ΑΝΩΤΑΤΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ
ΙΔΡΥΜΑ ΛΑΡΙΣΑΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

ΤΜΗΜΑ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

“ Υλοποίηση ενός open source προγράμματος παρακολούθησης κίνησης δικτύου (network packet sniffer) σε γλώσσα python, με δυνατότητες παρακολούθησης κίνησης και σε δίκτυα συνδεδεμένα με switch ”

Χατζόπουλος Δήμος Τ-1085
dimosch@linuxteam.cs.teilar.gr

ΕΠΙΒΛΕΠΩΝ: Σωμαράς Χρήστος

ΛΑΡΙΣΑ 2010

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

Λάρισα, …/…../2010

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1.

2.

3.

Περίληψη

Ένα network packet sniffer είναι λογισμικό με δυνατότητες παρακολούθησης και καταγραφής της κίνησης ή αλλιώς των πακέτων δικτύου που αποστέλλει και δέχεται η διεπαφή δικτύου (network interface) του Η/Υ.

Σε δίκτυα συνδεδεμένα με διανομείς (network hubs), ένα τέτοιο λογισμικό μπορεί να καταγράψει και τα πακέτα που απευθύνονται σε άλλους Η/Υ ή συσκευές του δικτύου. Αντίθετα σε δίκτυα συνδεδεμένα με μεταγωγείς (network switch) που παρέχουν επιπλέον ασφάλεια, αυτό είναι εφικτό μόνο σε συγκεκριμένες περιπτώσεις και λόγω κάποιων αδυναμιών ασφαλείας των πρωτοκόλλων που χρησιμοποιούνται.

Στόχος της παρούσας πτυχιακής εργασίας είναι η υλοποίηση ενός τέτοιου λογισμικού στην γλώσσα προγραμματισμού Python και η εκμετάλλευση των κενών στην ασφάλεια του πρωτοκόλλου ARP σε τοπικά δίκτυα που χρησιμοποιούν τα πρωτόκολλα IPv4 και Ethernet, για την επίτευξη του επιθυμητού αποτελέσματος.

Τέλος η επιλογή να γίνει open source το τελικό πρόγραμμά δεν έχει να κάνει με την υλοποίηση του σε πρώτη φάση, αλλά με την μετέπειτα εξέλιξη του σε open source project εάν υπάρξει ενδιαφέρον από άτομα στην κοινότητα του ελεύθερου λογισμικού, όπως και με την ελεύθερη διακίνηση και διάδοση της γνώσης.

...Θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου, κ. Σωμαρά Χρήστο, για την παροχή υλικού μελέτης, την καθοδήγησή του και την παροχή άμεσης βοήθειας σε όλα τα στάδια της υλοποίησης της εργασίας, όπως επίσης και τον φίλο και συνάδελφο στο LinuxTeam του ΤΕΙ Λάρισας, Κεφαλλωνίτη Φοίβο για την βοήθεια του στα low-level δικτυακά θέματα, αλλά και όλα τα μέλη του LinuxTeam του ΤΕΙ Λάρισας για την συνεχή ανταλλαγή γνώσεων, απόψεων και συμβουλών καθ' όλη την διάρκεια των τελευταίων ετών...

Ευχαριστώ θερμά

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Κεφάλαιο 1: Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ PYTHON 1

1.1 Εισαγωγή στην Python 2

1.2 Το διαδραστικό κέλυφος της Python 2

1.3 Η βασική λειτουργία της Python 3

1.3.1 Αριθμοί και εκφράσεις 3

1.3.2 Μεταβλητές 4

1.3.3 Εντολές 4

1.3.4 Συναρτήσεις 5

1.3.5 Modules 5

1.3.6 Αποθήκευση και εκτέλεση των προγραμμάτων 6

1.3.7 Strings 7

1.4 Λίστες και tuples 8

1.4.1 Συνηθισμένες λειτουργίες των ακολουθιών 8

1.4.2 Λίστες 11

1.4.3 Tuples 15

1.5 Λεξικά 16

1.5.1 Βασικές λειτουργίες των λεξικών 17

1.5.2 Μέθοδοι λεξικών 18

1.6 Συνθήκες και βρόχοι 20

1.6.1 Στοιχειοθέτηση και Blocks κωδικά 20

1.6.2 Εκτέλεση υπό συνθήκες και η εντολή if 20

1.6.3 Βρόχοι 21

1.7 Δημιουργία συναρτήσεων 22

1.8 Δημιουργία τάξεων 23

1.9 Εξαιρέσεις 23

Κεφάλαιο 2: NETWORK PACKET SNIFFING 25

2.1 Γενικά 26

2.2 Δίκτυα συνδεδεμένα με network hub 26

2.2.1 Η λειτουργία του network hub 26

2.2.2 Η κάρτα δικτύου (promiscuous mode) 26

2.2.3 Παρακολούθηση της κίνησης μεταξύ τρίτων κόμβων 27

2.3 Δίκτυα συνδεδεμένα με network switch 27

2.3.1 Η λειτουργία του network switch 27

2.3.2 Παρακολούθηση της κίνησης μεταξύ τρίτων κόμβων 28

Κεφάλαιο 3: Η ΕΠΙΘΕΣΗ MITM ΜΕ ARP CACHE POISONING 29

3.1 Η επίθεση τύπου MITM 30

3.2 Το πρωτόκολλο ARP 31

3.2.1 Γενικά 31

3.2.2 Η βασική λειτουργία 32

3.3 Η επίθεση ARP cache poisoning 35

3.3.1 Λειτουργία 35

3.3.2 Περιορισμοί και προϋποθέσεις 36

3.3.3 Άλλες χρήσεις 37

Κεφάλαιο 4: SOFTWARE ΚΑΙ ΕΡΓΑΛΕΙΑ 38

4.1 Python modules που χρησιμοποιήθηκαν 39

4.1.1 Modules της βασικής βιβλιοθήκης 39

4.1.2 Third-Party Modules 40

4.2 Ο επεξεργαστής κειμένου Nano 42

4.3 Το Wireshark 43

Κεφάλαιο 5: Η ΕΦΑΡΜΟΓΗ ΚΑΙ ΟΙ ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ 45

5.1 Δομή 46

5.1.1 Το module functs.py 46

5.1.2 Το module attacks.py 47

5.1.3 Το αρχείο sniffer.py 47

5.2 Λειτουργία 47

5.2.1 Εξαρτήσεις 47

5.2.2 Εκτέλεση 48

Κεφάλαιο 6: ΠΡΟΣΤΑΣΙΑ ΚΑΤΑ ΤΗΣ ΚΑΚΟΒΟΥΛΗΣ ΧΡΗΣΗΣ 54

6.1 Προστασία από την επίθεση ARP cache poisoning 55

6.2 Προστασία από το network packet sniffing 56

Κεφάλαιο 7: ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΕΡΑΙΤΕΡΩ ΕΞΕΛΙΞΗ 57

7.1 Συμπεράσματα 58

7.2 Περαιτέρω εξέλιξη 58

Βιβλιογραφία 59

Κεφάλαιο 1:

Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ PYTHON

1.1 Εισαγωγή στην Python

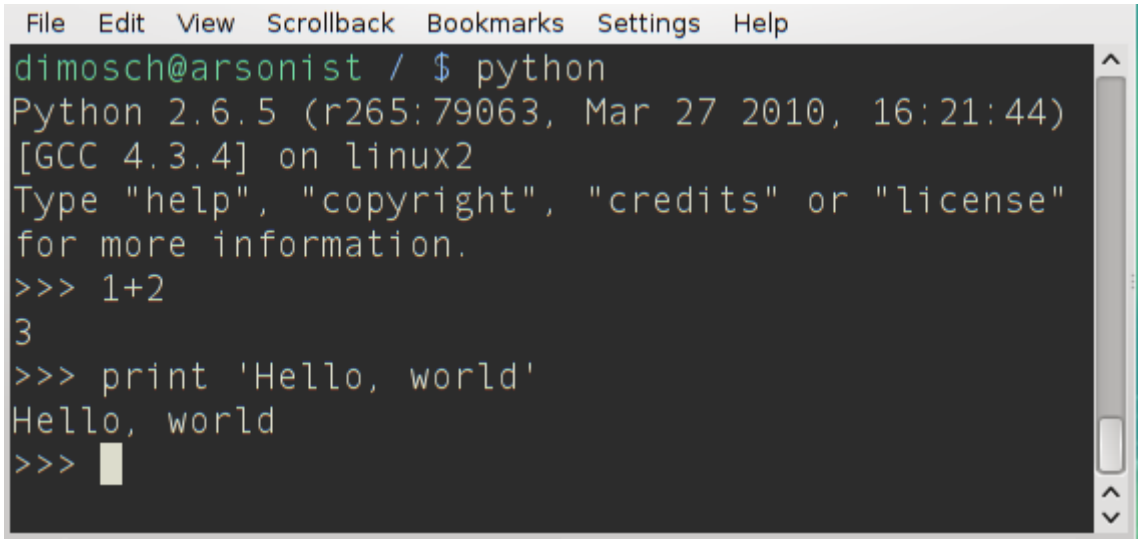
Η Python είναι μια διερμηνευόμενη (interpreted) γλώσσα προγραμματισμού υψηλού επιπέδου, που σχεδιάστηκε με έμφαση στην εύκολη ανάγνωση του κώδικα της. Η ανάπτυξή της ξεκίνησε το Δεκέμβριο του 1989 από τον Guido van Rossum στο Εθνικό Ινστιτούτο Έρευνας Πληροφορικής και Μαθηματικών της Ολλανδίας (CWI). Ο δημιουργός της εμπνεύστηκε το όνομα της από τούς Monty Pythons, το γνωστό γκρουπ κωμικών από την Μεγάλη Βρετανία.

Υποστηρίζει πολλά είδη προγραμματισμού, όπως αντικειμενοστραφή, διαδικαστικό, συναρτησιακό και άλλα, και παρέχει ένα πλήρως δυναμικό σύστημα τύπων και αυτόματη διαχείριση μνήμης. Είναι μια γλώσσα προγραμματισμού γενικής χρήσης, ιδανική για κατασκευή εφαρμογών όλων των ειδών ακόμα και δυναμικών ιστοσελίδων. Η βασική της βιβλιοθήκη είναι αρκετά μεγάλη και υπάρχει εκτενής τεκμηρίωση.

Η ανάπτυξή της ακολουθεί ένα ανοικτό πρότυπο που βασίζεται στην κοινότητα των χρηστών και συντονίζεται από το Python Software Foundation. Είναι λογισμικό ανοιχτού κώδικα (open source) και εκδίδεται υπό την άδεια PSFL. Οι δύο τρέχουσες εκδόσεις της είναι η 2.6 και η 3, έχουν αρκετές διαφορές μεταξύ τους και αναπτύσσονται παράλληλα. Επίσης η έκδοση 3 δεν είναι συμβατή προς τα πίσω με τις παλαιότερες εκδόσεις καθώς έχει αλλάξει η σύνταξη και η λειτουργία πολλών εντολών και εκφράσεων. Τέλος, η Python υποστηρίζει διάφορες αρχιτεκτονικές και λειτουργικά συστήματα (Windows, Linux, διάφορα συστήματα Unix).

1.2 Το διαδραστικό κέλυφος της Python

Ένα πολύ χρήσιμο χαρακτηριστικό της Python, είναι ότι ο διερμηνέας της, είναι ταυτόχρονα και ένα διαδραστικό κέλυφος (interactive shell), κάτι σαν μια γραμμή εντολών για την Python, στην οποία μπορούμε να εισάγουμε εντολές και να δούμε απευθείας τα αποτελέσματα. Εάν εκτελέσουμε στο τερματικό του λειτουργικού μας συστήματος την εντολή **python** χωρίς καμία παράμετρο, στην οθόνη μας εμφανίζεται το διαδραστικό κέλυφος του διερμηνέα, το οποίο μας παροτρύνει να εισάγουμε κάποια εντολή. Αυτό μπορεί να αποδειχθεί εξαιρετικά χρήσιμο σε ορισμένες περιπτώσεις, για παράδειγμα μπορούμε να εκτελέσουμε μια συνάρτηση του προγράμματος μας μεμονωμένα χωρίς να χρειαστεί να εκτελέσουμε ολόκληρο το πρόγραμμα.



***εικόνα 1.1**: Το διαδραστικό κέλυφος της Python σε λειτουργικό σύστημα Linux*

1.3 Η βασική λειτουργία της Python

1.3.1 Αριθμοί και εκφράσεις

Ο διαδραστικός διερμηνέας της Python μπορεί να χρησιμοποιηθεί και ως αριθμομηχανή, για παράδειγμα:

```
>>> 3 + 4
7
>>> 213123123 + 23123213
236246336
```

Οι περισσότεροι αριθμητικοί τελεστές λειτουργούν όπως είναι αναμενόμενο, με εξαίρεση την διαίρεση ακεραίων, η οποία σε εκδόσεις παλαιότερες της 3 κάνει στρογγυλοποίηση στο αποτέλεσμα ώστε αυτό να αποτελεί ακέραιο αριθμό:

```
>>> 1/2
0
>>> 7/2
3
```

Για να πάρουμε το αποτέλεσμα χωρίς στρογγυλοποίηση πρέπει να χρησιμοποιήσουμε πραγματικούς αριθμούς (floats) ως εξής:

```
>>> 1.0/2.0
0.5
```

Ένας ακόμα χρήσιμος τελεστής είναι ο % που μας δίνει το υπόλοιπο διαίρεσης:

```
>>> 9 % 4
1
```

Όπως επίσης και ο τελεστής δύναμης ** :

```
>>> 3 ** 2
9
```

Αξίζει ακόμα να σημειωθεί ότι ο διερμηνέας τηρεί την προτεραιότητα πράξεων, δηλαδή οι παραστάσεις 4+8*7 και (4+8)*7 θα μας δώσουν διαφορετικό αποτέλεσμα εάν τις εκτελέσουμε.

1.3.2 Μεταβλητές

Μια μεταβλητή είναι στην ουσία ένα όνομα που αναπαριστά μια τιμή. Εάν για παράδειγμα θέλουμε η μεταβλητή y να αναπαριστά την τιμή 7 θα πρέπει να της αναθέσουμε την τιμή αυτή ως εξής:

```
>>>y = 7
```

Αυτή η διαδικασία λέγεται ανάθεση τιμής και έπειτα μπορούμε να χρησιμοποιήσουμε την μεταβλητή y σε εκφράσεις:

```
>>>y*3
21
```

1.3.3 Εντολές

Με τις εντολές, σε αντίθεση με τις εκφράσεις, μπορούμε να πούμε στον διερμηνέα να κάνει κάτι, για παράδειγμα με την εντολή print μπορούμε να τυπώσουμε στην οθόνη:

```
>>>print "Hello, world"
Hello, world
```

1.3.4 Συναρτήσεις

Οι συναρτήσεις είναι μικρά υποπρογράμματα που μπορούμε να καλέσουμε για να εκτελέσουμε μια εργασία. Για παράδειγμα με την συνάρτηση pow() μπορούμε να υψώσουμε έναν αριθμό σε μία δύναμη:

```
>>> pow(2,3)
8
```

Όταν χρησιμοποιούμε μια συνάρτηση λέμε ότι την "καλούμε" ενώ οι τιμές που εσωκλείονται στην παρένθεση ονομάζονται παράμετροι της συνάρτησης. Οι συναρτήσεις μπορούν να χρησιμοποιηθούν σε εκφράσεις. Η Python διαθέτει αρκετές ενσωματωμένες συναρτήσεις και φυσικά ο χρήστης μπορεί να ορίσει τις δικές του.

Μια αρκετά χρήσιμη συνάρτηση είναι η input() με την οποία μπορούμε να διαβάσουμε μια τιμή που εισάγει ο χρήστης από το πληκτρολόγιο και να την αναθέσουμε σε κάποια μεταβλητή ή να την διαχειριστούμε όπως επιθυμούμε:

```
>>> x = input("x:")
x:14
>>> y = input("y:")
y:6
>>> print x+y
20
```

1.3.5 Modules

Τα modules είναι κάτι σαν επεκτάσεις που μπορούμε να εισάγουμε στην Python για να επεκτείνουμε της δυνατότητές της. Η βασική βιβλιοθήκη της Python διαθέτει πάρα πολλά για κάθε είδους λειτουργία και υπάρχουν ακόμα περισσότερα third-party modules που μπορούμε να εγκαταστήσουμε από διάφορες on-line πηγές. Για παράδειγμα το module math της βασικής βιβλιοθήκης διαθέτει πολλές χρήσιμες συναρτήσεις όπως την floor η οποία επιστρέφει την μικρότερη δυνατή άρτια τιμή η οποία είναι μικρότερη ή ίση με τον αριθμό που εισάγουμε ως παράμετρο της συνάρτησης. Για να εισάγουμε ένα module στην Python χρησιμοποιούμε την εντολή import και στην συνέχεια καλούμε την συνάρτηση ως εξής:

```
>>> import math
>>> math.floor(22.9)
22.0
```

Εάν θέλουμε να χρησιμοποιήσουμε μόνο μία συνάρτηση από ένα module, και δεν θέλουμε κάθε φορά να την καλούμε γράφοντας και το όνομα του module, μπορούμε να την εισάγουμε με έναν λίγο διαφορετικό τρόπο, και στην συνέχεια να την καλούμε χρησιμοποιώντας μόνο το όνομά της ως εξής:

```
>>> from math import floor
>>> floor(22.9)
22.0
```

1.3.6 Αποθήκευση και εκτέλεση των προγραμμάτων

Για να δημιουργήσουμε ένα πρόγραμμα στην Python, το οποίο θα παραμείνει στον Η/Υ μας, σε αντίθεση με όσα εισάγουμε στον διαδραστικό διερμηνέα τα οποία χάνονται μόλις τον κλείσουμε, αρκεί να γράψουμε της εντολές που θέλουμε σε κάποιον επεξεργαστή κειμένου και στην συνέχεια να τα αποθηκεύσουμε σε ένα αρχείο.

Εάν για παράδειγμα τοποθετήσουμε την εντολή print 'Hello, teilar' στο αρχείο teilar.py, τότε έχουμε ένα πρόγραμμα σε Python. Για να το εκτελέσουμε αρκεί να γράψουμε στο τερματικό του λειτουργικού μας συστήματος την εντολή python ακολουθούμενη από το όνομα του αρχείο:

```
$ python teilar.py
Hello, teilar
$
```

Όπως και σε άλλες γλώσσες προγραμματισμού, όταν γράφουμε μεγάλα προγράμματα χρειάζεται να προσθέσουμε κάποια σχόλια για να είναι πιο κατανοητός ο κώδικάς μας. Για να βάλουμε ένα σχόλιο στα προγράμματα python που γράφουμε αρκεί μπροστά από το σχόλιο να βάλουμε το σύμβολο της δίεσης #. Έτσι ο διερμηνέας κατά την εκτέλεση του προγράμματος θα αγνοήσει όλους του χαρακτήρες που βρίσκονται μετά από την δίεση.

Σε περιβάλλοντα UNIX, μπορούμε να κάνουμε το πρόγραμμά μας να συμπεριφέρεται σαν οποιοδήποτε κανονικό πρόγραμμα κατά την εκτέλεση του, και να μην χρειάζεται να δώσουμε την εντολή python πριν από το όνομα του για να εκτελεστεί. Αυτό γίνεται προσθέτοντας στην πρώτη γραμμή του προγράμματός μας τα σύμβολα της δίεσης και του θαυμαστικού ακολουθούμενα από την διαδρομή όπου βρίσκεται ο διερμηνέας της Python, δηλαδή #!/usr/bin/python , και στην συνέχεια δίνοντας δικαιώματα εκτέλεσης στο αρχείο μας. Έτσι μπορούμε να εκτελέσουμε για παράδειγμα το αρχείο hello.py που δημιουργήσαμε πριν απλά πληκτρολογώντας το όνομά του, δεδομένου ότι ο τρέχον φάκελος είναι αυτός που το περιέχει:

```
$ teilar.py
Hello, teilar
$
```

1.3.7 Strings

Η Python υποστηρίζει πολλούς τύπους δεδομένων, όπως int, float, long int, list κ.α.. Ένας από του βασικότερους όμως που συναντούμε σχεδόν σε κάθε πρόγραμμα γραμμένο σε Python είναι τα strings. Η κύρια χρήσης των strings είναι η αναπαράσταση κειμένου και αποτελούν τιμές όπως και οι ακέραιοι (int). Τα strings πρέπει να εσωκλείονται σε διπλά ή μονά εισαγωγικά, για παράδειγμα 'Hello, teilar' ή “Hello, teilar”. Στην περίπτωση που ένα string περιέχει εισαγωγικά, τα οποία όμως χρησιμοποιούμε για να ορίσουμε που αρχίζει και που τελειώνει το string, πρέπει να χρησιμοποιήσουμε πριν από αυτά την ανάποδη πλαγιοκάθετο για να “καταλάβει” ο διερμηνέας ότι πρέπει να το τυπώσει, δηλαδή:

```
>>> 'let\'s go'
"let's go"
```

Σε αντίθετη περίπτωση ο διερμηνέας θα μας τυπώσει ένα σφάλμα:

```
>>> 'let's go'
  File "<stdin>", line 1
    'let's go'
        ^
```

1.4 Λίστες και tuples

Η Python υποστηρίζει αρκετές δομές δεδομένων, οι οποίες είναι συλλογές στοιχείων δεδομένων. Η πιο βασική από αυτές είναι η ακολουθία (sequence). Σε κάθε στοιχείο της ακολουθίας ανατίθεται ένας αριθμός-ευρετηρίου (index). Η αρίθμηση του ευρετηρίου ξεκινάει από το μηδέν, το δεύτερο στοιχείο έχει αριθμό ένα και ούτω καθεξής. Επίσης είναι δυνατή η αρίθμηση ξεκινώντας από πίσω, με το τελευταίο στοιχείο να έχει αριθμό ευρετηρίου -1, το επόμενο -2 κλπ.

Υπάρχουν έξι ενσωματωμένοι τύποι ακολουθιών με κυριότερους από αυτούς τις λίστες (lists) και τα tuples. Η βασική διαφορά τους είναι ότι μπορούμε να μεταβάλουμε τα στοιχεία μια λίστας, να προσθέσουμε καινούρια ή να αφαιρέσουμε κάποια , αλλά δεν μπορούμε να μεταβάλουμε τα στοιχεία σε ένα tuple. Οι ακολουθίες είναι χρήσιμες όταν θέλουμε να δουλέψουμε με μια συλλογή δεδομένων. Τα στοιχεία μιας λίστας περικλείονται σε αγκύλες και χωρίζονται με κόμμα, για παράδειγμα:

```
>>> list = ['dimos', 24]
>>> print list
['dimos', 24]
```

Επίσης τα στοιχεία μια ακολουθίας μπορεί να είναι και τα ίδια ακολουθίες, πέρα από strings, integers, floats κτλπ.

1.4.1 Συνηθισμένες λειτουργίες των ακολουθιών

Υπάρχουν ορισμένες λειτουργίες που λειτουργούν σε όλων των ειδών τις ακολουθίες όπως: indexing, slicing, adding, multiplying και έλεγχος μέλους. Επιπλέον η Python διαθέτει αρκετές ενσωματωμένες συναρτήσεις για τον υπολογισμό του μήκους των ακολουθιών και για την εύρεση του μεγαλύτερου και του μικρότερου στοιχείου τους.

Indexing

Όπως είπαμε όλα τα στοιχεία κάθε ακολουθίας είναι αριθμημένα ξεκινώντας από το μηδέν. Έτσι μπορούμε να προσπελάσουμε κάθε στοιχείο ξεχωριστά ως εξής:

```
>>> var = 'hello'
>>> var[0]
'h'
>>> var[4]
'o'
>>> var[-1]
'o'
>>>
```

Τα strings αποτελούν και αυτά ακολουθίες και μπορούμε να προσπελάσουμε τα στοιχεία τους ακόμα και χωρίς την χρήση μεταβλητής:

```
>>> 'teilar'[-1]
'r'
>>>
```

Slicing

Με το slicing (τεμαχισμός) μπορούμε να προσπελάσουμε ένα εύρος στοιχείων μια ακολουθίας χρησιμοποιώντας δύο δείκτες χωρισμένους με άνω κάτω τελεία.

```
>>> numbers=[1,2,3,4,5,6,7,8,9,10]
>>> numbers[3:6]
[4, 5, 6]
>>>
```

Το slicing είναι πολύ χρήσιμο όταν θέλουμε να αποσπάσουμε συγκεκριμένα κομμάτια από μια ακολουθία. Ο πρώτος δείκτης είναι ο αριθμός ευρετηρίου του στοιχείου που θέλουμε να συμπεριλάβουμε πρώτο στον τεμαχισμό, ενώ ο δεύτερος είναι ο αριθμός ευρετηρίου του πρώτου στοιχείου αμέσως μετά τον τεμαχισμό. Τέλος μπορούμε να ορίσουμε ακόμα έναν

δείκτη ο οποίος αποτελεί το βήμα του τεμαχισμού:

```
>>> numbers[0:6]
[1, 2, 3, 4, 5, 6]
>>> numbers[0:6:2]
[1, 3, 5]
```

Adding

Με το adding (πρόσθεση) μπορούμε να συνενώσουμε ακολουθίες χρησιμοποιώντας τον τελεστή της πρόσθεσης +.

```
>>> [1,2,3]+[4,5,6]+[7,8,9]
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Multiplication

Με το multiplication (πολλαπλασιασμός) μπορούμε να δημιουργήσουμε μία νέα ακολουθία, στην οποία η ακολουθία που πολλαπλασιάζουμε με τον τελεστή * και μία τιμή x, επαναλαμβάνεται x φορές.

```
>>> 'teilar'*6
'teilarteilarteilarteilarteilarteilar'
>>>
```

Membership

Για να ελέγξουμε εάν μία τιμή υπάρχει σε μια ακολουθία, χρησιμοποιούμε τον τελεστή “in”. Ο τελεστής αυτός μας επιστρέφει True εάν η τιμή υπάρχει και False εάν δεν υπάρχει. Τέτοιου είδους τελεστές ονομάζονται boolen τελεστές:

```
>>> students = ['nikos', 'maria', 'dimitris']
>>> 'nikos' in students
True
>>> 'Petros' in students
False
```

Μήκος, Μέγιστο και Ελάχιστο

Οι ενσωματωμένες συναρτήσεις της Python len, max και min μπορεί να φανούν αρκετά χρήσιμες όταν δουλεύουμε με ακολουθίες. Η συνάρτηση len επιστρέφει τον αριθμό των στοιχείων που υπάρχουν σε μια ακολουθία, ενώ οι max και min μας επιστρέφουν το μεγαλύτερο και το ελάχιστο στοιχείο μιας ακολουθίας αντίστοιχα:

```
>>> numbers = [200, 567, 908]
>>> len(numbers)
3
>>> max(numbers)
908
>>> min(numbers)
200
>>> max(7,8)
8
>>> min (4,2,3,1,9)
1
>>>
```

1.4.2 Λίστες

Όπως είδαμε ένας πολύ χρήσιμος τύπος ακολουθίας της Python είναι οι λίστες και η βασική τους διαφορά σε σχέση με τα tuples και τα strings είναι ότι μπορούμε να μεταβάλλουμε τα περιεχόμενά τους. Υπάρχει ακόμα η δυνατότητα να δημιουργήσουμε μία λίστα από τα στοιχεία ενός string ώστε να είναι δυνατή η διαχείρισή τους, αυτό γίνεται με τον τύπο list() ως εξής:

```
>>> list('teilar')
['t', 'e', 'i', 'l', 'a', 'r']
>>>
```

Μεταβολή των στοιχειών μιας λίστας

Η μεταβολή ενός στοιχείου μιας λίστας είναι πολύ απλή και μοιάζει με την ανάθεση τιμής σε μεταβλητή, χρησιμοποιώντας όμως και τον δείκτη ευρετηρίου ώστε να δείξουμε ποιο στοιχείο θέλουμε να μεταβάλλουμε:

```
>>> x = [3, 3, 3,]
>>> x[1] = 4
>>> x
[3, 4, 3]
>>>
```

Επίσης μπορούμε να μεταβάλουμε ένα μόνο κομμάτι της λίστα χρησιμοποιώντας slicing:

```
>>> school = list('teilar')
>>> school
['t', 'e', 'i', 'l', 'a', 'r']
>>> school[3:] = list('ser')
>>> school
['t', 'e', 'i', 's', 'e', 'r']
>>>
```

Διαγραφή των στοιχείων μιας λίστας

Για να διαγράψουμε ένα στοιχείο μιας λίστας χρησιμοποιούμε την εντολή del και φυσικά τον δείκτη ευρετηρίου:

```
>>> x
[3, 4, 3]
>>> del x[1]
>>> x
[3, 3]
```

Μέθοδοι λιστών

Οι μέθοδοι μοιάζουν συναρτήσεις οι οποίες είναι στενά συνδεδεμένες με κάποια αντικείμενο (object), όπως ένα string, μια λίστα ή οποιοδήποτε άλλο. Οι μέθοδοι καλούνται όπως τις συναρτήσεις αλλά βάζοντας μπροστά το όνομα του αντικειμένου και τελεία:

object.method(argmunets)

Οι λίστες οι οποίες είναι αντικείμενα, έχουν αρκετές μεθόδους που μας επιτρέπουν να εξετάσουμε αλλά και να μεταβάλλουμε τα στοιχεία τους. Μερικές αρκετά χρήσιμες από αυτές είναι:

Η μέθοδος **append** χρησιμοποιείται για να προσθέσουμε ένα στοιχείο στο τέλος μιας λίστας:

```
>>> numbers = [1,2,3]
>>> numbers.append(4)
>>> numbers
[1, 2, 3, 4]
```

Η μέθοδος **count** μετράει πόσες φορές εμφανίζεται ένα στοιχείο σε μια λίστα:

```
>>> numbers = [1,2,3,1,4,5,1]
>>> numbers.count(1)
3
```

Η μέθοδος **extend** μας επιτρέπει να προσθέσουμε πολλά στοιχεία στο τέλος μιας λίστας, στην ουσία επεκτείνουμε μια λίστα με τα στοιχεία κάποιας άλλης.

```
>>> numbers = [1,2,3]
>>> a = [4,5,6]
>>> numbers.extend(a)
>>> numbers
[1, 2, 3, 4, 5, 6]
>>>
```

Η μέθοδος **index** αναζητεί το δείκτη ευρετηρίου της πρώτης εμφάνισης ενός στοιχείου

σε μια λίστα:

```
>>> users = ['nick', 'mary', 'john']
>>> users.index('mary')
1
```

Η μέθοδος **insert** χρησιμοποιείται για να προσθέσουμε ένα στοιχείο σε μια συγκεκριμένη θέση μιας λίστας:

```
>>> users = ['nick', 'mary', 'john']
>>> users.insert(1, 'peter')
>>> users
['nick', 'peter', 'mary', 'john']
```

Η μέθοδος **pop** αφαιρεί ένα στοιχείο από μια λίστα (από ορισμού το τελευταίο) και το επιστρέφει. Είναι η μόνη μέθοδος λίστας η οποία και μεταβάλει την λίστα και επιστρέφει μια τιμή:

```
>>> users
['nick', 'peter', 'mary', 'john']
>>> users.pop()
'john'
>>> users
['nick', 'peter', 'mary']
>>> users.pop(1)
'peter'
>>> users
['nick', 'mary']
```

Η μέθοδος **remove** χρησιμοποιείται για την αφαίρεση της πρώτης εμφάνισης ενός

αντικειμένου μιας λίστας:

```
>>> a = [2,4,5,2,7]
>>> a.remove(2)
>>> a
[4, 5, 2, 7]
```

Η μέθοδος **sort** ταξινομεί τα στοιχεία μιας λίστας, μεταβάλλοντας την ίδια την λίστα:

```
>>> a = [8, 3, 12, 1, 9, 25]
>>> a.sort()
>>> a
[1, 3, 8, 9, 12, 25]
>>>
```

1.4.3 Tuples

Τα tuples είναι ακολουθίες, όπως και οι λίστες, με την διαφορά ότι δεν μπορούμε να τις μεταβάλουμε. Η σύνταξη ενός tuple είναι πολύ απλή, αρκεί να χωρίσουμε μερικές τιμές με κόμμα και έχουμε ένα tuple. Προαιρετικά οι τιμές αυτές μπορούν να εσωκλείονται σε παρενθέσεις:

```
>>> 1,2,3
(1, 2, 3)
>>> (1,2,3)
(1, 2, 3)
```

Η ενσωματωμένη συνάρτηση tuple(), παίρνει ως παράμετρο μια ακολουθία και την μετατρέπει σε tuple. Εάν η ακολουθία είναι ήδη tuple επιστρέφεται ως έχει:

```
>>> tuple([1,2,3,4,5])
(1, 2, 3, 4, 5)
```

Τα tuples είναι απλές δομές δεδομένων και το μόνο που μπορούμε να κάνουμε είναι

να τις δημιουργήσουμε και να προσπελάσουμε τα στοιχεία τους, με τον ίδιο τρόπο που το κάνουμε σε άλλες ακολουθίες και ο βασικός λόγος ύπαρξής τους είναι ότι επιστρέφονται από ορισμένες ενσωματωμένες συναρτήσεις της Python.

1.5 Λεξικά

Τα λεξικά (dictionaries) είναι μια δομή δεδομένων της Python, στα στοιχεία της οποίας μπορούμε να αναφερόμαστε ονομαστικά. Τα στοιχεία ενός λεξικού δεν έχουν κάποια συγκεκριμένη σειρά, αλλά είναι αποθηκευμένα με βάση ένα κλειδί (όνομα), το οποίο μπορεί να είναι ένα string, ένας αριθμός ή ακόμα και ένα tuple.

Τα λεξικά απαρτίζονται από ζευγάρια κλειδιών και τιμών. Κάθε κλειδί χωρίζεται από την τιμή του με άνω κάτω τελεία, και όλα τα ζευγάρια εσωκλείονται σε αγκύλες {}. Στο παρακάτω παράδειγμα ενός τηλεφωνικού καταλόγου, τα ονόματα αποτελούν τα κλειδιά του λεξικού ενώ τα τηλέφωνα τις τιμές του:

```
>>> katalogos = {'nikos':'213123', 'maria':'234409', 'petros':'431214'}
>>> katalogos
{'petros': '431214', 'nikos': '213123', 'maria': '234409'}
```

Η αναζήτηση μιας τιμής γίνεται ως εξής:

```
>>> katalogos['maria']
'234409'
```

Για την κατασκευή ενός λεξικού από άλλες ακολουθίες (κλειδιών, τιμών) ή ακόμα και από άλλα λεξικά υπάρχει η ενσωματωμένη συνάρτηση dict():

```
>>> pairs = [('onoma', 'Nikos'), ('hlikia','56')]
>>> d = dict(pairs)
>>> d
{'hlikia': '56', 'onoma': 'Nikos'}
>>> d['onoma']
'Nikos'
```

Επίσης η συνάρτηση dict() μπορεί να χρησιμοποιηθεί απευθείας περνώντας της ως

παραμέτρους τα ζευγάρια κλειδιών-τιμών:

```
>>> d = dict(onoma='Nikos', hlikia='56')
>>> d
{'hlikia': '56', 'onoma': 'Nikos'}
```

1.5.1 Βασικές λειτουργίες των λεξικών

Η βασική λειτουργία των λεξικών σε πολλές περιπτώσεις είναι ίδια με αυτή των ακολουθιών:

- len(d) επιστρέφει τον αριθμό των ζευγαριών που υπάρχουν στο λεξικό d.
- d[k] επιστρέφει την τιμή που ανταποκρίνεται στο κλειδί k.
- d[k] = v συσχετισμός της τιμής v με το κλειδί k.
- del d[k] διαγραφή του ζευγαριού με κλειδί k.
- k in d ελεγχος εάν υπάρχει ζευγάρι στο λεξικό d με κλειδί k.

Παρόλο που οι λίστες και τα λεξικά έχουν αρκετά κοινά χαρακτηριστικά, έχουν και ορισμένες σημαντικές διαφορές:

Τύποι κλειδιών: Τα κλειδιά ενός λεξικού δεν χρειάζεται να είναι απαραιτήτως ακέραιοι αριθμοί. Μπορεί να είναι πραγματικοί αριθμοί, strings ή ακόμα και tuples.

Αυτόματη προσθήκη: Μπορούμε να αποδώσουμε μια τιμή σε ένα κλειδί ακόμα και αν αυτό το κλειδί δεν υπάρχει στο λεξικό, έτσι θα δημιουργηθεί αυτόματα ένα νέο ζευγάρι. Αντίθετα σε μια λίστα αυτό δεν είναι δυνατό και θα πρέπει να χρησιμοποιήσουμε την μέθοδο append.

Έλεγχος μέλους: Η έκφραση k in d (όπου d ένα λεξικό) ψάχνει για κλειδί και όχι για τιμή. Από την άλλη η έκφραση v in l (όπου l μια λίστα) ψάχνει για τιμή και όχι για αριθμό ευρετηρίου.

1.5.2 Μέθοδοι λεξικών

Τα λεξικά έχουν κάποιες χρήσιμες μεθόδους που αξίζει να αναφερθούν, καθώς καθιστούν την διαχείρισή τους αρκετά εύκολη:

Η μέθοδος **clear**, διαγράφει όλα τα στοιχεία ενός λεξικού και δεν επιστρέφει τίποτα:

```
>>> katalogos = {'nikos':'213123', 'maria':'234409', 'petros':'431214'}
>>> katalogos
{'petros': '431214', 'nikos': '213123', 'maria': '234409'}
>>> katalogos.clear()
>>> katalogos
{}
```

Η μέθοδος **copy** επιστρέφει ένα λεξικό με τα ίδια ζευγάρια κλειδιών-τιμών. Το λεξικό αυτό είναι ένα αντίγραφο του πρώτου άλλα οι τιμές του είναι οι ίδιες με το πρώτο και όχι αντίγραφα:

```
>>> katalogos = {'nikos':'213123', 'maria':'234409', 'petros':'431214'}
>>> y = katalogos.copy()
>>> y
{'maria': '234409', 'nikos': '213123', 'petros': '431214'}
```

Στο παραπάνω παράδειγμα, εάν αντικαταστήσουμε μια τιμή στο λεξικό y, στο λεξικό katalogos δεν γίνεται καμία αλλαγή. Εάν όμως επεξεργαστούμε μία τιμή στο y χωρίς να την αντικαταστήσουμε, η αλλαγή αυτή αντικατοπτρίζεται και στο λεξικό katalogos μιας και πρόκειται για την ίδια τιμή και όχι αντίγραφό της. Εάν θέλουμε να δημιουργήσουμε τελείως ανεξάρτητα αντίγραφα μπορούμε να χρησιμοποιήσουμε την συνάρτηση deepcopy, από το module copy.

Η μέθοδος **items** επιστρέφει όλα τα στοιχεία ενός λεξικού σε μια λίστα, στην οποία τα στοιχεία είναι της μορφής (κλειδί, τιμή). Τα στοιχεία δεν επιστρέφονται με κάποια συγκεκριμένη σειρά:

```
>>> katalogos = {'nikos':'213123', 'maria':'234409', 'petros':'431214'}
>>> katalogos.items()
[('petros', '431214'), ('nikos', '213123'), ('maria', '234409')]
```

Η μέθοδος **keys** επιστρέφει όλα τα κλειδιά ενός λεξικού σε μια λίστα:

```
>>> katalogos.keys()
['petros', 'nikos', 'maria']
```

Η μέθοδος **pop**, επιστρέφει την τιμή ενός συγκεκριμένου κλειδιού, και στην συνέχεια αφαιρεί ολόκληρο το ζευγάρι από την λίστα.

```
>>> katalogos.pop('nikos')
'213123'
>>> katalogos
{'petros': '431214', 'maria': '234409'}
```

Η μέθοδος **popitem** μοιάζει με την μέθοδο pop των λιστών, παρόλα αυτά επειδή τα λεξικά δεν έχουν τελευταίο αντικείμενο, η συνάρτηση επιστρέφει αυθαίρετα ένα τυχαίο στοιχείο του λεξικού και στην συνέχεια το διαγράφει:

```
>>> katalogos = {'nikos':'213123', 'maria':'234409', 'petros':'431214'}
>>> katalogos.popitem()
('petros', '431214')
>>> katalogos
{'nikos': '213123', 'maria': '234409'}
```

Η μέθοδος **update** ενημερώνει τα στοιχεία ενός λεξικού με τα στοιχεία ενός άλλου:

```
>>> katalogos = {'nikos':'213123', 'maria':'234409', 'petros':'431214'}
>>> x = {'giorgos':'098123', 'dimitra':'434353'}
```

```
>>> katalogos.update(x)
>>> katalogos
{'dimitra': '434353', 'petros': '431214', 'giorgos': '098123', 'nikos': '213123', 'maria': '234409'}
```

Η μέθοδος **values** επιστρέφει μια λίστα με όλες τις τιμές ενός λεξικού:

```
>>> katalogos.values()
['434353', '431214', '098123', '213123', '234409']
```

1.6 Συνθήκες και βρόχοι

1.6.1 Στοιχειοθέτηση και Blocks κωδικά

Τα blocks είναι σύνολα εντολών που εκτελούνται εάν μια συνθήκη είναι αληθής ή εκτελούνται πολλές φορές (βρόχοι). Σε αντίθεση με τις περισσότερες γλώσσες προγραμματισμού που χρησιμοποιούν κάποιους χαρακτήρες όπως αγκύλες "{}" για την οριοθέτηση των blocks, η Python χρησιμοποιεί την στοιχειοθέτηση (indentation), δηλαδή την προσθήκη κενών ή tabs μπροστά από κάθε γραμμή κώδικα. Όλες οι γραμμές κώδικα που ανήκουν στο ίδιο block πρέπει να έχουν ίσο αριθμό κενών ή tabs.

1.6.2 Εκτέλεση υπό συνθήκες και η εντολή if

Η εντολή if επιτρέπει την εκτέλεση κώδικα υπό συνθήκες. Η σύνταξή της είναι ως εξής:

```
if συνθήκη:
    εντολή 1
    εντολή 2
εντολή 1 εκτός block
```

Στην περίπτωση που η συνθήκη, δηλαδή η έκφραση πριν την άνω κάτω τελεία, είναι αληθής θα εκτελεστούν οι εντολές 1 και 2 με την σειρά, η οποίες ανήκουν στο ίδιο block. Στην συνέχεια θα συνεχιστεί η εκτέλεση των εντολών εκτός του block. Φυσικά στην περίπτωση που η συνθήκη δεν ισχύει οι εντολές εντός του block δεν θα εκτελεστούν.

Εάν θέλουμε να εισάγουμε ακόμα μία συνθήκη και να εκτελέσουμε κάποιο block στην περίπτωση που ισχύει μπορούμε να χρησιμοποιήσουμε την εντολή elif, της οποίας η σύνταξη είναι ίδια με αυτή της if. Τέλος εάν θέλουμε να εκτελέσουμε κάποιο block στην περίπτωση

που δεν ισχύει καμία από τις προηγούμενες συνθήκες, δηλαδή σε κάθε περίπτωση, χρησιμοποιούμε την εντολή else.

```
a = input('Δώσε έναν αριθμό: ')
if a > 0:
    print 'Θετικός Αριθμός'
elif a < 0:
    print 'Αρνητικό Αριθμός"
else:
    print 'Μηδέν'
```

Στο παραπάνω παράδειγμα ο χρήστης εισάγει έναν αριθμό. Στην συνέχεια το πρόγραμμα ελέγχει εάν ο αριθμός είναι μεγαλύτερος από το μηδέν οπότε και τυπώνει στην οθόνη 'Θετικός Αριθμός', εάν αυτό δεν ισχύει ελέγχει εάν είναι μικρότερος από το μηδέν οπότε τυπώνει 'Αρνητικό Αριθμός' και τέλος, εάν δεν ισχύει καμία από τις παραπάνω συνθήκες τυπώνει 'Μηδέν'.

Πρέπει να σημειωθεί ότι οι εντολές elif και else δεν μπορούν να χρησιμοποιηθούν μόνες τους χωρίς να έχει προηγηθεί μια εντολή if. Ακόμα η εντολή if μπορεί να χρησιμοποιηθεί και μέσα στο block κώδικα μιας άλλης εντολής if. Τότε το block κώδικα της δεύτερης εντολής if ονομάζεται εμφωλευμένο (nested) block.

1.6.3 Βρόχοι

Με τους βρόχους μπορούμε να επαναλαμβάνουμε μια διαδικασία, δηλαδή την εκτέλεση ενός block κώδικα, πολλές φορές ή όσο μια συνθήκη είναι αληθής.

Ο βρόχος while

Με τον βρόχο while μια διαδικασία επαναλαμβάνεται όσο μια συνθήκη είναι αληθής. Για παράδειγμα μπορούμε να τυπώσουμε τους αριθμούς από το 1 έως το 100 χωρίς να χρειάζεται να γράψουμε 100 διαφορετικές εντολές print:

```
x = 1
while x <= 100:
    print x
    x +=1
```

Οι εντολή print x που τυπώνει τον αριθμό και η εντολή x+=1 που αυξάνει τον αριθμό κατά 1 επαναλαμβάνονται μέχρι η τιμή του x να γίνει 100, όσο δηλαδή η συνθήκη x<= 100 είναι αληθής.

Ο βρόχος for

Με τον βρόχο for μπορούμε επαναλαμβάνουμε μια διαδικασία για ένα σύνολο στοιχείων, για παράδειγμα για όλα τα στοιχεία μιας λίστας:

```
names = ['nikos', 'maria', 'giorgos', 'makis']
for name in names:
    print name
```

Εάν θέλουμε για οποιοδήποτε λόγο να διακόψουμε την εκτέλεση ενός βρόχου, χρησιμοποιούμε την εντολή break. Η εντολή αυτή "σπάει" το βρόχο και συνεχίζεται η εκτέλεση των υπόλοιπων εντολών του προγράμματος. εάν αυτές υπάρχουν.

1.7 Δημιουργία συναρτήσεων

Η δημιουργία μιας συνάρτησης γίνεται με την εντολή def, και στην συνέχεια η συνάρτηση μπορεί να χρησιμοποιηθεί ακριβώς όπως τις ενσωματωμένες συναρτήσεις της Python:

```
>>> def hello(name):
...     return 'Hello, ' + name  + '!'
...
>>> print hello('dimos')
Hello, dimos!
```

Με την εντολή return η συνάρτηση επιστρέφει το αποτέλεσμα όταν την καλέσουμε. Σε ορισμένες περιπτώσεις οι συναρτήσεις δεν επιστρέφουν τίποτα, για παράδειγμα μπορεί

απλά να τυπώνουν στην οθόνη, αυτές οι συναρτήσεις δεν έχουν στο block τους την εντολή return ή εάν την έχουν δεν επιστρέφει τίποτα.

Στο παραπάνω παράδειγμα το name αποτελεί παράμετρο της συνάρτησης. Επίσης οι μεταβλητές και οι παράμετροι της συνάρτησης μπορούν να χρησιμοποιηθούν μόνο μέσα σε αυτήν, εκτός αν οριστούν ως global.

1.8 Δημιουργία τάξεων

Οι τάξεις (classes) είναι ένα είδος αντικειμένου, και όλα τα αντικείμενα που ανήκουν σε μια τάξη αποτελούν ένα στιγμιότυπο αυτής. Η δημιουργία μιας τάξης γίνεται με την εντολή class:

```
class Person:
    def setName(self, name):
        self.name = name
    def getName(self):
        return self.name
    def greet(self):
        print "Hello, world! I'm %s." % self.name
```

Στο παραπάνω παράδειγμα δημιουργούμε την τάξη Person, οι setName(), getName() και greet() αποτελούν τις μεθόδους της συνάρτησης. Οι μέθοδοι μοιάζουν με συναρτήσεις και ορίζονται με τον ίδιο τρόπο αλλά μέσα στο block κώδικα της τάξης. Η βασική διαφορά με τις συναρτήσεις είναι η παράμετρος self η οποία παίρνει την τιμή της από το στιγμιότυπο της τάξης κάθε φορά, οπότε ο χρήστης δεν χρειάζεται να την ορίσει.

1.9 Εξαιρέσεις

Κατά τη συγγραφή προγραμμάτων είναι δυνατό να συναντήσουμε κάποιες περίεργες ή εξαιρετικές καταστάσεις, όπως λάθη ή καταστάσεις που δεν περιμένουμε να συμβούν αρκετά συχνά, τις εξαιρέσεις (exceptions). Η διαχείριση τους στην Python γίνεται με τα exception αντικείμενα. Όταν συμβεί κάτι τέτοιο η Python κάνει raise ένα exception:

```
>>> 1/0
Traceback (most recent call last):
```

```
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
>>>
```

Στο παραπάνω παράδειγμα γίνεται raise ένα exception το οποίο είναι στιγμιότυπο της τάξης ZeroDivisionError. Η Python διαθέτει πολλές ενσωματωμένες εξαιρέσεις και υπάρχει η δυνατότητα ο χρήστης να ορίσει και δικές του.

Το ενδιαφέρον με τις εξαιρέσεις είναι ότι μπορούμε να τις διαχειριστούμε και να ενεργήσουμε ανάλογα όταν αυτές συμβούν, αυτό γίνεται με τις εντολές try και except:

```
try:
    x = input('Enter the first number: ')
    y = input('Enter the second number: ')
    print x/y
except ZeroDivisionError:
    print "The second number can't be zero!"
```

Στο παραπάνω παράδειγμα εάν ο χρήστης δώσει τον δεύτερο αριθμό μηδέν, η Python θα τυπώσει το μήνυμα 'The second number can't be zero' αντί της εξαίρεσης και του σφάλματος. Ακόμα υπάρχει η δυνατότητα να χρησιμοποιήσουμε περισσότερες από μία εντολές except μέσα στην ίδια try ή να ορίσουμε περισσότερες από μία εξαιρέσεις με μία εντολή except.

Κεφάλαιο 2:

NETWORK PACKET SNIFFING

2.1 Γενικά

Το network packet sniffing είναι η διαδικασία κατά την οποία είναι δυνατή η παρακολούθηση της κίνησης σε ένα δίκτυο με προγράμματα σχεδιασμένα ειδικά για αυτό το σκοπό, τα network packet sniffers. Αυτά χρησιμοποιούνται συνήθως από διαχειριστές συστημάτων και δικτύων, για επίλυση προβλημάτων σχετικά με το δίκτυο και την εξασφάλιση της ομαλής λειτουργίας του αλλά και από κακόβουλους χρήστες για υποκλοπή δεδομένων (π.χ. κωδικών πρόσβασης , ηλεκτρονικής αλληλογραφίας κ.α), για την παραβίαση δηλαδή της ασφάλειας του δικτύου και των κόμβων που συνδέονται σε αυτό. Δύο από τα πιο γνωστά netwrok packet sniffers είναι το tcpdump και το wireshark τα οποία διαθέτουν αρκετές δυνατότητες και είναι open source.

2.2 Δίκτυα συνδεδεμένα με network hub

2.2.1 Η λειτουργία του network hub

Το network hub (διανομέας δικτύου) είναι μια συσκευή στην οποία συνδέονται οι κόμβοι ενός δικτύου με καλώδια συνεστραμμένων ζευγών ή οπτικών ινών ώστε να είναι δυνατή η μεταξύ τους επικοινωνία. Λειτουργεί στο 1ο στρώμα (φυσικό) του μοντέλου OSI και μεταδίδει τα πακέτα που λαμβάνει σε κάποια θύρα του, σε όλες τις υπόλοιπες θύρες του, χωρίς κανένα απολύτως έλεγχο. Λειτουργεί δηλαδή σαν επαναλήπτης αλλά με περισσότερες από δυο θύρες. Για τον λόγο αυτό πολλές φορές αποκαλείται και multiport repeater (επαναλήπτης πολλαπλών θυρών)

2.2.2 Η κάρτα δικτύου (promiscuous mode)

Η κάρτα δικτύου ενός Η/Υ, σε κανονική λειτουργία, ελέγχει την διεύθυνση προορισμού κάθε πακέτου που δέχεται και περνάει στον πυρήνα του λειτουργικού συστήματος μόνο τα πακέτα των οποίων η διεύθυνση είναι ίδια με την δική της, απορρίπτοντας τα υπόλοιπα. Έτσι ένα packet sniffer, στην πιο απλή του λειτουργία, είναι σε θέση να παρακολουθεί μόνο τα δικτυακά πακέτα που λαμβάνει και στέλνει ο Η/Υ στον οποίο εκτελείται.

Οι περισσότερες κάρτες δικτύου και οι οδηγοί τους υποστηρίζουν μια ειδική κατάσταση που ονομάζεται promiscuous mode. Όταν η κάρτα δικτύου τεθεί σε αυτή την κατάσταση, περνάει στον πυρήνα του λειτουργικού συστήματος όλα τα δικτυακά πακέτα που φτάνουν σε αυτήν άσχετα με το ποιος είναι ο πραγματικός τους προορισμός.

2.2.3 Παρακολούθηση της κίνησης μεταξύ τρίτων κόμβων

Θέτοντας λοιπόν την κάρτα δικτύου σε promiscuous mode, και εξαιτίας του τρόπου λειτουργίας του hub, το packet sniffer είναι σε θέση να καταγράψει την κίνηση μεταξύ όλων των κόμβων που λειτουργούν στο ίδιο υποδίκτυο με τον Η/Υ στον οποίο αυτό εκτελείται. Αυτό είναι αρκετά χρήσιμο για την επίλυση τυχόν προβλημάτων στο δίκτυο, καθώς μπορούμε να παρακολουθήσουμε την κίνηση σε κόμβους που δεν έχουμε φυσική πρόσβαση, ή σε κόμβους που η εκτέλεση ενός packet sniffer δεν είναι δυνατή.

2.3 Δίκτυα συνδεδεμένα με network switch

2.3.1 Η λειτουργία του network switch

Το network switch (μεταγωγέας δικτύου) , εξυπηρετεί τον ίδιο σκοπό με το network hub αλλά η λειτουργία του διαφέρει. Λειτουργεί στο 2ο στρώμα (ζεύξης δεδομένων) του μοντέλου OSI, και ανάλογα με τον τρόπο προώθησης των πακέτων διακρίνεται στους εξής τύπους:

- **Store and forward** : Το switch αποθηκεύει προσωρινά το πακέτο και πραγματοποιεί έλεγχο σφαλμάτων (checksum) πριν το προωθήσει.

- **Cut through** : Το switch διαβάζει από το πακέτο την φυσική διεύθυνση προορισμού πρίν το προωθήσει, χωρίς να πραγματοποιήσει έλεγχο σφαλμάτων.

- **Fragment free** : Αυτή η κατηγορία αποτελεί μια προσπάθεια να συνδυαστούν οι δύο προηγούμενες κατηγορίες. Το switch διαβάζει τα πρώτα 64 bytes του πακέτου, στα οποία περιέχονται οι πληροφορίες διεύθυνσης αλλά και επιπλέον πληροφορίες που βοηθούν στην ανίχνευση σφαλμάτων (π.χ. Collisions), και στην συνέχεια προωθεί το πακέτο. Με αυτό τον τρόπο βέβαια δεν ανιχνεύονται σφάλματα στα ωφέλιμα δεδομένα (payload) του πακέτου.

- **Adaptive switching** : Τα switches αυτής της κατηγορίας έχουν την δυνατότητα να αλλάζουν αυτόματα μεταξύ των τριών προηγούμενων μεθόδων

Επιπλέον ένα switch μπορεί να ενσωματώνει λειτουργίες και από άλλα στρώματα του μοντέλου OSI, έτσι υπάρχουν switches που λειτουργούν στο 3ο στο 4ο ακόμη και στο 7ο στρώμα.

Ασχέτως με τον τρόπο με τον οποίο προωθεί τα πακέτα, κάθε switch διατηρεί στην προσωρινή του μνήμη ένα πίνακα αναζήτησης (lookup table ή CAM table) στον οποίο είναι

αποθηκευμένη η φυσική διεύθυνση και η θύρα σύνδεσης για κάθε κόμβο του δικτύου. Κάθε φορά που μεταδίδεται ένα πακέτο το switch αναζητεί την φυσική διεύθυνση προορισμού του πακέτου, στον πίνακα αναζήτησης και αποστέλλει το πακέτο μόνο στη θύρα που είναι συνδεδεμένος ο κόμβος για τον οποίο προορίζεται.

2.3.2 Παρακολούθηση της κίνησης μεταξύ τρίτων κόμβων

Εξαιτίας λοιπόν του τρόπου λειτουργίας του, το switch παρέχει μεγαλύτερη ασφάλεια σε σχέση με το hub καθώς ένα packet sniffer, υπό κανονικές συνθήκες, δεν μπορεί να καταγράψει την κίνηση μεταξύ άλλων κόμβων του δικτύου παρά μόνο την κίνηση από και προς τον κόμβο στον οποίο αυτό εκτελείται.

Παρόλα αυτά σε ορισμένα δίκτυα που χρησιμοποιούν switches για την σύνδεση των κόμβων τους, η καταγραφή της κίνησης μεταξύ τρίτων κόμβων παραμένει δυνατή υπό ορισμένες συνθήκες. Για παράδειγμα λόγω κάποιων αδυναμιών ασφάλειας στα πρωτόκολλα που χρησιμοποιούνται.

Κεφάλαιο 3:

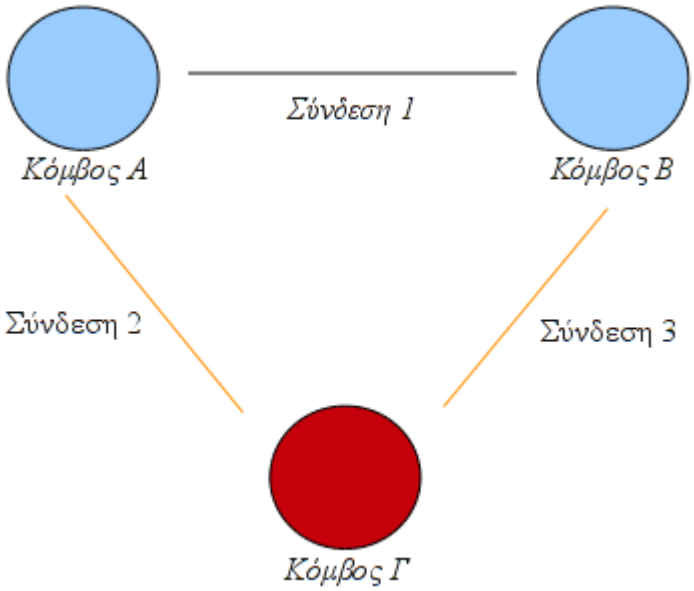
Η ΕΠΙΘΕΣΗ MITM ΜΕ ARP CACHE POISONING

3.1 Η επίθεση τύπου MITM

Η επίθεση Man-In-The-Middle (MITM) είναι ένας τύπος επίθεσης κατά τις ασφάλειας ενός δικτύου Η/Υ. Ο επιτιθέμενος εκμεταλλευόμενος κάποια αδυναμία ασφάλειας, συνήθως κάποιου πρωτοκόλλου επικοινωνίας, "ξεγελάει" δύο κόμβους ώστε να στείλουν τα δεδομένα που προορίζονται για την μεταξύ τους επικοινωνία σε αυτόν. Στην συνέχεια μπορεί:

- Να προωθήσει τα πακέτα στον αρχικό τους προορισμό (παθητική παρακολούθηση, χρησιμοποιείται για υποκλοπή δεδομένων)
- Να αλλοιώσει τα πακέτα και στην συνέχεια να τα προωθήσει στον αρχικό τους προορισμό (ενεργή παρακολούθηση)
- Να μην προωθήσει τα πακέτα διακόπτοντας την επικοινωνία μεταξύ των δύο κόμβων, πραγματοποιώντας έτσι μια επίθεση άρνησης παροχής υπηρεσιών (Denial Of Service attack)

Στο σχήμα 3.1 αναπαριστάται μια επίθεση MITM. Οι Κόμβοι Α και Β θεωρούν ότι επικοινωνούν μεταξύ τους μέσω της Σύνδεσης 1, όμως στην πραγματικότητα η επικοινωνία γίνεται μέσω των Συνδέσεων 2 και 3 τις οποίες ελέγχει ο Κόμβος Γ που είναι και ο επιτιθέμενος . Ο Κόμβος Γ στην ουσία παρεμβάλλεται στο μέσο της επικοινωνίας των Κόμβων Α και Β, εξ ου και η ονομασία της επίθεσης "Man In The Middle", δηλαδή ενδιάμεσος.

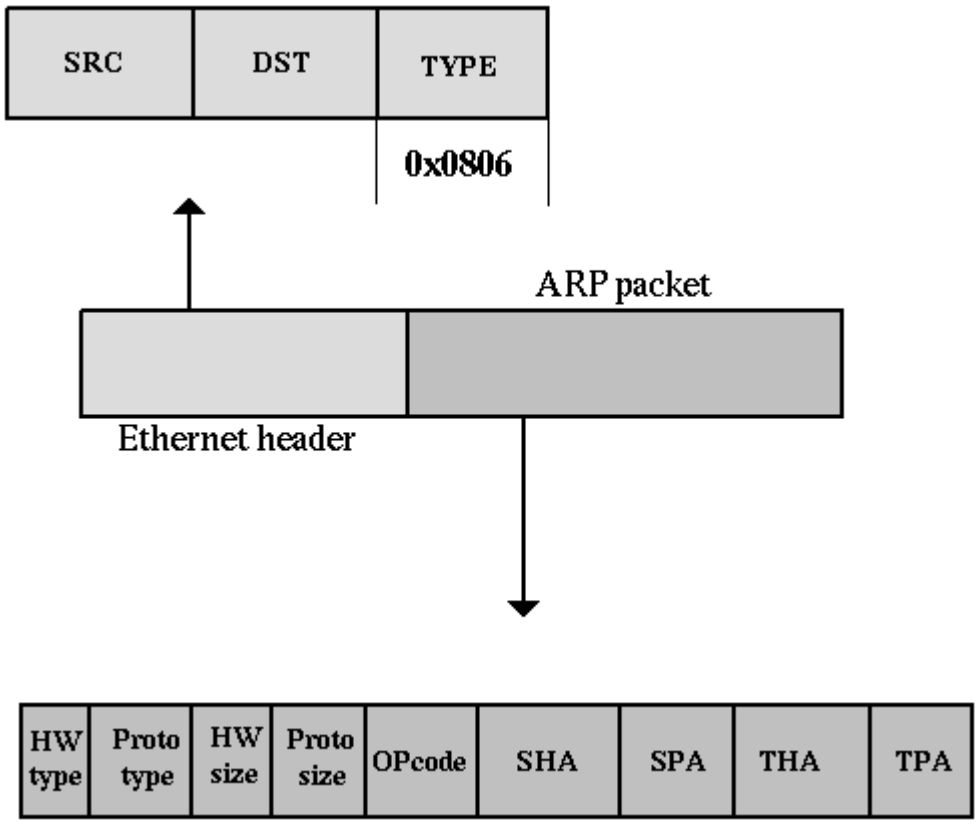


***σχήμα 3.1**: Αναπαράσταση μιας επίθεσης MITM.*

3.2 Το πρωτόκολλο ARP

3.2.1 Γενικά

Το πρωτόκολλο ARP (Address Resolution Protocol) είναι ένα πρωτόκολλο δικτύου Η/Υ που χρησιμοποιείται για την προσδιορισμό της φυσικής διεύθυνσης (διεύθυνση στρώματος ζεύξης δεδομένων) ενός κόμβου του δικτύου, όταν είναι γνωστή μόνο η διεύθυνση δικτύου (διεύθυνση στρώματος δικτύου) του, ώστε να είναι δυνατή η επικοινωνία μεταξύ των κόμβων. Θεωρείται πρωτόκολλο του 3ου στρώματος του μοντέλου OSI και ορίστηκε το 1982 με το RFC 826. Χρησιμοποιείται σε τοπικά δίκτυα συνήθως σε συνδυασμό με τα πρωτόκολλα IPv4 και Ethernet, παρόλα αυτά μπορεί να χρησιμοποιηθεί και με άλλα πρωτόκολλα.



σχήμα 3.2 *: Η δομή ενός πακέτου ARP ενθυλακωμένο σε ένα πακέτο Ethernet.*

Στο σχήμα 3.2 αναπαριστάται η δομή ενός πακέτου ARP ενθυλακωμένο σε ένα πακέτο Ethernet. Το πακέτο ARP έχει τα εξής πεδία:

- **HW type**: Το πεδίο αυτό περιέχει μια τιμή που προσδιορίζει το πρωτόκολλο του

στρώματος ζεύξης δεδομένων που χρησιμοποιείται, στην περίπτωση του Ethernet η τιμή αυτή είναι 1.

- **Proto type**: Το πεδίο αυτό περιέχει μια τιμή που προσδιορίζει το πρωτόκολλο του στρώματος δικτύου που χρησιμοποιείται, για το πρωτόκολλο IPv4 η τιμή αυτή είναι 0x0800.

- **HW size**: Το πεδίο αυτό περιέχει το μέγεθος, σε bytes, της διεύθυνσης του στρώματος ζεύξης δεδομένων, για διευθύνσεις Ethernet η τιμή είναι 6.

- **Proto size**: Το πεδίο αυτό περιέχει το μέγεθος, σε bytes, της διεύθυνσης του στρώματος δικτύου, για διευθύνσεις IPv4 η τιμή είναι 4.

- **OPcode**: Το πεδίο αυτό περιέχει μια τιμή που προσδιορίζει την ενέργεια ARP που εκτελεί ο αποστολέας, η τιμή είναι 1 για αιτήματα και 2 για απαντήσεις.

- **SHA**: Το πεδίο αυτό περιέχει την διεύθυνση του στρώματος ζεύξης δεδομένων του αποστολέα και το μέγεθος του ορίζεται από το πεδίο HW size.

- **SPA**: Το πεδίο αυτό περιέχει την διεύθυνση του στρώματος δικτύου του αποστολέα και το μέγεθος του ορίζεται από το πεδίο Proto size.

- **THA**: Το πεδίο αυτό περιέχει την διεύθυνση του στρώματος ζεύξης δεδομένων του παραλήπτη και το μέγεθος του ορίζεται από το πεδίο HW size.

- **TPA**: Το πεδίο αυτό περιέχει την διεύθυνση του στρώματος δικτύου του παραλήπτη και το μέγεθος του ορίζεται από το πεδίο Proto size.

Η επικεφαλίδα (header) του πακέτου Ethernet έχει τα πεδία:

- **SRC**: Το πεδίο αυτό περιέχει την Ethernet διεύθυνση (MAC address) του αποστολέα και το μέγεθος του είναι 6 bytes.

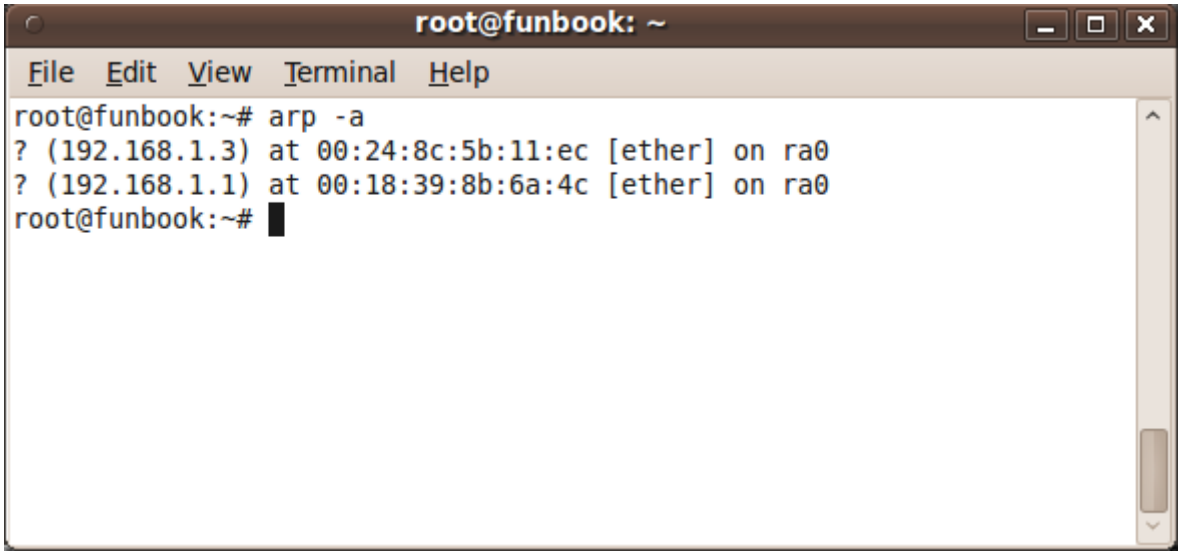
- **DST**: Το πεδίο αυτό περιέχει την Ethernet διεύθυνση (MAC address) του παραλήπτη και το μέγεθος του είναι 6 bytes.

- **TYPE**: Το πεδίο αυτό περιέχει μια τιμή που προσδιορίζει τον τύπο του πακέτου και στην περίπτωση του ARP είναι 0x0806.

3.2.2 Η βασική λειτουργία

Κάθε κόμβος σε ένα τοπικό δίκτυο διατηρεί ένα πίνακα, τον οποίο έχει αποθηκευμένο προσωρινά στην μνήμη του (cached), στον οποίο υπάρχουν εγγραφές με την αντιστοιχία IP

και Ethernet διεύθυνσης (υποθέτουμε για πρακτικούς λόγους ότι το δίκτυο χρησιμοποιεί τα πρωτόκολλα IPv4 και Ethernet) για άλλους κόμβους του δικτύου με τους οποίους έχει ήδη επικοινωνήσει, έχει γίνει δηλαδή επίλυση της διεύθυνσής τους με το πρωτόκολλό ARP. Ο πίνακας αυτός ονομάζεται ARP table ή απλά ARP cache, και χρησιμοποιείται για τον περιορισμό των αιτημάτων ARP που αποστέλλονται στο δίκτυο, καθώς κάθε κόμβος ελέγχει πρώτα στον πίνακα του εάν υπάρχει εγγραφή για τον κόμβο με τον οποίο θέλει να επικοινωνήσει και στέλνει ένα αίτημα ARP μόνο στην περίπτωση που δεν βρεθεί εγγραφή. Επίσης κάθε εγγραφή του πίνακα είναι προσωρινή, και διαγράφεται εάν δεν υπάρξει επικοινωνία με τον κόμβο στον οποίο αναφέρεται για ένα χρονικό διάστημα που καθορίζεται από το λειτουργικό σύστημα του κόμβου (π.χ. σε συστήματα Linux το χρονικό διάστημα αυτό είναι προκαθορισμένο στα 60 δευτερόλεπτα.).Ο διαχειριστής του κόμβου έχει την δυνατότητα να προβάλει (εικόνα 3.1) και να μεταβάλει τις εγγραφές αυτές και η διαχείριση στα περισσότερα λειτουργικά συστήματα γίνεται με την εντολή **arp**.



εικόνα 3.1*: Προβολή του ARP table στο λειτουργικό σύστημα Linux.*

Όταν ο κόμβος Α ενός τοπικού δικτύου θέλει να στείλει δεδομένα σε κάποιον άλλο κόμβο Β, καθορίζει με βάση την IP διεύθυνση (και την μάσκα υποδικτύου) του κόμβου Β εάν ο κόμβος Β βρίσκεται στο ίδιο τοπικό δίκτυο ή καλύτερα υποδίκτυο με αυτόν. Εάν αυτό ισχύει, ο κόμβος Α αναζητεί στον πίνακα ARP του την Ethernet διεύθυνση (MAC address) του κόμβου Β με βάση την IP διεύθυνση αυτού, εάν δεν την βρει στέλνει ένα αίτημα ARP προς όλους τους κόμβους του υποδικτύου (broadcast), ζητώντας την MAC διεύθυνση του κόμβου Β. Το Ethernet πακέτο που θα σταλεί, θα έχει στην επικεφαλίδα του στο πεδίο **SRC**

την MAC διεύθυνση του κόμβου Α και στο πεδίο **DST** την MAC διεύθυνση για broadcast στην οποία όλες οι οκτάδες είναι άσοι, δηλαδή FF:FF:FF:FF:FF:FF και στο πεδίο **TYPE** την τιμή 0x0806 αφού πρόκειται για ένα πακέτο ARP. Το πακέτο ARP (το οποίο είναι ενθυλακωμένο στο πακέτο Ethernet) θα έχει στο πεδίο **Opcode** την τιμή 1 (αίτημα), στα πεδία **SHA** και **SPA** την IP και MAC διεύθυνση του κόμβου Α αντίστοιχα και στο πεδίο **TPA** την IP διεύθυνση του κόμβου Β. Το πεδίο **THA**, που αναφέρεται στην MAC διεύθυνση του κόμβου Β που ζητούμε, θα περιέχει μια MAC διεύθυνση με όλες της οκτάδες μηδέν, δηλαδή 00:00:00:00:00:00 (unspecified MAC address).

Όλοι οι κόμβοι θα λάβουν το αίτημα ARP, αλλά μόνο ο κόμβος Β στον οποίο αυτό αναφέρεται θα απαντήσει, ενημερώνοντας τον κόμβο Α για την MAC διεύθυνση του. Η απάντηση ARP θα είναι ένα unicast πακέτο, δηλαδή θα έχει ένα παραλήπτη. Τα πεδία της απάντησης θα είναι: Στην επικεφαλίδα Ethernet το **SRC** και **DST** θα περιέχουν την MAC διεύθυνση του κόμβου Β και του κόμβου Α αντίστοιχα και το πεδίο **TYPE** θα έχει την τιμή 0x0806. Στο πακέτο ARP το πεδίο **Opcode** θα έχει την τιμή 2 (απάντηση) και τα πεδία **SHA** και **SPA** θα περιέχουν την MAC και IP διεύθυνση του κόμβου Β, ενώ τα πεδία **THA** και **TPA** την MAC και IP διεύθυνση του κόμβου Α αντίστοιχα.

Όταν λοιπόν ο κόμβος Α λάβει την απάντηση, αφού αποθηκεύσει την MAC διεύθυνση του κόμβου Β (από το πεδίο SHA της απάντησης) στον πίνακα ARP του για μελλοντικές αποστολές, ξεκινάει την αποστολή των δεδομένων, χρησιμοποιώντας την MAC διεύθυνση του κόμβου Β στο πεδίο **DST** τις επικεφαλίδας Ethernet, για τα πακέτα που αφορούν τον τελευταίο.

Τέλος, στην περίπτωση που ο κόμβος Β δεν βρίσκεται στο ίδιο υποδίκτυο με τον κόμβο Α, ο τελευταίος θα αποστείλει τα πακέτα χρησιμοποιώντας την MAC διεύθυνσή (στο πεδίο DST) της πύλης του δικτύου (δρομολογητή) και στη περίπτωση που αυτή δεν είναι γνωστή θα φροντίσει να την "μάθει" στέλνοντας ένα αίτημα ARP. Στην περίπτωση που ο κόμβος Α δεν μπορεί να "διαπιστώσει" εάν ο κόμβος Β ανήκει στο ίδιο υποδίκτυο, δεν έχει δηλαδή σαφή εικόνα για την τοπολογία του δικτύου, θα αποστείλει το αίτημα ARP, και εφόσον η πύλη "καταλάβει" ότι αυτό προορίζεται για κάποιον κόμβο εκτός του υποδικτύου και μπορεί να δρομολογήσει τα δεδομένα, θα στείλει στον κόμβο Α μια απάντηση ARP με την δική της διεύθυνση MAC.

3.3 Η επίθεση ARP cache poisoning

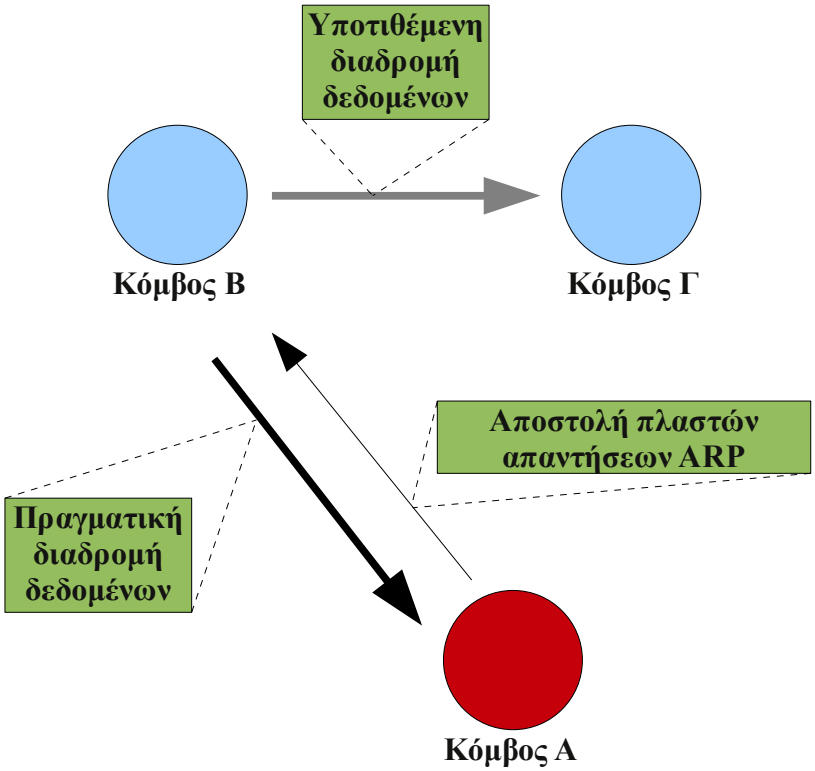
3.3.1 Λειτουργία

Το ARP cache poisoning, γνωστό και ως ARP spoofing, είναι μια κοινή επίθεση εναντίον της ασφάλειας τοπικών δικτύων, που χρησιμοποιούν τα πρωτόκολλα IPv4 και Ethernet για την επικοινωνία των κόμβων και φυσικά το πρωτόκολλο ARP για την επίλυση των MAC διευθύνσεων, με στόχο την ανακατεύθυνση της κίνησης μεταξύ των κόμβων του δικτύου. Ένα από τα πιο γνωστά προγράμματα που υλοποιούν αυτή την επίθεση είναι το ettercap το οποίο έχει κάποια πολύ ενδιαφέροντα χαρακτηριστικά και είναι open source.

Το ARP είναι ένα stateless πρωτόκολλο, που σημαίνει ότι ο αποστολέας και ο δέκτης πακέτων ARP δεν είναι υποχρεωμένοι να καταγράφουν την κατάσταση (state) της μεταξύ τους επικοινωνίας. Σύμφωνα με τις προδιαγραφές του πρωτοκόλλου, κάθε κόμβος είναι υποχρεωμένος να ενημερώσει τον ARP πίνακά του σε περίπτωση που λάβει ένα πακέτο ARP το οποίο δηλώνει ότι ένας άλλος κόμβος έχει διαφορετική MAC διεύθυνση από αυτή που ήδη υπάρχει στον ARP πίνακα του πρώτου. Ένα τέτοιο πακέτο ARP, το οποίο δεν θα έπρεπε να είχε σταλεί υπό κανονικές συνθήκες ονομάζεται Gratuitous ARP packet.

Εκμεταλλευόμενος τα παραπάνω, ένας επιτιθέμενος μπορεί να στείλει πλαστά πακέτα ARP σε κάποιο κόμβο του δικτύου ώστε να τον αναγκάσει να ενημερώσει τον ARP πίνακα του με λανθασμένες εγγραφές. Συγκεκριμένα, όπως φαίνεται στο σχήμα 2.3, εάν ο επιτιθέμενος που χειρίζεται τον κόμβο Α ενός δικτύου στείλει στο κόμβο Β πλαστές απαντήσεις ARP χρησιμοποιώντας στο πεδίο **SHA** την δική του MAC διεύθυνση αλλά στο πεδίο **SPA** την IP διεύθυνση του κόμβου Γ με τον οποίο ο κόμβος Β βρίσκεται σε επικοινωνία, θα αναγκάσει τον δεύτερο να ενημερώσει τον ARP πίνακά του με τα λάθος στοιχεία και στην συνέχεια να προσαρτά στο πεδίο **DST** τις επικεφαλίδας Ethernet την MAC διεύθυνση του κόμβου Α, σε όσα πακέτα προορίζονται για τον κόμβο Γ. Έτσι όσα πακέτα στέλνει ο κόμβος Β στον κόμβο Γ στην πραγματικότητα θα φτάνουν στον κόμβο Α. Πραγματοποιώντας παράλληλα την ίδια διαδικασία εναντίον και του κόμβου Γ (με την IP διεύθυνση του κόμβου Β) και στην συνέχεια προωθώντας τα πακέτα στον αρχικό τούς, κάθε φορά, προορισμό, ο επιτιθέμενος που χειρίζεται τον κόμβο Α στην ουσία πραγματοποιεί μια επίθεση τύπου MITM. Έτσι είναι σε θέση να καταγράψει την κίνηση μεταξύ των δύο κόμβων-θυμάτων (Α και Β) χρησιμοποιώντας κάποιο network packet sniffer ακόμα και στην περίπτωση που το δίκτυο είναι συνδεδεμένο με switch, το οποίο όπως αναφέραμε παρέχει

κάποια ασφάλεια εναντίον του packet sniffing. Ακόμη ο επιτιθέμενος μπορεί να αλλοιώσει τα πακέτα πριν τα προωθήσει ή ακόμα και να μη τα προωθήσει διακόπτοντας την επικοινωνία μεταξύ των δύο κόμβων (επίθεση άρνησης παροχής υπηρεσιών). Επίσης τα πλαστά πακέτα ARP μπορούν να σταλούν προς όλους τους κόμβους (broadcast), αναγκάζοντας τους όλους να ενημερώσουν τους πίνακές τους ή ο επιτιθέμενος μπορεί να περιμένει, ώστε ο κόμβος που τον ενδιαφέρει να στείλει ένα αίτημα, πριν στείλει την απάντηση ώστε να είναι δυσκολότερος ο εντοπισμός της επίθεσης. Τέλος, η επίθεση είναι δυνατόν να πραγματοποιηθεί και με αιτήματα ARP αντί για απαντήσεις.



***σχήμα 3.3**: Αναπαράσταση της επίθεσης ARP cache poisoning*

3.3.2 Περιορισμοί και προϋποθέσεις

Όπως είναι λογικό υπάρχουν κάποιοι περιορισμοί και προϋποθέσεις για την επίτευξη αυτής της επίθεσης:

- Η επίθεση μπορεί να πραγματοποιηθεί μόνο εναντίον κόμβων που ανήκουν στο ίδιο broadcast domain με τον κόμβο που ελέγχει ο επιτιθέμενος.
- Ο επιτιθέμενος πρέπει να γνωρίζει προς τα που θα ανακατευθύνει την κίνηση στην

περίπτωση που θέλει να προωθήσει τα πακέτα στον κανονικό τους προορισμό, πρέπει δηλαδή να γνωρίζει τις IP και MAC διευθύνσεις των κόμβων που τον ενδιαφέρουν.

- Η επίθεση μπορεί να μειώσει την απόδοση του δικτύου καθώς ο κόμβος που ελέγχει ο επιτιθέμενος πρέπει να επεξεργαστεί επιπλέον πακέτα, εκτός από αυτά που προορίζονται για τον ίδιο.
- Με την επίθεση αυτή δεν μπορεί να εξαναγκαστεί ο κόμβος-θύμα να προσθέσει νέες εγγραφές στον ARP πίνακά του, πάρα μόνο να ενημερώσει τις ήδη υπάρχουσες.
- Ο επιτιθέμενος θα πρέπει να στέλνει πλαστά πακέτα ανά τακτά χρονικά διαστήματα, καθώς οι εγγραφές του πίνακα ARP δεν είναι μόνιμες και διαγράφονται μετά από κάποιο χρονικό διάστημα αδράνειας.
- Για να αποκατασταθεί ομαλά η επικοινωνία μεταξύ των κόμβων-θυμάτων, μετά το πέρας της επίθεσης, θα πρέπει να διαγραφούν οι λανθασμένες εγγραφές από τους ARP πίνακές τους ή να ενημερωθούν με τα σωστά στοιχεία.

3.3.3 Άλλες χρήσεις

Αξίζει να σημειωθεί ότι το ARP cache poisoning χρησιμοποιείται και για άλλους σκοπούς πέραν της παραβίασης του δικτύου, για παράδειγμα για διαμοιρασμό του φόρτου (load balancing) προς servers που μοιράζονται μια εικονική IP διεύθυνσή ή για ανακατεύθυνση της κίνησης σε ένα δίκτυο. Ένα παράδειγμα τέτοιου προγράμματος αποτελεί το PacketFence, το οποίο είναι ένα open source πρόγραμμα ελέγχου πρόσβασης δικτύου (Network Access Control)

Κεφάλαιο 4:

SOFTWARE ΚΑΙ ΕΡΓΑΛΕΙΑ

Εισαγωγή

Η υλοποίηση του προγράμματος έγινε εξ' ολοκλήρου με την χρήση open source τεχνολογιών. Συγκεκριμένα η ανάπτυξη έγινε στην γλώσσα προγραμματισμού Python και στο λειτουργικό σύστημα Linux. Επίσης χρησιμοποιήθηκαν ορισμένες εφαρμογές για την συγγραφή του κώδικα και για τον έλεγχο των αποτελεσμάτων.

4.1 Python modules που χρησιμοποιήθηκαν

4.1.1 Modules της βασικής βιβλιοθήκης

Όπως είπαμε η βασική βιβλιοθήκη της python περιέχει αρκετά modules που προσφέρουν διαφορές λειτουργίες το καθένα, το πιο σημαντικά, από αυτά που χρησιμοποιήθηκαν για την υλοποίηση αυτής της εργασίας, είναι:

- Το **socket** module, που μας επιτρέπει να χρησιμοποιούμε την διεπαφή BSD socket, που υποστηρίζουν τα περισσότερα σύγχρονα λειτουργικά συστήματα. Συγκεκριμένα, χρησιμοποιήθηκαν sockets τύπου SOCK_RAW της οικογένειας PF_PACKET, που επιτρέπουν απευθείας πρόσβαση στον οδηγό της κάρτας δικτύου, για αποστολή και λήψη δικτυακών πακέτων, χωρίς καμία μεταβολή αυτών από τον πυρήνα του λειτουργικού συστήματος. Επίσης το module αυτό περιέχει κάποιες πολύ χρήσιμες συναρτήσεις, όπως την socket.inet_aton() που μετατρέπει μια IP διεύθυνση από την συνηθισμένη dotted-quad μορφή (π.χ. 192.168.1.1) σε ένα δυαδικό string με μέγεθος 4 bytes (binary packed), στην μορφή δηλαδή που πρέπει να μετατρέψουμε την διεύθυνση αυτή για την κατασκευή δικτυακών πακέτων. Για την αντίστροφη διαδικασία υπάρχει η συνάρτηση socket._inet_ntoa().

- Το **time** module, που περιέχει συναρτήσεις σχετικές με την ώρα, για παράδειγμα συναρτήσεις για μετατροπή μεταξύ διαφόρων μορφών ώρας.

- Το **binascii** module το οποίο μας επιτρέπει την μετατροπή δεδομένων μεταξύ δυαδικής και δεκαεξαδικής αναπαράστασης, όπως και άλλων μορφών αναπαράστασης/κωδικοποίησης.

- Το **signal** module, το οποίο μας επιτρέπει την διαχείριση των σημάτων, για την επικοινωνία μεταξύ των διεργασιών.

- Το **multiprocessing** module, το οποίο επιτρέπει την δημιουργία διεργασιών για την παράλληλη εκτέλεση ενεργειών, στην περίπτωση μας η επίθεση MITM με ARP cache

poisoning πρέπει να εκτελείται παράλληλα με το packet sniffing, και αυτό επιτυγχάνεται με το συγκεκριμένο module.

- Το **optparse** module, που παρέχει έναν εύκολο και ευέλικτο τρόπο διαχείρισης των επιλογών γραμμής εντολών (command-line options), που ο χρήστης εισάγει κατά την εκτέλεση του προγράμματος.

4.1.2 Third-Party Modules

Η Python δεν παρέχει δυνατότητες packet sniffing και κατασκευής πακέτων από την βασική της βιβλιοθήκη. Έτσι είναι απαραίτητη η χρήση κάποιων third-party modules.

Για το packet sniffing χρησιμοποιήθηκε το Pypcap module, το οποίο επιτρέπει την χρήση της βιβλιοθήκης libpcap, η οποία είναι γραμμένη για τις γλώσσες C και C++, στην Python και επιτρέπει την καταγραφή δικτυακών πακέτων. Λειτουργεί δηλαδή σαν wrapper της libpcap για την Python. Τα αντικείμενα pcap που μπορούμε να δημιουργήσουμε με το module αυτό είναι iterators του εαυτού τους, και μας επιστρέφουν κάθε φορά ένα tuple που περιέχει την χρονοσφραγίδα (timestamp) του πακέτου και το ίδιο το πακέτο, που συλλαμβάνεται από την κάρτα δικτύου μας. Κατά την δημιουργία των αντικειμένων pcap μπορούμε να ορίσουμε διάφορες παραμέτρους όπως την διεπαφή δικτύου που θα χρησιμοποιηθεί, το μέγεθος σε bytes του μέρους των πακέτων που θα καταγράψουμε και το εάν θέλουμε να θέσουμε την κάρτα δικτύου σε promiscuous mode.

Για την κατασκευή των πακέτων ARP χρησιμοποιήθηκε Dpkt module. Το module αυτό διαθέτει αρκετές τάξεις που αναπαριστούν δικτυακά πακέτα και υποστηρίζει διάφορα πρωτόκολλα. Ακόμα μπορούμε να περάσουμε τα πακέτα που έχουμε συλλάβει με το Pypcap module στις τάξεις του Dpkt και να διαχειριστούμε τα διάφορα πεδία τους, αυτό γίνεται αυτόματα, με την μέθοδο unpack() των συγκεκριμένων αντικειμένων, περνώντας τα πακέτα σαν παραμέτρους κατά την δημιουργία των αντικειμένων. Επίσης με την μέθοδο pack() που διαθέτουν τα αντικείμενα του Dpkt, παίρνουμε το πακέτο σε δυαδική μορφή έτοιμο για αποστολή με κάποιο socket. Τέλος οι συναρτήσεις unpack() και pack() λαμβάνουν υπόψιν τους και το endianess του συστήματος στο οποίο εκτελείται το πρόγραμμά μας. Αναλαμβάνουν δηλαδή την μετατροπή των δεδομένων από network byte order σε host byte order και αντίστροφα.

```
import dpkt, pcap


pc = pcap.pcap(name='eth0')
pc.setfilter('src or dst port 80')


for ts, pkt in pc:
        print repr(dpkt.ethernet.Ethernet(pkt))
```

παράδειγμα 4.1*: Χρήση των Pypcap και Dpkt.*

Η βιβλιοθήκη libpcap υποστηρίζει το φιλτράρισμα των πακέτων με λογικές εκφράσεις. Στο παράδειγμα 4.1, αφού κάνουμε import τα modules Pypcap και Dpkt, δημιουργούμε το αντικείμενο pc, που είναι ένα αντικείμενο pcap, ορίζοντας για χρήση την διεπαφή δικτύου eth0, στην συνέχεια με την μέθοδο setfilter() ορίζουμε σαν φίλτρο της καταγραφής την λογική έκφραση "src or dst port 80" που σημαίνει ότι θα συλλάβουμε μόνο τα πακέτα εκείνα των οποίων η port πηγής και προορισμού είναι η 80 (http). Στην συνέχεια με την εντολή for τυπώνουμε την string αναπαράσταση της τάξης Ethernet για κάθε πακέτο που θα συλληφθεί, αφού πρώτα περάσουμε το πακέτο στην τάξη αυτή. Η πλήρης λίστα των λογικών εκφράσεων που μπορούν να χρησιμοποιηθούν ως φίλτρα μπορεί να βρεθεί στην τεκμηρίωση της βιβλιοθήκης libpcap.

```
import dpkt, pcap


pc = pcap.pcap(name='eth0')
pc.setfilter('src or dst port 80')
writer = dpkt.pcap.Writer(open('katagrafi', 'wb'))


for ts, pkt in pc:
        writer.writepkt(pkt)
```

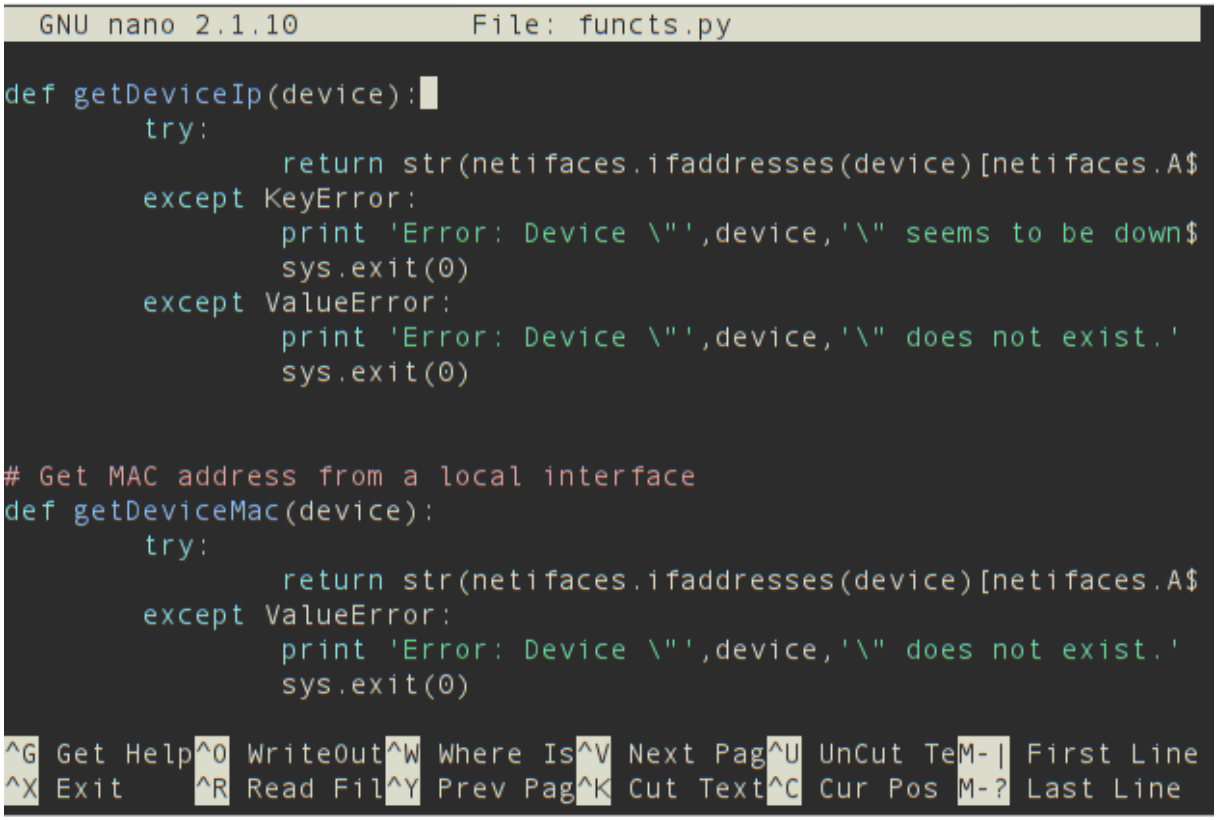
παράδειγμα 4.2*: Αποθήκευση των πακέτων σε ένα αρχείο .pcap*

Στο παράδειγμα 4.2, αφού δημιουργήσουμε ένα επιπλέον αντικείμενο , το writer που είναι ένα αντικείμενο pcap.Writer του module Dpkt, καλούμε την μέθοδο writepkt του αντικειμένου αυτού και το αποτέλεσμα είναι τα πακέτα που θα συλληφθούν να αποθηκευτούν στο αρχείο katagrafi το οποίο είναι τύπου .pcap. Τέτοιου τύπου αρχεία είναι συμβατά με τα περισσότερα network packet sniffers αλλά και με προγράμματα ανάλυσης/αποκωδικοποίησης πακέτων δικτύου.

Τελειώνοντας, επειδή η Python δεν μας παρέχει κάποιο τρόπο για πρόσβαση στις διεπαφές δικτύου του υπολογιστή μας, χρησιμοποιήθηκε το Netifaces module, το οποίο παρέχει αυτές ακριβώς της δυνατότητες. Έτσι με πολύ απλό σχετικά τρόπο, μπορούμε παραδείγματος χάρη να ανακτήσουμε την IP και MAC διεύθυνση οποιασδήποτε διεπαφής δικτύου του υπολογιστή μας. Αυτό είναι εξαιρετικά χρήσιμο για την κατασκευή των πλαστών πακέτων ARP, χωρίς να χρειάζεται να εισάγει ο χρήστης τις διευθύνσεις αυτές.

4.2 Ο επεξεργαστής κειμένου Nano

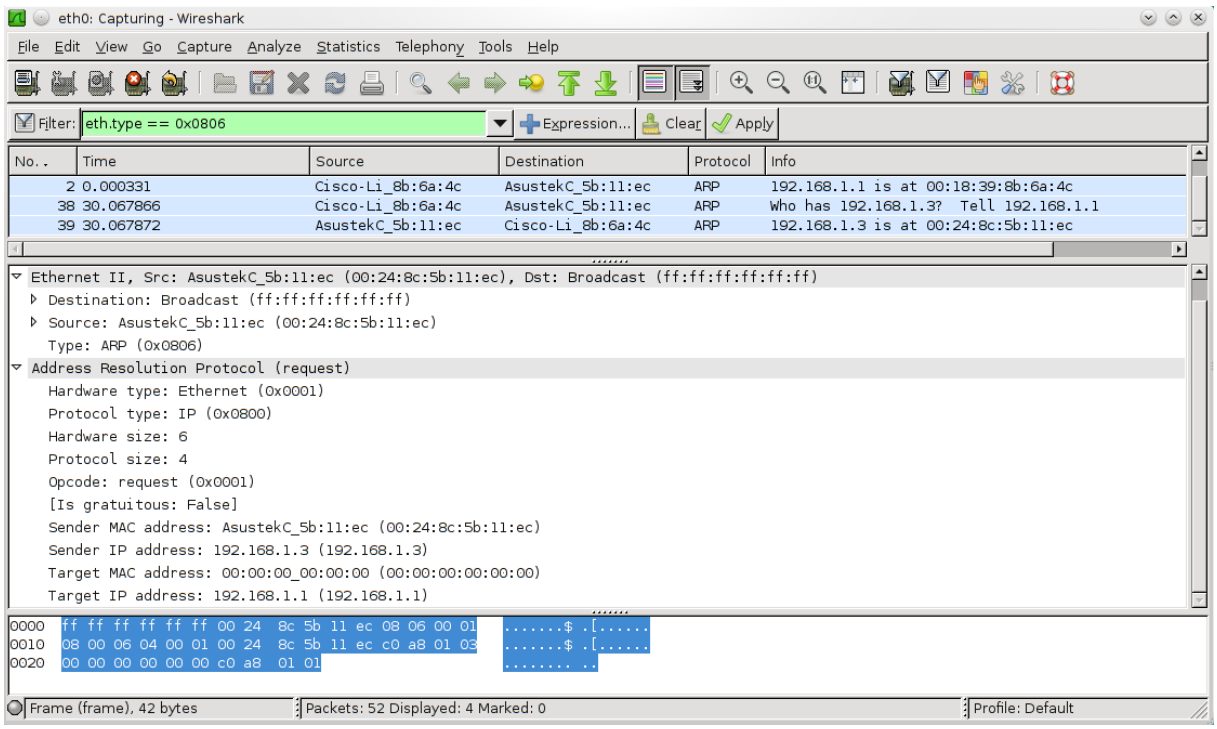
Για την συγγραφή του κώδικα χρησιμοποιήθηκε ο επεξεργαστής κειμένου Nano, οποίος είναι open source και αποτελεί κλώνο του παλαιότερου επεξεργαστή κειμένου Pico. Ο Nano είναι επεξεργαστής κειμένου γραμμής εντολών (command line editor) , που σημαίνει ότι δεν διαθέτει γραφικό περιβάλλον και εκτελείται στο κέλυφος (shell) του λειτουργικού συστήματος. Τρέχει στα περισσότερα συστήματα τύπου Unix και είναι πολύ εύχρηστος για συγγραφή και επεξεργασία κώδικα, αρχείων ρυθμίσεων (configuration files) και απλών αρχείων κειμένου. Είναι αρκετά διαδεδομένος μιας και αποτελεί τον προεπιλεγμένο επεξεργαστή κειμένου σε πολλές διανομές Linux, και διαθέτει χαρακτηριστικά όπως απαρίθμηση γραμμών και χρωματισμό του κώδικα ανάλογα με την γλωσσά προγραμματισμού που χρησιμοποιείται. Υποστηρίζει χρωματισμό κώδικα για τις γλώσσες: C, C++, Perl, Python, Ruby, Java, Assembly, Tcl, Bash scripts ενώ ο χρήστης έχει την δυνατότητα να προσθέσει χρωματισμό και για οποιαδήποτε άλλη γλώσσα επιθυμεί μέσα από το αρχείο ρυθμίσεών του, χρησιμοποιώντας μοτίβα.



***εικόνα 4.1**: Ο επεξεργαστής κειμένου Nano.*

4.3 To Wireshark

Το Wireshark είναι ένα open source πρόγραμμα παρακολούθησης κίνησης δικτύου, αλλά και ανάλυσης δικτυακών πακέτων και πρωτοκόλλων. Παλαιότερα ονομαζόταν Ethereal, αλλά το όνομα του άλλαξε λόγω εμπορικών σημάτων. Κατά την υλοποίηση της εφαρμογής μας, φάνηκε εξαιρετικά χρήσιμο, για την ανάλυση των πακέτων ARP και την διασφάλιση ότι αποτελούν ορθά πακέτα δικτύου όπως και για την ανάλυση των αρχείων pcap που το πρόγραμμά μας αποθηκεύει με την μέθοδο writepkt που αναφέραμε προηγουμένως.



***εικόνα 4.2**: Το πρόγραμμα καταγραφής και ανάλυσης πακέτων Wireshark*

Κεφάλαιο 5:

Η ΕΦΑΡΜΟΓΗ ΚΑΙ ΟΙ ΛΕΙΤΟΥΡΓΙΕΣ ΤΗΣ

5.1 Δομή

Ο κώδικας της εφαρμογής είναι χωρισμένος σε τρία αρχεία, το sniffer.py, το functs.py και το attacks.py, για την διευκόλυνση της ανάπτυξης, της ανάγνωσης και της αποσφαλμάτωσης, καθώς και της πιθανής επαναχρησιμοποίησης μέρους του κώδικα. Το sniffer.py είναι το βασικό αρχείο του προγράμματος και τα άλλα δύο φορτώνονται σε αυτό ως modules.

5.1.1 Το module functs.py

Το module αυτό περιέχει συναρτήσεις που χρησιμοποιούνται για τις επιμέρους διαδικασίες του προγράμματος. Οι συναρτήσεις αυτές είναι:

- Η **eth_ntoa(buffer)**, η οποία δέχεται ως παράμετρο μια διεύθυνση MAC σε δυαδική μορφή και επιστρέφει την δεκαεξαδική της αναπαράσταση σε οκτάδες χωρισμένες με άνω κάτω τελείες. Καλείται από άλλες συναρτήσεις του προγράμματος για την μετατροπή διευθύνσεων που έχουν ανακτηθεί από δικτυακά πακέτα.

- Η **eth_aton(buffer)**, η οποία δέχεται ως παράμετρο μια διεύθυνση MAC σε δεκαεξαδική μορφή και επιστρέφει την δυαδική της αναπαράσταση σε string μήκους 6 bytes. Καλείται από άλλες συναρτήσεις του προγράμματος για την μετατροπή διευθύνσεων πριν την ανάθεση τους ως τιμές στα διάφορα πεδία (attributes των τάξεων) κατά την δημιουργία των δικτυακών πακέτων.

- Η **getDeviceIp(device)**, η οποία δέχεται ως παράμετρο το όνομα κάποιας διεπαφής δικτύου του Η/Υ και επιστρέφει την IP διεύθυνση της διεπαφής αυτής, με την χρήση του module Netifaces.

- Η **getDeviceMac(device)**, η οποία δέχεται ως παράμετρο το όνομα κάποιας διεπαφής δικτύου του Η/Υ και επιστρέφει την MAC διεύθυνση της, με την χρήση του module Netifaces.

- Η **getMac(ipaddr, device)**, η οποία επιστρέφει την MAC διεύθυνση ενός απομακρυσμένου κόμβου του δικτύου, δεδομένης της IP διεύθυνσης αυτού, την οποία δέχεται ως παράμετρο μαζί με το όνομα της διεπαφής δικτύου που θα χρησιμοποιηθεί. Η συνάρτηση αφού κατασκευάσει ένα αίτημα ARP με το οποία με το οποίο ζητάει την MAC διεύθυνση του απομακρυσμένου κόμβου, το στέλνει και περιμένει την απάντηση ARP από την οποία και

εξάγει την ζητούμενη MAC διεύθυνση.

- Η **ethCapDesc(ts, pkt)**, η οποία τυπώνει μια απλή περιγραφή των πακέτων που συλλαμβάνονται από ένα αντικείμενο της τάξης pcap, από το module Pypcap. Η κλήση της γίνεται από την μέθοδο loop() ενός αντικειμένου pcap, η οποία της παρέχει τις παραμέτρους ts και pkt, δηλαδή την χρονοσφραγίδα του πακέτου και το ίδιο το πακέτο. Αποτελεί δηλαδή μια callback συνάρτηση.

- Η **checkIp(addr)**, η ποία ελέγχει κατά πόσο μια IP διεύθυνση, την οποία δέχεται ως παράμετρο, είναι στη σωστή μορφή. Επιστρέφει True ή False, και χρησιμοποιείται για τον έλεγχο IP διευθύνσεων που δίνει ο χρήστης στο πρόγραμμα.

5.1.2 Το module attacks.py

Το module αυτό περιέχει την συνάρτηση **mitm(target1, target2, device)**, στην οποία υλοποιείται η επίθεση MITM με ARP cache poisoning, εναντίον δύο IP διευθύνσεων τις οποίες δέχεται ως παραμέτρους μαζί με το όνομα της διεπαφής δικτύου που θα χρησιμοποιηθεί. Η συνάρτηση αφού κατασκευάσει δύο πλαστές απαντήσεις ARP (μία για κάθε κόμβο-θύμα της επίθεσης), τις αποστέλλει κάθε 10 δευτερόλεπτα. Εάν ο χρήστης διακόψει την διαδικασία, η συνάρτηση πριν την έξοδο της, αποστέλλει δύο νόμιμες απαντήσεις ARP , για την αποκατάσταση του ARP πίνακα των κόμβων-θυμάτων. Η συνάρτηση αυτή καλείται από το πρόγραμμα, εφόσον το επιλέξει ο χρήστης, σε ξεχωριστή διεργασία η οποία εκτελείται παράλληλα με το packet sniffing.

5.1.3 Το αρχείο sniffer.py

Το αρχείο αυτό είναι το βασικό αρχείο της εφαρμογής μας, και σε αυτό φορτώνονται τα modules attacks.py και functs.py με την εντολή import της Python. Επίσης εδώ ορίζονται οι διακόπτες γραμμής εντολών της εφαρμογής, και στην συνέχεια αφού γίνει η συλλογή αυτών και των τιμών τους που έχει δώσει ο χρήστης, ενεργοποιούνται οι αντίστοιχες λειτουργίες της εφαρμογής.

5.2 Λειτουργία

5.2.1 Εξαρτήσεις

Η ανάπτυξη της εφαρμογής έγινε στο λειτουργικό σύστημα Linux και σε αυτό έχουν γίνει και όλες οι δοκιμές, παρόλα αυτά όλα τα προγράμματα, τα modules και οι βιβλιοθήκες

που είναι απαραίτητα για την εκτέλεση της διαθέτουν αντίστοιχες εκδόσεις και για άλλα λειτουργικά συστήματα, όπως τα Windows. Για την εκτέλεση της θα πρέπει να έχουμε εγκατεστημένη στο σύστημα μας την γλώσσα προγραμματισμού Python της σειράς 2.6 και συγκεκριμένα την έκδοση 2.6.5 ή μεγαλύτερη, καθώς και τα python modules Pypcap, Dpkt και Netifaces και την βιβλιοθήκη libpcap. Επιπλέον η εφαρμογή δεν διαθέτει γραφικό περιβάλλον, και η εκτέλεσή της γίνεται στο κέλυφος του λειτουργικού μας συστήματος με δικαιώματα υπερχρήστη (root).

5.2.2 Εκτέλεση

Για να εκτελέσουμε την εφαρμογή πληκτρολογούμε στο κέλυφος την εντολή **python sniffer.py** ακολουθούμενη από κάποιους διακόπτες και παραμέτρους με τους οποίους καθορίζουμε την λειτουργία του προγράμματος και δεδομένου ότι ο τρέχον κατάλογος είναι αυτός που περιέχει την εφαρμογή. Οι διαθέσιμοι διακόπτες είναι:

- **-h** ή **–help**, ο οποίος τυπώνει ένα μήνυμα βοήθειας σχετικά με την χρήση του προγράμματος, όπως φαίνεται στο παράδειγμα 5.1 .
- **-i** ή **–interface**, ο οποίος παίρνει σαν παράμετρο το όνομα της διεπαφής δικτύου που θα χρησιμοποιήσουμε, για παράδειγμα -i eth0 ή –interface=eth0.
- **-f** ή **–filter**, ο οποίος παίρνει σαν παράμετρο μια λογική έκφραση που θα χρησιμοποιηθεί για το φιλτράρισμα των πακέτων που θα συλλάβει το πρόγραμμά μας. Η λογική αυτή έκφραση πρέπει να περικλείεται σε εισαγωγικά, για παράδειγμα -f “src or dst port 80” ή –filter= “src or dst port 80”.
- **-m** ή **–mitm**, ο οποίος ενεργοποιεί την επίθεση MITM με ARP cache poisoning παράλληλα με το packet sniffing σε ξεχωριστή διεργασία. Δέχεται σαν παραμέτρους δύο διευθύνσεις IP χωρισμένες με κενό, οι οποίες αποτελούν τος “στόχους” της επίθεσης μας, για παράδειγμα -m 192.168.1.1 192.168.1.3 ή –mitm=192.168.1.1 192.168.1.3 . Η χρήση του είναι δυνατή μόνο σε δίκτυα που χρησιμοποιούν τα πρωτόκολλα ARP, IPv4 και Ethernet. Επίσης πριν την εκτέλεση του προγράμματος με το συγκεκριμένο διακόπτη, ο χρήστης πρέπει να έχει ενεργοποιήσει στο λειτουργικού σύστημα του Η/Υ την προώθηση των πακέτων μεταξύ των διεπαφών δικτύου, σε αντίθετη περίπτωση τα πακέτα δεν θα προωθηθούν στον αρχικό τους προορισμό. Στο Linux η προώθηση των πακέτων ενεργοποιείται με την εντολή

echo 1 > /proc/sys/net/ipv4/ip_forward .

- **-x** ή **–hexdump**, στην περίπτωση που ορίσουμε αυτόν τον διακόπτη ο οποίος δεν δέχεται κάποια παράμετρο, στην οθόνη τυπώνεται η δεκαεξαδική και η ASCII αναπαράσταση των πακέτων που συλλαμβάνονται από την κάρτα δικτύου μας.
- **-s** ή **–snaplen**, ο οποίος δέχεται σαν παράμετρο το μέγεθος του μέρους κάθε πακέτου που θα συλλάβουμε σε bytes.
- **-w** ή **–dumpfile**, ο οποίος δέχεται σαν παράμετρο το όνομα του pcap αρχείου στο οποίο θα αποθηκευτούν τα πακέτα.
- **-p** ή **–promisc**, ο οποίος δεν δέχεται παραμέτρους και απλά ενεργοποιεί το promiscuous mode στην κάρτα δικτύου μας.

```
# python sniffer.py  --help
Usage: sniffer.py [options]
Options:
  -h, --help            show this help message and exit
  -i INTERFACE, --interface=INTERFACE
                        Network interface to listen (ex. -i eth0).
  -f FILTER, --filter=FILTER
                        Set the capture filter (logical expresion "quoted").
  -m TARGETS, --mitm=TARGETS
                        Perform ARP cache poisoning MITM attack, against two
                        targets (ex. -m 192.168.1.1 192.168.1.4) separated by
                        space.
  -x, --hexdump         Print the packet in hex and ASCII.
  -s SNAPLEN, --snaplen=SNAPLEN
                        Set the capture snapshot length in bytes (default is 65535).
  -w DUMPFILE, --dumpfile=DUMPFILE
                        Dump packets in a file (.pcap).
  -p, --promisc         Enable promiscuous mode for the network interface.
#
```

παράδειγμα 5.1*: Εκτύπωση του μηνύματος βοήθειας σχετικά με την χρήση της εφαρμογής.*

Από τους παραπάνω διακόπτες μόνο ο **-i** είναι απαραίτητος για την εκτέλεση του προγράμματος. Οι διακόπτες που δέχονται τιμές, εάν δεν οριστούν, χρησιμοποιούν κάποιες προκαθορισμένες, συγκεκριμένα ο **-f** είναι κενός, δηλαδή θα συλληφθούν όλα τα πακέτα, ο **-s** έχει προκαθορισμένη τιμή 65535 bytes και το promiscuous mode είναι ανενεργό εάν δεν ορισθεί ο διακόπτης **-p**. Επίσης οι διακόπτες -w και -x δεν μπορούν να χρησιμοποιηθούν ταυτόχρονα, και σε περίπτωση που δεν χρησιμοποιηθεί κανένας από τούς δύο το πρόγραμμά μας τυπώνει μια στοιχειώδης περιγραφή των πακέτων που συλλαμβάνονται. Η περιγραφή περιέχει την χρονοσφραγίδα του πακέτου, το πρωτόκολλο του και τις διευθύνσεις και ports της πηγής και του προορισμού εάν αυτές είναι διαθέσιμες. Τέλος αφού ξεκινήσει η εκτέλεση του προγράμματος, η διαδικασία συνεχίζεται μέχρι να την διακόψει ο χρήστης, αυτό γίνεται πληκτρολογώντας Ctrl+C και το πρόγραμμα τερματίζει την λειτουργία του αφού τυπώσει ορισμένα στατιστικά σχετικά με το πόσα πακέτα καταγράφτηκαν, πόσα απορρίφθηκαν από τον πυρήνα του λειτουργικού συστήματος και πόσα από την κάρτα δικτύου. Στην περίπτωση που έχει ενεργοποιηθεί η επίθεση MITM, πριν το τερματισμό του προγράμματος, αποστέλλονται στους στόχους "νόμιμες" απαντήσεις ARP ώστε να αποκατασταθούν οι εγγραφές στο ARP πίνακα τους και η μεταξύ τους επικοινωνία να συνεχιστεί κανονικά.

```
# python sniffer.py -i eth0
Listening on eth0:

11:53:24 IP(TCP) 192.168.1.3:34225  =>  65.54.189.65:1863
11:53:24 IP(TCP) 65.54.189.65:1863  =>  192.168.1.3:34225
11:53:24 IP(TCP) 192.168.1.3:34225  =>  65.54.189.65:1863
^C
3 packets received by filter
0 packets dropped by kernel
0 packets dropped by interface
#
```

παράδειγμα 5.2*: python sniffer.py -i eth0*

Στο παράδειγμα 5.2 φαίνεται η εκτέλεση του προγράμματος μόνο με διακόπτη **-i eth0** και επομένως καταγράφονται μόνο τα πακέτα από και προς τον Η/Υ (192.168.1.3) στον οποίο εκτελείται.

```
# python sniffer.py -i eth0 -x
Listening on eth0:

  0000:  00 18 39 8b 6a 4c 00 24 8c 5b 11 ec 08 00 45 00        ..9.jL.$.[....E.
  0016:  00 3a d2 b0 40 00 40 11 e2 ab c0 a8 01 03 c3 aa        .:..@.@.........
  0032:  00 01 aa 8a 00 35 00 26 0d 20 73 a5 01 00 00 01        .....5.&. s.....
  0048:  00 00 00 00 00 00 02 69 31 05 79 74 69 6d 67 03        .......i1.ytimg.
  0064:  63 6f 6d 00 00 01 00 01                                com.....

  0000:  00 24 8c 5b 11 ec 00 18 39 8b 6a 4c 08 00 45 00        .$.[....9.jL..E.
  0016:  00 89 c8 ab 00 00 39 11 33 62 c3 aa 00 01 c0 a8        ......9.3b......
  0032:  01 03 00 35 aa 8a 00 75 54 aa 71 5c 81 80 00 01        ...5...uT.q.....
  0048:  00 01 00 01 00 00 02 69 31 05 79 74 69 6d 67 03        .......i1.ytimg.
  0064:  63 6f 6d 00 00 1c 00 01 c0 0c 00 05 00 01 00 00        com.............
  0080:  05 db 00 11 05 79 74 69 6d 67 01 6c 06 67 6f 6f        .....ytimg.l.goo
  0096:  67 6c 65 c0 15 c0 30 00 06 00 01 00 00 01 75 00        gle...0.......u.
  0112:  26 03 6e 73 33 c0 32 09 64 6e 73 2d 61 64 6d 69        &.ns3.2.dns-admi
  0128:  6e c0 32 00 15 90 e3 00 00 03 84 00 00 03 84 00        n.2.............
  0144:  00 07 08 00 00 00 3c                                   ......<
^C
2 packets received by filter
0 packets dropped by kernel
0 packets dropped by interface
#
```

παράδειγμα 5.3*: python sniffer.py -i eth0 -x*

Στην παράδειγμα 5.3 φαίνεται η εκτέλεση του προγράμματος με τους διακόπτες **-i eth0 -x**, έτσι συλλαμβάνονται τα πακέτα από και προς τον Η/Υ στον οποίο εκτελείται το πρόγραμμα και τυπώνεται η δεκαεξαδική και ASCII αναπαράστασή τους.

```
# python sniffer.py -i eth0 -m 192.168.1.1 192.168.1.101 -f "src or dst port 80"
192.168.1.1 => 00:18:39:8b:6a:4c
192.168.1.101 => 00:25:d3:14:d2:75
Poisoning 192.168.1.1 and 192.168.1.101 ...


Listening on eth0:

20:42:28 IP(TCP) 192.168.1.101:54254  =>  74.125.105.217:80
20:42:28 IP(TCP) 192.168.1.101:54254  =>  74.125.105.217:80
20:42:28 IP(TCP) 74.125.105.217:80  =>  192.168.1.101:54254
20:42:28 IP(TCP) 74.125.105.217:80  =>  192.168.1.101:54254
20:42:29 IP(TCP) 192.168.1.101:46683  =>  209.85.227.147:80
20:42:29 IP(TCP) 192.168.1.101:46683  =>  209.85.227.147:80
20:42:29 IP(TCP) 209.85.227.147:80  =>  192.168.1.101:46683
20:42:29 IP(TCP) 209.85.227.147:80  =>  192.168.1.101:46683
20:42:29 IP(TCP) 192.168.1.101:46683  =>  209.85.227.147:80
20:42:29 IP(TCP) 192.168.1.101:46683  =>  209.85.227.147:80
20:42:29 IP(TCP) 192.168.1.101:54612  =>  209.85.227.99:80
20:42:29 IP(TCP) 192.168.1.101:54612  =>  209.85.227.99:80
^C

 MITM attack stopped.

re-ARPing 192.168.1.1 ... Done!

re-ARPing 192.168.1.101 ... Done!


12 packets received by filter
0 packets dropped by kernel
0 packets dropped by interface
#
```

παράδειγμα 5.4*: python sniffer.py -i eth0 -m 192.168.1.1 192.168.1.101 -f "src or dst port 80"*

Στο παράδειγμα 5.4 φαίνεται η εκτέλεση του προγράμματος με τους διακόπτες **-i eth0 -m 192.168.1.1 192.168.1.101 -f "src or dst port 80"**. Πραγματοποιείται επίθεση MITM εναντίον των κόμβων 192.168.1.1 και 192.168.1.101 και καταγράφονται μόνο τα πακέτα που έχουν port πηγής και προορισμού την 80. Από ότι φαίνεται έχουν καταγραφεί και πακέτα από και προς τον κόμβο 192.168.1.101 που σημαίνει ότι η επίθεση MITM λειτούργησε κανονικά.

```
# python sniffer.py -i eth0 -w katagrafi
Listening on eth0:

Writing packets to file katagrafi....

^C
3238 packets received by filter
0 packets dropped by kernel
0 packets dropped by interface
# ls -la
total 2060
drwxr-xr-x 2    dimosch users        4096 Apr 27 20:44 .
drwxr-xr-x 6    dimosch users        4096 Apr 27 20:49 ..
-rw-r--r-- 1    dimosch users        1947 Apr  2 21:53 attacks.py
-rw-r--r-- 1     root    root        1563 Apr 23 11:40 attacks.pyc
-rw-r--r-- 1    dimosch users        6039 Apr 21 20:18 functs.py
-rw-r--r-- 1    root    root        4862 Apr 23 11:40 functs.pyc
-rw-r--r-- 1    root    root     2067414 Apr 27 20:45 katagrafi
-rwxr-xr-x 1    dimosch users        3201 Apr 21 09:52 sniffer.py
#
```

παράδειγμα 5.5*: python sniffer.py -i eth0 -w katagrafi*

Στο παράδειγμα 5.5 φαίνεται η εκτέλεση του προγράμματος με τους διακόπτες **-i eth0 -w katagrafi** , έτσι τα πακέτα αποθηκεύονται στο αρχείο katagrafi. Τυπώνοντας τα περιεχόμενα του τρέχοντος φακέλου διαπιστώνουμε ότι το αρχείο όντως δημιουργήθηκε.

Κεφάλαιο 6:

ΠΡΟΣΤΑΣΙΑ ΚΑΤΑ ΤΗΣ ΚΑΚΟΒΟΥΛΗΣ ΧΡΗΣΗΣ

Εισαγωγή

Η επίθεση MITM με ARP cache poisoning αλλά και το network packet sniffing γενικά μπορούν να χρησιμοποιηθούν από κακόβουλους χρήστες για να υποκλέψουν ευαίσθητα δεδομένα που "ταξιδεύουν" στο δίκτυο και αυτό με την σειρά του μπορεί να οδηγήσει στην παραβίαση των κόμβων του δικτύου. Σε αυτό το κεφάλαιο θα δούμε ορισμένους τρόπους και τεχνικές που μπορεί να εφαρμόσει ο εκάστοτε διαχειριστής του δικτύου ή των κόμβων του δικτύου για την αποτροπή τέτοιου είδους παραβιάσεων.

6.1 Προστασία από την επίθεση ARP cache poisoning

Τα περισσότερα προγράμματα που χρησιμοποιούνται κατά της επίθεσης αυτής, περιορίζονται στην ανίχνευση της επίθεσης και στην ενημέρωση του διαχειριστή του κόμβου, αφήνοντας στον τελευταίο το, όχι πάντα εύκολο, έργο της αντιμετώπισής της. Ένα τέτοιο πρόγραμμα για συστήματα Unix είναι το ARPwatch. Στο παράδειγμα 6.1 φαίνεται ένα απόσπασμα από το αρχείο καταγραφής συμβάντων (log file) του ARPwatch από τον server του LinuxTeam ATEI Λάρισας.

```
Apr 22 14:09:15 linuxteam arpwatch: flip flop 194.42.54.136 0:f:ea:32:ae:22 (0:f:ea:32:53:36) eth0

Apr 22 14:15:46 linuxteam arpwatch: flip flop 194.42.54.136 0:f:ea:32:53:36 (0:f:ea:32:ae:22) eth0

Apr 22 14:16:29 linuxteam arpwatch: flip flop 194.42.54.136 0:f:ea:32:ae:22 (0:f:ea:32:53:36) eth0

Apr 22 14:44:51 linuxteam arpwatch: flip flop 194.42.51.156 0:14:85:dc:2d:be (0:26:18:e1:c5:f3) eth0

Apr 22 14:58:43 linuxteam arpwatch: flip flop 194.42.54.136 0:f:ea:32:53:36 (0:f:ea:32:ae:22) eth0

Apr 22 14:59:11 linuxteam arpwatch: changed ethernet address 194.42.51.161 0:14:85:db:eb:3b
(0:2:b3:8b:24:a4) eth0

Apr 22 15:09:16 linuxteam arpwatch: flip flop 194.42.54.136 0:f:ea:32:ae:22 (0:f:ea:32:53:36) eth0

Apr 22 15:17:08 linuxteam arpwatch: flip flop 194.42.54.136 0:f:ea:32:53:36
```

παράδειγμα 6.1*: Απόσπασμα απο το αρχείο καταγραφής συμβάντων του ARPwatch.*

Για την αυτόματη αντιμετώπιση της επίθεσης χρησιμοποιούνται κάποιες τεχνικές όπως το Dynamic ARP Insepction (DARPI) και το Static ARP Inspection (SARPI). Οι τεχνικές αυτές μπορούν να εφαρμοστούν είτε σε κάθε κόμβο του δικτύου ξεχωριστά, με προγράμματα όπως το ArpON, είτε στο switch του δικτύου.

Κατά το DARPI κάθε πακέτο ARP ελέγχεται για την εγκυρότητα του με βάση τον πίνακα DARPI που διατηρεί ο κόμβος ή το switch και απορρίπτεται εάν δεν είναι έγκυρο. Στην περίπτωση που εφαρμόζεται στον κόμβο, το πρόγραμμα που υλοποιεί την τεχνική δημιουργεί τον πίνακα DARPI με διάφορες μεθόδους, όπως για παράδειγμα απορρίπτοντας όλα τα αιτήματα ARP που δέχεται ο κόμβος στον οποίο το πρόγραμμα εκτελείται και στην συνέχεια στέλνοντας αιτήματα ARP τα οποία έχει κατασκευάσει το ίδιο. Στην περίπτωση που η τεχνική εφαρμόζεται στο switch, δεδομένο ότι ο κατασκευαστής του παρέχει αυτή τη δυνατότητα, ο πίνακας DARPI δημιουργείται κατά το DHCP snooping το οποίο πρέπει να είναι ενεργοποιημένο.

Κατά το SARPI, είτε αυτό εφαρμόζεται σε κάθε κόμβο είτε στο switch του δικτύου, κάθε πακέτο ARP ελέγχεται για την εγκυρότητα του με βάση τον πίνακα SARPI τον οποίο δημιουργεί ο διαχειριστής του κόμβου ή του switch και απορρίπτεται εάν δεν είναι έγκυρο. Αυτό βέβαια, όπως είναι φυσικό καθιστά πιο χρονοβόρα την συντήρηση του δικτύου.

Τα περισσότερα switches που υλοποιούν αυτές της τεχνικές έχουν και ένα επιπλέον χαρακτηριστικό που ονομάζεται port mirroring. Όταν ο διαχειριστής το ενεργοποιήσει, το switch μεταδίδει αντίγραφα από τα πακέτα που φτάνουν σε συγκεκριμένες θύρες, σε κάποια άλλη θύρα του. Έτσι ο διαχειριστής μπορεί να καταγράψει την κίνηση με κάποιο packet sniffer θέτοντας την κάρτα δικτύου του σε promiscuous mode. Φυσικά το port mirroring πρέπει να ενεργοποιείται μόνο για την αντιμετώπιση προβλημάτων ή για εξειδικευμένες διαδικασίες (π.χ. Intrusion Detection), διαφορετικά καθιστά την ασφάλεια του δικτύου ευάλωτη.

6.2 Προστασία από το network packet sniffing

Η πιο αποτελεσματική μέθοδος για την προστασία από το network packet sniffing γενικά είναι η κρυπτογράφηση των συνδέσεων ή καλύτερα των δεδομένων που μεταφέρονται μέσω αυτών, καθώς το ARP cache poisoning δεν αποτελεί το μόνο τρόπο για την επίτευξη μιας επίθεσης MITM. Στην περίπτωση της κρυπτογράφησης μπορεί η παρακολούθηση της κίνησης να είναι ακόμη εφικτή, αλλά τα δεδομένα που θα εκτεθούν στον επιτιθέμενο θα είναι στην ουσία άχρηστα. Η κρυπτογράφηση μπορεί να γίνει με πρωτόκολλα όπως το SSL το οποίο παρέχει ικανοποιητική ασφάλεια. Φυσικά η αποκρυπτογράφηση τους από τον επιτιθέμενο δεν είναι αδύνατη, παρόλα αυτά είναι εξαιρετικά δύσκολη.

Κεφάλαιο 7:

ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΕΡΑΙΤΕΡΩ ΕΞΕΛΙΞΗ

7.1 Συμπεράσματα

Σίγουρα ένα ασφαλές συμπέρασμα μετά την ολοκλήρωση της πτυχιακής εργασίας είναι ότι η Python είναι μια γλώσσα προγραμματισμού πολύ δυνατή και ευέλικτη. Παρόλο που είναι γλώσσα υψηλού επιπέδου, η ανάπτυξη εφαρμογών σχετικά χαμηλού επίπεδου είναι δυνατή χάρη στην επεκτασιμότητά της. Τα παραπάνω ενισχύονται και από το τελικό αποτέλεσμα πού είναι μια εφαρμογή με μέγεθος κάτι λιγότερο από 300 γραμμές κώδικα.

Ακόμα, λόγω έλλειψης προηγούμενης εμπειρίας με την συγκεκριμένη γλώσσα προγραμματισμού, ο κώδικας της εφαρμογής σίγουρα επιδέχεται αρκετές βελτιώσεις. Παρόλα αυτά παραμένει ευανάγνωστος, λόγω του σχεδιασμού της Python με έμφαση στο συγκεκριμένο χαρακτηριστικό. Επίσης η εξοικείωση με την Python είναι σχετικά εύκολη ακόμα και για κάποιον χωρίς προηγούμενη εμπειρία.

Τέλος, το γεγονός ότι η Python αποτελεί ελεύθερο λογισμικό όπως και η ύπαρξη εκτενούς τεκμηρίωσης, περιορίζει το συνολικό κόστος της υλοποίησης, στις αμέτρητες ώρες πειραματισμού, μελέτης και αναζήτησης.

7.2 Περαιτέρω εξέλιξη

Η τελική εφαρμογή θα διατεθεί υπό την άδεια χρήσης GPLv3, η οποία αποτελεί μια άδεια χρήσης open source λογισμικού. Αυτό γίνεται με το σκεπτικό της μελλοντικής εξέλιξης της σε ένα open source project σε περίπτωση εκδήλωσης ενδιαφέροντος από τα μέλη της κοινότητας ελεύθερου λογισμικού. Πιθανοί μελλοντικοί στόχοι είναι η βελτίωση του κώδικα και η ενσωμάτωση επιπλέον χαρακτηριστικών όπως:

- Υλοποίηση επιπλέον επιθέσεων τύπου MITM για τα πρωτόκολλα Ethernet και IPv4.
- Υλοποίηση επίθεσης τύπου MITM σε δίκτυα που χρησιμοποιούν την έκδοση 6 του πρωτοκόλλου IP και το πρωτόκολλο ND (Neighbor Discovery Protocol) για την επίλυση των διευθύνσεων, βασισμένη σε κάποια κενά ασφαλείας του τελευταίου.
- Δυνατότητα αλλοίωσης των συλληφθέντων πακέτων πριν την προώθηση τους με την χρήση φίλτρων (active packet sniffing)
- Δυνατότητες αποκωδικοποίησης των συλληφθέντων πακέτων και ιδιαίτερα των ευαίσθητων δεδομένων που πιθανών να περιέχουν, όπως κωδικούς πρόσβασης.
- Δημιουργία γραφικού περιβάλλοντος ώστε να είναι ευκολότερη η χρήση.

Βιβλιογραφία

Βιβλία

- Magnus Lie Hetland “Beginning Python: From Novice to Professional, Second Edition”, εκδόσεις Apress 2008, ISBN13: 978-1-59059-982-2
- Mark Lutz “Learning Python, Third Edition”, εκδόσεις O'Reilly 2008, ISBN-13: 978-0-596-51398-6
- Noah Gift & Jeremy M. Jones “Python for Unix and Linux System Administration”, εκδόσεις O'Reilly 2008, ISBN: 978-0-596-51582-9
- John Goerzen “Foundations of Python Network Programming”, εκδόσεις Apress 2004, ISBN-13: 978-1590593714
- Andrew S. Tanenbaum “Computer Networks, Fourth Edition”, εκδόσεις Prentice Hall 2003, ISBN-13: 978-0130661029

Ιστοσελίδες

- http://www.python.org
- http://docs.python.org/
- http://en.wikipedia.org/wiki/Python_%28programming_language%29
- http://code.google.com/p/dpkt/
- http://code.google.com/p/pypcap/
- http://www.gradstein.info/python/how-to-understand-the-arp-queries-and-replies-fields-with-pypcap/
- http://alastairs-place.net/netifaces/
- http://www.tcpdump.org/
- http://www.wireshark.org/
- http://www.gentoo.org/
- http://www.nano-editor.org/

- http://en.wikipedia.org/wiki/Packet_analyzer
- http://en.wikipedia.org/wiki/Ethernet_hub
- http://en.wikipedia.org/wiki/Promiscuous_mode
- http://en.wikipedia.org/wiki/Network_switch
- http://www.cisco.com/warp/public/473/lan-switch-cisco.pdf
- http://en.wikipedia.org/wiki/Man-in-the-middle_attack
- http://en.wikipedia.org/wiki/ARP_spoofing
- http://diablohorn.files.wordpress.com/2008/10/arp_poisoning_in_practice.pdf
- http://www.oxid.it/downloads/apr-intro.swf
- http://ettercap.sourceforge.net/
- http://en.wikipedia.org/wiki/Address_Resolution_Protocol
- http://tools.ietf.org/html/rfc826
- http://www.tildefrugal.net/tech/arp.php
- http://freequaos.host.sk/arpwatch/
- http://in.pycon.org/2009/smedia/slides/pycon_sec_.odp
- http://arpon.sourceforge.net/
- http://www.ciscopress.com/articles/article.asp?p=1181682&seqNum=8
- http://en.wikipedia.org/wiki/Encryption
- http://en.wikipedia.org/wiki/Transport_Layer_Security
- http://www.openssl.org/
- http://en.wikipedia.org/wiki/OpenSSL
- http://www.gnu.org/licenses/gpl.html