ΑΛΕΞΑΝΔΡΕΙΟ Τ.Ε.Ι ΘΕΣΣΑΛΟΝΙΚΗΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πτυχιακή εργασία

Υλοποίηση τεχνικής IP-over-DNS σε φορητές συσκευές με Windows Mobile.

του φοιτητή
Ταματέα Κωνσταντίνου
Αρ. Μητρώου: 03/2435

Επιβλέπων καθηγητής
Μηνάς Δασυγένης

Θεσσαλονίκη 2009

Πρόλογος

Όταν ξεκίνησα τη μελέτη του θέματος γνώριζα το Domain Name System επιφανειακά. Με την εκπόνηση την πρακτικής μου λίγους μήνες νωρίτερα εξοικειώθηκα στην C# γράφοντας εφαρμογές τηλεϊατρικής για Windows Mobile συσκευές. Έχοντας ως μόνη βάση τα παραπάνω, ξεκίνησα την αναζήτηση πληροφοριών στο διαδίκτυο η οποία αποδείχτηκε ελάχιστη. Με πολύ υπομονή και αφού ήρθα σε επαφή με δύο ξένους προγραμματιστές κατάφερα να κατανοήσω σε βάθος τη λειτουργία του DNS και τη γλώσσα προγραμματισμού C#, ώστε να αναπτύξω μια καινοτόμα εφαρμογή για την ενθυλάκωση πακέτων IP μέσα σε μηνύματα DNS σε συνδυασμό με τους αρχιτεκτονικούς περιορισμούς (μικρή ποσότητα μνήμης και χαμηλές επεξεργαστικές δυνατότητες) μιας φορητής συσκευής Windows Mobile.

Ευχαριστίες

Να ευχαριστήσω τον εισηγητή της πτυχιακής κ. Μηνά Δασυγένη
(mdasyg@ieee.org) για το πολύ ενδιαφέρον και επιμορφωτικό θέμα.

Αναμφίβολα τον Σουηδό προγραμματιστή του Iodine Protocol, Erik Ekman
(yarrick@kryo.se) του οποίου η βοήθεια ήταν καταλυτική για την επίλυση των πιο δύσκολων προβλημάτων που κλήθηκα να αντιμετωπίσω.

Τέλος, τον David Lemley (dev@ziggurat29.com) προγραμματιστή του OpenVPN για φορητές συσκευές του οποίου η αρωγή αν δεν υπήρχε δε θα είχα καταφέρει να υλοποιήσω το project για PDA.

Ευρετήριο Περιεχομένων

Ευρετήριο εικόνων και πινάκων

Σκοπός της Πτυχιακής Εργασίας

Η παρούσα εργασία έχει ως βασικό στόχο να περιγράψει αναλυτικά την όχι και τόσο διαδεδομένη τεχνική IP-over-DNS και έναν πρακτικό τρόπο εκμετάλλευσής της κάτω από ένα αυστηρά περιορισμένο δίκτυο όπου εμποδίζεται η χρήση όλων των γνωστών πρωτοκόλλων. Αρχικά σχεδιάστηκε μια εφαρμογή για Windows (x86) ως πλατφόρμα πάνω στην οποία δοκιμαζόταν εκτενώς η υλοποίηση της εν λόγω τεχνικής. Αργότερα, αφού επιβεβαιώθηκε το εφικτό της, ξεκίνησε η ανάπτυξη μιας εφαρμογής για Windows Mobile (ppc) που εκμεταλλεύεται την παγκόσμια υπηρεσία του DNS και τη χρησιμοποιεί ως πρωτόκολλο φορέα για να εδραιώσει μια αμφίδρομη επικοινωνία σε επίπεδο IP.

Abstract

This Diploma aims to describe in detail the not so common technique IP-over-DNS and a practical way of operating under a strictly limited network where the use of all known protocols is prohibited. Originally an application for Windows (x86) was designed as a platform on which the implementation of this technique was extensively tested. Later, having confirmed the feasibility of this technique began the development of an application for Windows Mobile (ppc) which exploits the world wide DNS service and uses it as a carrier protocol to establish a bidirectional communication on IP level.

Εισαγωγή

Ο όρος διασωλήνωση χρησιμοποιείται για να περιγράψουμε την κατάσταση όπου ένα πρωτόκολλο που ονομάζουμε ωφέλιμο (**payload protocol**) ενθυλακώνεται μέσα σε ένα άλλο πρωτόκολλο φορέα που ονομάζουμε παράδοσης (**delivery protocol**).

Το DNS επιλέχθηκε ως πρωτόκολλο παράδοσης μεταξύ άλλων επειδή είναι τελείως διαπερατό. Τα μηνύματα DNS θα διασχίσουν ολόκληρο το διαδίκτυο όπου συνήθως τίποτε άλλο δεν μπορεί. Συνήθως δεν υπόκεινται στους ίδιους περιορισμούς που υπόκεινται όλα τα άλλα πρωτόκολλα, και έτσι μπορούν και μεταφέρονται στους απομακρυσμένους διακομιστές, σε αντίθεση με τα υπόλοιπα πρωτόκολλα τα οποία εμποδίζονται από κάποιος τείχος προστασίας (firewall).

Επιτρέπει σε έναν να στέλνει και να λαμβάνει δεδομένα χρησιμοποιώντας ένα μη φιλτραρισμένο διαδικτυακό πρωτόκολλο. Η ενδιαφέρουσα όψη του DNS είναι ότι μπορεί να μετατραπεί σε ένα εκμεταλλεύσιμο συγκαλυμμένο και αμφίδρομο κανάλι δεδομένων το οποίο είναι ανεξάρτητο από τα λειτουργικά συστήματα που εφαρμόζουν το DNS διεθνώς. Το κανάλι γίνεται εφικτό κάνοντας χρήση δύο στοιχείων του πρωτόκολλου DNS. Ένα για τα upstream και ένα για τα downstream data.

Η πτυχιακή αποτελείται από τέσσερα κεφάλαια τα οποία περιγράφουν σταδιακά την πορεία που πρέπει να ακολουθήσουμε για να κατασκευάσουμε το τούνελ και από άλλα δύο τα οποία αναλύουν την εφαρμογή για Windows Mobile συσκευές. Αρχικά κάνουμε μια αναδρομή στη παγκόσμια υπηρεσία του Domain Name System για να θυμηθούμε τι μας προσφέρει και πως. Γνωρίζοντας πλέον τις ιδιαιτερότητες του, εισάγουμε στο 2^{ο} κεφάλαιο την έννοια της διασωλήνωσης εξηγώντας πως τα πρωτόκολλα διαφόρων επιπέδων μπορούν να ενθυλακώσουν το ένα το άλλο. Έχοντας τις παραπάνω γνώσεις μπαίνουμε στο 3^{ο} κεφάλαιο και αναλύουμε τα βήματα από τα οποία πρέπει να περάσει το πακέτο IP για να ενθυλακωθεί έγκυρα σε ένα DNS Query Message. Έπειτα εξηγούμε τι πρέπει να κάνουμε για να σταλεί επιτυχώς σε έναν ειδικό εξυπηρετητή ονομάτων και πως αυτός θα μας στείλει ένα DNS Response Message με ενθυλακωμένο ένα πακέτο IP. Στο 4^{ο} κεφάλαιο αναλύουμε και εφαρμόζουμε απόλυτα το πρωτόκολλο Iodine έκδοση 0.4.2 το οποίο μας προσφέρει λύσεις για καίρια ζητήματα που αφορούν

την συνεχόμενη αμφίδρομη και ελεγχόμενη αποστολή και λήψη πακέτων IP μέσω του DNS τούνελ ενώ στο 5^{ο} και 6^{ο} κεφάλαιο θα δούμε πως αναπτύσσουμε μια εφαρμογή σε C# με τα .NET Frameworks για φορητές συσκευές όπως PDA που τρέχουν το λειτουργικό σύστημα Windows Mobile έκδοση 5 ή 6.

Κεφάλαιο πρώτο

Domain Name System

1.1. Εισαγωγή στο DNS

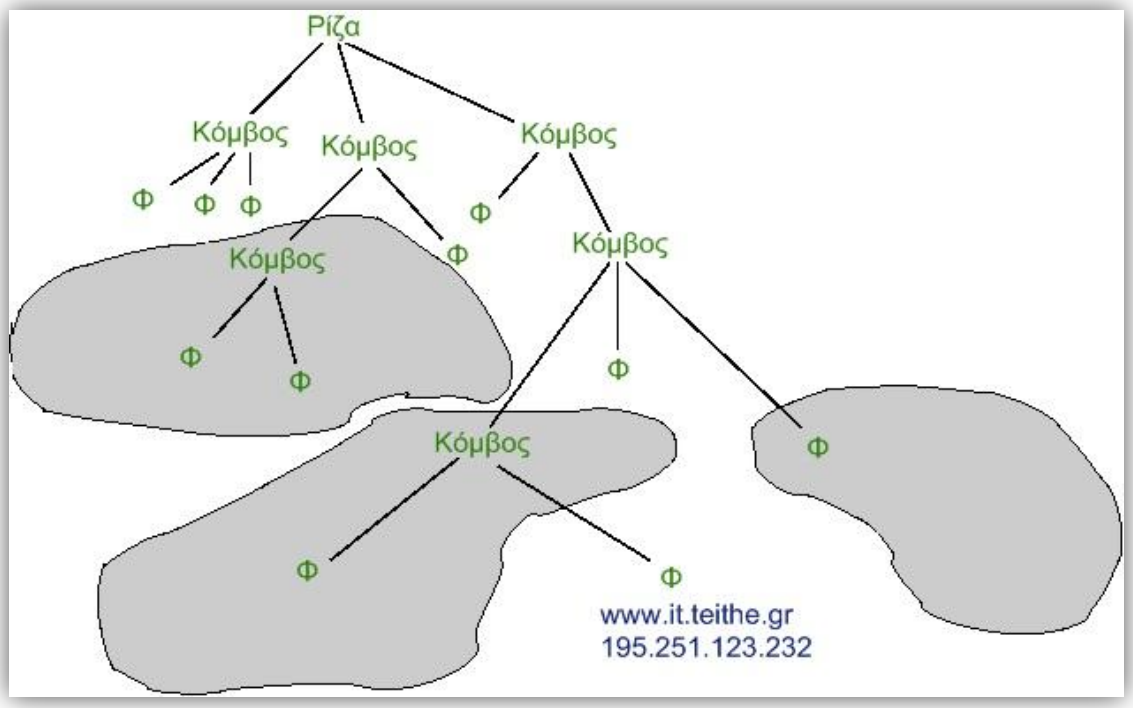
Το Domain Name System ή DNS εφευρέθηκε σύντομα μετά την ανάπτυξη του TCP/IP. Είναι ένα σύστημα ιεραρχίας που παρέχει ονόματα, τα λεγόμενα **domain names**, σε οποιαδήποτε συσκευή που συμμετέχει στο παγκόσμιο δίκτυο του Internet. Επινοήθηκε από τον Paul Mockapetris το 1983 ο οποίος έγραψε και την πρώτη υλοποίηση. Αμέσως έκαναν την εμφάνιση τους τα πρώτα Requests For Comments. Αυτά είναι τα RFC 882 & 883 τα οποία αντικαταστάθηκαν από τα RFC 1034 & 1035 όπου εφαρμόζονται μέχρι σήμερα. Η πιο γνωστή υλοποίηση μέχρι και σήμερα είναι ο Berkeley Internet Name Domain ή BIND. Παρόλα αυτά, λόγω του ανοιχτού του κώδικα, βρέθηκαν πολλά ελαττώματα - τρύπες που οδηγούσαν σε παραβίαση της ασφάλειας του συστήματος που τον χρησιμοποιούσε. Έτσι, αναπτύχθηκαν μια πληθώρα από εναλλακτικές λύσεις.

Η κύρια λειτουργία του είναι να αντιστοιχεί αυτά τα ονόματα σε μεγάλους πολύπλοκους αριθμούς. Αυτοί είναι διευθύνσεις που ανήκουν στο δίκτυο του Internet. Όπως για κάθε κατοικία στον κόσμο υπάρχει μια μοναδική διεύθυνση έτσι και για κάθε συσκευή στο διαδίκτυο υπάρχει μια διεύθυνση. Όντας μεγάλοι οι αριθμοί αυτοί είναι δύσκολο για τον άνθρωπο να τους απομνημονεύει και γι’ αυτό επινοήθηκε το DNS όπου μας παρέχει την διευκόλυνση να θυμόμαστε ονόματα παρά πολύπλοκους αριθμούς. Πιο απλά, το DNS θα μετατρέψει για εμάς τον αριθμό 195.251.123.232 στο όνομα www.it.teithe.gr. Προφανώς, μπορούμε να αντιληφθούμε την διαφορά όταν προσπαθήσουμε να τα απομνημονεύσουμε.

Διεθνώς το DNS απαρτίζεται από υπολογιστές ισχυρής επεξεργαστικής ισχύς τους λεγόμενους εξυπηρετητές ονομάτων ή name servers**.** Αυτές οι μηχανές είναι στην πραγματικότητα που θα αντιστοιχούν τα ονόματα σε διευθύνσεις. Τα ονόματα υποδιαιρούνται σε υπό ονόματα τα **sub domains** που αναλαμβάνονται διαδοχικά από άλλους εξυπηρετητές ονομάτων. Αυτός ο διαδοχικός μηχανισμός έκανε την διαχείριση του DNS κατανεμημένη, ανεκτή σε λάθη και βοήθησε ώστε να αποφευχθεί η ανάγκη για έναν κεντρικό ρυθμιστή που θα χρειαζόταν συνεχώς συντήρηση και ανανέωση.

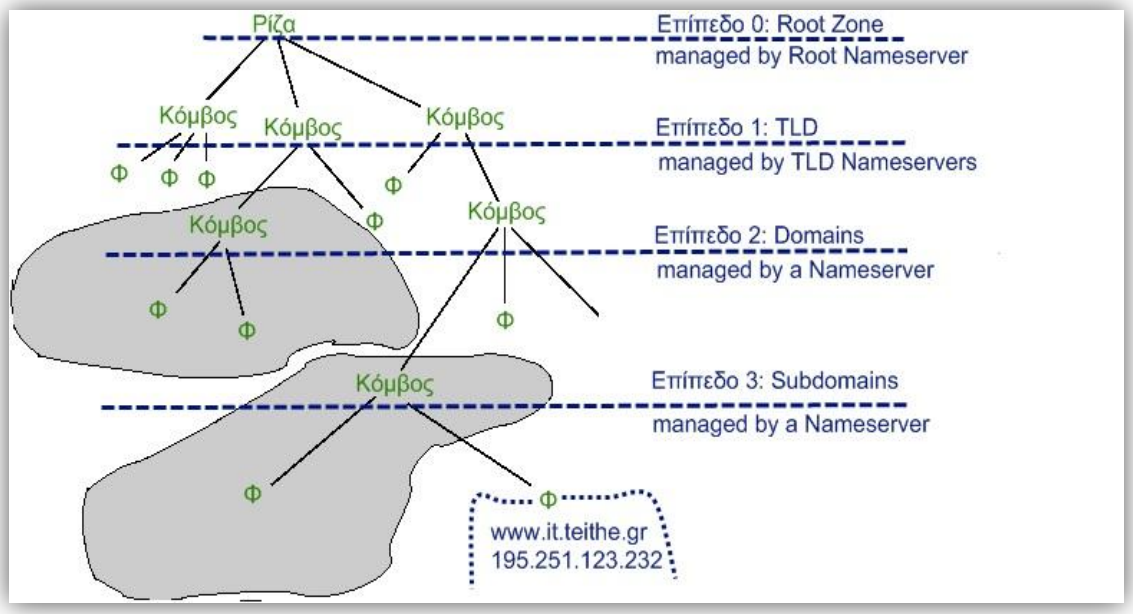
Η διαδικασία της μετατροπής ενός ονόματος σε διεύθυνση βασίζεται σε μια αναζήτηση που γίνεται μέσα σε ένα ιεραρχικό δέντρο το οποίο απαρτίζεται από κόμβους και φύλλα. Κάθε ένας από τους κόμβους είναι ένας εξυπηρετητής

ονομάτων που διαχειρίζεται σύνολα ονομάτων τις λεγόμενες ζώνες. Η αναζήτηση αρχίζει από την ρίζα της ιεραρχίας, διασχίζει κάποιους κόμβους και καταλήγει στο φύλλο ενός. Το φύλλο βρίσκεται στο τέρμα μιας συγκεκριμένης και μοναδικής διαδρομής και περιέχει τις πληροφορίες που ψάχνουμε.

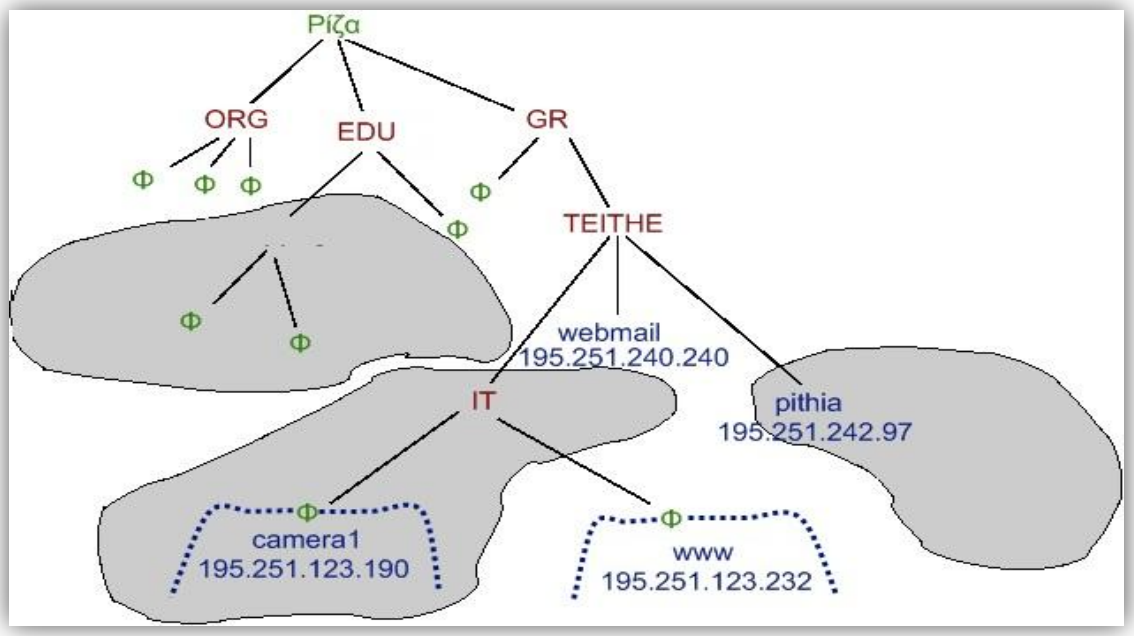


Εικόνα 1. Η ιεραρχία στο Domain Name System

Η ρίζα αλλιώς ονομάζεται **Root Zone** και συμβολίζεται με την τελεία. Την ζώνη της ρίζας την διαχειρίζονται υπέρ-υπολογιστές ανά την υφήλιο οι οποίοι ονομάζονται **Root Name servers**. Επιπλέον, εμπεριέχονται σε αυτήν κόμβοι που διαχειρίζονται άλλες ζώνες. Οι κόμβοι αυτοί είναι κάποιοι άλλοι υπέρ-υπολογιστές γνωστοί ως **TLD** εξυπηρετητές ονομάτων ή Top Level Domain εξυπηρετητές ονομάτων. Οι ζώνες των TLDs εμπεριέχουν με την σειρά τους άλλους κόμβους οι οποίοι διαχειρίζονται τις δικές τους ζώνες. Η διαχείριση των TLDs ανατέθηκε σε οργανισμούς από τον οργανισμό **ICANN** (Internet Corporation for Assigned Names and Numbers) ο οποίος χειρίζεται τον οργανισμό **IANA** (Internet Assigned Numbers Authority) και είναι επικεφαλής στην συντήρηση του Root Zone. Η ζώνη αυτή περιέχει τα γνωστά TLD όπως com για εμπορικούς οργανισμούς και edu για εκπαιδευτικά ιδρύματα.



Εικόνα 2. Επίπεδα ιεραρχίας του Domain Name System



Εικόνα 3. Παράδειγμα ιεραρχίας με πραγματικά δεδομένα

Στην παραπάνω εικόνα, η ζώνη της ρίζας περιέχει στην διαχείριση της τους TLD κόμβους ORG, EDU και GR. Κάθε ένα από αυτά τα TLD έχει στη ζώνη διαχείρισης του την ευθύνη για όλα τα υπό-ονόματα του. Η ζώνη διαχείρισης του κόμβου GR περιλαμβάνει ένα φύλλο και τον κόμβο TEITHE ο οποίος με την σειρά του έχει μια ζώνη διαχείρισης η οποία είναι υπεύθυνη για όλα τα υπό-ονόματα του

webmail, it και pithia. Τέλος βλέπουμε ότι τα υπό-ονόματα webmail και pithia είναι φύλλα ενώ ο IT είναι κόμβος και έχει μια ζώνη διαχείρισης που περιέχει μόνο φύλλα. Αυτά περιέχουν πληροφορίες για την αντιστοίχιση του εκάστοτε ονόματος σε μια διεύθυνση. Εκτός από μια διεύθυνση μπορεί να περιέχουν και άλλες επιπρόσθετες πληροφορίες. Όλες αυτές οι πληροφορίες τακτοποιούνται σε εγγραφές οι οποίες ονομάζονται **Resource Records** (**RR**).

1.2. Resource Records

Τα Resource Records είναι τα δομικά στοιχεία του DNS. Είναι εγγραφές που τακτοποιούν όλα τα είδη πληροφορίας που θα ζητούσαμε για ένα όνομα. Για να αποκτήσουμε ένα Resource Record με την πληροφορία που θέλουμε πρέπει να στείλουμε ένα ειδικό μήνυμα στον εξυπηρετητή ονομάτων. Αυτό ονομάζεται DNS Query Message. Εφόσον ο εξυπηρετητής λάβει ένα τέτοιο έγκυρο μήνυμα θα απαντήσει με ένα άλλο ειδικό μήνυμα που ονομάζεται DNS Response Message. Το τελευταίο περιέχει το Resource Record το οποίο περιέχει την συγκεκριμένη πληροφορία που ζητήσαμε. Η δομή ενός RR ορίζεται στο RFC 1035 και τα πεδία του παρουσιάζονται στον παρακάτω πίνακα.

Πίνακας 1. Δομή του Resource record

Πεδίο	Μήκος (bytes)
NAME	μεταβλητό
TYPE	2
CLASS	2
TTL	4
RDLENGTH	2
RDATA	μεταβλητό

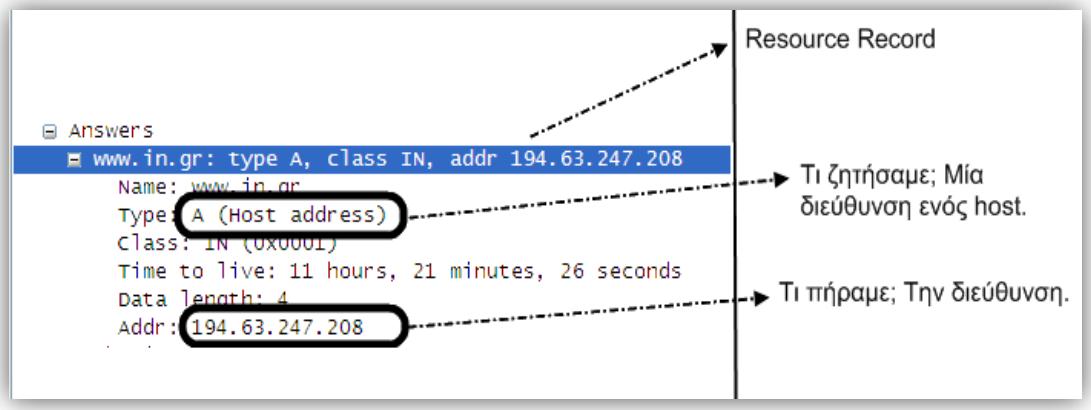
Το πεδίο NAME περιέχει το πλήρως έγκυρο όνομα ή fully qualified domain name (**FQDN**) του κόμβου στο δένδρο του DNS. Το FQDN προσδιορίζει την ακριβή τοποθεσία του κόμβου στο δένδρο. Είναι αυτό που βλέπουμε λίγες φορές ως *somehost.example.com.*

Το πεδίο TYPE δηλώνει το είδος της πληροφορίας που θα περιέχει το Resource Record. Βάση του πεδίου αυτού ξεχωρίζουμε τα είδη των Resource Records. Όπως είπαμε, το DNS είναι ένας μηχανισμός που μετατρέπει ονόματα σε διευθύνσεις. Είναι σαν το τηλεφωνικό κέντρο που καλούμε ζητώντας πληροφορίες για κάτι. Από την υπηρεσία αυτή εκτός του ότι θα μπορούσαμε να μάθουμε ένα απλό τηλέφωνο θα μπορούσαμε να μάθουμε ενδεχομένως και κάποια άλλη πληροφορία. Ομοίως, το DNS εκτός του να προσφέρει διευθύνσεις μπορεί να προσφέρει και άλλες πληροφορίες. Αυτό επιτυγχάνεται αν στείλουμε στον εξυπηρετητή DNS Query Messages με διαφορετικό TYPE. Έτσι τον κάνουμε να δημιουργήσει και να στείλει τον τύπο του Resource Record που εμείς θέλουμε. Ωστόσο κάθε τύπος RR έχει διαφορετικό μέγεθος και γι' αυτό θα προκύψει ένα πρόβλημα που θα αναλύσουμε παρακάτω.

Το πεδίο CLASS έχει σχεδόν πάντα την τιμή « IN » που σημαίνει Internet. Δηλαδή, ότι αυτό που ζητάμε αφορά την περιοχή του Internet. Το πεδίο TTL δηλώνει το χρόνο που το RR είναι έγκυρο ενώ το πεδίο RDLENGTH περιέχει το μέγεθος που ζυγίζει το πεδίο RDATA.

Το πεδίο RDATA περιέχει την πληροφορία σύμφωνα με το TYPE. Παρατηρούμε και εδώ ότι το μήκος του RDATA θα είναι μεταβλητό. Αυτό συμβαίνει επειδή ποτέ δε ξέρουμε τι μέγεθος έχει η πληροφορία που θα μας σταλεί ως απάντηση.

Στη συνέχεια θα δούμε ένα RR που συλλάβαμε στο wireshark. Στείλαμε σε ένα εξυπηρετητή ένα DNS Query Message τύπου A ζητώντας την διεύθυνση του ονόματος www.in.gr και πήραμε ως απάντηση ένα DNS Response Message το οποίο περιέχει ένα Resource Record με μια IPv4 διεύθυνση.

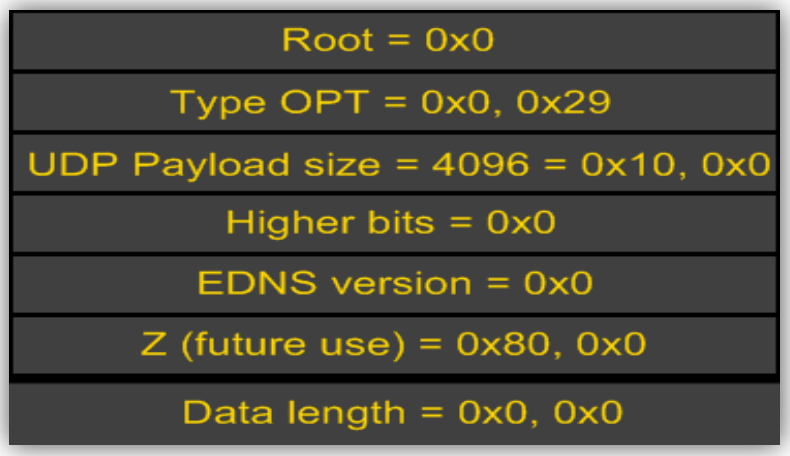


Εικόνα 4. Ένα Resource Record

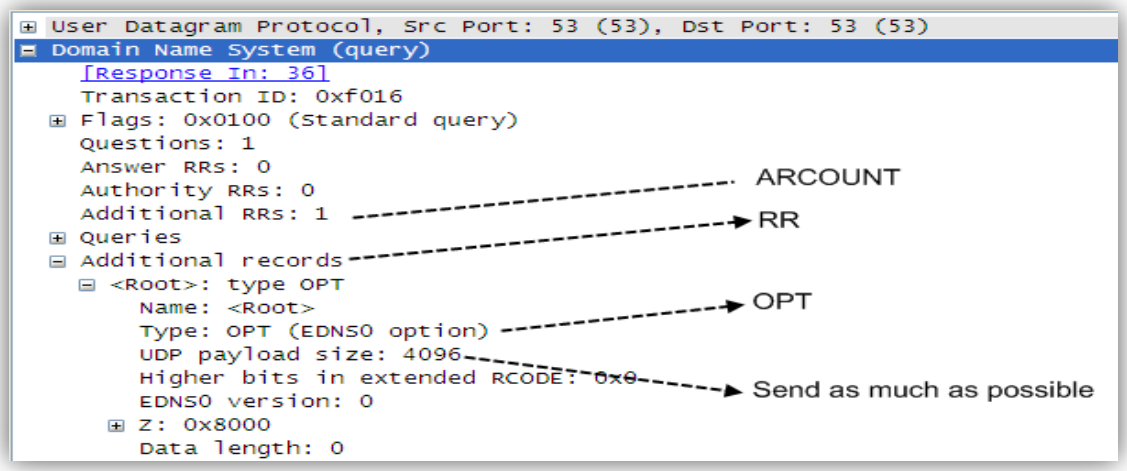
Για το παρών project η τιμή του TYPE είναι μία από τις βασικότερες παραμέτρους για την επίτευξη μιας γρήγορης και συνάμα σταθερής επικοινωνίας καθώς ορίζει το μέγεθος των downstream data που θα λάβουμε από τα DNS Response Messages των εξυπηρετητών. Εδώ όμως παρουσιάζεται ένας περιορισμός. Το RFC 1035 δηλώνει ότι κάθε DNS UDP Packet δεν μπορεί να υπερβαίνει τα 512 byte. Ωστόσο υπάρχει ένα άλλο στοιχείο του DNS που μπορεί να μας επιτρέψει να επεκτείνουμε λιγάκι το μέγεθος του DNS UDP Packet. Ο λόγος γίνεται για το **EDNS** που θα μελετήσουμε παρακάτω.

1.3. Extended DNS

Το EDNS είναι μια επέκταση του πρωτοκόλλου DNS και ορίζεται από το RFC 2671. Επιτρέπει σε αυτούς που στέλνουν τα DNS Query Messages να διαφημίσουν το μέγεθος του DNS UDP Packet που μπορούν να λάβουν και να αποκωδικοποιήσουν. Για να διαχωρίσουμε ένα EDNS μήνυμα από ένα κοινό εισάγουμε στο DNS Query Message ένα Resource Record το λεγόμενο **OPT RR**. Συγκεκριμένα το εισάγουμε στον τομέα **Additional Records** του DNS μηνύματος. Το OPT RR έχει μήκος 11 byte και το πεδίο που περιέχει την τιμή του επιθυμητού μεγέθους είναι το UDP Payload Size και έχει μήκος 16 bit. Δηλαδή η μέγιστη τιμή που μπορεί να πάρει είναι 4096. Επομένως έχουμε την δυνατότητα να πούμε στον εξυπηρετητή ότι μπορεί να μας στείλει ένα DNS UDP Packet μέγιστου μεγέθους 4096 byte. Τελικά από τα 512 byte που μας επιτρέπει το RFC 1035 κάνουμε χρήση του RFC 2671 για να τα αυξήσουμε σε 4096. Παρόλα αυτά τα UDP DNS Packets μπορούν να ζυγίζουν συνήθως μέχρι 1280 byte γιατί οι περισσότερες υλοποιήσεις θα είναι σίγουρα ρυθμισμένες να μην επιτρέπουν τόσο μεγάλα UDP Packets να διασχίζουν το Domain Name System. Οι παρακάτω εικόνες μας δείχνουν την δομή ενός OPT RR και πως αυτό φαίνεται στο πακέτο DNS.



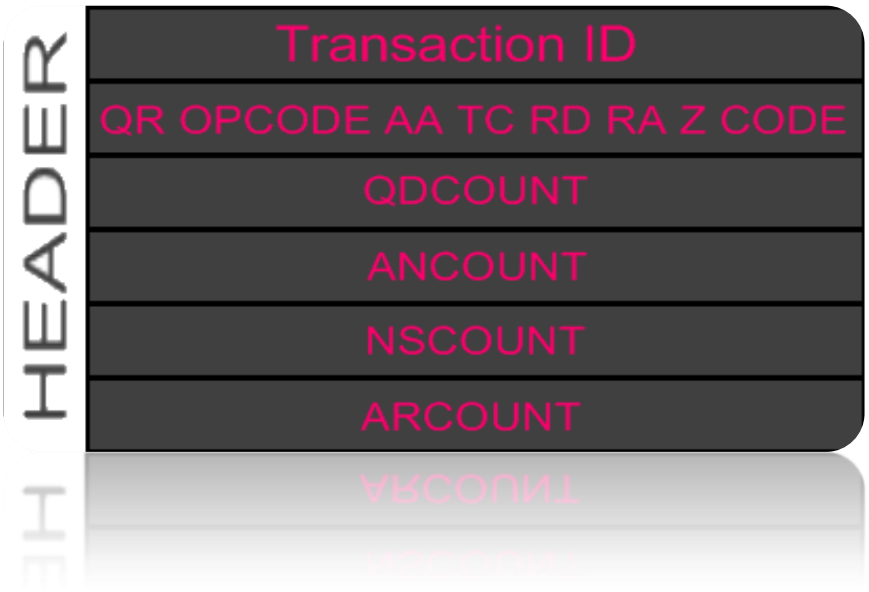
Εικόνα 5. Δομή του OPT RR



Εικόνα 6. Χρήση του ENDS στο DNS Query Message

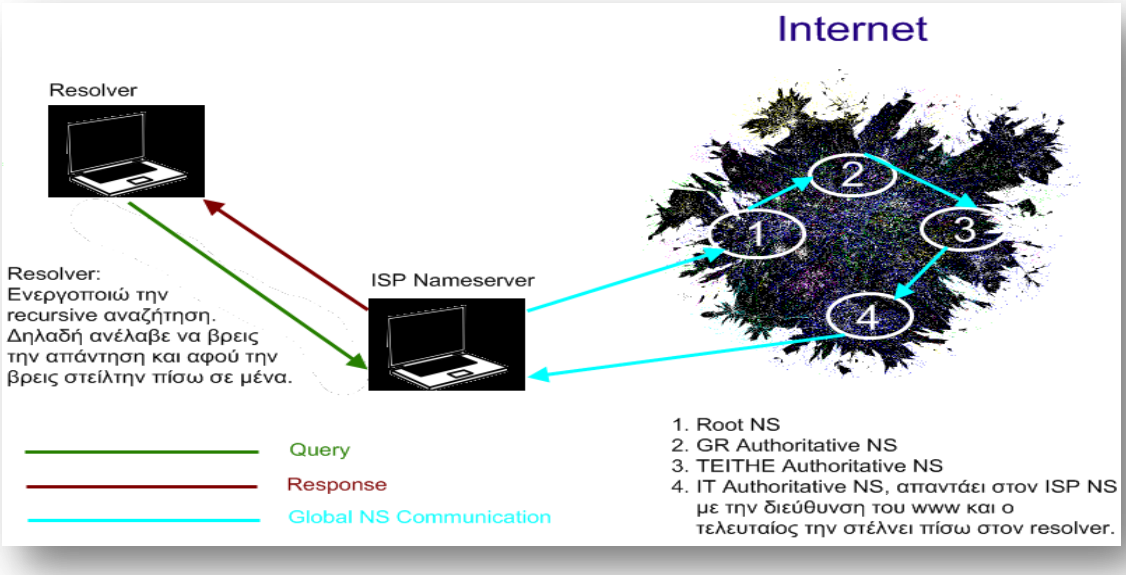
1.4. DNS lookups

Αφού στείλουμε ένα DNS Query Message στον εξυπηρετητή, ο τελευταίος πρέπει να ξεκινήσει την αναζήτηση της απάντησης. Στο DNS εφαρμόζονται δύο ειδών αναζητήσεις. Η **Recursive** και η **Iterative** και ποια θα χρησιμοποιηθεί αποφασίζεται από εμάς. Αυτό μπορούμε να το κάνουμε διαμορφώνοντας ένα κατάλληλο Header στο DNS Query Message που θα στείλουμε. Το πρώτο μπιτ στο τέταρτο byte του DNS Header αφιερώνεται για το πεδίο RD και αναλόγως την τιμή που του θέτουμε εναλλάσσεται μεταξύ των δύο αναζητήσεων. Η τιμή 1 ενεργοποιεί την Recursive και η τιμή 0 την Iterative.



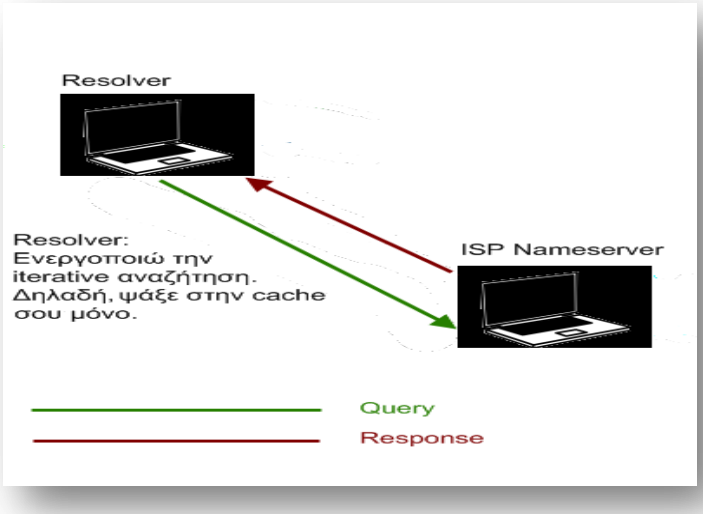
Εικόνα 7. DNS Message Header

Αν ενεργοποιήσουμε την **Recursive αναζήτηση** ο εξυπηρετητής θα αναλάβει να αναζητήσει αυτό που ζητήσαμε ρωτώντας διαδοχικά άλλους εξυπηρετητές παγκοσμίως. Είναι σαν το web proxy όπου λέμε σε έναν πολύ γρήγορο web server να κατεβάσει ολόκληρη την σελίδα και να μας την στείλει.



Εικόνα 8. Recursive αναζήτηση

Αν θέσουμε σε λειτουργία την **Iterative αναζήτηση** ο εξυπηρετητής θα αναζητήσει την απάντηση μόνο στην μνήμη cache που διαθέτει.



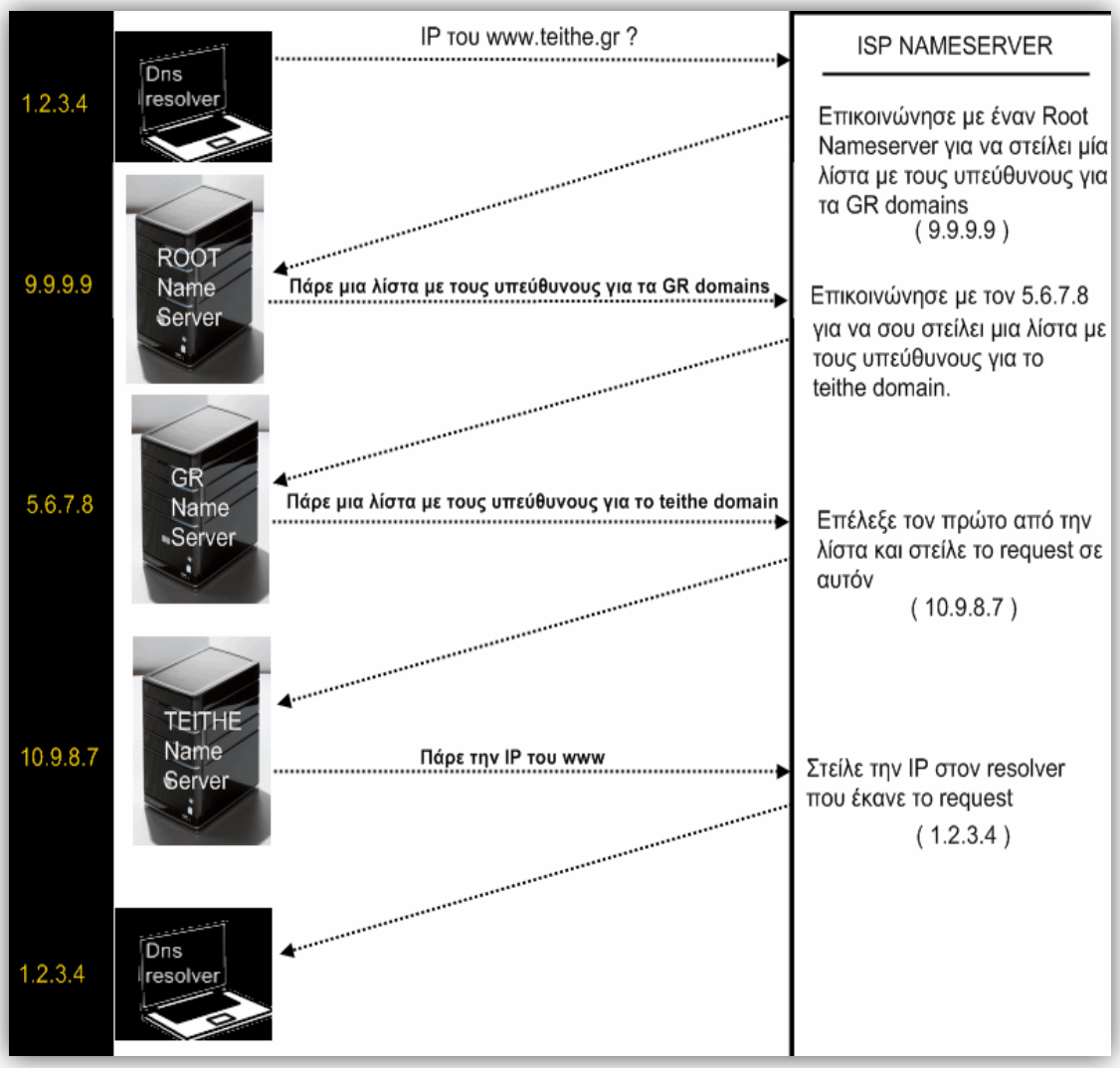
Εικόνα 9. Iterative αναζήτηση

Στις παραπάνω εικόνες βλέπουμε ότι μοναδική προϋπόθεση είναι να έχουμε επικοινωνία μέχρι τον εξυπηρετητή ονομάτων. Δηλαδή ο υπολογιστής μας θα πρέπει να ανήκει στο ίδιο δίκτυο που ανήκει και αυτός. Η προϋπόθεση αυτή είναι καταλυτική για το εγχείρημα που θέλουμε να επιτύχουμε και πάντα πληρείται στα Wi-Fi και 3G Hotspots που είναι και ο στόχος μας. Καθώς είναι φανερό ότι μας συμφέρει η αναδρομική (recursive) αναζήτηση αφού μας δίνει την δυνατότητα να μιλήσουμε με τον έξω κόσμο θα ασχοληθούμε μόνο με αυτήν. Ας δούμε λοιπόν με σύντομο και απλό τρόπο τι θα συμβεί όταν επισκεπτόμαστε ένα website και έχουμε θέσει σε λειτουργία την αναδρομική αναζήτηση.

Δίνουμε στον browser ένα όνομα όπως το www.teithe.gr και αυτός το στέλνει στον resolver του συστήματός μας. Αυτό παράγει και στέλνει ένα DNS Query Message στον εξυπηρετητή ονομάτων του ISP μας. Αυτός επικοινωνεί με κάποιον Root name server στο Internet. Ο Root name server του στέλνει μία λίστα με τους εξυπηρετητές ονομάτων που είναι υπεύθυνοι για το gr TLD. Ο εξυπηρετητής ονομάτων του ISP μας θα επιλέξει έναν εξυπηρετητή ονομάτων από την λίστα και θα επικοινωνήσει μαζί του. Ομοίως εκείνος θα του στείλει μια δεύτερη λίστα με τους εξυπηρετητές ονομάτων που είναι υπεύθυνοι για το domain teithe. Έτσι, ο εξυπηρετητής ονομάτων του ISP μας θα διαλέξει από την δεύτερη λίστα έναν εξυπηρετητή ονομάτων για το domain teithe και θα επικοινωνήσει άμεσα μαζί του στέλνοντάς του το DNS Query Message για να αποκτήσει την IP διεύθυνση

του host *www* που ανήκει στο domain teithe. Έχοντας λάβει ένα DNS Response Message το οποίο περιέχει την IP του *www*, την επιστρέφει στον υπολογιστή μας, ο οποίος με την σειρά του την επιστρέφει στην εφαρμογή που δημιούργησε την ερώτηση, δηλαδή τον browser.

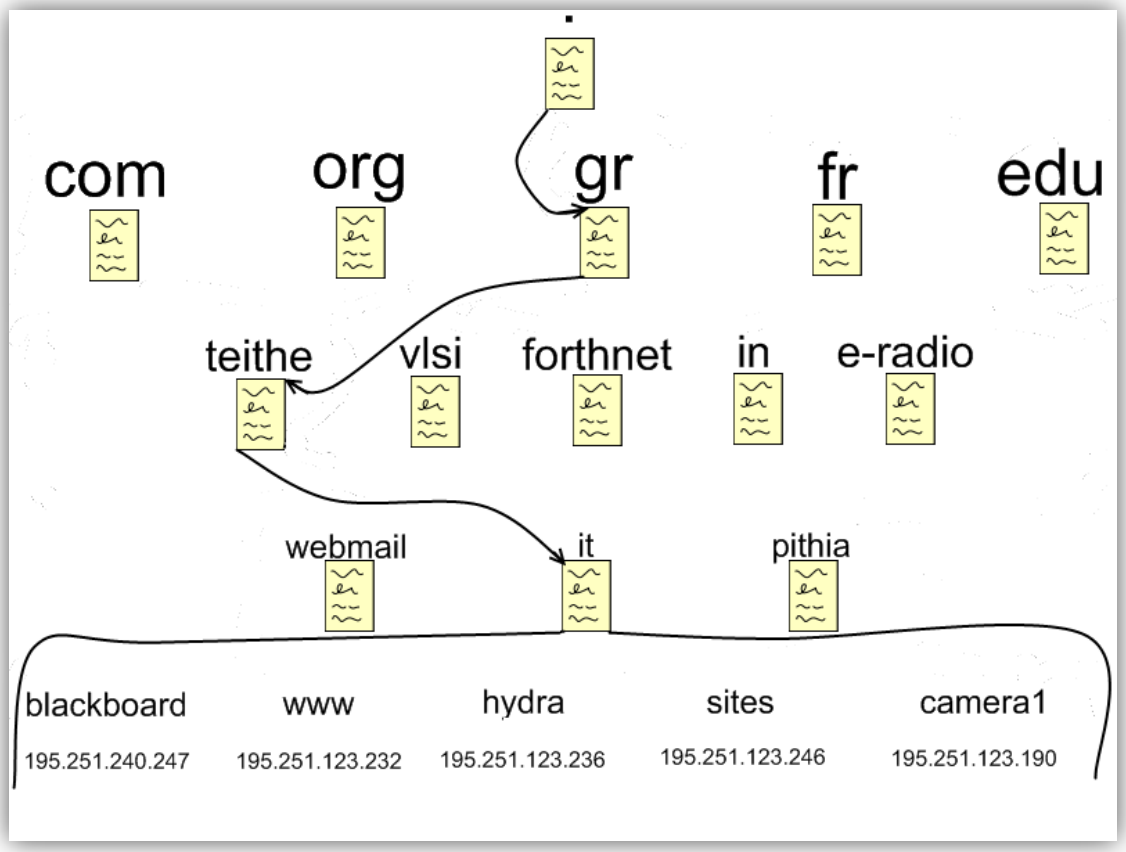
Όπως βλέπουμε η διαδικασία αυτή είναι χρονοβόρα και συνήθως αυτό γίνεται αντιληπτό και από τον χρήστη σε όλες τις εφαρμογές. Πολλά προβλήματα προκύπτουν σε ένα σύστημα όταν δεν έχει σωστά ρυθμισμένους τους εξυπηρετητές ονομάτων. Σχεδόν τίποτα διαδικτυακό δεν μπορεί να λειτουργήσει σε μια τέτοια περίπτωση. Παρακάτω θα δούμε αυτήν την διαδικασία σε μια απλοποιημένη εικόνα.



Εικόνα 10. Αναδρομικό DNS lookup

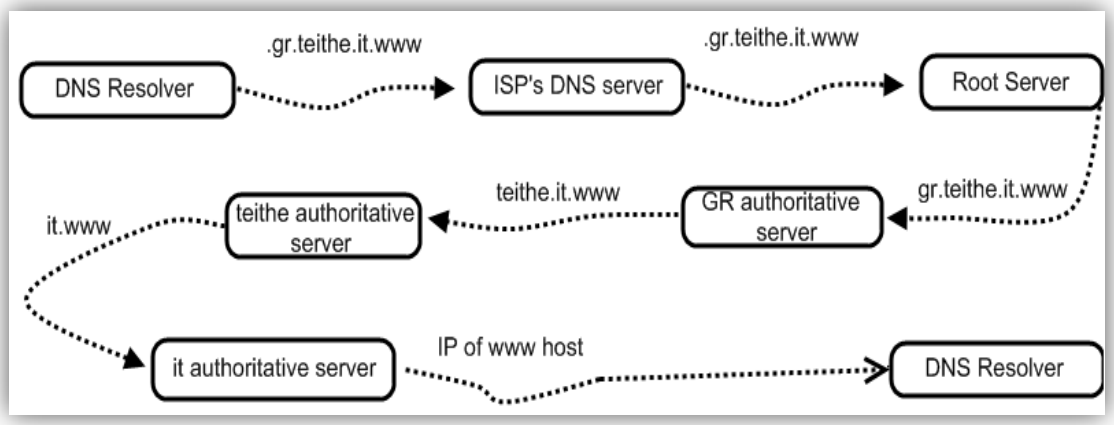
1.5. Δρομολόγηση ερωτήματος

Αναφέρθηκε προηγουμένως ότι μια ιδιαιτερότητα του DNS είναι το ιεραρχικό δέντρο του. Κάθε φορά που θέλουμε να βρούμε μια πληροφορία ενός ονόματος διασχίζεται η ιεραρχία αυτή. Για παράδειγμα, αναζητώντας την διεύθυνση του ονόματος www.it.teithe.gr θα διασχίσουμε την ιεραρχία ως εξής:



Εικόνα 11. Δρομολόγηση του ερωτήματος

Σημειώνουμε ότι, καθώς το όνομα ‘δρομολογείται’ μέσα στην ιεραρχία οι κόμβοι αφαιρούν από το όνομα το μέρος για το οποίο ήταν υπεύθυνοι και στέλνουν στον επόμενο κόμβο το υπολειπόμενο κομμάτι. Αυτή η ‘δρομολόγηση’ με την σταδιακή κατανάλωση του ονόματος από τους εμπλεκόμενους εξυπηρετητές ονομάτων είναι το στοιχείο που θα εκμεταλλευτούμε για να χτίσουμε το μονοπάτι του τούνελ.



Εικόνα 12. Κατανάλωση του ονόματος από εμπλεκόμενους DNS servers

Παραπάνω βλέπουμε ότι ο τελευταίος εμπλεκόμενος εξυπηρετητής ονομάτων είναι αυτός που εξυπηρετεί την ζώνη του IT. Στόχος μας θα είναι πάντα αυτός γιατί όπως φαίνεται και από την εικόνα είναι αυτός που θα επιστρέφει πληροφορίες πίσω σε εμάς.

Εφόσον είδαμε έως τώρα ότι μπορούμε να στείλουμε και να λάβουμε κάτι μέσα στο DNS ήρθε η ώρα να μελετήσουμε τι είδος πληροφορίας θα μπορούσε να είναι αυτή. Το επόμενο κεφάλαιο θα μας βοηθήσει σε αυτό.

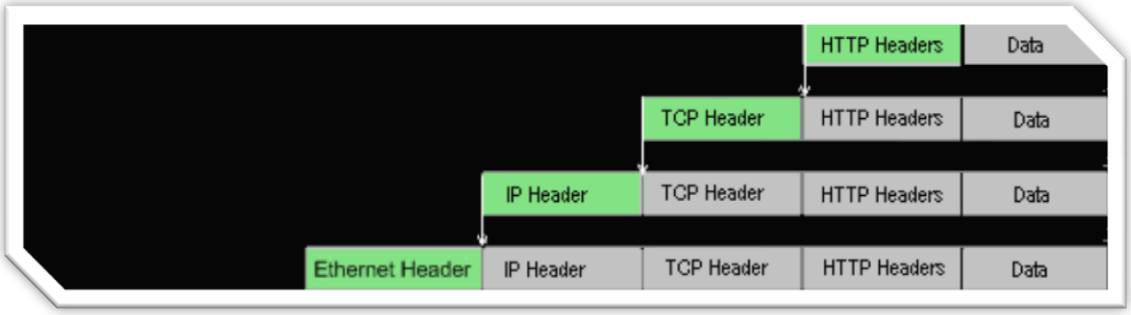
Κεφάλαιο δεύτερο

Διασωλήνωση

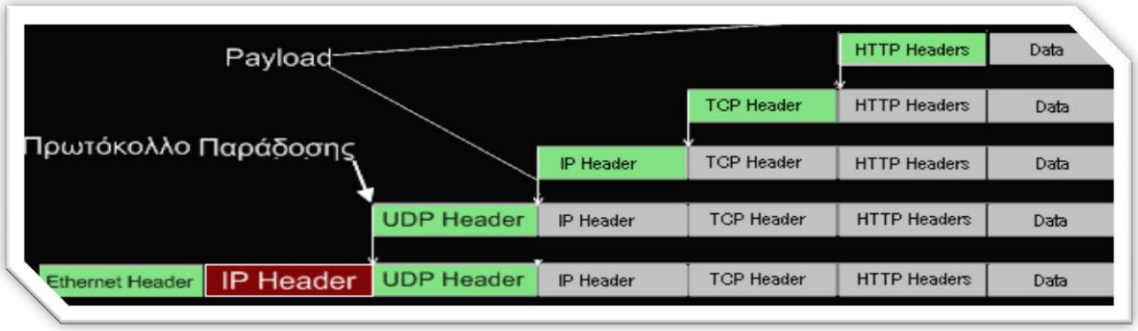
2.1. Εισαγωγή στη διασωλήνωση

Λέμε ότι φτιάχνουμε ένα τούνελ όταν ενθυλακώνουμε ένα πρωτόκολλο σε ένα άλλο. Όμως, η ενθυλάκωση πρωτοκόλλων σε συμβατικά μοντέλα όπως αυτό του TCP/IP δεν θα πρέπει να θεωρείται και διασωλήνωση.

Συνήθως, οι λόγοι που μας οδηγούν να χρησιμοποιήσουμε ένα τούνελ είναι όταν θέλουμε να μεταφέρουμε ένα ωφέλιμο πρωτόκολλο (**payload**) πάνω από ένα μη συμβατό πρωτόκολλο παράδοσης ή όταν είναι ανάγκη να έχουμε μια ασφαλή διαδρομή πάνω από ένα δίκτυο που δεν εμπιστευόμαστε.



Εικόνα 13. Ενθυλάκωση πρωτοκόλλων στο TCP/IP



Εικόνα 14. Ενθυλάκωση πρωτοκόλλου IP σε UDP

2.2. Το ωφέλιμο πρωτόκολλο

Το δικό μας εγχείρημα θέλει να περνάμε πακέτα IP μέσω του DNS τούνελ. Επομένως για εμάς το ωφέλιμο φορτίο είναι το πρωτόκολλο IP. Το πρώτο πράμα που πρέπει να εξετάσουμε είναι πως θα συλλαμβάνουμε τα πακέτα IP προτού

αυτά φύγουν από το δίκτυο αφού εμείς σκοπεύουμε να τα ενθυλακώσουμε στο DNS Query Message και όχι να τα αφήσουμε να φύγουν έτσι.

Για το διάβασμα των πακέτων από το σύστημα χρησιμοποιούμε τον **Tun/Tap Driver**. Ο driver αυτός πάρθηκε από το OpenVPN project. Είναι ανοιχτού κώδικα και μοιράζεται δημοσίως στο κοινό κάτω από την άδεια *OpenVPN GPL License.* Ο εν λόγω driver παρέχει στο λειτουργικό σύστημα εικονικές δικτυακές συσκευές. Οι συσκευές αυτές δεν είναι αληθινές δηλαδή δεν υφίστανται ως hardware αλλά είναι κομμάτι του λειτουργικού συστήματος. Έχουν τις ίδιες ιδιότητες με αυτές μιας τυπικής κάρτας δικτύου. Δηλαδή, έχουν IPv4 διευθύνσεις, MAC διευθύνσεις, ARP Cache, I/O buffers και συνεργάζονται με την TCP/IP Stack του συστήματος. Ο driver μπορεί να θέσει την συσκευή σε δύο καταστάσεις και μπορεί να επιλέξει ο προγραμματιστής σε ποια από τις δύο θα το βάλει.

- **Tun**. Σε αυτήν τη κατάσταση προσομοιώνουμε μία συσκευή που λειτουργεί με πακέτα 3^{ου} επιπέδου όπως τα IP packets και εκτελεί χρέη routing.

- **Tap**. Σε αυτή τη κατάσταση προσομοιώνουμε μία συσκευή που δουλεύει με πακέτα 2^{ου} επίπεδου όπως τα Ethernet Frames και εκτελεί χρέη bridging.

Εμείς προφανώς θα ασχοληθούμε με την πρώτη κατάσταση αφού θέλουμε να δουλέψουμε σε επίπεδο IP. Έτσι το μόνο που έχουμε να κάνουμε είναι να δρομολογούμε όλη την IP κίνηση του συστήματος στην εικονική συσκευή και από εκεί να την διαβάζουμε από το outgoing buffer της. Αυτή η εργασία θα γίνει από την εφαρμογή μας που είναι συνδεδεμένη με την εικονική συσκευή. Δηλαδή, η εφαρμογή θα διαβάζει πακέτα IP από το buffer της εικονικής συσκευής τα οποία προήλθαν από το λειτουργικό σύστημα.

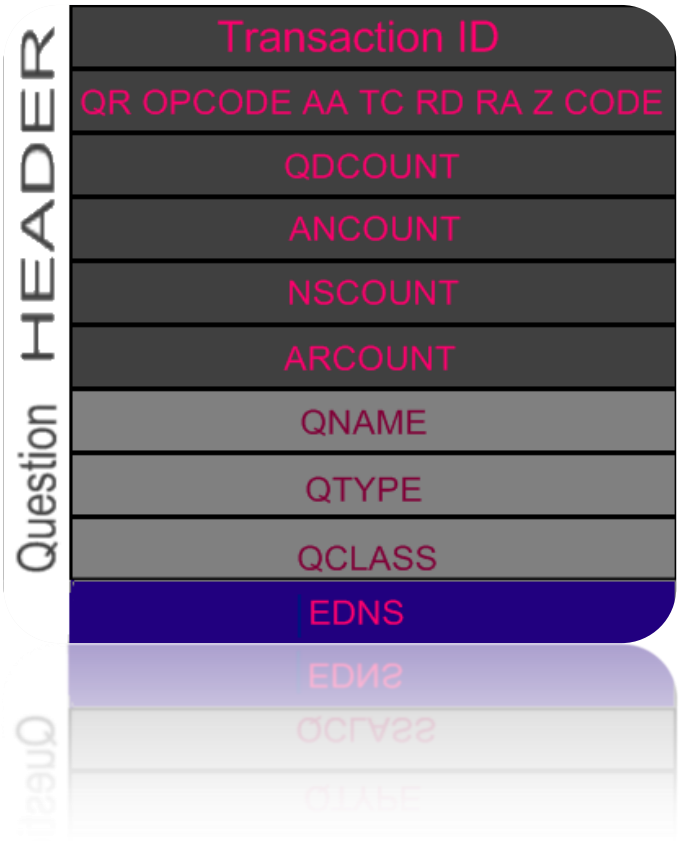
2.3. Το πρωτόκολλο παράδοσης

Στην προηγούμενη παράγραφο είπαμε πως η διασωλήνωση θέλει ένα πρωτόκολλο παράδοσης. Αυτό για εμάς είναι το πρωτόκολλο DNS πάνω στο οποίο γίνεται και η παρούσα εργασία. Τα DNS Messages θα μεταφέρουν για εμάς αυτό που θέλουμε να στείλουμε ως **upstream data** με τα DNS Query Messages

και αυτό που είναι να λάβουμε ως **downstream data** με τα DNS Response Messages. Η παράδοση των upstream data είναι το κομμάτι της αυτής της εργασίας ενώ η παράδοση των downstream data γίνεται από μια άλλη εφαρμογή για Linux με όνομα **Iodined.** Την δική μας εφαρμογή μπορούμε να την καλούμε client επειδή χειρίζεται το upstream ως αίτηση και τον Iodined να τον καλούμε server αφού χειρίζεται το downstream ως εξυπηρέτηση.

2.3.1. Παράδοση upstream data

Έχοντας αποκτήσει τα upstream data από την εικονική συσκευή ήρθε η στιγμή να εξετάσουμε έναν τρόπο ενθυλάκωσης τους στο DNS. Το πρώτο ερώτημα που πρέπει να απαντήσουμε είναι πως θα εκμεταλλευτούμε το DNS Query έτσι ώστε να ενθυλακώσουμε σ' αυτό τα επιθυμητά upstream data. Βάση του RFC 1035 και του RFC 2671 το DNS Query Message έχει την ακόλουθη δομή.



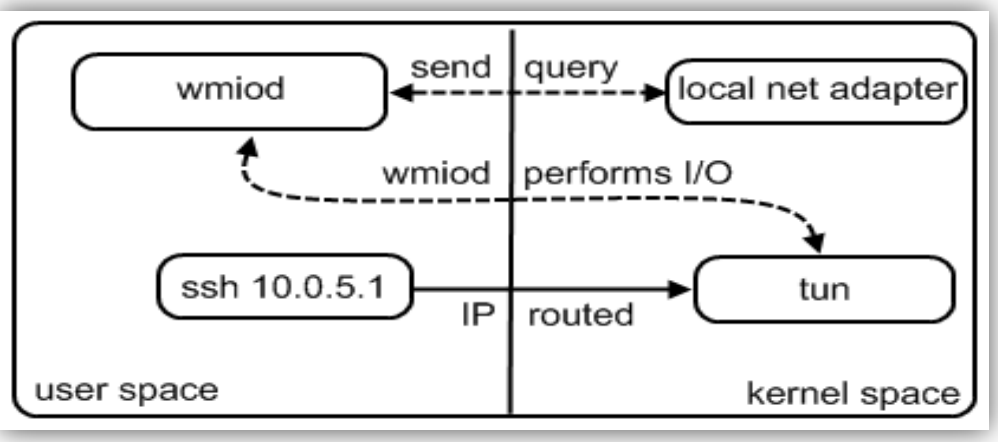
Εικόνα 15. DNS Query format

Πρέπει να διαλέξουμε ένα πεδίο στο μήνυμα που να επιτρέπει το μέγιστο αριθμό δεδομένων ώστε να έχουμε να στείλουμε όσο πιο πολλά upstream data. Σύμφωνα με το DNS Query Message που είδαμε παραπάνω ο ακόλουθος πίνακας θα μας δώσει την απάντηση.

Πίνακας 2. Μήκη πεδίων στο DNS Query Message

Πεδίο	Μέγιστο μήκος (bytes)	Χωρητικότητα (bits)
ID	2	16
Flags	2	16
QDCOUNT	2	16
ANCOUNT	2	16
NSCOUNT	2	16
ARCOUNT	2	16
QNAME	**256**	**2048**
QTYPE	2	16
QCLASS	2	16

Όπως είναι φανερό αυτό το πεδίο είναι το QNAME καθώς μας επιτρέπει να του περάσουμε 256 bytes δεδομένων. Γνωρίζοντας πλέον πως μπορεί να γίνει αποστολή δεδομένων παρακάτω θα δούμε πως μπορεί να γίνει η λήψη.



Εικόνα 16. Σύλληψη IP πακέτων από wmiod

2.3.2. Παράδοση downstream data

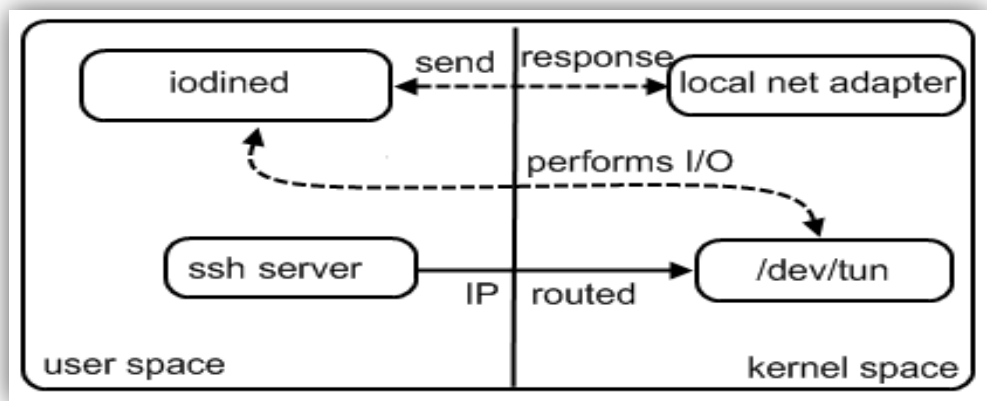
Στην ενότητα 1.2 είπαμε ότι το DNS εκτός του να προσφέρει διευθύνσεις μπορεί να προσφέρει και άλλες πληροφορίες. Αυτό επιτυγχάνεται στέλνοντας στον εξυπηρετητή DNS Query Messages με διαφορετικό TYPE. Έτσι τον κάνουμε να δημιουργήσει και να στείλει τον τύπο του Resource Record που εμείς θέλουμε.

Πρέπει να εκμεταλλευτούμε ένα TYPE που να προγραμματίζει τον εξυπηρετητή ονομάτων να κατασκευάζει ένα Resource Record με μεγάλη χωρητικότητα. Όσο πιο πολλά δεδομένα χωράει τόσο πιο μεγάλο θα είναι το πλήθος των downstream data ανά DNS Response Message. Υπάρχουν πολλά είδη TYPE και παρακάτω δίνεται μια σύντομη λίστα βάση του RFC 1035.

Πίνακας 3. Διάφορα είδη Resource Record

Τύπος	Τιμή	Είδος απάντησης	Μήκος
A	0x1	IPv4 Address	4
NS	0x2	Domain name	256
CNAME	0x5	Domain name	256
NULL	**0xA**	Raw un-encoded bytes	**65.535**
PTR	0xC	Domain name	256
TXT	0x10	ASCII TEXT	256

Μελετώντας τον παραπάνω πίνακα αμέσως βλέπουμε πως αν το πεδίο TYPE πάρει την τιμή 0xA έχουμε ένα NULL TYPE Resource Record. Μία τέτοια εγγραφή επιτρέπει στον εξυπηρετητή ονομάτων να αποθηκεύσει στο πεδίο RDATA δεδομένα μέγιστου μήκους 64 kilobyte. Δηλαδή, σύμφωνα με αυτό κάθε DNS Response Message που θα λαμβάνουμε θα ζυγίζει λίγο παραπάνω από 64 kilobyte μαζί με τα υπόλοιπα στοιχεία του μηνύματος. Βέβαια λόγω του περιορισμού που αναφέραμε στην ενότητα 1.2 τα δεδομένα αυτά δε μπορούν να ξεπεράσουν ένα όριο.



Εικόνα 17. Σύλληψη IP πακέτων από Iodined

2.4. Ανίχνευση του τούνελ

Ο καλύτερος τρόπος για να ανιχνευθεί ένα DNS τούνελ είναι να εκτελέσουμε στατιστικούς ελέγχους για χαρακτηριστικές ανωμαλίες στα DNS Messages. Αυτά τα χαρακτηριστικά μπορεί να είναι:

- ✓ Αυξημένος αριθμός από DNS queries & responses
- ✓ Το δεδομένα στο πεδίο Qname είναι ασυνήθιστα αυξημένα.
- ✓ Το hostname μέσα το πεδίο Qname συνήθως είναι γραμμένο σε μια γλώσσα. Η Base32 μας δείχνει κάτι που δε μοιάζει με ανθρώπινη γλώσσα.
- ✓ Το συνολικό μέγεθος των δεδομένων που μεταφέρεται μέσω της πόρτας udp/53 είναι πολύ περισσότερο από το σύνηθες.

Έχοντας μελετήσει πιθανούς τρόπους εντοπισμών των τούνελ και τρόπους ενθυλάκωσης δεδομένων στα DNS messages τώρα πρέπει να τα κάνουμε και στην πράξη. Το επόμενο κεφάλαιο θα μας δείξει γιατί και πώς θα πρέπει να μορφοποιήσουμε το payload ώστε να μπορέσει να εισαχθεί σε ένα DNS Query Message χωρίς να δημιουργήσουμε πρόβλημα στους εξυπηρετητές ονομάτων.

Κεφάλαιο τρίτο

Προετοιμασία και αποστολή Query

3.1. Συμπίεση

Επειδή έχουμε πολύ περιορισμένο bandwidth πρέπει να εξετάζουμε πάντα μεθόδους που μας βοηθάν να περνάμε όσο πιο πολλά payload ανά DNS Query. Από τα πρώτα πράματα που έρχεται στο μυαλό μας είναι η συμπίεση. Στην προκειμένη περίπτωση μπορούμε να συμπιέσουμε το πακέτο IP που επρόκειτο να στείλουμε. Αυτό γίνεται με χρήση της συμπίεσης zlib.

Η zlib είναι μια βιβλιοθήκη και χρησιμοποιείτε για συμπίεση δεδομένων. Γράφτηκε από τους Jean-Loup Gailly & Mark Adler και χρησιμοποιεί τον αλγόριθμο **deflate** ο οποίος ανήκει στην κατηγορία των lossless data compression algorithms οι οποίοι έχουν την ιδιότητα να επαναφέρουν ακριβώς τα γνήσια δεδομένα από τα συμπιεσμένα. Ο αλγόριθμος deflate αποτελεί υλοποίηση του **αλγόριθμου συμπίεσης LZ77** σε συνδυασμό με την **κωδικοποίηση Huffman**.

Ωστόσο, υπάρχει ένα σημαντικό μειονέκτημα όταν χρησιμοποιούμε αλγόριθμους συμπίεσης. Όταν τα προς συμπίεση δεδομένα είναι λίγα ο αλγόριθμος συμπίεσης δεν αποδίδει. Αποτέλεσμα αυτού είναι να παράγονται περισσότερα αντί για λιγότερα δεδομένα. Μια λύση θα μπορούσε να υλοποιηθεί στο Iodine Protocol που θα δούμε παρακάτω. Έχω ανοίξει το ticket #42 στον τομέα ανάπτυξης του πρωτοκόλλου. Ωστόσο δεν είναι άμεσα σχέδια των προγραμματιστές. Το ticket είναι διαθέσιμο στη διεύθυνση dev.kryo.se/iodine/ticket/42.

3.2. Κωδικοποίηση

Όπως δηλώνεται στο RFC 3696 τα hostnames επιτρέπεται να αποτελούνται μόνο από ένα μικρό υποσύνολο του ASCII character set γνωστό ως **LDH**. Τα αρχικά σημαίνουν **L**etters για τους χαρακτήρες A-Z a-z, **D**igits για τους χαρακτήρες 0-9 και **H**yphen για την παύλα “-”. Ο περιορισμός αυτός μας αναγκάζει να χρησιμοποιήσουμε μια κωδικοποίηση που να παράγει μόνο τους εν λόγω χαρακτήρες. Οι χαρακτήρες στο σύνολό τους είναι 36. Οπότε μας χρειάζονται έξι μπιτ για να αναπαραστήσουμε 36 διαφορετικούς χαρακτήρες. Το 2^6 όμως μπορεί να παράγει έως 64 χαρακτήρες. Ποιοι θα είναι αυτοί οι υπόλοιποι 28 χαρακτήρες όταν εμείς ζητάμε μόνο τους συγκεκριμένους 36. Επειδή δεν επιτρέπεται να

προσθέσουμε άλλους εκτός των συγκεκριμένων λόγω του περιορισμού η μόνη λύση είναι να χρησιμοποιήσουμε ένα μπιτ λιγότερο ώστε να παράγουμε ακριβώς 32 χαρακτήρες. Έτσι από τους 36 διαθέσιμους που επιτρέπεται να έχουμε δεν χρησιμοποιούμε τέσσερις όποιους θέλουμε και χρησιμοποιούμε μόνο τους υπόλοιπους 32. Αφού αυτοί οι 32 χαρακτήρες ανήκουν στους επιτρεπόμενους δεν έχουμε κανένα πρόβλημα. Την λύση για την παραγωγή αυτών μας την δίνει η άγνωστη σε πολλούς κωδικοποίηση **Base 32**.

Μέχρι τώρα είδαμε πως ένα πακέτο IP περνάει από διάφορα στάδια μέχρι να φτάσει στην τελική έγκυρη μορφή ώστε να μπορέσει να ενθυλακωθεί στο DNS Query. Οι παρακάτω εικόνες μας δείχνουν τα στάδια από τα οποία πέρασε ένα IP πακέτο προερχόμενο από ένα ssh session.

```
[ 552 bytes ]
45 00 02 28 01 3B 40 00 80 06 76 41 C0 A8 00 02 C0 A8 00 01 C5 4E 00 16 A9 19 E3 8A 65 55
03 12 50 18 00 43 E7 EC 00 00 00 00 02 64 0A 14 29 36 1C 9A 31 BC A7 A0 96 71 F9 4A 12 AA
D3 57 00 00 00 7E 64 69 66 66 69 65 2D 68 65 6C 6C 6D 61 6E 2D 67 72 6F 75 70 2D 65 78 63
68 61 6E 67 65 2D 73 68 61 32 35 36 2C 64 69 66 66 69 65 2D 68 65 6C 6C 6D 61 6E 2D 67 72
6F 75 70 2D 65 78 63 68 61 6E 67 65 2D 73 68 61 31 2C 64 69 66 66 69 65 2D 68 65 6C 6C 6D
61 6E 2D 67 72 6F 75 70 31 34 2D 73 68 61 31 2C 64 69 66 66 69 65 2D 68 65 6C 6C 6D 61 6E
2D 67 72 6F 75 70 31 2D 73 68 61 31 00 00 00 0F 73 73 68 2D 72 73 61 2C 73 73 68 2D 64 73 73
00 00 00 9F 61 65 73 32 35 36 2D 63 74 72 2C 61 65 73 32 35 36 2D 63 62 63 2C 72 69 6A 6E 64
61 65 6C 2D 63 62 63 40 6C 79 73 61 74 6F 72 2E 6C 69 75 2E 73 65 2C 61 65 73 31 39 32 2D
63 74 72 2C 61 65 73 31 39 32 2D 63 62 63 2C 61 65 73 31 32 38 2D 63 74 72 2C 61 65 73 31 32
38 2D 63 62 63 2C 62 6C 6F 77 66 69 73 68 2D 63 74 72 2C 62 6C 6F 77 66 69 73 68 2D 63 62
63 2C 33 64 65 73 2D 63 74 72 2C 33 64 65 73 2D 63 62 63 2C 61 72 63 66 6F 75 72 32 35 36 2C
61 72 63 66 6F 75 72 31 32 38 00 00 00 9F 61 65 73 32 35 36 2D 63 74 72 2C 61 65 73 32 35 36
2D 63 62 63 2C 72 69 6A 6E 64 61 65 6C 2D 63 62 63 40 6C 79 73 61 74 6F 72 2E 6C 69 75 2E
73 65 2C 61 65 73 31 39 32 2D 63 74 72 2C 61 65 73 31 39 32 2D 63 62 63 2C 61 65 73 31 32 38
2D 63 74 72 2C 61 65 73 31 32 38 2D 63 62 63 2C 62 6C 6F 77 66 69 73 68 2D 63 74 72 2C 62
6C 6F 77 66 69 73 68 2D 63 62 63 2C 33 64 65 73 2D 63 74 72 2C 33 64 65 73 2D 63 62 63 2C
```

Εικόνα 18. IP PACKET

```
[ 250 bytes ]
78 DA 63 60 E0 60 70 65 60 D2 60 7C E3 C0 D0 C0 56 3A E1 C0 0A 06 26 20 66 3C 7A 8C 41
EC 73 66 D9 FB F9 F1 D7 FF 06 48 30 38 B7 CF 60 00 02 A6 14 2E 91 55 99 07 6B 9D CD 3F 7F
FE 2F FE E8 6D 8F F7 B7 1D 40 E1 BA 94 CC B4 B4 CC 54 DD 8C D4 9C 9C DC C4 3C DD F4
A2 FC D2 02 DD D4 8A E4 8C C4 BC F4 54 DD E2 8C 44 23 53 33 1D 82 8A 0C B1 2A 31 34 C1
23 07 96 02 BA 80 BF B8 38 43 B7 A8 38 51 07 44 A7 14 17 03 85 E6 27 A6 16 03 AD D5 4D 2E
29 D2 81 31 93 92 75 8A 32 B3 F2 52 12 53 73 40 1C 87 9C CA E2 C4 92 FC 22 BD 9C CC 52
BD E2 54 90 32 43 4B 23 98 0E 30 13 A8 03 C4 34 B2 80 8B 82 98 40 D1 A4 9C FC F2 B4 4C
A0 6D 20 71 04 07 28 63 9C 92 5A 0C 16 85 30 40 26 14 25 A7 E5 97 16 81 02 01 CA 04 9A 32
34 9C 28 9F 91 9B 98 0C 89 80 0C 00 C3 1F BF 57
```

Εικόνα 19. Συμπιεσμένο IP PACKET

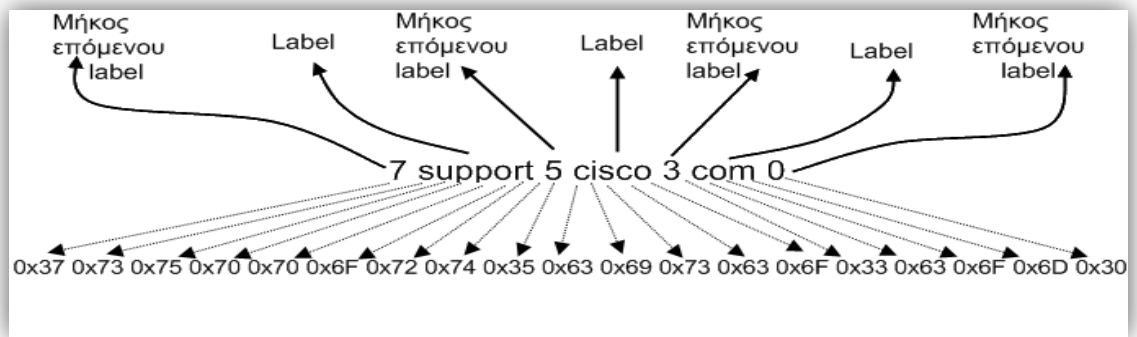
```
[ 255 bytes ]
3F 30 70 64 6E 67 67 79 68 61 6D 62 79 67 6B 79 67 73 6D 64 76 68 6B 79 64 69 6D 61 76 78 73
35 31 61 61 75 62 72 67 65 62 74 64 30 6A 77 61 65 63 31 78 33 6E 34 65 74 34 65 62 76 68 33
67 61 3F 61 6A 61 7A 74 34 32 32 61 67 65 62 71 65 76 79 69 73 35 68 6D 73 30 6A 68 30 69 66
30 67 76 76 32 7A 76 61 6D 78 34 34 76 79 66 6D 72 69 64 71 74 76 66 67 6D 77 73 30 6D 7A 76
67 33 62 74 3F 6B 6A 7A 68 67 32 79 73 34 6E 30 33 66 63 35 74 6A 69 66 78 6B 75 72 6C 73 69
7A 72 66 32 34 74 6B 6E 30 79 75 6D 79 73 72 76 67 6D 79 33 61 69 66 69 7A 6D 6A 6B 77 65
30 65 63 69 32 68 63 30 79 62 6C 75 61 66 35 78 63 32 6D 67 6E 7A 69 78 64 69 71 6F 72 62 68
63 73 6C 71 67 62 6C 67 75 34 74 62 6E 61 7A 6E 6B 78 67 30 32 6B 6B 73 6B 04 74 75 6E 33
04 76 6C 73 69 02 67 72 00
```

Εικόνα 20. Το συμπιεσμένο IP PACKET κωδικοποιημένο σε Base32

3.3. Μετατροπή σε έγκυρο όνομα και ενθυλάκωση

Έχοντας φέρει πλέον το επιθυμητό payload σε έγκυρη μορφή, μπορούμε να το αποθηκεύσουμε στο πεδίο QNAME. Υπενθυμίζουμε ότι το πεδίο αυτό επιβάλλεται να περιέχει κάτι σε μορφή domain name. Το RFC 608 που συντάχθηκε το 1974 δηλώνει πως το όνομα πρέπει να απαρτίζεται από μια ακολουθία από labels μέγιστου μήκους 63 bytes έκαστο και σε κάθε label πρέπει να προστίθεται στην αρχή του ένας αριθμός που προσδιορίζει το μήκος του label. Το domain name πρέπει να τερματιστεί με το byte 0x0 που δηλώνει το μήκος του τελευταίου label, δηλαδή ότι δεν υπάρχει άλλο label. Το συνολικό μήκος του domain name επιτρέπεται να είναι μέχρι 255 bytes.

Συνοψίζοντας τα παραπάνω ας δούμε πως φαίνεται σε εμάς αυτή η μετατροπή. Έστω ότι θέλουμε να βρούμε μια πληροφορία για το όνομα support.cisco.com. Το μετατρέπουμε σε 7support5cisco3com0 και αυτό θα πρέπει να το περάσουμε στο πεδίο QNAME ως μία ακολουθία από byte. Κώδικας στο παράρτημα στο Code snippet 5.

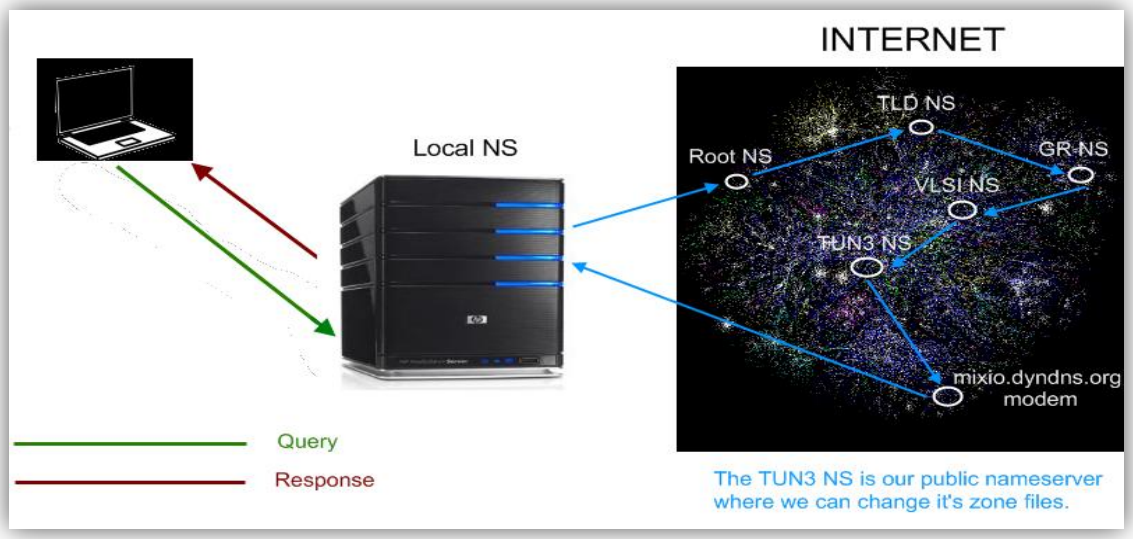


Εικόνα 21. Μετατροπή για εισαγωγή στο Qname

Το μόνο που μας έμεινε είναι να δημιουργήσουμε ένα DNS Query Message με την κλάση DnsPacket που περιγράφουμε στην παράγραφο 5.1 περνώντας στον δομητή της κλάσης το byte array.

3.4. Δρομολόγηση upstream και downstream data

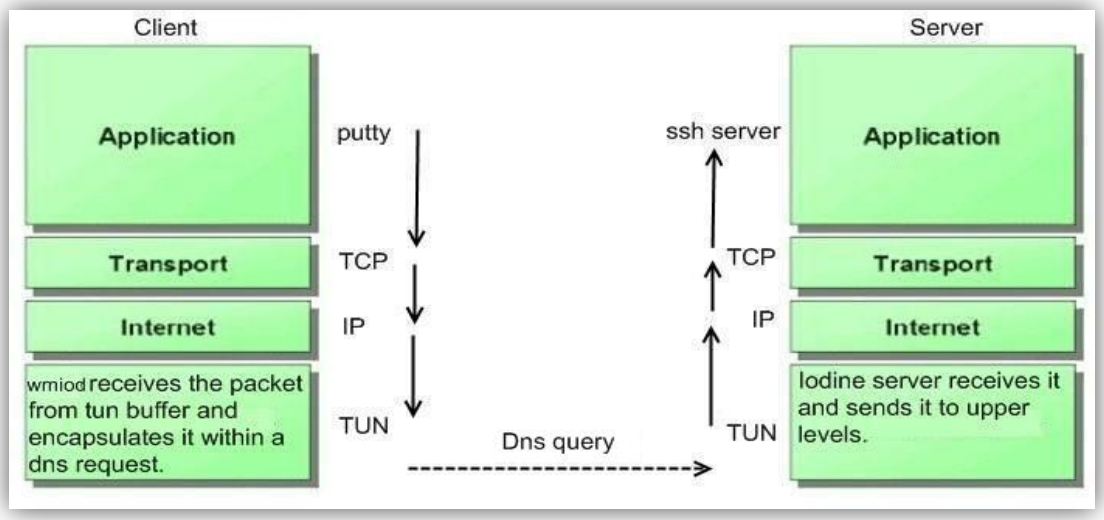
Το Dns Query Message αποτελεί μια ερώτηση και ο εξυπηρετητής ονομάτων αναλαμβάνει να αναζητήσει την απάντηση. Το query περιέχει ενθυλακωμένο μέσα του ένα πακέτο IP το οποίο θα δρομολογηθεί με τον τρόπο που δείξαμε στη παράγραφο 1.3 και θα φτάσει στον τελευταίο εμπλεκόμενο εξυπηρετητή ονομάτων ο οποίος είναι το μόνο μηχάνημα που μπορεί να στείλει οτιδήποτε προς τα πίσω σε εμάς. Εδώ, έρχεται η βασική απορία. Τι θα πάθει αυτό το πακέτο IP όταν εξαχθεί από τον εξυπηρετητή; Η απάντηση είναι απλή. Τίποτα. Ο εξυπηρετητής θα δει ότι το πεδίο QNAME του DNS Query Message δεν περιέχει κάποια ερώτηση που πρέπει να απαντηθεί και θα το απορρίψει σιωπηλά. Τι θα γινόταν όμως εάν είχαμε ένα μηχάνημα που θα μπορούσε να επεξεργαστεί κατάλληλα το QNAME και να σχηματίσει το αρχικό πακέτο IP; Με αυτό το σκεπτικό θέλουμε ένα ειδικό μηχάνημα να ακούει στην πόρτα udp/53 στο οποίο θα φθάνουν τα queries και θα είναι ικανό να εξάγει το πακέτο IP από αυτά. Η επόμενη ερώτηση που μας έρχεται στο μυαλό είναι πως θα δρομολογηθούν τα queries σε αυτό το μηχάνημα. Την λύση σε αυτό μας την δίνει η περίπτωση να έχουμε υπό την διαχείριση μας ένα δικό μας domain. Θα δίναμε μια ρύθμιση στον DNS server που διαχειρίζεται το domain μας η οποία θα έλεγε ότι για παράδειγμα το domain tun3.vlsi.gr μπορεί να εξυπηρετηθεί από τον εξυπηρετητή ονομάτων mixio.dyndns.org. Αυτομάτως, αυτός γίνεται ο τελευταίος εμπλεκόμενος και είναι ένα μηχάνημα στο διαδίκτυο υπό τον έλεγχο μας. Έτσι, όποιο query περιέχει το «tun3.vlsi.gr» θα φτάσει στην πόρτα udp/53 του mixio.dyndns.org.



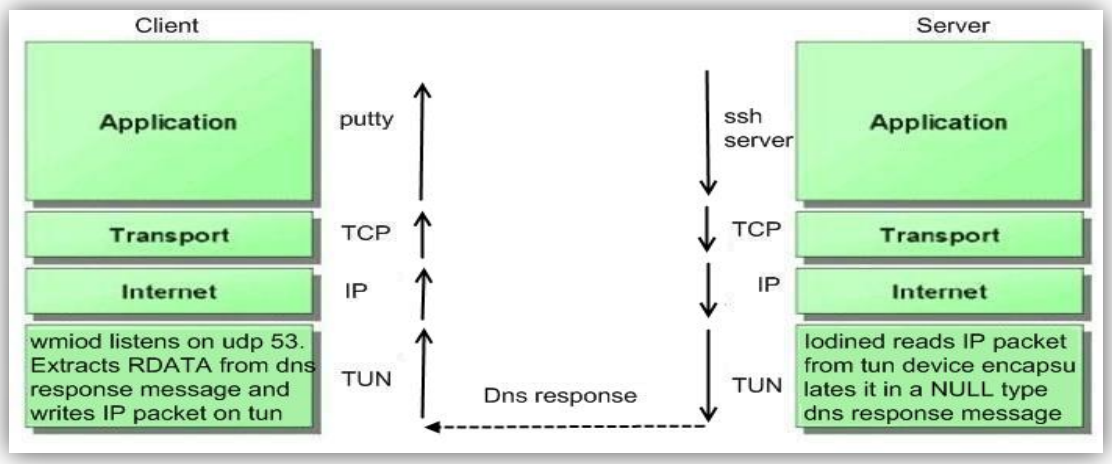
Εικόνα 22. Δρομολόγηση query στον rogue name server

Το μηχάνημα αυτό είναι ένας κατάλληλα διαμορφωμένος υπολογιστής που τρέχει τον Iodined server που ακούει και επεξεργάζεται ότι λαμβάνει στην πόρτα udp/53. Ο Iodined θα εξάγει το payload από το query, δηλαδή θα εξάγει το πακέτο IP που στείλαμε. Έπειτα θα το γράψει σε μία εικονική συσκευή και συγκεκριμένα στην **/dev/tun** η οποία είναι λειτουργεί ομοίως με την συσκευή του Tun/Tap driver.

Τελικά το πακέτο IP που στείλαμε ως payload στο DNS Query Message γράφεται στο /dev/tun του server και από εκεί περνάει στην TCP/IP stack ανεβαίνοντας τα επίπεδα του OSI. Αν υπάρξει κάποια απάντηση εξ' αιτίας του πακέτου IP που στείλαμε αυτή η απάντηση θα είναι ένα άλλο πακέτο IP που θα γραφεί στο /dev/tun από την TCP/IP stack του συστήματος του συστήματος του server. Ο Iodined θα το διαβάσει από το /dev/tun, θα το συμπιέσει και δίχως κάποια κωδικοποίηση θα το ενθυλακώσει στο πεδίο RDATA του DNS Response Message. Έπειτα θα το στείλει ως απάντηση για το Query Message που του στείλαμε. Το Response Message θα φθάσει απευθείας στην πόρτα udp/53 του εξυπηρετητή ονομάτων του ISP μας. Ο τελευταίος το προωθεί στο μηχάνημα που έκανε το Query Message. Στο μηχάνημα αυτό τρέχει η εφαρμογή **wmiod** που ακούει στην udp/53 και ότι λάβει το επεξεργάζεται καταλλήλως, εξάγει το συμπιεσμένο πακέτο IP από το πεδίο RDATA του Response Message και τέλος το γράφει στην τοπική εικονική συσκευή του μηχανήματος.

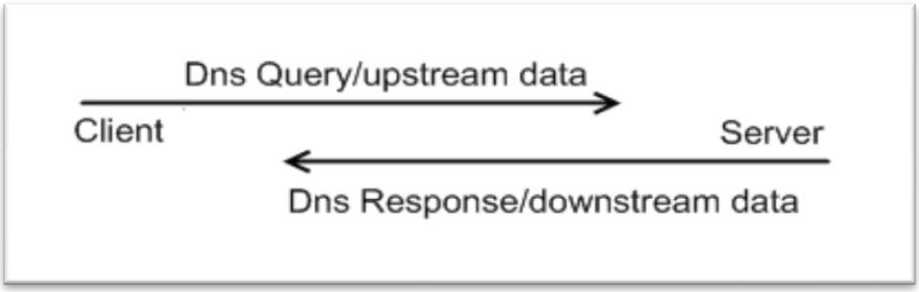


Εικόνα 23. Αποστολή πακέτου IP



Εικόνα 24. Λήψη πακέτου IP

Έτσι μιμηθήκαμε μια πλήρως αμφίδρομη επικοινωνία επιπέδου IP. Ως φορέα για τα upstream data χρησιμοποιούμε πεδίο QNAME του DNS query και ως φορέα για τα downstream data χρησιμοποιούμε το πεδίο RDATA του NULL DNS response.



Εικόνα 25. Upstream / Downstream data

Πλέον έχουμε κατασκευάσει ένα τούνελ όπου μεταδίδονται πακέτα IP χρησιμοποιώντας queries και responses πάνω σε ένα μονοπάτι που ελέγχεται από το Domain Name System. Ακριβώς, για τον λόγο ότι το όλο σύστημα ελέγχεται από το DNS πρέπει να πληρούμε όλους τους κανόνες που διέπουν την ομαλή λειτουργία του. Γίνεται σαφές ότι αν δεν ακολουθήσουμε τους κανόνες που έχουν ορίσει οι δημιουργοί αυτού του συστήματος τότε αυτό δεν θα μας επιτρέψει να το χρησιμοποιήσουμε. Για αυτούς τους κανόνες σε συνδυασμό με το τούνελ που θα τους χρησιμοποιήσει για να δρομολογήσει τα πακέτα DNS έχει κατασκευαστεί εδώ και μερικά χρόνια ένα πρωτόκολλο από κάποιους Σουηδούς προγραμματιστές που αναφέρονται στις ευχαριστίες. Το πρωτόκολλο αυτό ονομάζεται **Iodine protocol** και είναι το αντικείμενο που θα μας απασχολήσει στο επόμενο κεφάλαιο.

3.5. Θέματα ανωνυμίας

Στην ενότητα 2.4 είδαμε ότι ένα τέτοιο τούνελ μπορεί να ανακαλυφθεί σχετικά εύκολα. Έτσι όταν ανιχνευθεί το τούνελ θα φανεί αμέσως ότι γίνεται χρήση του tun3.vlsi.gr. Επειδή το tun3.vlsi.gr είναι δηλωμένο στον εξυπηρετητή ονομάτων που διαχειρίζεται το δικό μας ιδιόκτητο domain name αμέσως αποκαλύπτεται ο ιδιοκτήτης του.

Για σωστή ανωνυμία θα πρέπει κάποιος πρώτον να έχει πλήρως ανώνυμη πρόσβαση στην διαχείριση του domain μας και δεύτερων πλήρως ανώνυμη πρόσβαση σε ένα μηχάνημα στο διαδίκτυο που θα του εγκαταστήσει τον Iodine server. Από αυτό καταλαβαίνουμε ότι είναι αρκετά δύσκολο αλλά όχι και απίθανο.

Κεφάλαιο τέταρτο

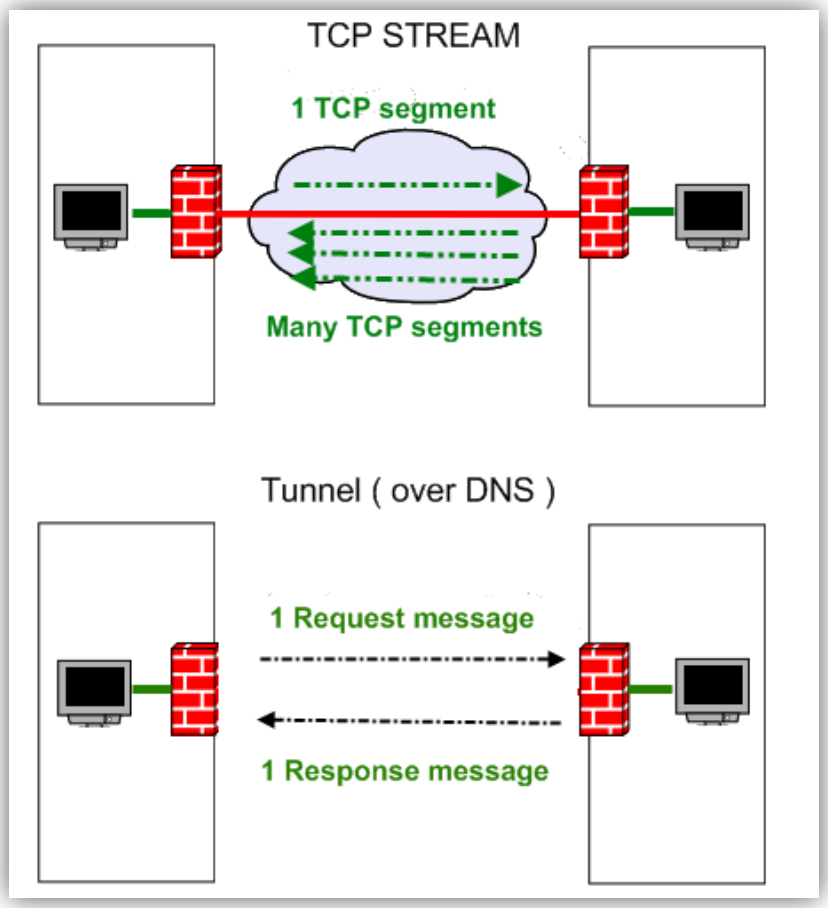
Πρωτόκολλο Iodine

4.1. Η έννοια του πρωτοκόλλου

Γενικότερα η έννοια του πρωτοκόλλου καθορίζει ένα σύνολο από κανόνες που πρέπει να ακολουθήσουμε αν θέλουμε να πραγματοποιήσουμε κάτι συγκεκριμένο. Τη λέξη πρωτόκολλο την συναντούμε σχεδόν παντού. Στην διπλωματία, την πολιτική, σε ένα νοσοκομείο, σε όλων των ειδών τις επικοινωνίες και σαφώς στον κόσμο της πληροφορικής. Έτσι, μπορούμε να δώσουμε πολλούς επιμέρους ορισμούς για κάθε έναν από τους παραπάνω τομείς της ζωής μας. Ειδικότερα για τον χώρο τον τηλεπικοινωνιών, πρωτόκολλο ονομάζεται ένα σύνολο από πρότυπους κανόνες για την αναπαράσταση δεδομένων, την σήμανση, την εξουσιοδότηση και εντοπισμό σφαλμάτων που απαιτούνται για να μεταδώσουμε πληροφορία μέσω ενός καναλιού επικοινωνίας. Στην περίπτωση μας, όπως κάθε πρωτόκολλο περιγράφει κανόνες έτσι και το πρωτόκολλο Iodine περιγράφει εκείνους τους κανόνες που πρέπει να υφίστανται για την ομαλή διεξαγωγή της επικοινωνίας μέσω του DNS. Το πρωτόκολλο περιγράφεται στη διεύθυνση svn.kryo.se/iodine/doc/proto_00000402.txt και υλοποιείται στην κλάση Iodine του πηγαίου κώδικα.

Για να δούμε γιατί μας συμφέρει να χρησιμοποιήσουμε το Iodine πρέπει καταρχήν να αναφέρουμε τα τέσσερα βασικότερα στοιχειά της επικοινωνίας μέσω του DNS και για την καλύτερη κατανόηση τους θα τα συγκρίνουμε με ένα πιο οικείο για εμάς πρωτόκολλο το TCP.

- Το DNS μεταφέρει ανεξάρτητα μηνύματα χωρίς κάποιον έλεγχο ενώ το TCP εφαρμόζει μια ροή δεδομένων όπου τα μηνύματα ελέγχονται και συνδέονται.
- Στο DNS ένα σημείο στέλνει ένα query και ο server απαντάει μόνο με ένα response ενώ στο TCP ένα σημείο στέλνει ένα segment και ο server απαντάει με όσα segments χρειαστεί.
- Το TCP επιτρέπει οποιαδήποτε από τις δυο πλευρές να μιλήσει πρώτη ενώ στο DNS ο server μπορεί να μιλήσει αν μόνο του το ζητήσει ο client.
- Το TCP μεταδίδει οποιονδήποτε χαρακτήρα από τους 256 που μπορεί να δώσει ένα byte ενώ το DNS μπορεί να μεταδώσει ένα περιορισμένο σύνολο χαρακτήρων που κατασκευάζεται από την κωδικοποίηση Base 32.



Εικόνα 26. TCP stream και DNS τούνελ

4.2. Cache Miss Counter

Σε μια αναδρομική αναζήτηση οι εξυπηρετητές ονομάτων πρώτα ψάχνουν στις τοπικές μνήμες τους για όμοια queries που ενδεχομένως έγιναν στο παρελθόν και έχουν ήδη εξυπηρετηθεί. Αν συμβεί κάτι τέτοιο λέμε ότι έχουμε **cache hit** και ο εξυπηρετητής ονομάτων δεν δρομολογεί το query στον επόμενο κατάλληλο. Ο Cache Miss Counter ή CMC είναι ένας 16-μπιτος αριθμός και χρησιμοποιείτε για να αποφεύγονται τα cache hits. Τον αρχικοποιούμε με έναν τυχαίο αριθμό και κάθε φορά που τον χρησιμοποιούμε τον αυξάνουμε κατά ένα. Τον ενσωματώνουμε σε συγκεκριμένα DNS Query Messages που αργότερα θα δούμε ότι από την φύση τους είναι όμοια. Επομένως, τα μηνύματα εξαιτίας του CMC είναι πάντα διαφορετικά μεταξύ τους ούτος ώστε ο εξυπηρετητής να μην έρθει πότε σε κατάσταση cache hit και δεν τα δρομολογήσει.

4.3. Handshake

Η πρώτη λειτουργία που πραγματοποιείται μεταξύ του client και του server είναι ένα handshake και περιλαμβάνει τα τρία εξής βήματα:

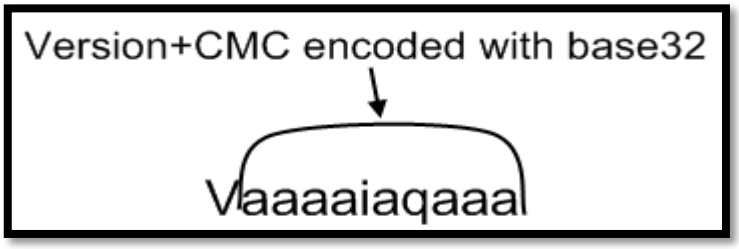
1. την εξακρίβωση της έκδοσης του πρωτόκολλου του χρηστή
2. την εξακρίβωση της ταυτότητας του χρηστή
3. τον έλεγχο των χαρακτήρων που μπορούν να μεταδοθούν μέσα στο τούνελ

4.3.1. Send Version

Κατά την εξακρίβωση της έκδοσης ο client στέλνει τον χαρακτήρα ‘v’ ή ‘V’. Έπειτα προσαρτεί μια ακολουθία base 32 χαρακτήρων οι όποιοι συμβολίζουν κωδικοποιημένα την έκδοση του πρωτόκολλου και ένα CMC. Η έκδοση πρέπει να είναι υπο μορφη big endian και αποτελείται από τέσσερα byte τα οποια παράγονται με χρήση Right Shift και ANDing με την στάθερα 255. Στην παρούσα φάση η έκδοση του πρωτοκόλλου Iodine είναι 00000402. Το CMC αποτελείται από δύο byte και παράγεται όπως και η εκδοση.

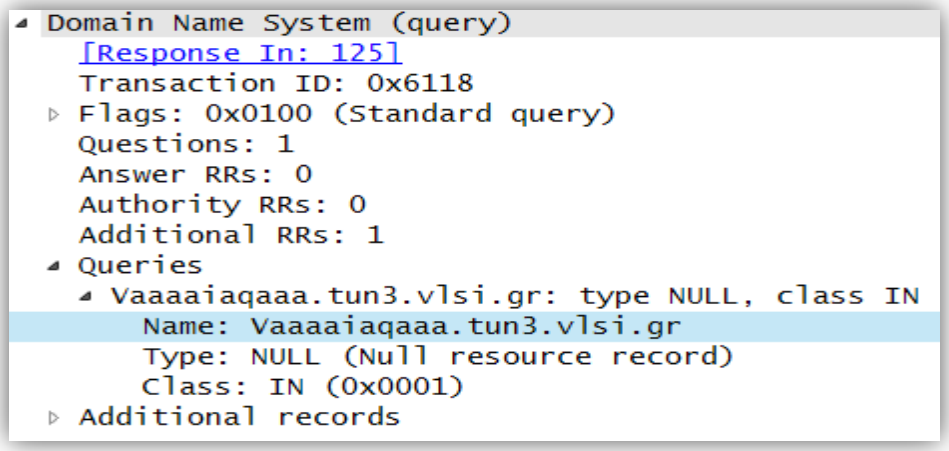
Version Byte 1: 0x00000402 >> 24 & 0xff
Version Byte 2: 0x00000402 >> 16 & 0xff
Version Byte 3: 0x00000402 >> 8 & 0xff
Version Byte 4: 0x00000402 >> 0 & 0xff

CMC Byte 1: 0x0 >> 8 & 0xff
CMC Byte 2: 0x0 >> 0 & 0xff



Εικόνα 27. Handshake Version

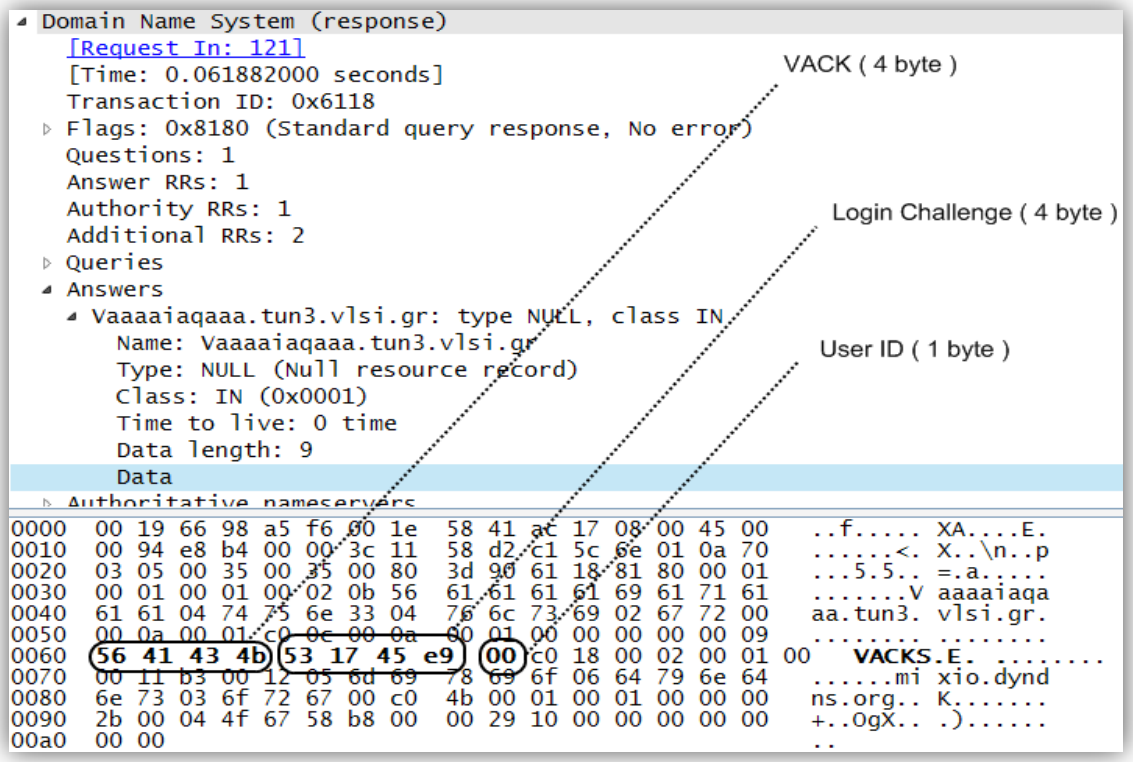
Μην ξεχνάμε ότι στο « Vaaaaiaqaaa » προστίθεται το όνομα tun3.vlsi.gr και πλέον το « Vaaaaiaqaaa.tun3.vlsi.gr » είναι αυτό που αποθηκεύεται στο πεδίο QNAME του query. Το όνομα θα κάνει τους εξυπηρετητές ονομάτων παγκοσμίως να δρομολογήσουν το payload στον υπό δικό μας έλεγχο server.



Εικόνα 28. Σύλληψη του Handshake Version στο wireshark

Ο server αφού λάβει το payload θα διαβάσει τον πρώτο χαρακτήρα για να εξετάσει τι είδους πληροφορία του στέλνουμε. Έχοντας στείλει το « Vaaaaiaqaaa » αναγνωρίζει το 'V' καταλαβαίνει ότι πρόκειται για επιβεβαίωση έκδοσης του πρωτόκολλου από τον client και προωθεί το υπόλοιπο payload « aaaaiaqaaa » στη κατάλληλη μέθοδο του προγράμματος του server. Εκεί γίνεται μια αντιστροφή της διαδικασίας που περιγράψαμε στον client. Δηλαδή, το πρόγραμμα του server θα αποκωδικοποιήσει τους base 32 χαρακτήρες σε άπλα byte και θα διαβάσει τις τιμές. Έτσι, αναλόγως τα byte που έλαβε απαντάει με μια από τις παρακάτω τρεις σημάνσεις που θα περιγράψουμε στην συνεχεία.

Σήμανση ***VACK*** με μήκος τεσσάρων byte, που σημαίνει version acknowledged ή έκδοση επιβεβαιώθηκε, προσαρτημένο από τέσσερα ακόμη byte για το login challenge που θα χρειαστεί αργότερα κατά την εξακρίβωση της ταυτότητας του και από ένα τελευταίο byte για το user id του νέου χρηστή. Εδώ πρέπει να σημειωθεί ότι ο Iodine server υποστηρίζει παράλληλους χρηστές. Η συγκεκριμένη έκδοση του πρωτόκολλου υποστηρίζει οχτώ χρηστές, από τον χρηστή μηδέν ως τον εφτά.



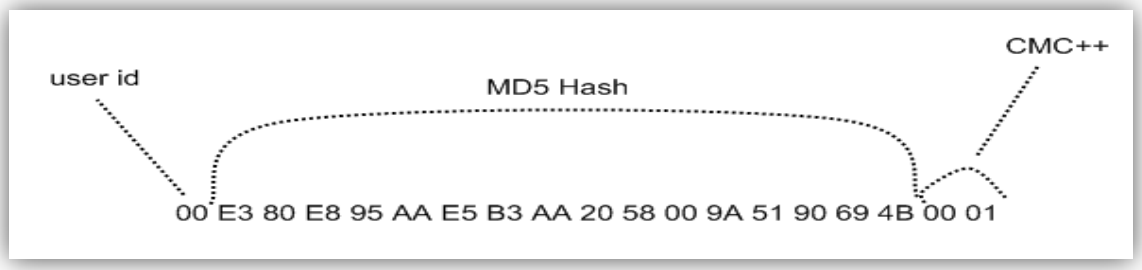
Εικόνα 29. Σύλληψη του Version Response στο wireshark

Σήμανση ***VNAK*** με μήκος τεσσάρων byte, που σημαίνει version not acknowledged ή έκδοση δεν επιβεβαιώθηκε, προσαρτημένο από τέσσερα επιπλέον byte τα όποια περιγράφουν την έκδοση του πρωτόκολλου του server. Έτσι, ο χρήστης βλέπει ποια έκδοση υποστηρίζει ο server και αναβαθμίζει τον client του.

Σήμανση ***VFUL*** με μήκος τεσσάρων byte, που σημαίνει server full προσαρτημένο από τέσσερα επιπλέον byte τα όποια περιγράφουν τον μέγιστο υποστηριζόμενο αριθμό clients. Κώδικας στο παράρτημα στο Code snippet 11.

4.3.2. Send Login

Κατά την εξακρίβωση της ταυτότητας του χρηστή ο client πρέπει να στείλει στον server δεκαεννέα byte τα όποια περιέχουν το user id, ένα MD5 hash και τέλος ένα CMC όπως φαίνεται στην επόμενη εικόνα.



Εικόνα 30. UserID και MD5 Hash

Το user id έχει μήκος ένα byte και αποκτήθηκε από το βήμα ένα. Το CMC αποτελειται από δυο byte και παραγεται όπως και το προηγουμενο CMC μονο που τωρα το αυξησαμε κατά ένα. Το MD5 hash έχει μήκος δεκαέξι byte και στην ουσία είναι το hash μιας ακολουθίας δεδομένων που θα εξηγήσουμε παρακάτω.

Για την ακολουθία αυτή το πρωτόκολλο λέει πως αρχικά πρέπει να δημιουργήσουμε οχτώ επαναλήψεις του login challenge που αποκτήσαμε στο πρώτο βήμα. Το login challenge έχει μήκος 4 byte και με τις 8 επαναλήψεις του παράγουμε ένα πινάκα Α χωρητικότητας 32 στοιχείων. Στην συνεχεία περνούμε το password που έδωσε ο χρήστης και το αποθηκεύουμε σε ένα πινάκα Β χωρητικότητας 32 στοιχείων και αυτός. Τέλος, για κάθε ένα από τα στοιχειά του πινάκα Α υπολογίζουμε την τιμή XOR με το αντίστοιχο στοιχειό του πινάκα Β. Αναλυτικότερα, όπως είδαμε στην εικόνα 28 το login challenge αποτελείται από τα παρακάτω byte:

[0x53] [0x17] [0x45] [0xε9]

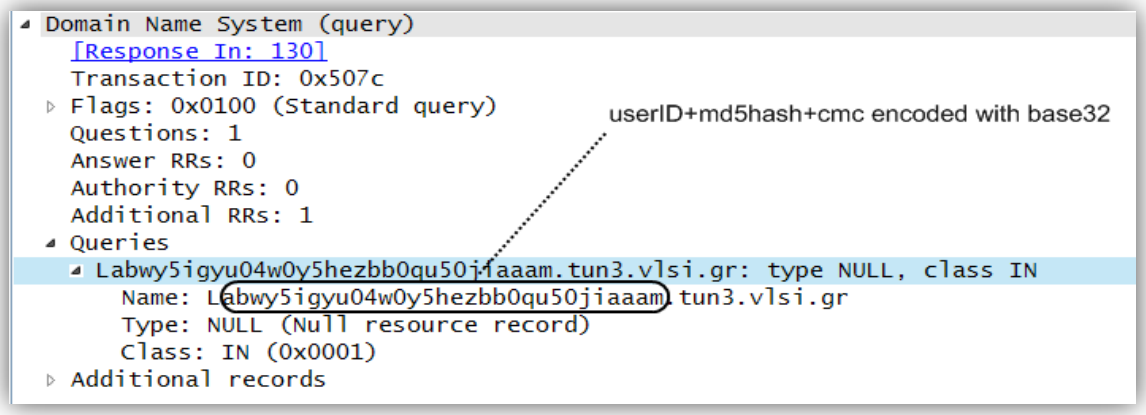
Έτσι, οι οχτώ επαναλήψεις του θα μας δώσουν την παρακάτω ακολουθία:

Byte 1: 0x53	Byte 5: 0x53	Byte 9: 0x53	Byte 13: 0x53
Byte 2: 0x17	Byte 6: 0x17	Byte 10: 0x17	Byte 14: 0x17
Byte 3: 0x45	Byte 7: 0x45	Byte 11: 0x45	Byte 15: 0x45
Byte 4: 0xe9	Byte 8: 0xe9	Byte 12: 0xe9	Byte 16: 0xe9
Byte 17: 0x53	Byte 21: 0x53	Byte 25: 0x53	Byte 29: 0x53
Byte 18: 0x17	Byte 22: 0x17	Byte 26: 0x17	Byte 30: 0x17
Byte 19: 0x45	Byte 23: 0x45	Byte 27: 0x45	Byte 31: 0x45
Byte 20: 0xe9	Byte 24: 0xe9	Byte 28: 0xe9	Byte 32: 0xe9

Αποθηκεύουμε αυτήν την ακολουθία στον πινάκα Α. Έχοντας και τον πινάκα Β που αποτελεί το password μπορούμε πλέον να υπολογίζουμε την τελική ακολουθία και να την αποθηκεύσουμε σε έναν τρίτο πινάκα.

```
for (int i = 0; i < 32; i++)
{
        πινακαςΓ[i] = πινακαςΑ[i] XOR πινακαςΒ[i];
}
```

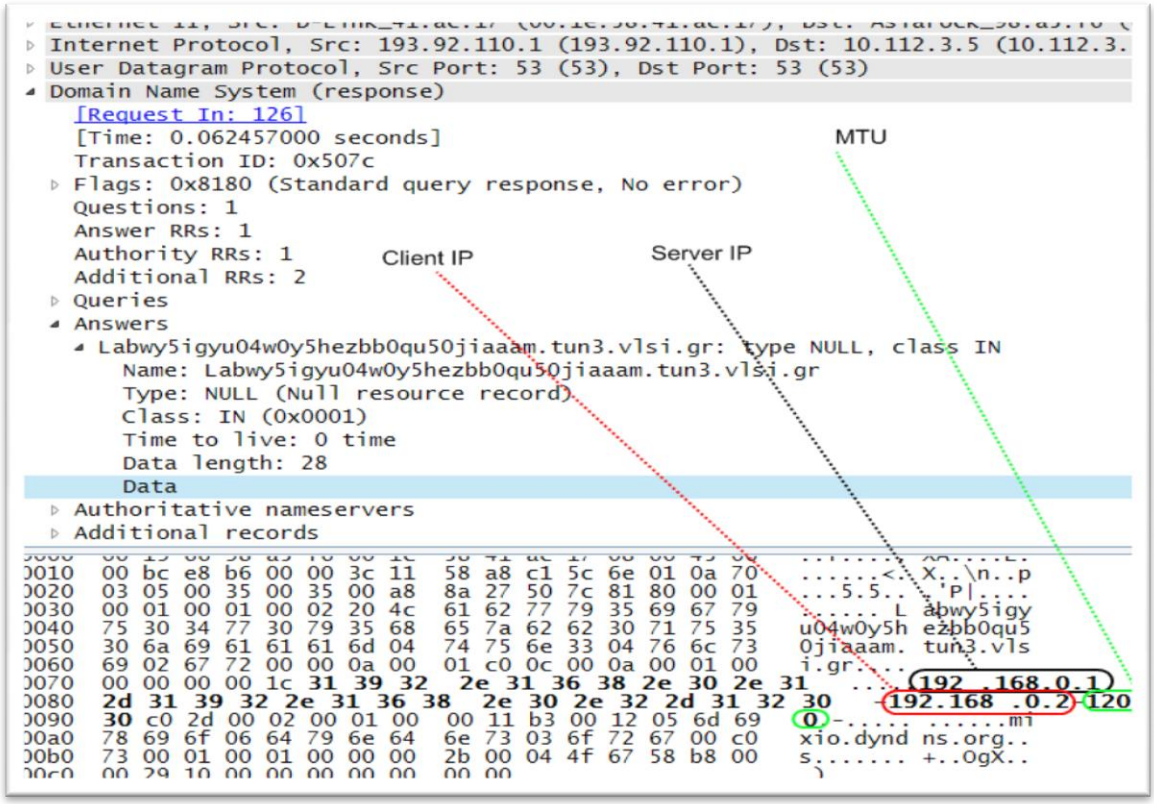
Ο νέος πινάκας Γ μεγέθους 32 byte είναι τα δεδομένα που θα περάσουμε σε μια συνάρτηση οπού θα μας υπολογίσει το MD5 hash. Αφού αποκτήσουμε και το hash είμαστε έτοιμοι να κωδικοποιήσουμε με base 32 το user id, το hash και το CMC. Αυτό που θα στείλουμε τελικώς είναι ο χαρακτήρας ‘l’ ή ‘L’ προσαρτημένος από τα κωδικοποιημένα δεδομένα.



Εικόνα 31. Σύλληψη του UserID και MD5 Hash στο wireshark

Ο server αφού λάβει το payload θα διαβάσει τον πρώτο χαρακτήρα για να εξετάσει τι είδους πληροφορία του στέλνουμε. Έχοντας στείλει την ακολουθία « Law5igyu04w0y5hezbb0qu50jiaaam » αναγνωρίζει το ‘L’ καταλαβαίνει ότι πρόκειται για επιβεβαίωση ταυτότητας του client και προωθεί το υπόλοιπο payload « abwy5igyu04w0y5hezbb0qu50jiaaam » στην κατάλληλη μέθοδο του προγράμματος του server. Εκεί γίνεται μια αντιστροφή της διαδικασίας που περιγράψαμε στον client. Δηλαδή, το πρόγραμμα του server θα αποκωδικοποιήσει τους base 32 χαρακτήρες θα πάρει τα 19 byte και θα διαβάσει τις τιμές τους. Έτσι, αναλόγως τα byte που έλαβε απαντάει ακολούθως:

Σήμανση ***LNAK*** με μήκος τεσσάρων byte, που σημαίνει login not acknowledged ή ταυτότητα δεν επιβεβαιώθηκε. Αν δεν απαντήσει με LNAK σημαίνει ότι επιβεβαιώθηκε ο χρήστης και στην απάντηση περιέχονται η IP του server, η IP του client και τέλος το MTU του client. Κώδικας στο παράρτημα στο Code snippet 13.



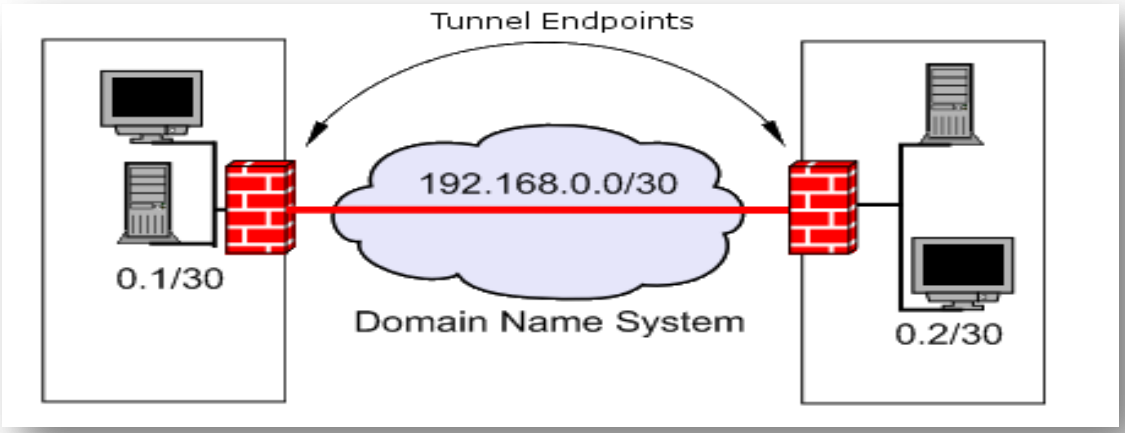
Εικόνα 32. Σύλληψη του Login Response στο wireshark

4.3.3. Send Case Check

Κατά τον έλεγχο των χαρακτήρων που μπορούν να μεταδοθούν μέσα από το DNS τούνελ ο client στέλνει την ακολουθία "zZaAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyY123-4560789" και ο server του απαντάει απλώς με την ίδια ακολουθία. Αυτή η λειτουργία θα χρειαστεί σε μελλοντικές εκδόσεις του πρωτοκόλλου οπού θα χρησιμοποιηθεί η κωδικοποίηση base 64. Στην τρέχουσα έκδοση δεν μας παρέχει κάτι και για τον λόγο αυτό δεν θα αναλυθεί περαιτέρω. Αφού τελειώσει και το βήμα τρία ολοκληρώνεται το Handshake. Όλη η διαδικασία του προγραμματίστηκε σε μια μέθοδο που ορίζεται ως **Handshake()**. Αυτή καλεί

για κάθε ένα από τα τρία βήματα τις αντίστοιχες μεθόδους. Τις εξετάζει μια-μια και αν οποιαδήποτε από τις τρεις επιστρέψει FALSE ακυρώνει όλο το Handshake.

Αν τελικά ολοκληρωθεί επιτυχώς, αυτό που αποκτήσαμε ως πληροφορία από τον server είναι η δική μας διεύθυνση IP, αυτή δηλαδή που θα ρυθμίσουμε στην εικονική συσκευή μας. Έτσι, έχουμε πλέον δυο άκρα που ανήκουν σε κάποιο private δίκτυο και επικοινωνούν μεταξύ τους ανταλλάσοντας IP πακέτα τα οποία έχουμε αναλάβει να μεταφέρουμε μέσω της παγκόσμιας υπηρεσίας DNS την οποία εκμεταλλευόμαστε κατάλληλα για να χρησιμεύσει σ' εμάς ως ένα transport network.



Εικόνα 33. Endpoints στο τούνελ

4.4. Domain Name fragmentation

Στο κεφάλαιο 3 μελετήσαμε τα στάδια από τα οποία πρέπει να περάσει ένα πακέτο IP προκειμένου να έρθει σε έγκυρη μορφή για να είναι ικανό να ενθυλακωθεί στο DNS Query Message.

Επίσης, στην παράγραφο 3.3 αναλύσαμε την μορφή που έχει ένα host name μέσα στο πεδίο QNAME του DNS Query Message. Είδαμε ότι το QNAME είναι ικανό να χωρέσει μέχρι 255 byte. Εδώ παρουσιάζεται ένα εύλογο ερώτημα. Τι θα γίνει σε περίπτωση που το κωδικοποιημένο πακέτο IP έχει μέγεθος μεγαλύτερο από αυτό που μπορεί να χωρέσει το QNAME; Οι απαντήσεις θα μπορούσαν να είναι δυο.

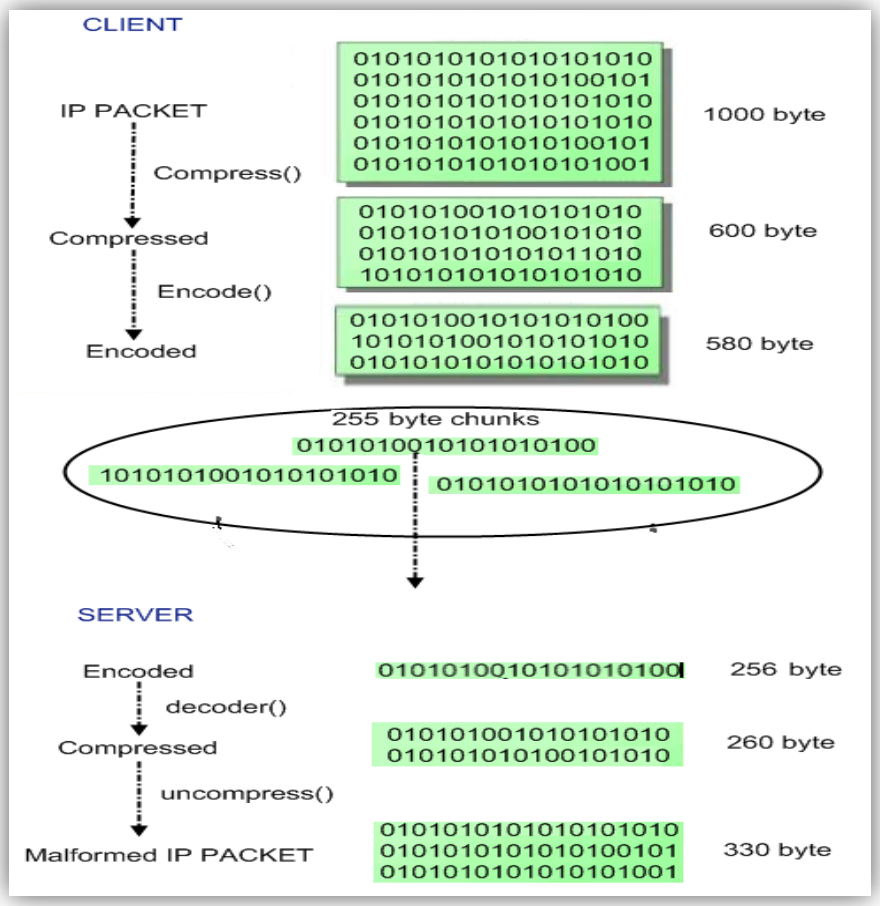
- ✪ να ενθυλακώσουμε παραπάνω από 255 byte στο QNAME
- ✪ να διαιρέσουμε το κωδικοποιημένο πακέτο IP σε δόσεις των 255 byte

Η πρώτη λύση προφανώς δεν μπορεί να υλοποιηθεί διότι έτσι παραβιάζουμε τις αρχές του πρωτόκολλου του DNS. Ωστόσο, η δεύτερη λύση περιγράφει κάτι που μπορούμε να προγραμματίσουμε χωρίς να παραβούμε κάποιον κανόνα. Ας δούμε ένα τέτοιο κωδικοποιημένο πακέτο και ας θεωρήσουμε ότι έχει μέγεθος 580 byte.

pdnggyhambygkyeqxw3jvanbqeyuba4iswaqtcazhempj7rtgmoggyaaabh6cbgggffw
ieqdkyeqxw3jvanbqeyuba4iswaqtcazhyeqxw3jvanbqeyuba4iswmnbvcxdfghjjhgfdt
qeyuba4iswaqtcazhempj7rtgmoggyaaabh6kjhgfdsadfghjkjhgfdcvbnmhgfdertyuikjh
gfd78fhjmbvcdfgjhgdhjhdfhflilioiirtwersccbgqtcazhyeqxw3jvanbqeyubhambygkyeq
xw3jvanbqeykyeqxw3jvanbqeyuba4i

Εικόνα 34. Base32 Encoded packet

Σύμφωνα με την δεύτερη λύση αυτό που πρέπει να κάνουμε είναι να τραβάμε και να στέλνουμε κάθε φορά 255 byte. Με πιο άπλα λόγια θα στέλνουμε όλο το κωδικοποιημένο πακέτο σε δόσεις των 255 byte η έκαστη. Το πρωτόκολλο Iodine ονομάζει αυτές τις δόσεις chunks. Όμως, με αυτόν τον τρόπο προκύπτει ένα άλλο πρόβλημα. Καθώς στέλνουμε τα chunks στην ουσία στέλνουμε το αρχικό πακέτο IP σε κομμάτια. Έτσι, ο Iodine server θα λάβει στην πόρτα udp/53 τρία DNS Query Messages το καθένα από τα οποία μεταφέρει και ένα chunk. Αν ο Iodine server επιχειρήσει να αποκωδικοποιήσει το κάθε chunk ξεχωριστά θα πάρει ένα δύσμορφο μη συμβατό πακέτο IP. Η επόμενη εικόνα δείχνει σχηματικά αυτή την περίπτωση.



Εικόνα 35. Λανθασμένη λήψη και αποκωδικοποίηση

Την λύση γι' αυτό το πρόβλημα μας την δίνει το Iodine. Η λύση αναφέρει ότι, κάθε φορά που στέλνουμε ένα chunk βάζουμε στην αρχή του ένα ειδικό χαρακτήρα. Όταν το chunk που στέλνουμε δεν είναι το τελευταίο κομμάτι προσθέτουμε στην αρχή του το χαρακτήρα "0". Όταν το chunk που στέλνουμε είναι το τελευταίο τότε βάζουμε τον χαρακτήρα "1". Έτσι, το κωδικοποιημένο πακέτο της εικόνας 3χ θα σταλθεί ως εξής:

0 pdnggyhambygkyeqxw3jvanbqeyuba4iswaqtcazhempj7rtgmoggyaaabh6cbgggffw
ieqdkyeqxw3jvanbqeyuba4iswaqtcazhyeqxw3jvanbqeyuba4iswmnbvcxdfghjjhgfdt

Εικόνα 36. Πρώτο chunk

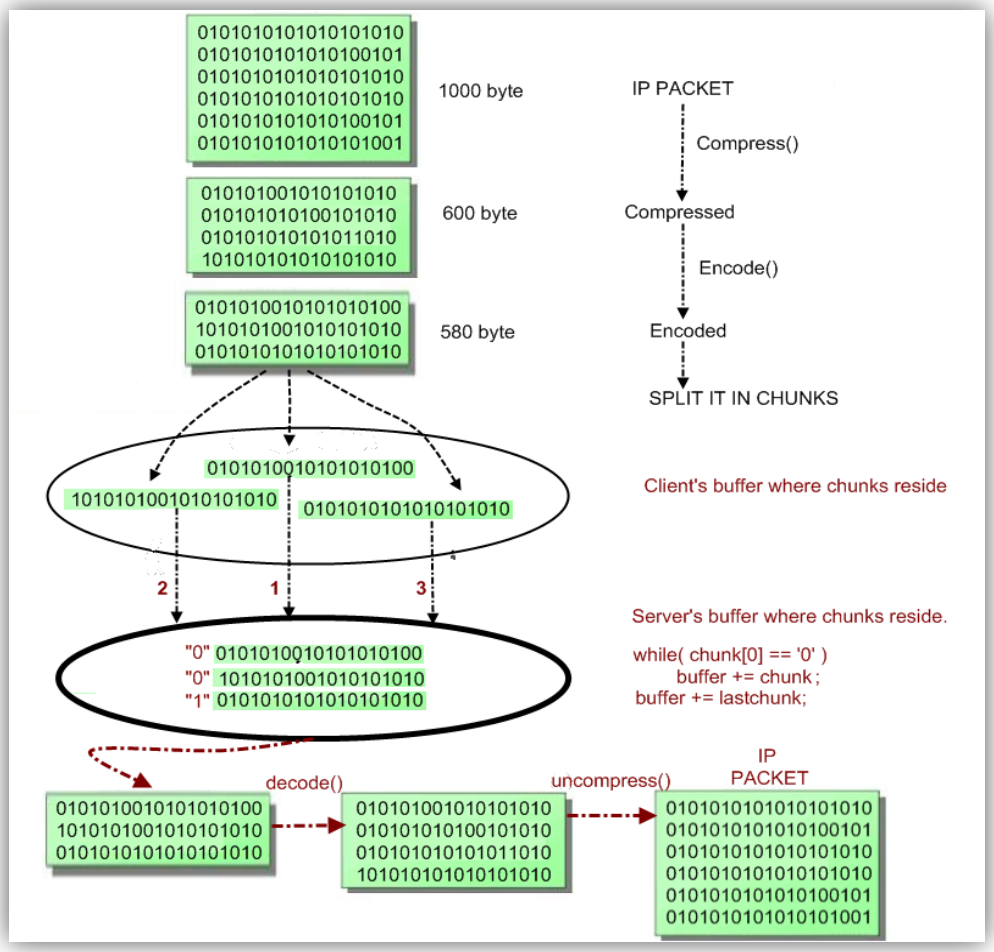
0 qeyuba4iswaqtcazhempj7rtgmoggyaaabh6kjhgfdsadfghjkjhgfdcvbnmhgfdertyuikjh
gfd78fhjmbvcdfgjhgdhjhdfhflilioiirtwersccbgqtcazhyeqxw3jvanbqeyubhambygkyeq

Εικόνα 37. Δεύτερο chunk

1 xw3jvanbqeykyeqxw3jvanbqeyuba4i

Εικόνα 38. Τελευταίο chunk

Στέλνοντας αυτήν την επιπλέον πληροφορία ναι μεν μας χρεώνει ένα byte από τα 255 του κάθε chunk αλλά έτσι μπορούμε κατά κάποιο τρόπο να τα σημαδέψουμε. Ο Iodine server καθώς αρχίζει να λαμβάνει τα DNS Query Messages εξάγει το QNAME από αυτά και διαβάζει τον πρώτο χαρακτήρα. Αν αυτός είναι ο “0” τότε εξάγει το υπολειπόμενο QNAME και το αποθηκεύει σε ένα τοπικό buffer μέσα στο Iodine server. Όσο λαμβάνει Qnames με “0” τα προσαρτεί στο buffer αυτό. Όταν λάβει QNAME με πρώτο χαρακτήρα τον “1” το προσαρτεί και αυτό στο buffer αλλά στην συνέχεια στέλνει όλο το buffer για αποκωδικοποίηση και αποσυμπίεση και πλέον έχει αποκτήσει ολοκληρωμένο το πακέτο IP που του έστειλε ο client. Η παρακάτω εικόνα παρουσιάζει σχηματικά αυτόν τον τρόπο. Κώδικας στο παράρτημα στο Code snippet 6.



Εικόνα 39. Σωστή λήψη και αποκωδικοποίηση

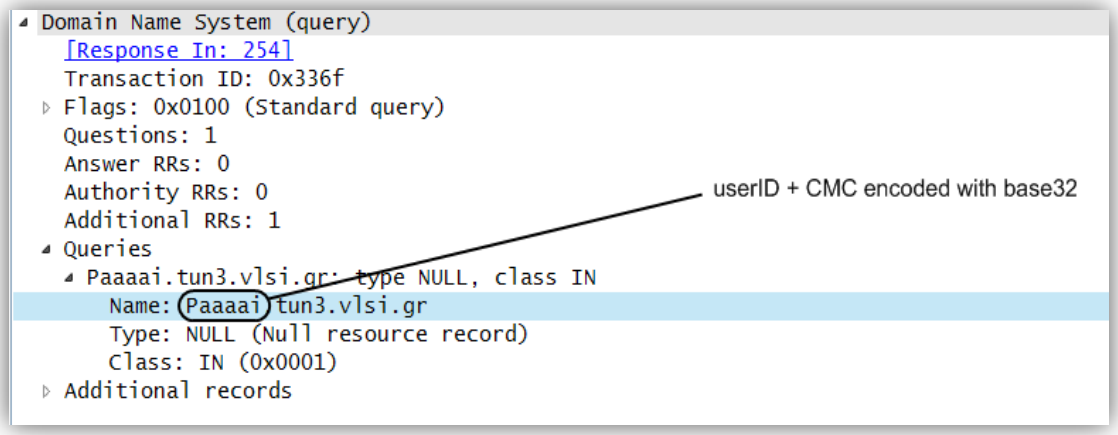
4.5. Polling

Στην αρχή αυτού το κεφαλαίου είπαμε ότι στο DNS επιτρέπεται μόνο στον client να μιλήσει πρώτο και ο server είναι αναγκασμένος να αποκριθεί όσες φορές του μιλήσει ο client. Ωστόσο, αυτό για εμάς αποτελεί ένα ακόμη εμπόδιο. Για να το καταλάβουμε αμέσως ας δούμε το ακόλουθο παράδειγμα.

Είμαστε ο πελάτης και βρισκόμαστε στο επίπεδο εφαρμογής. Κάνουμε μία αίτηση ssh σε κάποιον ssh server. Το επίπεδο εφαρμογής μιλάει με τα κατώτερα επίπεδα δημιουργείται ένα TCP segment το οποίο ενθυλακώνεται σε ένα IP packet το οποίο ενθυλακώνεται σε ένα DNS Query Message. Ο ssh server θα απαντήσει με ένα ssh response. Αυτό, αν δεν μπορέσει να ενθυλακωθεί σε ένα DNS Response Message θα χρειαστεί να σταλθούν περισσότερα. Όμως, επειδή μέχρι τώρα **στείλαμε ένα DNS Query Message** (με ενθυλακωμένη την αίτηση ssh) αυτός **θα αποκριθεί μόνο μια φορά**. Με ένα DNS Query Message μπορεί να μας στείλει το πρώτο από τα DNS Response Messages. Τι γίνεται με τα υπόλοιπα;

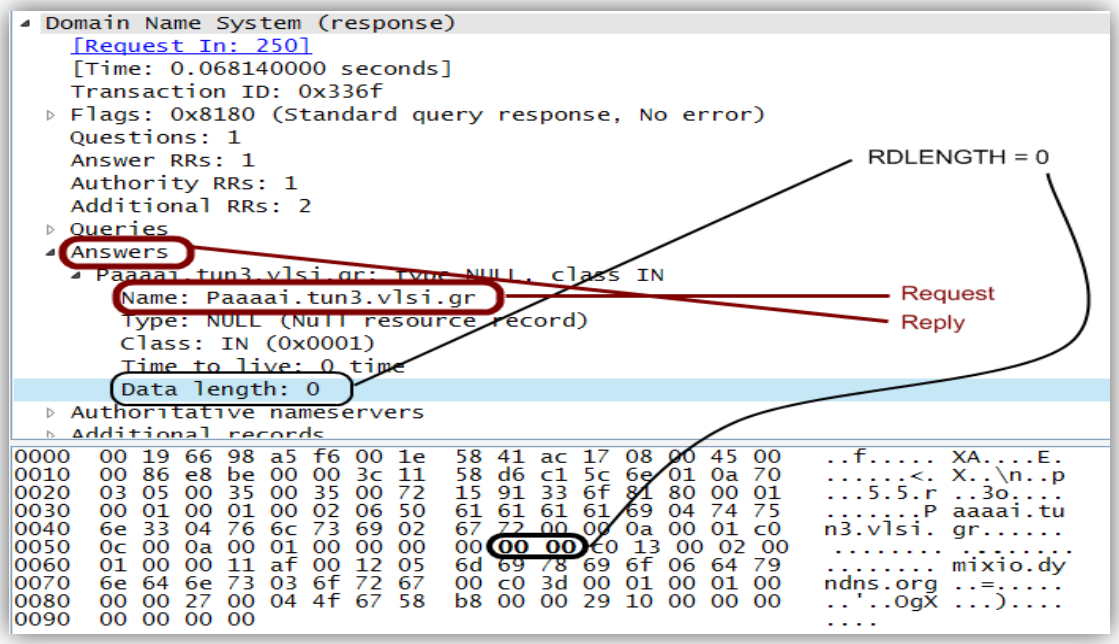
Όπως γνωρίζουμε πλέον πρέπει να στείλουμε ένα νέο DNS Query Message για να κάνουμε τον server να αποκριθεί άλλη μια φορά ούτως ώστε να μας στείλει το δεύτερο DNS Response Message. Η ερώτηση που μας έρχεται στο μυαλό είναι πόσες άλλες φορές να του στείλουμε DNS Query Messages; πόσα υπόλοιπα DNS Response Messages έχει ακόμα να μας στείλει; Την απάντηση γι' αυτό το ερώτημα δεν την γνωρίζουμε. Το καλύτερο που έχουμε να κάνουμε είναι να στέλνουμε κατά τακτά χρονικά διαστήματα ένα DNS Query Message ούτως ώστε αν έχει κάτι ο server να μας στείλει να τον κάνουμε να αποκριθεί. Αυτή είναι και η γενική φιλοσοφία του polling. Το πρωτόκολλο Iodine λύνει και αυτό το πρόβλημα με την αποστολή ενός κατάλληλα διαμορφωμένου DNS Query Message.

Κατά την αποστολή ενός τέτοιου μηνύματος ο client στέλνει τον χαρακτήρα 'p' ή 'P' προσαρτημένο από μια ακολουθία base32 χαρακτήρων οι όποιοι περιέχουν κωδικοποιημένο το user ID και ένα CMC.



Εικόνα 40. Σύλληψη ενός P packet στο Wireshark

Όταν ο server λαμβάνει τέτοια πακέτα κοιτάει στο buffer του να δει αν έχει κάτι να στείλει. Εάν έχει, το συμπιέζει και το ενθυλακώνει δίχως κάποια κωδικοποίηση σε ένα NULL TYPE DNS Response Message και το στέλνει. Όμως επειδή του στέλνουμε αυτά τα μηνύματα συνέχεια ο server θα αποκρίνεται όλη την ώρα. Έχει όμως πάντα κάτι να στείλει; Σε περίπτωση που δεν έχει, απαντάει με ένα DNS Response Message στο οποίο δεν υπάρχει τίποτα. Δηλαδή μας στέλνει μια κενή απάντηση. Μια τέτοια απάντηση φαίνεται στην παρακάτω εικόνα.



Εικόνα 41. Σύλληψη ενός κενού DNS Response Message στο Wireshark

Επίσης το polling λειτουργεί και ως timeout του client για τον server. Δηλαδή αν ο server σταματήσει να λαμβάνει P packets από έναν client για ένα λεπτό τότε τον αποσυνδέει. Κώδικας στο παράρτημα στο Code snippet 15.

4.6. BadIP

Ένα άλλο πρόβλημα που θα κληθούμε να αντιμετωπίσουμε καθώς ασχολούμαστε με το σύστημα του DNS είναι ότι ένας εξυπηρετητής ονομάτων μπορεί να έχει ρυθμιστεί να προωθεί ένα DNS Query Message σε περισσότερους από έναν εξουσιοδοτημένους εξυπηρετητές ονομάτων. Αυτό γίνεται μήπως και επιτευχθεί μια απάντηση γρηγορότερα ακολουθώντας ένα διαφορετικό μονοπάτι στο Domain Name System. Γνωρίζοντας αυτό θα δούμε ότι μερικές φορές θα καταφθάνουν στον Iodine server πολλαπλά queries από διαφορετικούς εξυπηρετητές ονομάτων και θα αναγκάζεται να απαντά πολλαπλές φορές κάτι που θα οδηγήσει σε κατάρρευση. Την λύση που δίνει το Iodine είναι να κλειδώνει στην IP του εξυπηρετητή του πρώτου query που κατέφθασε. Έτσι αν έρθει query από διαφορετική IP τότε μας επιστρέφει ένα NULL TYPE DNS Response Message με 5 byte στο πεδίο RDATA που ανακοινώνουν την σήμανση BadIP.

Κεφάλαιο πέμπτο

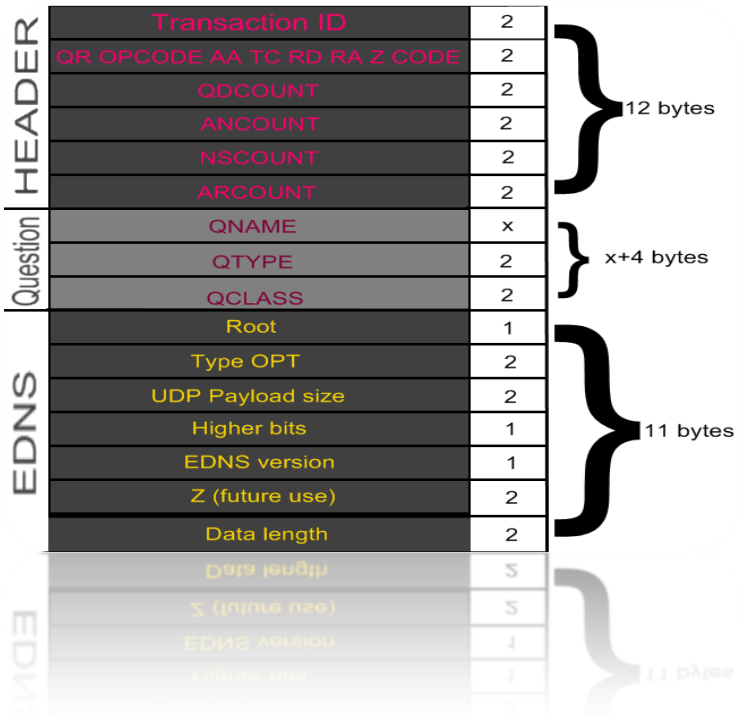
Ανάλυση βασικών κλάσεων εφαρμογής

5.1. Εισαγωγή

Ο κώδικας της εφαρμογής αποτελείται από κλάσεις-εργαλεία τα οποία γράφουν δεδομένα στην κυρίως τάξη όπου βρίσκεται η γραφική διεπαφή. Η συσχέτιση των κλάσεων είναι μια **συσχέτιση χρήσης**. Δηλαδή μία κλάση Α ζητάει πληροφορίες για κάτι από μια κλάση Β. Δεν υπάρχει εξάρτηση ούτε σταθερή επικοινωνία. Η Α δε περιλαμβάνει κάποια αναφορά στον κατάλογο των πεδίων της για την Β. Η Α απλά χρησιμοποιεί κάποια δημόσια στατική μέθοδο της Β. Έχοντας αυτά υπόψην μπορούμε να προχωρήσουμε παρακάτω και να αναλύσουμε τις πιο βασικές κλάσεις. Αυτές οι κλάσεις είναι όμοιες και για τις δύο εκδόσεις της εφαρμογής.

5.2. Η κλάση DnsPacket.cs

Η εν λόγω κλάση ευθύνεται για την παραγωγή μηνυμάτων DNS Query και για την εξαγωγή του ωφέλιμου φορτίου από τα DNS Responses. Όπως είδαμε στη παράγραφο 2.3.1 ένα DNS Query Message αποτελείται από τον Header το Question και το EDNS. Παρακάτω θα δούμε τι είναι όλα αυτά τα πεδία.



Εικόνα 42. Δομή DNS Message

Το **πεδίο ID** είναι ένας 16-μπιτος τυχαίος αναγνωριστικός αριθμός και παίρνει τιμές από 1 έως 65535. Χρησιμοποιείται από τον εξυπηρετητή ονομάτων ο οποίος το αντιγράφει στο DNS Response Message για να μπορέσει αργότερα ο resolver να αντιστοιχίσει το response με το query.

Το **πεδίο QR** είναι μια παράμετρος μήκους ενός μπιτ που διευκρινίζει εάν το μήνυμα είναι Query ή Response. Η τιμή μηδέν δηλώνει Query ενώ η τιμή ένα δηλώνει Response. Για εμάς θα είναι πάντα μηδέν καθώς δημιουργούμε μόνο Queries.

Το **πεδίο OPCODE ή OPERATION CODE** είναι ένας αριθμός μήκους 4 μπιτ και δηλώνει τον τύπο του μηνύματος. Στον παρακάτω πίνακα δίνονται οι διαθέσιμοι τύποι με τις αντίστοιχες τιμές. Εμείς κάνουμε χρήση μόνο του standard Query.

Πίνακας 4. Operation codes στο Query Message

Τιμή	Επεξήγηση
0000	standard query (QUERY)
0001	inverse query (IQUERY)
0010	server status query (STATUS)

Το **πεδίο AA ή Authoritative Answer** είναι μήκους ενός μπιτ και χρησιμοποιείται μόνο στα DNS Response Messages. Δηλώνει πως ο εξυπηρετητής ονομάτων που απάντησε στο Query message είναι ο υπεύθυνος. Οπότε για εμάς θα είναι πάντα μηδέν.

Το **πεδίο TC ή Truncation** είναι μήκους ενός μπιτ και δηλώνει ότι το μήνυμα περικόπηκε επειδή το μήκος όλου του μηνύματος είναι μεγαλύτερο από αυτό που επιτρέπει το κανάλι επικοινωνίας. Για εμάς θα είναι πάντα μηδέν.

Το **πεδίο RD ή Recursion Desired** είναι μήκους ενός μπιτ και αν ενεργοποιηθεί από τον Resolver κατευθύνει τον εξυπηρετητή ονομάτων να επιδιώξει να βρει την απάντηση με αναδρομική αναζήτηση. Για εμάς θα είναι πάντα ενεργοποιημένο.

Το **πεδίο RA ή Recursion Available** είναι μήκους ενός μπιτ και αν ενεργοποιηθεί από τον εξυπηρετητή ονομάτων σημαίνει πως υποστηρίζεται η αναδρομική αναζήτηση. Εμείς του θέτουμε μηδέν καθώς δεν αφορά τον client.

Το **πεδίο Z**. Είναι μήκους 3 μπιτ και είναι κρατημένο για μελλοντική χρήση. Πρέπει να έχει δυαδική τιμή 000 και στα Queries και στα Responses.

Το **πεδίο RCODE ή Response Code** είναι μήκους 4 μπιτ και ενεργοποιείται μόνο στα Response messages για να δηλώσει αν προέκυψε κάποιο λάθος. Στον παρακάτω πίνακα δίνονται οι διαθέσιμοι τύποι με τις αντίστοιχες τιμές.

Πίνακας 5. Response codes

Τιμή	Επεξήγηση
0000	No error condition
0001	Format error - The name server was unable to interpret the query.
0010	Server failure - Problem with the name server.
0011	Name Error - The domain name referenced in the query does not exist.
0100	Not Implemented - The name server doesn’t support this kind of query.
0101	Refused - Refused to perform the specified operation for policy reasons

Το **πεδίο QDCOUNT** είναι ένας 16-μπιτος αριθμός που προσδιορίζει τον αριθμό των ερωτήσεων που έχουμε την δυνατότητα να εισάγουμε στο τρέχων Query message. Επιπλέον το RFC ορίζει ότι μόνο μια ερώτηση είναι επιτρεπτό να εισάγουμε κάθε φορά. Έτσι το πεδίο αυτό πάντα θα παίρνει την τιμή ένα.

Το **πεδίο ANCOUNT** είναι ένας 16-μπιτος αριθμός και προσδιορίζει τον αριθμό των RRs στο Response message. Για εμάς είναι μηδέν καθώς δεν αφορά τον client.

Το **πεδίο NSCOUNT** είναι ένας 16-μπιτος αριθμός που δηλώνει τον αριθμό τον εξυπηρετητών ονομάτων που είναι υπεύθυνοι για το όνομα. Για εμάς είναι μηδέν καθώς δεν αφορά τον client.

Το **πεδίο ARCOUNT** είναι ένας 16-μπιτος αριθμός και δηλώνει τον αριθμό τον αριθμό των RRs στο DNS Query. Κανονικά θα έπρεπε να θέτουμε την τιμή μηδέν εδώ καθώς δεν στέλνουμε κανένα RR. Ωστόσο υπάρχει μια εξαίρεση. Στη παράγραφο 1.3 είδαμε ότι κάνουμε χρήση του EDNS και για να το εισάγουμε στο DNS Query Message χρησιμοποιούμε το OPT RR. Αυτομάτως το ARCOUNT πρέπει να πάρει την τιμή ένα.

Το **πεδίο QNAME ή Query Name** είναι ένα domain name που απαρτίζεται από μια ακολουθία από labels (63 byte έκαστο) και μπορεί να έχει μέγιστο μέγεθος 255 bytes. Αυτό το πεδίο αλλάζει σε κάθε νέο Query Message αφού για εμάς είναι το εκάστοτε νέο payload.

Το **πεδίο QTYPE ή Query Type** έχει μήκος 2 byte και δηλώνει τον τύπο του ερωτήματος. Εμείς, ως γνωστό πλέον χρησιμοποιούμε πάντα τον τύπο NULL που έχει τιμή 0x0A.

Το **πεδίο QCLASS ή Query Class** έχει μήκος 2 byte και δηλώνει την κλάση του ερωτήματος. Για μας έχει πάντα τιμή 0x00 που δηλώνει Internet.

Εν τέλει, βλέπουμε ότι από όλα τα πεδία μόνο το ID και το QNAME είναι αυτά που έχουν διαφορετική τιμή σε κάθε νέο DNS Query Message. Το ID είναι τυχαίο ενώ το QNAME εξαρτάται κάθε φορά από το εκάστοτε payload που στέλνουμε. Όλα τα υπόλοιπα στοιχεία είναι στατικά δεδομένα. Έτσι, με όλα τα παραπάνω μπορούμε εύκολα να φτιάξουμε έναν δομητή ο οποίος θα παίρνει ως παράμετρο το QNAME. Όλο το πακέτο θα έχει μέγεθος ίσον με το στατικό μέγεθος του Header που είναι 12 byte συν το μέγεθος του Question (που γνωρίζουμε ότι είναι x+4 λόγω του μεταβλητού QNAME) συν το στατικό μέγεθος του EDNS που είναι 11 bytes. Συνοπτικά έχουμε:

- ∅ DNSQuery = Header + Question + EDNS
- ∅ DNSQueryLength = 12 + Qname + 4 + 11

Οπότε ο δομητής περιμένει πρώτα να λάβει το QNAME και έπειτα υπολογίζει το μέγεθος του πακέτου. Στη συνέχεια παράγουμε έναν τυχαίο 16-μπιτο αριθμό. Βάζουμε κάθε byte στη θέση που πρέπει να μπει μέχρι και το EDNS. Κώδικας στο παράρτημα στο Code Snippet 1 .

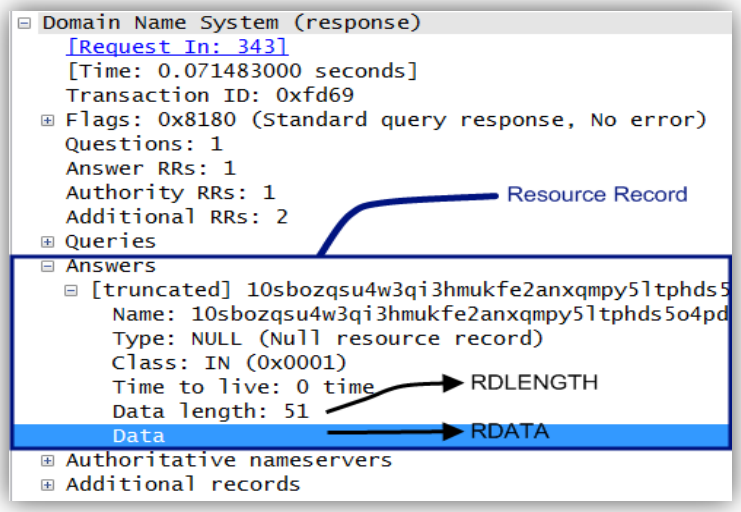
Η άλλη χρησιμότητα αυτής της κλάσης είναι να αποκωδικοποιεί ένα DNS Response και να εξάγει από το RR τα δεδομένα που βρίσκονται στο πεδίο RDATA. Στο RFC 1035 αναλύεται πώς πρέπει να γίνεται αυτό.

Όταν λαμβάνουμε ένα DNS Response πρέπει πρώτα να ελέγχουμε τον Header για να βεβαιωθούμε ότι δεν πήραμε ένα udp datagram το οποίο είναι ένα DNS Query Message όταν αυτό που περιμένουμε είναι ένα Response Message.

Αυτό γίνεται ελέγχοντας αν το τρίτο byte του Header ισούται με την τιμή ένα. Έπειτα ελέγχουμε αν το Response που λάβαμε ταιριάζει στο Query που στείλαμε. Η ταύτιση γίνεται συγκρίνοντας τα ID του Response με το ID του Query. Στο επόμενο βήμα ελέγχουμε αν το Response που πήραμε δηλώνει ότι έχει κάποιο λάθος. Αυτό μπορεί να γίνει ελέγχοντας το RDCODE στον Header το οποίο βρίσκεται στα τρία τελευταία μπιτ του τέταρτου byte και οι τιμές που μπορεί να πάρει αναλύθηκαν στον Πίνακα Response Codes παραπάνω. Για να απομονώσουμε τα τρία αυτά μπιτ χρησιμοποιούμε ANDing με την σταθερά 7.

Εικόνα 43. Ελέγχοντας το RDCODE

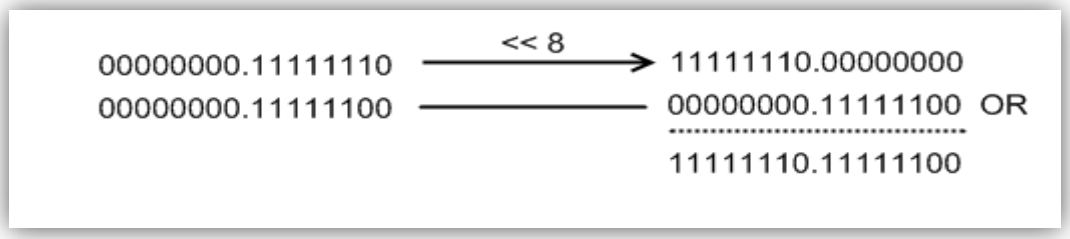
Όπως είπαμε στη παράγραφο 1.2 οι απαντήσεις όπως και άλλες πληροφορίες βρίσκονται ενθυλακωμένα στα RR. Εμείς για να εντοπίσουμε την θέση του RDATA πρέπει να διανύσουμε από την αρχή το Response.



Εικόνα 44. Η θέση του πεδίου RDATA στο DNS Response Message

Όπως φαίνεται από την εικόνα, για να φτάσουμε στο πεδίο RDATA πρέπει να προσπεράσουμε τα 11 byte του Header, τα byte του Query και τα 10 πρώτα byte του RR. Κώδικας στο παράρτημα στο Code Snippet 2 .

Το μέγεθος των δεδομένων του RDATA βρίσκεται στο πεδίο RDLENGTH σε δυο ξεχωριστά byte τα οποία πρέπει να ενώσουμε για να μας δώσουν τον αρχικό αριθμό. Για να το κάνουμε αυτό χρησιμοποιούμε left-shift-by-8 και OR. Γνωρίζοντας πλέον την τιμή του RDLENGTH και έχοντας φτάσει στην αρχή του RDATA μπορούμε να ξεκινήσουμε την εξαγωγή και την αποθήκευση των δεδομένων.



Εικόνα 45. Υπολογισμός του μεγέθους του RDATA

5.3. Η κλάση Base32.cs

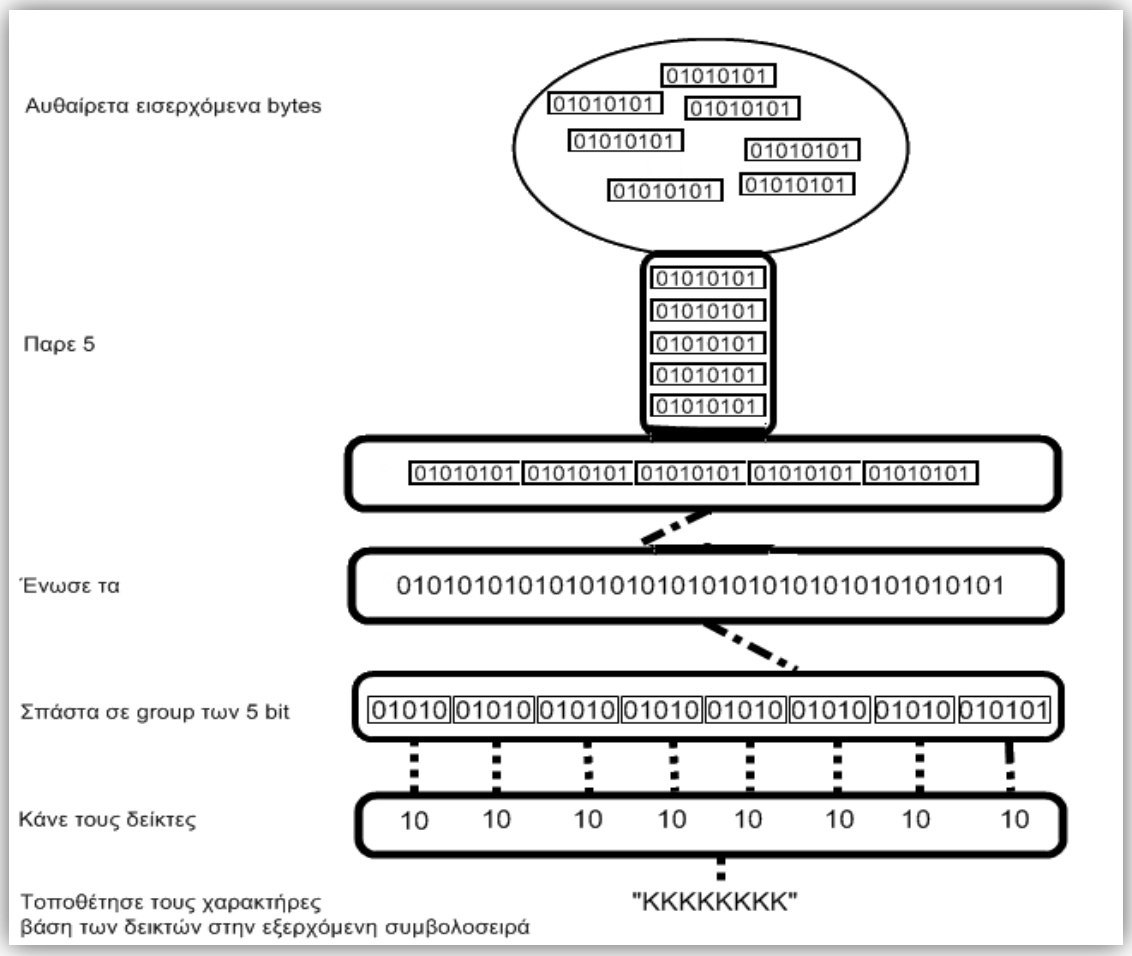
Η Base32 είναι μια κλάση-εργαλείο καθώς αυτό που μας παρέχει δεν είναι κάποιο αντικείμενο αλλά ένα ακόμη εργαλείο. Η Base32 σχεδιάστηκε να αναπαριστά αυθαίρετες ακολουθίες από byte με μια μορφή που χρειάζεται να είναι case insensitive αλλά όχι ανθρώπινα αναγνώσιμη. Χρησιμοποιείτε ένα υποσύνολο 32 χαρακτήρων της ASCII Character set, όπου επιτρέπει 5 bits να αναπαριστούν έναν από τους 32 χαρακτήρες (2^5=32) .

Πίνακας 6. Base32 Alphabet

#	Encoding	#	Encoding	#	Encoding	#	Encoding
0	A	10	K	20	U	30	4
1	B	11	L	21	V	31	5
2	C	12	M	22	W		
3	D	13	N	23	X		
4	E	14	O	24	Y		
5	F	15	P	25	Z		
6	G	16	Q	26	0		
7	H	17	R	27	1		
8	I	18	S	28	2		
9	J	19	T	29	3		

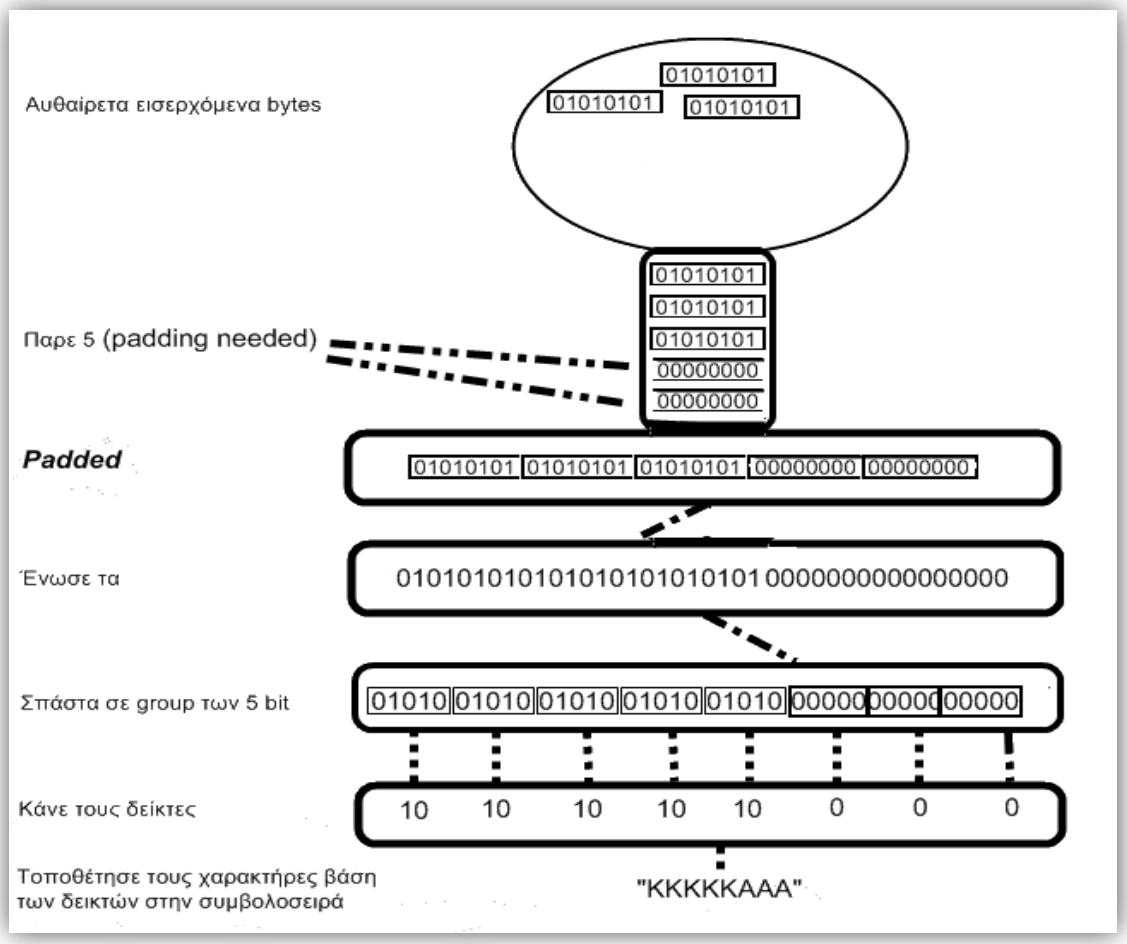
Η διαδικασία της κωδικοποίησης όπως ορίζεται στο RFC 4648 έχει ως εξής:

- *ενώνουμε πέντε byte και δημιουργείται ένα group των 40 bit*
- *σπάζουμε το group σε οχτώ μέρη των 5 bit,*
- *κάθε 5-μπιτο μέρος είναι πάντα ένας αριθμός από 0 ως 31*
- *Τα 5-μπιτα μέρη που μόλις φτιάξαμε χρησιμοποιούνται ως δείκτες που τοποθετούνται σε έναν πίνακα όπου περιέχονται οι 32 ASCII χαρακτήρες. Έτσι κάθε δείκτης θα δείξει σε κάποιον χαρακτήρα τον οποίο και τοποθετούμε στην εξερχόμενη συμβολοσειρά.*



Εικόνα 46. Η διαδικασία της κωδικοποίησης σε Base32

Όμως σε αυτή την διαδικασία υπάρχει ένα θέμα. Το πρώτο βήμα λέει ότι πρέπει να υπάρχουν στο buffer τουλάχιστον πέντε byte για να τα ενώσουμε. Τι γίνεται αν έχω στο buffer λιγότερα από πέντε byte; Εδώ υλοποιείται η γνωστή μέθοδος Padding. Το Padding συμπληρώνει με μηδενικά το απαραίτητο κενό. Αν δηλαδή έχω τρία byte στο buffer χρησιμοποιώ το Padding και τα κάνει πέντε.



Εικόνα 47. Base32 padding

Ωστόσο, αυτό μας οδηγεί σε αποστολή άχρηστων δεδομένων και κάτι τέτοιο είναι καταστροφικό σε μια εφαρμογή όπου όλα είναι περιορισμένα και ιδιαίτερα το bandwidth. Το πρόβλημα λύνεται υπολογίζοντας και απορροφώντας μόνο το πλήθος των χρήσιμων δεδομένων. Ο τύπος που υπολογίζει το πλήθος των χρήσιμων bytes είναι

$$((X * 8) / 5) + 1 \qquad (1.0)$$

όπου X το πλήθος των αρχικών byte πριν το padding. Στην περίπτωση μας με τρία byte στο buffer η formula δίνει: ((3 * 8) / 5) +1 = 5. Επομένως, θα απορροφήσουμε τους πέντε πρώτους χαρακτήρες και η συμβολοσειρά που θα σταλθεί είναι “KKKKK” . Κώδικας στο παράρτημα στο Code Snippet 3 και Code snippet 10

5.4. Η κλάση Zlib.cs

Η κλάση είναι μια κλάση-εργαλείο καθώς αυτό που μας παρέχει δεν είναι κάποιο αντικείμενο αλλά ένα εργαλείο συμπίεσης και αποσυμπίεσης δεδομένων. Εδώ γίνεται χρήση της μοναδικής έτοιμης βιβλιοθήκης για το παρών project. Η χρήση της είναι πολύ απλή.

Για να συμπιέσουμε δεδομένα χρησιμοποιούμε ένα αντικείμενο τύπου DeflaterOutputStream στο οποίο περνάμε ως παράμετρο ένα αντικείμενο τύπου MemoryStream και ένα αντικείμενο τύπου Deflater. Το MemoryStream είναι απλώς ένας χώρος στη μνήμη ενώ το Deflater είναι ο αλγόριθμος συμπίεσης. Επίσης η παράμετρος 9 που του δίνουμε είναι το μέγιστο επίπεδο συμπίεσης. Η συμπίεση ξεκινάει εκτελώντας την μέθοδο Write() του DeflaterOutputStream. Η Write() διαβάζει τα προς συμπίεση δεδομένα και αποθηκεύει τα συμπιεσμένα στη μνήμη που δέσμευσε το αντικείμενο MemoryStream.

Για να αποσυμπιέσουμε δεδομένα χρησιμοποιούμε ένα αντικείμενο τύπου InflaterInputStream στο οποίο περνάμε ως παράμετρο ένα αντικείμενο τύπου MemoryStream στο οποίο περάσαμε ως παράμετρο τα συμπιεσμένα δεδομένα. Η αποσυμπίεση ξεκινάει εκτελόντας την μέθοδο Read() του InflaterInputStream. Η Read() τα αποσυμπίεζει και τα αποθηκεύει σε ένα buffer. Κώδικας στο παράρτημα στο Code Snippet 4 .

5.5. Η κλάση Dns.cs

Είναι μια κλάση-εργαλείο καθώς αυτό που μας παρέχει δεν είναι κάποιο αντικείμενο αλλά τρία εργαλεία που θα δουλέψουν έτσι ώστε να μην παραβιάσουμε τους κανονισμούς και τις αρχές που διέπουν το σύστημα του DNS.

5.5.1. Κατασκευή του Hostname

Όταν φτάσουμε να έχουμε το κωδικοποιημένο πακέτο πριν το περάσουμε στο QNAME πρέπει να το φέρουμε σε μορφή hostname. Η κατασκευή του hostname γίνεται στην δημόσια στατική μέθοδο make_hostname() η οποία δέχεται στο μοναδικό της όρισμα μία συμβολοσειρά και παράγει ένα πίνακα από byte. Η συμβολοσειρά περιέχει το κωδικοποιημένο με base32 πακέτο IP και ο πίνακας των byte το έγκυρο hostname. Για να γίνει αυτό πρέπει να περάσει από συγκεκριμένα βήματα. Έστω, ότι έχουμε την παρακάτω κωδικοποιημένη συμβολοσειρά μεγέθους 70 bytes και θέλουμε να την στείλουμε στον server που τον δείχνει το domain tun3.vlsi.gr .

```
pdnggyhambygkyeqxw3jvanbqeyuba4lswaqtcazhem
pj7rtgmoggyaaabh6cbieqdkoyw
```

Εικόνα 48. κωδικοποιημένη συμβολοσειρά

1. Σαρώνουμε την συμβολοσειρά και εισάγουμε τον χαρακτήρα “.” μετά από κάθε εξηκοστό τρίτο χαρακτήρα. Αν η σάρωση τελειώσει και δεν έχει φτάσει στον επόμενο εξηκοστό τρίτο χαρακτήρα τότε δε βάζουμε τον χαρακτήρα “.”

```
pdnggyhambygkyeqxw3jvanbqeyuba4lswaqtcazhem
pj7rtgmoggyaaabh6cbi.eqdkoyw
```

Εικόνα 49. Διαχωρισμός labels με χρήση της τελείας

2. Έπειτα προσθέτουμε το tun3.vlsi.gr στο τέλος και προσθέτουμε άλλες δύο τελείες στην αρχή και στο τέλος του για να το φέρουμε σε μορφή FQDN. Μετά από αυτό έχουμε:

Εικόνα 50. Προσθήκη του Top Domain

3. Αντικαθιστούμε τις τελείες με τα μεγέθη των κομματιών που τις ακολουθούν. Αυτή είναι πλέον η συμβολοσειρά που τελικά θα μπει στο πεδίο QNAME.

Εικόνα 51. Αντικατάσταση τελειών με μεγέθη

5.5.2. Αποστολή κ' λήψη UDP/53 segments

Η αποστολή και λήψη των μηνυμάτων DNS γίνεται μέσω της δημόσιας στατικής μεθόδου send_query() η οποία δέχεται στο μοναδικό της όρισμα ένα πίνακα byte στον οποίο περιέχεται το εκάστοτε hostname που πρέπει να ενθυλακώσουμε στο QNAME και επιστρέφει ένα πίνακα byte στον οποίο περιέχεται το DNS Response Message. Για την επικοινωνία μας με τον έξω κόσμο χρησιμοποιούμε ένα αντικείμενο τύπου Socket. Με αυτό ανοίγουμε μια πόρτα udp/53 στον τοπικό υπολογιστή και μπορούμε να λάβουμε και να στείλουμε δεδομένα από και προς το δίκτυο που είναι συνδεδεμένος ο υπολογιστής. Συγκεκριμένα από την κλάση Socket μας παρέχονται η μέθοδοι SendTo() και ReceiveFrom() για αποστολή και λήψη και η μέθοδος Poll() για να ελέγχουμε τα timeout που ενδεχομένως θα προκύπτουν.

Για να στείλουμε ένα DNS Query Message πρώτα το φτιάχνουμε με την βοήθεια του δομητή DnsPacket που είδαμε στη παράγραφο 5.2. Αφού λάβουμε ένα DNS Response Message από τον εξυπηρετητή ονομάτων την αποθηκεύουμε σε ένα πίνακα byte. Μετά στέλνουμε τον πίνακα αυτό στην μέθοδο ExtractAnswer() όπου εκεί εξάγεται το πεδίο RDATA το οποίο περιέχει τα δεδομένα που έστειλε ο Iodined server. Αν τα δεδομένα είναι λιγότερα από 5 byte, τότε έχει προκύψει κάποιο σφάλμα. Για την αναγνώριση του σφάλματος η

ExtractAnswer() θα απαντήσει με ένα πίνακα byte ενός μόνο στοιχείου. Το στοιχείο αυτό ανάλογα με την τιμή του περιγράφει την αιτία του σφάλματος.

0. ID Mismatch: Το Message που έλαβε έχει άλλο ID από αυτό που έστειλε
1. Format error: Το Message έχει μη-συμβατή μορφή. Οφείλεται στον αποστολέα
2. Server failure: Πρόεκυψε σφάλμα στον DNS server. Δεν το γνωρίζουμε αυτό
3. Name Error: Το hostname στο query δεν μπορεί να βρεθεί. Οφείλεται στον αποστολέα. Το πιο πιθανό είναι ότι δώσαμε λάθος Top Domain.
4. Not Implemented: Ο εξυπηρετητής ονομάτων δεν υποστηρίζει τέτοιου είδους ερωτήσεις. Δηλαδή η υλοποίηση του δεν εφαρμόζει NULL ερωτήσεις.
5. Refused: Ο εξυπηρετητής ονομάτων αρνείται να εξυπηρετήσει την ερώτηση για λόγους πολιτικής.
6. Extracting Answer: Αν εμφανιστεί τέτοιο μήνυμα σημαίνει ότι προέκυψε σφάλμα κατά την εξαγωγή της απάντησης από το RDATA.
7. Not a dns response: Σημαίνει ότι ο εξυπηρετητής ονομάτων μας έστειλε κάτι εκτός από απάντηση. Δεν συμβαίνει συχνά αλλά το RFC-3696 προτείνει να γίνεται ένας τέτοιος έλεγχος.

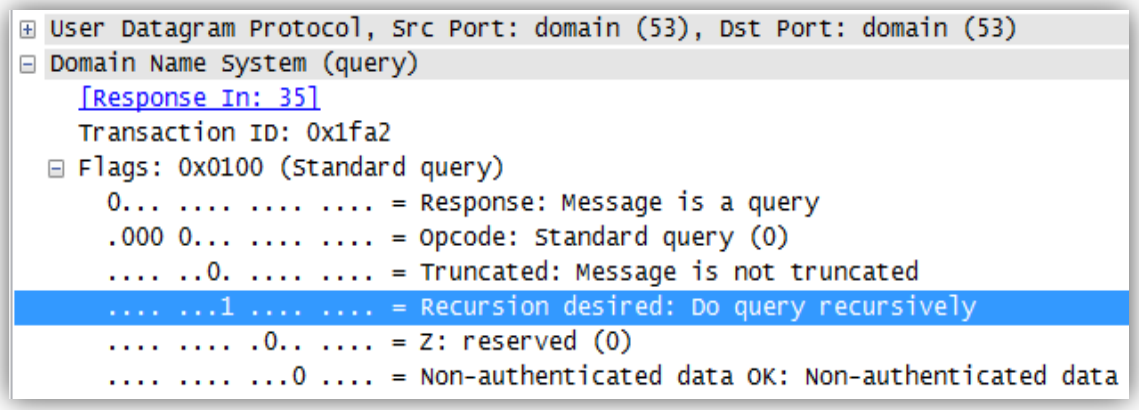
Αν τα δεδομένα είναι περισσότερα από 5 byte τότε τα στέλνουμε για αποσυμπίεση. Μετά την αποσυμπίεση προκύπτει ένα πακέτο IP που έχει ως Destination Address αυτήν της εικονικής συσκευής μας. Αυτό στάλθηκε από τον Iodined server και εμείς θα το γράψουμε στην εικονική συσκευή με τον τρόπο που θα δούμε στο κεφάλαιο 6. Κώδικας στο παράρτημα στο Code Snippet 2 .

5.5.3. Dns implementation check

Στην αρχή της εργασίας μιλήσαμε για τους τύπους των Queries και τα είδη αναζητήσεων και είδαμε τι μας συμφέρει πιο πολύ να εκμεταλλευτούμε. Ωστόσο υπάρχουν και εξυπηρετητές ονομάτων που ενδέχεται (για λόγους ασφάλειας ή πολιτικής) να έχουν απενεργοποιημένα κάποια χαρακτηριστικά τους. Δύο από αυτά που μας ενδιαφέρουν και αποτελούν τις πιο βασικές παραμέτρους για την επιτεύξη του στόχου είναι η δυνατότητα εκτέλεσης της αναδρομικής αναζήτησης και η υποστήριξη των NULL TYPE Queries από τον server. Αυτός ο έλεγχος

γίνεται από την δημόσια στατική μέθοδο dns_implementation_check() και επιστρέφει true/false αναλόγως αν ο server πληρεί τις προυποθέσεις.

Έτσι καλό θα ήταν πρωτού ξεκινήσει η εκτέλεση των μεθόδων του Iodine protocol να ελέγξουμε αν η υλοποίηση του εξυπηρετητή ονομάτων πληρεί τις εν λόγω προυποθέσεις. Για να το κάνουμε αυτό μπορούμε απλώς να στείλουμε ένα DNS Query Message του οποίου ο τύπος είναι NULL και στο οποίο θα ενεργοποιήσουμε την αναδρομική αναζήτηση. Ένα τέτοιο πακέτο φαίνεται παρακάτω.



Εικόνα 52. Dns implementation check

Αν ο εξυπηρετητής ονομάτων υποστηρίζει τον τύπο NULL τότε θα μας στείλει ένα DNS Response Message τύπου NULL στου οποίου το Header το πεδίο RCODE έχει την δυαδική τιμή 0000. Αν δεν το υποστηρίζει μας απαντάει με ένα DNS Response Message χωρίς σώμα. Στον Header του μηνύματος αυτού το πεδίο RCODE έχει την δυαδική τιμή 0001 και σημαίνει ότι ο εξυπηρετητής ονομάτων δε μπόρεσε να ερμηνεύσει το μήνυμα.

Πίνακας 7. RCODES

0000	No error condition
0001	Format error - The name server was unable to interpret the query.

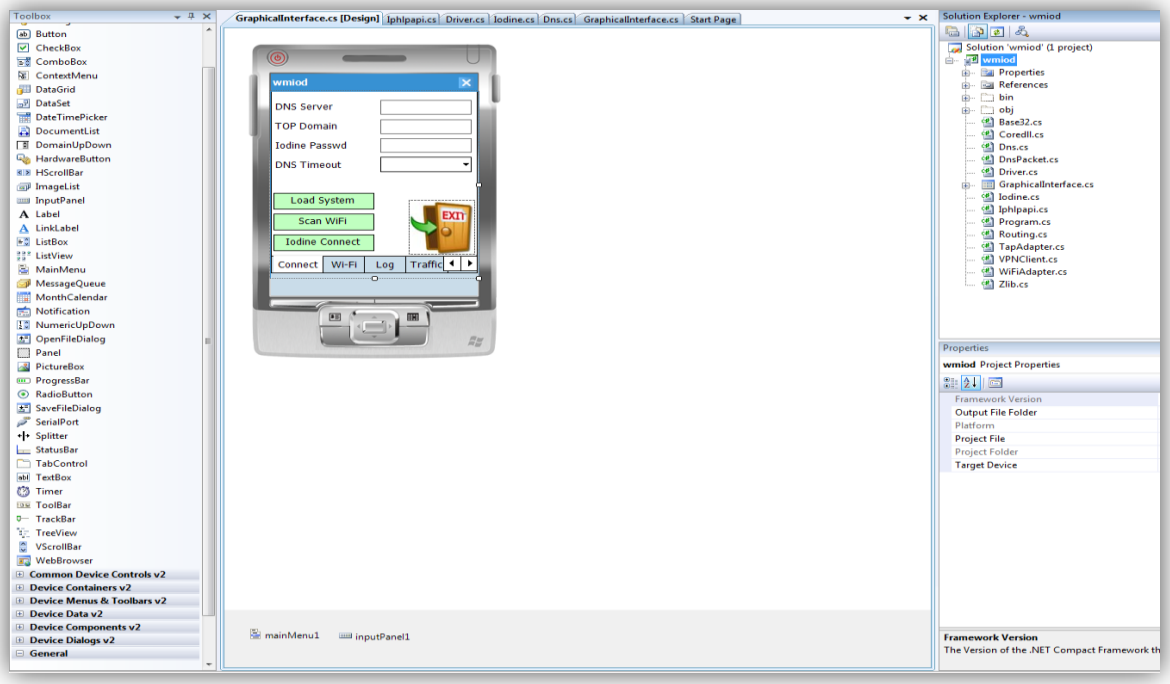
Επίσης, αν ο εξυπηρετητής ονομάτων υλοποιεί αναδρομικές αναζητήσεις τότε στον header του προηγούμενου μηνύματος το πεδίο RA είναι ενεργοποιημένο. Αν δεν είναι τότε δε την υλοποιεί. Έστω και ένα από τα δύο

χαρακτηριστικά να μην υλοποιούνται τότε πρέπει να διακόψουμε όλη την περαιτέρω διαδικασία.

Ο έλεγχος αυτός δεν αποτελεί κομμάτι του Iodine protocol. Το πρόσθεσα ο ίδιος διότι συνάντησα τέτοιες περιπτώσεις όπου δεν γνώριζα από πού προερχόταν το λάθος και δεν μπορούσαν να δρομολογηθούν τα DNS Query Messages από τον εξυπηρετητή ονομάτων κάποιου εγχώριου ISP. Έτσι, αν συμβεί μια όμοια περίπτωση η εφαρμογή μπορεί πλέον να αναγνωρίσει το σφάλμα και να ειδοποιήσει τον χρήστη. Πάντως, η εμπειρία έδειξε ότι ένα τέτοιο φαινόμενο συναντιέται σπανίως.

5.6. Η κλάση GraphicalInterface.cs

Η εν λόγω κλάση είναι υπεύθυνη για την δημιουργία της γραφικής διασύνδεσης. Όλα τα αντικείμενα που παράγουν κάτι γραφικό ονομάζονται controls. Η δημιουργία του γραφικού περιβάλλοντος είναι μια αυτοματοποιημένη διαδικασία και γίνεται πολύ εύκολα από το Visual Studio. Γι' αυτό τον λόγο δε θα δώσουμε καθόλου βάρος στην παραγωγή της γραφικής διασύνδεσης. Παρακάτω θα δούμε μια εικόνα για να δείξουμε πόσο εύκολη και ευχάριστη είναι αυτή η διαδικασία.



Εικόνα 53. Visual Studio Designer

Η κλάση **GraphicalInterface** πρέπει να είναι μοναδική και πρέπει να δημιουργείται μόνο ένα στιγμιότυπο της. Έτσι κρίνεται απαραίτητη η χρήση ενός σχεδιαστικού προτύπου (**design pattern**) και αυτό είναι το **μοναδιαίο**. Αυτό μας προσφέρει μια ειδική μέθοδο κατασκευής στιγμιοτύπων η οποία ελέγχει αν κάποιο αντικείμενο έχει ηδη δημιουργηθεί. Αν ναι, επιστρέφει απλώς έναν δείκτη προς το υπάρχον αντικείμενο. Αν όχι, τότε δημιουργεί ένα νέο αντικείμενο και επιστρέφει έναν δείκτη σε αυτό. Έτσι η κυρίως κλάση περιέχει τον παρακάτω ειδικό δομητή. Όποτε θέλουμε να επεξεργαστούμε μια μέθοδο ή control της κυρίως κλάσης από μια άλλη τότε απλώς ζητάμε το τρέχων στιγμιότυπο την κυρίως κλάσης.

```
public static GraphicalInterface getInstance() {
        if (this.instance == null)
                this.instance = new GraphicalInterface();
        return this.instance;
}
```

```
GraphicalInterface.getInstance().method_do_sth()
```

Επίσης η κλάση **GraphicalInterface** τρέχει στο κυρίως thread της διεργασίας. Είναι γενικός κανόνας ένα thread να μην έχει πρόσβαση στα στοιχεία ενός άλλου thread. Έτσι αν δημιουργηθεί ένα νέο thread αυτό δε θα έχει πρόσβαση στο thread της **GraphicalInterface.** Στην εφαρμογή μας όμως έχουμε νέα thread τα οποία παράγουν αποτελέσματα που πρέπει να περνάνε στο κυρίως thread ώστε να απεικονίζονται στην γραφική διασύνδεση. Αυτό το πρόβλημα λύνεται από ένα HowTo του MSDN που έχει τίτλο *Make Thread-Safe Calls to Windows Forms Controls*. Συνοπτικά μας περιγράφει να ακολουθήσουμε τα παρακάτω απλά βήματα:

1. Να εξεταστεί η ιδιότητα InvokeRequired του control που θέλουμε.
2. Αν η InvokeRequired επιστρέψει true, τότε να καλέσουμε την μέθοδο Invoke με μια delegate η οποία καλεί μία τοπική μέθοδο στην κυρίως κλάση και η οποία μέθοδος κάνει αυτό που επιθυμούμε στο control.

3. Αν η InvokeRequired επιστρέψει false, τότε να καλέσουμε άμεσα το control που θέλουμε.

Να αναφερθεί ότι η ιδιότητα InvokeRequired ενός αντικειμένου ανιχνεύει εάν το καλούμε από ένα άλλο thread από αυτό που δημιουργήθηκε. Για παράδειγμα αν θέλουμε να γράφουμε δεδομένα σε ένα control που κάνει logging φτιάχνουμε μια delegate.

```
delegate void SetLogCallback(string text);
```

και μια μέθοδο που κάνει append ένα string στο logging control.

```
private void SetLog(string text) {
        this.LogTextBox.Text += text;
}
```

Μετά φτιάχνουμε την μέθοδο που θα καλέσουμε από το άλλο thread η οποία εξετάζει αν το control χρειάζεται πρόσβαση από άλλο thread.

```
public void MsgLogger(string s) {
        if (this.LogTextBox.InvokeRequired) {
                SetLogCallback LGCB = new SetLogCallback(SetLog);
                this.Invoke(LGCB, new object[] {s});
         }
        else this.LogTextBox.Text += s;
}
```

Τέλος, καλούμε από το άλλο thread την μέθοδο MsgLogger() ως εξής:

```
GraphicalInterface.getInstance().MsgLogger("log sth");
```

Ακριβώς με τον ίδιο τρόπο κάνουμε αυτή την διαδικασία για κάθε ένα control που θα χρειαστεί να το ‘πειράξουμε’ από άλλο thread εκτός του κυρίως.

5.7. Η κλάση TapAdapter.cs

Σκοπός αυτής της κλάσης είναι να μας παρέχει εκείνα τα εργαλεία με τα οποία θα δουλέψουμε τον TAP adapter. Ασχολείται με το άνοιγμα και κλείσιμο της συσκευής, με το διάβασμα και γράψιμο της και τέλος με τις δικτυακές ρυθμίσεις της. Παρακάτω θα δούμε μερικές από τις πιο βασικές μεθόδους τις.

Μέθοδος CreateTap(). Καλεί την native συνάρτηση **CreateFile()** του coredll.dll που επιστρέφει μια αναφορά στην εφαρμογή με την οποία αργότερα θα ανοίξει την συσκευή για I/O. Αν το άνοιγμα επιτύχει τότε θα κληθεί η native συνάρτηση GetAdapterIndex() του iphlpapi.dll για να πάρουμε το interface index number της συσκευής. Αυτό θα μας χρειαστεί στη συνέχεια που θα χρειαστεί να πειράξουμε το routing table που εξηγούμε στην επομένη παράγραφο.

Μέθοδος CloseTap(). Καλεί την native συνάρτηση **CloseHandle()** η οποία αποσυνδέει την συσκευή από την εφαρμογή και την ελευθερώνει στο σύστημα.

Μέθοδος CreateTimer(). Δημιουργεί έναν Timer ο οποίος εκτελεί σε πολύ μικρά χρονικά διαστήματα της τάξης millisecond την μέθοδο ReadTun() που θα δούμε αμέσως παρακάτω.

Μέθοδος ReadTun(). Είναι αυτή που θα διαβάσει σε synchronous mode ότι δεδομένα έχουν γραφεί στη συσκευή από την TCP/IP stack του λειτουργικού συστήματος. Καλεί την native συνάρτηση **ReadFile()** του coredll.dll. Εφόσον τα δεδομένα είναι πάνω από 20 byte δηλαδή τουλάχιστον ένα IP Header τότε τα στέλνει για συμπίεση και έπειτα λαμβάνει χώρα η διαδικασία που περιγράψαμε στην παράγραφο 4.4. Επίσης ενημερώνει ένα control του κυρίως thread με τον όγκο των δεδομένων που διάβασε.

Μέθοδος WriteTun(). Θα γράψει σε synchronous mode ότι δεδομένα έχουν μέγεθος άνω τον 20 byte καλώντας την native συνάρτηση **WriteFile()** του coredll.dll. Επίσης ενημερώνει ένα control του κυρίως thread με τον όγκο των δεδομένων που έγραψε.

Μέθοδος set_layer3_mode(). Καλεί την native συνάρτηση **DeviceIoControl()** του coredll.dll. Με τον ειδικό κώδικα ελέγχου [TAP_IOCTL_CONFIG_TUN] θέτει την συσκευή σε layer-3 operation που σημαίνει ότι η συσκευή θα εκτελεί χρέη routing και θα έχει τα χαρακτηριστικά μιας τυπικής

δικτυακής συσκευής. Επίσης ενεργοποιεί το Address Resolution Protocol στην συσκευή. Ο κώδικας ελέγχου συνοδεύεται από 12 byte τα οποία αντιπροσωπεύουν την διεύθυνση IP του interface, την μάσκα υπό-δικτύου του και το δίκτυο στο οποίο ανήκει. Κώδικας στο παράρτημα στο Code Snippet 12 .

Μέθοδος set_dhcp_masq(). Στην πλατφόρμα των windows mobile δεν υπάρχει καμία απολύτως προγραμματιστική δυνατότητα να απενεργοποιήσεις το DHCP discovery. Έτσι κάθε interface που ενεργοποιείται ψάχνει κάποιον DHCP server για να πάρει μια διεύθυνση IP. Αν δεν βρεθεί server μέσα σε λίγα λεπτά τα windows του δίνουν αυτόματα μια προκαθορισμένη διεύθυνση. Εμείς έχοντας θέσει προγραμματιστικά διεύθυνση IP στο inteface καλώντας την **AddIPAddress()** που θα δούμε αργότερα αυτή μετά από λίγο χάνεται γιατί θα αντικατασταθεί από την προκαθορισμένη των windows. Αυτό, όπως είναι φανερό θα χαλάσει το routing και θα καταρρεύσει όλη η επικοινωνία. Η μοναδική λύση σε αυτό το πρόβλημα δίνεται από τον κώδικα ελέγχου [TAP_IOCTL_CONFIG_DHCP_MASQ] Η λειτουργία του είναι η εξής. Παρεμβάλλεται μεταξύ του interface και των DHCP πακέτων που φθάνουν σε αυτό. Αλλάζει τα πεδία με τις προκαθορισμένες διεύθυνσεις των windows με τις διευθύνσεις που εμείς επιθυμούμε. Ο κώδικας ελέγχου συνοδεύεται από 16 byte δεδομένων που περιέχουν την επιθυμητή διεύθυνση IP, την επιθυμητή μάσκα υποδικτύου, ξανά την επιθυμητή διεύθυνση IP και τέλος το lease time. Επομένως, καλούμε την DeviceIoControl() native συνάρτηση του coredll.dll. Με τον ειδικό κώδικα ελέγχου και τα συνοδευόμενα byte ενεργοποιούμε αυτήν την λειτουργία. Έτσι κρατάμε πάντα την διεύθυνση μας. Κώδικας στο παράρτημα στο Code Snippet 16 .

Μέθοδος set_media_status(). Καλεί την native συνάρτηση DeviceIoControl() του coredll.dll. Χρησιμοποιώντας τον ειδικό κώδικα ελέγχου [TAP_IOCTL_SET_MEDIA_STATUS] ενεργοποιεί ή απενεργοποιεί την συσκευή. Συνοδεύεται από έναν ακέραιο αριθμό που παίρνει τιμές 0 ή 1 αναλόγως. Κώδικας στο παράρτημα στο Code Snippet 14 .

Μέθοδος flush_arp(). Θα καλέσει την native συνάρτηση **FlushIpNetTable()** του iphlpapi.dll η οποία θα αναλάβει να καθαρίσει την ARP CACHE της συσκευής από τίποτα παλαιότερες καταχωρήσεις που ενδεχωμένως δεν σβήστηκαν ακόμα.

Εδώ πρέπει να σημειώσουμε μια ασυμβατότητα που υπάρχει μεταξύ του TAP driver των Windows και του TAP driver για το Linux όπου τρέχει ο iodine server. Το tap device του Linux μπροστά από το πακέτο IP βάζει και ένα header μήκους τεσσάρων byte. Αυτά συνήθως έχουν τις τιμές [0x0 0x0 0x8 0x0] όταν το tap device προέρχεται από Linux. Έτσι όταν σχηματίσουμε το IP packet από τα λαμβανόμενα DNS Response Messages πρέπει να αφαιρέσουμε αυτόν τον header πριν γράψουμε το πακέτο στο TAP device των Windows. Επίσης, το αντίστροφο γίνεται και όταν στέλνουμε ένα πακέτο IP από τα windows στο Linux. Αφού διαβάσουμε ένα πακέτο από το tap device στα windows πρέπει να του προσθέσουμε το εν λόγω header μπροστά από το πακέτο ούτως ώστε όταν φτάσει στο linux να μπορέσει να αποκωδικοποιηθεί καταλλήλως.

5.8. Η κλάση Routing.cs

Η εφαρμογή εκτός του να παρέχει την τεχνική IP-over-DNS σε mobile συσκευές εκτελεί και χρέη routing. Έτσι η εφαρμογή εδραιώνει την επικοινωνία και με το private δίκτυο με τον iodine server και την επικοινωνία μέσω του DNS τούνελ. Η παρούσα κλάση υπάρχει για να κάνει αυτή την μετατροπή.

Αυτό που πρέπει να κάνει η εφαρμογή είναι να ανακατευθύνει όλη την κίνηση προς το διαδίκτυο στο εικονικό interface έτσι ώστε να περνάνε όλα τα δεδομένα μέσα από το DNS τούνελ. Όμως πρέπει να υπάρχει και ένα στατικό route προς τον εξυπηρετητή ονομάτων αφού ξέρουμε πάντα ότι πρέπει να έχουμε δικτυακή επικοινωνία με αυτόν και μόνο. Ένα route στα Windows περιγράφεται ως μια δομή με όνομα **MIB_IPFORWARDROW** και περιγράφεται στην παράγραφο 6.2.8. Η μέθοδος **redirect_default_gateway()** είναι υπεύθυνη για την εισαγωγή των κατάλληλων route. Τα route αυτά είναι συγκεκριμένα στατικά και είναι τρία.

Πίνακας 8. Δρομολόγια για ανακατεύθυνση του default gateway

Destination	Mask	NextHop	Interface
0x00000000	0x00000080	Remote_endpoint	TAP
0x00000080	0x00000080	Remote_endpoint	TAP
Name server	0x11111111	Name server	LocalNet

Τα δύο πρώτα καλύπτουν το internet ως προορισμό, ενώ το τελευταίο είναι για την άμεση επικοινωνία με τον εξυπηρετητή ονομάτων. Όλα τα route είναι INDIRECT δηλαδή ο NextHop δεν είναι τοπικός. Για την εισαγωγή τους στο route table καλούμε με unmanaged code την συνάρτηση **CreateIpForwardEntry()** του iphlpapi.dll.

Επίσης η κλάση αυτή περιέχει και την μέθοδο με την οποία θέτουμε διεύθυνση IP και μάσκα υπό-δικτύου στην εικονική συσκευή. Η μέθοδος αυτή καλεί με unmanaged code την συνάρτηση AddIPAddress() του iphlpapi.dll. Τέλος, πρέπει να προσέξουμε πως όταν πρόκειται να κλείσουμε την εφαρμογή πρέπει όλα αυτά τα route που προσθέσαμε να τα διαγράψουμε. Αυτό γίνεται στην μέθοδο restore_routing() η οποία καλεί με unmanaged code την συνάρτηση **DeleteIpForwardEntry()** του iphlpapi.dll για όλα τα route που είχαμε είσαγει.

Κεφάλαιο έκτο

Ανάπτυξη εφαρμογής για φορητές συσκευές

6.1. Τεχνικά χαρακτηριστικά της C#

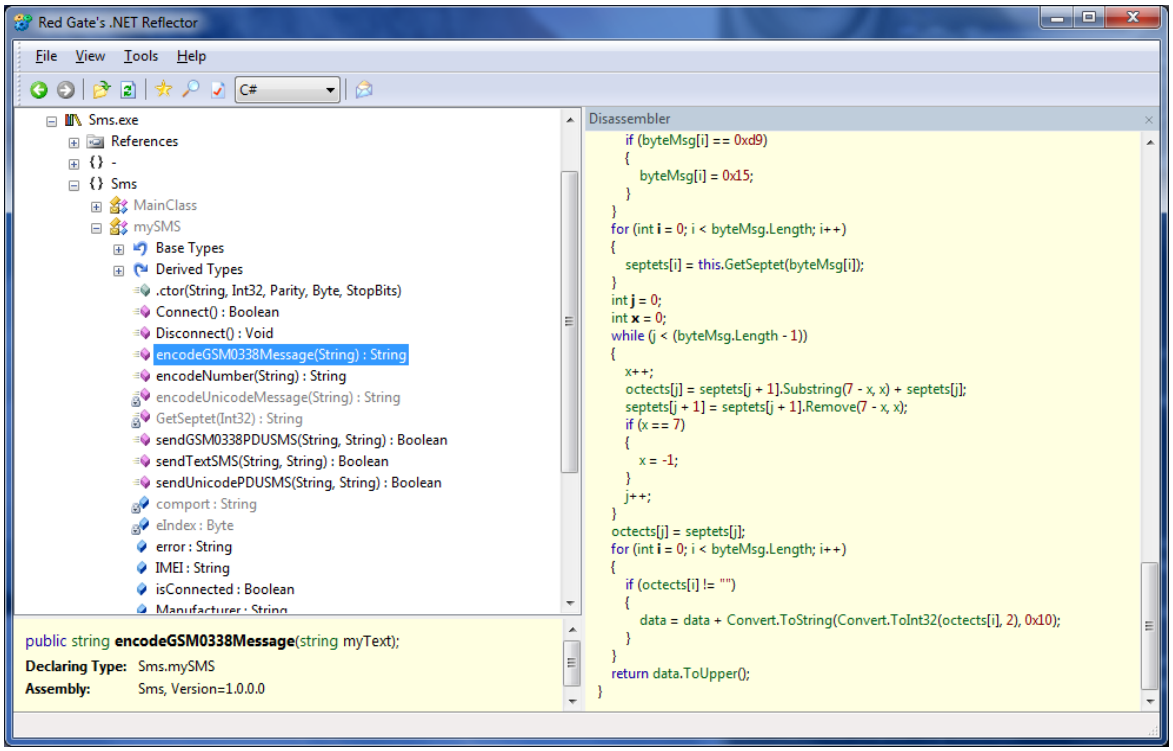
Η C# αναπτύχθηκε από την Microsoft και αρχικά της είχανε δώσει το όνομα COOL από το "C - like Object Oriented Language". Τον Ιούλιο του 2007 που την δημοσίευσαν στον κοινό της έδωσαν το όνομα C Sharp ή C#. Για να δούμε με απλό τρόπο πως δουλεύει αρκεί να την συγκρίνουμε με την JAVA την οποία και διδαχθήκαμε εκτενώς από την σχολή. Ακολουθεί ο πίνακας σύγκρισης.

Πίνακας 9. Πίνακας σύγκρισης Java, C#

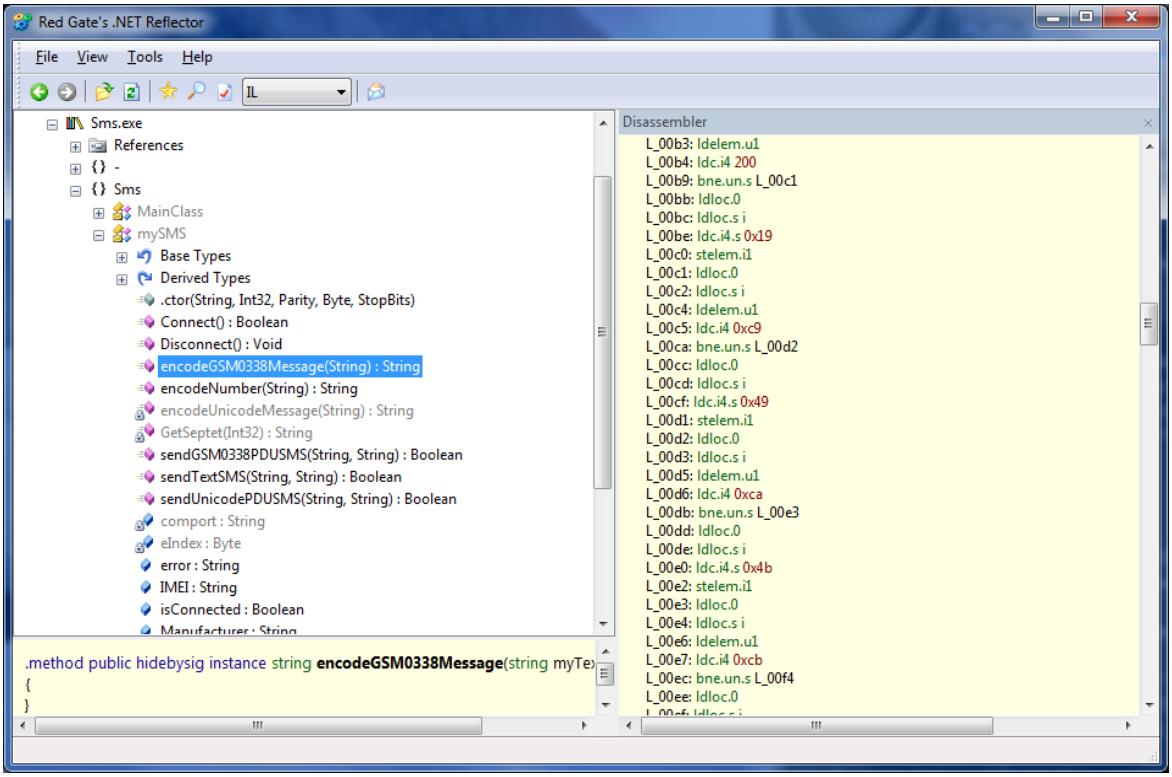
	JAVA	**C#**
Prog Lang	sources	sources
↓	javac.exe	csc.exe
IL	**Byte Code**	**Managed Code**
↓	Compiled by JVM	Compiled by JIT
Binary	**Machine Code**	**Machine Code**
↓	Executed & managed by JRE	Executed & managed by CLR

- JVM για Java Virtual Machine
- JRE για Java Runtime Environment
- JIT για Just-in-Time Compiler
- CLR για Common Language Runtime
- Το Byte Code και το Managed Code είναι IL ή Intermediate Language.

Ο πηγαίος κώδικας μετατρέπεται σε managed code από τον csc.exe compiler. Ο managed code γραμμένος σε γλώσσα **IL** (**Intermediate Language**) θα είναι το εκτελέσιμο αρχείο το οποίο αλλιώς λέγεται **assembly**. Όταν τρέξουμε το assembly, αυτό θα μεταγλωττιστεί επί τόπου από τον JIT και θα παραχθεί ο κώδικας μηχανής ο οποίος θα τρέξει υπό την επίβλεψη του CLR. Αυτή η επί τόπου μεταγλώττιση είναι που κάνει αυτού του τύπου τα προγράμματα να φορτώνονται φανερώς καθυστερημένα.

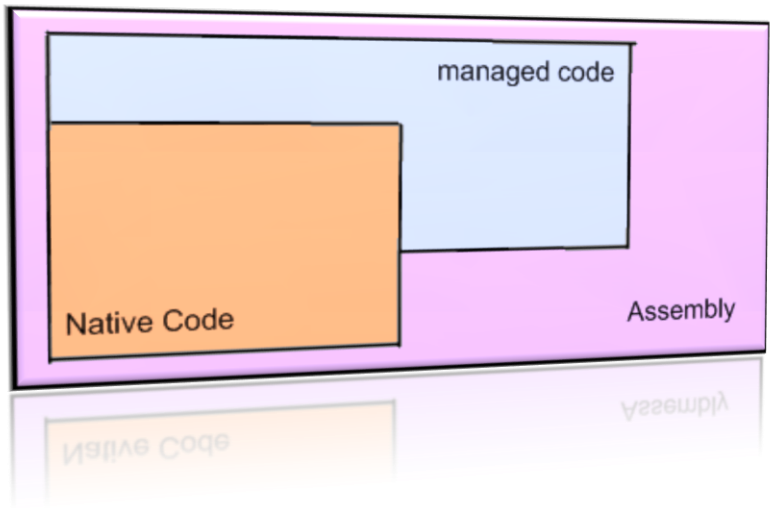


Εικόνα 24. Γλώσσα C#



Εικόνα 25. Intermediate Language

Η C# μας δίνει την δυνατότητα μέσα στο assembly να έχουμε εκτός από τον managed code και unmanaged code ή όπως ονομάζεται Native code.



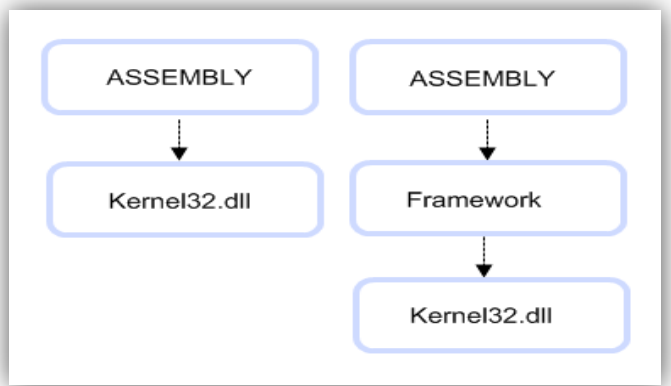
Εικόνα 56. Συγκέντρωση managed και unmanaged code στο ίδιο assembly

Ο Native Code δεν θα εκτελεστεί υπό την επίβλεψη του CLR αλλά κατευθείαν από την μηχανή. Προφανώς, γίνεται κατανοητό ότι ο Native Code θα εκτελεστεί πολύ ταχύτερα από τον Managed Code. Αυτή την χρήσιμη δυνατότητα μας τη προσφέρει η C# μέσω της υπηρεσίας **PInvoke** ή αλλιώς **Platform Invocation service**. Πιο συγκεκριμένα η PInvoke μας δίνει την δυνατότητα να καλέσουμε unmanaged code από τον managed code στο ίδιο assembly.

Γιατί όμως να θέλουμε κάτι τέτοιο; Μερικές φορές το Framework που μας προσφέρει έτοιμες κλάσεις και αυτοματοποιημένες λειτουργίες δεν είναι αρκετά.
Το Framework είναι στην ουσία ένας wrapper γύρω από όλες τις συναρτήσεις που μας προσφέρουν οι βιβλιοθήκες των Windows. Δηλαδή μας προσφέρει μια διευκόλυνση. Ωστόσο, το framework δεν περιλαμβάνει εξ' ολοκλήρου τις δυνατότητες όλων των βιβλιοθηκών του συστήματος. Έτσι, κάθε φορά που θέλουμε να χρησιμοποιήσουμε μία εξειδικευμένη μέθοδο από ένα DLL αρχείο πρέπει να κάνουμε μια δήλωση του εν λόγω αρχείου στον κώδικα. Στην C# αυτό γίνεται με την μάκρο-εντολή **DllImport**. Κάτω από αυτήν δηλώνεται και η υπογραφή της μεθόδου που θα χρησιμοποιήσουμε. Ως παράδειγμα δίνεται το παρακάτω απόσπασμα κώδικα.

```
[System.Runtime.InteropServices.DllImport("kernel32.dll")]
IntPtr  CreateFile
(
String filename, uint fileaccess, IntPtr fileshare, IntPtr securityattributes, uint creationdisposition, uint flags, IntPtr template
);
```

Το παραπάνω μας περιγράφει ότι θα καλέσουμε την μέθοδο CreateFile() με την συγκεκριμένη υπογραφή από το αρχείο-βιβλιοθήκη kernel32.dll. Η μέθοδος αυτή υπάρχει εκεί ως Machine Code και θα κληθεί από το assembly μας.



Εικόνα 57. Απευθείας κλήσεις σε Unmanaged Code

Στην παρούσα εφαρμογή πολλές λειτουργίες που χρειάστηκαν δεν υποστηρίζονται από το .NET Framework. Όλες οι συναρτήσεις που καλούμε απευθείας από τα DLL κάνουν χρήση στοιχείων της C++ όπως pointers, structs και references. Αυτό έκανε την υλοποίηση του project αρκετά δύσκολο και χρονοβόρο. Εδώ να πούμε ότι η C# επιτρέπει την χρήση στοιχείων της C++. Στην επόμενη παράγραφο θα περιγράψουμε εκτενώς τις λειτουργίες που χρειαστήκαν και δεν τις παρείχαν τα framework.

6.2. Βιβλιοθήκες συστήματος

Τα Windows παρέχουν ένα σύνολο συναρτήσεων σε native code οι οποίες βρίσκονται στα αρχεία coredll.dll και iphlpapi.dll στο κατάλογο **/Windows/** για τα Windows Mobile. Από αυτές χρειαζόμαστε τις CreateFile, CloseHandle,

DeviceIoControl, ReadFile και WriteFile από το αρχείο coredll.dll και τις AddIPAddress, DeleteIPAddress, CreateIpForwardEntry, DeleteIpForwardEntry και GetAdapterIndex από το αρχείο iphlpapi.dll.

6.2.1. CreateFile

Η CreateFile ανοίγει μια συσκευή ή έναν πόρο. Οι πιο συχνά χρησιμοποιούμενοι πόροι είναι: το αρχείο, ο κατάλογος, ο φυσικός δίσκος, το buffer, ένα network device, ένα pipe. Η CreateFile έχει διάφορες υπογραφές και αναλόγως ποια θα χρησιμοποιήσουμε θα πρέπει να την δηλώσουμε ρητά στον κώδικα. Τη χρησιμοποιούμε για να ανοίξουμε την εικονική συσκευή χρησιμοποιώντας την παρακάτω υπογραφή.

```
IntPtr CreateFile
(
    String Filename, uint Fileaccess, IntPtr Fileshare,
    IntPtr Securityattributes, uint Creationdisposition, uint Flags,
    IntPtr Template
)
```

Filename: Το όνομα της συσκευής. Επειδή δουλεύουμε με δικτυακό πόρο και όχι με κάποιο συγκεκριμένο αρχείο το filename είναι ένα path στην Registry.
Fileaccess: Ο τρόπος πρόσβασης στην συσκευή. Εμείς κάνουμε χρήση του **Both** επειδή θα ανοίξουμε την συσκευή και για INPUT και για OUTPUT. Μπορεί να είναι ένα από τα παρακάτω:

Πίνακας 10. File Access Codes

Read	GENERIC_READ	0x80000000
Write	GENERIC_WRITE	0x40000000
Both	GENERIC_READ | GENERIC_WRITE	0x10000000

Fileshare: Ο τρόπος μοιράσματος της συσκευής από άλλα handles δηλαδή από άλλες διεργασίες που θέλουν να την δεσμεύσουν. Χρησιμοποιούμε το **None** γιατί δεν θέλουμε η συσκευή να μοιράσει του πόρους της σε άλλα προγράμματα ταυτόχρονα.

Πίνακας 11. File Share Codes

Read	FILE_SHARE_READ	0x00000001
Write	FILE_SHARE_WRITE	0x00000002
None	FILE_SHARE_NONE	0x00000000

Securityattributes: Ορίζει διάφορα χαρακτηριστικά ασφάλειας. Επειδή δεν είναι χρήσιμα για εμάς δεν μπαίνουμε σε λεπτομέρειες και θέτουμε κατευθείαν τιμή μηδέν.

Creationdisposition: Τι ενέργεια να λάβει χώρα σε περίπτωση που η συσκευή υπάρχει ή δεν υπάρχει. Στην περίπτωση μας που ενεργούμε σε συσκευή μπορούμε να χρησιμοποιήσουμε μόνο την **OPEN_EXISTING** που σημαίνει ‘άνοιγμα μόνο αν υπάρχει’ . Οι διαθέσιμες ενέργειες είναι:

Πίνακας 12. Create Disposition Codes

CREATE_NEW	1
CREATE_ALWAYS	2
OPEN_EXISTING	3
OPEN_ALWAYS	4
TRUNCATE_EXISTING	5

Flags: Ορίζει ιδιαίτερα χαρακτηριστικά για την συσκευή. Εμείς χρησιμοποιούμε μόνο το **FILE_ATTRIBUTE_SYSTEM** καθώς σημαίνει πως η συσκευή είναι κομμάτι του λειτουργικού συστήματος. Μερικά από αυτά είναι:

Πίνακας 13. Flag Codes

FILE_ATTRIBUTE_NORMAL	0x80
FILE_ATTRIBUTE_HIDDEN	0x2
FILE_ATTRIBUTE_SYSTEM	0x4
FILE_FLAG_OVERLAPPED	0x40000000

Template: Η συγκεκριμένη παράμετρος δεν χρησιμοποιείται και η τιμή που της δίνεται είναι μηδέν.

Εάν η συνάρτηση ολοκληρωθεί επιτυχώς θα μας επιστρέψει μία τιμή που ονομάζουμε αναφορά (**handle**) και την οποία χρησιμοποιούμε αργότερα για να αποκτήσουμε πρόσβαση στην συσκευή. Εάν αποτύχει, μας επιστρέφει την -1 **.** Επίσης η CreateFile() ορίζει εάν η συσκευή θα ανοίξει σε synchronous ή asynchronous I/O. Σε synchronous operation όταν αρχίσει το I/O το πρόγραμμα μπλοκάρει έως ότου τελειώσει η εργασία ενώ σε asynchronous mode όταν αρχίσει το I/O το πρόγραμμα επιστρέφει αμέσως χωρίς να περιμένει να τελειώσει η εργασία. Σε ποια κατάσταση θα μπει η συσκευή ορίζεται στην παράμετρο flags με χρήση της σταθεράς 0x40000000.

6.2.2. CloseHandle

Η CloseHandle αποσυνδέει την αναφορά που αποκτήσαμε από την CreateFile από τον πόρο. Έχει και αυτή διάφορες υπογραφές και εμείς χρησιμοποιούμε την εξής: **int CloseHandle(IntPtr DevHandle)**

Η μοναδική παράμετρος είναι η αναφορά του πόρου του συστήματος που επιθυμούμε να αποσυνδέσουμε από την τρέχουσα διεργασία. Εάν η συνάρτηση δεν έχει ολοκληρωθεί επιτυχώς μας επιστρέφει μηδέν.

6.2.3. DeviceIoControl

Η DeviceIoControl έχει διάφορες υπογραφές και αναλόγως ποια θα χρησιμοποιήσουμε θα πρέπει να την δηλώσουμε ρητά στον κώδικα. Την

χρειαζόμαστε για να μιλήσουμε στον TAP driver και η υπογραφή της είναι η παρακάτω και επιστρέφει μηδέν αν δεν επιστρέψει επιτυχώς.

```
int DeviceIoControl
(
        SafeFileHandle DevHandle, uint IoControlCode, IntPtr InBuffer,
        uint InBufferSize, IntPtr OutBuffer, uint OutBufferSize,
        out int BytesReturned, IntPtr Overlapped
)
```

DevHandle: Η αναφορά στην εικονική συσκευή όπου η εκάστοτε λειτουργία πρέπει να εκτελεστεί. Την αναφορά την αποκτήσαμε από την συνάρτηση CreateFile().
IoControlCode: Ο κώδικας ελέγχου. Αντικατοπτρίζει την συγκεκριμένη λειτουργία και την συσκευή που θα πρέπει να την εκτελέσει. Ο κώδικας αυτός συνδυάζεται με την παράμετρο InBuffer που περιέχει τα δεδομένα που θα τον συνοδεύσουν.
InBuffer: Ένας pointer στο buffer που 'κατοικούν' τα δεδομένα που απαιτούνται από τον κώδικα ελέγχου. Το format αυτών των δεδομένων εξαρτάται από τον κώδικα ελέγχου.
InBufferSize: Το μέγεθος του παραπάνω buffer σε bytes.
OutBuffer: Ένας pointer στο buffer που θα αποθηκευτούν τα δεδομένα που επιστράφηκαν από την ολοκλήρωση της λειτουργίας. Το format αυτών των δεδομένων εξαρτάται από τον κώδικα ελέγχου.
OutBufferSize: Το μέγεθος του παραπάνω buffer σε bytes.
BytesReturned: Ένας pointer σε μια μεταβλητή που κρατάει το μέγεθος των δεδομένων σε bytes που αποθηκεύτηκαν στο OutBuffer .
Overlapped: Ένας pointer σε μια δομή τύπου **OVERLAPPED**. Εάν ανοίξαμε την συσκευή σε *synchronous operation* αυτή η παράμετρος δεν μας χρειάζεται και θέτουμε την τιμή της σε NULL. Εάν ανοίξαμε την συσκευή σε *asynchronous operation* αυτή η παράμετρος είναι υποχρεωτικό να δείχνει σε μια έγκυρη δομή. Παρακάτω δίνεται ο πίνακας με τις διαθέσιμες εντολές ελέγχου που υπάρχουν διαθέσιμες για τον TAP driver. Με έντονα δείχνω αυτές που χρειάστηκαν στο παρών project.

Πίνακας 14. Εντολές ελέγχου TAP driver

TAP_IOCTL_GET_MAC	TAP_CONTROL_CODE (1)
TAP_IOCTL_GET_VERSION	TAP_CONTROL_CODE (2)
TAP_IOCTL_GET_MTU	TAP_CONTROL_CODE (3)
TAP_IOCTL_GET_INFO	TAP_CONTROL_CODE (4)
TAP_IOCTL_SET_MEDIA_STATUS	TAP_CONTROL_CODE (6)
TAP_IOCTL_CONFIG_DHCP_MASQ	TAP_CONTROL_CODE (7)
TAP_IOCTL_GET_LOG_LINE	TAP_CONTROL_CODE (8)
TAP_IOCTL_CONFIG_DHCP_SET_OPT	TAP_CONTROL_CODE (9)
TAP_IOCTL_CONFIG_TUN	TAP_CONTROL_CODE (10)
TAP_IOCTL_QUEUE_READ	TAP_CONTROL_CODE (20)
TAP_IOCTL_CANCEL_IO	TAP_CONTROL_CODE (21)

6.2.4 ReadFile

Επειδή δεν υπάρχουν δομητές FileStream στο .NET Compact Framework για Windows Mobile που να υποστηρίζουν άνοιγμα συσκευής βάση του handle αναγκαζόμαστε να καλέσουμε τις ReadFile() και WriteFile(). Η ReadFile διαβάζει τα δεδομένα από το outgoing data buffer μιας συσκευής. Η συνάρτηση σχεδιάστηκε και για *synchronous* και για *asynchronous operations.* Έχει διάφορες υπογραφές και αναλόγως ποια θα χρησιμοποιήσουμε θα πρέπει να την δηλώσουμε ρητά στον κώδικα. Η υπογραφή της είναι η παρακάτω και επιστρέφει μηδέν αν δεν επιστρέψει επιτυχώς.

```
int ReadFile
(
        IntPtr DevHandle, byte[] Buffer, int NumberOfBytesToRead,
        out int NumberOfBytesRead, IntPtr Overlapped
)
```

DevHandle: Η αναφορά στην εικονική συσκευή από την οποία θέλουμε να διαβάσουμε δεδομένα. Την αναφορά την αποκτήσαμε από την συνάρτηση CreateFile().
Buffer: Ένας pointer στο buffer όπου θα αποθηκεύσουμε τα δεδομένα που διαβάσαμε από την συσκευή. Το buffer πρέπει να μείνει έγκυρο καθ' όλη την διάρκεια του διαβάσματος.
NumberOfBytesToRead: Τον μέγιστο πλήθος των bytes που θέλουμε να διαβάσουμε.
NumberOfBytesRead: Η μεταβλητή στην οποία αποθηκεύτηκε το πλήθος των bytes που διαβάστηκαν.
Overlapped: Ένας pointer σε μια δομή τύπου **OVERLAPPED**. Εάν ανοίξαμε την συσκευή σε *synchronous operation* αυτή η παράμετρος δεν μας χρειάζεται και θέτουμε την τιμή της σε NULL. Εάν ανοίξαμε την συσκευή σε *asynchronous operation* αυτή η παράμετρος είναι υποχρεωτικό να δείχνει σε μια έγκυρη δομή.

6.2.5. WriteFile

Η WriteFile γράφει δεδομένα στο incoming data buffer μιας συσκευής. Η εν λόγω συνάρτηση σχεδιάστηκε και για *synchronous* και για *asynchronous operations.* Έχει διάφορες υπογραφές και αναλόγως ποια θα χρησιμοποιήσουμε θα πρέπει να την δηλώσουμε ρητά στον κώδικα. Η υπογραφή της είναι η παρακάτω και επιστρέφει μηδέν αν δεν επιστρέψει επιτυχώς.

```
int WriteFile
(
    IntPtr DevHandle, byte[] Buffer, int NumberOfBytesToWrite,
    out int NumberOfBytesWritten, IntPtr Overlapped
)
```

DevHandle: Η αναφορά στην εικονική συσκευή στην οποία θέλουμε να γράψουμε δεδομένα. Την αναφορά την αποκτήσαμε από την συνάρτηση CreateFile().
Buffer: Ένας pointer στο buffer όπου υπάρχουν τα δεδομένα που θέλουμε να γράψουμε στην συσκευή.

NumberOfBytesToWrite: Το πλήθος των bytes που θέλουμε να γράψουμε.
NumberOfBytesWritten: Η μεταβλητή στην οποία αποθηκεύτηκε το πλήθος των bytes που γράφηκαν.
Overlapped: Ένας pointer σε μια δομή τύπου **OVERLAPPED**. Εάν ανοίξαμε την συσκευή σε *synchronous operation* αυτή η παράμετρος δεν μας χρειάζεται και θέτουμε την τιμή της σε NULL. Εάν ανοίξαμε την συσκευή σε *asynchronous operation* αυτή η παράμετρος είναι υποχρεωτικό να δείχνει σε μια έγκυρη δομή, ειδάλλως οι επιπτώσεις θα είναι απρόβλεπτες. Επειδή τα Windows Mobile δεν υποστηρίζουν asynchronous operations θέτουμε την τιμή σε NULL. Κώδικας στο παράρτημα στο Code Snippet 8 .

6.2.6. AddIPAddress

Η συνάρτηση AddIPAddress θέτει μια IPv4 διεύθυνση σε μία συγκεκριμένη συσκευή. Εμείς την χρειαζόμαστε για να αποδώσουμε διεύθυνση στην εικονική συσκευή ώστε να εδραιωθεί το private δίκτυο με τον iodine server. Η υπογραφή της είναι η παρακάτω και επιστρέφει μηδέν αν επιστρέψει επιτυχώς.

```
uint AddIPAddress
(
        UInt32 Address, UInt32 Mask, uint IfIndex,
        out IntPtr NTEContext, out IntPtr NTEInstance
)
```

Address, η διεύθυνση ως unsigned integer σε network byte order
Mask, η μάσκα υποδικτύου ως unsigned integer network byte order
IfIndex, το index του interface που αποκτήσαμε από την συνάρτηση GetAdapterIndex του iphlpapi.dll
NTEContext, ένας pointer σε μια unsigned long μεταβλητή. Σε περίπτωση που έχουμε επιτυχή επιστροφή αυτή η παράμετρος δείχνει στο Net Table Entry (NTE) context της διεύθυνσης που προσθέσαμε. Έχοντας αυτό το context το χρησιμοποιούμε αργότερα με την συνάρτηση DeleteIPAddress για να διαγράψουμε την διεύθυνση από το interface.

NTEInstance, ένας pointer σε μια unsigned long μεταβλητή. Σε περίπτωση που έχουμε επιτυχή επιστροφή αυτή η παράμετρος δείχνει στο Net Table Entry instance της διεύθυνσης που προσθέσαμε. Η IPv4 διεύθυνση που βάζουμε δεν είναι μόνιμη. Υπάρχει όσο το interface είναι ενεργό. Επίσης ορισμένα PnP events μπορεί να διαγράψουν την διεύθυνση όπως επίσης και η επανεκίνηση του adapter. Γι'αυτό και χρησιμοποιούμε τον κώδικα ελέγχου *CONFIG_DHCP_MASQ* που αναλύσαμε στην ενότητα 5.7. Κώδικας στο παράρτημα στο Code Snippet 9 .

6.2.7. DeleteIPAddress

Η συνάρτηση DeleteIPAddress διαγράφει την IPv4 διεύθυνση βάση του NTEContext που αποκτήσαμε από την AddIPAddress. Η υπογραφή της είναι η παρακάτω και την χρησιμοποιούμε κατά το κλείσιμο της εφαρμογής:

```
uint DeleteIPAddress(IntPtr NTEContext)
```

6.2.8. CreateIpForwardEntry

Αυτή η συνάρτηση καταχωρεί ένα δρομολόγιο στου τοπικού υπολογιστή τον IPv4 πίνακα δρομολογίων. Η χρήση της είναι απαραίτητη για να ανακατευθύνουμε την τοπική IP κίνηση του μηχανήματος προς την εικονική συσκευή. Η υπογραφή της είναι η παρακάτω επιστρέφει μηδέν αν επιστρέψει επιτυχώς και **Route** είναι μια αναφορά με τύπο την δομη **MIB_IPFORWARDROW**.

```
uint CreateIpForwardEntry(ref MIB_IPFORWARDROW Route)

struct MIB_IPFORWARDROW {
        UInt32 Destination; UInt32 Mask; UInt32 Policy; UInt32 NextHop;
        UInt32 Index; UInt32 Type; UInt32 Proto; UInt32 Age;
        UInt32 NextHopAS; int Metric1; int Metric2; int Metric3; int Metric4;
        int Metric5;
}
```

Destination, ο προορισμός μας
Mask, η μάσκα υποδικτύου του προορισμού
Policy, πολιτική για multi-path route
NextHop, το επόμενο μηχάνημα στο δρομολόγιο
Index, το index της συσκευής οπου θέτουμε το route
Type, αν το route είναι direct ή indirect δηλαδή αν το NextHop είναι local ή remote
Proto, ο μηχανισμός που παρήγαγε το route π.χ OSPF ή static
Age, τα δευτερόλεπτα που είναι εγκατεστημένο το route
NextHopAS, το αυτόνομο σύστημα του NextHop όταν το Proto είναι OSPF
MetricX, η τιμή του metric που θέτει κάποιο πρωτόκολλο δρομολόγησης

6.2.9. DeleteIpForwardEntry

Αυτή η συνάρτηση διαγράφει ένα υπάρχων δρομολόγιο στου τοπικού υπολογιστή τον IPv4 πίνακα δρομολογίων. Η υπογραφή της είναι η παρακάτω την χρησιμοποιούμε κατά το κλείσιμο της εφαρμογής και **Route** είναι η αναφορά που χρησιμοποιήσαμε στην CreateIpForwardEntry:
uint DeleteIpForwardEntry(ref MIB_IPFORWARDROW Route)

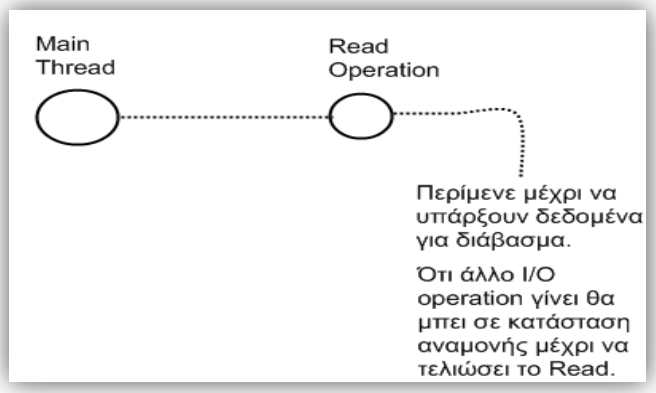
6.2.10. GetAdapterIndex

Αυτή η συνάρτηση μας επιστρέφει το index μιας συσκευής βάση του ονόματος της. Η υπογραφή της είναι η παρακάτω, επιστρέφει μηδέν αν ολοκληρωθεί επιτυχώς και την χρησιμοποιούμε για να πάρουμε τα indexes όλων των δικτυακών συσκευών καθώς θα μας χρειαστούν για να θέσουμε αργότερα τα route: **uint GetAdapterIndex(string adapter, out uint index)** όπου **adapter** το όνομα της συσκευής στο σύστημα και **index** η μεταβλητή που θα περιέχει το Index.

6.3. Asynchronous I/O στο TAP Device

Στο παρών project συναντιέται μια περίπτωση όπου το I/O βγαίνει σε αδιέξοδο. Στη παράγραφο 4.5 εξηγήσαμε γιατί χρειαζόμαστε το Polling. Τα ‘P’

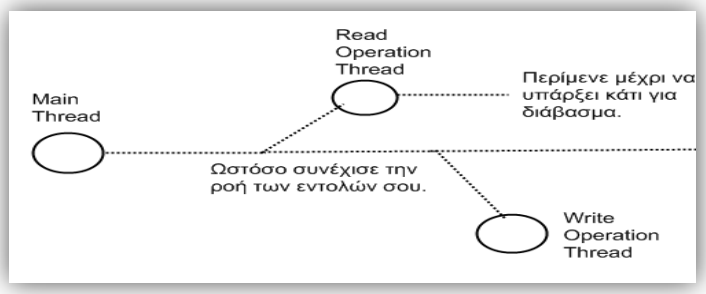
packets που στέλνουμε παράλληλα χρησιμοποιώντας ξεχωριστό thread μπορεί τελικά να μας προμήθευσαν τα υπολειπόμενα δεδομένα και να καλέσαμε την WriteFile() για να τα γράψουμε αλλά αφού ακόμα δεν έχει ολοκληρωθεί η ReadFile() που εκτελείται από το main thread μπλοκάρονται και οι WriteFile(). Είδαμε την περίπτωση όπου ο iodine server πρέπει να λάβει τα 'P' packets για να στείλει τα υπολειπόμενα DNS Response Messages. Χωρίς αυτά, προφανώς ο client σε επίπεδο εφαρμογής δεν έχει λάβει ακόμα ένα ολόκληρο πακέτο IP και γι' αυτό δεν πρόκειται να αποκριθεί. Έτσι το read buffer της εικονικής συσκευής δεν έχει δεδομένα και η ReadFile() που καλούμε μπαίνει σε διαρκή αναμονή. Το πρόγραμμα εισέρχεται σε μια κατάσταση παγώματος και δεν μπορεί να συνεχίσει παρά μόνο όταν η ReadFile() επιστρέψει. Μέχρι να ολοκληρωθεί το πρώτο I/O operation τότε όλα τα άλλα I/O operations που θα κληθούν θα εισέλθουν σε κατάσταση αναμονής μέχρι να ολοκληρωθεί το αρχικό.



Εικόνα 58. Synchronous I/O

Το παραπάνω πρόβλημα έχει δύο διαφορετικές λύσεις για τις δύο εκδόσεις της εφαρμογής σε windows και windows mobile αντίστοιχα. Για την έκδοση των windows μπορεί να λυθεί χρησιμοποιώντας asynchronous I/O. Το asynchronous I/O ξεκινάει οποιοδήποτε operation σε ξεχωριστό thread από αυτό του main thread. Έτσι, η ReadFile() που είναι μπλοκαρισμένη σε άλλο thread περιμένει. Όμως το main thread συνεχίζει κανονικά την εκτέλεση του και τα 'P' packets που μας φέρνουν δεδομένα καλούν ανεμπόδιστα τις WriteFile() και τα γράφουν. Μόλις γραφούν όλα τα δεδομένα στην συσκευή, αυτή προφανώς αποκτάει ένα νέο πακέτο IP κάνει ότι είναι να κάνει αυτό το πακέτο και έπειτα αποκτάει ένα άλλο

πακέτο IP προς αποστολή το οποίο θα γραφεί στο Read buffer και η ReadFile() ξεμπλοκάρει.



Εικόνα 59. Asynchronous I/O

Η κλάση FileStream που μας παρέχει το .NET Framework υποστηρίζει Asynchronous I/O χρησιμοποιώντας έναν από τους δομητές της. Έπειτα χρησιμοποιούμε τις μεθόδους BeginWrite() και BeginRead() οι οποίες ξεκινούν το asynchronous operation η κάθε μία σε ξεχωριστό thread. Να μην ξεχάσουμε όμως όταν ανοίξουμε την συσκευή με την CreateFile() να δηλώσουμε ότι την ανοίγουμε σε overlapped mode θέτοντας στο όρισμα flags την τιμή [**FILE_FLAG_OVERLAPPED**] .

Ωστόσο, για την έκδοση των windows mobile τα πράματα είναι πολύ διαφορετικά. Η πλατφόρμα των mobile δεν υποστηρίζει γενικότερα Asynchronous I/O. Έτσι όταν χρησιμοποιήσουμε την CreateFile() για να ανοίξουμε την συσκευή θέτουμε στο όρισμα flags αντί του FILE_FLAG_OVERLAPPED το [**FILE_ATTRIBUTE_SYSTEM**] αλλιώς η συσκευή δεν θα ανοίξει. Επίσης καλούμε πάντα την ReadFile() δίνοντας στο όρισμα lpOverlapped την τιμή NULL. Πλέον, έχοντας απορρίψει το μοντέλο του Asynchronous I/O δεν υπάρχει άλλος προγραμματιστικός τρόπος να λύσουμε το πρόβλημα. Η ReadFile δεν επιστρέφει μέχρις ότου η λειτουργία ολοκληρωθεί και αυτό θα γίνει όταν υπάρξουν δεδομένα για κατανάλωση. Όσο δεν υπάρχουν δεδομένα να διαβάσουμε η εφαρμογή μπλοκάρει στο συγκεκριμένο σημείο και αυτό θα μας παρούσιαζε ένα μεγάλο εμπόδιο αλλά ο David Lemley developer του TAP driver έχει ενσωματώσει στον TAP driver ένα εξομοιωτή για Asynchronous I/O. Έτσι αν η ReadFile δεν έχει δεδομένα προς κατανάλωση θα μας επιστρέψει μηδέν. Όμως επειδή επιστρέφει αμέσως η ροή του προγράμματος συνεχίζει κανονικά και πλέον δεν έχουμε το πρόβλημα της διαρκούς αναμονής. Κώδικας στο παράρτημα στο Code Snippet 7 .

Κεφάλαιο έβδομο

Συμπεράσματα και Μελλοντικές Κατευθύνσεις

7.1. Συμπεράσματα

- Το DNS εφαρμόζεται παγκοσμίως
- Ενώ όλο το υπόλοιπο IP networking φιλτράρεται συνεχώς και περισσότερο, το DNS παραμένει το ίδιο από τότε που δημιουργήθηκε.
- Το προηγούμενο έχει ως αποτέλεσμα να μπορεί κάποιος ανενόχλητα να περάσει και να λάβει δεδομένα σε επίπεδο byte.
- Το DNS δε μπορεί να απενεργοποιηθεί, να φιλτραριστεί και να ξαναγραφεί.

Η εφαρμογή από τη φύση της δεν μπορεί να έχει ευρεία χρήση. Σήμερα σε κάθε γειτονιά θα βρεις και ένα Wi-Fi δίκτυο που θα μοιράζει internet χωρίς χρέωση. Δύο γνωστά σενάρια στα οποία μπορεί να χρησιμοποιηθεί είναι στα Wi-Fi και 3G hotspots των ξενοδοχείων, αεροδρομίων, καραβιών και σε χώρες όπου γίνεται λογοκρισία από τις κυβερνήσεις τους. Σε αυτές τις περιπτώσεις είναι κλειδωμένα όλα τα πρωτόκολλα 4ου επιπέδου αλλά επιτρέπονται τα DNS lookups.

Ωστόσο, είναι ένα ενδιαφέρον πείραμα που μας δείχνει πως μπορούμε να το εκμεταλλευτούμε το DNS ώστε να φτιάξουμε ένα κρυφό μονοπάτι αμφίδρομης επικοινωνίας για πακέτα IP. Η εφαρμογή για Windows Mobile που δοκιμάστηκε εντός Ελλάδας πέτυχε 10KB/s downstream και 1-2KB/s upstream. Στην Γερμανία όπου χρησιμοποιήθηκε και από τους πρώτους χρήστες της έπιασε 100KB/s downstream και 5KB/s upstream.

7.2. Μελλοντικές κατευθύνσεις

Το παρών project αποτελεί μοναδική δουλειά παγκοσμίως και είναι μια δυνατή αρχή για μελλοντικές βελτιώσεις και προσθέσεις. Μερικά ακόμη πράματα θα ήταν καλό να γίνουν.

Το πρώτο είναι η αναβάθμιση των εφαρμογών στη τελευταία έκδοση του Iodine protocol από την τρέχουσα που είναι 0.4.2. Οι νεότερες εκδόσεις υποστηρίζουν αυτό που λέγεται **downstream fragmentation** και **auto probing of downstream size**. Η πρώτη λειτουργία εφαρμόζεται στον server και στον client και αυτό που κάνει είναι να στέλνει τα response messages σε κομμάτια μικρότερου

μεγέθους ώστε να μην εντοπίζονται από τυχών firewalls που εμποδίζουν την μετάδοση ασυνήθιστων πακέτων DNS. Η δεύτερη λειτουργία εφαρμόζεται στον client και είναι ένας αλγόριθμος ο οποίος υπολογίζει το μέγιστο μέγεθος ενός πακέτου DNS που μπορεί να φθάσει στον client χωρίς να ‘πιαστεί’ από ένα firewall. Όταν υπολογιστεί το μέγεθος αυτό ο client ενημερώνει τον server και τον προγραμματίζει να χρησιμοποιήσει το downstream fragmentation κάνοντας χρήση αυτού του μεγέθους.

Το δεύτερο είναι η ανάπτυξη της server-side εφαρμογής για Windows πλατφόρμες. Δεν είναι επιτακτική ανάγκη αλλά θα ήταν καλύτερο να υπάρχει πλήρη συμβατότητα με τον TUN/TAP Driver που είναι κοινός για όλες τις πλατφόρμες Windows.

Το τελευταίο αντικείμενο με το οποίο θα μπορούσε να κάνει κάποιος είναι να φτιαχτεί ένας wrapper γύρω από όλες τις unmanaged μεθόδους που χρησιμοποιούμε. Αυτός ο wrapper θα έκρυβε τα στοιχεία της C++ που απαιτούνται και που θα μπορούσαν να οδηγήσουν το πρόγραμμα σε πιθανά σφάλματα. Θα έδινε στον προγραμματιστή ένα πιο εύκολο API.

Κεφάλαιο όγδοο

Παράρτημα Α: Code Snippets

```
95  // Packet Constructor
96  public DnsPacket(byte[] qname)
97  {
98      this.questionSize = qname.Length + 4;
99      this.query = new byte[12 + questionSize + 11];
100     this.id = BitConverter.GetBytes((ushort)new Random().Next(1, 65535));
101
102     // DNS MESSAGE - HEADER PART
103
104     query[0] = id[0];   // Transaction ID.
105     query[1] = id[1];
106     query[2] = 0x1;     // QR-Opcode-AA-TC-RD-RA-Z-RCODE = 0-0000-0-0-1-0-000-0000 = 1 0
107     query[3] = 0x0;
108     query[4] = 0x0;     // QCOUNT - Question counter = 01
109     query[5] = 0x1;
110     query[6] = 0x0;     // ANCOUNT - Answer counter = 00
111     query[7] = 0x0;
112     query[8] = 0x0;     // NSCOUNT - Authority counter = 00
113     query[9] = 0x0;
114     query[10] = 0x0;    // ARCOUNT - Resource record counter = 01
115     query[11] = 0x1;
116
117     // DNS MESSAGE - QUESTION PART
118
119     Array.Copy(qname, 0, query, 12, qname.Length);      // Add qname to packet
120     query[qname.Length + 12] = 0x0; // Add the TYPE of question which is 'NULL'
121     query[qname.Length + 13] = 0xA;
122     query[qname.Length + 14] = 0x0; // Add the CLASS of question which is 'IN'
123     query[qname.Length + 15] = 0x1;
124
125     // DNS MESSAGE - EDNS PART - 11 bytes (extend default udp message size)
126
127     query[qname.Length + 16] = 0x0;
128     query[qname.Length + 17] = 0x0;
129     query[qname.Length + 18] = 0x29;
130     query[qname.Length + 19] = 0x10;
131     query[qname.Length + 20] = 0x0;
132     query[qname.Length + 21] = 0x0;
133     query[qname.Length + 22] = 0x0;
134     query[qname.Length + 23] = 0x80;
135     query[qname.Length + 24] = 0x0;
136     query[qname.Length + 25] = 0x0;
137     query[qname.Length + 26] = 0x0;
138 }
```

Εικόνα 60. Code Snippet 1

```
157 public byte[] ExtractAnswer(byte[] response)
158 {
159     // Check whether the packet is a response.
160     if (response[2] >> 7 == 0x1)
161     {
162         // Compare transaction ids'
163         if (this.id[0] != response[0] || this.id[1] != response[1])
164             return new byte[] { 0x0, response[0], response[1] };
165 
166         // Check whether response has errors.
167         if ((response[3] & 0x7) == 0x0)
168         {
169             try
170             {
171                 // Calculate length of RDATA
172                 byte r1 = response[12 + questionSize + 10];
173                 byte r2 = response[12 + questionSize + 11];
174                 int rdlength = r1 << 8 | r2;
175 
176                 if (rdlength == 0)
177                     return new byte[] { 0x8 };
178 
179                 // Position at the beginning of RDATA = header + question + RR header
180                 int position = 12 + questionSize + 12;
181 
182                 // Collect the RDATA
183                 byte[] rdata = new byte[rdlength];
184                 Array.Copy(response, position, rdata, 0, rdlength);
185                 return rdata;
186             }
187             catch (Exception)
188             {
189                 return new byte[] { 0x6 };
190             }
191         }
192         else if ((response[3] & 0x7) == 1) return new byte[] { 0x1 }; // Format Error
193         else if ((response[3] & 0x7) == 2) return new byte[] { 0x2 }; // Server Failure
194         else if ((response[3] & 0x7) == 3) return new byte[] { 0x3 }; // Name in query does not exist
195         else if ((response[3] & 0x7) == 4) return new byte[] { 0x4 }; // Query not supported
196         else if ((response[3] & 0x7) == 5) return new byte[] { 0x5 }; // Query Refused
197         else return null;
198     }
199     else return new byte[] { 0x7 }; // Packet is a not a dns response;
200 }
```

Εικόνα 61. Code snippet 2

```
83  private static void Pack(byte[] in5, out byte[] out8)
84  {
85
86      out8 = new byte[8];
87
88      // A 40bit group can only fit in a 64-bit memory cell.
89      UInt64 group = 0;
90
91      // Join 5 bytes to produce a 40bit group.
92      for (int i = 0; i < in5.Length; i++)
93      {
94          if (i != 0)
95          {
96              group = group << 8;
97          }
98          group = group | in5[i];
99      }
100
101     // Then take 40bit group and cut it in 8 pieces of 5 bits.
102     // Run this loop 8 times, each time will form a 5bit piece.
103     for (int j = 7; j >= 0; j--)
104     {
105         // Form the 5bit piece and store it in a unsigned byte. Must be up to 31.
106         out8[7 - j] = (byte)((group >> (j * 5)) & 31);
107     }
108 }
```

Εικόνα 62. Code snippet 3

```
26  public static byte[] compress(byte[] data)
27  {
28      MemoryStream compressed = new MemoryStream();
29      DeflaterOutputStream c = new DeflaterOutputStream(compressed, new Deflater(9));
30      c.Write(data, 0, data.Length);
31      c.Close();
32      return compressed.ToArray();
33  }
34
35  public static byte[] decompress(byte[] data)
36  {
37      byte[] buffer = new byte[8 * 1024];
38
39      MemoryStream uncompressed = new MemoryStream(data);
40      InflaterInputStream d = new InflaterInputStream(uncompressed);
41      int length = d.Read(buffer, 0, buffer.Length);
42      d.Close();
43
44      byte[] outBytes = new byte[length];
45      Array.Copy(buffer, 0, outBytes, 0, length);
46      return outBytes;
47  }
```

Εικόνα 63. Code snippet 4

```
124 public static byte[] make_hostname(string base32Str)
125 {
126     if (base32Str.Length > 63)
127     {
128         string sum = "";
129         int labels = base32Str.Length / 63;
130         int remaining_bytes = base32Str.Length % 63;
131 
132         for (byte b = 0; b < labels; b++) {
133             sum += base32Str.Substring(b * 63, 63);
134             if (b < labels - 1)
135                 sum += '.';
136         }
137 
138         if (remaining_bytes > 0)
139             sum += '.' + base32Str.Substring(labels * 63, remaining_bytes);
140 
141         // leading dot + topdomain + Root Domain dot
142         base32Str = '.' + sum + topDomain + '.';
143     }
144     else {
145         // leading dot + topdomain + Root Domain dot
146         base32Str = '.' + base32Str + topDomain + '.';
147     }
148 
149     byte[] qname = encoder.GetBytes(base32Str);
150     int dIndex = 0;
151     byte ndcounter = 0;
152 
153     for (int i = 0; i < qname.Length; ) {
154         if (qname[i] == 46) {
155             dIndex = i;
156             i++;
157         }
158         else {
159             while (qname[i] != 46) {
160                 ndcounter++;
161                 i++;
162             }
163             qname[dIndex] = ndcounter;
164             ndcounter = 0;
165         }
166     }
167     qname[qname.Length - 1] = 0x0; // QNAME is now formatted based on RFC standart.
168     return qname;
169 }
```

Εικόνα 64. Code snippet 5

```
238 public static void dn_fragmentation(byte[] zIPTunPacket)
239 {
240     byte[] chunk; byte[] hostname; byte[] rdata;
241     int offset = 0; int csize = 0;  int esize = 0; int hsize = 0;
242 
243     csize = zIPTunPacket.Length;
244     esize = Base32.GetBase32Length(csize);
245     // hsize = Current Encoded Length + Current #dots + Reserved
246     hsize = esize + (esize + 1) / 63 + ReservedSize;
247 
248     if (hsize > 255) {
249         // Split zIPTunPacket into chunks sized 255 bytes and send them one by one.
250         while (esize > base32L) {
251             esize = Base32.GetBase32Length(--csize);
252             if (esize == base32L) {
253                 chunk = new byte[csize];
254                 Array.Copy(zIPTunPacket, offset, chunk, 0, csize);
255                 chunk_header = (byte)((chunk_header >> 1) | 0);
256                 string base32Str = chunk_header.ToString() + Base32.Encoder(chunk);
257                 hostname = Dns.make_hostname(base32Str);
258                 Dns.send_query(hostname);
259 
260                 offset += csize;
261                 csize = zIPTunPacket.Length - offset;
262                 esize = Base32.GetBase32Length(csize);
263             }
264         }
265         chunk = new byte[csize];
266         Array.Copy(zIPTunPacket, offset, chunk, 0, csize);
267     }
268     else
269         chunk = zIPTunPacket;
270 
271     // Send last chunk and write received decompressed data
272     chunk_header = (byte)((chunk_header >> 1) | 1);
273     string lbase32Str = chunk_header.ToString() + Base32.Encoder(chunk);
274     hostname = Dns.make_hostname(lbase32Str);
275     rdata = Dns.send_query(hostname);
276 
277     if (rdata != null)
278         TapAdapter.WriteTun(Zlib.decompress(rdata));
279 
280     // Send P packets in order to poll the server for any IP packets left behind
281     do {
282         rdata = Send_P();
283         if (rdata != null)
284             TapAdapter.WriteTun(Zlib.decompress(rdata));
285     }
286     while (rdata != null && rdata.Length > 1);
287 }
```

Εικόνα 65. Code snippet 6

```
68  public static void ReadTun(object source, ElapsedEventArgs e)
69  {
70      int bytesRead = 0;
71      byte[] Readbuffer = new byte[1500];
72      byte[] zIPTunPacket;
73
74      if (Coredll.ReadFile(handle, Readbuffer, Readbuffer.Length, out bytesRead, IntPtr.Zero) != 0)
75      {
76          readTimer.Stop();
77          if (bytesRead > 20)
78          {
79              GraphicalInterface.getInstance().TXIPPacketsLogger(++TXIPPackets);
80              // Before we send the ip packet we must add the tun header
81              // 0          4                                    n
82              // ---------------------------------------------
83              // Tun Header |             IP Packet
84              // ---------------------------------------------
85
86              // Add the linux tun header to OutBytes
87              byte[] OutBytes = new byte[bytesRead + 4];
88              OutBytes[0] = 0x0;
89              OutBytes[1] = 0x0;
90              OutBytes[2] = 0x8;
91              OutBytes[3] = 0x0;
92
93              // Add the actual IP Datagram
94              Array.Copy(Readbuffer, 0, OutBytes, 4, bytesRead);
95
96              // Send it for compression
97              zIPTunPacket = Zlib.compress(OutBytes);
98
99              Iodine.StopTimer();
100
101             // Manipulate it in a suitable way to be tunnel-ed
102             Iodine.dn_fragmentation(zIPTunPacket);
103
104             Iodine.StartTimer();
105         }
106         readTimer.Start();
107     }
108     else
109     {
110         GraphicalInterface.getInstance().MsgLogger("DeviceIoControl Error: " +
111             new Win32Exception().Message);
112     }
113 }
```

Εικόνα 66. Code snippet 7

```
115 public static void WriteTun(byte[] data)
116 {
117     // Data should be at least an IP Header(20 bytes)
118     // Anything below 20 bytes is a malformed packet
119     if (data != null && data.Length > 20)
120     {
121         // Omit the first 4 bytes which is the linux tun header
122         byte[] packet = new byte[data.Length - 4];
123         Array.Copy(data, 4, packet, 0, data.Length - 4);
124
125         int BytesWritten = 0;
126         if (Coredll.WriteFile(handle, packet, packet.Length, out BytesWritten, IntPtr.Zero) != 0)
127         {
128             //GraphicalInterface.getInstance().MsgLogger("Written " + BytesWritten);
129         }
130         else
131         {
132             GraphicalInterface.getInstance().MsgLogger("Couldn't write: " + packet.Length + " bytes");
133         }
134
135         GraphicalInterface.getInstance().RXIPPacketsLogger(++RXIPPackets);
136     }
137 }
```

Εικόνα 67. Code snippet 8

```
145 public static bool AddStaticIP(string ipAddress, string netmask, uint AdapterIndex)
146 {
147     IntPtr NTEContext = new IntPtr(0);
148     IntPtr NTEInstance = new IntPtr(0);
149
150     // Enable TAP interface and set it's network
151     uint result = Iphlpapi.AddIPAddress(IP2Int(ipAddress), IP2Int(netmask), AdapterIndex,
152                                         out NTEContext, out NTEInstance);
153
154     if (result != 0)
155     {
156         GraphicalInterface.getInstance().MsgLogger("Failed to add static IP on TUN adapter.");
157         GraphicalInterface.getInstance().MsgLogger("AddIPAddress returned: " + result);
158         return false;
159     }
160     else
161     {
162         GraphicalInterface.getInstance().MsgLogger("Added IP Address on TUN adapter.");
163         TapIP_NTE = NTEContext;
164         TAPIP_Added = true;
165         return true;
166     }
167 }
```

Εικόνα 68. Code snippet 9

```
43 public static string Encoder(byte[] bytes)
44 {
45
46     byte[] outbuf = new byte[8];
47     byte[] inbuf = new byte[5];
48     int outsize = 0;
49     string result = "";
50
51     for (int j = 0; j < bytes.Length / 5; j++)
52     {
53         Array.Copy(bytes, j * 5, inbuf, 0, 5); // Take 5 bytes
54         Pack(inbuf, out outbuf);               // Produce 8 5-bit pieces
55         for (int i = 0; i < 8; i++)            // Get 8 corresponding characters from base32 alphabet
56             result += b32a[outbuf[i]];
57     }
58
59     // How many bytes remain that will form a group less than 40 bits
60     int r = bytes.Length % 5;
61     if (r > 0)
62     {
63         // We don't need previous inbuf elements. Zero all.
64         inbuf.Initialize();
65
66         // Copy to inbuf the rest bytes that form a group less than 40 bits.
67         Array.Copy(bytes, bytes.Length - r, inbuf, 0, r);
68
69         // Don't count the padding pieces.
70         outsize = ((r * 8) / 5) + 1;
71
72         // Produce 8 5bit pieces
73         Pack(inbuf, out outbuf);
74
75         // Get corresponding characters from base32 alphabet
76         for (int i = 0; i < outsize; i++)
77             result += b32a[outbuf[i]];
78     }
79
80     return result.ToString();
81 }
```

Εικόνα 69. Code snippet 10

```
73  private static bool send_version()
74  {
75      byte[] sdata = new byte[6];
76      sdata[0] = (0x00000402 >> 24) & 0xff;
77      sdata[1] = (0x00000402 >> 16) & 0xff;
78      sdata[2] = (0x00000402 >> 8) & 0xff;
79      sdata[3] = (0x00000402 >> 0) & 0xff;
80      sdata[4] = (0x0 >> 8) & 0xff;
81      sdata[5] = (0x0 >> 0) & 0xff;
82
83      rand_seed++;
84      string base32Str = 'V' + Base32.Encoder(sdata); // Encode sdata to base32
85      byte[] hostname = Dns.make_hostname(base32Str); // Build hostname
86      byte[] rdata = Dns.send_query(hostname);        // Send it
87
88      if (rdata != null && rdata.Length == 9)
89      {
90          string rdataStr = Encoding.Default.GetString(rdata, 0, rdata.Length);
91          if (rdataStr.StartsWith("VACK"))
92          {
93              Array.Copy(rdata, 4, challenge, 0, 4);  // Copy login challenge
94              userid = rdata[rdata.Length - 1];       // Copy user id
95              header = userid;
96              GraphicalInterface.getInstance().MsgLogger("You are user #" + userid);
97              return true;
98          }
99          else if (rdataStr.StartsWith("VNAK")) {
100             GraphicalInterface.getInstance().MsgLogger("Version mismatch. Quiting..");
101             return false;
102         }
103         else if (rdataStr.StartsWith("VFUL")) {
104             GraphicalInterface.getInstance().MsgLogger("All slots already taken. Try again later..");
105             return false;
106         }
107         else {
108             GraphicalInterface.getInstance().MsgLogger("Handshake error");
109             return false;
110         }
111     }
112     else
113         return false;
114 }
```

Εικόνα 70. Code snippet 11

```
139 public static bool set_layer3_mode(string local_ip, string remote_netmask)
140 {
141     // local ip
142     int IpAddress = BitConverter.ToInt32(IPAddress.Parse(local_ip).GetAddressBytes(), 0);
143
144     // remote netmask
145     int RemNetMask = BitConverter.ToInt32(IPAddress.Parse(remote_netmask).GetAddressBytes(), 0);
146
147     // remote network
148     int NetAddress = IpAddress & RemNetMask;
149
150     // For ARP
151     IntPtr network = Marshal.AllocHGlobal(12);
152     Marshal.WriteInt32(network, 0, IpAddress);
153     Marshal.WriteInt32(network, 4, NetAddress);
154     Marshal.WriteInt32(network, 8, RemNetMask);
155
156     // Put device in TUN mode, enable ARP and masq the dhcp
157     if (Coredll.DeviceIoControl(handle,
158         Driver.TAP_CONTROL_CODE(10, 0),
159         network,
160         12,
161         network,
162         12,
163         0,
164         IntPtr.Zero) != 0)
165     {
166         GraphicalInterface.getInstance().MsgLogger("Switched to TUN mode with ARP enabled.");
167         return true;
168     }
169     else
170     {
171         GraphicalInterface.getInstance().MsgLogger("Could not switch to tun mode.");
172         GraphicalInterface.getInstance().ConnectButtonText("Connect");
173         GraphicalInterface.getInstance().ConnectButtonEnabled(true);
174         return false;
175     }
176 }
```

Εικόνα 71. Code snippet 12

```
128 private static bool send_login()
129 {
130     byte[] hash = new byte[16]; byte[] chal = new byte[32];
131     byte[] temp = new byte[32]; byte[] pass = new byte[32];
132     byte[] sdata = new byte[19];
133
134     // make 8 repetitions of login challenge
135     for (int i = 0; i < 32; i += 4) {
136         chal[i] = challenge[0];
137         chal[i + 1] = challenge[1];
138         chal[i + 2] = challenge[2];
139         chal[i + 3] = challenge[3];
140     }
141
142     for (int i = 0; i < iodine_pass.Length; i++)
143         pass[i] = (byte)iodine_pass[i];
144
145     // (32 bytes password) XOR (32 bytes login challenge)
146     for (int i = 0; i < 32; i++)
147         temp[i] = (byte)(chal[i] ^ pass[i]);
148
149     hash = md.ComputeHash(temp);                // Create it's hash
150     sdata[0] = userid;                          // byte 0 - userid
151     Array.Copy(hash, 0, sdata, 1, hash.Length); // byte 1 to 16 - hash
152     sdata[17] = (byte)(rand_seed >> 8);         // byte 17 to 18 - CMC
153     sdata[18] = (byte)(rand_seed >> 0);
154
155     rand_seed++;
156     string base32Str = 'L' + Base32.Encoder(sdata); // Encode sdata with base32
157     byte[] hostname = Dns.make_hostname(base32Str); // Build hostname
158     byte[] rdata = Dns.send_query(hostname);        // send it
159
160     if (rdata != null) {
161         string rdataStr = Encoding.Default.GetString(rdata, 0, rdata.Length);
162         if (!rdataStr.StartsWith("LNAK")) {
163             try {
164                 login_response = rdataStr.Split(new char[] { '-' });
165                 return true;
166             }
167             catch (Exception) {}
168         }
169         else if (rdataStr.StartsWith("LNAK")) GraphicalInterface.getInstance().MsgLogger("Login failed !");
170         else GraphicalInterface.getInstance().MsgLogger("Handshake error.");
171     }
172     return false;
173 }
```

Εικόνα 72. Code snippet 13

```
217 public static bool set_media_status(int value)
218 {
219     IntPtr status = Marshal.AllocHGlobal(4);
220     Marshal.WriteInt32(status, value);
221
222     if (Coredll.DeviceIoControl(handle,
223         Driver.TAP_CONTROL_CODE(6, 0),
224         status,
225         4,
226         IntPtr.Zero,
227         0,
228         0, IntPtr.Zero) != 0)
229     {
230         return true;
231     }
232     else
233     {
234         GraphicalInterface.getInstance().MsgLogger("MediaStatus: DeviceIoControl Error");
235         return false;
236     }
237 }
```

Εικόνα 73. Code snippet 14

```
217 private static void Send_P(object source, OpenNETCF.Timers.ElapsedEventArgs e)
218 {
219     byte[] sdata = new byte[3];
220
221     sdata[0] = userid;
222     sdata[1] = (byte)(rand_seed >> 8);
223     sdata[2] = (byte)(rand_seed >> 0);
224
225     // Encode it
226     string base32Str = 'P' + Base32.Encoder(sdata);
227
228     // Make it a valid hostname
229     byte[] hostname = Dns.make_hostname(base32Str);
230
231     // For the next P
232     rand_seed++;
233
234     P_timer.Stop();     // Send P packets one by one.
235
236     byte[] rdata = Dns.send_query(hostname);
237
238     if (rdata != null)
239     {
240         byte[] data = Zlib.decompress(rdata);
241         TapAdapter.WriteTun(data);
242     }
243
244     P_timer.Start();  // Send P packets one by one.
245 }
```

Εικόνα 74. Code snippet 15

```
171 public static bool set_dhcp_masq(string local_ip, string remote_netmask)
172 {
173     // local ip
174     int IpAddress = BitConverter.ToInt32(IPAddress.Parse(local_ip).GetAddressBytes(), 0);
175
176     // remote netmask
177     int RemNetMask = BitConverter.ToInt32(IPAddress.Parse(remote_netmask).GetAddressBytes(), 0);
178
179     // 1 day
180     int leasetime = 86400;
181
182     // For DHCP_MASQ
183     IntPtr dhcp_masq = Marshal.AllocHGlobal(16);
184     Marshal.WriteInt32(dhcp_masq, 0, IpAddress);
185     Marshal.WriteInt32(dhcp_masq, 4, RemNetMask);
186     Marshal.WriteInt32(dhcp_masq, 8, IpAddress);
187     Marshal.WriteInt32(dhcp_masq, 12, leasetime);
188
189     if (Coredll.DeviceIoControl(handle,
190         Driver.TAP_CONTROL_CODE(7, 0),
191         dhcp_masq,
192         16,
193         dhcp_masq,
194         16,
195         0,
196         IntPtr.Zero) != 0)
197     {
198         GraphicalInterface.getInstance().MsgLogger("DHCP masq enabled.");
199         return true;
200     }
201     else
202     {
203         GraphicalInterface.getInstance().MsgLogger("Could not enable DHCP masq.");
204         return false;
205     }
206 }
```

Εικόνα 75. Code snippet 16

Κεφάλαιο ένατο

Παράρτημα Β: Παρουσίαση της εφαρμογής

9.1. Λίγα λόγια για την εφαρμογή

Για την δοκιμή της εφαρμογής σε PDA δαπανήθηκαν 150 ευρώ (ιδία χρηματοδότηση) και δουλεύει με Windows Mobile 5 και 6.1. Μπορεί να ανιχνεύσει Wi-Fi και GPRS συσκευές και να προσαρμόσει κατάλληλα το route table της συσκευής. Η ιστοσελίδα του project είναι www.it.teithe.gr/~kontam/wiod/ και εκεί βρίσκονται διαθέσιμες οι εκδόσεις και για mobile και για desktop καθώς και ότι οδηγία είναι απαραίτητη. Η εφαρμογή προγραμματίστηκε σε C# με το περιβάλλον Microsoft Visual Studio 2008 Professional το οποίο αποκτήθηκε δωρεάν από το Microsoft academic license software center του Τμήματος Πληροφορικής ΑΤΕΙΘ που παρέχει στους φοιτητές του τμήματος δωρεάν λογισμικό από την Microsoft. Οι περισσότερες εικόνες δημιουργήθηκαν εξ' ολοκλήρου στο Paint.NET. Το κείμενο συντάχθηκε σε MS Word '07 και μορφοποιήθηκε σύμφωνα με τον νέο κανονισμό πτυχιακών εργασιών που ανακοινώθηκε την Κυριακή 8.3.2009 στη ιστοσελίδα των εκπροσώπων. Η εφαρμογή έγινε δεκτή και από τους Σουηδούς προγραμματιστές του Iodine ως συμπληρωματική για πλατφόρμες Windows και την καταχώρησαν υπό το όνομα μου στο Wiki http://dev.kryo.se/iodine/ .

Ο κώδικας για την Mobile έκδοση είναι περίπου 3.700 σειρές ενώ για την Desktop περίπου 2.500 σειρές. Στην Mobile έκδοση γίνεται χρήση δύο εξωτερικών βιβλιοθηκών μια για την zlib συμπίεση/αποσυμπίεση και μια για την διαχείριση των στοιχείων του Wi-Fi και 3G adapter. Η 1η βιβλιοθήκη πάρθηκε από την ιστοσελίδα www.icsharpcode.net . Διανέμεται στο κοινό δωρεάν και υπόκειται στην άδεια GPL. Η 2η βιβλιοθήκη πάρθηκε από την ιστοσελίδα www.opennetcf.com και μας παρέχει ένα σύνολο κλάσεων που είναι διαθέσιμα στο .NET Framework αλλά όχι στο .NET Compact Framework. Εκτός αυτής της γεφύρωσης του κενού μεταξύ των δύο Framework προσφέρει στους προγραμματιστές και ένα σύνολο wrapper κλάσεων που απλοποιούν τον προγραμματισμό και μειώνουν το χρόνο που θα δαπανήσουν. Διανέμεται στο κοινό δωρεάν ως πακέτο με όνομα SDF Community Edition και υπόκειται στην άδεια OpenNETCF Shared Source License (http://www.opennetcf.com/sharedsourcelicense.ocf) .

Για να λειτουργήσει όλο το εγχείρημα πρέπει να έχουμε υπό τον έλεγχο μας ένα domain. Επίσης θα πρέπει να κάνουμε ρυθμίσεις στο DNS server που διαχειρίζεται το domain μας και να έχουμε άλλο ένα μηχάνημα συνδεδεμένο στο διαδίκτυο με δημόσια διεύθυνση IP στο οποίο θα είναι φορτωμένος ο Iodined server. Έστω ότι έχουμε το domain vlsi.gr. Θα πρέπει να δημιουργήσουμε την παρακάτω καταχώρηση στο zone file του DNS server που διαχειρίζεται το domain αυτό.

tunnel.vlsi.gr. IN NS iodineServer.gr

Όπου tunnel.vlsi.gr είναι το όνομα που θα χρησιμοποιήσουμε για να δρομολογήσουμε τα μηνύματα DNS στον rogue εξυπηρετητή ονομάτων iodineServer.gr .

9.2. Εγκατάσταση εφαρμογής

Απαιτήσεις της client εφαρμογής *wmiod* για windows mobile είναι:

OpenVPN TAP Driver από www.it.teithe.gr/~kontam/wiod/ovpnppc.en.arm.cab
.NET CF 2.0 από www.it.teithe.gr/~kontam/wiod/NETCFv2.wm.armv4i.cab

ενώ απαιτήσεις της client εφαρμογής *wiod* για windows είναι:

OpenVPN TAP Driver από www.it.teithe.gr/~kontam/wiod/Tap32_Driver_V9.zip
.NET Framework 2.0 από την ιστοσελίδα της Microsoft

Για server χρησιμοποιούμε τον iodine-0.4.2 σε Linux που είναι διαθέσιμος στην ιστοσελίδα http://code.kryo.se/iodine/ . Απαιτήσεις του είναι:

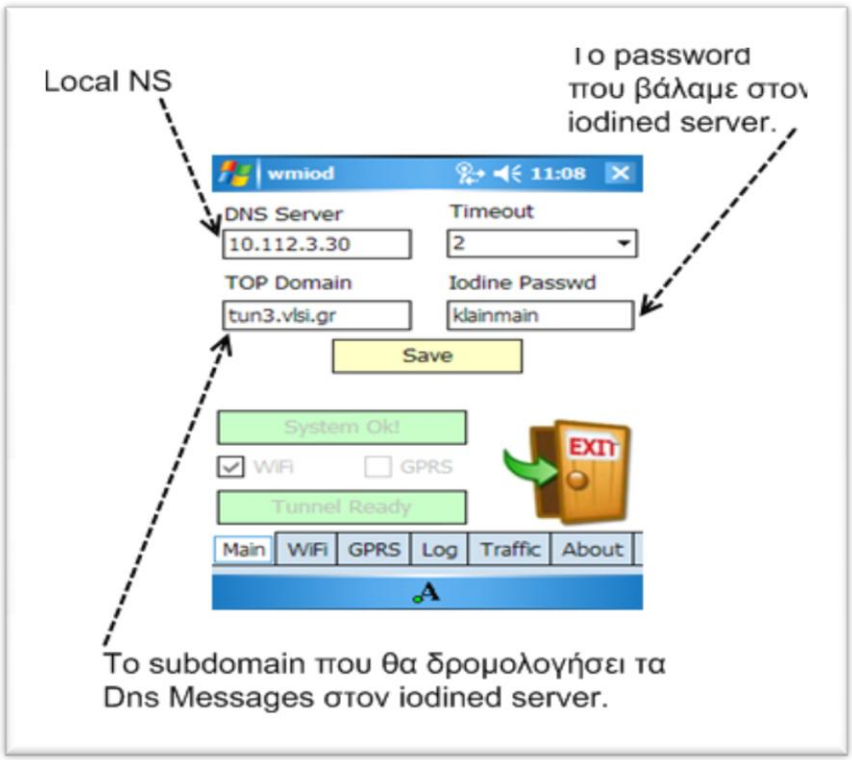
- Το tun device στο path /dev/tun. Το φορτώνουμε με **modprobe tun**.
- Τα iptables. Με αυτά θα εφαρμόσουμε **masquerading** έτσι ώστε ότι έρχεται από το private δίκτυο του τούνελ και έχει προορισμό το διαδίκτυο να μπορεί να δρομολογηθεί. Παρακάτω δίνονται ακριβώς οι εντολές που χρειάζεται.

iptables -t nat -A POSTROUTING -s 172.16.x.x/16 -o eth0 -j MASQUERADE
echo "1" >> /proc/sys/net/ipv4/ip_forward

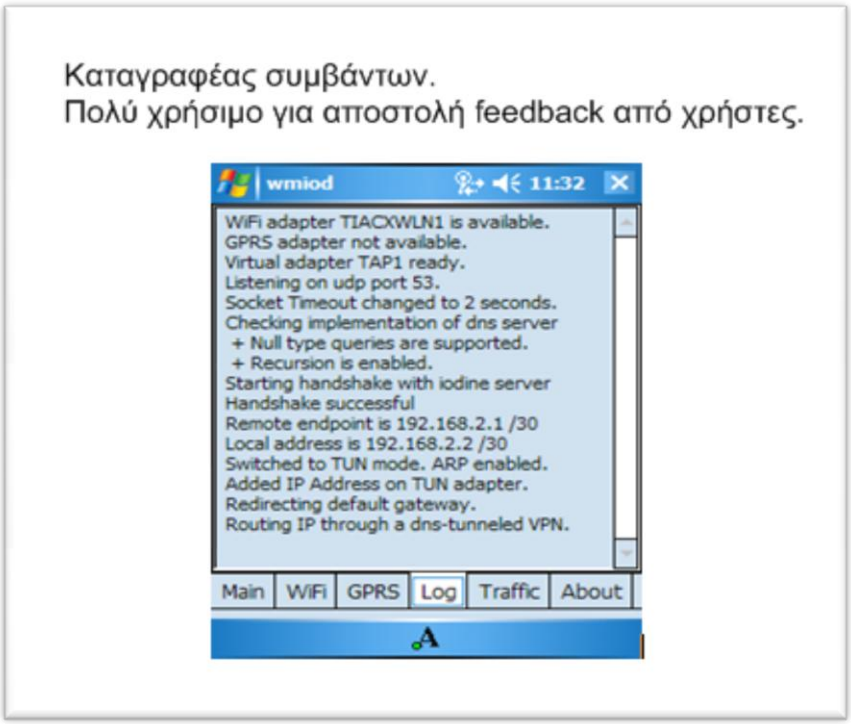
όπου 172.16.x.x/16 το private δίκτυο του τούνελ.

9.3. Επεξήγηση των μενού

Τα μενού είναι ίδια και για τις δύο εκδόσεις της εφαρμογής. Εμείς θα δείξουμε μόνο αυτά της Window Mobile.

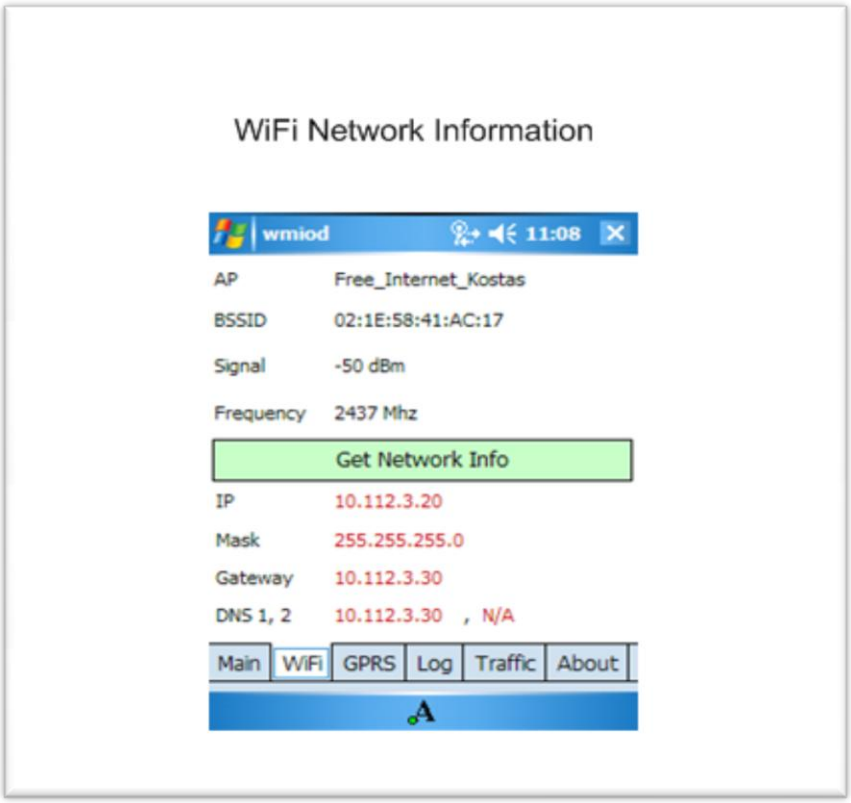


Εικόνα 76. Main Form



Εικόνα 77. Log Form

Εικόνα 78. Traffic Form



Εικόνα 79. Wi-Fi Form

Εικόνα 80. About Form

Αναφορές

Iodine ver0.4.2 Protocol (http://svn.kryo.se/iodine/doc/proto_00000402.txt)

- RFCs

RFC 920 - Domain Requirements (http://tools.ietf.org/html/rfc920)

RFC 1034 - Domain names - concepts and facilities (http://www.faqs.org/rfcs/rfc1034.html)

RFC 1035 - Domain names – implementation (http://www.faqs.org/rfcs/rfc1035.html)

RFC 4648 - The Base16, Base32, and Base64 Data Encodings (http://tools.ietf.org/html/rfc4648)

RFC 3696 - Application Techniques for Checking Names (http://tools.ietf.org/html/rfc3696)

RFC 3467 - Role of the Domain Name System (http://tools.ietf.org/html/rfc3467)

RFC 2671 - Extension Mechanisms for DNS (http://tools.ietf.org/html/rfc2671)

- MSDN

MSDN: CreateFile (http://msdn.microsoft.com/en-us/library/aa363858(VS.85).aspx)

MSDN: CloseHandle (http://msdn.microsoft.com/en-us/library/ms724211(VS.85).aspx)

MSDN: DeviceIoControl (http://msdn.microsoft.com/en-us/library/aa363216(VS.85).aspx)

MSDN: ReadFile (http://msdn.microsoft.com/en-us/library/aa365467(VS.85).aspx)

MSDN: WriteFile (http://msdn.microsoft.com/en-us/library/aa365747(VS.85).aspx)

MSDN: CreateIpForwardEntry (http://msdn.microsoft.com/en-us/library/aa365860(VS.85).aspx)

MSDN: DeleteIpForwardEntry (http://msdn.microsoft.com/en-us/library/365878(VS.85).aspx)

MSDN: AddIPAddress (http://msdn.microsoft.com/en-us/library/aa365801(VS.85).aspx)

MSDN: DeleteIPAddress (http://msdn.microsoft.com/en-us/library/aa365875(VS.85).aspx)

Thread-Safe Calls to Controls (http://msdn.microsoft.com/en-us/library/ms171728.aspx)