, όπως αναφέρεται και εδώ [41], με σκοπό να ενωθούν όλα τα .pcap σε ένα ενιαίο (1 ενιαίο αρχείο για τα κακόβουλα και 1 ενιαίο αρχείο για τα μη κακόβουλα). Περισσότερες λεπτομέρειες σχετικά με το εργαλείο merg pcap μπορούν να βρεθούν στην ιστοσελίδα του [42]. Η παράμετρος -F καθορίζει τη μορφή του αρχείου εξόδου, δηλαδή .pcap, επειδή η προκαθορισμένη μορφή του εργαλείου είναι η .pcapng. Η παράμετρος -w καθορίζει το αρχείο εξόδου (ben\_output.pcap και mal\_output.pcap) που θα παραχθεί από την ένωση όλων των .pcap αρχείων, τα οποία επιλέγονται με τον αστερίσκο ( \* ), δηλαδή οποιοδήποτε όνομα αρχείου βρίσκεται στο συγκεκριμένο φάκελο που έχει επέκταση .pcap. Παρακάτω φαίνονται 2 φωτογραφίες με τις εντολές του merg pcap για τους 2 φακέλους. Οι εντολές είναι:

- Μη κακόβουλο: `merg pcap -F pcap -w ben_output.pcap *.pcap`

```
george@ubuntu:~/Downloads/thesis/benign_dataset$ merg pcap -F pcap -w ben_output.pcap *.pcap
george@ubuntu:~/Downloads/thesis/benign_dataset$
george@ubuntu:~/Downloads/thesis/benign_dataset$
```

- Κακόβουλο: `merg pcap -F pcap -w mal_output.pcap *.pcap`

```
george@ubuntu:~/Downloads/thesis/malware_dataset$ merg pcap -F pcap -w mal_output.pcap *.pcap
george@ubuntu:~/Downloads/thesis/malware_dataset$
george@ubuntu:~/Downloads/thesis/malware_dataset$
```

Αυτή τη στιγμή έχουν δημιουργηθεί τα 2 αρχεία ben\_output.pcap και mal\_output.pcap, τα οποία στη συνέχεια μπαίνουν σαν είσοδος στο εργαλείο mercury της Cisco [43]. Το εργαλείο mercury είτε κάνει καταγραφή πακέτων μέσα από κάποιο πρόγραμμα όπως το Wireshark είτε παίρνει ένα αρχείο .pcap, όπως στην περίπτωση μας, και αναλύει τη δικτυακή κίνηση των πακέτων και τα μεταδεδομένα τους. Επίσης, τα μετατρέπει από .pcap αρχεία σε .json [44] για να μπορούμε να τα χειριστούμε εμείς αργότερα χάρη στην ειδική δομή ζευγαριού που έχουν, κλειδιού και τιμής. Επομένως, για τις εντολές του εργαλείου mercury χρησιμοποιούμε τις εξής παραμέτρους: -r για να διαβάσουμε το αρχείο εισόδου (ben\_output.pcap και mal\_output.pcap), --select=tls για να επιλέξουμε πακέτα σχετικά με το πρωτόκολλο TLS, όπως clientHello, serverHello και certificates, --metadata για να εξάγουμε διάφορα μεταδεδομένα που θα μας χρειαστούν και --certs-json για να είναι η έξοδος σε μορφή json. Τέλος, την έξοδο του mercury τη βάζουμε στα αρχεία ben\_output.json και mal\_output.json αντίστοιχα με το σύμβολο «μεγαλύτερο», δηλαδή ( > ). Παρακάτω φαίνονται 2 φωτογραφίες με τις εντολές του mercury για τους 2 φακέλους. Οι εντολές είναι:

- Μη κακόβουλο: `mercury -r ben_output.pcap --select=tls --metadata --certs-json > ben_output.json`

```
george@ubuntu:~/Downloads/thesis/benign_dataset$ mercury -r ben_output.pcap --select=tls --metadata --certs-json > ben_output.json
george@ubuntu:~/Downloads/thesis/benign_dataset$
george@ubuntu:~/Downloads/thesis/benign_dataset$
```

- Κακόβουλο: `mercury -r mal_output.pcap --select=tls --metadata --certs-json > mal_output.json`

```
george@ubuntu:~/Downloads/thesis/malware_dataset$ mercury -r mal_output.pcap --select=tls --metadata --certs-json > mal_output.json
george@ubuntu:~/Downloads/thesis/malware_dataset$
george@ubuntu:~/Downloads/thesis/malware_dataset$
```

Επομένως, εδώ έχουν δημιουργηθεί τα `ben_output.json` και `mal_output.json` στον μη κακόβουλο και κακόβουλο φάκελο αντίστοιχα. Τα αρχεία αυτά έχουν ένα πρόβλημα, καθώς δεν είναι ακριβώς σε μορφή json για να μπορέσουμε να τα χειριστούμε. Για την ακρίβεια λείπουν οι αγκύλες στην αρχή και στο τέλος του κάθε αρχείου, ( [ ) και ( ] ), και επίσης οι γραμμές δεν χωρίζονται με κόμμα ( , ), όπως φάνηκε και σε έναν online JSON editor [45]. Για αυτό το λόγο φτιάξαμε ένα script (`final_json.py`) γραμμένο σε python3, το οποίο βάζει τις αγκύλες στην αρχή και στο τέλος και χωρίζει την κάθε γραμμή με κόμμα. Αυτή η διαδικασία πάλι γίνεται 2 φορές, μία για τον κάθε φάκελο. Ο κώδικας του script παρουσιάζεται στο Παράρτημα Α - κώδικας. Οι εντολές για να τρέξουμε αυτό το script φαίνονται στις φωτογραφίες που ακολουθούν και είναι οι εξής:

- Μη κακόβουλο: `python3 final_json.py`

```
george@ubuntu:~/Downloads/thesis/benign_dataset$ python3 final_json.py
george@ubuntu:~/Downloads/thesis/benign_dataset$
george@ubuntu:~/Downloads/thesis/benign_dataset$
```

- Κακόβουλο: `python3 final_json.py`

```
george@ubuntu:~/Downloads/thesis/malware_dataset$ python3 final_json.py
george@ubuntu:~/Downloads/thesis/malware_dataset$
george@ubuntu:~/Downloads/thesis/malware_dataset$
```

Αυτή τη στιγμή έχουν δημιουργηθεί 2 νέα αρχεία, τα `ben_final.json` και `mal_final.json`, τα οποία είναι σε κανονική μορφή json, οπότε στη συνέχεια με το script `classify.py` εξάγουμε τα χαρακτηριστικά που μας ενδιαφέρουν και βρίσκονται στο Κεφάλαιο 2.3.1 Επιλογή υποσυνόλου χαρακτηριστικών. Το script `classify.py` μπορεί να χωριστεί σε 2 μέρη, το πρώτο κάνει την εξαγωγή των χαρακτηριστικών και τα βάζει σε δυναμικούς πίνακες και το δεύτερο παίρνει τα χαρακτηριστικά από τους πίνακες και τα χωρίζει σε 25% για εκπαίδευση των αλγορίθμων και 75%

για την αξιολόγηση της απόδοσης σε δεδομένα που δεν έχουν ξαναδεί οι αλγόριθμοι. Το δεύτερο μέρος θα μας απασχολήσει στο επόμενο Κεφάλαιο 3.3 Υλοποίηση αλγορίθμων και ταξινόμηση. Επίσης, να αναφέρουμε ότι ο κώδικας και αυτού του script θα βρίσκεται στο Παράρτημα Α - κώδικας.

Αρχικά, το script classify.py τρέχει μία φορά, καθώς βρίσκεται εκτός των φακέλων με τα κακόβουλα και μη αρχεία. Η εντολή είναι η εξής όπως φαίνεται και στην παρακάτω φωτογραφία:

- `python3 classify.py`



```
george@ubuntu:~/Downloads/thesis$ python3 classify.py
```

Ξεκινάει διαβάζοντας το τελικό αρχείο για τα μη κακόβουλα δεδομένα (ben\_final.json) και διαχωρίζει την κάθε γραμμή του αρχείου είτε σε πληροφορία πελάτη είτε σε πληροφορία διακομιστή. Αυτό κρίνεται ανάλογα το περιεχόμενο της γραμμής, δηλαδή αν είναι αίτηση που έγινε από τον πελάτη ή απάντηση που δόθηκε από τον διακομιστή. Τα χαρακτηριστικά: θύρες προέλευσης και προορισμού και IP προέλευσης και προορισμού (src\_port, dst\_port, src\_ip και dst\_ip) υπάρχουν και στον πελάτη και στον διακομιστή, αλλά κάποια άλλα όπως οι αλγόριθμοι κρυπτογράφησης και το όνομα του διακομιστή (ciphersuites και server name) υπάρχουν μόνο στον πελάτη. Αντίστοιχα, χαρακτηριστικά όπως τα subject key identifier και authority key identifier, που βρίσκονται στα extensions, η εγκυρότητα του πιστοποιητικού (validity not before) και ο αλγόριθμος κρυπτογράφησης δημοσίου κλειδιού (public key encryption algorithm) βρίσκονται μόνο στον διακομιστή. Για αυτό το λόγο, όταν έχουμε π.χ. πληροφορία από διακομιστή στις στήλες με χαρακτηριστικά ciphersuites και server name βάζουμε τιμή «unknown», αφού αυτά βρίσκονται μόνο σε πληροφορία πελάτη. Αντίστοιχα λειτουργούμε και όταν έχουμε πληροφορία από πελάτη βάζοντας στις στήλες με χαρακτηριστικά του διακομιστή τιμή «unknown». Αυτό γίνεται επειδή τοποθετούμε τα χαρακτηριστικά σε δυναμικούς πίνακες, οπότε αυτοί πρέπει να έχουν ίδιο αριθμό θέσεων όταν θα τους χρησιμοποιήσουν οι αλγόριθμοι τεχνητής νοημοσύνης. Μία άλλη προσέγγιση θα ήταν να βάζαμε το μέσο όρο του κάθε χαρακτηριστικού στα πεδία που δεν έχουμε τιμές. Αφού τελειώσει αυτή η διαδικασία για το μη κακόβουλο φάκελο γίνεται ακριβώς η ίδια και για τον κακόβουλο με το αρχείο mal\_final.json.

Στη συνέχεια, επειδή οι αλγόριθμοι τεχνητής νοημοσύνης μπορούν να χειριστούν μόνο αριθμητικά δεδομένα και όχι αλφαριθμητικά όπως στην περίπτωση του server name και σε άλλες, τότε κάνουμε μία κωδικοποίηση (fit transformation) στα δεδομένα πριν περαστούν στους τελικούς πίνακες. Η κωδικοποίηση αυτή γίνεται από μία συνάρτηση που χρησιμοποιούμε, τη LabelEncoder(), η οποία με την fit\_transform παράγει τα τελικά αριθμητικά δεδομένα μέσα στους πίνακες. Εδώ ολοκληρώνεται η λειτουργία του πρώτου μέρους του script classify.py, οπότε έχουν παραχθεί πίνακες ίσου μεγέθους με τα χαρακτηριστικά μέσα κωδικοποιημένα και είναι έτοιμοι να χρησιμοποιηθούν από τους αλγόριθμους τεχνητής νοημοσύνης με τον τρόπο που παρουσιάζεται στο επόμενο Κεφάλαιο.

### 3.3 Υλοποίηση αλγορίθμων και ταξινόμηση

Το δεύτερο μέρος του script `classify.py`, όπως αναφέρθηκε και προηγουμένως, παίρνει τις τιμές των χαρακτηριστικών από τους δυναμικούς πίνακες και τις διαχωρίζει σε ποσοστό 25% και 75%. Το ποσοστό 25% χρησιμοποιείται για την εκπαίδευση των αλγορίθμων τεχνητής νοημοσύνης. Αυτό σημαίνει ότι οι αλγόριθμοι έχουν μάθει ότι ένα μέρος του 25% είναι κακόβουλα και ένα άλλο μέρος είναι μη κακόβουλα δεδομένα. Επομένως, το υπόλοιπο 75% αναλύεται και όταν κάποιο δεδομένο εμφανίζει ομοιότητες με τα κακόβουλα οι αλγόριθμοι το ταξινομούν σαν κακόβουλο αλλιώς αν εμφανίζει ομοιότητες με μη κακόβουλα το ταξινομούν σαν μη κακόβουλο. Αυτή είναι γενικά η εκπαίδευση για τα μοντέλα της μηχανικής μάθησης, αλλά ο κάθε αλγόριθμος που χρησιμοποιούμε έχει τη δική του λειτουργία όπως π.χ. ο Random Forests χρησιμοποιεί ένα πλήθος δέντρων αποφάσεων για την ταξινόμηση, ο K-Nearest Neighbors χρησιμοποιεί πλήθος κοντινότερων γειτόνων και ο Support Vector Machine παίρνει σαν παράμετρο τον πυρήνα, Radial Basis Function στην περίπτωση μας. Η λειτουργία των αλγορίθμων που χρησιμοποιούμε περιγράφηκε στο Κεφάλαιο 2.4.2 Μοντέλα ταξινόμησης. Στη συνέχεια ο κάθε αλγόριθμος χρησιμοποιεί τις μετρικές `accuracy`, `precision`, `recall` και `f1-score` (Κεφάλαιο 2.4.3), που έχουν δημιουργηθεί από τη βιβλιοθήκη `scikit-learn` και με βάση προβλέψεις που κάνουν παίρνουμε τελικά τα αποτελέσματα του Κεφαλαίου 2.4.4.

Τέλος, οι εντολές συγκεντρωμένες από τη στιγμή που έχουμε τους 2 φακέλους με τα `.pcap` αρχεία μέχρι και την ταξινόμηση σε κακόβουλο λογισμικό ή όχι παρουσιάζονται παρακάτω:

- 1) `mergecap -F pcap -w ben_output.pcap *.pcap`  
2) `mergecap -F pcap -w mal_output.pcap *.pcap`
- 1) `mercury -r ben_output.pcap --select=tls --metadata --certs-json > ben_output.json`  
2) `mercury -r mal_output.pcap --select=tls --metadata --certs-json > mal_output.json`
- 1) `python3 final_json.py`  
2) `python3 final_json.py`
- `python3 classify.py`

## 4 Συμπεράσματα

### 4.1 Σύνοψη

Συμπεραίνοντας, μπορούμε να καταλάβουμε πόσο σημαντικό είναι ένα μοντέλο σαν αυτό που παρουσιάστηκε σε αυτή τη διπλωματική εργασία, το οποίο δεν παραβιάζει τα προσωπικά δεδομένα των χρηστών και αυξάνει κατακόρυφα την ταχύτητα ανίχνευσης κακόβουλου λογισμικού, αφού δεν κάνει αποκρυπτογράφηση του κάθε πακέτου ξεχωριστά, αλλά βασίζεται σε μεταδεδομένα και χαρακτηριστικά που έχουμε επιλέξει εμείς. Σε αντίθεση με αυτό το μοντέλο, άλλα παλαιότερα αποκρυπτογραφούν τα πακέτα, με αποτέλεσμα να φανερώνονται προσωπικά δεδομένα όπως κωδικοί πρόσβασης, κάτι που δεν είναι επιθυμητό, ενώ υπάρχει και καθυστέρηση στην επικοινωνία πελατών και διακομιστών, κάτι που επίσης δεν είναι επιθυμητό σε ένα παραγωγικό περιβάλλον.

Μαζί με τα προηγούμενα σημαντικά πλεονεκτήματα, παρατηρήσαμε τις εξαιρετικές αποδόσεις του μοντέλου μας, το οποίο με τον αλγόριθμο Random Forests κάνει τέλεια ταξινόμηση, ενώ με τους άλλους 2 (KNN και SVM) οδηγεί σε εξίσου ικανοποιητικά αποτελέσματα. Τα αποτελέσματα συγκεντρωτικά μπορούμε να τα δούμε, όπως παρουσιάστηκαν στο Κεφάλαιο 2.4.4 Αποτελέσματα, στον παρακάτω πίνακα:

	Accuracy	Precision	Recall	F1-score
Random Forests	1.000	1.000	1.000	1.000
K-Nearest Neighbors	0.984	0.987	0.972	0.979
Support Vector Machine	0.897	0.828	0.929	0.875

Επιπλέον, τα αποτελέσματα είναι αρκετά ικανοποιητικά, καθώς αναφερόμαστε σε δεδομένα που δεν έχουν ξαναδεί οι αλγόριθμοι, καθιστώντας το μοντέλο μας κατάλληλο για αντικατάσταση των παραδοσιακών IDS (Intrusion Detection Systems) συστημάτων σε παραγωγικά περιβάλλοντα.

## 4.2 Μελλοντική εργασία

Το μοντέλο αυτό σίγουρα επιδέχεται βελτιώσεις, καθώς οι συγγραφείς κακόβουλου λογισμικού δημιουργούν συνεχώς καινούριο λογισμικό, οπότε μοντέλα σαν το δικό μας πρέπει να είναι συνεχώς ενημερωμένα στις νέες τάσεις και να έχουν τη δυνατότητα να αναγνωρίζουν ακόμα και τις νεότερες εκδόσεις κακόβουλου λογισμικού.

Επίσης, η χρήση του πρωτοκόλλου TLSv1.3 συνεχώς αυξάνεται από τις ιστοσελίδες, οπότε θα αρχίσει να αυξάνεται ακόμα περισσότερο η εμφάνιση κακόβουλου λογισμικού που θα χρησιμοποιεί το πρωτόκολλο αυτό.

Αναφέραμε, επίσης, αντίστοιχες έρευνες που έχουν γίνει στον ίδιο τομέα εδώ [11] και εδώ [26] με ένα μεγαλύτερο σύνολο χαρακτηριστικών, οπότε μελλοντικά θα μπορούσαμε να διευρύνουμε το σύνολο των χαρακτηριστικών μας.

Ακόμα, οι αλγόριθμοι τεχνητής νοημοσύνης εξελίσσονται συνεχώς και δημιουργούνται νέοι, οπότε μελλοντικά μπορούμε να έχουμε ακόμα καλύτερα αποτελέσματα και σε λιγότερο χρόνο.

Τέλος, όπως αναφέρθηκε και προηγουμένως το μοντέλο αυτό μπορεί να εφαρμοστεί και σε ένα περιβάλλον εταιρείας κάνοντας ζωντανή ανίχνευση κακόβουλου λογισμικού στη δικτυακή κίνηση των εργαζομένων χωρίς να τους καθυστερεί στην εργασία τους.

## Παράρτημα Α - κώδικας

Έχουν δημιουργηθεί 3 scripts συνολικά. Τα 2 είναι ίδια, αλλά χρησιμοποιούνται σε διαφορετικούς φακέλους, το πρώτο στον φάκελο με τα μη κακόβουλα δεδομένα και το δεύτερο στον φάκελο με τα κακόβουλα δεδομένα. Η λειτουργία τους είναι να φέρουν σε σωστή μορφή τα .json αρχεία, διορθώνοντας τα προβλήματα που αναφέρθηκαν στο Κεφάλαιο 3.2 Διαδικασία εξαγωγής χαρακτηριστικών.

Ο κώδικας του πρώτου script [final\\_json.py](#) στον φάκελο με τα μη κακόβουλα δεδομένα παρουσιάζεται παρακάτω:

```
import json

with open('ben_output.json', 'r') as test:
    lineList = test.readlines()

with open('ben_output.json', 'r') as istr:
    i=0
    with open('ben_final.json', 'w') as ostr:
        print('[',file=ostr)
        for i, line in enumerate(istr):
            i=i+1
            line = line.rstrip('\n')
            if i!=len(lineList):
                line += ','
                print(line, file=ostr)
            else:
                line += ']'
                print(line, file=ostr)

with open('ben_final.json', 'r') as file:
    data = file.read().replace('\n', '')
```

Επίσης, ο κώδικας του δεύτερου script [final\\_json.py](#) στον φάκελο με τα κακόβουλα δεδομένα παρουσιάζεται παρακάτω:

```
import json

with open('mal_output.json', 'r') as test:
    lineList = test.readlines()

with open('mal_output.json', 'r') as istr:
    i=0
    with open('mal_final.json', 'w') as ostr:
        print('[',file=ostr)
        for i, line in enumerate(istr):
            i=i+1
            line = line.rstrip('\n')
            if i!=len(lineList):
                line += ','
                print(line, file=ostr)
            else:
                line += ']'
                print(line, file=ostr)

with open('mal_final.json', 'r') as file:
    data = file.read().replace('\n', '')
```

Τέλος, ο κώδικας του script [classify.py](#) που κάνει την εξαγωγή των χαρακτηριστικών και την ταξινόμηση παρουσιάζεται παρακάτω:

```
import json

from sklearn.preprocessing import LabelEncoder

import numpy as np

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import train_test_split

from sklearn.metrics import precision_score

from sklearn.metrics import recall_score

from sklearn.metrics import f1_score

from sklearn.metrics import accuracy_score

from sklearn.neighbors import KNeighborsClassifier
```



[illegible]

```

temp_list.append(mydata[i]['tls']['client']['server_name'])

temp_list.append('unknown')
temp_list.append('unknown')
temp_list.append('unknown')
temp_list.append('unknown')

temp_list.append('neg')

final_data.append(temp_list)

if 'server' in mydata[i]['tls'] and 'certs' in
mydata[i]['tls']['server']:
    temp_list=[]

    temp_list.append(mydata[i]['src_port'])

    temp_list.append(mydata[i]['dst_port'])

    temp_list.append(mydata[i]['src_ip'])

    temp_list.append(mydata[i]['dst_ip'])

    temp_list.append('unknown')

    temp_list.append('unknown')

    if 'subject_public_key_info' in
mydata[i]['tls']['server']['certs'][0]['cert']:
temp_list.append(mydata[i]['tls']['server']['certs'][0]['cert']['subject_public_key_info']['algorithm_identifier']['algorithm'])

    else:
        temp_list.append('unknown')

    if 'extensions' in mydata[i]['tls']['server']['certs'][0]['cert']:
        if 'subject_key_identifier' in
mydata[i]['tls']['server']['certs'][0]['cert']['extensions']:
temp_list.append(mydata[i]['tls']['server']['certs'][0]['cert']['extensions']['subject_key_identifier'])

        else:
            temp_list.append('unknown')
    else:
        temp_list.append('unknown')

```

```

        if 'extensions' in mydata[i]['tls']['server']['certs'][0]['cert']:
            if 'authority_key_identifier' in
mydata[i]['tls']['server']['certs'][0]['cert']['extensions']:

temp_list.append(mydata[i]['tls']['server']['certs'][0]['cert']['extensions']
['authority_key_identifier'])

        else:

            temp_list.append('unknown')
    else:
        temp_list.append('unknown')

temp_list.append(mydata[i]['tls']['server']['certs'][0]['cert']['validity'][0
]['not_before'])

        temp_list.append('neg')

        final_data.append(temp_list)

tlsv1 = 0
tlsv2 = 0
tlsv3 = 0

if 'fingerprints' in mydata[i]:
    if 'tls' in mydata[i]['fingerprints']:
        if mydata[i]['fingerprints']['tls'][1:5] == '0301':
            tlsv1 = tlsv1 + 1
        elif mydata[i]['fingerprints']['tls'][1:5] == '0302':
            tlsv2 = tlsv2 + 1
        elif mydata[i]['fingerprints']['tls'][1:5] == '0303':
            tlsv3 = tlsv3 + 1
        else:
            pass
    elif 'tls_server' in mydata[i]['fingerprints']:
        if mydata[i]['fingerprints']['tls_server'][1:5] == '0301':
            tlsv1 = tlsv1 + 1
        elif mydata[i]['fingerprints']['tls_server'][1:5] == '0302':
            tlsv2 = tlsv2 + 1
        elif mydata[i]['fingerprints']['tls_server'][1:5] == '0303':
            tlsv3 = tlsv3 + 1
        else:
            pass
    else:
        pass

with open('/home/george/Downloads/thesis/malware_dataset/mal_final.json',
'r') as file:
    data = file.read().replace('\n', '')

mydata = json.loads(data)

print('Number of Malware flows: ',len(mydata))

```

```

for i in range(len(mydata)):

    if 'client' in mydata[i]['tls']:

        temp_list=[]

        temp_list.append(mydata[i]['src_port'])

        temp_list.append(mydata[i]['dst_port'])

        temp_list.append(mydata[i]['src_ip'])

        temp_list.append(mydata[i]['dst_ip'])

        temp_list.append(mydata[i]['tls']['client']['cipher_suites'])

        if mydata[i]['tls']['client']['cipher_suites'] ==
'003c002f003d00350005000ac027c013c014c02bc023c02cc024c009c00a00400032006a0038
00130004':
            mal_ciph1 = mal_ciph1 +1
            elif mydata[i]['tls']['client']['cipher_suites'] ==
'002f00350005000ac013c014c009c00a0032003800130004':
                mal_ciph2 = mal_ciph2 +1
                elif mydata[i]['tls']['client']['cipher_suites'] ==
'c011c007c00cc0020005c014c00a0039003800880087c00fc00500350084c013c00900330032
00450044c00ec004002f0041c012c00800160013c00dc003000a00ff':
                    mal_ciph3 = mal_ciph3 +1
                    elif mydata[i]['tls']['client']['cipher_suites'] ==
'c028c027c014c013009f009e009d009c003d003c0035002fc02cc02bc024c023c00ac009006a
004000380032000a001300050004':
                        mal_ciph4 = mal_ciph4 +1
                        elif mydata[i]['tls']['client']['cipher_suites'] ==
'000400050009002f0035003c003d':
                            mal_ciph5 = mal_ciph5 +1
                            elif mydata[i]['tls']['client']['cipher_suites'] ==
'c014c013c00ac0090035002f00380032000a001300050004':
                                mal_ciph6 = mal_ciph6 +1
                                elif mydata[i]['tls']['client']['cipher_suites'] ==
'c00ac0140088008700390038c00fc00500840035c007c009c011c0130045004400330032c00c
c00ec002c0040096004100040005002fc008c01200160013c00dc003feff000a00ff':
                                    mal_ciph7 = mal_ciph7 +1
                                    elif mydata[i]['tls']['client']['cipher_suites'] ==
'c00ac0140088008700390038c00fc00500840035c007c009c011c01300450044006600330032
c00cc00ec002c0040096004100050004002fc008c01200160013c00dc003feff000a':
                                        mal_ciph8 = mal_ciph8 +1
                                        elif mydata[i]['tls']['client']['cipher_suites'] ==
'cacac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':
                                            mal_ciph9 = mal_ciph9 +1
                                            elif mydata[i]['tls']['client']['cipher_suites'] ==
'4a4ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':
                                                mal_ciph10 = mal_ciph10 +1
                                                elif mydata[i]['tls']['client']['cipher_suites'] ==
'7a7ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':
                                                    mal_ciph11 = mal_ciph11 +1
                                                    elif mydata[i]['tls']['client']['cipher_suites'] ==
'6a6ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':

```

```
        mal_ciph12 = mal_ciph12 +1
        elif mydata[i]['tls']['client']['cipher_suites'] ==
'dadac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':
            mal_ciph13 = mal_ciph13 +1
            elif mydata[i]['tls']['client']['cipher_suites'] ==
'c028c027c014c013009f009e00390033009d009c003d003c0035002fc02cc02bc024c023c00a
c009006a004000380032000a0013':
                mal_ciph14 = mal_ciph14 +1
                elif mydata[i]['tls']['client']['cipher_suites'] ==
'1a1ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':
                    mal_ciph15 = mal_ciph15 +1
                    elif mydata[i]['tls']['client']['cipher_suites'] ==
'2a2ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':
                        mal_ciph16 = mal_ciph16 +1
                        elif mydata[i]['tls']['client']['cipher_suites'] ==
'c02bc02fc00ac009c013c014c012c007c0110033003200450039003800880016002f00410035
0084000a0005000400ff':
                            mal_ciph17 = mal_ciph17 +1
                            elif mydata[i]['tls']['client']['cipher_suites'] ==
'c02bc02cc02fc030009e009fc009c00ac013c01400330039c007c011009c009d002f00350005
00ff':
                                mal_ciph18 = mal_ciph18 +1
                                elif mydata[i]['tls']['client']['cipher_suites'] ==
'c02bc02fc00ac009c013c014c01200330032003900380016002f0035000a00ff':
                                    mal_ciph19 = mal_ciph19 +1
                                    elif mydata[i]['tls']['client']['cipher_suites'] ==
'c030c02cc028c024c014c00a00a500a300a1009f006b006a006900680039003800370036c032
c02ec02ac026c00fc005009d003d0035c02fc02bc027c023c013c00900a400a200a0009e00670
040003f003e0033003200310030009a009900980097c031c02dc029c025c00ec004009c003c00
2f0096c012c008001600130010000dc00dc003000a00ff':
                                        mal_ciph20 = mal_ciph20 +1
                                        elif mydata[i]['tls']['client']['cipher_suites'] ==
'aaaac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':
                                            mal_ciph21 = mal_ciph21 +1
                                            elif mydata[i]['tls']['client']['cipher_suites'] ==
'eaecac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':
                                                mal_ciph22 = mal_ciph22 +1
                                                elif mydata[i]['tls']['client']['cipher_suites'] ==
'5a5ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':
                                                    mal_ciph23 = mal_ciph23 +1
                                                    elif mydata[i]['tls']['client']['cipher_suites'] ==
'3a3ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a':
                                                        mal_ciph24 = mal_ciph24 +1
                                                        elif mydata[i]['tls']['client']['cipher_suites'] ==
'cca9cca8cc14cc13c02bc02fc02cc030c009c013c00ac014009c009d002f0035000a':
                                                            mal_ciph25 = mal_ciph25 +1
                                                            elif mydata[i]['tls']['client']['cipher_suites'] ==
'8a8ac02bc02fc02cc030cca9cca8c013c014009c009d002f0035000a':
                                                                mal_ciph26 = mal_ciph26 +1
                                                                elif mydata[i]['tls']['client']['cipher_suites'] ==
'0a0ac02bc02fc02cc030cca9cca8c013c014009c009d002f0035000a':
                                                                    mal_ciph27 = mal_ciph27 +1
                                                                    elif mydata[i]['tls']['client']['cipher_suites'] ==
'0033002f0035000a0005000400ff':
                                                                        mal_ciph28 = mal_ciph28 +1
                                                                        elif mydata[i]['tls']['client']['cipher_suites'] ==
'c0270067009cc011c007c00cc0020005c030c02cc028c024c014c00a00a500a300a1009f006b
```

```

006a0069006800390038003700360088008700860085c032c02ec02ac026c00fc005009d003d0
0350084c02fc02bc023c013c00900a400a200a0009e0040003f003e0033003200310030004500
4400430042c031c02dc029c025c00ec004003c002f004100ff':
    mal_ciph29 = mal_ciph29 +1
    elif mydata[i]['tls']['client']['cipher_suites'] ==
'cca9cca8cc14cc13c02bc02fc00ac014c009c013009c0035002f000a':
        mal_ciph30 = mal_ciph30 +1
        elif mydata[i]['tls']['client']['cipher_suites'] ==
'c02cc030009fcca9cca8ccaac02bc02f009ec024c028006bc023c0270067c00ac0140039c009
c0130033009d009c003d003c0035002f00ff':
            mal_ciph31 = mal_ciph31 +1
            elif mydata[i]['tls']['client']['cipher_suites'] ==
'130213031301c02fc02bc030c02c009ec0270067c028006b00a3009fcca9cca8ccaac0afc0ad
c0a3c09fc05dc061c057c05300a2c0aec0acc0a2c09ec05cc060c056c052c024006ac0230040c
00ac01400390038c009c01300330032009dc0a1c09dc051009cc0a0c09cc050003d003c003500
2f00ff':
                mal_ciph32 = mal_ciph32 +1
                elif mydata[i]['tls']['client']['cipher_suites'] ==
'130113031302c02bc02fcca9cca8c02cc030c00ac009c013c01400330039002f0035000a':
                    mal_ciph33 = mal_ciph33 +1
                    elif mydata[i]['tls']['client']['cipher_suites'] ==
'c030c02cc028c024c014c00a00a500a300a1009f006b006a0069006800390038003700360088
008700860085c032c02ec02ac026c00fc005009d003d00350084c02fc02bc027c023c013c0090
0a400a200a0009e00670040003f003e0033003200310030009a00990098009700450044004300
42c031c02dc029c025c00ec004009c003c002f00960041c012c008001600130010000dc00dc00
3000a0007c011c007c00cc0020005000400ff':
                        mal_ciph34 = mal_ciph34 +1
                        elif mydata[i]['tls']['client']['cipher_suites'] ==
'c030c02fc028c027c014c013009f009e009d009c003d003c0035002fc02cc02bc024c023c00a
c009006a004000380032000a0013':
                            mal_ciph35 = mal_ciph35 +1
                            elif mydata[i]['tls']['client']['cipher_suites'] ==
'fafa130113021303c02bc02fc02cc030cca9cca8c013c014009c009d002f0035':
                                mal_ciph36 = mal_ciph36 +1
                                elif mydata[i]['tls']['client']['cipher_suites'] ==
'c02cc02bc030c02fc024c023c028c027c00ac009c014c013009d009c003d003c0035002f000a
':
                                    mal_ciph37 = mal_ciph37 +1
                                    elif mydata[i]['tls']['client']['cipher_suites'] ==
'c02fc030c02bc02cca8cca9c013c009c014c00a009c009d002f0035c012000a130113031302
':
                                        mal_ciph38 = mal_ciph38 +1
                                        else:
                                            pass

if 'server_name' in mydata[i]['tls']['client']:

    temp_list.append(mydata[i]['tls']['client']['server_name'])

else:
    temp_list.append('unknown')

temp_list.append('unknown')
temp_list.append('unknown')
temp_list.append('unknown')

```

```

        temp_list.append('unknown')

        temp_list.append('pos')

        final_data.append(temp_list)

        if 'server' in mydata[i]['tls'] and 'certs' in
mydata[i]['tls']['server']:

            temp_list=[]

            temp_list.append(mydata[i]['src_port'])

            temp_list.append(mydata[i]['dst_port'])

            temp_list.append(mydata[i]['src_ip'])

            temp_list.append(mydata[i]['dst_ip'])

            temp_list.append('unknown')

            temp_list.append('unknown')

            if 'subject_public_key_info' in
mydata[i]['tls']['server']['certs'][0]['cert']:

temp_list.append(mydata[i]['tls']['server']['certs'][0]['cert']['subject_publ
ic_key_info']['algorithm_identifier']['algorithm'])

            else:

                temp_list.append('unknown')

            if 'extensions' in mydata[i]['tls']['server']['certs'][0]['cert']:
                if 'subject_key_identifier' in
mydata[i]['tls']['server']['certs'][0]['cert']['extensions']:

temp_list.append(mydata[i]['tls']['server']['certs'][0]['cert']['extensions']
['subject_key_identifier'])

            else:

                temp_list.append('unknown')

            else:

                temp_list.append('unknown')

            if 'extensions' in mydata[i]['tls']['server']['certs'][0]['cert']:
                if 'authority_key_identifier' in
mydata[i]['tls']['server']['certs'][0]['cert']['extensions']:

```

```

temp_list.append(mydata[i]['tls']['server']['certs'][0]['cert']['extensions']
[i]['authority_key_identifier'])

        else:
            temp_list.append('unknown')

    else:
        temp_list.append('unknown')

temp_list.append(mydata[i]['tls']['server']['certs'][0]['cert']['validity'][0]
['not_before'])

temp_list.append('pos')

final_data.append(temp_list)

if 'fingerprints' in mydata[i]:
    if 'tls' in mydata[i]['fingerprints']:
        if mydata[i]['fingerprints']['tls'][1:5] == '0301':
            tlsv1 = tlsv1 + 1
        elif mydata[i]['fingerprints']['tls'][1:5] == '0302':
            tlsv2 = tlsv2 + 1
        elif mydata[i]['fingerprints']['tls'][1:5] == '0303':
            tlsv3 = tlsv3 + 1
        else:
            pass
    elif 'tls_server' in mydata[i]['fingerprints']:
        if mydata[i]['fingerprints']['tls_server'][1:5] == '0301':
            tlsv1 = tlsv1 + 1
        elif mydata[i]['fingerprints']['tls_server'][1:5] == '0302':
            tlsv2 = tlsv2 + 1
        elif mydata[i]['fingerprints']['tls_server'][1:5] == '0303':
            tlsv3 = tlsv3 + 1
        else:
            pass
    else:
        pass

print('\nTLS version 1: ', tlsv1)
print('TLS version 2: ', tlsv2)
print('TLS version 3: ', tlsv3)

print("\nBenign ciphersuites: \n")
print("1) 002f00350005000ac013c014c009c00a0032003800130004: ", ben_ciph1)
print("2) 003c002f003d00350005000ac027c013c014c02bc023c02cc024c009c00a00400032006a003800130004: ", ben_ciph2)
print("3) c02bc02fcc9cca8c02cc030c00ac009c013c01400330039002f0035000a: ", ben_ciph3)

print("\nMalware ciphersuites: \n")

```



```
print("1)
003c002f003d00350005000ac027c013c014c02bc023c02cc024c009c00a00400032006a00380
0130004: ", mal_ciph1)
print("2) 002f00350005000ac013c014c009c00a0032003800130004: ", mal_ciph2)
print("3)
c011c007c00cc0020005c014c00a0039003800880087c00fc00500350084c013c009003300320
0450044c00ec004002f0041c012c00800160013c00dc003000a00ff: ", mal_ciph3)
print("4)
c028c027c014c013009f009e009d009c003d003c0035002fc02cc02bc024c023c00ac009006a0
04000380032000a001300050004: ", mal_ciph4)
print("5) 000400050009002f0035003c003d: ", mal_ciph5)
print("6) c014c013c00ac0090035002f00380032000a001300050004: ", mal_ciph6)
print("7)
c00ac0140088008700390038c00fc00500840035c007c009c011c0130045004400330032c00cc
00ec002c0040096004100040005002fc008c01200160013c00dc003feff000a00ff: ",
mal_ciph7)
print("8)
c00ac0140088008700390038c00fc00500840035c007c009c011c01300450044006600330032c
00cc00ec002c0040096004100050004002fc008c01200160013c00dc003feff000a: ",
mal_ciph8)
print("9) cacac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a:
", mal_ciph9)
print("10) 4a4ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a:
", mal_ciph10)
print("11) 7a7ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a:
", mal_ciph11)
print("12) 6a6ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a:
", mal_ciph12)
print("13) dadac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a:
", mal_ciph13)
print("14)
c028c027c014c013009f009e00390033009d009c003d003c0035002fc02cc02bc024c023c00ac
009006a004000380032000a0013: ", mal_ciph14)
print("15) 1a1ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a:
", mal_ciph15)
print("16) 2a2ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a:
", mal_ciph16)
print("17)
c02bc02fc00ac009c013c014c012c007c0110033003200450039003800880016002f004100350
084000a0005000400ff: ", mal_ciph17)
print("18)
c02bc02cc02fc030009e009fc009c00ac013c01400330039c007c011009c009d002f003500050
0ff: ", mal_ciph18)
print("19) c02bc02fc00ac009c013c014c01200330032003900380016002f0035000a00ff:
", mal_ciph19)
print("20)
c030c02cc028c024c014c00a00a500a300a1009f006b006a006900680039003800370036c032c
02ec02ac026c00fc005009d003d0035c02fc02bc027c023c013c00900a400a200a0009e006700
40003f003e0033003200310030009a009900980097c031c02dc029c025c00ec004009c003c002
f0096c012c008001600130010000dc00dc003000a00ff: ", mal_ciph20)
print("21) aaaac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a:
", mal_ciph21)
print("22) eaeac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a:
", mal_ciph22)
print("23) 5a5ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a:
", mal_ciph23)
```

```

print("24) 3a3ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a: ",
mal_ciph24)
print("25)
cca9cca8cc14cc13c02bc02fc02cc030c009c013c00ac014009c009d002f0035000a: ",
mal_ciph25)
print("26) 8a8ac02bc02fc02cc030cca9cca8c013c014009c009d002f0035000a: ",
mal_ciph26)
print("27) 0a0ac02bc02fc02cc030cca9cca8c013c014009c009d002f0035000a: ",
mal_ciph27)
print("28) 0033002f0035000a0005000400ff: ", mal_ciph28)
print("29)
c0270067009cc011c007c00cc0020005c030c02cc028c024c014c00a00a500a300a1009f006b0
06a0069006800390038003700360088008700860085c032c02ec02ac026c00fc005009d003d00
350084c02fc02bc023c013c00900a400a200a0009e0040003f003e00330032003100300045004
400430042c031c02dc029c025c00ec004003c002f004100ff: ", mal_ciph29)
print("30) cca9cca8cc14cc13c02bc02fc00ac014c009c013009c0035002f000a: ",
mal_ciph30)
print("31)
c02cc030009fcc9cca8ccaac02bc02f009ec024c028006bc023c0270067c00ac0140039c009c
0130033009d009c003d003c0035002f00ff: ", mal_ciph31)
print("32)
130213031301c02fc02bc030c02c009ec0270067c028006b00a3009fcc9cca8ccaac0afc0adc
0a3c09fc05dc061c057c05300a2c0aec0acc0a2c09ec05cc060c056c052c024006ac0230040c0
0ac01400390038c009c01300330032009dc0a1c09dc051009cc0a0c09cc050003d003c0035002
f00ff: ", mal_ciph32)
print("33)
130113031302c02bc02fcc9cca8c02cc030c00ac009c013c01400330039002f0035000a: ",
mal_ciph33)
print("34)
c030c02cc028c024c014c00a00a500a300a1009f006b006a00690068003900380037003600880
08700860085c032c02ec02ac026c00fc005009d003d00350084c02fc02bc027c023c013c00900
a400a200a0009e00670040003f003e0033003200310030009a009900980097004500440043004
2c031c02dc029c025c00ec004009c003c002f00960041c012c008001600130010000dc00dc003
000a0007c011c007c00cc0020005000400ff: ", mal_ciph34)
print("35)
c030c02fc028c027c014c013009f009e009d009c003d003c0035002fc02cc02bc024c023c00ac
009006a004000380032000a0013: ", mal_ciph35)
print("36) fafa130113021303c02bc02fc02cc030cca9cca8c013c014009c009d002f0035: ",
mal_ciph36)
print("37)
c02cc02bc030c02fc024c023c028c027c00ac009c014c013009d009c003d003c0035002f000a: ",
mal_ciph37)
print("38)
c02fc030c02bc02ccca8cca9c013c009c014c00a009c009d002f0035c012000a130113031302: ",
mal_ciph38)

final_array = np.array(final_data)

final_array[:,2] = le.fit_transform(final_array[:,2])
final_array[:,3] = le.fit_transform(final_array[:,3])
final_array[:,4] = le.fit_transform(final_array[:,4])
final_array[:,5] = le.fit_transform(final_array[:,5])

```

```

final_array[:,6] = le.fit_transform(final_array[:,6])
final_array[:,7] = le.fit_transform(final_array[:,7])
final_array[:,8] = le.fit_transform(final_array[:,8])
final_array[:,9] = le.fit_transform(final_array[:,9])

X = final_array[:,0:10]
y = final_array[:,10]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)

print("\nRandom Forests: \n")
#dimiourgia montelou
clf=RandomForestClassifier(n_estimators=200)

#ekpaideysi
clf.fit(X_train,y_train)

#test
y_pred=clf.predict(X_test)

precision = precision_score(y_test, y_pred, average='binary',
pos_label='pos')
print('Precision: %.3f' % precision)

recall = recall_score(y_test, y_pred, average='binary',pos_label='pos')
print('Recall: %.3f' % recall)

f1 = f1_score(y_test, y_pred, average='binary',pos_label='pos')
print('F1-score: %.3f' % f1)

accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.3f \n' % accuracy)

knn = KNeighborsClassifier(n_neighbors=3)

#ekpaideysi
knn.fit(X_train,y_train)

#test
y_pred=knn.predict(X_test)

print("\nKNN: \n")

precision = precision_score(y_test, y_pred, average='binary',
pos_label='pos')
print('Precision: %.3f' % precision)

recall = recall_score(y_test, y_pred, average='binary',pos_label='pos')
print('Recall: %.3f' % recall)

```

```

f1 = f1_score(y_test, y_pred, average='binary',pos_label='pos')
print('F1-score: %.3f' % f1)

accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.3f \n' % accuracy)

print("SVM: \n")
#Create a svm Classifier
clf = svm.SVC(kernel='rbf') # Radial Basis Function Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred = clf.predict(X_test)

precision = precision_score(y_test, y_pred, average='binary',
pos_label='pos')
print('Precision: %.3f' % precision)

recall = recall_score(y_test, y_pred, average='binary',pos_label='pos')
print('Recall: %.3f' % recall)

f1 = f1_score(y_test, y_pred, average='binary',pos_label='pos')
print('F1-score: %.3f' % f1)

accuracy = accuracy_score(y_test, y_pred)
print('Accuracy: %.3f \n' % accuracy)

```

Τέλος, μπορούμε να βρούμε τον παραπάνω κώδικα καθώς και τα δεδομένα (αρχεία .pcap και αρχεία .json) στο σύνδεσμο του Google Drive [46].

## Παράρτημα Β – πίνακας ciphersuites

	<b>Ciphersuites (hash)</b>	<b>Benign or malware</b>	<b>Number</b>
<b>1)</b>	002f00350005000ac013c014c009c00a0032003800130004	Benign	19
<b>2)</b>	003c002f003d00350005000ac027c013c014c02bc023c02cc024c009c00a00400032006a003800130004	Benign	28
<b>3)</b>	c02bc02fcca9cca8c02cc030c00ac009c013c01400330039002f0035000a	Benign	26510
<b>4)</b>	003c002f003d00350005000ac027c013c014c02bc023c02cc024c009c00a00400032006a003800130004	Malware	4766
<b>5)</b>	002f00350005000ac013c014c009c00a0032003800130004	Malware	3890
<b>6)</b>	c011c007c00cc0020005c014c00a0039003800880087c00fc00500350084c013c0090033003200450044c00ec004002f0041c012c00800160013c00dc003000a00ff	Malware	8
<b>7)</b>	c028c027c014c013009f009e009d009c003d003c0035002fc02cc02bc024c023c00ac009006a004000380032000a001300050004	Malware	81
<b>8)</b>	000400050009002f0035003c003d	Malware	1
<b>9)</b>	c014c013c00ac0090035002f00380032000a001300050004	Malware	70
<b>10)</b>	c00ac0140088008700390038c00fc00500840035c007c009c011c0130045004400330032c00cc00ec002c0040096004100040005002fc008c01200160013c00dc003feff000a00ff	Malware	814
<b>11)</b>	c00ac0140088008700390038c00fc00500840035c007c009c011c01300450044006600330032c00cc00ec002c0040096004100050004002fc008c01200160013c00dc003feff000a	Malware	1
<b>12)</b>	cacac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	3

13)	4a4ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	1
14)	7a7ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	4
15)	6a6ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	1
16)	dadac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	5
17)	c028c027c014c013009f009e00390033009d009c003d003c0035002fc02cc02bc024c023c00ac009006a004000380032000a0013	Malware	2
18)	1a1ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	2
19)	2a2ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	2
20)	c02bc02fc00ac009c013c014c012c007c0110033003200450039003800880016002f004100350084000a0005000400ff	Malware	150
21)	c02bc02cc02fc030009e009fc009c00ac013c01400330039c007c011009c009d002f0035000500ff	Malware	159
22)	c02bc02fc00ac009c013c014c01200330032003900380016002f0035000a00ff	Malware	10
23)	c030c02cc028c024c014c00a00a500a300a1009f006b006a006900680039003800370036c032c02ec02ac026c00fc005009d003d0035c02fc02bc027c023c013c00900a400a200a0009e00670040003f003e0033003200310030009a009900980097c031c02dc029c025c00ec004009c003c002f0096c012c008001600130010000dc00dc003000a00ff	Malware	144
24)	aaaac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	2
25)	eaeac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	2
26)	5a5ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	1
27)	3a3ac02bc02fc02cc030cca9cca8cc14cc13c013c014009c009d002f0035000a	Malware	1

<b>28)</b>	cca9cca8cc14cc13c02bc02fc02cc030c009c013c00ac014009c009d002f0035000a	Malware	211
<b>29)</b>	8a8ac02bc02fc02cc030cca9cca8c013c014009c009d002f0035000a	Malware	12
<b>30)</b>	0a0ac02bc02fc02cc030cca9cca8c013c014009c009d002f0035000a	Malware	21
<b>31)</b>	0033002f0035000a0005000400ff	Malware	476
<b>32)</b>	c0270067009cc011c007c00cc0020005c030c02cc028c024c014c00a00a500a300a1009f006b006a0069006800390038003700360088008700860085c032c02ec02ac026c00fc005009d003d00350084c02fc02bc023c013c00900a400a200a0009e0040003f003e00330032003100300045004400430042c031c02dc029c025c00ec004003c002f004100ff	Malware	19
<b>33)</b>	cca9cca8cc14cc13c02bc02fc00ac014c009c013009c0035002f000a	Malware	138
<b>34)</b>	c02cc030009fcca9cca8ccaac02bc02f009ec024c028006bc023c0270067c00ac0140039c009c0130033009d009c003d003c0035002f00ff	Malware	524
<b>35)</b>	130213031301c02fc02bc030c02c009ec0270067c028006b00a3009fcca9cca8ccaac0afc0adc0a3c09fc05dc061c057c05300a2c0aec0acc0a2c09ec05cc060c056c052c024006ac0230040c00ac01400390038c009c01300330032009dc0a1c09dc051009cc0a0c09cc050003d003c0035002f00ff	Malware	44
<b>36)</b>	130113031302c02bc02fcca9cca8c02cc030c00ac009c013c01400330039002f0035000a	Malware	4
<b>37)</b>	c030c02cc028c024c014c00a00a500a300a1009f006b006a0069006800390038003700360088008700860085c032c02ec02ac026c00fc005009d003d00350084c02fc02bc027c023c013c00900a400a200a0009e00670040003f003e0033003200310030009a0099009800970045004400430042c031c02dc029c025c00ec004009c003c002f00960041c012c008001600130010000dc00dc003000a0007c011c007c00cc0020005000400ff	Malware	92

<b>38)</b>	c030c02fc028c027c014c013009f009e009d009c0 03d003c0035002fc02cc02bc024c023c00ac00900 6a004000380032000a0013	Malware	157
<b>39)</b>	fafa130113021303c02bc02fc02cc030cca9cca8c0 13c014009c009d002f0035	Malware	71
<b>40)</b>	c02cc02bc030c02fc024c023c028c027c00ac009c 014c013009d009c003d003c0035002f000a	Malware	1749
<b>41)</b>	c02fc030c02bc02ccca8cca9c013c009c014c00a0 09c009d002f0035c012000a130113031302	Malware	164



# Βιβλιογραφία

- [1] SSL, <https://el.wikipedia.org/wiki/SSL>
- [2] SSL χειραψία, [https://upload.wikimedia.org/wikipedia/commons/1/17/SSL\\_Handshake.jpg](https://upload.wikimedia.org/wikipedia/commons/1/17/SSL_Handshake.jpg)
- [3] TLSv1.2 χειραψία, <https://tlseminar.github.io/first-few-milliseconds/>
- [4] TLSv1.3, [https://en.wikipedia.org/wiki/Transport\\_Layer\\_Security#TLS\\_1.3](https://en.wikipedia.org/wiki/Transport_Layer_Security#TLS_1.3)
- [5] TLSv1.3, <https://wiki.openssl.org/index.php/TLS1.3>
- [6] TLSv1.3 χειραψία, <https://timtaubert.de/blog/2015/11/more-privacy-less-latency-improved-handshakes-in-tls-13/>
- [7] Website protocol support, [https://en.wikipedia.org/wiki/Transport\\_Layer\\_Security#Websites](https://en.wikipedia.org/wiki/Transport_Layer_Security#Websites)
- [8] Qualys SSL pulse, <https://www.ssllabs.com/ssl-pulse/>
- [9] Stratosphere IPS, <https://www.stratosphereips.org/datasets-normal>
- [10] Malware traffic analysis, <https://www.malware-traffic-analysis.net/>
- [11] Detecting Encrypted Malware Traffic (Without Decryption), <https://blogs.cisco.com/security/detecting-encrypted-malware-traffic-without-decryption>
- [12] TCP and UDP ports, [https://en.wikipedia.org/wiki/List\\_of\\_TCP\\_and\\_UDP\\_port\\_numbers](https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers)
- [13] Ephemeral ports, [https://en.wikipedia.org/wiki/Ephemeral\\_port#Range](https://en.wikipedia.org/wiki/Ephemeral_port#Range)
- [14] Sample List of Higher Risk IP Addresses, <https://www.maxmind.com/en/high-risk-ip-sample-list>
- [15] High Risk IP Addresses, <https://www.fraudlabspro.com/high-risk-ip-address>
- [16] TLS Cipher Suites, <https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml#tls-parameters-4>
- [17] TLS Ciphersuite Search, <https://ciphersuite.info/>
- [18] TLS extensions, <https://www.iana.org/assignments/tls-extensiontype-values/tls-extensiontype-values.xhtml#tls-extensiontype-values-1>
- [19] Subject key identifier, <https://www.rfc-editor.org/rfc/rfc3280#section-4.2.1.2>
- [20] Authority key identifier, <https://datatracker.ietf.org/doc/html/rfc5280#section-4.2.1.1>
- [21] Public key certificates, [https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/tls.html#public\\_key\\_certificates](https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/tls.html#public_key_certificates)
- [22] Maximum SSL/TLS Certificate Validity, <https://www.globalsign.com/en/blog/maximum-ssltls-certificate-validity-now-one-year>

- [23] Changes to SSL/TLS Certificate Validity,  
<https://www.pkisolutions.com/changes-to-ssl-tls-certificate-validity-periods-september-2020/>
- [24] SNI, [https://en.wikipedia.org/wiki/Server\\_Name\\_Indication](https://en.wikipedia.org/wiki/Server_Name_Indication)
- [25] Alexa top sites, <https://www.alexa.com/topsites>
- [26] Detecting Malware in TLS Traffic,  
<https://www.imperial.ac.uk/media/imperial-college/faculty-of-engineering/computing/public/1819-pg-projects/Detecting-Malware-in-TLS-Traf%EF%AC%81c.pdf>
- [27] Let's Encrypt, <https://letsencrypt.org/>
- [28] Supervised vs Unsupervised Learning,  
<https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning>
- [29] Supervised ML, Unsupervised ML and Deep Learning,  
<https://analystprep.com/study-notes/cfa-level-2/quantitative-method/supervised-machine-learning-unsupervised-machine-learning-deep-learning/>
- [30] A survey of techniques for internet traffic classification using machine learning, <https://ieeexplore.ieee.org/document/4738466>
- [31] Sci-kit learn, <https://github.com/scikit-learn/scikit-learn>
- [32] Overfitting, <https://www.investopedia.com/terms/o/overfitting.asp>
- [33] Introduction to Random Forest in Machine Learning,  
<https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/>
- [34] Understanding Random Forest,  
<https://towardsdatascience.com/understanding-random-forest-58381e0602d2>
- [35] Machine Learning Basics with the K-Nearest Neighbors Algorithm,  
<https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761>
- [36] SVM Kernel Functions, <https://data-flair.training/blogs/svm-kernel-functions/>
- [37] Support Vector Machine – Introduction to Machine Learning Algorithms,  
<https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>
- [38] Performance Metrics for Classification problems in Machine Learning,  
<https://medium.com/@MohammedS/performance-metrics-for-classification-problems-in-machine-learning-part-i-b085d432082b>
- [39] Classification Metrics, <https://neptune.ai/blog/performance-metrics-in-machine-learning-complete-guide>
- [40] Evaluation Metrics, <https://deeptai.org/machine-learning-glossary-and-terms/evaluation-metrics>

- [41] Easiest way to convert pcap to JSON,  
<https://stackoverflow.com/questions/12330927/easiest-way-to-convert-pcap-to-json>
- [42] Mergecap, <https://www.wireshark.org/docs/man-pages/mergecap.html>
- [43] Mercury Cisco, <https://github.com/cisco/mercury>
- [44] Introducing JSON, <https://www.json.org/json-en.html>
- [45] JSON Editor Online, <https://jsoneditoronline.org/>
- [46] Google Drive,  
[https://drive.google.com/drive/folders/1lVu9DTEADJ0\\_Nteitsotx6UUwOmGzFRz?usp=sharing](https://drive.google.com/drive/folders/1lVu9DTEADJ0_Nteitsotx6UUwOmGzFRz?usp=sharing)