2012

Πτυχιακή Εργασία

Αλγοριθμικές Τεχνικές Γραφημάτων Υδατοσήμανσης Λογισμικού

Όνομα: **Χιόνης Ιωάννης**

Επιβλέπων καθηγητής: **Σταύρος Δ. Νικολόπουλος**

Εκπονήθηκε στο: **Τμήμα Πληροφορικής του Πανεπιστημίου Ιωαννίνων**

Ιωάννινα, Φεβρουάριος 2012

Αλγοριθμικές Τεχνικές Γραφημάτων Υδατοσύμανσης Λογισμικού

Η ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

υποβάλλεται στην ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης του Τμήματος Πληροφορικής Εξεταστική Επιτροπή

από τον

Ιωάννη Χιόνη

ως μέρος των υποχρεώσεων για την λήψη του

ΠΤΥΧΙΟΥ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

Φεβρουάριος 2012

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Σταύρο Δ.Νικολόπουλο για τις πολύτιμες συμβουλές που μου παρείχε καθ' όλη την διάρκεια εκποίησης της πτυχιακής μου εργασίας. Η καθοδήγησή του ήταν άμεση, πάντα επί του θέματος και καθοριστική, δίχως αοριστολογίες.

Επιπλέον θα ήθελα να ευχαριστήσω τα μέλη της τριμελούς επιτροπής, τον Αναπληρωτή Καθηγητή κ. Λεωνίδα Παληό και τον Επίκουρο Καθηγητή κ. Χρήστο Νομικό για την επιστημονική συμβουλή τους καθώς και για τον χρόνο που διέθεσαν καθ' όλη την διάρκεια της παρουσιάσεως της πτυχιακής εργασίας.

Τέλος, θα ήθελα να ευχαριστήσω την Υποψήφια Διδάκτωρ κ. Μαρία Χρόνη για τις εύστοχες παρατηρήσεις της κατά τη διάρκεια δημιουργίας της τελικής παρουσίασης.

Ιωάννης Χιόνης
Ιωάννινα, 2012

Πίνακας περιεχομένων….

1

Εισαγωγή

1.1 Τι είναι τα υδατογραφήματα λογισμικού

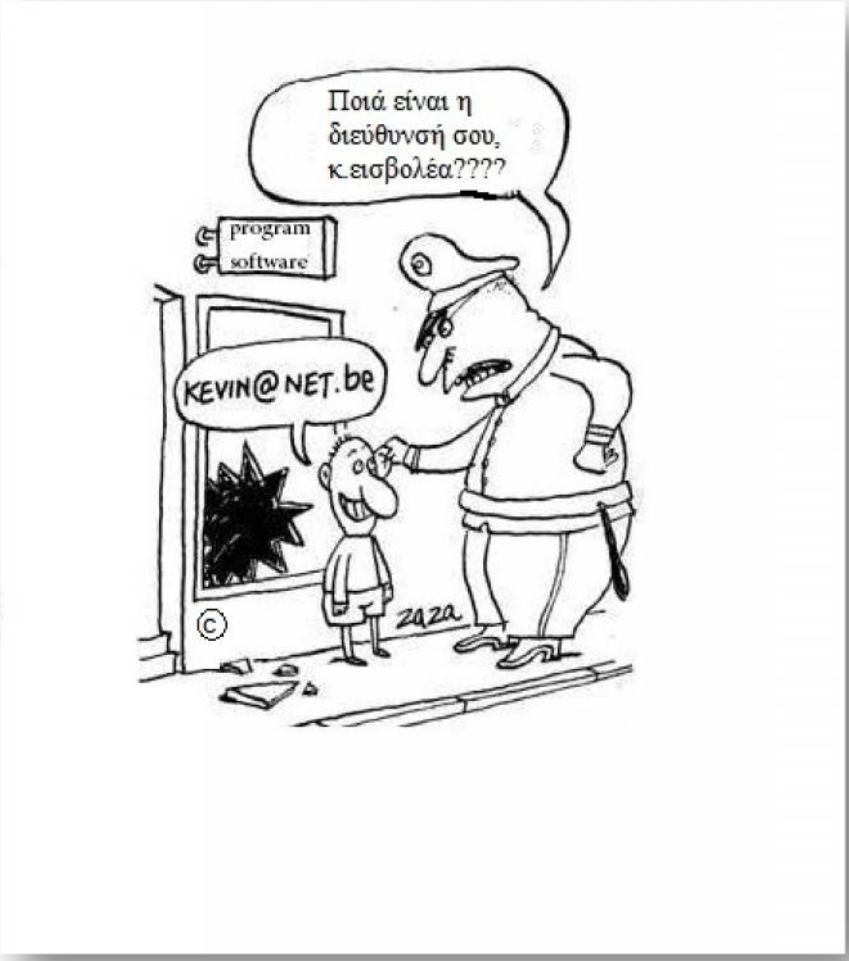
Υδατογράφημα, είναι ένα αναγνωριστικό που εισάγει κάποιος σε ένα αντικείμενο με σκοπό να το προστατεύσει από τυχόν υποκλοπές που ενδεχομένως υποστεί στο μέλλον. Υδατογραφήματα μπορούν να λάβουν χώρα σε διάφορα αντικείμενα, όπως διαφόρου τύπου έγγραφα, λογισμικά, χρήματα, νομίσματα, εικόνες ακόμη και βίντεο. Πιο συγκεκριμένα το αναγνωριστικό (υδατογράφημα), που ενσωματώνεται και παράλληλα υδατογραφεί το εκάστοτε αντικείμενο μπορεί να λάβει διάφορες μορφές, όπως εικόνας, αριθμού, γραφήματος, πίνακα καθώς και διάφορες άλλες έχοντας πάντα ως κύριο στόχο την προστασία του αντικειμένου όπου και ενσωματώνεται. Από την άλλη πλευρά όμως, ένα υδατογράφημα μπορεί επίσης να μην ακολουθεί την έννοια της προσθήκης ενός αντικείμενου σε ένα ή περισσότερα σημεία, (για παράδειγμα πρόσθεση – ενσωμάτωση γραφήματος σε ένα λογισμικό).

Το τελευταίο είδος υδατογραφημάτων επικεντρώνεται σε αλλαγές που μπορούν να γίνουν στο αντικείμενο προς υδατογράφηση, χωρίς βέβαια να το αλλοιώνουν στο ελάχιστο κάνοντας κάποιες σκόπιμες αλλαγές στο ίδιο το αντικείμενο, με αποτέλεσμα να το χαρακτηρίζουν μοναδικά προστατεύοντας το στον ίδιο περίπου βαθμό, όπως κάνουν και τα προηγούμενα είδη υδατογραφημάτων. Ένα παράδειγμα, της τελευταίας μορφής που μπορεί να λάβει ένα υδατογράφημα είναι οι περιστροφές που μπορούν να γίνουν σε κόμβους από δέντρα που δημιουργούνται καθώς εκτελείται ένα πρόγραμμα γραμμένο σε γλώσσα προγραμματισμού Java (τα δέντρα αυτά είναι γνωστά με τον αγγλικό όρο bytecode και περιέχουν εντολές που εκτελεί το λογισμικό το οποίο είναι γραμμένο στη γλώσσα αυτή).

Τα υδατογραφήματα λογισμικού ακολουθούν την γενική κουλτούρα των υδατογραφημάτων, η οποία όπως αναφέρθηκε δεν είναι άλλη από την αποτροπή υποκλοπών. Είναι δηλαδή μία τεχνική που έχει εφευρεθεί για να μας βοηθά να ενσωματώσουμε ένα μοναδικό αναγνωριστικό (για παράδειγμα έναν φυσικό αριθμό ή είναι μήνυμα copyright ή μία εικόνα ή οτιδήποτε άλλο μπορεί να χαρακτηρίσει σε νόμιμο επίπεδο μία οντότητα) σε ένα λογισμικό για την αποθάρρυνση της υποκλοπής του.

Στο σημείο όμως αυτό, αξίζει να σημειωθεί πως η υδατογράφηση λογισμικού δεν εμποδίζει την κλοπή του εκάστοτε λογισμικού αλλά, αποθαρρύνει τους κακόβουλος χρήστες να δράσουν, παρέχοντας ένα αποδεικτικό μέσο το οποίο μπορεί να χρησιμοποιήσει ο αληθινός ιδιοκτήτης σε μία πιθανή διαμάχη στα δικαστήρια, προκειμένου να αποδείξει την γνησιότητα του πράγματος.

Παρακάτω παρατίθενται μία σατιρική εικόνα σχετική με την παραβίαση λογισμικών από εισβολείς:



Το κρυφό υδατογράφημα μπορεί να εξαχθεί είτε, αμέσως μετά της διαδικασίας υδατογράφησης είτε, σε μεταγενέστερη ημερομηνία με την χρήση ενός προγράμματος

αναγνώρισης, αλλιώς (αν η εν λόγω εξαγωγή δεν είναι εφικτή) η δημιουργία του είναι μάταια. Προφανώς το λογισμικό το οποίο αναγνωρίζει το υδατογράφημα το οποίο με την σειρά του δίνει σαν αποτέλεσμα το τελικό υδατόσημα (για παράδειγμα υδατόσημα είναι ένας φυσικός αριθμός ο οποίος έχει δεχθεί μία πληθώρα αλλαγών καταλήγοντας σε ένα κατευθυνόμενο γράφημα το οποίο και αρχικά εντοπίζει το λογισμικό αναγνώρισης) δεν ενσωματώνεται με το υδατογραφημένο πρόγραμμα για τετριμμένους λόγους, αλλά συνδέεται με αυτό κατά την διαδικασία εξαγωγής - αναγνώρισης.

1.2 Στάδια μεταγλώττισης που προσφέρονται για εισαγωγή υδατογραφημάτων λογισμικού

Στο σημείο αυτό, είναι χρήσιμο να εξηγήσουμε ποια είναι τα στάδια της μεταγλώττισης ενός προγράμματος δίνοντας με αυτόν τον τρόπο τα μέρη στα όποια θα μπορούσε κάποιος να εισάγει κάποιο υδατογράφημα. Παράλληλα τα μέρη αυτά προσφέρονται και για την εξαγωγή του εκάστοτε υδατογραφήματος (ανεξάρτητα σε ποιο στάδιο έχει γίνει η ενσωμάτωση), καθώς όπως έχει ήδη προαναφερθεί είναι άσκοπη η υδατογράφηση λογισμικού χωρίς την δυνατότητα εξαγωγής του υδατογραφήματος.

Αρχικά το μη υδατογραφημένο πρόγραμμα είναι γραμμένο σε μία γλώσσα προγραμματισμού υψηλού επιπέδου (γλώσσα προγραμματισμού υψηλού επιπέδου είναι μία γλώσσα ευκόλως κατανοητή από τον άνθρωπο, όπως η γλώσσα C++ ή η Java). Εδώ είναι και το πρώτο σημείο στο μπορεί να εισάγει κάποιος το υδατογράφημα, το σημείο αυτό είναι ο πηγαίος κώδικας. Με τον τρόπο αυτό, αυτός που εισάγει το υδατογράφημα δεν χρειάζεται να μεταγλωττίσει το πρόγραμμα και ύστερα να το ενσωματώσει, αλλά το εισάγει απευθείας στον πηγαίο κώδικα του λογισμικού. Στην συνέχεια, σε περίπτωση που αυτός επιθυμεί να εξάγει το υδατογράφημά του, αυτό που χρειάζεται είναι να συνδέσει τον αναγνωριστή με το πρόγραμμα, δηλαδή ή στον πηγαίο κώδικα ή στον κώδικα Assembly που δημιουργείται μετά την διαδικασία της μεταγλώττισης ή διερμηνείας αντίστοιχα ή ακόμα και στο εκτελέσιμο του προγράμματος προκειμένου να το εξάγει - αναγνωρίσει.

Μετέπειτα, το πρόγραμμα αυτό μεταγλωττίζεται ή διερμηνεύεται ανάλογα με το ποια γλώσσα προγραμματισμού έχει χρησιμοποιηθεί (για παράδειγμα στην γλώσσα C χρησιμοποιείται μεταγλώττιση, ενώ στην Haskell χρησιμοποιείται διερμηνεία) και παράγεται ένα πρόγραμμα σε γλώσσα χαμηλού επιπέδου, περνώντας πρώτα από τα προκαθορισμένα στάδια της μεταγλώττισης ή διερμηνείας τα οποία είναι η λεκτική ανάλυση, η συντακτική ανάλυση, η σημασιολογική ανάλυση, ο ενδιάμεσος κώδικας, ο πίνακας συμβόλων και ο τελικός κώδικας. Ένα πρόγραμμα σε μία γλώσσα χαμηλού επιπέδου (γλώσσα χαμηλού επιπέδου είναι μία γλώσσα ευκόλως κατανοητή από τον ηλεκτρονικό υπολογιστή), η οποία είναι γνώστη ως γλώσσα Assembly είναι το δεύτερο και πιο συνηθισμένο σημείο στο οποίο μπορεί να εισαχθεί ένα υδατογράφημα καθώς και να συνδεθεί ο αναγνωριστής για να το εξάγει. Πιο συγκεκριμένα, το σημείο αυτό βρίσκεται σε χαμηλό επίπεδο, πράγμα που σημαίνει πως τόσο η εισαγωγή του υδατογραφήματος, όσο και η εξαγωγή του γίνεται πιο δύσκολα. Σημειώνεται επίσης, πως η εξαγωγή του υδατογραφήματος μπορεί να γίνει και σε διαφορετικό στάδιο από ότι είναι η γλώσσα χαμηλού επιπέδου. Για παράδειγμα μπορεί να εξαχθεί είτε από τον πηγαίο κώδικα, με την βοήθεια βέβαια εντολών που πραγματοποιούν την μεταφορά από την γλώσσα Assembly στον πηγαίο κώδικα, είτε από το εκτελέσιμο, το

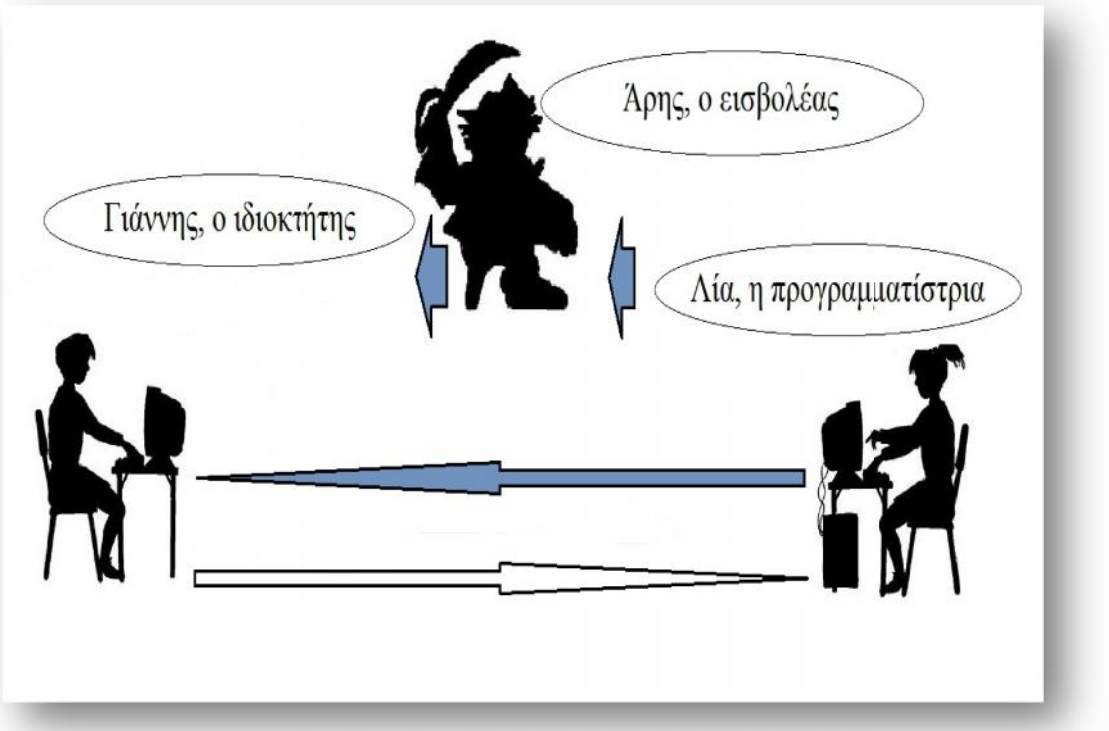
οποίο είναι το χαμηλότερο επίπεδο την διαδικασίας μεταγλώττισης προγράμματος και βρίσκεται αμέσως μετά την γλώσσα χαμηλού επιπέδου.

Τέλος, η μεταγλώττιση φτάνει στο τελικό της στάδιο το οποίο δεν είναι άλλο από το δυαδικό 0 1, όπου είναι και το τελευταίο σημείο που μπορεί κάποιος να εισάγει ή να εξάγει ένα υδατογράφημα, με την προϋπόθεση βέβαια ότι είναι ιδιαίτερα εξοικειωμένος με τέτοιου είδους μορφές αρχείων, καθώς σε κάθε άλλη περίπτωση είναι πολύ πιθανόν να μην καταφέρει τίποτα καταστρέφοντας ταυτόχρονα το λογισμικό. Σημειώνεται επίσης στο σημείο αυτό, πως η εξαγωγή μπορεί να γίνει και στα προηγούμενα δύο υψηλότερα επίπεδα, παρόλο που η εισαγωγή έγινε στην εκτελέσιμη μορφή.

Σε κάθε ένα από τα στάδια εισαγωγής και εξαγωγής που αναφέρθηκαν υπάρχουν ειδικά προγράμματα και εντολές που βοηθούν να ανοίξουν προς επεξεργασία το πρόγραμμα στην εκάστοτε φάση μεταγλώττισης και στην συνέχεια να το ξαναφτιάξουν στην μορφή του εκτελέσιμου που είναι και αυτή που έχει στα χέρια του ο πιθανός κακόβουλος χρήστης. Τέτοια προγράμματα είναι για παράδειγμα, ο text editor για τον πηγαίο κώδικα και ο hex editor για κάθε είδους εκτελέσιμο τα οποία προσφέρονται σε γραφικό περιβάλλον. Όπως επίσης υπάρχουν και εντολές (μη γραφικό περιβάλλον) για να ανοιχτεί το εκτελέσιμο πρόγραμμα σε διάφορα format (δυαδικό, οχταδικό, δεκαδικό, δεκαεξαδικό) με την βοήθεια βέβαια τερματικού για να δούμε τι ακριβώς περιέχει μέσα, αυτές είναι οι od και η xdd με τα διάφορα ορίσματά τους, με την διαφορά πως η πρώτη δίνει τα αποτελέσματα με ένα είδος προσέγγισης. Τα προαναφερθέντα προγράμματα και εντολές ισχύουν για το λειτουργικό σύστημα των Linux-os, για άλλα λειτουργικά συστήματα υπάρχουν προγράμματα παρόμοια με τα προηγούμενα.

Κάτι που δεν έχει ειπωθεί ως τώρα, είναι πως το υδατογράφημα δεν πρέπει σε καμία περίπτωση να αλλάζει την ροή εκτέλεσης του μη υδατογραφημένου προγράμματος καθώς και τον χρόνο εκτέλεσής του. Με άλλα λόγια, αν ένα πρόγραμμα εκτελείται σε γραμμικό χρόνο πριν την υδατογράφηση του, πρέπει να συνεχίσει να εκτελείται σε γραμμικό χρόνο και μετά το πέρας αυτής, σε αντίθετη περίπτωση θα κινήσει υποψίες από τους εκάστοτε εισβολείς και ίσως να μην είναι πλέον χρήσιμο λόγω μεγάλης καθυστέρησης.

Παρακάτω παρατίθεται μία εικόνα σχετική με την επισκόπηση της διαδικασίας υδατογραφήσεως λογισμικού:



Στην εικόνα αυτή, ο Γιάννης είναι ο πελάτης που γράφει το πρόγραμμα και στην συνέχεια απευθύνεται στην Λία για να το υδατογραφήσει, προστατεύοντάς το από διάφορους εισβολείς όπως ο Άρης. Ο Άρης είναι ο κακόβουλος χρήστης που υποκλέπτει το πλέον υδατογραφημένο λογισμικό.

1.3 Λόγοι για τους οποίους τα υδατογραφήματα λογισμικού είναι χρήσιμα

Ο λόγος για τον οποίον τα υδατογραφήματα λογισμικού είναι σημαντικά και πλέον καθιστούν ένα αναπόσπαστο κομμάτι των περισσότερων, αν όχι όλων, των λογισμικών που βγαίνουν στην αγορά είναι ένας και μοναδικός. Η ικανότητα να αποδείξει ο νόμιμος ιδιοκτήτης (είτε αυτός είναι ένα μεμονωμένο άτομο, είτε είναι ολόκληρη εταιρία) πως του ανήκει ένα συγκεκριμένο λογισμικό μέσω κάποιων ειδικών προγραμμάτων.

Στο σημείο αυτό, ένα παράδειγμα θα διαφώτιζε καλύτερα και σίγουρα πιο ευχάριστα τον προαναφερθέν λόγο. Μία εταιρία λογισμικού λοιπόν, εν ονόματι Soft, η οποία ασχολείται με τα διαδικτυακά ηλεκτρονικά παιχνίδια έχει εφεύρει έναν αλγόριθμο σύμφωνα με τον οποίον οι κινήσεις μέσα στο παιχνίδι γίνονται με πολύ ομαλό και ταυτόχρονα έξυπνο τρόπο, συγκριτικά με τα ήδη υπάρχοντα παιχνίδια της αγοράς, ο οποίος τείνει στα πραγματικά δεδομένα. Αυτόν τώρα τον αλγόριθμο θέλει να τον ενσωματώσει σε κάθε της παιχνίδι, διότι με τον τρόπο αυτόν θα προσελκύσει πολλά καινούρια άτομα (κάνοντας πρώτα την κατάλληλα διαφήμιση), πετυχαίνοντας έτσι μεγαλύτερο κέρδος για αυτήν και καθιστώντας την παράλληλα ως την εταιρία με τα πιο έξυπνα γραφικά. Για την ιστορία αξίζει να σημειωθεί πως στον αλγόριθμό της έχει εισάγει μεθόδους ευφυής βελτιστοποίησης, οι οποίοι έχουν την ικανότητα με την πάροδο του χρόνου να ‘μαθαίνουν’ μόνοι τους, έχοντας βέβαια αρχικά τα κατάλληλα δεδομένα. Με τον τρόπο αυτό, όταν εφαρμοστεί ο αλγόριθμος σε ένα παιχνίδι, έχοντας δεχτεί πρώτα τις κατάλληλες τροποποιήσεις να προσομοιώνει το

περιβάλλον του παιχνιδιού στα πραγματικά δεδομένα κάτι που είναι ο πρώτος στόχος στις σημερινές εταιρίες ηλεκτρονικών παιχνιδιών. Όμως η εταιρία αυτή γνωρίζει πολύ καλά πως με αυτήν της την κίνηση, θα προσελκύσει και το ενδιαφέρον πολλών κακόβουλων χρηστών, οι οποίοι έχουν ως στόχο να κατορθώσουν να βρουν έναν τρόπο και να υποκλέψουν τον εν λόγω αλγόριθμο, ο οποίος βέβαια τώρα έχει πάρει την μορφή προγράμματος και να τον χρησιμοποιήσουν για δικούς τους σκοπούς δίχως βέβαια να γίνουν αντιληπτοί με αυτήν τους την κίνηση.

Τα άτομα αυτά, σε περίπτωση που είναι ιδιαίτερα εξοικειωμένα σε τέτοιες καταστάσεις μπορούν είτε με εύκολο, είτε με δύσκολο τρόπο να υποκλέψουν αυτό που επιθυμούν χωρίς να γίνουν εκείνη την στιγμή αντιληπτοί. Στο σημείο όμως αυτό, εισέρχεται η συνεισφορά των υδατογραφημάτων λογισμικού, η οποία σε τέτοιες καταστάσεις είναι όχι μόνο χρήσιμή αλλά και αναγκαία.

Σε περίπτωση που το λογισμικό, το οποίο έχει υποκλαπεί δεν έχει υποστεί υδατογράφηση, οι άνθρωποι που ενεπλάκησαν στην κλοπή μπορούν ανενόχλητοι να χρησιμοποιήσουν τον αλγόριθμο σε δικά τους λογισμικά δίχως να πληρώσουν τίποτα για τα πνευματικά δικαιώματα των νόμιμων ιδιοκτητών, όπου στην συγκεκριμένη περίπτωση είναι η εταιρία Soft και δίχως να υποστούν κυρώσεις από τον οποιονδήποτε. Ακόμη και σε περίπτωση όπου η εν λόγω κλοπή, γίνει αντιληπτή από την εταιρία, αυτή δεν είναι σε θέση να αποδείξει πως ο αλγόριθμος που έχει χρησιμοποιηθεί σε άλλα λογισμικά είναι δικός της. Με άμεσο αποτέλεσμα της ιστορίας αυτής, να χάσει το μονοπώλιο στον τομέα αυτόν, χάνοντας παράλληλα και πολλούς εν δυνάμει πελάτες από τους οποίους και θα αποκόμιζε και υψηλότερα κέρδη.

Όμως, αν η εταιρία είχε προνοήσει πριν βγάλει το λογισμικό και ταυτόχρονα τον αλγόριθμο της στο εμπόριο και το είχε υδατογραφήσει με κάποια έξυπνη τεχνική θα ήταν στην ευχάριστη θέση όταν γίνονταν αντιληπτή η κλοπή να προσφύγει στα δικαστήρια και με εύκολο και γρήγορο τρόπο, να αποδείξει έναντι της δικαιοσύνης, πως αυτή είναι η νόμιμος ιδιοκτήτης του αλγορίθμου και της ανήκουν εξ ολοκλήρου όλα τα πνευματικά δικαιώματα.

Στο σημείο αυτό αξίζει να σημειωθεί πως η υδατογράφηση του λογισμικού της θα μπορούσε να γίνει με έναν δυναμικό αλγόριθμο υδατογράφησης, ενσωματώνοντας ένα συγκεκριμένο υδατογράφημα. Το υδατογράφημα αυτό θα μπορούσε να είναι αρχικά ένα κατευθυνόμενο γράφημα, το οποίο είχε κρυπτογραφήσει με κατάλληλους αλγορίθμους έναν φυσικό αριθμό. Στην συνέχεια, αυτό το κατευθυνόμενο γράφημα θα το είχε μετατρέψει με έναν κατάλληλο και ορθό τρόπο σε ένα διάγραμμα ελέγχου ροής (ο αγγλικός όρος είναι control flow graph) και τέλος με εμπεριστατωμένη μελέτη του αρχικού κώδικα του λογισμικού θα το είχε ενσωματώσει μέσα του με τέτοιον τρόπο, έτσι ώστε η εξαγωγή του να ήταν, ως επί το πλυστών βέβαια, εφικτή μόνο μέσω ενός ειδικού αλγορίθμου εξαγωγής υδατογραφήματος.

Είναι αναγκαίο να τονιστεί πως και ο αλγόριθμος υδατογράφησης, ο οποίος ενσωματώνει ένα συγκεκριμένο υδατογράφημα και ο αλγόριθμος εξαγωγής του εν λόγω υδατογραφήματος αποτελούν αμφότεροι μέρος της διαδικασίας υδατογράφησης λογισμικού, καθώς το να ενσωματώσει κάποιος ένα αναγνωριστικό σε ένα λογισμικό και στην συνέχεια να μην είναι σε θέση να το εντοπίσει και να το εξάγει δεν αποτελεί ικανή και ορθή μέθοδο υδατογράφησης λογισμικού. Αυτό συμβαίνει προφανώς, διότι δεν μπορεί να αποδείξει κανείς

πως μέσα σε ένα λογισμικό βρίσκεται τοποθετημένο ένα συγκεκριμένο υδατογράφημα, αν δεν είναι θέση πρώτα να το εξάγει.

Με τον τρόπο αυτόν, η εταιρία αυτή έχοντας πρώτα υδατογραφήσει το λογισμικό της θα μπορέσει να προβεί σε απόδειξη ύπαρξης πνευματικών δικαιωμάτων (δηλαδή, πως αυτή είναι ο πραγματικός ιδιοκτήτης) και εν συνεχεία να απαιτήσει από αυτούς που ευθύνονταν για την κλοπή, η οποία και έγινε εις βάρος της, πρώτον την απόσυρση των λογισμικών που δημιουργήθηκαν με την βοήθεια του αλγόριθμού της και δεύτερον, μία γερή αποζημίωση για την ζημία που υπέστη.

Παρακάτω παρατίθεται μία εικόνα, στην οποία σκιαγραφείται από την σατιρική πλευρά το αποτέλεσμα μίας επίθεσης ενός εισβολέα σε ένα λογισμικό υδατογραφημένο με μεταθετικό αναγώγιμο γράφημα:

Από την άλλη πλευρά όμως, έχοντας τελειώσει με το προηγούμενο παράδειγμα, αξίζει να τονιστεί πως ανέκαθεν υπήρχαν άτομα τα οποία έκαναν πράγματα προς όφελος του κοινού καλού. Έτσι και στον τομέα της πληροφορικής υπάρχουν άτομα τα οποία δημιουργούν λογισμικά προς το κοινό συμφέρον, δίχως να απαιτούν για την χρήση και την περεταίρω ανάπτυξη αυτών κάποια πληρωμή, όπως είναι για παράδειγμα το ανοικτό λειτουργικό σύστημα Linux -os, το οποίο αναπτύχθηκε το 1969, από τους Ken Thompson, Dennis Ritchie (δημιουργός και της γλώσσας C), Douglas McIlroy και Joe Ossanna. Αυτό, για την ιστορία ήταν γραμμένο εξολοκλήρου σε γλώσσα Assembly και

ανήκει στην κατηγορία των Unix συστημάτων. Επομένως, η υδατογράφηση σε τέτοιου είδους προγράμματα δεν είναι καθόλου αναγκαία ούτε και απαραίτητη.

Παρόλα αυτά όμως, οι υδατογραφήσεις λογισμικού δεν απαγορεύουν την υποκλοπή. Αυτό που κάνουν είναι να αποθαρρύνουν τους εκάστοτε εισβολείς και να τους προειδοποιούν πως σε πιθανή κλοπή θα βρεθούν αντιμέτωποι με την δικαιοσύνη. Κάτι που από ότι φαίνεται σύμφωνα με τα πραγματικά δεδομένα, δεν είναι και τόσο σημαντικός λόγος για αυτούς, γι αυτό άλλωστε όσο αναπτύσσονται καινούρια λογισμικά τόσο συνεχίζονται και οι υποκλοπές αυτών.

Παρακάτω υπάρχει μία εικόνα η οποία σατιρίζει την δράση των κακόβουλων χρηστών:

1.4 Κατηγορίες υδατογραφημάτων λογισμικού και χρηστικές ιδιότητες αυτών

Τα υδατογραφήματα λογισμικού κατηγοριοποιούνται με βάσει τις ιδιότητές τους σε:

- Εύθραυστα (fragile) και εύρωστα (robust) υδατογραφήματα
- Ορατά (visible) και μη ορατά (invisible) υδατογραφήματα
- Στατικά (static) και δυναμικά (dynamic) υδατογραφήματα
- Ενημερωμένα (informed) και τυφλά(blind) υδατογραφήματα
- Εστιασμένα (focus) και διάχυτα (spread spectrum) υδατογραφήματα

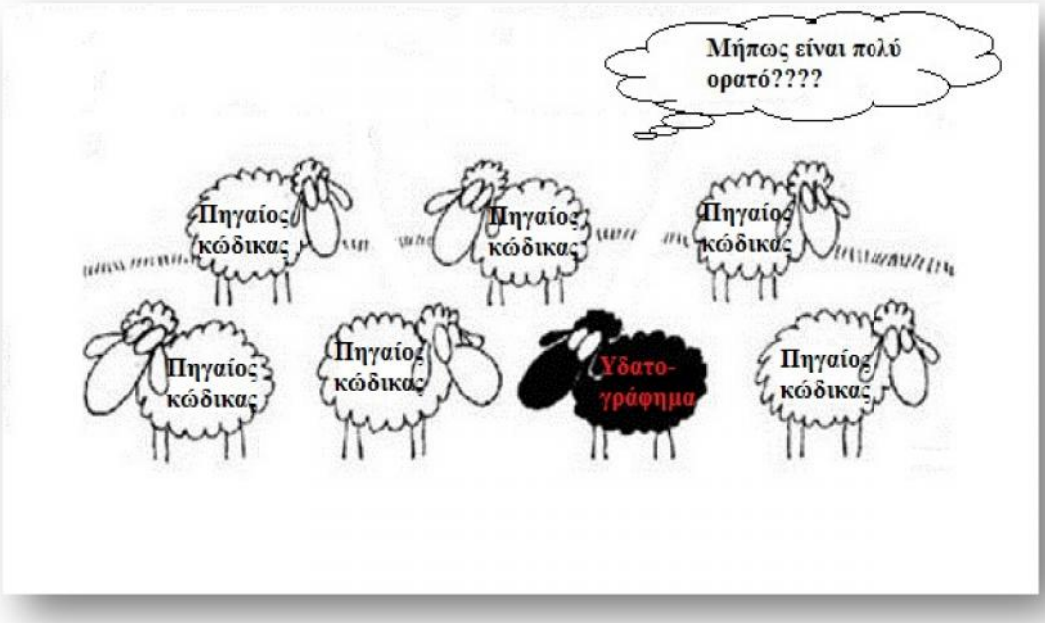
Η πρώτη κατηγορία βασίζεται στην ευρωστία, δηλαδή πόσο αντέχουν στις εκάστοτε επιθέσεις εναντίων τους. Για παράδειγμα σε μερικές εφαρμογές η δύναμη των

υδατογραφημάτων είναι επιθυμητή και πρέπει να μεγιστοποιηθεί, ενώ σε άλλες το επιθυμητό είναι ακριβώς το αντίθετο.

Πιο συγκεκριμένα τα fragile watermarks είναι σχεδιασμένα για να είναι δυσανάγνωστα, δηλαδή εάν κάποιος εισβολέας εφαρμόσει κάποια απαγορευμένη μετατροπή στο υδατογραφημένο πρόγραμμα να μην μπορέσει εύκολα να τα εντοπίσει και να τα εξάγει κάνοντας έτσι το πρόγραμμα μη αναγνωρίσιμο από τον ιδιοκτήτη του. Με αυτόν τον τρόπο καθίστανται ιδιαιτέρως χρήσιμα για την απόδειξη της γνησιότητας ενός προγράμματος. Αξίζει να σημειωθεί πως τα fragile watermarks είναι κάτι ανάλογο των υδατογραφημάτων σε νόμισμα, τα οποία είναι σχεδιασμένα να αναπαράγονται εύκολα όταν αντιγραφούν. Από την άλλη πλευρά, τα robust watermarks είναι κατασκευασμένα να ενσωματώνουν κάποιες πληροφορίες στο πρόγραμμα με τέτοιον τρόπο, έτσι ώστε να είναι πολύ δύσκολο να εντοπιστούν και να καταστραφούν από κάποιον κακόβουλο χρήστη. Με άλλα λόγια, ένα τέτοιου είδους υδατογράφημα θα αντισταθεί στις επιθέσεις και θα συνεχίσει να υπάρχει παρά τις προσπάθειες εναντίον του με στόχο τον εντοπισμό και την εξαγωγή ή ακόμα και την μετάλλαξη αυτού. Οι πληροφορίες που ενσωματώνονται μπορεί να λειτουργήσουν ως ένα αναγνωριστικό ιδιοκτησίας (ένα είδος ταυτότητας) για την αποφυγή πειρατείας. Για παράδειγμα, η ενσωμάτωση ενός αναγνωριστικού το οποίο χαρακτηρίζει τον πελάτη που θα αγοράσει το λογισμικό, θα μπορούσε να είναι το όνομα αυτού. Με τον τρόπο αυτόν αυξάνεται η πιθανότητα μίας παράνομης διανομής να πιαστεί με βάση τον νόμο που επικρατεί.

Η δεύτερη κατηγορία βασίζεται στην ορατότητα ή μη των υδατογραφημάτων. Για παράδειγμα ένα ορατό υδατογράφημα περιγράφει έναν αλγόριθμο υδατοσήμανσης όπου η συνάρτηση αναγνώρισης είναι γνωστή. Έτσι με αυτόν τον τρόπο καθένας μπορεί να δει το υδατογράφημα και να το καταστρέψει εντελώς ή να το αλλάξει με όποιον τρόπο επιθυμεί δημιουργώντας ένα καινούργιο υδατογράφημα μη αναγνωρίσιμο από τον νόμιμο ιδιοκτήτη του προγράμματος. Από την άλλη πλευρά, τα μη ορατά υδατογραφήματα περιγράφουν μία συνάρτηση αναγνώρισης, η οποία δεν είναι γνωστή, παρά μόνο σε αυτόν που δημιουργεί το υδατογράφημα καθιστώντας με αυτόν τον τρόπο δύσκολη την δουλεία κάποιου να το εντοπίσει και να το βγάλει ή ακόμα και να το μεταλλάξει.

Παρακάτω βρίσκεται μία εικόνα στην οποία σατιρίζεται η αποτελεσματικότητα των ορατών υδατογραφημάτων:



Η τρίτη κατηγορία είναι τα στατικά και τα δυναμικά υδατογραφήματα τα οποία διαχωρίζονται σύμφωνα με τον τρόπο που εφαρμόζονται στο πρόγραμμα. Για παράδειγμα τα στατικά ενσωματώνονται είτε στον κώδικα του προγράμματος, είτε στα δεδομένα αυτού και δεν απαιτούν να εκτελεστεί το πρόγραμμα για να για να αναγνωριστούν. Από την άλλη όμως πλευρά, τα δυναμικά υδατογραφήματα δεν αποθηκεύονται στο ίδιο μέρος με τα στατικά. Αυτά τοποθετούνται σε δομές δεδομένων, οι οποίες δημιουργούνται κατά την διάρκεια εκτέλεσης του προγράμματος, ενώ παράλληλα έχουν την δυνατότητα να αλλάζουν κατά την διάρκεια αυτή κάτι που δυσκολεύει την δημιουργία τους, ενώ παράλληλα ανεβαίνει και ένα επίπεδο η δυσκολία εντοπισμού τους. Αυτός θα μπορούσε να είναι και ένας σοβαρός λόγος για να γίνουν πιο ελκυστικά σε πιθανούς πελάτες οι οποίοι επιθυμούν την καλύτερη ασφάλεια για τα προγράμματά τους. Εδώ αξίζει να ειπωθεί πως τα στατικά και τα δυναμικά υδατογραφήματα χωρίζονται και σε περεταίρω κατηγορίες.

Συνεχίζοντας πάνω σε αυτήν την κατηγορία είμαστε σε θέση να πούμε πως τα στατικά υδατογραφήματα, χωρίζονται στα code watermarks τα οποία λαμβάνουν μέρος στο τμήμα του εκτελέσιμου που περιέχει τις εντολές, και στα data watermarks τα οποία λαμβάνουν χώρα σε οποιοδήποτε άλλο μέρος του εκτελέσιμου, όπως στα τμήματα που περιέχουν τις κεφαλίδες, στα τμήματα που περιέχουν τα αλφαριθμητικά, στα τμήματα που περιέχουν πληροφορίες για το debugging ή όπου αλλού είναι εφικτό κρατώντας παράλληλα σωστή την ροή εκτέλεσης του προγράμματος.

Από την άλλη πλευρά όμως, τα δυναμικά υδατογραφήματα, χωρίζονται και αυτά με την σειρά τους στα easter eggs watermarks, στα dynamic data structure watermarks και στα dynamic execution trace watermarks.

Τα easter eggs watermarks, εκτελούν κάποιες λειτουργίες οι οποίες είναι άμεσα αντιληπτές από τον χρήστη κάνοντας την εξαγωγή τους τετριμμένη. Τυπικά αυτό που κάνουν είναι να τυπώνουν ένα μήνυμα copyright ή μία εικόνα στην οθόνη όπου βλέπουμε την

εκτέλεση του προγράμματος. Αυτό όμως καθιστά τέτοιου είδους υδατογραφήματα πολύ εύκολα στο να εντοπιστούν από κάποιον εισβολέα οποίος μπορεί είτε, να τα εντοπίσει και να τα βγάλει τελείως είτε, να τα απενεργοποιήσει, σβήνοντας με τον τρόπο αυτόν την ταυτότητα του νόμιμου χρήστη από το πρόγραμμα.

Τα dynamic data structure watermarks ενσωματώνονται μέσα στο πρόγραμμα είτε, μέσα σε global μεταβλητές είτε, μέσα σε σωρούς είτε, μέσα σε στοίβες από δεδομένα. Το υδατογράφημα εξάγεται, εξετάζοντας τις τρέχον τιμές από τις μεταβλητές στις οποίες έχει ενσωματωθεί μετά το τέλος βέβαια από μία ακολουθία εισόδου. Αυτό μπορεί να λάβει μέρος είτε, φτιάχνοντας μία ειδική ρουτίνα (συνάρτηση) εξόρυξης υδατογραφήματος, η οποία είναι συνδεδεμένη με την εκτέλεση του προγράμματος είτε, εκτελώντας το πρόγραμμα με έναν debugger. Αυτή η κατηγορία υδατογραφημάτων έχει και κάποιες ενδιαφέρουσες ιδιότητες όπως για παράδειγμα, το ότι δεν παράγουν κάποια έξοδος κατά την εκτέλεση του προγράμματος δεν τα κάνει αμέσως αντιληπτά από τον εν δυνάμει εισβολέα που δίνει μία ακολουθία εισόδου στο πρόγραμμα. Με αποτέλεσμα αυτού του είδους τα υδατογραφήματα να είναι της ακριβώς αντίθετης λογικής σε σύγκριση με τα easter eggs watermarks, όπου κάνουν το ακριβώς αντίθετο. Επιπλέον, στο σημείο αυτό αξίζει να σημειωθεί πως η ρουτίνα αναγνώρισης του υδατογραφήματος δεν αποστέλλεται με το εκάστοτε πρόγραμμα, αλλά συνδέεται με αυτό κατά την διάρκεια εκτέλεσης, με αποτέλεσμα να υπάρχει μια μικρή πληροφορία στο εκτελέσιμο για το που μπορεί να είναι τοποθετημένο το υδατογράφημα.

Τελειώνοντας, με αυτήν την υπό κατηγορία έχουμε τα dynamic execution trace watermarks, τα οποία ενσωματώνονται μέσα στο ίχνος (εντολές και διευθύνσεις) πού αφήνει το πρόγραμμα όταν εκτελεστεί μετά από μία σειρά από ακολουθίες εισόδου. Τα υδατογραφήματα αυτά μπορούν να εξαχθούν παρακολουθώντας κάποιες πιθανώς στατιστικές ιδιότητες από τα ίχνη διευθύνσεων ή από κάποια ακολουθία τελεστών οι οποίοι έχουν εκτελεστεί.

Προχωρώντας, πάνω στην αρχική αρίθμηση μεταβαίνουμε στην τέταρτη κατηγορία και στα informed και blind watermarks τα οποία βασίζονται στις πληροφορίες που χρειάζονται για να προσδιορίσουμε τα υδατογράφημα. Δηλαδή, για να εξάγουν το υδατογράφημα οι αναγνωριστές και των informed και των blind watermarks απαιτούν το υδατογραφημένο πρόγραμμα και πιθανώς και ένα κλειδί. Σε αντίθεση με τους αναγνωριστές των blind watermarks, οι informed αναγνωριστές απαιτούν επίσης το μη υδατογραφημένο πρόγραμμα ή σκέτο το υδατογράφημα ή και τα δύο. Το πλεονέκτημα των blind αναγνωριστών είναι ότι δεν χρειάζονται ούτε, το μη υδατογραφημένο πρόγραμμα ούτε, το υδατογράφημα. Δίνοντας ένα παράδειγμα σε αυτό το σημείο, θα μπορούσαμε να πούμε πως εάν ένας πελάτης έχει ζητήσει να υδατογραφηθεί το πρόγραμμα του με ένα informed watermark και κάποιος κακόβουλος χρήστης του κλέψει στην συνέχεια το πρόγραμμα και η όλη διαδικασία φτάσει στο τέλος στα δικαστήρια. Τότε, ο νόμιμος χρήστης χρειάζεται όπως ειπώθηκε και προηγουμένως εκτός του υδατογραφημένου προγράμματος και το μη υδατογραφημένο ή τουλάχιστον το υδατογράφημα το οποίο εισήχθη. Με αποτέλεσμα αν χαθεί κάτι από αυτά και μείνει μόνο το τελικό πρόγραμμα ο εισβολέας μπορεί να υδατογραφήσει εκ νέου το πρόγραμμα καθιστώντας το πλέον ως, εν δυνάμει δικό του αφαιρώντας με τον τρόπο αυτό τα νόμιμα δικαιώματα από τον αρχικό ιδιοκτήτη.

Τέλος μεταβαίνουμε στην πέμπτη κατηγορία και στα focus και spread spectrum watermarks. Αυτά διαχωρίζονται σύμφωνα με τον τρόπο που ενσωματώνονται στο εκάστοτε λογισμικό. Για παράδειγμα ένα focus watermark θα κάνει αλλαγές σε ένα μικρό τμήμα του

λογισμικού, το οποίο θα είναι και η θέση του υδατογραφήματος, ενώ ένα spectrum watermark θα κάνει αλλαγές σε ολόκληρη την έκταση του και όχι σε ένα συγκεκριμένο σημείο.

Έχοντας κάνει λόγο για όλες τις κατηγορίες των υδατογραφημάτων λογισμικού (ο όρος στο αγγλικά είναι software watermarks), είναι αναγκαίο να αναφέρουμε και κάποιες είναι οι χρηστικές ιδιότητες που πρέπει να φέρουν για να μπορούν να λάβουν μέρος στα εκάστοτε λογισμικά.

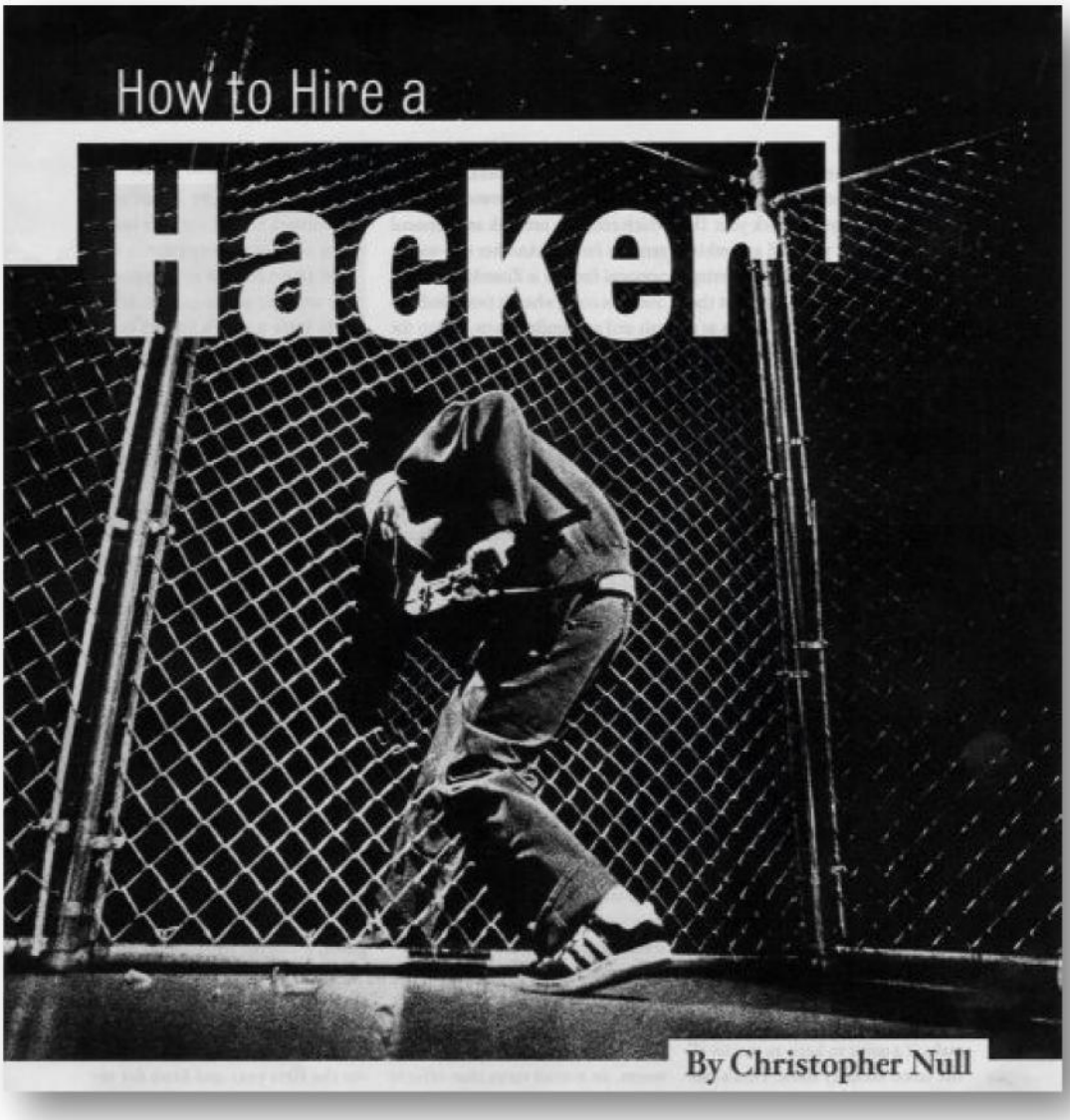
Αρχικά, ένα υδατογράφημα λογισμικού πρέπει να είναι ορθό. Αυτό σημαίνει πως και το υδατογραφημένο πρόγραμμα και το μη υδατογραφημένο πρέπει για οποιαδήποτε ακολουθία εισόδων να δίνουν ακριβώς το ίδιο αποτέλεσμα και σε περίπου τον ίδιο χρόνο.

Επιπλέον, πριν ενσωματωθεί ένα υδατογράφημα μέσα σε ένα πρόγραμμα, χρειάζεται μία μικρή προ επεξεργασία. Αυτό περιλαμβάνει ένα σχολιασμό στο πηγαίο κώδικα για το που μπορεί να εισαχθεί το υδατογράφημα και που όχι (για παράδειγμα ιδιαίτερα βελτιστοποιημένη ή εξαιρετικά εύθραυστη περιοχή κώδικα) και διάφορες πληροφορίες χρήσιμες για την ενσωμάτωση. Στο σημείο αυτό, αξίζει να σημειωθεί πως το κόστος δημιουργίας και ενσωμάτωσης υδατογραφημάτων ποικίλει από κατηγορία σε κατηγορία. Επίσης, κάποια υδατογραφήματα απαιτούν και εκτεταμένη αλληλεπίδραση με τον προγραμματιστή είτε, κατά την διάρκεια της ανάπτυξης είτε, κατά την διάρκεια ενσωμάτωσης.

Τα υδατογραφήματα λογισμικού για να είναι ευρέως αποδεκτά πρέπει να είναι όσο το δυνατών λιγότερο εξαρτημένα από το υλικό των ηλεκτρονικών υπολογιστών και να είναι σε θέση να ενσωματωθούν σε προγράμματα που εκτελούνται σε οποιοδήποτε λογισμικό. Όμως η ενσωμάτωση ενός υδατογραφήματος δεν πρέπει να καθιστά την εκτέλεση του προγράμματος πιο αργή, ή τουλάχιστον η διαφορά στην εκτέλεση του υδατογραφημένου και του μη υδατογραφημένου να είναι σχεδόν ασήμαντη.

Σε αυτό το σημείο πρέπει να σημειωθεί πως η υδατογράφηση λογισμικού είναι μια διαδικασία δημιουργίας κατά την οποία ο δημιουργός του υδατογραφήματος έχει πάρει τις απαραίτητες πληροφορίες από τον ιδιοκτήτη του εκάστοτε προγράμματος για το τι είδους προστασία επιθυμεί να δώσει στο πρόγραμμά του και στη συνέχεια αφού έχει μελετήσει τον κώδικα του προγράμματος διεξοδικά, κατασκευάζει το υδατογράφημα και το ενσωματώνει είτε, απευθείας στον πηγαίο κώδικα του προγράμματος είτε, στο εκτελέσιμο που δημιουργείται μετά την μεταγλώττιση του είτε ακόμα, σε ένα ενδιάμεσο επίπεδο το οποίο εμπεριέχει κώδικα Assembly. Με άλλα λόγια, δεν υπάρχει κάποια μαγική συνταγή που πρέπει να ακολουθείται πιστά για να υδατογραφηθεί ένα πρόγραμμα, αυτό που χρειάζεται είναι γνώσεις προγραμματισμού, γνώσεις υδατογράφησης και άπλετη φαντασία, καθώς ο δημιουργός του υδατογραφήματος πρέπει να μπαίνει στην θέση των κακόβουλων χρηστών και να καταλαβαίνει ποιος είναι ο καλύτερος τρόπος να θωρακίσει το εκάστωτε λογισμικό κλείνοντας όσο το δυνατόν περισσότερες ‘πόρτες’.

Παρακάτω υπάρχει μία σατιρική εικόνα, στην οποία απεικονίζεται ένας κακόβουλος χρήστης που ως δια μαγείας βρίσκει τρόπο να εισέλθει εκεί που θέλει:



1.5 Ιδιότητες προστασίας λογισμικού

Οι ιδιότητες προστασίας των υδατογραφημάτων λογισμικού καθορίζουν άμεσα ή έμμεσα το εκάστοτε υδατογράφημα που λαμβάνει χώρα σε ένα λογισμικό. Οι ιδιότητες αυτές είναι η ανθεκτικότητα στις όποιες επιθέσεις δέχεται είτε, το λογισμικό είτε, το υδατογράφημα που περιέχεται μέσα σε αυτό, η αξιοπιστία και η ικανότητα να μην γίνεται αντιληπτό (ο όρος στο αγγλικά είναι stealth).

Αρχικά, αναλύοντας την κάθε ιδιότητα χωριστά, στην ανθεκτικότητα στόχος είναι η βέλτιστη λειτουργία αναγνώρισης σε υδατογραφημένα αντικείμενα τα οποία έχουν υποστεί ποικίλους μετασχηματισμούς, όπως για παράδειγμα συμπίεση δεδομένων, δεύτερη μεταγλώττιση ή ακόμα και συσκοτίσεις κώδικα. Επιπλέον, ως στόχος μπορεί να θεωρηθεί η υλοποίηση ενός δυσανάγνωστου υδατογραφήματος το οποίο σε περίπτωση που εντοπιστεί από τον εισβολέα να μην του δώσει εύκολα την ευκαιρία να το κατανοήσει και να δράσει πάνω σε αυτό. Με άλλα λόγια το είδος, η πολυπλοκότητα και το κόστος του συνόλου των μετασχηματισμών που καθιστούν ένα υδατογράφημα δυσανάγνωστο είναι ένα μέρος της

ανθεκτικότητας. Αξίζει να σημειωθεί πως ένα υδατογράφημα με αυτήν την ιδιότητα έχει πολύ χαμηλό ποσοστό σε σφάλματα κατά την διαδικασία αναγνώρισης του.

Συνεχίζοντας γίνεται η μετάβαση, στην ιδιότητα της αξιοπιστίας των υδατογραφημάτων λογισμικού. Η ιδιότητα αυτή εκφράζει την πιθανότητα ένα μη υδατογραφημένο λογισμικό να φαίνεται ,λανθασμένα βέβαια, πως περιέχει υδατογράφημα. Υπάρχουν δύο ειδή λογισμικών, τα οποία λανθασμένα πάντα, φαίνονται να περιέχουν υδατογράφημα, τα μη κακόβουλα (ο όρος στα αγγλικά είναι non malicious) και τα επιτιθέμενα (ο όρος στα αγγλικά είναι attacked). Πιο συγκεκριμένα, τα μη κακόβουλα λογισμικά, ξεφεύγουν από τον κίνδυνο κάποιας πιθανής επίθεσης καθώς φαίνονται καθαρά από την άποψη της ύπαρξης πνευματικής ιδιοκτησίας. Με άμεσο αποτέλεσμα του γεγονότος αυτού, να υποκλέπτονται χωρίς ιδιαίτερες μετατροπές καθιστώντας την απόδειξη πνευματικής ιδιοκτησίας τετριμμένα εφικτή από προγράμματα αναγνώρισης. Από την άλλη πλευρά όμως, τα επιτιθέμενα λογισμικά δέχονται διάφορες επιθέσεις από τους εκάστοτε εισβολείς σε αντίθεση με τα μη κακόβουλα. Όμως το ενδιαφέρον είναι, πως οι εισβολείς δεν είναι σε θέση να μπορέσουν εύκολα να ενσωματώσουν το δικό τους υδατογράφημα και να μπερδέψουν τον αναγνωριστή του νόμιμου ιδιοκτήτη.

Τέλος η ιδιότητα των υδατογραφημάτων να μην γίνονται αντιληπτά, προσδιορίζει ποσοτικά τη διαφορά μεταξύ των εντολών που χρησιμοποιήθηκαν για την ενσωμάτωση του υδατογραφήματος και των εντολών που χρησιμοποιήθηκαν για να κατασκευαστεί το κανονικό (αρχικό, μη υδατογραφημένο) λογισμικό. Δυστυχώς, αυτού του τύπου η ιδιότητα είναι άκρως υποκειμενική και δύσκολα μετρήσιμη καθώς δεν υπάρχουν ικανοποιητικές μετρήσεις. Αξίζει στο σημείο αυτό να αναφερθεί πως, ο τρόπος με τον οποίον ο κάθε εισβολέας ψάχνει για πιθανές εντολές υδατογράφησης διαφέρει ανάλογα με το λειτουργικό σύστημα στο οποίο έγινε η μεταγλώττιση του κώδικα και με την γλώσσα που χρησιμοποιήθηκε για την συγγραφή του λογισμικού.

Για παράδειγμα δύο εκτελέσιμα πρόγραμμα μεταγλωττισμένα από το λειτουργικό σύστημα των Linux –os και των Windows –os αντίστοιχα έχουν κάποιες μικρές αλλά υπαρκτές διαφορές λόγο της διαφορετικής αρχιτεκτονικής των δύο λειτουργικών συστημάτων, ή για παράδειγμα ένα πρόγραμμα γραμμένο σε γλώσσα C είναι εντελώς διαφορετικής λογικής με ένα πρόγραμμα γραμμένα σε Java καθώς η μία γλώσσα εντάσσεται στις συναρτησιακές γλώσσες και η άλλη στις αντικειμενοστραφείς.

Επιπλέον, ένα λογισμικό μπορεί να χαρακτηριστεί είτε, στατικά κρυφό είτε, δυναμικά κρυφό είτε, γενικώς κρυφό (οι όροι στα αγγλικά είναι statically stealthy, dynamically stealthy και generally stealthy). Αυτοί οι χαρακτηρισμοί υφίστανται σύμφωνα με τα εξής αποτελέσματα. Όσο αναφορά τον χαρακτηρισμό στατικά κρυφό, ένα πρόγραμμα χαρακτηρίζεται με τον τρόπο αυτόν εάν μία αυτοματοποιημένη επιθεώρηση από ένα εργαλείο στατικής ανάλυσης δεν είναι σε θέση να εντοπίσει ένα υδατογράφημα μέσα στο εκτελέσιμο ενός λογισμικού καθώς επίσης και να ξεχωρίσει ένα υδατογραφημένο πρόγραμμα από ένα που δεν είναι. Στη συνέχεια, για να χαρακτηριστεί ένα λογισμικό δυναμικά κρυφό πρέπει σε σχέση με μία δεδομένη ακολουθία εισόδου, μία αυτοματοποιημένη επιθεώρηση με ένα δυναμικό εργαλείο ανάλυσης, όπως και στα στατικά κρυφά λογισμικά, να μην είναι σε θέση να εντοπίσει ένα υδατογράφημα καθώς και να διακρίνει ένα υδατογραφημένο πρόγραμμα από ένα που δεν είναι. Στο σημείο αυτό είναι αναγκαίο να τονιστεί, πως δυναμικά εργαλεία ανάλυσης, όπως προγράμματα εντοπισμού σφαλμάτων, είναι ιδιαιτέρως επικίνδυνα και αποτελούν σημαντική απειλή για την υδατογράφηση ενός προγράμματος. Τελειώνοντας,

υπάρχει και ο χαρακτηρισμός γενικά κρυφά προγράμματα, τα οποία λαμβάνουν αυτόν τον χαρακτηρισμό επειδή κάποιος εισβολέας με αρκετά καλές γνώσεις υδατογράφησης λογισμικών και με την βοήθεια εργαλείων στατικών και δυναμικών δεν είναι σε θέση να εντοπίσει το υδατογράφημα, ούτε και να διακρίνει ένα υδατογραφημένο λογισμικό από ένα που έχει υποστεί αυτήν την διαδικασία.

1.6 Πλήρης ιστορική αναδρομή

Η ιστορική αναδρομή των υδατογραφημάτων λογισμικού ξεκινά από την στιγμή που δημιουργήθηκαν τέτοιου είδους λογισμικά τα οποία έκαναν κάτι το μοναδικό, κάτι το πρωτότυπο και ήταν διαθέσιμα στην αγορά έναντι πληρωμής. Παραδείγματα τέτοιων λογισμικών αποτελεί το λειτουργικό σύστημα Windows –os υλοποιημένο από την εταιρία Microsoft το οποίο είναι πολύ μεγάλης έκτασης ή διάφορα άλλα λογισμικά μικρότερης έκτασης τα οποία τα είχαν δημιουργήσει για κάποια συγκεκριμένη λειτουργία με απώτερο σκοπό επίσης την αποκομιδή κέρδους, όπως είναι το προγραμματιστικό εργαλείο της Mat lab. Με αποτέλεσμα να είναι αναγκαία κάθε είδους ασφάλειας γύρο από αυτά, όπως είναι η υδατογράφησή τους καθώς και η από κάθε πλευρά κλειστότητά τους (για παράδειγμα το Microsoft office επίσης της εταιρίας Microsoft είναι ένα κλειστό λογισμικό έναντι πληρωμής, στο οποίο δεν μπορούμε ούτε, να δούμε ούτε, να αλλάξουμε τον κώδικά του, σε αντίθεση με το Open office το οποίο είναι ένα ανοιχτό και δωρεάν διαθέσιμο λογισμικό το οποίο παρέχει περίπου τις ίδιες δυνατότητες με το Microsoft office).

Η έρευνα γύρο από την υδατογράφηση λογισμικού καθώς και οι πρώτες εφαρμογές της άρχιζαν να λαμβάνουν μέρος στα τέλη της δεκαετίας του οδόντα, με αρχές τις δεκαετίας του ενενήντα. Στο σημείο αυτό αξίζει να σημειωθεί πως το έτος 1992 βγήκε στην αγορά ένα βιβλίο σχετικό με τα υδατογραφήματα και την προστασία λογισμικού, με τίτλο: The protection of computer software its technology and applications από τον Derrick Grover, το οποίο έδωσε ένα γερό έναυσμα για την δημιουργία ενός καινούριου και μελλοντικά πολύ χρήσιμου τομέα.

Παρακάτω υπάρχει μία εικόνα με το σήμα των πνευματικών δικαιωμάτων, το πλέον γνωστό σε όλους με τον αγγλικό όρο copyright:

Μετά από δύο χρόνια και συγκεκριμένα στις 02-15-1994 δημοσιεύτηκε μία εργασία με τίτλο: Apparatus and method for serializing and validating copies of computer software

από τον Peter R. Samson, η οποία πρωτινέ μία συσκευή και μία μέθοδο για την απενεργοποίηση μη εξουσιοδοτημένων προγραμμάτων για τον ηλεκτρονικό υπολογιστή.

Στην συνέχεια μετά από δύο χρόνια, στις 01-17-1996, δημοσιεύτηκε μία εργασία με τίτλο: Method for stega chiper protection of computer code από τον Moskowitz Scott A. Σε αυτήν την εργασία λάμβανε μέρος μία μέθοδος σχετική με την προστασία των πνευματικών δικαιωμάτων για προγράμματα ηλεκτρονικού υπολογιστή με την κωδικοποίηση του κώδικα μέσα σε μία πηγή δεδομένων με ψηφιακό υδατογράφημα. Το ψηφιακό υδατογράφημα περιείχε πληροφορίες χορήγησης αδειών συνυφασμένων με βασικούς πόρους του κώδικα, καθιστώντας με αυτόν τον τρόπο μόνο τον ιδιοκτήτη του προγράμματος ικανό να εισέλθει σε βασικούς πόρους του λογισμικού, δίνοντας όταν του ζητηθεί, έναν κρυφό κωδικό για πρόσβαση, αποκρύπτοντας με αυτό τον τρόπο παράλληλα τις όποιες πληροφορίες από τρίτους. Με τον τρόπο αυτό, δόθηκε ένα γερό έναυσμα για να δημιουργηθούν μέθοδοι υδατογράφησης και συστήματα για όσο το δυνατόν περισσότερη προστασία των εκάστωτε λογισμικών. Τον ίδιο ακριβώς χρόνο, στις 30-06-1996, μία εργασία με τον τίτλο: Method and system for generating and auditing a signature for a computer program από τους Robert Davidson και Nathan Myhrvold δημοσιεύτηκε με στόχο να προτείνει μία μέθοδο και ένα σύστημα για την παράγωγη και τον έλεγχο ενός υδατοσήματος προς ενσωμάτωση σε εκτελέσιμα προγράμματα υπό συγκεκριμένους όρους. Αξίζει να σημειωθεί πως αυτό το υδατόσημα χαρακτήριζε μοναδικά όλα τα εξουσιοδοτημένα αντίγραφα των εκτελέσιμων προγραμμάτων.

Συνεχίζοντας την αναδρομή στα υδατογραφήματα λογισμικού, το έτος 1998 ο Christian Colleberg, ο Clarck Tomborson και ο Douglas Low δημοσιεύουν μία εργασία τους με τίτλο: Manufacturing cheap resilient and stealthy opaque constructors. Σε αυτήν την εργασία περιγράφονται κάποιοι μετασχηματισμοί πάνω σε προγράμματα γραμμένα σε γλώσσα προγραμματισμού Java, οι οποίοι συσκοτίζουν το διάγραμμα ελέγχου ροής (ο αγγλικός όρος είναι control-flow graph) των προγραμμάτων και μετασχηματίζοντας κάποιες δομές εν ονόματι bytecode (είναι ένα επίπεδο πιο πάνω από την γλώσσα Assembly, το οποίο το δημιουργεί μόνο αυτή η γλώσσα) κάνοντας αλλαγές πάνω σε αυτές, όπως για παράδειγμα κάποιες περιστροφές κόμβων στα δέντρα που δημιουργούνται, χωρίς παράλληλα να επηρεάζεται η εκτέλεση του προγράμματος.

Τον ίδιο ακριβώς χρόνο, δημοσιεύεται και μία άλλη εργασία με τον τίτλο: Analysis of watermarking techniques for graph coloring problem, από τους Gang Qu και Miodrag Potkonjak. Αυτή η εργασία, έδινε ένα θεωρητικό υπόβαθρο στην αξιολόγηση των τεχνικών υδατογράφησης λογισμικού για την προστασία της πνευματικής ιδιοκτησίας, δίνοντας δύο αλγορίθμους βασισμένους στον χρωματισμό γραφημάτων. Στη συνέχεια, επίσης τον ίδιο χρόνο και συγκεκριμένα στις 26-08-1998, οι Christian Colleberg και Clarck Tomborson δημοσίευσαν ακόμη μία εργασία τους με τίτλο: On the limits of software watermarking, στην οποία αναφέρονται στην ενσωμάτωση ενός μυστικού μηνύματος, μέσα σε ένα αντικείμενο. Πιο συγκεκριμένα το μυστικό αυτό μήνυμα είναι συνήθως το γνωστό copyright για την προστασία των πνευματικών δικαιωμάτων και το αντικείμενο που το περιέχει είναι μία ψηφιακή εικόνα. Έχοντας καταφέρει με τον τρόπο αυτό την αποθάρρυνση της κλοπής πνευματικής ιδιοκτησίας ή, όταν αυτό έχει ήδη συμβεί να είναι εφικτή με εύκολο και γρήγορο τρόπο η απόδειξη της γνησιότητας από τους νόμιμους ιδιοκτήτες.

Ένα χρόνο αργότερα, το έτος 1999, οι Julien P.Stern, Gael Hachez, Francois Koeune, Jean Jacques Quisquarter, Ucl Crypto Group, Batiment Maxwell και Place Du Levant

δημοσίευσαν μία εργασία με τίτλο: Robust object watermarking: application to code. Σε αυτήν την εργασία οι άνθρωποι που δούλεψαν για να την φέρουν σε πέρας είχαν επικεντρωθεί σε ένα βήμα υδατογράφησης το οποίο σήμερα έχει πλέον αγνοηθεί. Αυτό που έκαναν λοιπόν ήταν, να εισάγουν ένα παράδειγμα εξόρυξης διανύσματος το οποίο έκανε την μετατροπή μεταξύ ψηφιακών δεδομένων και μίας αφηρημένης αναπαράστασης διανύσματος από τα δεδομένα αυτά. Με πιο απλά λόγια, πρότειναν μία μέθοδο εισαγωγής υδατογραφήματος σε εκτελέσιμο κώδικα.

Τον ίδιο χρόνο προτάθηκαν διάφορες μέθοδοι υδατογράφησης σε Java προγράμματα στηριζόμενες πάνω στην εργασία των Christian Colleberg, Clarck Tomborson και Douglas Low, η οποία είχε δημοσιευτεί ένα χρόνο πριν. Αξίζει να σημειωθεί, πως επίσης τον ίδιο χρόνο με έναυσμα την εργασία τους που και αυτοί είχαν δημοσιεύσει ένα χρόνο πριν οι Gang Qu και Miodrag Potkonjak, δημοσίευσαν δύο καινούριες εργασίες με τίτλους: Optimization intensive watermarking techniques for decision problem και Hiding signatures in graph coloring solutions. Οι οποίες όπως ήταν αναμενόμενο, μιλούσαν για μία βελτιστοποιημένη τεχνική υδατογράφησης για προβλήματα απόφασης και εύρεσης κρυμμένης υπογραφής (υδατογράφημα) μέσα σε χρωματικά γραφήματα.

Τέλος, μετά από ένα πολύ μικρό χρονικό διάστημα του ίδιο έτους μία εργασία των Christian Collberg και Clark Thomborson με τον τίτλο: Software watermarking: models and dynamic embeddings, η οποία σχετίζονταν με δυναμική ενσωμάτωση υδατογραφημάτων έρχεται για να φέρει μία, κατά κάποιον τρόπο, επανάσταση στις υδατογραφήσεις λογισμικού. Πιο συγκεκριμένα, οι υδατογραφήσεις λογισμικού με την τεχνική αυτή προσπαθούν να ενσωματώσουν ένα υδατογράφημα, όπου στην συγκεκριμένη περίπτωση είναι μία δομή, μέσα σε ένα πρόγραμμα με τέτοιον τρόπο, έτσι ώστε η εξαγωγή του να είναι εύκολη και γρήγορη ακόμα και μετά από βελτιστοποιήσεις, συσκοτίσεις και μετασχηματισμούς κώδικα. Παράλληλα όμως, η δομή αυτή δεν πρέπει να γίνεται φανερή στους εκάστοτε εισβολείς.

Με αυτόν τον τρόπο, τη νέα χιλιετία, άρχισαν να προτάσσονται καινούριες και καινοτόμες μέθοδοι υδατογράφησης και από τους προαναφερθέντες, αλλά και από καινούρια άτομα που έμπαιναν χρόνο με το χρόνο σε αυτόν τον κλάδο της πληροφορικής. Έτσι, δίχως φλυαρίες και πιο συγκεκριμένα οι Akito Monden, Hajimu Iida, Ken Ichi Matsumoto, Katsuro Inoue και Koji Torii, στις 28-10-2000, στηριζόμενοι πάνω στις δημοσιεύσεις του Christian Colleberg και των συνεργατών του πρότειναν μία πρακτική μέθοδο υδατογράφησης λογισμικού σε Java προγράμματα, στην εργασία τους με τον τίτλο: A practical method for watermarking Java programs. Εμβαθύνοντας λίγο σε αυτήν την εργασία, γίνεται ευκόλως φανερό πως πρότειναν μία μέθοδο αποθάρρυνσης κλοπής, ενσωματώνοντας ένα ψηφιακό υδατόσημα σε Java πρόγραμμα, το οποίο επιζεί μετά από δύο ειδών επιθέσεις. Η πρώτη επίθεση στηρίζεται σε συσκοτισμένους μετασχηματισμούς (ο αγγλικός όρος είναι obfuscator attacks) και η δεύτερη στην αντίστροφη διαδικασία της μεταγλώττιση προγραμμάτων και στην επαναμεταγλώττιση αυτών (ο αγγλικός όρος είναι decompile recompile attack και είναι κάπως πιο κατανοητός).

Προχωρώντας πάνω στην καινούρια χιλιετία και στο ίδιο έτος οι J. Palsberg, S. Krishnaswamy, Minseok Kwon, D. Ma, Qiuyun Shao και Y. Zhang δημοσιεύουν, στις 15-12-2000, μία εργασία με τίτλο: Experience with software watermarking. Αυτή η εργασία έκανε λόγο για κάποια πειράματα που είχαν κάνει οι προαναφερθέντες για να γίνουν πιο αποδοτικά τα υδατογραφήματα, με συγκεκριμένες αυξήσεις σε μέγεθος του κώδικα, του χρόνου εκτέλεσης και του χρησιμοποιούμενου χώρου από τους σωρούς που δημιουργούνται,

δημιουργώντας ως αποτέλεσμα έναν κώδικα ανθεκτικό σε μία ποικιλία από επιθέσεις με μετασχηματισμούς.

Ένα χρόνο αργότερα το έτος 2001, δημοσιεύτηκε μία εργασία για υδατογραφήσεις λογισμικού από τους Ramanathnam Venkatesan, Vijay Vazirani και Saurabh Sihna με τίτλο: A graph theoretic approach to software watermarking, έχοντας ως κύριο στόχο μία προσέγγιση στις υδατογραφήσεις λογισμικού με τη βοήθεια της θεωρίας γραφημάτων δημιουργώντας εύρωστα υδατογραφήματα για μικρά και μεγάλα σε έκταση προγράμματα λογισμικού. Πιο συγκεκριμένα, δημιουργούσαν ένα κατευθυνόμενο γράφημα κρύβοντας σε αυτό κάποιες πληροφορίες για το υδατόσημα και στη συνέχεια το μετέτρεπαν σε ένα διάγραμμα ελέγχου ροής (ο όρος στα αγγλικά είναι control flow graph), το οποίο και ενσωμάτωναν στο διάγραμμα ελέγχου ροής του αρχικού προγράμματος. Έτσι, μετέτρεπαν αυτό το καινούριο διάγραμμα σε πρόγραμμα δημιουργώντας μία υδατογραφημένη εκδοχή του αρχικού προγράμματος με έξυπνο τρόπο.

Καθώς περνούσε ο χρόνος, όλο και περισσότεροι έστρεφαν το ενδιαφέρον τους στα υδατογραφήματα λογισμικού με στόχο την καλύτερη δυνατή μέθοδο υδατογράφησης και έφερναν στο φως ευρεσιτεχνίες που βοηθούσαν σε ικανοποιητικό βαθμό την εξάρθρωση κλοπών ή την αποθάρρυνση αυτών. Έτσι, έρχεται το έτος 2002 ο Genevieve Arboit να δημοσιεύσει μία εργασία του με τίτλο: A method for watermarking Java programs via opaque predicates, η οποία πρότεινε δύο αλγορίθμους. Ο πρώτος από αυτούς τους δύο, ήταν κατά κάποιον τρόπο απλώς στην εφαρμογή και στην ανάλυσή του, έχοντας ως μειονέκτημα πως κάποιες στρεβλωτικές επιθέσεις μπορούσαν να εξαγάγουν το υδατογράφημα που εισήγαγε, ενώ ο δεύτερος ήταν πιο σύνθετος με το πλεονέκτημα όμως πως οι αποδόσεις του κάτω από ποικίλες επιθέσεις ήταν πολύ καλές συγκριτικά με τον πρώτο.

Βαδίζοντας πάνω στο έτος του 2002 και συγκεκριμένα τον μήνα Φεβρουάριο, οι Christian Collberg, Jasvir Nagra καθώς και ο Clark Thomborson δημοσιεύουν μία εργασία με τίτλο: A functional taxonomy for software watermarking. Σε αυτήν τους την εργασία έδιναν ακριβείς και αναλυτικές ορολογίες για τα υδατογραφήματα λογισμικού με στόχο την καλύτερη κατανόησή τους, δίνοντας παράλληλα σωστές βάσεις για την συνέχιση της έρευνας στον τομέα αυτόν. Στην συνέχεια και συγκεκριμένα τον μήνα Αύγουστο, και πάλι οι Christian Colleberg και Clarck Tomborson στηριζόμενοι στην δημοσιευμένη εργασία τους με τίτλο: On the limits of software watermarking, έρχονται να δημοσιεύσουν μία καινούρια εργασία τους με τίτλο: Watermarking, tamper proofing, and obfuscation tools for software protection, στην οποία πρότειναν τρείς τεχνικές υδατογράφησης με στατικές και δυναμικές τεχνικές χρησιμοποιώντας κατευθυνόμενα γραφήματα.

Με τον τρόπο αυτό, πλησιάζοντας το έτος 2002 στη δύση του, δημοσιέυεται μία εργασία με τίτλο: Tamperproofing a software watermark by encoding constants από τον Yong He, την οποία κατέθεσε για να πάρει το μεταπτυχιακό του δίπλωμα, προσθέτοντας ταυτόχρονα ακόμα ένα λιθαράκι στην έρευνα που λαμβάνει χώρα στις υδατογραφήσεις λογισμικού. Πιο συγκεκριμένα στην εργασία αυτή, oYong He δηλώνει πως τα δυναμικά υδατογραφήματα λογισμικού με χρήση κατευθυνόμενων γραφημάτων είναι μία πολύ καλή τεχνική προστασίας του λογισμικού και προτείνει μία καινούρια ιδέα στηριζόμενη σε αυτά, όπου κωδικοποιούσε κάποιες σταθερές οι οποίες βρισκόντουσαν μέσα στον κώδικα του προγράμματος και τις μετέτρεπε σε μορφή δέντρου. Ενώ παράλληλα ανέπτυξε και αλγορίθμους για την ανάκτηση της κρυμμένης σταθεράς μέσα από το γράφημα με αλγορίθμους, οι οποίοι έκαναν ακριβώς την αντίστροφη διαδικασία.

Φτάνοντας ούτε λίγο ούτε πολύ στο έτος 2003 και στην εργασία με τίτλο: Securing Java through software watermarking από τους D. Curran, N. J. Hurley και M. O. Cinneide, οι οποίοι έκαναν λόγο για το πλεονέκτημα και το μειονέκτημα της μεταφερσιμότητας που έχει η γλώσσα Java λόγο του bytecode που δημιουργεί κατά την διάρκεια μεταγλωττισμού της. Πιο συγκεκριμένα, από τη μία πλευρά το bytecode που δημιουργείται βοηθάει στην ανεξαρτητοποίηση της Java από μηχανήματα ενώ, παράλληλα δίνει πρόσβαση στους εκάστοτε εισβολείς στον πηγαίο κώδικα. Πιο συγκεκριμένα, αυτό που έκαναν ήταν να στηριχτούν πάνω στην θεωρία ανίχνευσης σημάτων τα οποία λάμβαναν μέρος στις υδατογραφήσεις πολυμέσων και να προτείνουν ένα λογισμικό σύστημα υδατογράφησης.

Λίγο χρόνο μετά, στο πέμπτο διεθνές εργαστήριο για απόκρυψη πληροφορίας που έλαβε μέρος στην Ολλανδία, οι Van Len και Yvo Desmedt έρχονται να δημοσιεύσουν μία εργασία με τον τίτλο: Cryptanalysis of UCLA watermarking schemes for intellectual property protection, στην οποία ανέλυαν τέσσερα, για εκείνη την εποχή, πρόσφατα συστήματα υδατογράφησης για την προστασία των πνευματικών δικαιωμάτων των ψηφιακών σχεδίων, στηριζόμενοι στην δημοσιευμένη εργασία με τίτλο: Hiding signatures in graph coloring solutions. Βασιζόμενοι στην ίδια ακριβώς εργασία, ο Christian Colleberg και ο Ginger Myles δημοσιεύουν και αυτοί με την σειρά τους, άλλη μία εργασία με τίτλο: Software watermarking through register allocation: implementation, analysis, and attacks, μέσα στην οποία εξερευνούσαν τον αλγόριθμο υδατοσήμανσης λογισμικού QP, ο οποίος πείρε το όνομά του από τα αρχικά των δημιουργών του (Qu και Potkonjak). Στο σημείο αυτό αξίζει να σημειωθεί πως τα άτομα αυτά κατέληξαν στο συμπέρασμα, αποδεικνύοντας πρώτα ότι αυτός ο αλγόριθμος δεν επιτρέπει την ακριβή αναγνώριση του υδατογραφήματος χωρίς σημαντικές τροποποιήσεις, πως ο αλγόριθμος αυτός χρειάζεται κάποιες τροποποιήσεις προκειμένου να δουλέψει καλύτερα, τις οποίες και έκαναν σε μεταγενέστερα έτη.

Τον ίδιο ακριβώς χρόνο, μία εργασία με τον τίτλο: Error correcting graphs for software watermarking, πάλι από τον Christian Colleberg και Clarck Tomborson καθώς και από δύο καινούρια άτομα πάνω σε αυτόν τον τομέα, τους Stephen Kouborov και Edward Carter, έρχεται να κάνει μία προσέγγιση των υδατογραφημάτων λογισμικού μέσω θεωρίας γραφημάτων και μίας μεθόδου με το όνομα δακτυλικό αποτύπωμα (ο όρος στα αγγλικά είναι fingerprinting), βασισμένη σε μία εργασία που είχε δημοσιευθεί ένα χρόνο πριν από τον Christian Colleberg και κάποιων συνεργατών του. Εμβαθύνοντας λίγο σε αυτήν την εργασία, μπορούμε να πούμε, πως τα άτομα που την έφεραν σε πέρας δίνουν δύο αλγορίθμους οι οποίοι κωδικοποιούν πληροφορίες στο λογισμικό μέσω δομών βασισμένων στα κατευθυνόμενα γραφήματα και στην συνέχεια βγάζουν το υδατογράφημα χωρίς να επηρεάζουν την ορθότητα εκτέλεσης του προγράμματος. Επιπλέον, παρουσίασαν και κάποιες κλάσεις γραφημάτων τα οποία μπορούν να χρησιμοποιηθούν για υδατογράφηση και fingerprinting, δίνοντας την ίδια στιγμή αναλύσεις και ιδιότητες αυτών.

Τελειώνοντας, με την ιστορική αναδρομή του 2003, μία πρωτότυπη εργασία με τίτλο: Mobile agent watermarking and fingerprinting: tracing malicious hosts, από τους Oscar Esparza, Marsel Fernadez, Miguel Soriano, Jose L, Munoz και Jordi Forne έρχεται να δώσει μια άλλη οπτική γωνία στις υδατογραφήσεις λογισμικού. Με άλλα λόγια σε αυτήν την εργασία, γίνεται μία προσέγγιση με στόχο την ανίχνευση επιθέσεων χειραγώγησης που λαμβάνουν μέρος κατά την εκτέλεση ενός πράκτορα (οντότητα λογισμικού η οποία αποτελείτε από κώδικα και δεδομένα τα οποία με την σειρά τους μπορούν να εκτελούνται ανεξαρτήτου μηχανήματος). Έτσι ούτε λίγο ούτε πολύ τελειώνει και αυτό το έτος δίνοντας

την σειρά του σε ένα καινούριο με πολλές και ποικίλες δημοσιεύσεις ανοίγοντας ακόμα τους ορίζοντες για έξυπνες και αποτελεσματικές υδατογραφήσεις λογισμικού.

Φτάνοντας με αυτόν τον τρόπο στο έτος του 2004 και στην δημοσιευμένη εργασία με τίτλο: The evaluation of Davidson's digital signature scheme, από τους Kazuhiro Hattanda και Nonmember, Shuichi Ichikawa, η οποία έλαβε χώρα τον μήνα Ιανουάριο. Αυτή η εργασία είχε στηριχτεί πάνω σε μία άλλη η οποία είχε δημοσιευτεί το 1996 με τίτλο: Method and system for generating and auditing a signature for a computer program, από τους Robert Davidson και Nathan Myhrvold. Αυτή λοιπόν η εργασία, πραγματεύονταν την ενσωμάτωση μίας ψηφιακής υπογραφής (υδατογράφημα), σε ένα λογισμικού. Πιο συγκεκριμένα, η ιδέα της εργασίας αυτής ήταν η ενσωμάτωση ενός υδατογραφήματος σε μία αναδιατεταγμένη ακολουθία από blocks (ο όρος στα ελληνικά είναι τμήμα). Τα μειονεκτήματα αυτής της μεθόδου ήταν η αύξηση του μεγέθους του προγράμματος και η υποβάθμιση των επιδόσεων του, εξαιτίας των επιπροσθέτων αλμάτων των εντολών.

Συνεχίζοντας πάνω στο έτος 2004 και πιο συγκεκριμένα στις 03-03-2004, δημοσιεύτηκε από τον πλέον γνωστό στον τομέα των υδατογραφημάτων, Christian Colleberg και τον Tapas Ranjan Sahoo μία εργασία με τίτλο: Software watermarking in the frequency of domain: implementation, analysis and attacks. Αυτή η εργασία όπως και πολλές άλλες, είχε τις βάσεις της σε μία εργασία που είχε ήδη δημοσιευτεί το έτος 2000 με τον τίτλο: Robust object watermarking: application to code, από τους Julien P.Stern, Gael Hachez, Francois Koeune, Jean Jacques Quisquarter, Ucl Crypto Group, Batiment Maxwell και Place Du Levant. Αυτή η εργασία λοιπόν, ανέλυε τον SHKQ (από τα αρχικά των δημιουργών του Stern, Hachez, Koeune και Quisquarter) αλγόριθμο υδατογράφησης λογισμικού, ο οποίος είχε εφαρμοστεί εντός του Sandmark (ένα σύστημα σχεδιασμένο να μελετά την αποτελεσματικότητα των αλγορίθμων προστασίας λογισμικού, υλοποιημένο από τον Christian Colleberg) με απώτερο σκοπό το bytecode που δημιουργούσε η Java. Επίσης, αναφέρουμε πως αυτός ο αλγόριθμος ενσωμάτωνε ένα υδατογράφημα σε ένα πρόγραμμα χρησιμοποιώντας την spread spectrum τεχνική (έχει γίνει λόγος για αυτή στο παρόν κεφάλαιο και πιο συγκεκριμένα στο υποκεφάλαιο με τίτλο: Κατηγορίες υδατογραφημάτων λογισμικού και χρηστικές ιδιότητες αυτών). Με απλά λόγια, το υδατογράφημα διαδίδεται σε ολόκληρη την εφαρμογή, τροποποιώντας την συχνότητα των εντολών φτιάχνοντας ένα υδατογραφημένο λογισμικό ανθεκτικό σε αρκετά είδη επιθέσεων.

Το ίδιο έτος, μία εργασία με τον τίτλο: Hydan: hiding information in program binaries, από τους Rakan El Khalil and Angelos D. Keromytis δημοσιεύεται προτείνοντας ένα σύστημα για steganographically (δεν υπάρχει ο ακριβής όρος στα ελληνικά) ενσωμάτωση πληροφοριών σε ένα συγκεκριμένο είδος εκτελέσιμων προγραμμάτων. Αυτή η εργασία είχε βασιστεί σε δύο άλλες με τίτλους: Robust object watermarking: application to code, από τους Julien P.Stern, Gael Hachez, Francois Koeune, Jean Jacques Quisquarter, Ucl Crypto Group, Batiment Maxwell και Place Du Levant καθώς και A practical method for watermarking Java programs, από τους Akito Monden, Hajimu Iida, Ken Ichi Matsumoto, Katsuro Inoue και Koji Torii, οι οποίοι και αυτοί με την σειρά τους είχαν επηρεαστεί από δημοσιεύσεις του Christian Colleberg και των συνεργατών του. Πιο συγκεκριμένα η εργασία των Rakan El Khalil and Angelos D. Keromytis όριζε σύνολα από λειτουργικά ισοδύναμα σύνολα εντολών και στην συνέχεια χρησιμοποιούσε ένα κλειδί το οποίο προερχόταν από διαδικασία της επιλογής για την κωδικοποίηση πληροφοριών σε κώδικα μηχανής, χρησιμοποιώντας βέβαια τις εντολές από κάθε σύνολο που δημιουργήθηκε. Τέλος, τα δύο αυτά άτομα τόνιζαν και ανέφεραν κάποια αποτελέσματα από πειράματά τους, τα οποία έκαναν λόγο για ένα μικρό

ποσοστό από πληροφορίες που μπορεί να ενσωματωθεί κάθε φόρα σε ένα εκτελέσιμο πρόγραμμα.

Με έναυσμα αυτή τους την εργασία, η Smita Thacker τον Μάιο του ίδιου έτους, δημοσιεύει μία εργασία της με τίτλο: Software watermarking via Assembly code transformations, για να αποκτήσει το μεταπτυχιακό της δίπλωμα. Στην εργασία αυτή, η Smita έκανε λόγο για την πειρατεία λογισμικού και το πόσο κοστίζει αυτή στις διάφορες εταιρίες λογισμικού, καθώς και για το πώς μπορεί να καταστραφεί ένα υδατογράφημα με κάποιο ειδικό λογισμικό, εισάγοντας με αυτόν τον τρόπο την ιδέα της, η οποία είχε προέλθει από μεταμορφωμένους ιούς ηλεκτρονικών υπολογιστών. Εμβαθύνοντας λίγο στην εργασία της, φαίνεται πως είχε επικεντρωθεί κατά κύριο λόγο σε αυτήν την συγκεκριμένη εφαρμογή των εν λόγο μετασχηματισμών. Πρότεινε δηλαδή κάποιους μετασχηματισμούς στο επίπεδο της γλώσσας Assembly (έχει γίνει αναφορά σε αυτό το επίπεδο στο υποκεφάλαιο με τίτλο: Κατηγορίες υδατογραφημάτων λογισμικού και χρηστικές ιδιότητες αυτών), το οποίο δημιουργούσε μοναδικά αλλά εξίσου λειτουργικά κομμάτια του αρχικού λογισμικού. Συνεχίζοντας, την ίδια χρονολογία, δημοσιεύεται μία εργασία με τον τίτλο: Cover communication through executables, από τους Bertrand Anckaert, Bjorn De Sutter and Koen De Bosschere η οποία και αυτή με την σειρά της είχε στηριχτεί, στην εργασία των Rakan El Khalil and Angelos D. Keromytis και στην εργασία με τίτλο: Method and system for generating and auditing a signature for a computer program, των Robert Davidson και Nathan Myhrvold, η οποία είχε λάβει χώρα το έτος 1996. Κάνοντας ένα είδος εισαγωγής για αυτή την εργασία, είναι αναγκαίο να ειπωθεί πως διερευνούσε τις δυνατότητες των εκτελέσιμων προγραμμάτων να επικοινωνούν με ένα είδος συγκαλυμμένης επικοινωνίας. Πιο συγκεκριμένα, στην δημοσίευση αυτή παρουσιάζονταν τρεις αλγόριθμοι για ενσωμάτωση μυστικής πληροφορίας καθώς επίσης έπερναν μέρος περιγραφές και μετρήσεις πιθανών επιθέσεων στην stealthiness (ο όρος στα ελληνικά είναι μυστικότητα) αυτών των τριών τεχνικών.

Αυτήν την χρονιά επίσης, μία εργασία με τίτλο: A software fingerprinting scheme for Java using classfiles obfuscation, από τους Kazuhide Fukushima και Kouichi Sakurai δημοσιεύεται, γνωστοποιώντας με τον τρόπο αυτό ένα ψηφιακό σύστημα γνωστό με τον αγγλικό του όρο ως fingerprinting για την γλώσσα Java, ενώ παράλληλα είχαν προταθεί και μέθοδοι βελτιστοποίησης για την ήδη δημοσιευμένη μέθοδο του Monden και των συνεργατών του ενσωματώνοντας ένα αναγνωριστικό χρήστη. Όμως σύμφωνα με τους, Kazuhide Fukushima Kouichi Sakurai υπήρχε ένα πολύ μεγάλο και σοβαρό πρόβλημα ενσωματώνοντας αυτό το αναγνωριστικό. Το πρόβλημα ήταν, πως μετά την ενσωμάτωση ο προσδιορισμός του ήταν αρκετά εύκολος. Έτσι για να δώσουν μία λύση στο πρόβλημα που είχε δημιουργηθεί, εφάρμοσαν obfuscations classfiles (ο ελληνικός όρος είναι συσκοτίσεις αρχείων και προσδιορίζει απόλυτα την διαδικασία) κάνοντας με αυτόν τον τρόπο τον εντοπισμό του αναγνωριστικού ιδιαίτερα δύσκολο.

Συνεχίζοντας στο ίδιο σκεπτικό υδατογράφησης και στο ίδιο έτος, έρχεται να δημοσιευτεί μία εργασία με τίτλο: Software watermarking via opaque predicates: implementation, analysis and attacks, από τους Christian Collberg και Ginger Myles. Η εργασία αυτή στηρίζονταν σε μία εργασία του ίδιου του Christian Collberg και του Clarck Tomborson η οποία είχε δημοσιευτεί το έτος 1998 με τίτλο: Manufacturing cheap resilient and stealthy opaque constructors. Πιο συγκεκριμένα, η εργασία αυτή έθιγε το όλο και περισσότερο αναπτυσσόμενο πρόβλημα της πειρατείας λογισμικού, δίνοντας ως λύση μία τεχνική η οποία ονομαζόταν πρόληψη λογισμικού υδατογράφησης (η οποία αρχικά

προτάθηκε από τον Genevieve Arboit στην εργασία του με τίτλο: A method for watermarking Java programs via opaque predicates). Με άλλα λόγια, η τεχνική αυτή ενσωματώνει ένα υδατογράφημα με την προσθήκη αδιαφανών κατηγορημάτων σε κάποια εφαρμογή.

Παράλληλα όμως, εκείνη την εποχή αναπτύσσονταν και τεχνικές δυναμικής υδατογράφησης, οι οποίες βασίζονταν στα κατευθυνόμενα γραφήματα. Μία από αυτές ήταν και η εργασία με τίτλο: Graph theoretic software watermarks: implementation, analysis and attacks από τους Christian Collberg, Andrew Huntwork, Edward Carter and Gregg Townsend. Αυτή η εργασία αναφερόταν σε ένα εκτελέσιμο πρόγραμμα το οποίο χαρακτηρίζονταν από την προσθήκη του κώδικα, με βάση την οποία η τοπολογία του διαγράμματος ελέγχου ροής κωδικοποιούσε ένα υδατογράφημα. Επιπλέον, στην εργασία αυτή είχαν γίνει μετρήσεις της έκτασης του κώδικα καθώς επίσης και του συνολικού χρόνου εκτέλεσης αυτού, δίνοντας διάφορες αξιολογήσεις σε επιθέσεις που λάμβαναν χώρα στο υδατογράφημα. Στο ίδιο μήκος κύματος κυμαινόταν και μία εργασία των Clark Thomborson, Jasvir Nagra, Ram Somaraju και Charles He του ίδιου έτος με τίτλο: Tamper proofing software watermarks, στην οποία προτείνονταν και πάλι κάποιοι μετασχηματισμοί, αυτή τη φορά σε αριθμητικές ή μη αριθμητικές σταθερές οι οποίες ήταν τοποθετημένες στο εσωτερικό του υδατογραφημένου κώδικα και πιο συγκεκριμένα στις κλήσεις των συναρτήσεων των οποίων οι τιμές ήταν εξαρτημένες από τα δεδομένα τις υδατογραφημένης δομής. Επιτυγχάνοντας με αυτόν τον τρόπο μία καλή ασφάλεια του λογισμικού, καθώς ο εκάστοτε κακόβουλος χρήστης, υπό λογικές πάντα συνθήκες, δεν θα μπορεί να είναι βέβαιος πως έχουν γίνει μεταβολές στις σταθερές του προγράμματος.

Συνεχίζοντας την αναδρομή στα υδατογραφήματα λογισμικού, το έτος 2004 μία καινοτόμα εργασία με τίτλο: Threading software watermarking, από τους Jasvir Nagra και Clark Thomborson έρχεται να φωτίσει μία άλλη οπτική γωνία των υδατογραφημάτων λογισμικού. Σε αυτήν την εργασία γίνονταν λόγος για μία καινούρια τεχνική σύμφωνα με την οποία ένα εύρωστο υδατογράφημα ενσωματώνονταν μέσα σε ένα πρόγραμμα χρησιμοποιώντας νηματική σύνδεση. Έτσι το υδατογράφημα γινόταν ελαστικό σε διάφορους μετασχηματισμούς στους οποίους οι μέχρι τότε τεχνικές ήταν ιδιαίτερα ευαίσθητες. Πιο συγκεκριμένα, οι Jasvir Nagra και Clark Thomborson κωδικοποιούσαν ένα υδατογράφημα σε ένα αλφαριθμητικό από bits και ένα σχεδιάγραμμα, τα οποία στην συνέχεια τα εντόπιζαν με την τεχνική της νηματικής επικοινωνίας. Έχοντας ως αποτέλεσμα καταφέρει με την τεχνική αυτή να μην γίνεται πολύ μεγαλύτερος σε έκταση ο κώδικας του εκάστωτε λογισμικού και η ταχύτητα του να παραμένει στα ίδια επίπεδα με τον αρχικό.

Ταυτόχρονα με αυτή την εργασία, άλλη μία με τίτλο: An abstract interpretation based framework for software watermarking, από τους Patrick και Radhia Cousot έρχεται να προσφέρει μία καινούρια οπτική γωνία ενσωμάτωσης. Με άλλα λόγια, πρότειναν την εισαγωγή υδατογραφήματος απευθείας μέσα στον κώδικα του προγράμματος με τέτοιον τρόπο όμως που να μπορεί να γίνει η εξαγωγή του μόνο από αφηρημένη κλάση του συγκεκριμένου κώδικα. Επιτυγχάνοντας έτσι και την εκτέλεση του κώδικα αβίαστα δίχως να εξαρτάται η ροή του από αυτήν την κλάση και την μη εφικτή εξαγωγή του υδατογραφήματος σε περίπτωση απουσίας της αφηρημένης κλάσης.

Στο σημείο αυτό είναι σκόπιμο να τονιστεί, πως υπάρχουν κάποιες ειδικού τύπου εφαρμογές οι οποίες είναι κατασκευασμένες με τέτοιον τρόπο, που αφαιρούν από τα εκτελέσιμα οτιδήποτε δεν χρειάζεται για την εκτέλεση του προγράμματος καθώς και ότι

χρησιμοποιείται αλλά δεν συνεισφέρει καθόλου στο τελικό αποτέλεσμα. Έτσι ότι και να έχει τοποθετήσει κάποιος στο πρόγραμμα στη μορφή της ακριβώς παραπάνω εργασίας είναι αρκετά εύκολα να εντοπιστεί και να καταστραφεί. Για αυτόν ακριβώς τον λόγο, πάνω σε αυτήν την ιδέα έγιναν, τέσσερα χρόνια μετά, μόνο δύο εργασίες με τίτλους: Hiding information in completeness holes και Hiding software watermarks in loop structures οι οποίες προχώρησαν την βασική ιδέα κατά ένα μικρό ποσοστό, δίνοντας ένα κατά κάποιον τρόπο τέλος σε αυτήν την λογική.

Παρόλα αυτά όμως, καινούριες ιδέες έρχονταν στην επιφάνεια και γίνονταν εργασίες. Μία από αυτές ήταν η εργασία με τίτλο: Dynamic path based software watermarking, δημοσιευμένη από τους C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn και M. Stepp. Αυτά τα άτομα εισήγαγαν την ιδέα της δυναμικής διακλάδωσης στην συμπεριφορά των προγραμμάτων. Δηλαδή, έδειξαν πως οι τεχνικές error correcting και tamper proofing που είχαν ανακαλυφθεί, η πρώτη δύο χρόνια πριν και η δεύτερη ένα χρόνο πριν μπορούσαν να χρησιμοποιηθούν για να δημιουργήσουν υδατογραφήματα βασισμένα σε μονοπάτια, τα οποία ήταν ανθεκτικά σε ένα μεγάλο πλήθος επιθέσεων, δίνοντας παράλληλα και κάποια πειραματικά αποτελέσματα βασισμένα στο κόστος ενσωμάτωσης τέτοιου είδους υδατογραφημάτων με την βοήθεια του bytecode της γλώσσας προγραμματισμού Java σε μεγάλης έκτασης κώδικες.

Προχωρώντας λίγο ακόμα φτάνοντας στο τέλος αυτού του έτους, στις 24-10-2004 και στην εργασία με τίτλο: SWuS: software watermarking using slices, από τους Mullick Amarnath και Srikant. Σύμφωνα με αυτούς, οι μέχρι τότε τεχνικές υδατογράφησης έπασχαν από σοβαρά μειονεκτήματα, όπως το ότι τοποθετούσαν υδατογραφήματα τα οποία δεν ήταν ιδιαίτερα εύρωστα, ή ότι ήταν αρκετά ορατά δίνοντας έτσι την δυνατότητα στους εκάστοτε εισβολείς να τα εξάγουν χωρίς ιδιαίτερη δυσκολία. Για αυτόν τον λόγο, η εργασία τους ξέφευγε από τις συνηθισμένες τεχνικές, υιοθετώντας την έννοια του program slice (ο ελληνικός όρος είναι φέτα προγράμματος), η οποία με βάση τους συγγραφείς της εργασίας ήταν ιδιαίτερα καλή στο θέμα της ορατότητας και ιδιαίτερα εύρωστη όσο αναφορά τους κακόβουλους μετασχηματισμούς.

Φτάνοντας, ούτε λίγο ούτε πολύ στο έτος 2005 και στην εργασία με τίτλο: Dither modulation watermarking of dynamic memory traces, από τους Alan J. Larkin, Felix Balado, Neil J. Hurley και Guenole C. M. Silvestre. Τα άτομα αυτά βασισμένα πάνω σε μία δημοσίευση που είχε λάβει χώρα ένα χρόνο πριν με τον τίτλο: Software watermarking in the frequency of domain: implementation, analysis and attacks, από τους Christian Colleberg και τον Tapas Ranjan Sahoo, έγραψαν και δημοσίευσαν μία νέα εργασία μέσα στην οποία περιέγραφαν μία δυναμική μέθοδο υδατογράφησης λογισμικού. Πιο συγκεκριμένα, η ενσωμάτωση του υδατογραφήματος γίνονταν μέσα στο ίχνος μνήμης που δημιουργούσε το εκτελέσιμο όταν έτρεχε. Από την άλλη πλευρά οι Bertrand Anckaert, Bjorn De Sutter και Koen De Bosschere έρχονται να δημοσιεύσουν και αυτοί με την σειρά τους μία εργασία με τίτλο: Steganography for executables, η οποία και αυτή με την σειρά της είχε τις βάσεις της σε μία δημοσίευση που είχε λάβει χώρα ένα χρόνο πριν με τίτλο: Covert communication through executables, από τους ίδιους. Στη νέα εργασία γίνονταν λόγος για ενσωμάτωση ενός κρυφού μηνύματος σε ένα φαινομενικά ακίνδυνο αντικείμενο. Εμβαθύνοντας κάπως σε αυτήν την εργασία, αντιλαμβάνεται κάποιος πως η μέθοδος steganography μελετά τεχνικές επικοινωνίας με τέτοιον τρόπο, όπου η ανίχνευση ενός μηνύματος πρέπει να είναι εφικτή. Ενώ ταυτόχρονα, το κρυμμένο μήνυμα είναι κωδικοποιημένο με τεχνικές κρυπτογράφησης

πριν ενσωματωθεί. Έχοντας με αυτόν τον τρόπο δημιουργήσει μία δύσκολη μορφή για ανίχνευση από τρίτους.

Βαδίζοντας στο ίδιο έτος, έρχονται στην επιφάνεια αξιολογήσεις δύο διαφορετικών αλγορίθμων, στην εργασία με τον τίτλο: The evaluation of two software watermarking algorithms, από τους Christian Collberg, Ginger Myles, Zachary Heidepriem και Armand Navabi. Αυτοί οι αλγόριθμοι είχαν δημοσιευτεί, ο πρώτος από τον Akito Monden και τους συνεργάτες του και ο δεύτερος από Robert L. Davidson and Nathan Myhrvold από την εταιρία της Microsoft. Με άλλα λόγια αυτό που έκαναν αυτά τα άτομα ήταν να εφαρμόσουν αυτούς τους δύο αλγορίθμους σε ένα σύστημα ονόματι Sandmark του Christian Collberg, το οποίο όπως έχει αναφερθεί και πιο πάνω, είναι ένα σύστημα, σχεδιασμένο να μελετά την αποτελεσματικότητα των αλγορίθμων προστασίας λογισμικού. Με τον τρόπο αυτό, απέδειξαν πως αυτοί οι αλγόριθμοι έπασχαν από υψηλό bit rate (ο ελληνικός όρος είναι ρυθμός από bits) και από χαμηλά επίπεδα από ορατότητας υδατογραφήματος με αποτέλεσμα να είναι ευάλωτα σε πολλές επιθέσεις κακόβουλων χρηστών.

Την ίδια χρονιά αλλά σε διαφορετικό μήκος κύματος μία εργασία δημοσιεύεται από τους William Zhu και Clark Thomborson με τίτλο: On the QP Algorithm in Software Watermarking, η οποία είχε θεμελιώδεις βάσεις σε μία εργασία των Qu και Potkonjak. Σε αυτήν την εργασία προτείνονταν τρεις διορθώσεις πάνω στον αλγόριθμο QP, δίνοντας παράλληλα και παραδείγματα για την κάθε διόρθωση. Στην συνέχεια, μία εργασία με τίτλο: A survey of software watermarking έρχεται να δημοσιευτεί από τους William Zhu, Clark Thomborson και Fei Yue Wang. Σε αυτή την εργασία δεν αναφερόταν κάτι το καινούριο, όμως ο στόχος της ήταν να συγκεντρώσει, να ταξινομήσει και να περιγράψει όσο το δυνατόν περισσότερα πράγματα στον τομέα των υδατογραφημάτων λογισμικού με όσο γίνεται πιο κατανοητό τρόπο.

Αλλάζοντας λίγο το ήδη υπάρχον κλίμα, τον ίδιο χρόνο δημοσιεύεται μία εργασία, με επιρροές από μία ήδη δημοσιευμένη ένα χρόνο πριν από τους C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn και M. Stepp με τίτλο: Dynamic path based software watermarking. Στην καινούρια εργασία λαμβάνουν μέρος οι Matias Madou, Bertrand Anckaert, Bjorn De Sutter και Koen De Bosschere και η εργασία είχε τον τίτλο: Hybrid static dynamic attacks against software protection mechanisms. Σε αυτήν την δημοσίευση υπήρχε καινοτόμο υλικό το οποίο μιλούσε για ένα υβριδικό είδος επιθέσεων πάνω σε λογισμικά και πιο συγκεκριμένα είχε επικεντρωθεί σε λεπτομερείς περιγραφές για στατικές και για δυναμικές επιθέσεις που θα μπορούσαν να δεχτούν τα λογισμικά προκειμένου να εντοπιστούν τα όποια υδατογραφήματα μέσα σε αυτά.

Προς το τέλος του έτους, δύο επίσης καινοτόμες εργασίες πάνω στον τομέα των υδατογραφημάτων λογισμικού εμφανίζονται, με τους τίτλους: Software ip protection based on watermarking techniques και Self validating branch based software watermarking και δημοσιεύονται. Οι συγγραφείς τις πρώτης εργασίας ήταν οι Vyacheslav Yarmolik και Siarhei Partsianka και είχαν επικεντρωθεί σε τεχνικές υδατογράφησης του εκτελέσιμου κώδικα καθώς και σε ποικίλες αναλύσεις στο θέμα της ενσωμάτωσης υδατογραφημάτων, θίγοντας ταυτόχρονα το θέμα των διάφορων εφαρμογών που είναι διαθέσιμων στο διαδίκτυο, οι οποίες γίνονται πολύ συχνά στόχοι των πειρατών του διαδικτύου προς καθαρά δική τους ωφέλεια. Από την άλλη όμως πλευρά, βρισκόντουσαν Ginger Myles και Hongxia Jin (συγγραφείς της δεύτερης εργασίας), οι οποίοι είχαν δημιουργήσει μία τεχνική δυναμικής υδατογράφησης με στοιχεία από κακόβουλους μετασχηματισμούς (ο αγγλικός όρος είναι obfuscations)

εφαρμόζοντας την ίδια στιγμή και διάφορες επιθέσεις πάνω σε αυτήν για την καλύτερη εξαγωγή συμπερασμάτων.

Τέλος, μία τελευταία εργασία για το έτος του 2005 με τον τίτλο: Copyright protection of J2EE web applications through watermarking από τους Feiyuan Wang, Jiying Zhao, Abdulmotaleb El Saddik, δημοσιεύεται έχοντας ως κίνητρο τις εκάστοτε εφαρμογές που βρίσκονται στο διαδίκτυο υλοποιημένες σε γλωσσά Java και γίνονται εφικτές προς χρήση μέσω των πρωτοκόλλων TCP/IP και HTTP. Τα άτομα αυτά, που εργάστηκαν για την εν λόγο δημοσίευση, γνωρίζοντας το πλήθος των επιθέσεων οι οποίες έχουν λάβει μέρος στα προηγούμενα πρωτοκόλλα, δημιούργησαν μία τεχνική υδατογράφησης και ανίχνευσης μέσω αυτών των πρωτόκολλων, πάνω σε αυτές τις εφαρμογές.

Παρακάτω παρατίθενται μία εικόνα, στην οποία σατιρίζονται οι τρόποι με τους οποίους υδατογραφούνται τα εκάστοτε λογισμικά:

Με αυτόν τον τρόπο, κλείνει το έτος 2005, δίνοντας την σκυτάλη στον αμέσως επόμενο χρόνο, ο οποίος κληρονομεί πολλές γνώσεις γύρο από τον τομέα υδατογράφησης

λογισμικού ερχόμενες από χρόνων έρευνας. Έτσι δίχως πολλές φιλοσοφίες, μία εργασία με τον τίτλο: Opaque predicates detection by abstract interpretation από τους Mila Dalla Preda, Matias Madou, Koen De Bosschere και Roberto Giacobazzi έρχεται για να ανοίξει την αυλαία του 2006, στηριζόμενη βέβαια όπως και η πλειονότητα των εργασιών σε κάποια ήδη δημοσιευμένη με τον τίτλο: Manufacturing cheap resilient and stealthy opaque constructors από τους Christian Colleberg, Clarck Tomborson και Douglas Low. Στην νέα εργασία προτείνεται μία μέθοδος υδατογράφησης βασισμένη σε μεθόδους αφαίρεσης αδιαφανών κατηγορημάτων από τα προγράμματα, με την βοήθεια μίας αφηρημένης ερμηνείας. Η ερμηνεία αυτή παρέχει το σωστό πλαίσιο για την απόδειξη της ορθότητας αυτών των προσεγγίσεων (της συγκεκριμένης εργασίας), σε συνδυασμό με μία αποτελεσματική μέθοδο σχεδιασμού αποτελεσματικών επιθέσεων στα αδιαφανή κατηγορήματα των εκάστοτε προγραμμάτων.

Συνεχίζοντας την ως τώρα πλήρη ιστορική αναδρομή, φτάνει και η δημοσίευση της εργασίας από τους William Zhu και Clark Thomborson με τίτλο: Algorithms to watermark software through register allocation. Η εργασία αυτή, βασισμένη σε μία προγενέστερή της, από τους ιδίους με τίτλο: On the QP Algorithm in Software Watermarking, αναφέρει κάποιες δυσκολίες που συναντώνται στον γνωστό αλγόριθμο QP, επισημαίνοντας το πρόβλημα της εξαγωγής του υδατογραφήματος, το οποίο έχει τοποθετηθεί με την βοήθεια του προαναφερθέν αλγορίθμου. Δίνοντας, ταυτόχρονα και κάποιες προτεινόμενες βελτιώσεις πάνω σε αυτόν.

Τον ίδιο καιρό περίπου, πάλι οι William Zhu και Clark Thomborson, έχοντας ως έναυσμα τον ίδιο αλγόριθμο QP, δημοσιεύουν άλλες δύο παρεμφερείς εργασίες τους με τίτλους: Extraction in software watermarking και Recognition in software watermarking. Στις εργασίες αυτές έκαναν λόγο για την επικίνδυνα αυξανόμενη πειρατεία του διαδικτύου, σε θέματα μη εξουσιοδοτημένων τροποποιήσεων σε λογισμικά, προτείνοντας ένα πρωτότυπο μοντέλο ενσωμάτωσης υδατογραφήματος καθώς και το σύστημα αναγνώρισης και εξαγωγής αυτού.

Στη συνέχεια, μία άλλη εργασία με τον τίτλο: Technique for producing through watermarking highly tamper-resistant executable code and resulting “watermarked” code so formed από τους Venkatesan Ramarathnam και Vazirani Vijay, στηριζόμενη πάνω σε δύο δημοσιεύσεις των ίδιων και των Christian Collberg και Clark Thomborson, οι οποίες είχαν λάβει χώρα προ πέντε και εφτά ετών αντίστοιχα με τίτλους: A graph theoretic approach to software watermarking και Software watermarking: models and dynamic embedding. Αυτή η εργασία είχε στόχο να προτείνει συσκευές και μεθόδους για διαμόρφωση και ενσωμάτωση υδατογραφημάτων σε λογισμικά. Πιο συγκεκριμένα, αυτό που πρότειναν ήταν κάποιες ρουτίνες (συναρτήσεις) προς ενσωμάτωση στο διάγραμμα ελέγχου ροής δημιουργώντας με αυτόν τον τρόπο ένα υδατογραφημένο πρόγραμμα, το οποίο έμοιαζε σε ένα πολύ μεγάλο ποσοστό στο αρχικό μη υδατογραφημένο έχοντας ως αποτέλεσμα την μη εύκολα εφικτή αναγνώριση του υδατογραφήματος από τρίτους.

Στο ίδιο μήκος κύματος, κυμαινόταν και επόμενη εργασία η οποία έφερε τον τίτλο: A constant encoding algorithm which is tamper proofs the CT watermark, δημοσιευμένη από τον Lei Wang, καθώς και αυτή είχε δεχτεί επιρροές από την δημοσίευση με τίτλο: Software watermarking: models and dynamic embeddings. Εμβαθύνοντας λίγο στην εργασία αυτή, γίνεται φανερό πως αναφερόταν σε έναν αλγόριθμο υδατογράφησης των Christian Collberg

και Clark Thomborson, αναλύοντάς τον, δίνοντας παράλληλα διάφορες εφαρμογές του καθώς και έναν δικό του αλγόριθμο τον οποίο τον είχε σχεδιάσει για μία τεχνική υδατογράφησης.

Λίγους μήνες μετά, έρχεται η δημοσίευση μίας εργασίας με τίτλο: A low cost attack on branch based on software watermarking schemes, από τους Gaurav Gupta και Josef Pieprzyk. Κάνοντας ένα είδους προλόγου για αυτή την εργασία, αξίζει να σημειωθεί πως ένα χρόνο πριν, οι Ginger Myles και Hongxia Jin στην τότε εργασία τους με τίτλο: Self validating branch based software watermarking, είχαν προτείνει μία τεχνική υδατογράφησης βασισμένη στην μετατροπή εντολών αλμάτων και στις άνευ όρων διακλαδώσεις κάνοντας κλήσεις σε μία ειδική συνάρτηση ανίχνευσης που υπολόγιζε την σωστή διεύθυνση στόχο των άνευ όρων διακλαδώσεων ως συνάρτηση των παραγόμενων αποτελεσμάτων της. Έτσι σε περίπτωση αλλοίωσης του προγράμματος, το υδατόσημα, οι έλεγχοι ακεραιότητας και οι διευθύνσεις των στόχων δεν θα μπορούσαν να υπολογιστούν σωστά. Βασισμένοι λοιπόν σε όλα αυτά, οι Gaurav Gupta και Josef Pieprzyk εργάστηκαν και δημιούργησαν μία επίθεση εντοπισμού και τροποποίησης του δείκτη στοίβας με σκοπό την διάλυση σχεδιαγράμματος του προγράμματος παρέχοντας ταυτόχρονα και κάποιες λεπτομέρειες εκτέλεσης. Το σημείο κλειδί της όλης διαδικασίας ήταν η μετακίνηση του υδατογραφήματος και ο έλεγχος ακεραιότητας του παραγόμενου κώδικα μετά την απομάκρυνση της διεύθυνσης στόχου. Τέλος παρουσίαζαν και δύο επιθέσεις, μία βασισμένη στην αφαίρεση και μία στην αντικατάσταση του υδατογραφήματος.

Προχωρώντας, έρχεται ο μήνας Οκτώβριος και πιο συγκεκριμένα η 16-10-2006 με την εργασία με τίτλο: Copyright Protection of Web Applications through Watermarking από τους Ronghui Tu, Feiyuan Wang, Jiying Zhao και El Saddik. Τα άτομα αυτά, είχαν στηριχτεί σε μία εργασία τους που την είχαν δημοσιεύσει έναν χρόνο πριν με τον τίτλο: Copyright protection of J2EE web applications through watermarking. Κοιτάζοντας αυτήν την εργασία με μία πιο αναλυτική άποψη, είναι ξεκάθαρο πως μιλούσε για προστασία εφαρμογών υλοποιημένων σε γλώσσα Java, οι οποίες ήταν διαθέσιμες εντός τους διαδικτύου. Δηλαδή, πρότειναν ένα είδος προστασίας τέτοιου είδους εφαρμογών με έναν αλγόριθμο, ο οποίος χρησιμοποιούσε αμφότερα και συσκοτισμένους μετασχηματισμούς και την τεχνική αλλοιωμένης απόδειξης (οι αγγλικοί όροι είναι code obfuscations και tamper proofing), δίνοντας και κάποια παραδείγματα στο τέλος για την καλύτερη κατανόησή του θέματος.

Με τον τρόπο μία ακόμη εργασία από τους Fenlin Liu, Bin Lu και Xiangyang Luo με τίτλο: A Chaos-Based Robust Software Watermarking, έρχεται να δηλώσει το τέλος και αυτού του έτους. Πιο συγκεκριμένα, στην εργασία αυτήν προτείνοντας ένας αλγόριθμος υδατογράφησης λογισμικού βασισμένος σε αρκετούς χαοτικούς περιορισμούς του αρχικού (μη υδατογραφημένου) λογισμικού. Με απλά λόγια, ο αλγόριθμος αυτός συνδύαζε την τεχνική της αντίστροφης μεταγλώττισης (ο όρος στα αγγλικά είναι anti-reverse engineering), του χαοτικού συστήματος και την υποκατηγορία υδατογραφημάτων λογισμικού εν ονόματι Easter eggs. Οι εφαρμογές αυτές λάμβαναν χώρα σε επεξεργαστή Intel i386 καθώς και σε λειτουργικό σύστημα Windows -os, δίνοντας και κάποιες αναλύσεις, οι οποίες ανέφεραν πως αυτή η τεχνική αντιστέκεται σε ποικίλα είδη επιθέσεων.

Πλησιάζοντας όλο και περισσότερο στην ολοκλήρωση της ιστορικής αναδρομής, έρχεται το έτος 2007 και η εργασία με τίτλο: A stern based collusion secure software watermarking algorithm and its implementation από τους Jieqing Ai, Xingming Sun, Yunhao Liu, Ingemar J. Cox, Guang Sun και Yi Luo. Στην εργασία αυτή παρουσιάζονταν ένας αλγόριθμος βασισμένος σε εύρωστες και στατικές μεθόδους υδατοσύμανσης λογισμικού, οι

οποίες ήταν ανθεκτικές σε ποικίλες επιθέσεις, δίνοντας παράλληλα και κάποια παραδείγματα από αυτές. Στην συνέχεια αυτού τους έτους και πιο συγκεκριμένα τον μήνα Αύγουστο, δημοσιεύεται μία εργασία με τίτλο: Concepts and techniques in software watermarking and obfuscation από τον William Feng Zhu η οποία ήταν βασισμένη σε τέσσερις άλλες ήδη δημοσιευμένες προ διετίας με τίτλους: Algorithms to watermark software through register allocation, Extraction in software watermarking, Recognition in software watermarking καθώς και A survey of software watermarking. Αν κάποιος εστιάσει σε αυτήν την εργασία, βλέπει πως αναφέρεται σε μία έρευνα πάνω στα λογισμικά υδατογράφησης και σε διάφορους μετασχηματισμούς αυτών. Επιπλέον, επισημοποιεί δύο σημαντικές ιδέες πάνω σε αυτόν τον τομέα, οι οποίες δεν είναι άλλες από την εξαγωγή και αναγνώριση των υδατογραφημάτων. Για τον σκοπό αυτό, χρησιμοποιείται μία μορφή ενός συνεπτυγμένου αλγορίθμου προς απεικόνιση αυτών των θεμάτων καθώς επίσης γίνεται και μία παρουσίαση μίας τεχνικής εν ονόματι ομομορφικές συναρτήσεις (ο αγγλικός όρος είναι homomorphic functions), μέσω των οποίων κάποιοι αριθμοί υπόλειμμα, συσκοτίζονται σε μεταβλητές και δομές δεδομένων του αρχικού (μη υδατογραφημένου) προγράμματος.

Ακολουθώντας, την ροή σύμφωνα με την οποία λαμβάνουν χώρα οι δημοσιεύσεις πάνω στον τομέα των υδρογραφημάτων λογισμικού, έρχεται η σειρά της δημοσίευσης με τον τίτλο: Informed recognition in software watermarking, πάλι από τον William Zhu. Αυτή η εργασία, όπως και η προηγούμενη του είχε τις βάσεις της σε μία εξίσου δική του, με τον τίτλο: Recognition in software watermarking. Ο στόχος αυτής της εργασίας ήταν αρχικά να προτείνει ιδέες ενσωμάτωσης υδατογραφήματος, εξαγωγής και αναγνώρισης αυτού καθώς και να δηλώσει την ιδέα της ενημερωμένης αναγνώρισης (ο αγγλικός όρος είναι informed recognition).

Από την άλλη πλευρά, στις 28-05-2007 μία άλλη εργασία με τον τίτλο: Dynamic graph based software fingerprinting, έρχεται για να δημοσιευτεί από τους Christian Collberg, Clark Thomborson, και Gregg M. Townsend. Τα άτομα αυτά σε αυτή τους την εργασία, πρότειναν μία μέθοδο υδατογράφησης, η οποία είχε δεχθεί πολλές επιρροές από μία δημοσίευση τους προ οχταετίας με τίτλο: Software watermarking: models and dynamic embeddings. Για να γίνει το τοπίο κάπως πιο καθαρό αξίζει να σημειωθεί πως η ιδέα αυτών των ανθρώπων ήταν η ενσωμάτωση μίας δομής, ή αλλιώς ενός υδατογραφήματος στο εσωτερικό ενός λογισμικού, με τέτοιον τρόπο έτσι ώστε καθώς το πρόγραμμα εκτελείται να αλλάζει δυναμικά, να είναι όσο το δυνατόν πιο κρυφή γίνεται καθώς επίσης να μην παραμερίζεται και το θέμα εξαγωγής αυτής με εύκολο και γρήγορο τρόπο. Έτσι έχοντας αυτά κατά νου, σε αυτή τους την εργασία, δημιούργησαν ένα δυναμικό υδατογράφημα λογισμικού, το οποίο δεν είναι κάτι άλλο πέρα από μία δομή δεδομένων και το ενσωμάτωσαν στον εκτελέσιμο κώδικα ενός λογισμικού.

Αλλάζοντας τώρα σκηνικό, με μία εργασία με τον τίτλο: Software watermarking resilient to debugging attacks από τους Gaurav Gupta και Josef Pieprzyk, επηρεασμένη από μία πάλι δική τους με τίτλο: A low cost attack on branch based on software watermarking schemes, έρχεται να συνεχίσει κάτι που είχαν ξεκινήσει οι ίδιοι έναν χρόνο πριν. Πιο συγκεκριμένα σε αυτή τους την εργασία, τα άτομα αυτά παρουσίαζαν μία επίθεση βασισμένη στα υδατογραφήματα τα οποία χειρίζονται τις διακλαδώσεις του σχήματος λογισμικού, η οποία με την δική της σειρά είχε προταθεί το 2005 από τους Ginger Myles και Hongxia Jin στην τότε εργασία τους με τον τίτλο: Self validating branch based software watermarking. Επομένως, βάσει αυτής της επίθεσης οι Gaurav Gupta και Josef Pieprzyk δημιούργησαν ένα μοντέλο υδατογράφησης, το οποίο παρείχε ασφάλεια έναντι αυτών των επιθέσεων.

Από την άλλη πλευρά, μία πρωτότυπη εργασία με τίτλο: Information hiding in software with mixed boolean arithmetic transforms από τους Yongxin Zhou, Alec Main, Yuan X. Gu και Harold Johnson έρχεται να αλλάξει κάπως το τοπίο στα υδατογραφήματα λογισμικού προτείνοντας κάτι το ιδιαίτερο και να κλείσει με την σειρά της το έτος του 2007. Με άλλα λόγια, σε αυτήν την εργασία προτείνονται πολλοί συσκοτισμένοι μετασχηματισμοί μέσω μεικτού τύπου μετασχηματισμών πάνω στην δυαδική άλγεβρα (ο αγγλικός όρος είναι boolean algebra) οι οποίοι είχαν αντίκτυπο σε συναρτήσεις και σε δεδομένα του αρχικού λογισμικού. Επιπλέον γίνεται συζήτηση για ευρείες χρήσεις και συγκεκριμένες εφαρμογές ενός συγκεκριμένου κλειδιού καθώς επίσης και για γενικές υδατογραφήσεις λογισμικού.

Συνεχίζοντας, έρχεται η μετάβαση για το έτος 2008 και για την πρώτη εργασία εκείνου του έτος στο θέμα των υδατογραφήσεων λογισμικού, βασισμένη σε μία παλαιότερη με τίτλο: Hydan: hiding information in program binaries από τους Rakan El Khalil and Angelos D. Keromytis. Η καινούρια εργασία, έφερε τον τίτλο: Semblance based disseminated software watermarking algorithm και υπεύθυνοι για αυτήν ήταν οι Z. Pervez, Y. Mahmood και H. F. Ahmad. Οι άνθρωποι που εργάστηκαν πάνω σε αυτήν, υλοποίησαν μία τεχνική για να ενσωματώνει μιμητικές εντολές στον αντικειμενικό κώδικα χρησιμοποιώντας ένα καθορισμένο είδους αλφαριθμητικού, εν ονόματι dasiastring, προκειμένου να γίνει η χαρτογράφηση των εντολών. Καταφέρνοντας με τον τρόπο αυτόν, την εφικτή αναγνώριση του νόμιμου ιδιοκτήτη ενώ παράλληλα είναι υπεύθυνη για την διανομή των πειρατικών αντιγράφων. Επίσης αξίζει να σημειωθεί, πως αυτή η τεχνική έχει αξιολογηθεί σύμφωνα με μία δημοσιευμένη υλοποίηση, λεγόμενη Sandmark του Christian Colleberg.

Βαδίζοντας ένα βήμα πιο κάτω φτάνει η σειρά μίας άλλης εργασίας, η οποία δημοσιεύεται στις 11-04-2008, καθώς επίσης και αυτή με την σειρά της είχε τις βάσεις της στην εργασία των Rakan El Khalil and Angelos D. Keromytis. Η καινούρια αυτή εργασία έφερε τον τίτλο: Software watermarking by equation reordering. Πάνω σε αυτήν την εργασία δούλεψαν οι Shirali-Shahreza M. και Shirali-Shahreza S. υλοποιώντας έναν αλγόριθμο υδατογράφησης βασισμένο στην εξίσωση αναδιάταξης (ο αγγλικός όρος είναι equation reordering). Πιο συγκεκριμένα αυτό που έκαναν ήταν, να αλλάξουν την σειρά των τελεστών σε μία εξίσωση σύμφωνα πάντα με πληροφορίες που ήθελαν να κρύψουν με τέτοιον τρόπο έτσι ώστε το αποτέλεσμα της εξίσωσης να παραμένει αμετάβλητο. Με αποτέλεσμα, να έχουν υδατογραφήσει ένα πρόγραμμα δίχως επιπλέον κώδικα, διατηρώντας την εκτέλεσή του εξίσου γρήγορη με την εκτέλεση του αρχικού προγράμματος. Όμως, τέτοιου είδους επεμβάσεις στα λογισμικά είναι ιδιαίτερα εύθραυστες σε κακόβουλους μετασχηματισμούς. Για πληροφοριακούς λόγους αυτή η μέθοδος υδατογράφησης λογισμικού κατατάσσεται στις μεθόδους όπου δεν προστίθεται κάτι καινούριο μέσα στο λογισμικό προς υδατογράφηση, παρά μόνο αλλάζει ο ήδη υπάρχων κώδικας.

Στη συνέχεια μία εργασία με τίτλο: New approaches for software watermarking by register allocation, από τους Hakun Lee και Keiichi Kaneko, οι οποία είχε τα θεμέλια της στο πρόβλημα χρωματισμού γραφημάτων και πιο συγκεκριμένα σε μία εργασία που είχε δημοσιευτεί πριν πέντε χρόνια από τότε με τίτλο: Software watermarking through register allocation: Implementation, analysis and attacks από τους Christian Colleberg και ο Ginger Myles. Πιο συγκεκριμένα στην εργασία αυτή, τα άτομα που μελέτησαν και έγραψαν για να την βγάλουν σε πέρας αναφέρουν αρχικά κάποια προβλήματα των πλέον γνωστών αλγορίθμων QP και QPS. Τα προβλήματα αυτά ήταν πως η ανάκτηση υδατογραφήματος δεν είναι πάντοτε εφικτή λειτουργία για τον πρώτο αλγόριθμο και πως η ενσωμάτωση μπορεί να γίνει μόνο με χρήση του μικρότερου δυνατού υδατοσήματος για τον δεύτερο κατά σειρά.

Επίσης σε αυτή τους την μελέτη προτείνουν και άλλους δύο καινούριους αλγορίθμους, οι οποίοι ενσωμάτωναν ένα υδατογράφημα κάνοντας αλλαγή στα χρώματα του γραφήματος δίχως να μετατρέπουν στο ελάχιστο την ήδη υπάρχουσα δομή. Στο σημείο αυτό αξίζει να σημειωθεί πως αυτοί οι δύο τελευταίοι αλγόριθμοι μπορούσαν να ενσωματώσουν υδατόσημα οποιουδήποτε μήκους και στη συνέχεια να το εξάγουν χωρίς καμία δυσκολία.

Ακολουθώντας την χρονική ροή των δημοσιεύσεων, έρχεται και η σειρά της εργασίας με τον τίτλο: Dynamic graph watermark algorithm based on the threshold scheme, στηριζόμενη σε μία ήδη δημοσιευμένη με τίτλο: Software watermarking: models and dynamic embeddings από τους Christian Collberg, Clark Thomborson, και Gregg M. Townsend. Σε αυτήν την εργασία λαμβάνουν μέρος οι Yang Xia Luo, Jian-Hua Cheng και Ding-Yi Fang και την δημοσιεύουν στις 22-12-2008. Πιο συγκεκριμένα, τα άτομα αυτά, αυτό που έκαναν ήταν να προτείνουν έναν αλγόριθμο δυναμικού υδατογραφήματος με την τεχνική διάδοσης σε ολόκληρο το πρόγραμμα, ενώ παράλληλα έδωσαν και κάποια αποτελέσματα τα οποία έδειχναν πως ο συγκεκριμένος αλγόριθμος εμποδίζει αρκετά τους εκάστωτε εισβολείς να εντοπίσουν το υδατογράφημα μέσα στο λογισμικό.

Προχωρώντας, το σκηνικό αλλάζει και πάλι με δύο παρεμφερείς εργασίες με τίτλους: Hiding information in complete holes και Hiding software watermarks in loop structures. Οι άνθρωποι που εργάστηκαν πάνω σε αυτές τις εργασίες ήταν ο Roberto Giacobazzi, ο οποίος έλαβε μέρος και στις δύο και οι Mila Dalla Preda και Enrico Visentini οι οποίοι εργάστηκαν μόνο για την δεύτερη. Εμβαθύνοντας τώρα στις εν λόγω εργασίες, πρέπει να σημειωθεί πως η πρώτη είχε ως στόχο τη δημιουργία ενός σκοτεινού προγράμματος ή με άλλα λόγια ενός προγράμματος μέσα στο οποίο θα κρύβονταν κάποιες πληροφορίες, οι οποίες θα ήταν τοποθετημένες με τέτοιον τρόπο έτσι ώστε ο εν δυνάμει κακόβουλος χρήστης να ήταν ανίκανος να τις εξάγει από αυτό. Ενώ, από την άλλη πλευρά στην δεύτερη κατά σειρά εργασία προτείνεται μία νέα τεχνική υδατογράφησης βασισμένη στην ιδέα πως διαφορετικές σημασιολογικές περιπτώσεις μπορούν να αντληθούν από το ίδιο συντακτικό αντικείμενο. Δηλαδή, αυτό που πρότειναν τα άτομα αυτά ήταν να επικεντρωθούν σε βρόχους και να ενσωματώσουν ένα υδατογράφημα σε μία συγκεκριμένη σημασιολογική περίπτωση μοιράζοντάς το στην αντίστοιχη συντακτική δομή.

Καθώς η ιστορική αναδρομή προχωρά, στις 17-08-2008, μία εργασία με τον τίτλο: Software watermarking based on dynamic program slicing, από τους Xiaohong Deng, Guowen Xu, Guang Sun και Junfeng Man, δημοσιεύεται προτείνοντας μία καινούρια ιδέα στον τομέα υδατογράφησης λογισμικού. Πιο συγκεκριμένα, αυτό που έκαναν ήταν να διαχωρίσουν το λογισμικό σε ανοιχτές και κλειστές ενότητες, με την βοήθεια του δυναμικού διαχωρισμού ενσωματώνοντας την πληροφορία υδατογράφησης αλλάζοντας την εξάρτηση των δεδομένων μεταξύ αυτών των ενοτήτων. Στην συνέχεια, για να αναγνωρίσουν το υδατογράφημα, αυτό που έκαναν ήταν να καταγράφουν τις σχέσεις εξάρτησης καθ' όλη την διάρκεια εκτέλεσης. Έτσι, με τον τρόπο αυτό κατάφεραν να υλοποιήσουν μία τεχνική υδατογράφησης ισχυρή έναντι των αντίστροφων μηχανικών επιθέσεων (ο αγγλικός όρος είναι anti-reverse engineering) και σύμφωνα με πειράματα που έκαναν, η επίδραση αυτής της τεχνικής στον χρόνο εκτέλεσης του εκάστοτε προγράμματος είναι ιδιαιτέρως χαμηλή.

Συνεχίζοντας την αναδρομή, πλησιάζοντας στο τέλος και αυτού του έτους, έρχονται τρεις ξεχωριστές και ιδιαίτερες εργασίες με τίτλους: Source code watermarking based on function dependency oriented sequencing από τους Gupta G. και Pieprzyk J., Hash function based on software watermarking, από τους Xuesong Zhang, Fengling He και Wanli Zuo και

τέλος Hiding information on Java class file, από τους Daofu Gong, Fenlin Liu, Bin Lu, Ping Wang και Lan Ding. Στην πρώτη κατά σειρά εργασία, προτείνεται μία τεχνική υδατογράφησης για κώδικες γραμμένους σε γλώσσα προγραμματισμού C ή C++ με την βοήθεια των περιορισμών που διαθέτει η γλώσσα. Πιο συγκεκριμένα, αυτό που έκαναν ήταν να εισάγουν το υδατογράφημα στον κώδικα, επιβάλλοντας μία διάταξη σχετικά με τις ανεξάρτητες συναρτήσεις εισάγοντας ψεύτικες εξαρτήσεις μεταξύ αυτών. Με αποτέλεσμα, το υδατογράφημα να τοποθετηθεί με τέτοιον τρόπο, έτσι ώστε ο εκάστοτε κακόβουλος χρήστης να μην είναι σε θέση, κάτω από λογικές συνθήκες βέβαια, να κατανοήσει τον τρόπο με τον οποίο έγινε η ενσωμάτωσή του. Στην δεύτερη κατά σειρά εργασία, τα άτομα που συμμετείχαν σε αυτή πρότειναν μία συνάρτηση κατακερματισμού για την ενσωμάτωση υδατογραφήματος λογισμικού, η οποία βασιζόταν σε ένα δυναμικό αλγόριθμο. Με τον τρόπο αυτό, δίνοντας τις κατάλληλες παραμέτρους στην συνάρτηση κατακερματισμού, αυτή με την σειρά της σου έδινε ως αποτέλεσμα το προς εισαγωγή υδατόσημα. Τέλος στην τρίτη και τελευταία εργασία, προτείνονταν μία τεχνική βασιζόμενη στην steganography. Η τεχνική αυτή ενσωμάτωνε το υδατογράφημα μέσα σε εκτελέσιμα αρχεία από Java προγράμματα. Με τον τρόπο αυτό δημιουργούταν το υδατογραφημένο πρόγραμμα δίχως να μεταβαλλόταν το μέγεθος του αρχείου, κρατώντας παράλληλα το υδατογράφημα κρυφό.

Έτσι, το κεφάλαιο του έτους 2008 φτάνει στη δύση του, δίνοντας την σειρά του στο επόμενο και τελευταίο κατά χρονολογική σειρά έτος. Η εργασία που ανοίγει την αυλαία για τον χρόνο αυτόν είχε τις βάσεις της σε μία άλλη ήδη δημοσιευμένη πριν από δεκατρία χρόνια με τίτλο: Method and system for generating and auditing a signature for a computer program, από τους Robert Davidson και Nathan Myhrvold. Ο τίτλος λοιπόν αυτής της εργασίας ήταν: A novel method based software watermarking scheme, από τους Zhu Jianqi, Liu YanHeng, Yin Ke και Yin KeXin. Η εργασία αυτή βασίστηκε σε τρεις σημαντικές ιδιότητες των υδατογραφημάτων λογισμικού οι οποίες είναι η ανθεκτικότητα, η ορατότητα και ο ρυθμός δεδομένων με σκοπό την υλοποίηση μίας τεχνικής στατικής υδατογράφησης. Εμβαθύνοντας λίγο περισσότερο στην τεχνική αυτή, είναι φανερό πως αυτό που πρότεινε ήταν μία μυστική ανταλλαγή, η οποία εισαγόταν σε ένα τμήμα του υδατογραφήματος καθώς και της ανάκτησης, έχοντας ταυτόχρονα βασιστεί σε κάποιες ειδικές τιμές. Πιο απλά, θα μπορούσε να ειπωθεί πως η ιδέα της συγκεκριμένης τεχνικής ήταν η ενσωμάτωση του κώδικα υδατογραφήματος σε μία συγκεκριμένη μέθοδο (συνάρτηση), η οποία έχει άπειρη χωρητικότητα προς αποθήκευση υδατογραφημάτων, χωρίς παράλληλα να αυξάνεται πολύ η έκταση του αρχικού κώδικα.

Στο ίδιο μήκος κύματος, κυμαίνονται και επόμενες δύο εργασίες καθώς είχαν και οι δύο τις βάσεις τους στην δημοσίευση των Robert Davidson και Nathan Myhrvold, η οποία όπως προαναφέρθηκε είχε λάβει χώρα προ δεκατριών ετών, αλλά ήταν άμεσα επηρεασμένες και από την εργασία των Rakan El Khalil and Angelos D. Keromytis με τίτλο: Hydan: hiding information in program binaries, η οποία είχε δημοσιευτεί προ πενταετίας. Η πρώτη λοιπόν από τις δύο αυτές εργασίες, έφερε τον τίτλο: Software watermarking algorithm by coefficients of equation και οι άνθρωποί που εργάστηκαν για να την φέρουν σε πέρας ήταν οι Zonglu Sha, Hua Jiang και Aicheng Xuan. Κάνοντας μία μικρή εισαγωγή για την εν λόγω εργασία, αξίζει να σημειωθεί πως σκοπός της ήταν η υλοποίηση ενός αλγορίθμου βασισμένου στους αντίστροφους αριθμούς των τελεστών που λάμβαναν μέρος στις εξισώσεις με στόχο την μέγιστη ταχύτητα του προγράμματος και την όσο το δυνατόν καλύτερη ποιότητα κάλυψης του υδατογραφήματος. Λόγω της ένα προς ένα σχέσης μεταξύ αντίστροφων και δυαδικών αριθμών, δημιουργούταν ο ορθογραφικός χάρτης (ο όρος στο

αγγλικά είναι dictionary mapping). Η λειτουργία αυτού του χάρτη βοηθούσε στο να γίνει εφικτή η τοποθέτηση του υδατογραφήματος στο εκάστοτε λογισμικό. Με τον τρόπο αυτό γίνεται εφικτή η υδατογράφηση ενός προγράμματος δίχως περεταίρω κώδικα και δίχως να επιβαρυνθεί ο αρχικός χρόνος εκτέλεσης, δημιουργώντας παράλληλα μία τεχνική αποτελεσματικότερη από αυτήν της αναδιαταγμένης εξίσωσης, η οποία προτείνονταν στην μία εργασία του έτους 2008 με τίτλο: Software watermarking by equation reordering των Shirali-Shahreza M. και Shirali-Shahreza S.

Συνεχίζοντας γίνεται η μετάβαση στην δεύτερη εργασία με τον τίτλο: A fragile software watermarking for tamper proof, από τους Changle Zhang, Hong Peng, Xianzhong Long, Zheng Pan και Ying Wu. Σε αυτήν την εργασία προτάσσεται μία εύθραυστη τεχνική υδατογράφησης για την επαλήθευση της ακεραιότητας του λογισμικού. Με μία πιο αναλυτική προσέγγιση γίνεται φανερό πως στην εργασία αυτήν εμφανιζόταν ένας αλγόριθμος υλοποιημένος με τέτοιον τρόπο έτσι ώστε να διατηρείται η σημασιολογία του λογισμικού, αντικαθιστώντας την ίδια στιγμή κώδικα για να ενσωματώσει το υδατογράφημα. Το αποτέλεσμα της όλης διαδικασίας είναι η δημιουργία ενός υδατογραφήματος, το οποίο είναι πολύ στενά συνδεδεμένο με το αρχικό λογισμικό και παράλληλα το νέο υδατογραφημένο πλέον λογισμικό είναι εξαιρετικά εύθραυστο σε διάφορες επιθέσεις εναντίον του. Για τον λόγο αυτό, προς ενίσχυση της τεχνικής, το παραγόμενο υδατογράφημα καθώς και η θέση στην οποία θα τοποθετηθεί ελέγχονται από δύο βασικά και εντελώς διαφορετικά κλειδιά έτσι το σύστημα να είναι ικανό να εντοπίζει εισβολείς καθώς και το είδος των αλλοιώσεων που προκαλούνται κάθε φορά από τις ενέργειες αυτών.

Συνεχίζοντας, πλησιάζοντας όλο και πιο κοντά στην ολοκλήρωση της πλήρους ιστορικής αναδρομής στον τομέα των υδατογραφημάτων λογισμικού, έρχεται η δημοσίευση της εργασίας με τίτλο: A software watermarking method based on public key cryptography and graph coloring, στην οποία και έλαβαν μέρος οι Zetao Jiang, Rubing Zhong και Bina Zheng. Η εργασία αυτή, όπως και πολλές άλλες ήταν επηρεασμένη από μία δημοσίευση, η οποία είχε λάβει χώρα πριν από έντεκα χρόνια και είχε τον τίτλο: Analysis of watermarking techniques for graph coloring problem, από τους Gang Qu και Miodrag Potkonjak. Πιο συγκεκριμένα όμως, τις άμεσες επιρροές τις είχε από μία εργασία του 2006 με τον τίτλο: Algorithms to watermark software through register allocation από τους William Zhu και Clark Thomborson. Στην εργασία αυτή γίνονταν λόγος, για μία καινούρια μέθοδο υδατογράφησης βασιζόμενης σε κρυπτογράφηση δημόσιου κλειδιού καθώς και στο πρόβλημα χρωματισμού γραφήματος (ο αγγλικός όρος είναι graph coloring problem). Με άλλα λόγια οι πληροφορίες που κρύβει μέσα το υδατογράφημα κρυπτογραφούνται από τον αλγόριθμο της μεθόδου με την βοήθεια του δημοσίου κλειδιού και ενσωματώνονται στο πρόγραμμα με την βοήθεια της τεχνικής χρωματισμού των γραφημάτων. Τέλος στην εργασία αυτή αναφέρονται και κάποια πειραματικά αποτελέσματα για το πόσα χρώματα χρειάζονται για αυτήν την διαδικασία.

Την ίδια περίοδο περίπου δημοσιεύεται και μία εργασία με τίτλο: Two new algorithms for software watermarking by register allocation and their empirical evaluation, από τους Hakun Lee και Kaneko K. Τα άτομα αυτά, ένα χρόνο πριν είχαν δημοσιεύσει άλλη μία εργασία, η οποία συνέβαλλε σε αυτήν, με τον τίτλο: New approaches for software watermarking by register allocation, η οποία είχε και αυτήν με την δική της σειρά τα θεμέλιά της σε μία δημοσίευση προ δέκα ετών με τίτλο: Analysis of watermarking techniques for graph coloring problem, από τους Gang Qu και Miodrag Potkonjak. Στην εργασία αυτή λοιπόν προτάσσονταν δύο νέοι αλγόριθμοι για ενσωμάτωση υδατογραφήματος, ενώ

παράλληλα διασφάλιζαν την εξαγωγή του από το υδατογραφημένο πλέον λογισμικό σε κάθε περίπτωση.

Στην συνέχεια του έτους, έχοντας ως βάση την εργασία των J. Palsberg, S. Krishnaswamy, Minseok Kwon, D. Ma, Qiuyun Shao και Y. Zhang με τίτλο: Experience with software watermarking, η οποία είχε δημοσιευτεί πριν από εννιά χρόνια, δημοσιεύεται μία νέα εργασία από τους Jianqi Zhu, Kexin Yin και Yanheng Liu, με τον τίτλο: A novel dgw scheme based on 2d_ppct and permautation. Στην εργασία αυτή προτείνονταν μία καινούρια μέθοδος υδατογράφησης, η οποία συνδύαζε μεταθέσεις και 2d_ppct, με αποτέλεσμα να κάνει πλήρης χρήση των διαρθρωτικών πλεονεκτημάτων της ppct, έχοντας ταυτόχρονα υψηλές επιδόσεις στις κωδικοποιήσεις μεταθέσεων.

Βαδίζοντας ένα βήμα πιο κάτω, έρχεται άλλη μία εργασία βασισμένη στην ίδια ακριβώς δημοσίευση με την προηγούμενη, με τον τίτλο: A novel dynamic graph software watermark scheme από τους Jianqi Zhu, Yanheng Liu και Kexin Yin. Στην εργασία αυτήν προτάσσονται δύο καινούριες δομές υδατογράφησης λογισμικού. Η πρώτη δομή είναι δύο διαστάσεων και έχει εξαιρετικά μεγάλο ρυθμό δεδομένων, ενώ η δεύτερη κάτω από ορισμένες συνθήκες, όπως ο ίδιος αριθμός κόμβων φύλλων, ο ρυθμός δεδομένων φτάνει στο ανώτατο όριο του, με τον αριθμό διάστασης να αυξάνεται παράλληλα και αυτός. Με άλλα λόγια η δομή 2D_IPPCT, έχει πολύ ικανοποιητικό ποσοστό κωδικοποίησης.

Συνεχίζοντας πάνω στο ίδιο σκεπτικό υδατογράφησης, δημοσιεύονται άλλες δύο εργασίες. Ο τίτλος της πρώτης κατά σειρά ήταν: Evaluating effectiveness of tamper proofing on dynamic graph software watermarks, δημοσιευμένη από τους Malik Sikandar, Hayat Khiyal, Aihab Khan, Sehrish Amjad και M. Shahid Khalil και δημοσιεύτηκε έχοντας ως σκοπό να βελτιώσει το επίπεδο προστασίας των δυναμικών γραφημάτων υδατοσύμανσης λογισμικού. Κάνοντας μία μικρή παύση στο σημείο αυτό, αξίζει να σημειωθεί πως οι βάσεις αυτής της εργασίας, όπως και της επόμενης που θα αναφερθεί βρισκόντουσαν σε μία εργασία, η οποία είχε δημοσιευτεί πριν από δέκα χρόνια με τον τίτλο: Software watermarking: models and dynamic embeddings, από τους Christian Collberg, Clark Thomborson, και Gregg M. Townsend, η οποία ήταν η αρχή μίας καινοτόμου ιδέας και παράλληλα το έναυσμα για την δημιουργία της εργασία με τίτλο: Tamperproofing a software watermark by encoding constants από τον Yong He, πάνω στην οποία οι Malik Sikandar Hayat Khiyal, Aihab Khan, Sehrish Amjad και M. Shahid Khalil στηρίχτηκαν για να ξεκινήσουν την δική τους εργασία. Πιο συγκεκριμένα, σε αυτή τους την εργασία εφάρμοσαν μία τεχνική του Clark Thomborson μαζί με την τεχνική του σπασίματος σταθεράς (ο αγγλικός όρος είναι breaking constant), η οποία δίνει την δυνατότητα όλες οι σταθερές να κωδικοποιηθούν χωρίς καμία επιβάρυνση στις τιμές τους, έχοντας πάντα σεβασμό στην τιμή υδατογράφησης. Όμως πειραματικές αναλύσεις, για αυτήν την μέθοδο υδατογράφησης έδειξαν πως αυξάνει σημαντικά τον μέγεθος του κώδικα καθώς και τον χρόνο εκτέλεσης του. Από την άλλη πλευρά, η δεύτερη εργασία έφερε τον τίτλο: A dynamic graph watermark scheme of tamper resistance και δημοσιεύτηκε στις 20-08-2009 από τους XiaoJiang Chen, DingYi Fang, JingBo Shen, Feng Chen, WenBo Wang, Lu He. Σε αυτήν την δημοσίευση πρότειναν μία καινούρια τεχνική υδατογράφησης λογισμικού, στην οποία δημιουργούσαν πολλά ψεύτικα υδατογραφήματα λογισμικού, κωδικοποιώντας πολλαπλές σταθερές, τα οποία (δηλαδή , η δομή τους) ήταν παρόμοια με το μοναδικό αληθινό και ενσωματωνόντουσαν με σκοπό την ενίσχυση της όλης τεχνικής, υλοποιώντας με τον τρόπο αυτό ένα λογισμικό ικανοποιητικά ασφαλές από διάφορες επιθέσεις.

Στο σημείο αυτό, η οπτική γωνία αλλάζει και πάλι δίνοντας την σκυτάλη σε τρεις εργασίες άμεσα επηρεασμένες από την εργασία των Ginger Myles και Hongxia Jin με τον τίτλο: Self validating branch based software watermarking. Η πρώτη από αυτές είχε τον τίτλο: A robust dynamic software watermarking και τα άτομα που εργάστηκαν πάνω σε αυτήν ήταν οι Yin Ke xin, Yin Ke και Zhu Jian qi. Αρχικά, οι άνθρωποι αυτοί πρότειναν μία τεχνική, σύμφωνα με την οποία το υδατογράφημα διαχωρίζονταν σε κομμάτια χρησιμοποιώντας μία ήδη δημοσιευμένη μέθοδο, η οποία ήταν υλοποιημένη και σχετίζονταν με την πλήρη και σωστή ανάκτηση του υδατογραφήματος χρησιμοποιώντας βασικές πληροφορίες, ενώ παράλληλα αύξανε την ανθεκτικότητα της υδατογράφησης. Πιο συγκεκριμένα, στην τεχνική αυτή, τα προαναφερθέντα κομμάτια έτρεχαν μέσω μία μονοδιάστατης λογιστικής-χαοτικής συνάρτησης κρυπτογράφησης, πριν ενσωματωθούν στην δυναμική δομή του προγράμματος, έχοντας ως αποτέλεσμα δημιουργήσει ένα λογισμικό εύρωστο σε πολλές επιθέσεις. Κλείνοντας με την εργασία αυτή, αξίζει να τονιστεί πως η τεχνική διόρθωσης λαθών θα μπορούσε να χρησιμοποιηθεί για να αυξήσει περεταίρω την ανθεκτικότητα του συγκεκριμένου υδατογραφήματος.

Στην συνέχεια, γίνεται η μεταφορά στην δεύτερη κατά χρονολογική σειρά εργασία των Christian Collberg, Clark Thomborson και Gregg M. Townsend με τον τίτλο: Dynamic data flow graph based software watermarking. Εμβαθύνοντας, σε αυτήν την εργασία γίνεται φανερό πως πρότειναν έναν αλγόριθμο βασισμένο στα δυναμικά δεδομένα που δημιουργούνται κατά την διάρκεια εκτέλεσης του κώδικα. Δηλαδή, τα άτομα αυτά ενσωμάτωναν το υδατογράφημα μέσα στο DDFG (dynamic data flow graph, το οποίο στα ελληνικά σημαίνει δυναμικό γράφημα δεδομένων) του προγράμματος και στην συνέχεια συζητούσαν λεπτομέρειες για την εισαγωγή, την εξαγωγή και την γενίκευση αυτού. Με τον τρόπο αυτό, η τρίτη κατά σειρά εργασία έρχεται για να δώσει ένα προσωρινό τέλος στην ιδέα των δυναμικών υδατογραφημάτων λογισμικού και είχε τον τίτλο: A robust dynamic watermarking scheme based on STBDW. Σε αυτήν την εργασία έλαβαν μέρος οι Zhu Jian-qi, Liu Yan heng, Yin Ke και Yin Ke xin, έχοντας ως κύριο στόχο την δημιουργία μίας εύρωστης και δυναμικής τεχνικής υδατογραφήσεως. Η τεχνική αυτή, χρησιμοποιούσε μία ήδη δημοσιευμένη μέθοδο για να διαχωρίσει την τιμή υδατογράφησης, το λεγόμενο υδατόσημα, σε κομμάτια τα οποία βοηθούσαν στην ανάκτηση του αυθεντικού υδατοσήματος χρησιμοποιώντας παράλληλα βασικές πληροφορίες δημιουργώντας έναν κώδικα ανθεκτικό σε διάφορες κακόβουλες επιθέσεις εναντίον του. Πιο συγκεκριμένα τα υλοποιημένα κομμάτια έτρεχαν μέσω των μπλοκ κρυπτογράφησης και των λειτουργιών σχετικών με ισομορφικούς χάρτες (ο αγγλικός όρος είναι self isomorphic mapping operations), δημιουργώντας με τον τρόπο αυτό ένα είδος σύνδεσης μεταξύ των άσχετων κομματιών. Αξίζει βέβαια να σημειωθεί πως, η διαδικασία αυτή λάμβανε χώρα στην δυναμική δομή που δημιουργούταν κατά την εκτέλεση του προγράμματος και έφτιαχνε ένα λογισμικό ανθεκτικό σε επιθέσεις σημασιολογικής διατήρησης του κώδικα καθώς και σε κακόβουλους μετασχηματισμούς (οι αγγλικοί όροι είναι semantic preserving code transformations attacks και obfuscations attacks).

Με τον τρόπο αυτό, έρχεται και η σειρά των τελευταίων δύο εργασιών, οποίες τοποθετούν την πλήρη ιστορική αναδρομή των υδατογραφημάτων λογισμικού ένα βήμα πριν από το τέλος της. Η πρώτη από αυτές είχε τον τίτλο: A semi dynamic multiple watermarking scheme for Java applications, από τους Changjiang Zhang, Jianmin Wang, Clark Thomborson, Chaokun Wang και Christian Collberg. Τα άτομα αυτά, είχαν προτείνει μία τεχνική, η οποία ξέφευγε κάπως από την λογική της απευθείας ενσωμάτωσης

υδατογραφήματος, είτε στο κώδικα, είτε σε δομές που δημιουργούνται κατά την εκτέλεση του προγράμματος. Αυτό που έκανε ήταν, να προτείνει μία νέα μέθοδο υδατογράφησης, η οποία δημιουργούσε πολλαπλά υδατογραφήματα, δίχως να όμως να τα ενσωματώνει στο απευθείας στον κώδικα. Πιο συγκεκριμένα κωδικοποιούσε το υδατογράφημα μέσα συναρτήσεις χαρτογράφησης και στην συνέχεια ενσωμάτωνε τους κώδικες χαρτογράφησης, οι οποίοι είχαν δημιουργηθεί μέσα σε αυτές τις συναρτήσεις, στα σημεία άρθρωσης του διαγράμματος έλεγχου ροής. Αξίζει να τονιστεί επίσης πως με την τεχνική αυτή, η αρχική έκταση του προγράμματος καθώς και η επίδοση αυτού παραμένουν σταθερές. Συνεχίζοντας δίχως φλυαρίες, πλησιάζοντας όλο και περισσότερο στη στην ολοκλήρωση της αναδρομής, έρχεται και η σειρά της δεύτερης εργασίας με τον τίτλο: Feature n - gram set based software zero watermarking από τους Bin Lu, Fenlin Liu, Xin Ge και Ping Wang. Τα άτομα αυτά, σε αυτήν την εργασία αυτή πρότειναν την ενσωμάτωση μίας ειδικής εικόνας μέσα στο λογισμικό υδατογράφησης σε ένα σημείο κατάλληλα επιλεγμένο.

Κλείνοντας, φτάνει και η σειρά των Christian Collberg και Jasvir Nagra οι οποίοι έγραψαν το βιβλίο με τον τίτλο: Surreptitious Software: obfuscation, watermarking and tamperproofing for software protection. Σε αυτό το βιβλίο γίνονται εκτενής αναφορές σε υδατογραφήματα λογισμικού, σε τρόπους που μπορούν αυτά να ενσωματωθούν, σε πιθανούς μετασχηματισμούς που μπορούν να λάβουν χώρα στα εκάστοτε λογισμικά και σε πολλά άλλα και ενδιαφέροντα θέματα πάνω σε αυτόν τον τομέα. Με τον τρόπο αυτό οι υδατογραφήσεις λογισμικού γίνονται ποιο ξεκάθαρες και σε αρχάριους πάνω σε αυτόν τον τομέα, αλλά και σε έμπειρους προσφέροντάς τους την δυνατότητα να έχουν μία οργανωμένη δίοδο σε πολλά θέματα που ίσως τους ενδιαφέρουν, διότι καθώς περνά ο καιρός όλο και περισσότερα λογισμικά δημιουργούνται και η προστασία τους τείνει να γίνει απαραίτητη.

Παρακάτω παρατίθενται δύο εικόνες στις οποίες φαίνονται, από την σατιρική άποψη, οι αλλαγές που έχουν δημιουργηθεί στην ζωή όλων χάρης στα εκάστοτε λογισμικά:

Έτσι ούτε λίγο ούτε πολύ, ολοκληρώνεται η πλήρης ιστορική αναδρομή στα υδατογραφήματα λογισμικού σύμφωνα με την χρονολογική σειρά που λάμβαναν μέρος οι εκάστοτε εργασίες από το έτος 1992 ως και το 2009.

2

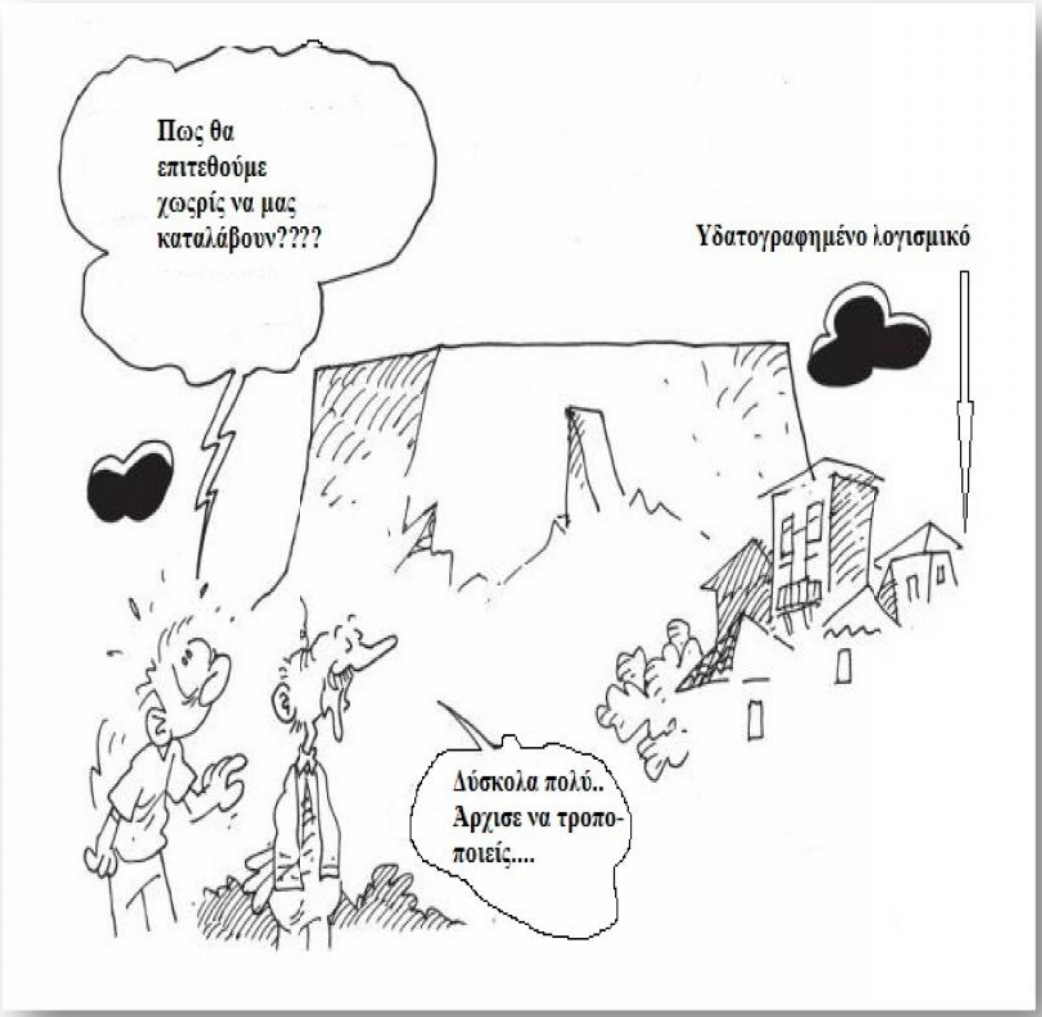
Βασικές Έννοιες - Ορισμοί

2.1 Οι βάσεις της παρούσας πτυχιακής εργασίας

Στην παρούσα πτυχιακή εργασία, ο πρώτος στόχος είναι να προταθούν τέσσερις αλγόριθμοι για υδατογράφηση λογισμικού, οι οποίοι είναι υλοποιημένοι σε δύο ξεχωριστά προγράμματα (το ένα είναι υπεύθυνο για την δημιουργία του υδατογραφήματος το οποίο δε είναι άλλο από ένα μεταθετικό αναγώγιμο γράφημα, έχοντας ως είσοδο έναν φυσικό αριθμό και είναι η ορθή διαδικασία και το άλλο είναι υπεύθυνο για την μετατροπή του υδατογραφήματος αυτού στον φυσικό αριθμό από τον οποίο και προήλθε, η οποία είναι και η ακριβώς αντίστροφη διαδικασία). Στο σημείο όμως αυτό, αξίζει να σημειωθεί πως στο δεύτερο πρόγραμμα, στην αντίστροφη διαδικασία δηλαδή, είναι ενσωματωμένοι όλοι οι πιθανοί έλεγχοι που αφορούν, τις εκάστοτε επιθέσεις που ενδεχομένως να δεχτεί το υδατογράφημα από τους εισβολείς λογισμικού και οι οποίοι αποτελούν και τον δεύτερο κατά σειρά στόχο.

Αυτές οι επιθέσεις, μπορούν να λάβουν χώρα στο μεταθετικό αναγώγιμο γράφημα (το ενσωματωμένο υδατογράφημα) και να επηρεάζουν διάφορα τμήματά του, ορατά ή μη. Τα ορατά είναι οι κόμβοι, οι ετικέτες των κόμβων και οι ακμές και τα μη ορατά είναι η αυτοαναστρέφουσα μετάθεση, η bitonic permutation και η δυαδική δομή (Β'). Πιο συγκεκριμένα, οι εκάστοτε εισβολείς μπορούν να επιτεθούν είτε στους κόμβους τους υδατογραφήματος προσθέτοντας, αφαιρώντας κόμβους ή ακόμα και αλλάζοντας όλες τις ετικέτες τους, είτε στις ακμές με τον ίδιο τρόπο (εξαιρώντας βέβαια την αλλαγή ετικετών καθώς οι ακμές δεν έχουν ετικέτες), προσέχοντας βέβαια να διατηρούν τις ιδιότητες του γραφήματος μετά από κάθε αλλαγή (υπό την προϋπόθεση πως οι εισβολείς γνωρίζουν τις ιδιότητες που σχετίζονται με το γράφημα). Επιπλέον, οι εισβολείς μπορούν όπως προαναφέρθηκε να επιτεθούν σε κρυμμένα τμήματα του υδατογραφήματος όπως στην αυτοαναστρέφουσα μετάθεση, στην bitonic permutation και στην δυαδική δομή, πάλι βέβαια τροποποιώντας το ίδιο το μεταθετικό αναγώγιμο γράφημα (υδατογράφημα), αυτό όμως προϋποθέτει πλήρη γνώση του αλγορίθμου αποκωδικοποίησης σε περίπτωση που οι κακόβουλοι χρήστες δεν θέλουν να γίνει αντιληπτή η επίθεσή τους.

Παρακάτω παρατίθεται μία σατιρική εικόνα στην οποία φαίνεται η δυσκολία που αντιμετωπίζουν οι εκάστοτε εισβολείς στον χειρισμό του υδατογραφήματος:



Συνεχίζοντας γίνεται η μετάβαση στον τρίτο κατά σειρά στόχο αυτής της εργασίας. Ο στόχος αυτός σχετίζεται με την παρουσίαση δύο μεθόδων ενσωμάτωσης του υδατογραφήματος που δημιουργείται από τον ορθό αλγόριθμο υδατογράφησης στο λογισμικό και στη συνέχεια παρατίθεται ένα παράδειγμα υδατογράφησης λογισμικού με τα αντίστοιχα σχόλια. Τέλος, έρχεται η σειρά του τέταρτου κατά σειρά στόχου, ο οποίος είναι η πειραματική μελέτη επιθέσεων. Δηλαδή, γίνονται κάποιες επιθέσεις σε υδατογραφήματα (μεταθετικά αναγώγιμα γραφήματα) σε όλα τα επίπεδά τους και εξάγονται οι πιθανότητες των επιθέσεων αυτών είτε από την πλευρά των εισβολέων είτε, από την πλευρά του νόμιμου ιδιοκτήτη.

Εντούτοις, οι στόχοι αυτοί καθώς και όλη η παρούσα πτυχιακή εργασία έχει της βάσεις της σε εργασίες οι οποίες έχουν ήδη δημοσιευτεί και είναι σχετικές βέβαια με τα υδατογραφήματα λογισμικού. Πιο συγκεκριμένα, ένα πρώτο έναυσμα έδωσε μία εργασία των Davidson και Myhrvold με τον τίτλο: Method and system for generating and auditing a

signature for a computer program, στην οποία καθώς και σε υπόλοιπες που θα ακολουθήσουν σε αυτήν την περιγραφή, γίνεται αναφορά στο κεφάλαιο ένα και πιο συγκεκριμένα στο υποκεφάλαιο με τον τίτλο: Πλήρης ιστορική αναδρομή.

Για λόγους καλύτερης κατανόησης, αλλά παράλληλα και συνοχής, αναφέρεται πως σε αυτήν την εργασία προτείνονταν ένα αλγόριθμος υδατογράφησης λογισμικού, ο οποίος ήταν στατικός και ενσωματώνονταν στο εκάστοτε λογισμικό αναδιατάσσοντας τα βασικά μπλοκ του διαγράμματος ελέγχου. Βαδίζοντας, πάνω στις εργασίες που αποτελούν την βάση αυτής της πτυχιακής εργασίας, έρχεται και η σειρά της εργασίας των Venkatesan, Vazirani και Sinha, με τίτλο: A graph theoretic approach to software watermarking και είναι επηρεασμένη από την προηγούμενη εργασία των Davidson και Myhrvold. Στην εργασία αυτή λοιπόν, προτάσσεται ένας αλγόριθμος υδατογράφησης λογισμικού, ο οποίος είναι βασισμένος στα γραφήματα, πιο συγκεκριμένα ενσωματώνει το υδατογράφημα στο λογισμικό επεκτείνοντας τον προαναφερθέν αλγόριθμό των Davidson και Myhrvold μέσω της εισαγωγής ενός κατευθυνόμενου υπογραφήματος (δηλώνει πως και αυτός με την σειρά του είναι ένας στατικός αλγόριθμος).

Τέλος έρχεται και η σειρά των Christian Collberg και των συνεργατών του για να ολοκληρώσουν τις βάσεις της παρούσας εργασίας με την εφαρμογή του ακριβώς προηγούμενου αλγορίθμου στην εργασία τους με τον τίτλο: Graph theoretic software watermarks: implementation, analysis, and attacks. Εμβαθύνοντας λίγο στην εργασία αυτή, γίνεται φανερό πως το υδατογράφημα κωδικοποιείται σε μία μορφή με όνομα μεταθετικό αναγώγιμο γράφημα (γνωστό και με τον αγγλικό όρο ως reducible permutation graph) μέσα στην οποία βρίσκεται κρυμμένος ένας φυσικός αριθμός, χρησιμοποιώντας μία αυτοαναστρέφουσα μετάθεση. Κλείνοντας την περιγραφή αυτή, σημειώνεται πως οι Collberg και Thomborson μετά από ένα χρονικό διάστημα δημοσιεύουν και ένα δυναμικό αλγόριθμο υδατογράφησης λογισμικού σε μία εργασία τους, ενσωματώνοντας ένα υδατογράφημα μέσω μίας δομής βασισμένης σε γράφημα, η οποία χτίζεται, δημιουργώντας σωρούς, κατά τον χρόνο εκτέλεσης του προγράμματος.

2.2 Βασικοί ορισμοί

Για να συνεχίσει να είναι η παρούσα πτυχιακή εργασία ευχάριστη και συγχρόνως πιο κατανοητή στο σημείο αυτό γίνεται παράθεση κάποιων χρήσιμων ορισμών και διευκρινήσεων πάνω σε αντικείμενα τα οποία και θα συζητηθούν εκτενώς στην συνέχεια.

Ορισμός 1. W: είναι το υδατόσημα που πρόκειται να ‘κρυφτεί’ σε ένα μεταθετικό αναγώγιμο γράφημα (ο όρος στα αγγλικά είναι reducible permutation graph). Στην συγκεκριμένη περίπτωση είναι ένας φυσικός αριθμός από το 4 έως το 127.

Ορισμός 2. S.i.p: είναι μία αυτοαναστρέφουσα μετάθεση (ο όρος στα αγγλικά είναι self inverting permutation). Πιο συγκεκριμένα, το αντίστροφο μίας μετάθεσης ($\pi_1, \pi_2, \ldots, \pi_n$) είναι η μετάθεση ($q_1, q_2, \ldots, q_n$) με $q_{\pi i} = \pi_{qi} = i$. Επομένως, η αυτοαναστρέφουσα μετάθεση είναι από μόνη της αντίστροφη, δηλαδή ππi = i. Αξίζει να σημειωθεί, πως κάθε κύκλος μίας τέτοιας ακολουθίας έχει μήκος είτε, 1είτε, 2.

Ορισμός 3. D.a.g: είναι ένα κατευθυνόμενο γράφημα δυνατά συνδεδεμένο, στο οποίο υπάρχει πάντα μονοπάτι ανάμεσα σε δύο κόμβους.

Ορισμός 4. Reducible permutation graph ή τελικό γράφημα: είναι ένα μεταθετικό αναγώγιμο γράφημα. Αξίζει να σημειωθεί, πως αυτό το γράφημα είναι η τελική μορφή υδατογράφησης η οποία και θα ενσωματωθεί σε ένα πρόγραμμα. Επίσης κάθε κόμβος σε αυτό το γράφημα είναι προσπελάσιμος από τον αρχικό κόμβο του γραφήματος.

Ορισμός 5. Didomination relation: έστω πως έχουμε μία μετάθεση π_1, …., π_n. Στην συνέχεια σε αυτήν την μετάθεση, το i στοιχείο κυριαρχεί (ο όρος στα αγγλικά είναι dominates) του j στοιχείου, αν i > j και η θέση του i είναι μικρότερη από την θέση του j. Τώρα, αν το i, κυριαρχεί του j και δεν υπάρχει κάποιο άλλο στοιχείο k, τέτοιο ώστε το i να κυριαρχεί του k και το k να κυριαρχεί του j, τότε έχουμε την απευθείας κυριαρχία (ο όρος στα αγγλικά είναι didomination relation).

Ορισμός 6. Σχέση γειτνίασης: είναι ποιοι κόμβοι συνδέονται με τον i κόμβο (όπου i είναι ο κόμβος ενός γραφήματος) μέσω ακριβώς μίας ακμής. Με άλλα λόγια η γειτονιά απόστασης ένα, ενός κόμβου.

Ορισμός 7. D.f.s διάσχιση: είναι η διάσχιση κατά βάθος σε ένα γράφημα. Πιο συγκεκριμένα έστω πως βρίσκεται κάποιος σε ένα κόμβο ενός γραφήματος η σάρωση των κόμβων από το σημείο αυτό και κάτω θα γίνει από το αριστερότερο παιδί του κόμβου και κάτω. Μόλις αυτός φτάσει στο τέλος του μονοπατιού οπισθοχωρεί προς τα πάνω μέχρι να εντοπίσει κόμβο με παιδί που δεν έχει επισκεφθεί και από εκεί συνεχίζει με τον ίδιο ακριβώς τρόπο.

Ορισμός 8. Bitonic permutation: έστω μία μετάθεση $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$. Για να καλείται αυτή η μετάθεση bitonic permutation πρέπει είτε, μονοτονικά να αυξάνεται και μετά μονοτονικά να μειώνεται είτε μονοτονικά να μειώνεται και μετά μονοτονικά να αυξάνεται. Για παράδειγμα οι μεταθέσεις, π = (1, 4, 6, 7, 5, 3, 2) και π = (6, 4, 3, 1, 2, 5, 7) είναι αμφότερες bitonic permutation.

3

Αλγόριθμοι και Παραδείγματα Υδατογράφησης Λογισμικού

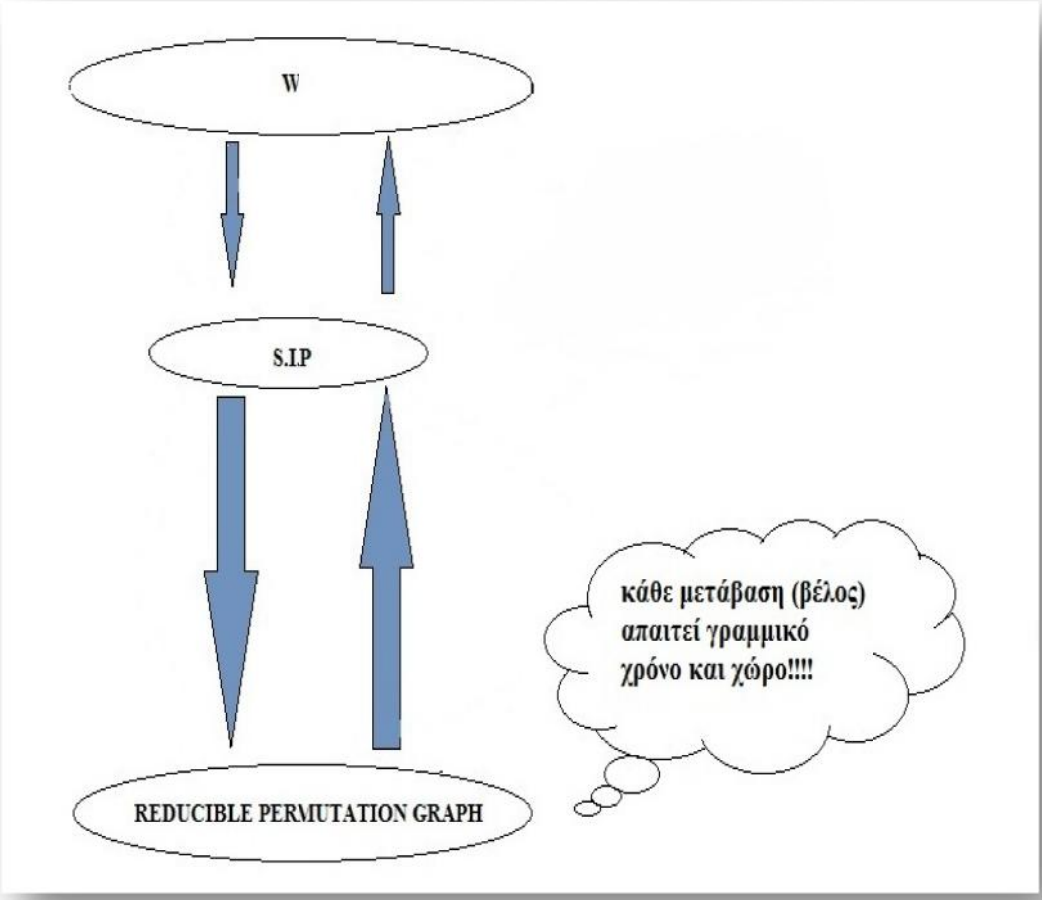
3.1 Εισαγωγή στους τέσσερις αλγορίθμους υδατογράφησης

Ο πρώτος στόχος αυτής της πτυχιακής εργασίας είναι η κωδικοποίηση ενός υδατοσήματος έστω w (φυσικός αριθμός), σε μία αυτοαναστρέφουσα μετάθεση και στην συνέχεια η μεταφορά από την αυτοαναστρέφουσα μετάθεση σε ένα μεταθετικό αναγώγιμο γράφημα, καθώς και το αντίστροφο αυτής της διαδικασίας. Με αποτέλεσμα την δημιουργία μίας δομής προς ενσωμάτωση, η οποία είναι ιδιαίτερα δύσκολη στην αποκωδικοποίησή της από κάποιον εισβολέα. Επίσης, η μορφής της την βοηθά στο να ενσωματωθεί εύκολα σε ένα πρόγραμμα δίχως να γίνει αντιληπτή από κακόβουλα εργαλεία τα οποία επεμβαίνουν στον κώδικα του λογισμικού καθώς έχει την μορφή διαγράμματος ελέγχου ροής που έχει και το εκάστοτε λογισμικό. Ο χρόνος και ο χώρος που απαιτείται για την μετατροπή αυτή καθώς και για την αντίστροφή της είναι γραμμικός (O(n)), πράγμα που δηλώνει πως πολύ γρήγορα, εύκολα και με ελάχιστο κόστος, δημιουργείτε μία δομή έτοιμη προς ενσωμάτωση σε ένα λογισμικό.

Παρακάτω υπάρχουν τέσσερις αλγόριθμοι που υλοποιούν αυτήν την διαδικασία, οι δύο πρώτοι είναι υπεύθυνοι για την ορθή μετατροπή και οι επόμενοι δύο για την αντίστροφη. Επιπλέον μετά από κάθε διαδικασία μετατροπής (από τον φυσικό αριθμό στο μεταθετικό αναγώγιμο γράφημα και το αντίστροφο), βρίσκονται και δύο παραδείγματα εκτέλεσης προκειμένου να γίνει καλύτερη και πιο ευχάριστη η κατανόησή τους.

Σημειώνεται πως τα παραδείγματα αυτά έχουν εκτελεστεί στο λειτουργικό σύστημα των Linux -os 10.10 και είναι παρμένα απευθείας από το τερματικό στο οποίο και έγινε η εκτέλεση.

Παρακάτω, παραπείθεται μία εικόνα σχετική με τον πρώτο στόχο, και πιο συγκεκριμένα με την ροή των τεσσάρων αλγορίθμων, για να γίνει καλύτερα αντιληπτή η σειρά με την οποία εφαρμόζονται οι αλγόριθμοι για την δημιουργία ενός υδατογραφήματος:



3.2 Πρώτος αλγόριθμος (w to s.i.p)

Αρχικά, η διαδικασία ξεκινά με την είσοδο ενός φυσικού αριθμού, ο οποίος βρίσκεται μεταξύ του 4 και του 127, στον αλγόριθμο και ύστερα από μία σειρά από αυστηρά καθορισμένες και εκτελέσιμες σε πεπερασμένο χρόνο εντολές δημιουργείται μία αυτοαναστρέφουσα μετάθεση (ο όρος στα αγγλικά είναι self inverting permutation).

Η όλη διαδικασία μέχρι και την μετάθεση αυτήν χωρίζεται σε έξι βήματα, τα οποία και αποτελούν τον πρώτο κατά σειρά αλγόριθμο.

- Βήμα 1. Ανάθεση σε μία μεταβλητή, έστω w, την τιμή προς υδατογράφηση και στην συνέχεια μετατροπή της εν λόγω τιμής σε δυαδική μορφή. Αποθήκευση της τιμής αυτής σε έναν πίνακα, έστω B, n θέσεων.

- Βήμα 2. Δημιουργία ενός πίνακα, έστω B′, ο οποίος έχει σε αριθμό τα διπλά συν ένα στοιχεία του B (n′ = (2 * n) + 1, όπου n τα στοιχεία του πίνακα B και n′ τα στοιχεία του B′). Τα πρώτα n στοιχεία του B’ είναι μηδενικά, τα υπόλοιπα n είναι αυτά του πίνακα B (με την ίδια σειρά, βέβαια) και το τελευταίο στοιχείο είναι ο αριθμός ένα. Στην συνέχεια, δημιουργείτε ένας πίνακας B*, n′ θέσεων, ο οποίος είναι το συμπλήρωμα του B′.

- Βήμα 3. Δημιουργία δύο πινάκων, έστω X και Y, οι οποίοι αποτελούνται από τις θέσεις των μηδενικών και των άσσων αντίστοιχα του πίνακα B*, ξεκινώντας την αρίθμηση φυσικά από τα αριστερά προς τα δεξιά.

- Βήμα 4. Δημιουργία της bitonic permutation (μετάθεση στην οποία τα πρώτα k, όπου k < n′, στοιχεία είναι τοποθετημένα κατά αύξουσα σειρά και τα υπόλοιπα κατά φθίνουσα σειρά), με τον εξής τρόπο. Τοποθετώντας τα στοιχειά του πίνακα X στη σειρά και στην συνέχεια τοποθετώντας τα στοιχεία του πίνακα Y κατά την αντίστροφη φορά. Για παράδειγμα, έστω X = (x_1, …., x_k) και Y = (y_1, …, y_m), τότε bitonic permutation = (x_1, …, x_k, y_m, …, y_1).

- Βήμα 5. Επιλογή των στοιχείων από την bitonic permutation με τέτοιον τρόπο έτσι ώστε να δημιουργηθούν τα επιθυμητά ζευγάρια. Πιο συγκεκριμένα, με την δημιουργία από το προηγούμενο βήμα της bitonic permutation = (x_1, …, x_k, y_m, …, y_1) = (π_1, π_2, …, π_n′) και έστω i = 1 και j = n′ τα ζευγάρια επιλέγονται με τον εξής τρόπο. Για κάθε i < j, δημιουργία του ζευγαριού (π_i, π_j) (κύκλος μήκους δύο). Στην συνέχεια, αυξάνετε η τιμή του i κατά ένα, ενώ παράλληλα μειώνεται η τιμή του j κατά ένα. Επιπλέον, σε περίπτωση όπου το i = j, τότε δημιουργείτε το ζευγάρι (π_i, π_i) (κύκλος μήκους ένα).

- Βήμα 6. Δημιουργία και επιστροφή της αυτοαναστρέφουσας μετάθεσης σύμφωνα με τον εξής τρόπο. Έστω ένα ζευγάρι (π_i, π_j) το οποίο δημιουργήθηκε στο βήμα 5, τότε το π_i θα μπει στην θέση π_j και το π_j θα μπει στη θέση π_i της αυτοαναστρέφουσας μετάθεσης.

3.3 Δεύτερος αλγόριθμος (s.i.p to r.p.g)

Συνεχίζοντας, γίνεται η μεταφορά στον δεύτερο κατά σειρά αλγόριθμο ο οποίος μετατρέπει μία αυτοαναστρέφουσα μετάθεση σε ένα μεταθετικό αναγώγιμο γράφημα. Αυτός ο αλγόριθμος, δημιουργεί δηλαδή την τελική μορφή στην οποία μετατρέπεται ένας φυσικός αριθμός, η οποία και είναι έτοιμη να υδατογραφήσει, με την ενσωμάτωσή της, κάποιο λογισμικό.

Η όλη διαδικασία μεταφοράς από την αυτοαναστρέφουσα μετάθεση στο τελικό γράφημα χωρίζεται στα πέντε ακόλουθα βήματα.

- Βήμα 1. Δημιουργία ενός άκυκλου κατευθυνόμενου γραφήματος, εν ονόματι d.a.g, με τον εξής τρόπο. Δημιουργία τόσων κόμβων όσων και τα στοιχεία της

αυτοαναστρέφουσας μετάθεσης, δηλαδή για το i στοιχείο της αυτοαναστρέφουσας μετάθεσης δημιουργία του κόμβου vi. Υπολογισμός την σχέσης didomination relation για κάθε στοιχείο της αυτοαναστρέφουσας μετάθεσης από το πρώτο μέχρι και το προτελευταίο. Στην συνέχεια, για κάθε στοιχείο που ανήκει στη σχέση didomination relation, έστω j, ενός στοιχείου της αυτοαναστρέφουσας μετάθεσης, έστω i, προστίθεται μία ακμή από το i στο j. Επιπλέον στο σύνολο των κόμβων προστίθενται δύο επιπλέον κόμβοι, έστω s και t, οι οποίοι είναι αντίστοιχα ο $n' + 1$ και το 0 κόμβος μέσα στο σύνολο των κόμβων. Τέλος, σε όσους κόμβους του d.a.g έχουν μηδενικό εισερχόμενο αριθμό από ακμές προστίθεται μία επιπλέον ακμή από το κόμβο s στον κόμβο αυτόν και σε όσους κόμβους έχουν μηδενικό αριθμό από εξερχόμενες ακμές προστίθεται μία επιπλέον ακμή από αυτούς τους κόμβους στον κόμβο t του γραφήματος.

- Βήμα 2. Για κάθε κόμβο από το σύνολο των κόμβων, έστω i, εκτός του πρώτου και του τελευταίου δημιουργείται ένα σύνολο από κόμβους, έστω $P(v_i)$, με την προϋπόθεση πως στο d.a.g υπάρχει η ακμή από τους κόμβους αυτούς στον κόμβο i. Στην συνέχεια από κάθε τέτοιο σύνολο διαλέγεται ο κόμβος με την μεγαλύτερη τιμή.

- Βήμα 3. Κατασκευή ενός κατευθυνόμενου γραφήματος με τον εξής τρόπο. Για κάθε i από το n' μέχρι και το 0, προστίθεται μία ακμή από τον κόμβο v_{i+1} στον v_i. Με τον τρόπο αυτόν δημιουργείτε μία λίστα από $n' + 1$ στοιχεία με την προϋπόθεση πως η αρίθμηση ξεκινά από το 0 και όχι από το 1.

- Βήμα 4. Έστω πως η αρίθμηση των κόμβων ξεκινά από το 0, όπως προαναφέρθηκε στο προηγούμενο βήμα. Τότε, για κάθε i = 1 έως το n' προσθέτεται μία ακμή από τον κόμβο v_i στον κόμβο, που έχει την μεγαλύτερη τιμή σύμφωνα με το δεύτερο βήμα.

- Βήμα 5. Επιστροφή το γραφήματος που έχει δημιουργηθεί. Αυτό το γράφημα ονομάζεται μεταθετικό αναγώγιμο γράφημα.

3.4 Παραδείγματα πρώτου και δεύτερου αλγορίθμου

Παρακάτω, υπάρχουν δύο παραδείγματα εφαρμογής αυτής της διαδικασίας μετατροπής, τα οποία μετατρέπουν έναν φυσικό αριθμό σε ένα μεταθετικό αναγώγιμο γράφημα. Πιο συγκεκριμένα στο πρώτο παράδειγμα μετατρέπεται ο αριθμός 4, ενώ στο δεύτερο παράδειγμα ο αριθμός 52. Στα παραδείγματα αυτά, γίνονται φανερά όλα τα βήματα των πρώτων δύο αλγορίθμων, υπεύθυνων για την ορθή μετατροπή.

Παράδειγμα 1.

```
######################################################################
Edw ksekinaei h diadikasia metatrophs tou fusikou arithmou sthn s.i.p#
######################################################################

######################################################################
```

```
Plhktrologiste ton fusiko arithmo pros udatografhsh(w>=4 kai w<=127):#
#########################################################################

4

#########################################################################
Briskomaste sto sunolo N7 twn fusikwn arithmwn!!!!!!!!!!            #
#########################################################################

#########################################################################
B:                                                                      #
#########################################################################

(1, 0, 0)

#########################################################################
B_tonos:                                                                #
#########################################################################

(0, 0, 0, 1, 0, 0, 1)

#########################################################################
B_asteraki:                                                             #
#########################################################################

(1, 1, 1, 0, 1, 1, 0)

#########################################################################
X:                                                                      #
#########################################################################

(4, 7)

#########################################################################
Y:                                                                      #
#########################################################################

(1, 2, 3, 5, 6)

#########################################################################
BP:                                                                     #
#########################################################################

(4, 7, 6, 5, 3, 2, 1)

#########################################################################
Z (bohthitikos):                                                        #
#########################################################################

(4, 7, 6, 5, 3, 2, 1)

#########################################################################
Ta zeugraia:                                                            #
#########################################################################

(4, 1)
(7, 2)
(6, 3)
(5, 5)

#########################################################################
S.i.p:                                                                  #
#########################################################################

------
(4, 7, 6, 1, 5, 3, 2)
------
```

```
#########################################################################
Edw teliwnei h diadikasia metatrophs tou fusikou arithmou sthn s.i.p #
#########################################################################


#########################################################################
Edw ksekinaei h diadikasia metatrophs ths s.i.p sto r.p.g             #
#########################################################################

#########################################################################
D.a.g (opou 1000 einai o s kombos kai -1000 einai o t kombos):        #
#########################################################################

(4)---->(1)---->(3)---->NULL
(7)---->(6)---->NULL
(6)---->(1)---->(5)---->NULL
(1)---->(-1000)---->NULL
(5)---->(3)---->NULL
(3)---->(2)---->NULL
(2)---->(-1000)---->NULL
(1000)---->(4)---->(7)---->NULL
(-1000)---->NULL

#########################################################################
Final_graph (opou 1000 einai o s kombos kai -1000 einai o t kombos): #
#########################################################################

------
(1000)---->(7)---->NULL
(7)---->(6)---->(1000)---->NULL
(6)---->(5)---->(7)---->NULL
(5)---->(4)---->(6)---->NULL
(4)---->(3)---->(1000)---->NULL
(3)---->(2)---->(5)---->NULL
(2)---->(1)---->(3)---->NULL
(1)---->(-1000)---->(6)---->NULL
(-1000)---->NULL
------


#########################################################################
Edw teliwnei h diadikasia metatrophs ths s.i.p sto r.p.g              #
#########################################################################
```

Παράδειγμα 2.

```
#########################################################################
Edw ksekinaei h diadikasia metatrophs tou fusikou arithmou sthn s.i.p#
#########################################################################

#########################################################################
Plhktrologiste ton fusiko arithmo pros udatografhsh(w>=4 kai w<=127):#
#########################################################################

52

#########################################################################
Briskomaste sto sunolo N13 twn fusikwn arithmwn!!!!!!!!!!             #
#########################################################################

#########################################################################
B:                                                                     #
#########################################################################
```

```
(1, 1, 0, 1, 0, 0)

##########################################################################
B_tonos:                                                                 #
##########################################################################

(0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1)

##########################################################################
B_asteraki:                                                              #
##########################################################################

(1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0)

##########################################################################
X:                                                                       #
##########################################################################

(7, 8, 10, 13)

##########################################################################
Y:                                                                       #
##########################################################################

(1, 2, 3, 4, 5, 6, 9, 11, 12)

##########################################################################
BP:                                                                      #
##########################################################################

(7, 8, 10, 13, 12, 11, 9, 6, 5, 4, 3, 2, 1)

##########################################################################
Z (bohthitikos):                                                         #
##########################################################################

(7, 8, 10, 13, 12, 11, 9, 6, 5, 4, 3, 2, 1)

##########################################################################
Ta zeugraia:                                                             #
##########################################################################

(7, 1)
(8, 2)
(10, 3)
(13, 4)
(12, 5)
(11, 6)
(9, 9)

##########################################################################
S.i.p:                                                                   #
##########################################################################

------
(7, 8, 10, 13, 12, 11, 1, 2, 9, 3, 6, 5, 4)
------


##########################################################################
Edw teliwnei h diadikasia metatrophs tou fusikou arithmou sthn s.i.p     #
##########################################################################


##########################################################################
Edw ksekinaei h diadikasia metatrophs ths s.i.p sto r.p.g                #
##########################################################################
```

```
#########################################################################
D.a.g (opou 1000 einai o s kombos kai -1000 einai o t kombos):       #
#########################################################################

(7)---->(1)---->(2)---->(3)---->(6)---->NULL
(8)---->(1)---->(2)---->(3)---->(6)---->NULL
(10)---->(1)---->(2)---->(9)---->NULL
(13)---->(12)---->NULL
(12)---->(11)---->NULL
(11)---->(1)---->(2)---->(9)---->NULL
(1)---->(-1000)---->NULL
(2)---->(-1000)---->NULL
(9)---->(3)---->(6)---->NULL
(3)---->(-1000)---->NULL
(6)---->(5)---->NULL
(5)---->(4)---->NULL
(4)---->(-1000)---->NULL
(1000)---->(7)---->(8)---->(10)---->(13)---->NULL
(-1000)---->NULL

#########################################################################
Final_graph (opou 1000 einai o s kombos kai -1000 einai o t kombos): #
#########################################################################

------
(1000)---->(13)---->NULL
(13)---->(12)---->(1000)---->NULL
(12)---->(11)---->(13)---->NULL
(11)---->(10)---->(12)---->NULL
(10)---->(9)---->(1000)---->NULL
(9)---->(8)---->(11)---->NULL
(8)---->(7)---->(1000)---->NULL
(7)---->(6)---->(1000)---->NULL
(6)---->(5)---->(9)---->NULL
(5)---->(4)---->(6)---->NULL
(4)---->(3)---->(5)---->NULL
(3)---->(2)---->(9)---->NULL
(2)---->(1)---->(11)---->NULL
(1)---->(-1000)---->(11)---->NULL
(-1000)---->NULL
------


#########################################################################
Edw teliwnei h diadikasia metatrophs ths s.i.p sto r.p.g              #
#########################################################################
```

3.5 Τρίτος αλγόριθμος (r.p.g to s.i.p)

Με τον τρόπο αυτό έχει ολοκληρωθεί το ορθό της διαδικασίας μετατροπής ενός φυσικού αριθμού σε ένα μεταθετικό αναγώγιμο γράφημα με χρήση δύο αλγορίθμων, δίνοντας και δύο παραδείγματα για να γίνει πιο σαφής η όλη διαδικασία. Στην συνέχεια, θα ακολουθήσει το αντίστροφο, δηλαδή η μεταφορά από το τελικό γράφημα στον φυσικό αριθμό. Αυτή η διαδικασία είναι χρήσιμη για την απόδειξη γνησιότητας του εκάστοτε υδατογραφήματος και κατ επέκταση του εκάστοτε λογισμικού.

Ούτε λίγο ούτε πολύ οι παραπάνω δύο αλγόριθμοι επιτυγχάνουν την δημιουργία μίας δομής έτοιμης προς ενσωμάτωσης σε ένα λογισμικό μετατρέποντας ένας φυσικό αριθμό σε ένα μεταθετικό αναγώγιμο γράφημα αποθαρρύνοντας με τον τρόπο αυτόν την υποκλοπή του από κακόβουλους χρήστες. Με άλλα λόγια οι πρώτοι δύο αλγόριθμοι βοηθούν στην παραγωγή ενός υδατογραφήματος λογισμικού, ενώ οι άλλοι δύο που ακολουθούν βοηθούν

στην απόδειξη πως σε αυτό το μεταθετικό αναγώγιμο γράφημα (το οποίο είναι καλά κρυμμένο στο εκάστοτε λογισμικό ή απευθείας μέσα στον πηγαίο κώδικα ή μέσα στο κώδικα Assembly ή μέσα στο εκτελέσιμο) είναι κωδικοποιημένος ένας φυσικός αριθμός, με την προϋπόθεση βέβαια πως πρώτα έχει λάβει μέρος η εξαγωγή του εν λόγω υδατογραφήματος.

Η όλη διαδικασία μεταφοράς από το τελικό γράφημα (μεταθετικό αναγώγιμο γράφημα ή αλλιώς υδατογράφημα) στην αυτοαναστρέφουσα μετάθεση την χειρίζεται ο τρίτος κατά σειρά αλγόριθμος και χωρίζεται στα ακόλουθα έξι βήματα.

- Βήμα 1. Διαγραφή όλων των ακμών του τελικού γραφήματος (v_{i+1}, v_i), για κάθε $0 \leq i \leq n'$. Επίσης διαγραφή του κόμβου t από το σύνολο των κόμβων, ο οποίος είναι ο v_0 κόμβος του τελικού γραφήματος.

- Βήμα 2. Αναποδογύρισμα όλων των εναπομενόντων ακμών (με τον τρόπο αυτόν δημιουργείται ένα δέντρο, με ρίζα τον κόμβο s, ο οποίος είναι ο κόμβος $v_{n'+1}$ του τελικού γραφήματος).

- Βήμα 3. Πάνω στο δέντρο που έχει δημιουργηθεί, εκτελείται d.f.s διάσχιση ξεκινώντας από τον κόμβο s (ρίζα) του δέντρου προελαύνοντας πάντα τον κόμβο με την μικρότερη τιμή.

- Βήμα 4. Τοποθέτηση σε μία ακολουθία τους κόμβους (τις τιμές των κόμβων), σύμφωνα με την σειρά που προσπελάστηκαν στην διάσχιση που έλαβε χώρα στο προηγούμενο βήμα.

- Βήμα 5. Διαγραφή του κόμβου s από την ακολουθία αυτή.

- Βήμα 6. Επιστροφή της αυτοαναστρέφουσας μετάθεσης που έχει δημιουργηθεί.

3.6 Τέταρτος αλγόριθμος (s.i.p to w)

Στην συνέχεια, λαμβάνει χώρα ο τέταρτος και τελευταίος αλγόριθμος όσο αναφορά την διαδικασία παραγωγής ενός υδατογραφήματος. Πιο συγκεκριμένα ο αλγόριθμος αυτός μετατρέπει μία αυτοαναστρέφουσα ακολουθία σε έναν φυσικό αριθμό. Με αποτέλεσμα στο σημείο αυτό ολοκληρώνεται και το ευθύ και το αντίστροφο της διαδικασίας υδατογράφησης λογισμικού.

Η όλη διαδικασία μεταφοράς από την αυτοαναστρέφουσα ακολουθία στον αριθμό (υδατόσημα) χωρίζεται στα επτά ακόλουθα βήματα.

- Βήμα 1. Υπολογισμός των ζευγαριών από την αυτοαναστρέφουσα μετάθεση με τον εξής τρόπο. Έστω πως η αυτοαναστρέφουσα ακολουθία είναι $\pi = (\pi_1, \pi_2, \ldots, \pi_{n'})$, όπου n′

= (2* n) + 1, τότε για κάθε i = 1 έως το πάνω ακέραιο μέρος του n′ / 2, επιλέγεται το ζευγάρι ($\pi_{\pi i}$, π_i) και τοποθετείται σε έναν δισδιάστατο πίνακα με δύο στήλες, μεγέθους πάνω ακέραιο μέρος του n′ / 2.

- Βήμα 2. Αρχικοποίηση της μεταβλητής i σε 1.

- Βήμα 3. Δημιουργία της bitonic permutation με τον ακόλουθο τρόπο. Έστω ένας μονοδιάστατος πίνακας όπου θα δεχτεί την bitonic permutation μεγέθους n′. Διατρέχοντας τον πίνακα των ζευγαριών από την πρώτη γραμμή έως και την τελευταία, εισάγονται στον πίνακα της bitonic permutation τα στοιχεία κάθε γραμμής ως εξής, το στοιχείο της πρώτης στήλης εισάγεται στην θέση που υποδεικνύει το δεύτερο στοιχείο και το δεύτερο στοιχείο της στήλης εισάγεται στην θέση που υποδεικνύει το πρώτο.

- Βήμα 4. Εύρεση της αύξουσας ακολουθίας X και της φθίνουσας ακολουθίας Y με τον εξής τρόπο. Επιλογή με την σειρά από την bitonic permutation εκείνων των αριθμών όπου ο ένας είναι μεγαλύτερος από τον άλλον εισάγοντάς τα στην ακολουθία X και επιλογή πάλι με σειρά εκείνων των αριθμών (των υπόλοιπων δηλαδή) όπου ο ένας είναι μικρότερος από τον άλλον εισάγοντάς τα στην ακολουθία Y με την αντίστροφη σειρά. Δηλαδή από τον πίνακα της bitonic permutation παίρνονται όλα τα στοιχεία που αυξάνονται μονοτονικά και εισάγονται στην ακολουθία X και όλα τα στοιχεία που μειώνονται μονοτονικά και εισάγονται στον ακολουθία Y.

- Βήμα 5. Κατασκευή της ακολουθίας B* μήκους n′, τοποθετώντας μηδενικά στις θέσεις που δηλώνουν τα στοιχεία της ακολουθίας X και άσσους στις θέσεις που δηλώνουν τα στοιχεία της ακολουθίας Y.

- Βήμα 6. Κατασκευή της ακολουθίας B′ μήκους επίσης n′, η οποία είναι το ακριβώς αντίστροφο της B*.

- Βήμα 7. Από την B′, έστω (π_1, π_2, …, π_n, π_{n+1}, …, $\pi_{n'}$, $\pi_{n'+1}$) υπολογίζεται και επιστρέφεται ο φυσικός αριθμός με τον εξής τρόπο. Μετατρέποντας από δυαδικό σε φυσικό αριθμό, τον αριθμό που δημιουργείται από τα στοιχεία των θέσεων n+1 έως n′.

3.7 Παραδείγματα τρίτου και τέταρτου αλγορίθμου

Παρακάτω βρίσκονται δυο παραδείγματα εφαρμογής της αντίστροφης διαδικασίας, δηλαδή των τελευταίων δύο αλγορίθμων, οι οποίοι είναι υπεύθυνοι για την μετατροπή ενός μεταθετικού αναγώγιμου γραφήματος σε έναν φυσικό αριθμό. Η διαδικασία αυτή είναι παράλληλα και απόδειξη γνησιότητας του υδατογραφήματος λογισμικού. αξίζει να σημειωθεί πως σε αυτά τα δύο παραδείγματα δεν γίνονται έλεγχοι επιθέσεων ,καθώς αυτοί θα γίνουν

στην συνέχεια σε άλλα κεφάλαια. Τέλος για λόγους συνοχής και καλύτερης κατανόησης στα παραδείγματα αυτά έχουν χρησιμοποιηθεί τα μεταθετικά αναγώγιμα γραφήματα, τα οποία παράγονται από τους φυσικούς αριθμούς 4 και 52 τα οποία λάβανε χώρα στα πρώτα δύο παραδείγματα.

Παράδειγμα 1.

```
########################################################################
Edw ksekinaei h diadikasia metatrophs tou r.p.g sthn s.i.p            #
########################################################################

To arxeio my_file anoikse gia diabasma!!!!

   1: 1000 7 0
   2: 7 6 1000 0
   3: 6 5 7 0
   4: 5 4 6 0
   5: 4 3 1000 0
   6: 3 2 5 0
   7: 2 1 3 0
   8: 1 -1000 6 0
   9: -1000 0

O sunolikos arithmos apo grammes tou arxeiou my_file einai isos me 9

########################################################################
Menu epithesewn sto udatografhma:                                      #
########################################################################

Gia elegxo epitheshs se kombous plhktrologhste 0.....................
Gia elegxo epitheshs se akmes plhktrologhste 2.......................
Gia elegxo epitheshs sthn s.i.p plhktrologhste 3.....................
Gia elegxo epitheshs sthn bitonic permutation plhktrologhste 4........
Gia elegxo epitheshs se labels plhktrologhste 5......................
Xwris elegxo epitheshs plhktrologhste 0..............................

0

########################################################################
Final_graph (opou 1000 einai o s kombos -1000 einai o t kombos):      #
########################################################################

(1000)---->(7)---->NULL
(7)---->(6)---->(1000)---->NULL
(6)---->(5)---->(7)---->NULL
(5)---->(4)---->(6)---->NULL
(4)---->(3)---->(1000)---->NULL
(3)---->(2)---->(5)---->NULL
(2)---->(1)---->(3)---->NULL
(1)---->(-1000)---->(6)---->NULL
(-1000)---->NULL

########################################################################
Endiameso grafhma (sbhnontas tis akmes apo to vi+1 sto vi):           #
########################################################################

(1000)---->NULL
(7)---->(1000)---->NULL
(6)---->(7)---->NULL
(5)---->(6)---->NULL
(4)---->(1000)---->NULL
(3)---->(5)---->NULL
(2)---->(3)---->NULL
(1)---->(6)---->NULL
(-1000)---->NULL
```

```
########################################################################
Endiameso grafhma(anapodogurizntas tis enapomhnantes akmes (dentro)):#
########################################################################

(1000)---->NULL
(1000)---->(7)---->NULL
(7)---->(6)---->NULL
(6)---->(5)---->NULL
(1000)---->(4)---->NULL
(5)---->(3)---->NULL
(3)---->(2)---->NULL
(6)---->(1)---->NULL
(-1000)---->NULL

########################################################################
S.i.p:                                                                 #
########################################################################

------
(4, 7, 6, 1, 5, 3, 2)
------

########################################################################
Edw teliwnei h diadikasia metatrophs tou r.p.g sthn s.i.p              #
########################################################################


########################################################################
Edw ksekinaei h diadikasia metatrophs ths s.i.p ston fusiko arithmo  #
########################################################################

########################################################################
Zeugaria:                                                              #
########################################################################

(1, 4)
(2, 7)
(3, 6)
(5, 5)

########################################################################
BP:                                                                    #
########################################################################

(4,  7,  6,  5,  3,  2,  1)

########################################################################
X:                                                                     #
########################################################################

(4, 7)

########################################################################
Y:                                                                     #
########################################################################

(6, 5, 3, 2, 1)

########################################################################
B_asteraki:                                                            #
########################################################################

(1, 1, 1, 0, 1, 1, 0)

########################################################################
B_tonos:                                                               #
########################################################################
```

```
(0, 0, 0, 1, 0, 0, 1)

##########################################################################
Briskomaste sto sunolo N7 twn fusikwn arithmwn!!!!!!!!!!              #
##########################################################################

##########################################################################
To udatoshma einai:                                                     #
##########################################################################

------
(4)
------


##########################################################################
Edw teliwnei h diadikasia metatrophs ths s.i.p ston fusiko arithmo   #
##########################################################################
```

Παράδειγμα 2.

```
##########################################################################
Edw ksekinaei h diadikasia metatrophs tou r.p.g sthn s.i.p           #
##########################################################################

To arxeio my_file anoikse gia diabasma!!!!

   1: 1000 13 0
   2: 13 12 1000 0
   3: 12 11 13 0
   4: 11 10 12 0
   5: 10 9 1000 0
   6: 9 8 11 0
   7: 8 7 1000 0
   8: 7 6 1000 0
   9: 6 5 9 0
  10: 5 4 6 0
  11: 4 3 5 0
  12: 3 2 9 0
  13: 2 1 11 0
  14: 1 -1000 11 0
  15: -1000 0

O sunolikos arithmos apo grammes tou arxeiou my_file einai isos me 15

##########################################################################
Menu epithesewn sto udatografhma:                                       #
##########################################################################

Gia elegxo epitheshs se kombous pieste to 1.........................
Gia elegxo epitheshs se akmes pieste to 2...........................
Gia elegxo epitheshs sthn s.i.p pieste to 3.........................
Gia elegxo epitheshs sthn bitonic permutation pieste to 4............
Xwris elegxo epitheshs pieste to 0..................................

0

##########################################################################
Final_graph (opou 1000 einai o s kombos -1000 einai o t kombos):     #
##########################################################################

(1000)---->(13)---->NULL
(13)---->(12)---->(1000)---->NULL
(12)---->(11)---->(13)---->NULL
(11)---->(10)---->(12)---->NULL
```

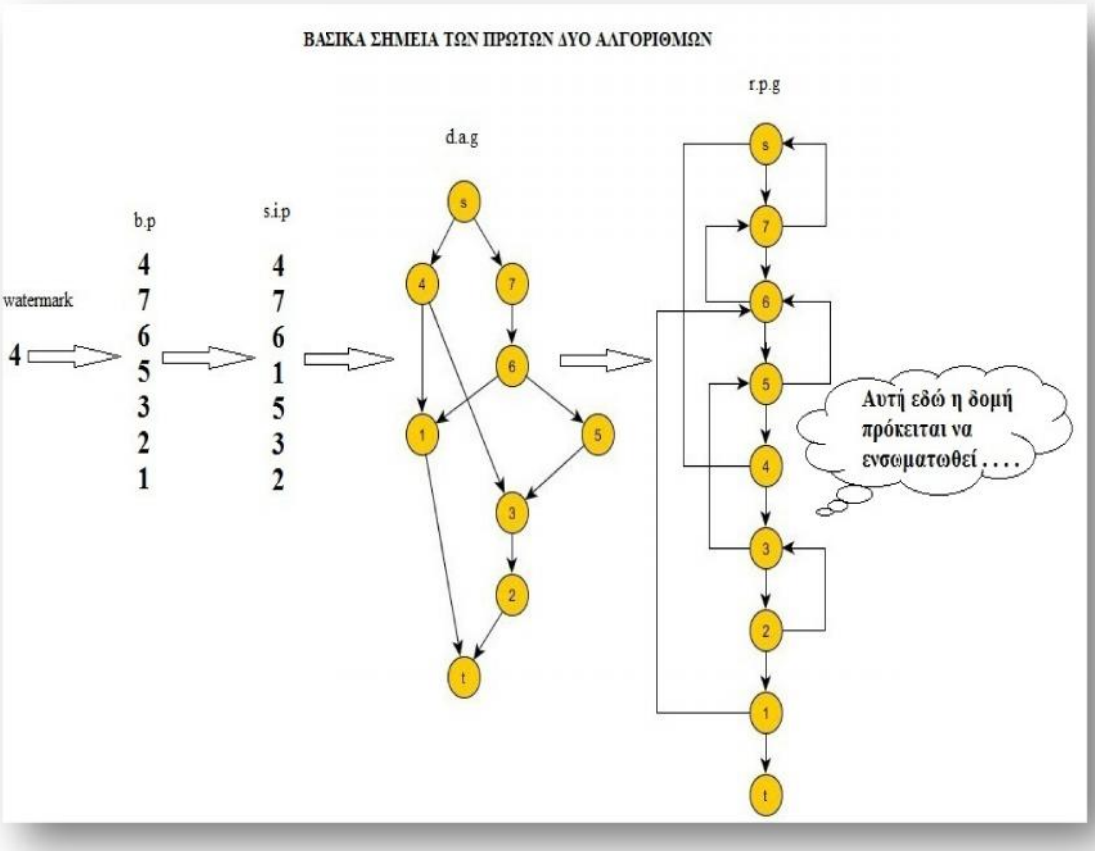
```
(10)---->(9)---->(1000)---->NULL
(9)---->(8)---->(11)---->NULL
(8)---->(7)---->(1000)---->NULL
(7)---->(6)---->(1000)---->NULL
(6)---->(5)---->(9)---->NULL
(5)---->(4)---->(6)---->NULL
(4)---->(3)---->(5)---->NULL
(3)---->(2)---->(9)---->NULL
(2)---->(1)---->(11)---->NULL
(1)---->(-1000)---->(11)---->NULL
(-1000)---->NULL

#########################################################################
Endiameso grafhma (sbhnontas tis akmes apo to vi+1 sto vi):          #
#########################################################################

(1000)---->NULL
(13)---->(1000)---->NULL
(12)---->(13)---->NULL
(11)---->(12)---->NULL
(10)---->(1000)---->NULL
(9)---->(11)---->NULL
(8)---->(1000)---->NULL
(7)---->(1000)---->NULL
(6)---->(9)---->NULL
(5)---->(6)---->NULL
(4)---->(5)---->NULL
(3)---->(9)---->NULL
(2)---->(11)---->NULL
(1)---->(11)---->NULL
(-1000)---->NULL

#########################################################################
Endiameso grafhma(anapodogurizntas tis enapomhnantes akmes (dentro)):#
#########################################################################

(1000)---->NULL
(1000)---->(13)---->NULL
(13)---->(12)---->NULL
(12)---->(11)---->NULL
(1000)---->(10)---->NULL
(11)---->(9)---->NULL
(1000)---->(8)---->NULL
(1000)---->(7)---->NULL
(9)---->(6)---->NULL
(6)---->(5)---->NULL
(5)---->(4)---->NULL
(9)---->(3)---->NULL
(11)---->(2)---->NULL
(11)---->(1)---->NULL
(-1000)---->NULL

#########################################################################
S.i.p:                                                                 #
#########################################################################

------
(7, 8, 10, 13, 12, 11, 1, 2, 9, 3, 6, 5, 4)
------

#########################################################################
Edw teliwnei h diadikasia metatrophs tou r.p.g sthn s.i.p            #
#########################################################################


#########################################################################
Edw ksekinaei h diadikasia metatrophs ths s.i.p ston fusiko arithmo  #
#########################################################################
```

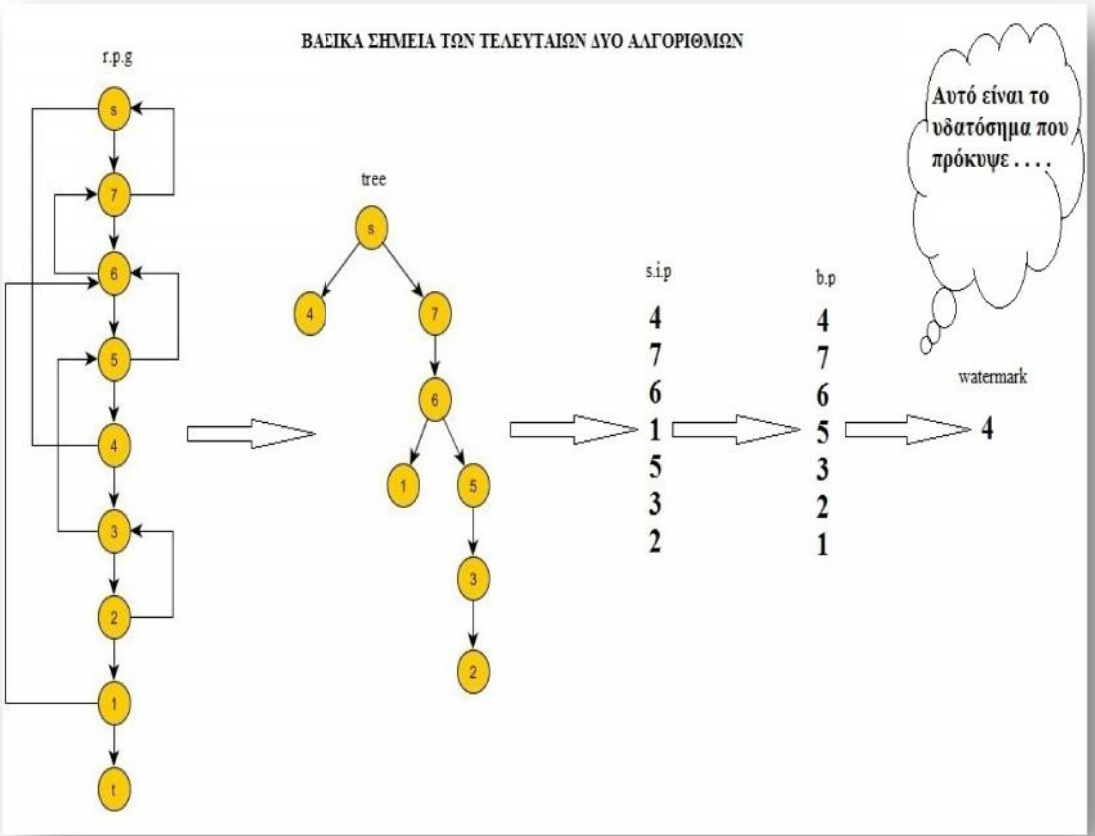
```
########################################################################
Zeugaria:                                                              #
########################################################################

(1, 7)
(2, 8)
(3, 10)
(4, 13)
(5, 12)
(6, 11)
(9, 9)

########################################################################
BP:                                                                    #
########################################################################

(7,  8,  10,  13,  12,  11,  9,  6,  5,  4,  3,  2,  1)

########################################################################
X:                                                                     #
########################################################################

(7, 8, 10, 13)

########################################################################
Y:                                                                     #
########################################################################

(12, 11, 9, 6, 5, 4, 3, 2, 1)

########################################################################
B_asteraki:                                                            #
########################################################################

(1, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0)

########################################################################
B_tonos:                                                               #
########################################################################

(0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1)

########################################################################
Briskomaste sto sunolo N13 twn fusikwn arithmwn!!!!!!!!!!              #
########################################################################

########################################################################
To udatoshma einai:                                                    #
########################################################################

------
(52)
------


########################################################################
Edw teliwnei h diadikasia metatrophs ths s.i.p ston fusiko arithmo     #
########################################################################
```

Στο σημείο αυτό, ολοκληρώνεται η πλήρης περιγραφή των τεσσάρων αλγορίθμων υδατογράφησης λογισμικού, έχοντας καλύψει το πρώτο στόχο αυτής της πτυχιακής εργασίας καθώς και το μισό κομμάτι όσο αναφορά όλη την διαδικασία υδατογράφησης (δημιουργία υδατογραφήματος και ενσωμάτωση αυτού σε λογισμικό). Προχωρώντας γίνεται η μεταφορά

στον δεύτερο κατά σειρά στόχο, ο οποίος είναι οι διάφορες επιθέσεις που μπορούν να λάβουν χώρα στο υδατογράφημα λογισμικού που έχει δημιουργηθεί (μεταθετικό αναγώγιμο γράφημα, αυτοαναστρέφουσα ακολουθία και bitonic permutation).

Πριν όμως συμβεί αυτό, για λόγους οπτικοποίησης παρακάτω βρίσκονται δύο εικόνες σχετικές με τις εσωτερικές μεταβάσεις των τεσσάρων αλγορίθμων που παρουσιάστηκαν ως αυτήν την στιγμή. Στην πρώτη εμφανίζονται με σχήματα οι τέσσερις βασικές μεταβάσεις των πρώτων δύο αλγορίθμων, οι οποίοι κωδικοποιούν των φυσικό αριθμό 4 σε ένα μεταθετικό αναγώγιμο γράφημα (υδατογράφημα), ενώ στην δεύτερη εμφανίζονται και πάλι σχηματικά οι τέσσερις βασικές μεταβάσεις των τελευταίων αλγορίθμων οι οποίοι αποκωδικοποιούν το μεταθετικό αναγώγιμο γράφημα της αμέσως προηγούμενης εικόνας στον φυσικό αριθμό 4. Με αποτέλεσμα, μέσα σε αυτές τις δύο εικόνες να απεικονίζεται η ροή των αλγορίθμων δημιουργίας υδατογραφήματος λογισμικού της παρούσας πτυχιακής εργασίας.


ΒΑΣΙΚΑ ΣΗΜΕΙΑ ΤΩΝ ΠΡΩΤΩΝ ΔΥΟ ΑΛΓΟΡΙΘΜΩΝ
watermark
4
b.p
4 7 6 5 3 2 1
s.i.p
4 7 6 1 5 3 2
d.a.g
r.p.g
Αυτή εδώ η δομή πρόκειται να ενσωματωθεί . . . .


ΒΑΣΙΚΑ ΣΗΜΕΙΑ ΤΩΝ ΤΕΛΕΥΤΑΙΩΝ ΔΥΟ ΑΛΓΟΡΙΘΜΩΝ
r.p.g
tree
s.i.p
4 7 6 1 5 3 2
b.p
4 7 6 5 3 2 1
watermark
4
Αυτό είναι το υδατόσημα που πρόκυψε . . . .

4

Μοντέλα Επιθέσεων Υδατογραφημάτων Λογισμικού

4.1 Μοντέλα επιθέσεων υδατογραφημάτων λογισμικού

Τα μοντέλα επιθέσεων σε υδατογραφήματα λογισμικού ποικίλουν και ως επί το πλείστον, είναι ανάλογα πρώτον με το υδατογράφημα το οποίο είναι κρυφά τοποθετημένο στο εκάστοτε λογισμικό και δεύτερον με τον αν ο εισβολέας γνωρίζει σε ποια θέση είναι τοποθετημένο ή όχι. Οι επιθέσεις σε ένα μεγάλο ποσοστό γίνονται στα εκτελέσιμα των προγραμμάτων καθώς η απευθείας πρόσβαση στον πηγαίο κώδικα του λογισμικού είναι ιδιαιτέρως σπάνια και δύσκολη, καθώς η πλήρης ανακατασκευή του αρχικού κώδικα από το εκτελέσιμο το οποίο είναι σε δυαδική μορφή είναι γενικά δύσκολη. Αξίζει να σημειωθεί πως ο γενικός στόχος των επιθέσεων αυτών είναι ο εντοπισμός των εκάστοτε υδατογραφημάτων λογισμικού μέσα στο εκτελέσιμο πρόγραμμα και στη συνέχεια είτε η καταστροφή είτε, η απενεργοποίηση είτε, η τροποποίηση αυτών δημιουργώντας με τον τρόπο αυτόν ένα λογισμικό μη αναγνωρίσιμο από τον νόμιμο ιδιοκτήτη του. Κλείνοντας την εισαγωγή στις επιθέσεις υδατογραφημάτων λογισμικού, αναφέρεται πως όσο μεγαλύτερη είναι η αρχική

έκταση του λογισμικού, τόσο αποτελεσματικότερη είναι και η υδατογράφησή του, διότι ο εκάστοτε κακόβουλος χρήστης είναι αναγκασμένος να ψάξει για τυχών υδατογραφήσεις σε πολύ μεγαλύτερη έκταση, πράγμα που δεν είναι καθόλου εύκολο για τον οποιοδήποτε, όσο εξοικειωμένος και αν είναι σε εκτελέσιμες μορφές αρχείων. Έτσι, ως άμεσο αποτέλεσμα του γεγονότος αυτού, μικρά σε έκταση λογισμικά δεν αποτελούν κατάλληλες ευκαιρίες προς ενσωμάτωση πολύπλοκων υδατογραφημάτων.

Η πρώτη επίθεση που θα συζητηθεί είναι η γενική επίθεση. Σε αυτού του είδους τις επιθέσεις οι εισβολείς προσπαθούν να εφαρμόσουν σημασιολογικούς μετασχηματισμούς ομοιόμορφα σε όλη την έκταση του προγράμματος, δίχως να γνωρίζουν πληροφορίες για το σημείο όπου θα μπορούσε να είχε τοποθετηθεί το υδατογράφημα. Οι γενικές επιθέσεις χωρίζονται στην συνέχεια σε άλλα δύο είδη τα οποία εξαρτώνται ανάλογα με την πρόθεση του εκάστοτε κακόβουλου χρήστη, στις γενικές στρεβλωτικές επιθέσεις και στις γενικές προσθετικές επιθέσεις.

Πιο συγκριμένα στις γενικές στρεβλωτικές επιθέσεις, ο σκοπός είναι ο εισβολέας να εφαρμόσει τροποποιήσεις οι οποίες θα συσκοτίσουν, θα βελτιστοποιήσουν και θα αναδιατάξουν των κώδικα με στόχο να στρεβλώσουν το ενσωματωμένο υδατογράφημα και να μπερδέψουν το πρόγραμμα αναγνώρισης υδατογραφήματος κάνοντας το να παράγει λανθασμένα αποτελέσματα. Από την άλλη πλευρά, στις γενικές προσθετικές επιθέσεις ο εισβολέας κάνει εισαγωγή ψεύτικων δομών δεδομένων στο εκτελέσιμο πρόγραμμα (για παράδειγμα εισάγει ένα δεύτερο υδατογράφημα), με αποτέλεσμα να δημιουργήσει ένα πρόγραμμα στο οποίο ο αναγνωριστής να μην είναι σε θέση να εντοπίσει το σωστό. Το αποτέλεσμα αυτής της επίθεσης, είναι η λανθασμένη αναγνώριση από τον αναγνωριστή υδατογραφήματος ακυρώνοντας ταυτόχρονα την υδατογράφηση του λογισμικού.

Συνεχίζοντας, έρχεται η σειρά της στοχευόμενης επίθεσης. Σε αυτό το είδος επίθεσης, όπως άλλωστε φανερώνεται και από το όνομά της, ο εισβολέας γνωρίζει σε ποιο σημείο είναι τοποθετημένο το υδατογράφημα και προσπαθεί να το αφαιρέσει από το εκτελέσιμο, χωρίς βέβαια να δημιουργήσει αλλαγές στην εκτέλεσή του λογισμικού (κάτι εντελώς ανεπιθύμητο) ή, ακόμα να το παραμορφώσει με τέτοιον τρόπο, ώστε ο αναγνωριστής να μην είναι σε θέση να το εντοπίσει. Με τον τρόπο αυτό, ο εισβολέας πετυχαίνει τον στόχο του, όπως συμβαίνει άλλωστε και στο προηγούμενο είδος επίθεσης, με την διαφορά ότι στην δεύτερη περίπτωση απατώνται βασικές πληροφορίες, όπως σε ποιο σημείο είναι το υδατογράφημα, ή ποιο είναι το υδατογράφημα πράγμα που δυσχεραίνει την εφαρμογή αυτής της επίθεσης.

Επιπλέον στα μοντέλα επιθέσεων, υπάρχει η επίθεση αναγνώρισης στην οποία ο εισβολέας εντοπίζει και απενεργοποίει το υδατογράφημα κάνοντας τις απαραίτητες λειτουργίες, δίχως όπως προαναφέρθηκε να αλλάζει την ροή εκτέλεσης του προγράμματος με τον τρόπο αυτό, το υδατογράφημα παραμένει μέσα στο εκτελέσιμο αλλά, είναι ανενεργό.

Στο σημείο αυτό όμως, είναι χρήσιμο να αναφερθούν και οι επιθέσεις που μπορούν να εφαρμοστούν στο υδατογράφημα (μεταθετικό αναγώγιμο γράφημα) αυτής της πτυχιακής εργασίας, οι οποίες όπως είναι αναμενόμενο ανήκουν στα προαναφερθέντα μοντέλα επιθέσεων. Πιο συγκεκριμένα σε αυτό το γράφημα, ένα πρώτο είδος επίθεσης που θα μπορούσε να λάβει χώρα είναι η πρόσθεση ή η αφαίρεση ακμών. Σημειώνεται πως μία τέτοια επίθεση στο εν λόγω γράφημα γίνεται πολύ εύκολα αντιληπτή από αυτόν που έχει δημιουργήσει το υδατογράφημα καθώς γνωρίζει τον ακριβή αριθμό των κόμβων του

γραφήματος. Στην συνέχεια, ένα δεύτερο είδος επίθεσης θα μπορούσε να είναι η πρόσθεση ή η αφαίρεση ακμών. Αυτή η επίθεση είτε, μπορεί να γίνει αντιληπτή είτε, όχι, διότι μία από τις κύριες ιδιότητες του τελικού γραφήματος (υδατογράφημα) είναι πως ο εξερχόμενος αριθμός από ακμές σε κάθε κόμβο είναι ακριβώς δύο εκτός από δύο συγκεκριμένους κόμβους που έχουν εξερχόμενο αριθμό ένα και μηδέν αντίστοιχα. Έτσι, αν κάποιος κακόβουλος χρήστης αλλάξει των αριθμό των ακμών χαλώντας την ιδιότητα που αναφέρθηκε προηγουμένως τότε η αλλαγή αυτή γίνεται εύκολα αντιληπτή. Άλλη μία επίθεση είναι να γίνει αλλαγή στις φορές κάποιων ακμών. Αυτή η αλλαγή μπορεί να γίνει αισθητή εφαρμόζοντας την προηγούμενη βασική ιδιότητα.

Προχωρώντας, στην επόμενη επίθεση στο τελικό γράφημα θα γνωστοποιηθεί και μία ακόμη σημαντική ιδιότητα του μεταθετικού αναγώγιμου γραφήματος (υδατογράφημα). Πιο συγκεκριμένα, αν κάποιος εισβολέας εντοπίσει το υδατογράφημα και αντί να το εξάγει εντελώς, πάει και μπερδέψει απλώς τις ετικέτες από τους κόμβους, τότε χάρης στην ιδιότητα του μοναδικού Hamiltonian μονοπατιού (ο όρος στα αγγλικά είναι unique Hamiltonian path) αυτός που δημιούργησε το υδατογράφημα είναι σε θέση να ξαναβρεί τις ετικέτες των κόμβων με τον εξής τρόπο. Πηγαίνοντας, στον πρώτο κόμβο του γραφήματος (ο οποίος έχει εισερχόμενο αριθμό από ακμές μηδέν και εξερχόμενο ένα) και σβήνοντάς τον και στην συνέχεια αναζητώντας για να συνεχίσει την διαδικασία τον κόμβο με εισερχόμενο αριθμό από ακμές να είναι ίσος με μηδέν. Με τον τρόπο αυτόν θα διατρέξει όλους τους κόμβους του γραφήματος με την σειρά και στο τέλος αφού τελειώσει με την προσπέλαση τους, τους αριθμεί από το τέλος προς την αρχή δημιουργώντας και πάλι την αρχική μορφή του υδρογραφήματός του.

Τέλος αξίζει να σημειωθούν δύο τελευταίες επιθέσεις στο τελικό γράφημα. Έστω πως οι εκάστοτε εισβολείς κατόρθωσαν και εντόπισαν το κρυμμένο υδατογράφημα μέσα στο λογισμικό και κατάλαβαν πως αυτό κωδικοποιεί έναν φυσικό αριθμό δια μέσου μίας αυτοαναστρέφουσας ακολουθίας. Τότε, μπορούν να επέμβουν στο γράφημα και να μεταφερθούν στην συνέχεια στην προαναφερθείσα ακολουθία αλλάζοντάς της, την σειρά των αριθμών χαλώντας παράλληλα την ιδιότητα της αυτοαναστρέφουσα ακολουθίας, η οποία είναι πως κάθε κύκλος μέσα στην ακολουθία είναι μήκους ένα ή δύο. Με τον τρόπο αυτό μπορούν να ξαναγυρίσουν στο τελικό γράφημα χρησιμοποιώντας την αλλαγμένη πια ακολουθία και να εισάγουν και πάλι το υδατογράφημα στο λογισμικό. Από την άλλη πλευρά όμως, κάποιοι εισβολείς θα μπορούσαν να αλλάξουν την σειρά των αριθμών στην αυτοαναστρέφουσα ακολουθία και να διατηρήσουν την ιδιότητα της αυτοαναστρέφουσας ακολουθίας, όμως αν επιχειρήσουν μία τέτοια επίθεση αυτή θα γίνει επίσης εύκολα αντιληπτή καθώς στους αλγορίθμους υπάρχει ένα βήμα μεταφοράς από την αυτοαναστρέφουσα ακολουθία σε μία άλλη ακολουθία εν ονόματι bitonic permutation και το αντίστροφο. Έτσι με τον τρόπο αυτό καλύπτεται και αυτή η επίθεση καθώς αν ο εισβολέας αλλάξει τους αριθμούς διατηρώντας την μορφή της s.i.p, τότε ως επί το πλείστον δεν θα διατηρήσει και την μορφή της bitonic permutation καθώς οι αυτοαναστρέφουσες ακολουθίες που λαμβάνουν χώρα στην εργασία αυτή έχουν μονάχα έναν μονό κύκλο και όλους τους άλλους διπλούς. Με αποτέλεσμα, μία αλλαγή να χαλάει με πολύ μεγάλη πιθανότητα την ιδιότητα αυτή. Σε περίπτωση που, ο εισβολέας κατορθώσει και διατηρήσει την ιδιότητα αυτήν, τότε η κίνησή του θα γίνει αντιληπτή σε ένα ακόμη χαμηλότερο επίπεδο, εν ονόματι δυαδική δομή, το οποίο θα αναφερθεί στο έκτο κατά σειρά κεφάλαιο όπου θα γίνει εκτενής πειραματική μελέτη επιθέσεων.

Με τον τρόπο αυτόν, το υδατογράφημα (μεταθετικό αναγώγιμο γράφημα) είναι σχεδόν πλήρως καλυμμένο από οποιαδήποτε επίθεση δεχτεί, δίνοντας έτσι την δυνατότητα στον δημιουργό του να δράσει ανάλογα σε τυχόν επιθέσεις που επρόκειτο να δεχθεί το λογισμικό του. Αξίζει να σημειωθεί πως, στην προηγούμενη πρόταση χρησιμοποιήθηκε η λέξη σχεδόν, διότι η μοναδική 'τρύπα' των υδατογραφημάτων αυτής της πτυχιακής εργασίας είναι, ο εισβολέας να αλλάξει το υδατογράφημα σε άλλο έγκυρο υδατογράφημα. Με άλλα λόγια να αλλάξει το υδατογράφημα που δημιουργήθηκε από τον φυσικό αριθμό πέντε στο υδατογράφημα που δημιουργήθηκε από τον αριθμό έξι.

Παρακάτω υπάρχει μια σατιρική εικόνα στην οποία γίνεται εμφανής πόσο εύκολα κάποιοι κακόβουλοι χρήστες εντοπίζουν και εξάγουν τα υδατογραφήματα σε ένα εκτελέσιμο πρόγραμμα:

4.2 Επιθέσεις σε κόμβους του μεταθετικού αναγώγιμου γραφήματος

Αυτού του είδους οι επιθέσεις στο μεταθετικό αναγώγιμο γράφημα, το επηρεάζουν επεμβαίνοντας στο σύνολο των κόμβων του. Πιο συγκεκριμένα, με αυτές τις επιθέσεις είτε προστίθενται κόμβοι στο σύνολο των κόμβων του γραφήματος, είτε αφαιρούνται, είτε ακόμα αλλάζουν όνομα (ετικέτα, ο αγγλικός όρος είναι label).

Γνωρίζοντας όμως, τις τρείς ιδιότητες του μεταθετικού αναγώγιμου γραφήματος αξίζει να σημειωθεί πως τέτοιου είδους επιθέσεις γίνονται εύκολα αντιληπτές από τον δημιουργό του υδατογραφήματος σε περίπτωση που ο κακόβουλος εισβολέας δεν είναι ιδιαίτερα προσεκτικός με τις ιδιότητες αυτές.

Οι ιδιότητες αυτές σχετίζονται με τους βαθμούς των εξερχόμενων ακμών σε κάθε κόμβο. Με άλλα λόγια, η πρώτη ιδιότητα αναφέρει πως ο βαθμός των εξερχόμενων ακμών του κόμβου ρίζα (ο όρος στα αγγλικά είναι root node) του γραφήματος πρέπει να είναι ένα, δηλαδή ο κόμβος ρίζα να έχει μονάχα μία εξερχόμενη ακμή. Η δεύτερη ιδιότητα αναφέρει πως ο κόμβος παιδί του γραφήματος (ο όρος στα αγγλικά είναι footer node) πρέπει να έχει εξερχόμενο βαθμό από ακμές μηδέν, δηλαδή να μην έχει καθόλου εξερχόμενες ακμές. Τέλος, η τρίτη ιδιότητα αναφέρει πως οποιοσδήποτε άλλος κόμβος του μεταθετικού αναγώγιμου γραφήματος έχει εξερχόμενο βαθμό από ακμές ακριβώς δύο.

Επιπλέον, στο σημείο αυτό είναι χρήσιμες και κάποιες περεταίρω πληροφορίες σχετικά με το συγκεκριμένο γράφημα. Πιο συγκεκριμένα, αν αυτός που έχει φτιάξει το υδατογράφημα γνωρίζει πως ο κόμβος με την μεγαλύτερη ετικέτα είναι ο vi και η τιμή του είναι x, δει έναν άλλον κόμβο με τιμή μεγαλύτερη του vi, τότε θα υποψιαστεί και θα ελέγξει το γράφημα για τυχόν επιθέσεις σε κόμβους. Επίσης, αν πάλι αυτός που έφτιαξε το υδατογράφημα γνωρίζει την ίδια πληροφορία με πριν και δει πως λείπει κάποιος κόμβος μεταξύ του 1 και του x (στο γράφημα πρέπει να εμφανίζονται όλοι οι κόμβοι με την σειρά από τον 1 μέχρι και τον x), τότε πάλι θα του κινηθούν υποψίες και θα κάνει έλεγχο για τυχόν επιθέσεις.

Με αποτέλεσμα, μόνον αν ο εισβολέας γνωρίζει τις ιδιότητες του συγκεκριμένου γραφήματος μπορεί να μετατρέψει του γράφημα διατηρώντας αυτούσιες τις ιδιότητες του. Παρακάτω παρατίθενται ένα παράδειγμα επίθεσης σε κόμβους του μεταθετικού αναγώγιμου γραφήματος.

4.3 Παράδειγμα επίθεσης σε κόμβους

Έστω πως έχει δοθεί ως είσοδο στο πρόγραμμα υδατογράφησης που μετατρέπει έναν φυσικό αριθμό από το 4 έως και το 127 σε ένα μεταθετικό αναγώγιμο γράφημα ο αριθμός 4, ο οποίος θα αποτελεί και το υδατόσημα το οποίο θα είναι κωδικοποιημένο μέσα στο υδατογράφημα. Τότε το τελικό αποτέλεσμα του προγράμματος θα είναι το εξής γράφημα:

```
(1000)----->(7)----->NULL
(7)----->(6)----->(1000)----->NULL
(6)----->(5)----->(7)----->NULL
(5)----->(4)----->(6)----->NULL
(4)----->(3)----->(1000)----->NULL
(3)----->(2)----->(5)----->NULL
(2)----->(1)----->(3)----->NULL
(1)----->(-1000)----->(6)----->NULL
(-1000)----->NULL
```

Αυτό είναι και το γράφημα που είναι κρυμμένο (υδατογράφημα λογισμικού) και στο λογισμικό που ένας κακόβουλος χρήστης έχει υποκλέψει στο συγκεκριμένο παράδειγμα. Υποθέτοντας στο σημείο αυτό, πως αυτός ο χρήστης έχει καταφέρει και έχει εξάγει το κρυμμένο υδατογράφημα χωρίς να αλλάξει στο ελάχιστο την όλη ροή του λογισμικού και επιθυμεί να κάνει επίθεση στους κόμβους το γραφήματος και πιο συγκεκριμένα προσθέτοντας έναν καινούριο κόμβο. Στο σημείο αυτό είναι αναγκαίο να σημειωθεί πως οι εκάστοτε επιθέσεις από τους κακόβουλους χρήστες προϋποθέτουν πλήρη γνώση των ιδιοτήτων που χαρακτηρίζουν το υδατογράφημα καθώς και την διαδικασία υδατογράφησης καθώς σε περίπτωση που οι ιδιότητες αυτές δεν του είναι γνωστές, τότε οι όποιες επιθέσεις θα είναι προβούν μάταιες.

Προχωρώντας γίνεται η υπόθεση πως ο εισβολέας εισάγει τον κόμβο a, του οποίο η ετικέτα είναι ανάμεσα από το 6 και το 5. Τότε το γράφημα που προκύπτει θα είναι το εξής (ο κόμβος x στο γράφημα είναι κάποιος κόμβος με ετικέτα μεγαλύτερη του a):

```
(1000)----->(7)----->NULL
(7)----->(6)----->(1000)----->NULL
(6)----->(a)----->(7)----->NULL

(a)----->(5)----->(x)----->NULL

(5)----->(4)----->(6)----->NULL
```

```
(4)---->(3)---->(1000)---->NULL
(3)---->(2)---->(5)---->NULL
(2)---->(1)---->(3)---->NULL
(1)---->(-1000)---->(6)---->NULL
(-1000)---->NULL
```

Με τον τρόπο αυτόν, ο εισβολέας έχει επιτεθεί στο υδατογράφημα, κρατώντας το στην μορφή του μεταθετικού αναγώγιμου γραφήματος και όπως είναι αναμενόμενο το επανεισάγει στο λογισμικό που έχει υποκλέψει.

Στο σημείο όμως αυτό, έρχεται το πρόγραμμα που κάνει την ακριβώς αντίθετη διαδικασία, δηλαδή μετατρέπει το μεταθετικό αναγώγιμο γράφημα στον φυσικό αριθμό από τον οποίο προήλθε. Ταυτόχρονα όμως ελέγχει και για διαφόρου είδους επιθέσεις που τυχόν να έχει δεχθεί το υδατογράφημα από εισβολείς. Οι επιθέσεις αυτές αφορούν τους κόμβους, τις ετικέτες των κόμβων και τις ακμές του μεταθετικού αναγώγιμου γραφήματος καθώς και δύο μεταθέσεις οι οποίες δεν γίνονται φανερές από την πρώτη ματιά, αυτές είναι η αυτοαναστρέφουσα μετάθεση και η γνωστή με τον αγγλικό όρο bitonic permutation.

Στο συγκεκριμένο παράδειγμα, αν γίνει έλεγχος επίθεσης στους κόμβους θα γίνει κατευθείαν αντιληπτό από τον πρόγραμμα, το οποίο και εκτελεί την αντίστροφη διαδικασία πως το υδατογράφημα έχει υποστεί επίθεση στους κόμβους του και θα τεθεί το ερώτημα (από το πρόγραμμα) αν επιθυμεί αυτός που το εκτελεί να λάβει το διορθωμένο υδατόσημα (αν είναι δυνατή βέβαια η διόρθωση) ή να το τερματίσει στο σημείο που βρέθηκε η επίθεση. Έτσι η εκτέλεση του αντίστροφου προγράμματος θα σταματήσει στο σημείο όπου θα έχει δημιουργηθεί το μεταθετικό αναγώγιμο γράφημα.

Αυτό θα συμβεί διότι αυτός που εισήγαγε το υδατογράφημα γνωρίζει πως ο η μεγαλύτερη ετικέτα κόμβου αυτού του υδατογραφήματος είναι το 7, επομένως ο συνολικός αριθμός κόμβων που θα έπρεπε να έχει το υδατογράφημα είναι 7 + 2 = 9. Όμως, το πρόγραμμα βρίσκει πως το υδατογράφημα αυτό έχει 10 κόμβους, όπου 10 != 9, με αποτέλεσμα να σταματήσει την ροή εκτέλεσης και να θέσει το ερώτημα διόρθωσής ή τερματισμού.

Πάρα ταύτα, αν ο δημιουργός και κατά συνέπεια το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία δεν γνωρίζει πιο είναι το κωδικοποιημένο υδατόσημα μέσα στο υδατογράφημα, τότε και πάλι η επίθεση θα γίνει αντιληπτή από το πρόγραμμα, αλλά στον έλεγχο για επίθεση είτε στην αυτοαναστρέφουσα μετάθεση είτε στην bitonic permutation.

Προχωρώντας στις επιθέσεις στους κόμβους του μεταθετικού αναγώγιμου γραφήματος, γίνεται η υπόθεση πως ο εισβολέας πραγματοποιεί επίθεση αφαιρώντας κάποιον από τους ήδη υπάρχοντες κόμβους. Έστω πως αφαιρεί τον κόμβο που έχει την ετικέτα 6, τότε το υδατογράφημα που θα δημιουργηθεί θα είναι το εξής:

```
(1000)---->(7)---->NULL
(7)---->(1000)---->NULL
(5)---->(4)---->NULL
(4)---->(3)---->(1000)---->NULL
(3)---->(2)---->(5)---->NULL
(2)---->(1)---->(3)---->NULL
(1)---->(-1000)---->NULL
(-1000)---->NULL
```

Με τον τρόπο αυτόν ο εισβολέας έχει επιτεθεί στους κόμβους του μεταθετικού αναγώγιμου γραφήματος, με την διαφορά πως αυτήν την φορά με αυτήν του την πράξη έχει αλλοιώσει την ιδιότητα του γραφήματος, όπου κάθε κόμβος έχει εξερχόμενο αριθμό από ακμές ακριβώς δύο, εκτός από τον s που έχει ένα και τον t που έχει μηδέν. Με άμεσο αποτέλεσμα το πρόγραμμα να σταματήσει την εκτέλεσή του στο σημείο όπου έχει δημιουργήσει το υδατογράφημα και να απαντήσει πως έχει βρεθεί επίθεση στους κόμβους στο υδατογραφήματος και η διόρθωση δεν είναι εφικτή.

Στο σημείο αυτό, γίνεται φανερό αμέσως από τις δύο προαναφερθείσες επιθέσεις που έλαβαν χώρα σε αυτό το παράδειγμα πως δεν είναι βέβαιη η επιδιόρθωση του γραφήματος

έτσι ώστε να δημιουργηθεί το σωστό μεταθετικό αναγώγιμο γράφημα στο οποίο είναι κωδικοποιημένο το σωστό υδατόσημα. Με άλλα λόγια, κάποιες επιθέσεις στο υδατογράφημα επιδέχονται διόρθωση και κάποιες άλλες όχι.

4.4 Επιθέσεις σε ακμές του μεταθετικού αναγώγιμου γραφήματος

Οι επιθέσεις σε ακμές στο μεταθετικό αναγώγιμο γράφημα το επηρεάζουν στο σύνολο των ακμών του είτε προσθέτοντας ακμές είτε αφαιρώντας. Με άλλα λόγια με αυτές τις επιθέσεις οι εισβολείς προσθέτουν ή αφαιρούν ακμές στους κόμβους του συγκεκριμένου γραφήματος (αλλάζοντας δηλαδή είτε τον βαθμό των εξερχόμενων ακμών στους κόμβους του υδατογραφήματος, είτε τον εισερχόμενο).

Όμως, τέτοιου είδους επιθέσεις γίνονται απευθείας αντιληπτές από αυτόν που έφτιαξε το υδατογράφημα, καθώς οποιαδήποτε αλλαγή και να λάβει χώρα θα χαλάσει μία από τις ιδιότητες του μεταθετικού αναγώγιμου γραφήματος. Αυτός συμβαίνει διότι, αν για παράδειγμα ο κόμβος με την μεγαλύτερη ετικέτα είναι ο vi και η τιμή του είναι 11, ο αριθμός όλων των ακμών του γραφήματος είναι σταθερός και ίσος με 2 * 11 + 1. Παρακάτω παρατίθενται ένα παράδειγμα επίθεσης σε ακμές του μεταθετικού αναγώγιμου γραφήματος.

4.5 Παράδειγμα επίθεσης σε ακμές

Έστω πως έχει δοθεί ως είσοδο στο πρόγραμμα που εκτελεί την διαδικασία μετατροπής ενός μεταθετικού αναγώγιμου γραφήματος στον φυσικό αριθμό από τον οποίον προήλθε, ο ίδιος φυσικός αριθμός που λαμβάνει χώρα στο υποκεφάλαιο με τίτλο: Παράδειγμα επίθεσης σε κόμβους. Αυτός ο αριθμός είναι ο 4 και το υδατογράφημα που δημιουργείται από αυτόν τον αριθμό είναι το εξής:

```
(1000)----->(7)----->NULL
(7)----->(6)----->(1000)----->NULL
(6)----->(5)----->(7)----->NULL
(5)----->(4)----->(6)----->NULL
(4)----->(3)----->(1000)----->NULL
(3)----->(2)----->(5)----->NULL
(2)----->(1)----->(3)----->NULL
(1)----->(-1000)----->(6)----->NULL
(-1000)----->NULL
```

Όπως γίνεται απευθείας αντιληπτό, αυτό είναι και το υδατογράφημα το οποίο είναι κρυμμένο στο λογισμικό το οποίο και έχει υποκλέψει ο εισβολέας αυτού του συγκεκριμένου παραδείγματος και επρόκειτο να δεχτεί επιθέσεις από αυτόν. Πιο συγκεκριμένα, ο εισβολέας θα διαπράξει επιθέσεις στις ακμές αυτού του γραφήματος με πρόσθεση και αφαίρεση ακμών.

Αρχικά έστω πως επιθυμεί να κάνει πρόσθεση ακμών εισάγοντας την ακμή από τον κόμβο 2 στο κόμβο 5. Με τον τρόπο αυτόν, αμέσως δημιουργείται το εξής γράφημα το οποίο ο εισβολέας θα επανεισάγει στο υποκλεμμένο λογισμικό:

```
(1000)----->(7)----->NULL
(7)----->(6)----->(1000)----->NULL
(6)----->(5)----->(7)----->NULL
(5)----->(4)----->(6)----->NULL
```

```
(4)---->(3)---->(1000)---->NULL
(3)---->(2)---->(5)---->NULL

(2)---->(1)---->(3)---->(5)---->NULL

(1)---->(-1000)---->(6)---->NULL
(-1000)---->NULL
```

Αξίζει να σημειωθεί, πως στο σημείο αυτό ο κακόβουλος χρήστης έχει διαπράξει επίθεση στις ακμές του υδατογραφήματος με πρόσθεση κάποιας ακμής, με αποτέλεσμα να αλλοιώσει την ιδιότητα των κόμβων του υδατογραφήματος, με βάση την οποία κάθε κόμβος έχει εξερχόμενο αριθμό από ακμές ίσο με δύο εκτός του s και του t, οι οποίοι έχουν ένα και μηδέν αντίστοιχα. Αυτό συμβαίνει καθώς ο κόμβος δύο έχει αριθμό από εξερχόμενες ακμές ίσο με τρία και όχι με δύο.

Επομένως το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης, κατά την οποία γίνεται η μεταφορά από το μεταθετικό αναγώγιμο γράφημα στον φυσικό αριθμό από τον οποίο προήλθε, εντοπίζει την επίθεση και θέτει το ερώτημα σε αυτόν που το εκτελεί αν επιθυμεί να διορθώσει την επίθεση ή να σταματήσει την εκτέλεση. Σε περίπτωση που επιλεχθεί η διόρθωση της επίθεσης, τότε το πρόγραμμα σβήνει τον επιπλέον κόμβο με την ετικέτα 5, ο οποίος είναι και ο περιττός κόμβος και συνεχίζει κανονικά την εκτέλεσή του δίνοντας ως τελικό αποτέλεσμα το σωστό υδατόσημα.

Βαδίζοντας στις επιθέσεις στις ακμές του μεταθετικού αναγώγιμου γραφήματος γίνεται η υπόθεση πως ο εισβολέας διαπράττει επίθεση στις ακμές αφαιρώντας ακμή. Πιο συγκεκριμένα έστω, πως γίνεται η αφαίρεση της ακμής από τον κόμβο s (είναι ο 1000 στο μεταθετικό αναγώγιμο γράφημα) στον κόμβο 7. Με τον τρόπο αυτό, ο εισβολέας δημιουργεί το εξής υδατογράφημα, το οποίο και θα τοποθετήσει ξανά μέσα στο λογισμικό το οποίο και έχει υποκλέψει:

```
(1000)---->NULL

(7)---->(6)---->(1000)---->NULL
(6)---->(5)---->(7)---->NULL
(5)---->(4)---->(6)---->NULL
(4)---->(3)---->(1000)---->NULL
(3)---->(2)---->(5)---->NULL
(2)---->(1)---->(3)---->NULL
(1)---->(-1000)---->(6)---->NULL
(-1000)---->NULL
```

Άμεσο αποτέλεσμα αυτής της επίθεσης είναι η αλλοίωση της ιδιότητας του γραφήματος, κατά την οποία ο κόμβος s έχει μία εξερχόμενη ακμή, καθώς στο συγκεκριμένο παράδειγμα ο κόμβος s έχει μηδενικό αριθμό από εξερχόμενες ακμές. Με αποτέλεσμα, το πρόγραμμα το οποίο εκτελεί την αντίστροφη διαδικασία υδατογράφησης να εντοπίσει την επίθεση που έχει λάβει χώρα και να θέσει το ερώτημα της διόρθωσης ή του τερματισμού στο σημείο αυτό. Σε περίπτωση που ο χρήστης που εκτελεί το αντίστροφο πρόγραμμα επιλέξει την διόρθωση του υδατογραφήματος προστίθεται η ακμή από τον κόμβο s στον κόμβο με την αμέσως μεγαλύτερη ακμή, ο οποίος στο συγκεκριμένο παράδειγμα είναι ο 7, ενώ στην περίπτωση που αυτός ο χρήστης επιλέξει τον τερματισμό τότε η εκτέλεση του προγράμματος σταματά στο σημείο αυτό.

4.6 Επιθέσεις σε αυτοαναστρέφουσα μετάθεση

Οι επιθέσεις που λαμβάνουν χώρα στην αυτοαναστρέφουσα μετάθεση από τους εκάστοτε εισβολείς πραγματοποιούνται μόνον στην περίπτωση όπου, αυτοί έχουν εντοπίσει και στην συνέχεια εξάγει το υδατογράφημα από το εκτελέσιμο, έχουν καταλάβει πως αυτό το υδατογράφημα είναι ένα μεταθετικό αναγώγιμο γράφημα και τέλος έχουν ανακαλύψει τον αλγόριθμο που πρέπει να εφαρμοστεί για να γίνει η μεταφορά από αυτό το γράφημα στο υδατόσημα που έχει κωδικοποιηθεί μέσα σε αυτό, μέσω βέβαια της αυτοαναστρέφουσας ακολουθίας.

Με άλλα λόγια, όπως γίνεται αντιληπτό από την προηγούμενη περιγραφή, μπορεί οι πιθανότητες να ισχύουν όλα αυτά να είναι πολύ μικρές ωστόσο υπάρχει πάντα περίπτωση κάποιος εισβολέας να καταφέρει και να ανακαλύψει όλες τις προηγούμενες γνώσεις και να φτάσει στην αυτοαναστρέφουσα μετάθεση. Όμως και σε αυτό το σημείο ισχύουν κάποιες ιδιότητες για την μετάθεση αυτήν, οι οποίες σε περίπτωση που παραβιαστούν γίνονται κατευθείαν αντιληπτές από τον δημιουργό του υδατογραφήματος και κατ' επέκταση και από το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης (οι ιδιότητες αυτές γίνονται κατανοητές στον ορισμό που έχει δοθεί για την αυτοαναστρέφουσα μετάθεση).

Πάρα ταύτα όμως, ο κακόβουλος χρήστης μπορεί να επέμβει στην ακολουθία αυτή, να την τροποποιήσει και συγχρόνως να μην χαλάσει την ιδιότητά της (κάθε κύκλος της είναι είτε μήκους ένα είτε μήκους δύο). Παρακάτω παρατίθενται ένα παράδειγμα επίθεσης σε αυτοαναστρέφουσα μετάθεση, στην οποία φτάνει κάποιος χρησιμοποιώντας το μεταθετικό αναγώγιμο γράφημα.

4.7 Παράδειγμα επίθεσης σε αυτοαναστρέφουσα μετάθεση

Στο σημείο αυτό, όπως και στα προηγούμενα παραδείγματα επιθέσεων αυτού του κεφαλαίου γίνεται η υπόθεση πως το πρόγραμμα παραγωγής υδατογραφήματος λογισμικού δέχεται ως είσοδο τον φυσικό αριθμό 4 και παράγει ως τελικό αποτέλεσμα το εξής μεταθετικό αναγώγιμο γράφημα:

```
(1000)---->(7)---->NULL
(7)---->(6)---->(1000)---->NULL
(6)---->(5)---->(7)---->NULL
(5)---->(4)---->(6)---->NULL
(4)---->(3)---->(1000)---->NULL
(3)---->(2)---->(5)---->NULL
(2)---->(1)---->(3)---->NULL
(1)---->(-1000)---->(6)---->NULL
(-1000)---->NULL
```

Συνεχίζοντας με τον ίδιο ακριβώς τρόπο, ο οποίος και ακολουθείται στα προηγούμενα παρεμφερή παραδείγματα επιθέσεων, γίνεται η υπόθεση πως το υδατογράφημα αυτό είναι κρυμμένο σε ένα λογισμικό το οποίο και έχει υποκλαπεί από κάποιον κακόβουλο χρήστη. Ο χρήστης αυτός, μετά από επεξεργασία του λογισμικού που έχει υποκλέψει είτε μέσω ειδικών προγραμμάτων βελτιστοποίησης είτε, απλά και μόνο με το μάτι του έχει εξάγει το εν λόγω υδατογράφημα και στη συνέχεια έχει κατανοήσει τον αλγόριθμο από τον οποίο προήλθε. Με τον τρόπο αυτόν, αυτός ο εισβολέας καθίσταται ικανός να διαπράξει επίθεση στην αυτοαναστρέφουσα μετάθεση που δημιουργείται από το γράφημα αυτό. Με άλλα λόγια,

διαπράττει μία επίθεση σε μία εμφωλευμένη δομή του εν λόγω υδατογραφήματος, η οποία είναι αρκετά δύσκολη να ελεγχθεί απλά με το μάτι κοιτάζοντας το τροποποιημένα πλέον υδατογράφημα.

Πάρα ταύτα, για να γίνει πράξη μία τέτοιου είδους επίθεση ο εισβολέας θα πρέπει να επέμβει στο υδατογράφημα το οποίο και έχει εξάγει. Η επέμβαση αυτή, προκειμένου να μην αλλοιώσει καμία ιδιότητα του μεταθετικού αναγώγιμου γραφήματος πρέπει αναγκαστικά να διαπραχθεί αλλάζοντας τις ακμές από τους κόμβους που δείχνουν σε κόμβους με μεγαλύτερη ετικέτα (πίσω ακμές). Δηλαδή, αν μία ακμή για παράδειγμα δείχνει από το κόμβο με την ετικέτα 2 στον κόμβο με την ετικέτα 5, 'φεύγει' και τοποθετείται σε κάποιον άλλον κόμβο με ετικέτα μεγαλύτερη του 2 και διάφορη φυσικά του 5. Αυτό συμβαίνει προκειμένου ο κόμβος με την ετικέτα 2 να συνεχίσει να έχει πίσω ακμή (ο αγγλικός όρος είναι back edge), κρατώντας ταυτόχρονα την δομή του υδατογραφήματος σωστή.

Έστω, πως στο συγκεκριμένο παράδειγμα η επίθεση του εισβολέα που λαμβάνει χώρα αλλάζει την ακμή από τον κόμβο 4 στον s και την εισάγει στον κόμβο με την ετικέτα 7. Έτσι το μεταθετικό αναγώγιμο γράφημα που δημιουργείται είναι το εξής:

```
(1000)---->(7)---->NULL
(7)---->(6)---->(1000)---->NULL
(6)---->(5)---->(7)---->NULL
(5)---->(4)---->(6)---->NULL

(4)---->(3)---->(7)---->NULL

(3)---->(2)---->(5)---->NULL
(2)---->(1)---->(3)---->NULL
(1)---->(-1000)---->(6)---->NULL
(-1000)---->NULL
```

Με τον τρόπο αυτό, ο εισβολέας έχει δημιουργήσει μία μεταλλαγμένη μορφή του υδατογραφήματος, η οποία διατηρεί την ιδιότητά του (σε κάθε κόμβο ο αριθμός τον εξερχόμενων ακμών είναι 2 εκτός από τους κόμβος s και t οι οποίοι έχουν ένα και μηδέν αντίστοιχα) και το επανεισάγει στο λογισμικό το οποίο και έχει υποκλέψει από τον νόμιμο κάτοχό του.

Εν τούτης όμως, το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης με την βοήθεια των ελέγχων που διεξάγει για τυχόν επιθέσεις εντοπίζει και πάλι με επιτυχία την επίθεση που έχει κάνει ο κακόβουλος χρήστης αυτήν την φορά σε εμφωλευμένες δομές, οι οποίες δεν είναι ορατές απευθείας από τον δημιουργό του υδατογραφήματος. Η δομή αυτή είναι η αυτοαναστρέφουσα μετάθεση και η αλλαγή που έκανε ο εισβολέας της αλλοίωσε την ιδιότητά της, όπου κάθε κύκλος της έχει μήκος είτε ένα είτε, δύο. Για λόγους οπτικοποίησης και καλύτερης κατανόησης παρατίθεται και η αυτοαναστρέφουσα μετάθεση η οποία είναι η εξής: (7, 4, 6, 1, 5, 3, 2).

4.8 Επιθέσεις σε bitonic permutation

Οι επιθέσεις που λαμβάνουν χώρα στην bitonic permutation, γίνονται υπό την προϋπόθεση πως ισχύουν οι ίδιες ακριβώς υποθέσεις όπως και στις επιθέσεις που γίνονται

από κακόβουλους χρήστες στην αυτοαναστρέφουσα μετάθεση. Με άλλα λόγια είναι εξωραϊστικά δύσκολο να συμβούν, αλλά όχι απίθανο.

Πιο συγκεκριμένα, αν κάποιος εισβολέας επιθυμεί να επιτεθεί στην bitonic permutation υπάρχει τρόπος να το κάνει χωρίς να χαλάσει την ιδιότητα της. Στο σημείο όμως αυτό, αξίζει να σημειωθεί πως αυτές οι ακολουθίες (οι οποίες είναι και το κλειδί στο να μετατραπεί ένας φυσικός αριθμός σε μία αυτοαναστρέφουσα μετάθεση και το αντίθετο) είτε μονοτονικά αυξάνονται και μετά μονοτονικά μειώνονται, είτε μονοτονικά μειώνονται και μετά μονοτονικά αυξάνονται (στην πτυχιακή αυτή εργασία χρησιμοποιούνται ακολουθίες που στην αρχή αυξάνονται μονοτονικά και στην συνέχεια μειώνονται μονοτονικά). Όμως για να συμβεί μία τέτοια επίθεση, δηλαδή να βρεθεί ένας εισβολέας με όλες τις προηγούμενες υποθέσεις και να πράξει μία τέτοια επίθεση πρέπει οποιαδήποτε αλλαγή και αν κάνει να είναι στο μεταθετικό αναγώγιμο γράφημα, καθώς αυτό είναι και το υδατογράφημα. Με αποτέλεσμα, ο μόνος εφικτός και ταυτόχρονα ακίνδυνος τρόπος για να το φέρει σε πέρας να είναι να αλλάξει τις ακμές που δείχνουν σε πίσω κόμβους (ο όρος στα αγγλικά είναι back edges), για παράδειγμα αν υπάρχει μία ακμή από τον κόμβο με την ετικέτα 10 στον κόμβο με την ετικέτα 5, να αλλάξει και να δείχνει σε έναν κόμβο με ετικέτα μικρότερη του 10 και διάφορη του 5.

Πάρα ταύτα, υπάρχει μεγάλη πιθανότητα με τέτοιου είδους αλλαγές, ο εισβολέας να χαλάσει η ιδιότητα της αυτοαναστρέφουσας μετάθεσης και η επίθεση να γίνει αντιληπτή. Παρακάτω παρατίθενται ένα παράδειγμα επίθεσης σε bitonic permutation, στην οποία φτάνει κάποιος χρησιμοποιώντας το μεταθετικό αναγώγιμο γράφημα.

4.9 Παράδειγμα επίθεσης σε bitonic permutation

Προχωρώντας, γίνεται η μετάβαση στις επιθέσεις των κακόβουλων χρηστών που λαμβάνουν χώρα στην μετάθεση γνωστή με τον αγγλικό όρο bitonic permutation. Όπως θα γίνει φανερό παρακάτω, σε αυτό το υποκεφάλαιο, προκειμένου να γίνει πράξη μία τέτοιου είδους επίθεση πρέπει αναγκαστικά ο εισβολέας να παρέμβει στο μεταθετικό αναγώγιμο γράφημα και να αλλάξει την ‘φορά’ κάποιας ακμής. Αυτό όπως έχει γίνει κατανοητό από τα προηγούμενα υποκεφάλαια, πρέπει να γίνει ακριβώς με τον ίδιο ακριβώς τρόπο όπως έγινε στο υποκεφάλαιο που σχετίζονταν με τις επιθέσεις στην αυτοαναστρέφουσα μετάθεση. Με άλλα λόγια αυτό που έχει να κάνει ο εισβολέας, προκειμένου να επιτεθεί στο υδατογράφημα στο σημείο όπου βρίσκεται η bitonic permutation είναι να αλλάξει κάποια ή ακόμα και κάποιες από τις πίσω ακμές του γραφήματος, διατηρώντας την ιδιότητά τους ως πίσω ακμές.

Αρχικά, όπως έχει γίνει η παραδοχή και στα προηγούμενα υποκεφάλαια τα οποία αφορούν τις εκάστοτε επιθέσεις, γίνεται και στο σημείο αυτό η υπόθεση πως το υδατόσημα το οποίο είναι κωδικοποιημένο στο υδατογράφημα είναι το 4. Το υδατογράφημα αυτό, είναι προφανώς το παρακάτω μεταθετικό αναγώγιμο γράφημα:

```
(1000)---->(7)---->NULL
(7)---->(6)---->(1000)---->NULL
(6)---->(5)---->(7)---->NULL
(5)---->(4)---->(6)---->NULL
(4)---->(3)---->(1000)---->NULL
(3)---->(2)---->(5)---->NULL
(2)---->(1)---->(3)---->NULL
```

```
(1)---->(-1000)---->(6)---->NULL
(-1000)---->NULL
```

Έστω, πως ο κακόβουλος χρήστης επιθυμεί για να επιτεθεί σε αυτό το υδατογράφημα στην bitonic permutation να αλλάξει την φορά της ακμής η οποία δείχνει από τον κόμβο με την ετικέτα 5 στον κόμβο με την ετικέτα 6 και να την βάλει να δείχνει στο κόμβο s, ο οποίος είναι και ο κόμβος ρίζα του γραφήματος. Με τον τρόπο αυτόν δημιουργείται το εξής γράφημα:

```
(1000)---->(7)---->NULL
(7)---->(6)---->(1000)---->NULL
(6)---->(5)---->(7)---->NULL

(5)---->(4)---->(1000)---->NULL

(4)---->(3)---->(1000)---->NULL
(3)---->(2)---->(5)---->NULL
(2)---->(1)---->(3)---->NULL
(1)---->(-1000)---->(6)---->NULL
(-1000)---->NULL
```

Επομένως, στο σημείο αυτό, αυτό που έχει να κάνει πλέον ο κακόβουλος χρήστης είναι να επανατοποθετήσει στο λογισμικό, το οποίο και έχε υποκλέψει από τον νόμιμο κάτοχό του, το μεταλλαγμένο υδατογράφημα δίχως βέβαια να μεταβάλλει την ροή εκτέλεσής του.

Όμως στο σημείο αυτό, το πρόγραμμα το οποίο εκτελεί την αντίστροφη διαδικασία υδατογράφησης η οποία αποτελεί και το εργαλείο με βάση το οποίο γίνεται η απόδειξη γνησιότητας του λογισμικού, έρχεται να ‘φωτίσει’ το ομιχλώδες σκηνικό που έχει δημιουργήσει ο εισβολέας. Αξίζει να σημειωθεί επίσης πως και η επίθεση στην bitonic permutation και η επίθεση στην αυτοαναστρέφουσα μετάθεση λαμβάνουν χώρα σε εσωτερικές δομές του υδατογραφήματος, μέσω αλλαγών που διαπράττονται όπως γίνεται άλλωστε και σε όλες τις επιθέσεις, στο μεταθετικό αναγώγιμο γράφημα το οποίο και εξάγει ο εκάστοτε εισβολέας από το υποκλεμμένο λογισμικό. Για λόγους οπτικοποίησης και καλύτερης κατανόησης παρατίθεται και η μεταλλαγμένη πλέον bitonic permutation η οποία είναι η εξής: (2, 5, 4, 6, 4, 3, 1).

4.10 Επιθέσεις σε ετικέτες κόμβων

Οι επιθέσεις στις ετικέτες των κόμβων, θα μπορούσαν να χαρακτηριστούν ως οι εξυπνότερες επιθέσεις που θα μπορούσαν να λάβουν χώρα σε ένα μεταθετικό αναγώγιμο γράφημα, συγκριτικά με όλες τις προηγούμενες. Αυτό, προκύπτει από το γεγονός, πως οι εκάστοτε εισβολείς εκτελώντας μία τέτοιου είδους επίθεση επεμβαίνουν στο υδατογράφημα, αφού προηγουμένως το έχουν εξάγει από το λογισμικό που βρίσκονταν, χωρίς να αλλάξουν στο ελάχιστο την εξωτερική δομή του. Με τον όρο εξωτερική δομή, εννοείται η δομή του μεταθετικού αναγώγιμου γραφήματος δίχως να λαμβάνονται υπόψη οι ετικέτες των κόμβων.

Με άλλα λόγια, οι κακόβουλοι χρήστες εκτελώντας μία τέτοια επίθεση, στην ουσία αυτό που κάνουν είναι να αλλάζουν τις ετικέτες των κόμβων χωρίς βέβαια να τις διαγράφουν και στην συνέχεια να ενσωματώνουν ξανά το μεταλλαγμένο πλέον υδατογράφημα στην ίδια ακριβώς θέση που είχαν εντοπίσει και το αρχικό.

Όμως, το μεταθετικό αναγώγιμο γράφημα αντιστέκεται σθεναρά και σε αυτήν την επίθεση καθιστώντας τους εκάστοτε εισβολείς ανίσχυρους να επέμβουν σε αυτό και με αυτού του είδους την επίθεση. Πιο συγκεκριμένα, αν λάβει χώρα μία τέτοια επίθεση σε ένα υδατογράφημα, τότε το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης εντοπίζοντάς την, θα ακολουθήσει ένα συγκεκριμένο μονοπάτι παράγοντας μία ακολουθία με βάσει την οποία θα κάνει τις απαραίτητες αντικαταστάσεις στους κόμβους του μεταλλαγμένου υδατογραφήματος, προκειμένου να το φέρει στην σωστή του μορφή. Η αρχή αυτού του μονοπατιού είναι ο κόμβος με εξωτερικό βαθμό από ακμές ίσο με ένα, καθώς αυτός ο κόμβος θα αποτελεί και τον κόμβο s του σωστού υδατογραφήματος. Ο συλλογισμός αυτός είναι καθόλα σωστός καθώς σε ένα σωστό μεταθετικό αναγώγιμο γράφημα ο μοναδικός κόμβος που έχει μία και μόνο ακμής είναι ο s. Έτσι, αρχίζει η δημιουργία της ακολουθίας με βάση την οποία θα δημιουργηθεί αργότερα το σωστό υδατογράφημα.

Εμβαθύνοντας λίγο στο τρόπο με τον οποίο δημιουργείτε αυτή η ακολουθία, γίνεται απολύτως φανερό πως το πρώτο της στοιχείο είναι ο κόμβος του μεταλλαγμένου γραφήματος με εξωτερικό βαθμό ακμών μονάδα και τα υπόλοιπα στοιχεία που το ακολουθούν θα προκύπτουν μεταβαίνοντας από αυτόν τον αυτόν τον κόμβο, στον πρώτο παιδί του. Εν συνεχεία, αφού δηλαδή πραγματοποιηθεί αυτή η μετάβαση διαγράφεται από όλη την δομή ο κόμβος πατέρας και επαναλαμβάνεται από τη αρχή όλη η διαδικασία έως ότου διαγραφούν όλα τα παιδία των αρχικών κόμβων. Με τον τρόπο αυτό, δημιουργείται μία ακολουθία, έχοντας ως στοιχεία όλους τους κόμβους από μία φορά τον καθέναν. Η ακολουθία αυτή λοιπόν, στη συνέχεια χρησιμοποιείται για να γίνουν οι απαραίτητες αντικαταστάσεις στο γράφημα, προκειμένου να δημιουργηθεί και πάλι το αρχικό υδατογράφημα που είχε ενσωματώσει ο προγραμματιστής. Συγκεκριμένα, κάθε ένα στοιχείο της εν λόγω ακολουθίας αντικαθιστά το στοιχείο το οποίο αντιστοιχεί στην συγκεκριμένη θέση. Η πρώτη θέση είναι του s, η δεύτερη είναι του κόμβου με την μεγαλύτερη τιμή, η τρίτη του κόμβου με την αμέσως επόμενη μεγαλύτερη τιμή και ούτε καθεξής, με την τελευταία θέση να ανήκει στον κόμβο t.

4.11 Παράδειγμα επίθεσης σε ετικέτες κόμβων

Οι επιθέσεις στις ετικέτες των κόμβων, θα μπορούσαν να χαρακτηριστούν ως οι εξυπνότερες επιθέσεις που θα μπορούσαν να λάβουν χώρα σε ένα μεταθετικό αναγώγιμο γράφημα, συγκριτικά με όλες τις προηγούμενες. Αυτό, προκύπτει από το γεγονός, πως οι εκάστοτε εισβολείς εκτελώντας μία τέτοιου είδους επίθεση επεμβαίνουν στο υδατογράφημα, αφού προηγουμένως το έχουν εξάγει από το λογισμικό που βρίσκονταν, χωρίς να αλλάξουν στο ελάχιστο την εξωτερική δομή του. Με τον όρο εξωτερική δομή, εννοείται η δομή του μεταθετικού αναγώγιμου γραφήματος δίχως να λαμβάνονται υπόψη οι ετικέτες των κόμβων.

Με άλλα λόγια, οι κακόβουλοι χρήστες εκτελώντας μία τέτοια επίθεση, στην ουσία αυτό που κάνουν είναι να αλλάζουν τις ετικέτες των κόμβων χωρίς βέβαια να τις διαγράφουν και στην συνέχεια να ενσωματώνουν ξανά το μεταλλαγμένο πλέον υδατογράφημα στην ίδια ακριβώς θέση που είχαν εντοπίσει και το αρχικό.

Όμως, το μεταθετικό αναγώγιμο γράφημα αντιστέκεται σθεναρά και σε αυτήν την επίθεση καθιστώντας τους εκάστοτε εισβολείς ανίσχυρους να επέμβουν σε αυτό και με αυτού του είδους την επίθεση. Πιο συγκεκριμένα, αν λάβει χώρα μία τέτοια επίθεση σε ένα

υδατογράφημα, τότε το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης εντοπίζοντάς την, θα ακολουθήσει ένα συγκεκριμένο μονοπάτι παράγοντας μία ακολουθία με βάσει την οποία θα κάνει τις απαραίτητες αντικαταστάσεις στους κόμβους του μεταλλαγμένου υδατογραφήματος, προκειμένου να το φέρει στην σωστή του μορφή. Η αρχή αυτού του μονοπατιού είναι ο κόμβος με εξωτερικό βαθμό από ακμές ίσο με ένα, καθώς αυτός ο κόμβος θα αποτελεί και τον κόμβο s του σωστού υδατογραφήματος. Ο συλλογισμός αυτός είναι καθόλα σωστός καθώς σε ένα σωστό μεταθετικό αναγώγιμο γράφημα ο μοναδικός κόμβος που έχει μία και μόνο ακμής είναι ο s. Έτσι, αρχίζει η δημιουργία της ακολουθίας με βάση την οποία θα δημιουργηθεί αργότερα το σωστό υδατογράφημα.

Εμβαθύνοντας λίγο στο τρόπο με τον οποίο δημιουργείτε αυτή η ακολουθία, γίνεται απολύτως φανερό πως το πρώτο της στοιχείο είναι ο κόμβος του μεταλλαγμένου γραφήματος με εξωτερικό βαθμό ακμών μονάδα και τα υπόλοιπα στοιχεία που το ακολουθούν θα προκύπτουν μεταβαίνοντας από αυτόν τον αυτόν τον κόμβο, στον πρώτο παιδί του. Εν συνεχεία, αφού δηλαδή πραγματοποιηθεί αυτή η μετάβαση διαγράφεται από όλη την δομή ο κόμβος πατέρας και επαναλαμβάνεται από τη αρχή όλη η διαδικασία έως ότου διαγραφούν όλα τα παιδία των αρχικών κόμβων. Με τον τρόπο αυτό, δημιουργείται μία ακολουθία, έχοντας ως στοιχεία όλους τους κόμβους από μία φορά τον καθέναν. Η ακολουθία αυτή λοιπόν, στη συνέχεια χρησιμοποιείται για να γίνουν οι απαραίτητες αντικαταστάσεις στο γράφημα, προκειμένου να δημιουργηθεί και πάλι το αρχικό υδατογράφημα που είχε ενσωματώσει ο προγραμματιστής. Συγκεκριμένα, κάθε ένα στοιχείο της εν λόγω ακολουθίας αντικαθιστά το στοιχείο το οποίο αντιστοιχεί στην συγκεκριμένη θέση. Η πρώτη θέση είναι του s, η δεύτερη είναι του κόμβου με την μεγαλύτερη τιμή, η τρίτη του κόμβου με την αμέσως επόμενη μεγαλύτερη τιμή και ούτε καθεξής, με την τελευταία θέση να ανήκει στον κόμβο t.

Έστω πως κάποιος εισβολέας έχει υποκλέψει ένα λογισμικό και έχει εξάγει μέσα από αυτό το εξής μεταθετικό αναγώγιμο γράφημα, το οποίο για λόγους καλύτερης κατανόησης είναι το ίδιο με τα προηγούμενα υποκεφάλαια τα οποία σχετίζονται, με παραδείγματα επιθέσεων:

```
(1000)----->(7)----->NULL
(7)----->(6)----->(1000)----->NULL
(6)----->(5)----->(7)----->NULL
(5)----->(4)----->(6)----->NULL
(4)----->(3)----->(1000)----->NULL
(3)----->(2)----->(5)----->NULL
(2)----->(1)----->(3)----->NULL
(1)----->(-1000)----->(6)----->NULL
(-1000)----->NULL
```

Επομένως, ο κακόβουλος χρήστης έχοντας φτάσει στο σημείο αυτό επιθυμεί να επιτεθεί στο εν λόγω υδατογράφημα διαπράττοντας επίθεση στις ετικέτες των κόμβων. Πιο συγκεκριμένα, τον κόμβο s τον τοποθετεί στη θέση του 1, τον κόμβο 7 στη θέση του 6, τον κόμβο 6 στη θέση του 7, τους κόμβους 5, 4, 3 και 2 δεν τους αλλάζει θέση, εν συνεχεία τον κόμβο 1 τον τοποθετεί στη θέση του s και τέλος τον κόμβο t τον αφήνει στην αρχική του θέση. Με τον τρόπο αυτόν δημιουργείται το εξής γράφημα:

```
(1000)----->(-1000)----->(7) ----->NULL
(7)----->(5)----->(6)----->NULL
(6)----->(7)----->(1)----->NULL
(5)----->(4)----->(7)----->NULL
```

```
(4)---->(3)---->(1)---->NULL
(3)---->(2)---->(5)---->NULL
(2)---->(1000)---->(3)---->NULL
(1)---->(6)---->NULL
(-1000)---->NULL
```

Επομένως, μετά τις επεμβάσεις αυτές, αυτό που έχει να κάνει πλέον ο κακόβουλος χρήστης είναι να επανατοποθετήσει στο λογισμικό και συγκεκριμένα στα ίδια ακριβώς σημεία με τα αρχικά, το αλλαγμένο υδατογράφημα χωρίς βέβαια να προκαλέσει κάποια άλλη αλλαγή στη ροή εκτέλεσης του λογισμικού.

Όμως ως από μηχανής θεός, το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης έρχεται και αποδεικνύει πως το μεταλλαγμένο υδατογράφημα έχει υποστεί επίθεση στις ετικέτες των κόμβων και αντιστοιχεί στο πρώτο υδατογράφημα που παρατίθεται σε αυτό το υποκεφάλαιο. Ο τρόπος με τον οποίον γίνεται αυτή η απόδειξη βασίζεται στις εξής ακολουθία, η οποία προκύπτει με την μέθοδο που αναφέρεται στο ακριβώς προηγούμενο υποκεφάλαιο:

```
(1), (6), (7), (5), (4), (3), (2), (s), (t)
```

Γίνεται αμέσως φανερό πως, ο κόμβος 1 βρίσκεται στη θέση s, ο κόμβος 6 στην θέση 2, ο κόμβος 7 στη θέση 6 και ούτω καθεξής με τελευταίους τους κόμβους s και t που βρίσκονται στις θέσεις 1 και t αντίστοιχα. Επομένως αντικαθίστανται ο κόμβος 1 με τον s, ο κόμβος 6 με τον 7, ο κόμβος 7 με τον 6 και ούτω καθεξής δημιουργώντας το αρχικό υδατογράφημα το οποίο έχει ενσωματωθεί αρχικά στο λογισμικό.

Παρακάτω παρατίθεται μία σατιρική εικόνα σχετική με τις επιθέσεις που πιθανών να δεχτεί το υδατογράφημα που βρίσκεται ενσωματωμένο σε κάποιο λογισμικό, από τους εκάστοτε κακόβουλους χρήστες


Επιθέσεις υδατογραφημένων λογισμικού....
ΜΠΟΥΜ
ΜΠΑΜ
ΜΠΟΥΜ
ΜΠΑΜ
Επίθεση στις ετικέτες των κόμβων....
Επίθεση στις ακμές....
Επίθεση στους κόμβους....
Επίθεση στην s.i.p....
Επίθεση στην b.p....
Βλέπεις αποτέλεσμα????
Όχι ακόμα....

5

Ενσωμάτωση Υδατογραφήματος Λογισμικού σε Λογισμικό

5.1 Μέθοδοι υδατογράφησης λογισμικού

Στο σημείο αυτό, ξεκινά η ανάλυση δύο μεθόδων με βάση των οποίων το υδατογράφημα το οποίο δημιουργείται από τους δύο πρώτους αλγορίθμους υδατογράφησης και το οποίο δεν είναι κάτι άλλο πέραν από ένα μεταθετικό αναγώγιμο γράφημα, ενσωματώνεται σε ένα λογισμικό. Πολύ γενικά, η ιδέα της πρώτης μεθόδου είναι να πραγματοποιηθεί η ενσωμάτωση του υδατογραφήματος δίχως να χρειαστεί να διασπαστεί σε επιμέρους τμήματα, σε ένα συγκεκριμένο σημείο, ενώ η ιδέα της δεύτερης μεθόδου είναι να πραγματοποιηθεί η εν λόγω υδατογράφηση χωρίζοντας το υδατογράφημα σε επιμέρους τμήματα ίσου ή ακόμα και άνισου μεγέθους το καθένα και εν συνεχεία η ενσωμάτωση αυτών σε περισσότερα του ενός σημεία.

Προτού όμως αρχίσει η ανάλυση των δύο αυτών μεθόδων, αξίζει να σημειωθεί πως τα σημεία στα οποία θα ενσωματωθεί το εκάστοτε υδατογράφημα διαφέρουν από λογισμικό σε λογισμικό και σχετίζονται ανάλογα με το είδος του λογισμικού, το διάγραμμα ελέγχου ροής του, την έκτασή του αλλά και από άλλα στοιχεία που το χαρακτηρίζουν μοναδικά.

Πιο συγκριμένα, όταν γίνεται αναφορά στο είδος του λογισμικού εννοείται η δουλειά για την οποία έχει υλοποιηθεί καθώς επίσης και τα άτομα τα οποία θα το χειρίζονται. Όταν γίνεται αναφορά στο διάγραμμα ελέγχου ροής εννοείται το διάγραμμα το οποίο προκύπτει (ζωγραφιά) ύστερα από όλες τις πιθανές εισόδους που δίνονται στο λογισμικό. Με άλλα λόγια το άτομο το οποίο θα υδατογραφήσει ένα λογισμικό πρέπει να γνωρίζει ακριβώς τι κάνει το εν λόγω πρόγραμμα και από ποιες φάσεις περνάει για κάθε είσοδο που του δίνεται. Συνεχίζοντας, όταν γίνεται αναφορά για την έκταση του λογισμικού είναι προφανές πως εννοείται πόσο μεγάλο είναι σε ποσότητα το εν λόγω πρόγραμμα. Αυτή η τελευταία πληροφορία είναι χρήσιμη καθώς ένα μικρό λογισμικό μπορεί να υποστηρίξει (να δεχτεί χωρίς να γίνουν αντιληπτά) μόνο μικρά σε έκταση υδατογραφήματα, δηλαδή υδατογραφήματα τα οποία έχουν προκύψει από μικρούς ακεραίους, ενώ ένα μεγάλο σε

έκταση λογισμικό θα μπορούσε να υποστηρίξει πολλά περισσότερα υδατογραφήματα ανεξαρτήτου μεγέθους. Επίσης σε μικρά λογισμικά τις περισσότερες φορές, δεν μπορεί να γίνει ακόμα και ο επιμερισμός του υδατογραφήματος σε μικρότερα τμήματα καθώς δεν υπάρχουν ελεύθερα μέρη για να εισαχθούν τα εκάστοτε τμήματα του χωρίς να γίνουν αντιληπτά από τους εκάστοτε εισβολείς, με αποτέλεσμα να ενσωματώνεται το μεταθετικό αναγώγιμο γράφημα σε ένα σημείο και μόνο, αυξάνοντας τις πιθανότητες για εντοπισμό από κακόβουλους χρήστες.

Δίνοντας ένα παράδειγμα για καλύτερη κατανόηση του τρόπου σκέψης που έχει κάθε άτομο (προγραμματιστής) πριν αλλά και κατά τη διάρκεια υδατογράφησης ενός λογισμικού θα γίνει πιο προσιτή η μεθοδολογία. Έστω ως πρώτη υπόθεση, πως το εν λόγω πρόγραμμα έχει υλοποιηθεί από μία εταιρία λογισμικού, καταξιωμένη στο χώρο της, για να μηχανογραφεί άλλες εταιρίες που σχετίζονται με το εμπόριο και είναι αρκετά μεγάλο σε έκταση προσφέροντας διαδραστικό γραφικό περιβάλλον. Με απλά λόγια, το συγκεκριμένο λογισμικό συνδυάζει κώδικα που υλοποιεί κάποιους μαθηματικούς τύπους και χειρισμό πινάκων αλλά και κώδικα που υλοποιεί διαδραστικά γραφικά, πράγμα που απαιτεί μία αποτελεσματική στις εκάστοτε επιθέσεις και ταυτόχρονα μη επιβαρυντική ως προς τον χρόνο εκτέλεσης υδατογράφηση.

Παρακάτω παρατίθενται μία εικόνα σατιρική, σχετικά με τον τρόπο που σκέφτονται κάποιοι προγραμματιστές προκειμένου να λύσουν ένα πρόβλημα πριν αρχίσουν να γράφουν κώδικα:

Αρχικά, ο προγραμματιστής πρέπει να έχει στο μυαλό του πως πρόκειται να υδατογραφήσει και εν συνεχεία να προστατέψει ένα πρόγραμμα μεγάλης εταιρίας λογισμικού

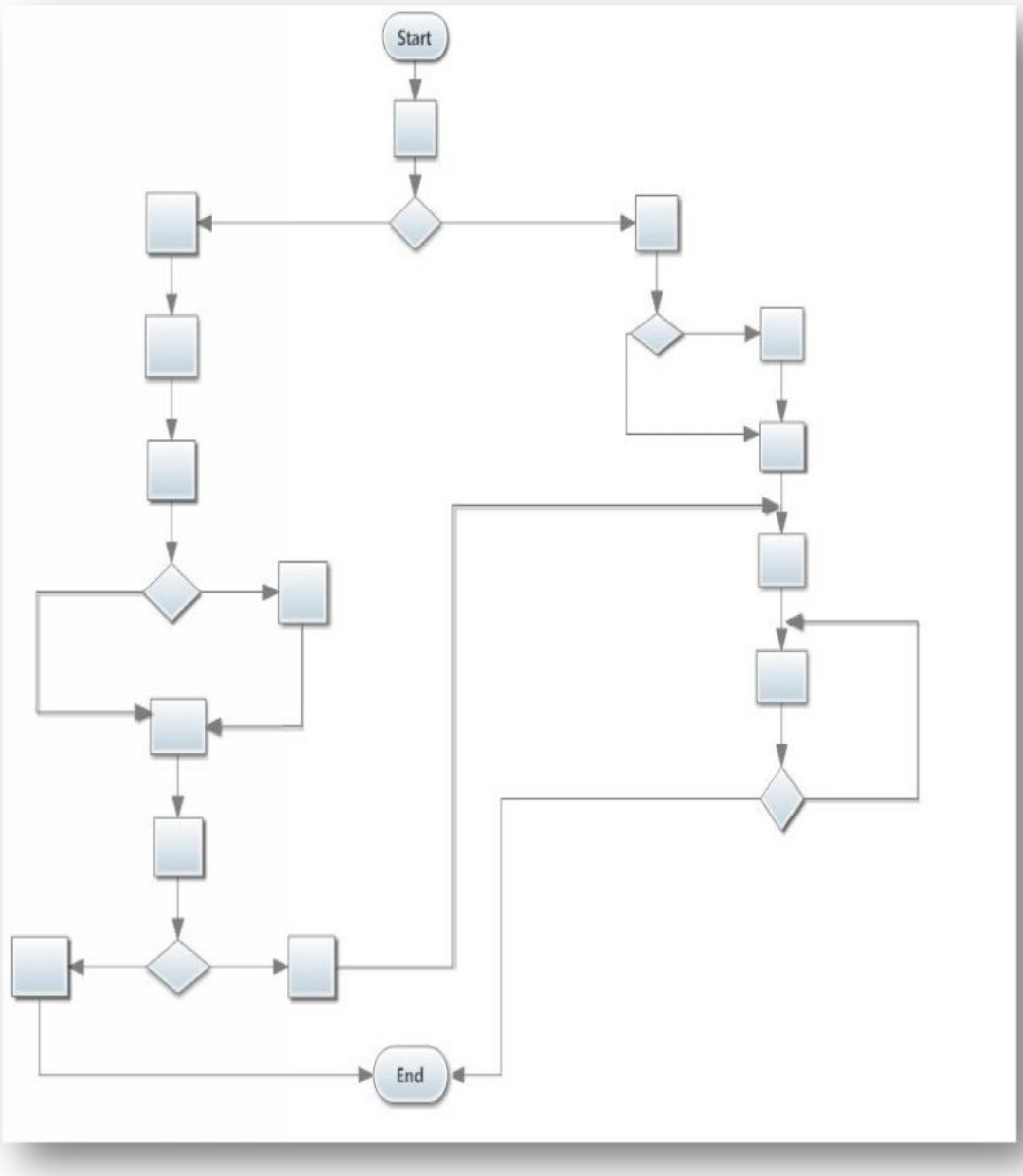
με αποτέλεσμα να πρέπει να είναι ιδιαίτερα προσεκτικός επιλέγοντας το πλέον αποτελεσματικό υδατογράφημα καθώς και την πιο ευέλικτη μέθοδο υδατογράφησης. Στην συνέχεια πρέπει να διαβάσει ενδελεχώς τον κώδικα του λογισμικού και να σχεδιάσει το διάγραμμα ελέγχου ροής του κατανοώντας παράλληλα πλήρως την εκτέλεσή του για κάθε πιθανή είσοδο. Τέλος έχοντας κάνει όλα τα παραπάνω και γνωρίζοντας την έκταση του λογισμικού ο εν λόγω προγραμματιστής θα χρησιμοποιήσει τους πρώτους δύο αλγορίθμους υδατογράφησης λογισμικού, οι οποίοι είναι υπεύθυνοι για την ορθή μετατροπή ενός φυσικού ακεραίου σε ένα μεταθετικό αναγώγιμο γράφημα, θα επιλέξει έναν μεγάλο αριθμό και θα τον δώσει ως είσοδο στο πρόγραμμα που εκτελεί τους δύο αυτούς αλγορίθμους και θα πάρει ως τελικό αποτέλεσμα ένα υδατογράφημα. Το αποτέλεσμα αυτό θα είναι μεταθετικό αναγώγιμο γράφημα με πολλούς κόμβους άρα μεγάλο σε έκταση, επομένως εφόσον η έκταση του προγράμματος του το επιτρέπει, ο προγραμματιστής θα το ‘τεμαχίσει’ σε άλλα μικρότερα διαφορετικού μήκους το καθένα τμήματα, διότι στο διάγραμμα ελέγχου ροής προφανώς δεν υπάρχουν σημεία με την ίδια ακριβώς επισκεψημότητα και θα εισάγει σε αυτά τα εκάστοτε τμήματά του εισάγοντας σε μέρη με μεγάλη επισκεψημότητα τα μικρά τμήματα και σε μέρη με μικρή επισκεψημότητα τα μικρότερα τμήματα.

Η εισαγωγή των εκάστοτε τμημάτων του υδατογραφήματος γίνεται με αυτόν τον τρόπο καθώς, τα τμήματα του αρχικού λογισμικού που έχουν μεγάλη επισκεψημότητα δεν είναι αποδεκτό να επιβαρυνθούν περεταίρω με άλλες καινούριες εντολές, διότι ο χρόνος εκτέλεσης όπως έχει ήδη αναφερθεί πρέπει να κρατηθεί στα αρχικά επίπεδα. Επομένως, γίνεται φανερό πως τα μεγάλα τμήματα του εκάστοτε υδατογραφήματος εισάγονται σε τμήματα του αρχικού λογισμικού με μικρή επισκεψημότητα προκειμένου ο συνολικός χρόνος εκτέλεσης να διατηριθεί στα αρχικά επίπεδα.

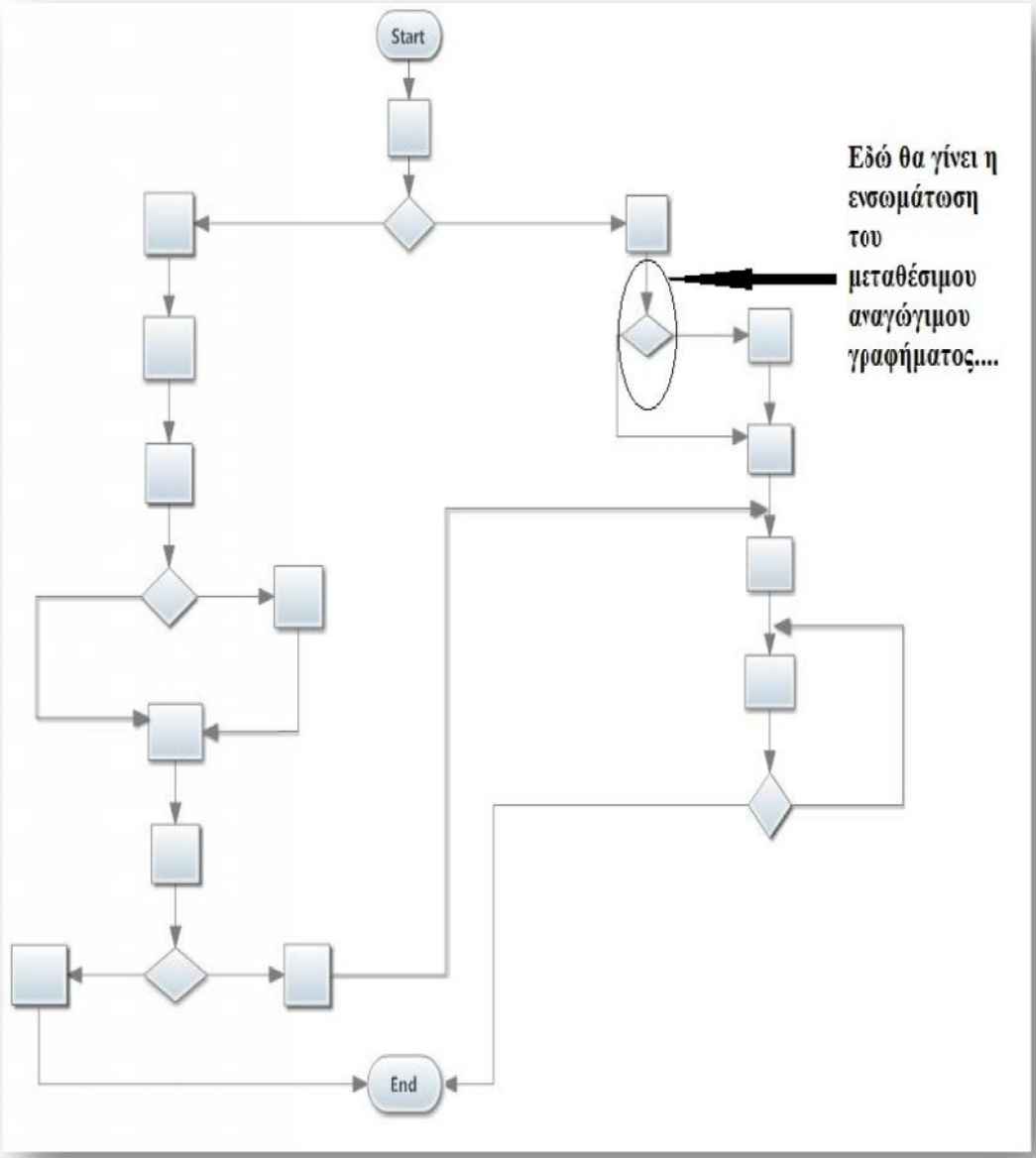
Με τον όρο επισκεψημότητα σημείων, δίνεται να εννοηθεί τα σημεία στο διάγραμμα ελέγχου ροής από τα οποία περνά πιο συχνά η εκτέλεση του προγράμματος και η οποία προκύπτει ως ο μέσος όρος επισκέψεων που δέχεται ένα σημείο ύστερα από όλες τις πιθανές εισόδους. Έτσι με τον τρόπο αυτό, ο προγραμματιστής θα προστατέψει και εν συνεχεία θα θωρακίσει με τον καλύτερο δυνατό τρόπο το λογισμικό που του δόθηκε αποτρέποντας τους εκάστοτε εισβολείς να το υποκλέψουν για δική τους χρήση. Αυτό θα συμβεί καθώς το υδατογράφημα έχει ενσωματωθεί στο λογισμικό σε διάφορα τμήματά του δημιουργώντας μία πολύ δύσκολή ’τροφή’ για τα εργαλεία βελτιστοποίησης που ενδεχομένως να χρησιμοποιήσουν οι κακόβουλοι χρήστες καθώς επίσης και για τους ίδιους σε περίπτωση που επέμβουν στο εκτελέσιμο με τον παραδοσιακό τρόπο του γυμνού οφθαλμού.

Μετά από αυτή την εισαγωγή στις μεθόδους υδατογράφησης, λαμβάνει χώρα η πρώτη μέθοδος κατά την οποία το υδατογράφημα λογισμικού δεν διασπάται και εν συνεχεία ενσωματώνεται στο πρόγραμμα σε κάποιο σημείο.

Αρχικά, από το πρόγραμμα (πηγαίος κώδικας) προκύπτει το διάγραμμα ελέγχου ροής αφού πρώτα έχει γίνει πλήρως κατανοητή η εκτέλεσή του. Έστω πως το εν λόγω διάγραμμα είναι το εξής:



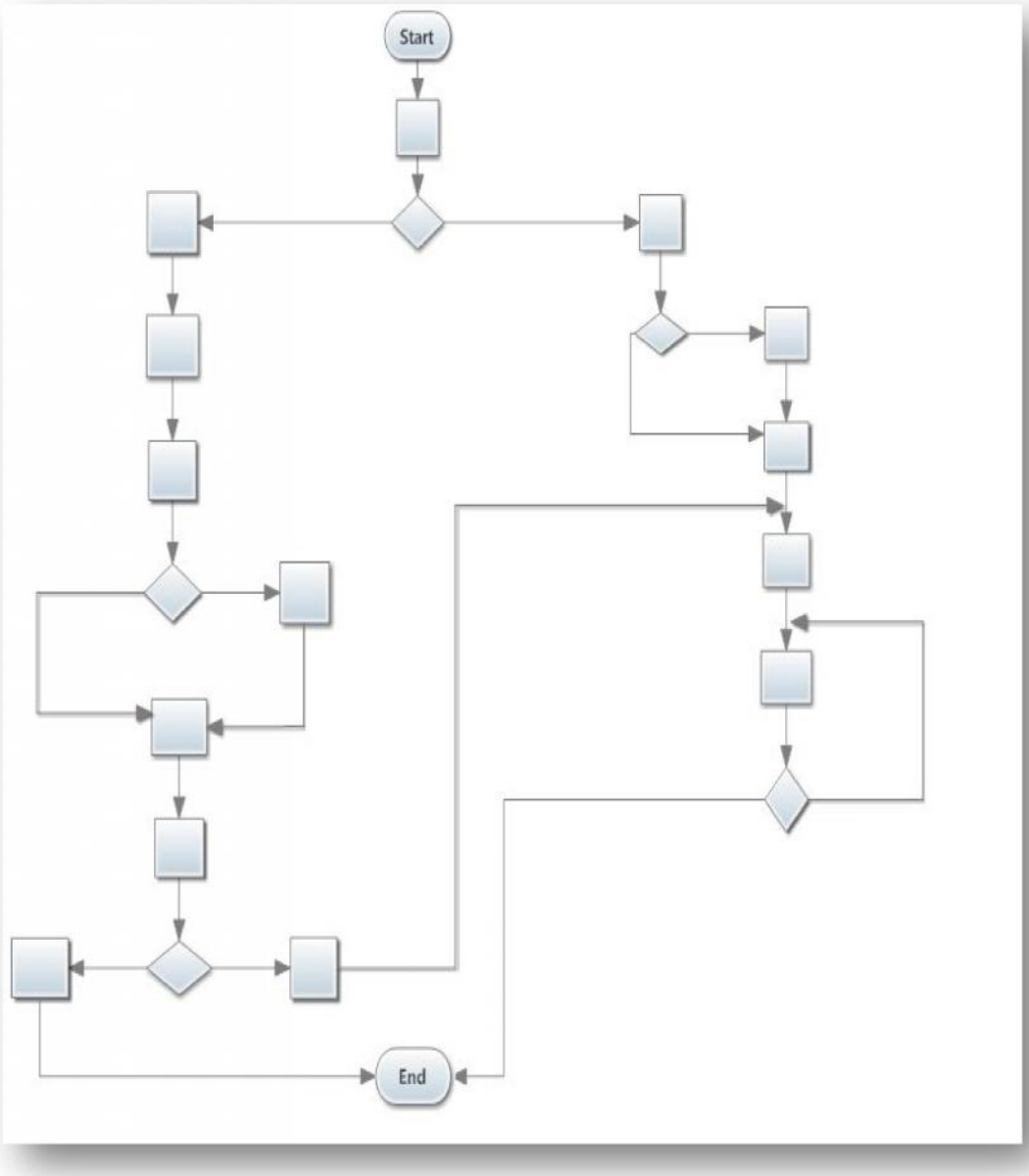
Εντοπίζεται ένα κομβικό σημείο, σημείο δηλαδή που έχει πολλές πιθανότητες να περάσει η ροή εκτέλεσης του προγράμματος από αυτό και κυκλώνεται για λόγους οπτικοποίησης και μόνο. Με τον τρόπο αυτόν προκύπτει το εξής διάγραμμα:



Στην συνέχεια, αφού έχουν προηγηθεί οι προηγούμενες ενέργειες ενσωματώνεται το υδατογράφημα, το οποίο όπως έχει προαναφερθεί είναι ένα μεταθετικό αναγώγιμο γράφημα στο εν λόγω σημείο. Δηλαδή η ακμή που δείχνει αρχικά στον κυκλωμένο κόμβο αλλάζει και δείχνει στην πρώτο κόμβο του μεταθέσιμου αναγώγιμου γραφήματος και η τελευταία ακμή του γραφήματος αυτού δείχνει τώρα στον κυκλωμένο κόμβο. Πάρα ταύτα, δεν είναι καθόλου απίθανο μετά την ενσωμάτωση του υδατογραφήματος η χρονική διάρκεια εκτέλεσης του υδατογραφημένου πλέον λογισμικού να αυξηθεί κατά τέτοιον τρόπο, έτσι ώστε να κινεί υποψίες σε κακόβουλους χρήστες ή ακόμα και να αλλοιώνει την λειτουργία για την οποία είναι υλοποιημένο, όπως για παράδειγμα στο προαναφερθέν παράδειγμα με το διαδραστικό λογισμικό. Επομένως, η ενέργεια κατά την οποία ο δημιουργός του υδατογραφήματος ψάχνει πιθανά σημεία εισαγωγής στο διάγραμμα ελέγχου ροής να πρέπει να ξαναγίνει ξανά και ξανά, επιλέγοντας όλα τα πιθανά σημεία εισαγωγής και εισάγοντας το υδατογράφημα σε αυτά προκειμένου να επιλεγεί στο τέλος μετά από όλα τα πειράματα το σημείο στο οποίο αν γίνει η

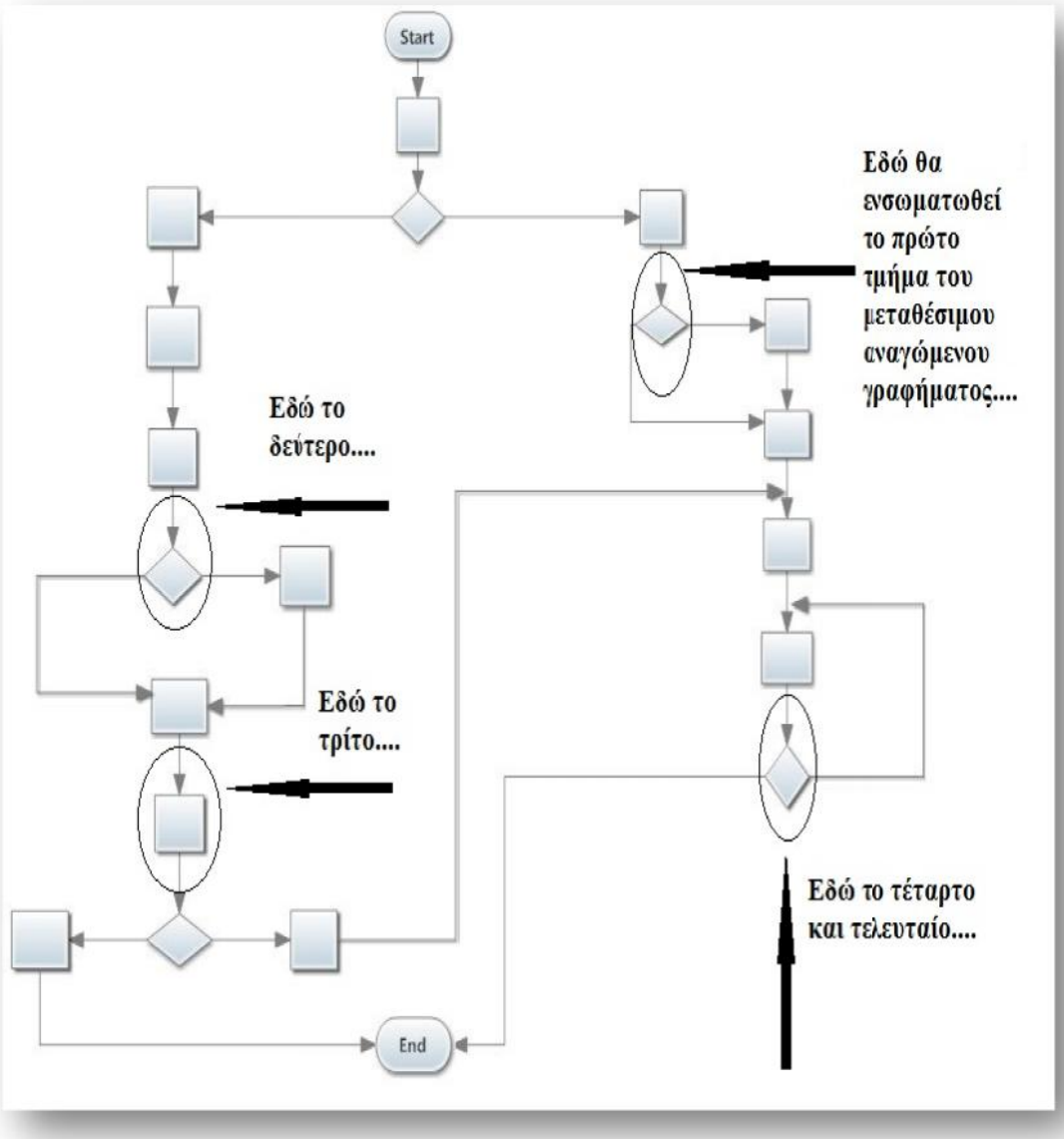
ενσωμάτωση, το υδατογραφημένο πρόγραμμα θα έχει την μικρότερη χρονική διάρκεια εκτέλεσης, η οποία βέβαια θα πρέπει να είναι όσο πιο κοντά γίνεται στην αρχική, καθώς επίσης και το σημείο στο οποίο έχει εισαχθεί το υδατογράφημα είναι όσο γίνεται πιο ακίνδυνο.

Από την άλλη πλευρά, ενδεχομένως ο προγραμματιστής και παράλληλα ο δημιουργός του υδατογραφήματος να επιθυμεί να τμήσει το μεταθετικό αναγώγιμο γράφημα σε περισσότερα του ενός τμήματα, ίσως και άνισα μεταξύ τους και στη συνέχεια, αυτά να εισάγει στο λογισμικό. Πιο συγκεκριμένα, έστω πως το διάγραμμα ελέγχου ροής είναι το αρχικό για λόγους καλύτερης κατανόησης:



Συνεχίζοντας εντοπίζονται περισσότερα του ενός κομβικά σημεία, δηλαδή σημεία που έχουν πολλές πιθανότητες να περάσει η ροή εκτέλεσης του προγράμματος από αυτά και

κυκλώνονται για λόγους οπτικοποίησης και μόνο. Με τον τρόπο αυτόν προκύπτει το εξής διάγραμμα:



Εν συνεχεία, ο δημιουργός του υδατογραφήματος πηγαίνει και εισάγει κάθε ένα τμήμα του μεταθετικού αναγώγιμου γραφήματος, με την σειρά στον πηγαίο κώδικα. Δηλαδή, στο πρώτο κυκλωμένο σημείο εισάγει το πρώτο τμήμα του υδατογραφήματος, στο δεύτερο σημείο το δεύτερο τμήμα και ούτω καθεξής. Έτσι με τον τρόπο αυτόν, δημιουργείται ένα υδατογραφημένο λογισμικό, θωρακισμένο πλέον έναντι ποικίλων επιθέσεων από πιθανούς εισβολείς. Εν τούτοις, δεν πρέπει να περιθωριοποιείται ο χρόνος εκτέλεσης του υδατογραφημένου λογισμικού, ο οποίος κινεί, στις περισσότερες των περιπτώσεων, τις υποψίες κακόβουλων χρηστών. Με άλλα λόγια ο προγραμματιστής πρέπει ύστερα από μία υδατογραφημένη έκδοση του λογισμικού του, να το εκτελεί δίνοντάς του όλες τις πιθανές εισόδους και να καταγράφει τα αποτελέσματα των χρόνων εκτέλεσης. Με αποτέλεσμα μετά

από όλες τις πιθανές εκδόσεις των υδατογραφημένων λογισμικών να διαπλεχθεί η πλέον γρήγορη και ταυτόχρονα αποτελεσματικότερη έκδοση, στα σημεία της οποίας έχουν εισαχθεί τα καταλληλότερα τμήματα του υδατογραφήματος.

Ως το σημείο αυτό, έχει γίνει ανάλυση για την υδατογράφηση λογισμικού με μεταθετικό αναγώγιμο γράφημα, το οποίο προφανώς αποτελείται από ένα σύνολο κόμβων και ένα σύνολο ακμών. Πιο συγκεκριμένα οι κόμβοι του γραφήματος αποτελούν συναρτήσεις που ενσωματώνονται στο προς υδατογράφηση λογισμικό και οι ακμές σε κλήσεις αυτών των συναρτήσεων. Το ερώτημα επομένως που τίθεται στο σημείο αυτό είναι τι ακριβώς θα περιέχουν οι συναρτήσεις αυτές προκειμένου να μην κινούν υποψίες είτε στο 'γυμνό' μάτι των εισβολέων είτε, στα προγράμματα βελτιστοποίησης που χρησιμοποιούν οι εισβολείς αυτοί προκειμένου να ξεφορτωθούν από το υποκλεμμένο λογισμικό τα περιττά δεδομένα είτε αυτά είναι υδατογραφήματα, είτε αυτά είναι άχρηστες εντολές.

Μία πρώτη προσέγγιση στο ερώτημα αυτό, είναι η εισαγωγή στις συναρτήσεις αυτές εντολές που στην ουσία είναι μη σχετικές με το υπόλοιπο πρόγραμμα. Όμως, όπως προαναφέρθηκε μία τέτοια ενέργεια θα κινούσε απευθείας τα βλέμματα των εισβολέων και το υδατογράφημα θα γίνονταν εύκολος στόχος για το προγράμματα βελτιστοποίησης κώδικα.

Μία δεύτερη και πιο υποψιασμένη προσέγγιση, είναι η εισαγωγή εντολών σχετικών με το αρχικό λογισμικό οι οποίες δεν θα επηρεάζουν την ροή εκτέλεσης του προγράμματος, δηλαδή αν υπάρχει μία σειρά εντολών στο αρχικό πρόγραμμα η οποία εκτελεί μία συγκεκριμένη λειτουργία, να χρησιμοποιηθεί η ίδια ακριβώς σειρά εντολών και στις συναρτήσεις οι οποίες αποτελούν τους κόμβους του υδατογραφήματος. Όμως και πάλι, αυτού του είδους οι ενέργειες κινούν υποψίες και γίνονται εύκολος στόχος στα προγράμματα βελτιστοποίησης κώδικα, τα όποια χρησιμοποιούν κατά κόρον οι κακόβουλοι χρήστες.

Μία τρίτη και μέχρι το σημείο η πιο υποψιασμένη προσέγγιση, είναι η εισαγωγή δεδομένων στις συναρτήσεις που δεν σχετίζονται απλά με το αρχικό λογισμικό, αλλά έχουν και άμεση επιρροή σε αυτό σε περίπτωση κατά την οποία εντοπιστούν και αφαιρεθούν από αυτό. Με άλλα λόγια, αν οι συναρτήσεις οι οποίες αποτελούν τους κόμβους του μεταθετικού αναγώγιμου γραφήματος αφαιρεθούν αλλάζει η ροή εκτέλεσης του προγράμματος και παράγονται αποτελέσματα λανθασμένα στην εκτέλεσή του.

Εμβαθύνοντας σε αυτήν την προσέγγιση αξίζει να σημειωθεί πως τα δεδομένα που επρόκειτο να προστεθούν στις συναρτήσεις που αποτελούν τους κόμβους του υδατογραφήματος μπορεί να είναι για παράδειγμα κάποιες μεταβλητές του αρχικού προγράμματος, οι οποίες καθώς εκτελείται το υδατογραφημένο πλέον λογισμικό και έρχεται και η σειρά εκτέλεσης του υδατογραφήματος, να δέχονται κάποιες τροποποιήσεις (πρόσθεση, αφαίρεση, διαίρεση, πολλαπλασιασμός και διάφορες άλλες οι οποίες εξαρτώνται από τους τύπους των μεταβλητών) καθώς γίνονται οι μεταβάσεις από κόμβο σε κόμβο του υδατογραφήματος. Με αποτέλεσμα, αν κάποιος εισβολέας εξάγει κάποιον κόμβο ή κόμβους (συναρτήσεις) το πρόγραμμα να παράγει λανθασμένα αποτελέσματα. Όμως αν ο εισβολέας εξάγει όλους τους κόμβους επιτυχώς τότε η υδατογράφηση λογισμικού παύει να ισχύει και τότε λογισμικό είναι πλέον εξολοκλήρου στα λάθος χέρια. Για αυτόν ακριβώς τον λόγο πριν γίνει η μετάβαση στον πρώτο κόμβο του υδατογραφήματος, αν βέβαια δεν έχει τμηθεί σε τμήματα, αλλιώς πριν από κάθε αρχικό κόμβο κάθε τμήματος, ο προγραμματιστής οφείλει με κάποιον τρόπο να προστατέψει το υδατογράφημά του με μία εντολή η οποία και θα σημαίνει είσοδος σε υδατογράφημα. Μία τέτοια εντολή θα μπορούσε να είναι μία σημαία έναρξης (ο

αγγλικός όρος είναι flag) η οποία θα παίρνει δυαδικές τιμές αρχικοποιώντας την με 0 με αποτέλεσμα όταν η ροής εκτέλεσης είναι μέσα στο υδατογράφημα ή αντίστοιχα σε τμήμα υδατογραφήματος να γίνεται 1 και όταν εξέρχεται από αυτήν να ελέγχεται η τιμή της, αφήνοντας την ροή εκτέλεσης να συνεχίσει μόνον αν είναι μονάδα. Με τον τρόπο αυτόν θωρακίζεται καλύτερα το λογισμικό από πιθανούς εντοπισμούς ολόκληρου του μεταθετικού αναγώγιμου γραφήματος.

Πριν όμως, συνεχίσει η ανάλυση αξίζει να σημειωθούν και άλλου είδους θωρακίσεις υδατογραφήματος. Μία από αυτές θα μπορούσε να είναι η ενέργεια όπου καθ' όλη την διάρκεια εκτέλεσης να υπάρχει ένας μετρητής (αρχικοποιημένος στο 0) που θα αυξάνεται κατά ένα κάθε φορά που γίνεται η εισαγωγή σε ένα τμήμα του μεταθετικού αναγώγιμου γραφήματος, ή ακόμα καλύτερα θα αυξομειώνεται κάθε φορά που υπάρχει εισαγωγή στα εν λόγω τμήματα με κάποιον αλγόριθμο που θα έχει επινοήσει ο προγραμματιστής για τον συγκεκριμένο σκοπό. Με αποτέλεσμα όταν τελειώσει η εκτέλεση όλου του υδατογραφήματος να λάβει χώρα ένας διεξοδικός έλεγχος, αν όντως έγιναν όλες οι εισαγωγές στα εκάστοτε τμήματα του γραφήματος. Μία δεύτερη θα μπορούσε επίσης να είναι, η συνέχιση εκτέλεσης του λογισμικού από εσκεμμένα λανθασμένο σημείο. Αυτό σημαίνει πως, στην περίπτωση όπου κάποιος κακόβουλος χρηστής εντοπίσει με επιτυχία και εξάγει του υδατογράφημα η ροή εκτέλεσης να συνεχίσει από διαφορετικό από το αναμενόμενο σημείο παράγοντας προφανώς εντελώς λανθασμένα αποτελέσματα ή ακόμα επιτυγχάνοντας την πλήρη 'πτώση' του λογισμικού. Αυτό μπορεί να συμβεί, αν ο προγραμματιστής προνοήσει και φτιάξει μία 'διακλάδωση' στην οποία αν υπάρχει το υδατογράφημα τότε η ροή περνά μέσα από αυτό και η εκτέλεση συνεχίζει ακάθεκτη ή αν δεν υπάρχει τότε η εκτέλεση μεταβαίνει σε λάθος σημείο προκαλώντας μοιραία προβλήματα στην εκτέλεση.

Προχωρώντας την ανάλυση, όπως είναι αναμενόμενο έρχεται η σειρά της εξαγωγής του υδρογραφήματος από το λογισμικό με σκοπό να κλείσει όλη την διαδικασία υδατογράφησης (δημιουργία υδατογραφήματος, επιθέσεις σε αυτό, ενσωμάτωση σε λογισμικό). Στο σημείο αυτό πρέπει να γίνει πλήρως κατανοητό, πως το υδατογράφημα όπου στην συγκεκριμένη πτυχιακή εργασία είναι ένα μεταθετικό αναγώγιμο γράφημα, ενσωματώνεται στο λογισμικό με την βοήθεια του διαγράμματος ελέγχου ροής στον πηγαίο κώδικα του λογισμικού, ενώ η εξαγωγή του λαμβάνει χώρα όχι πλέον στον κλασσικό πηγαίο κώδικα όπως η ενσωμάτωση αλλά στο εκτελέσιμο, το οποίο είναι σε μορφή κατανοητή από τον ηλεκτρονικό υπολογιστή (δυαδική μορφή). Αυτό θα συμβεί, καθώς οι εκάστοτε εισβολείς αυτό που έχουν στα χέρια τους είναι το εκτελέσιμο του προγράμματος και όχι τον αυθεντικό κώδικα και σε μία πιθανή διαμάχη, για παράδειγμα στα δικαστήρια, ο αληθινός ιδιοκτήτης θα κληθεί να εξάγει το υδατογράφημα που έχει εισάγει στο λογισμικό του από τον εκτελέσιμο κώδικα που έχει και ο εκάστοτε εισβολέας. Σημειώνεται, πως θα μπορούσε να γίνει διαφορετικά η όλη διαδικασία εισαγωγής και εξαγωγής, αλλά στη συγκεκριμένη περίπτωση αναλύεται ο τρόπος που ειπώθηκε στην παράγραφο αυτή.

Η εξαγωγή του υδατογραφήματος από το εκτελέσιμο του πλέον υδατογραφημένου προγράμματος μπορεί να ακούγεται αρχικά κάπως τρελή και επικίνδυνη, καθώς το να ξεχωρίσει κάποιος πληροφορίες ανάμεσα σε μηδενικά και άσσους φαντάζει ακατόρθωτο και ριψοκίνδυνο διότι μπορεί να αλλοιωθεί η εκτέλεση του λογισμικού εν τούτοις υπάρχουν τρόποι γρήγοροι και ακίνδυνοι για να επιτύχουν την εξαγωγή με επιτυχία.

Όπως έχει ειπωθεί στο κεφάλαιο με τον τίτλο: Εισαγωγή, κάθε λογισμικό περνά από κάποια στάδια μεταγλώττισης προτού φτάσει σε μορφή εκτελέσιμου και παράλληλα μορφή

πλήρως κατανοητή από τον υπολογιστή. Τα στάδια αυτά ξεκινούν από τον πηγαίο κώδικα (υψηλότερο επίπεδο) και καταλήγουν στο εκτελέσιμο (χαμηλότερο επίπεδο), δέχονται δηλαδή ως είσοδο έναν κώδικα γραμμένο σε κάποια γλώσσα προγραμματισμού και τον περνούν από διάφορα στάδια μέχρις ότου καταλήξουν στην εκτελέσιμη μορφή, την οποία και έχουν στα χέρια τους όποιοι χρησιμοποιούν αυτό το λογισμικό, πλην του ιδιοκτήτη που έχει στην κατοχή του και τον αρχικό και τον υδατογραφημένο κώδικα του λογισμικού. Αξίζει όμως να σημειωθεί πως μπορεί να γίνει και η ακριβώς αντίστροφη διαδικασία μεταγλώττισης, δηλαδή η μεταφορά από την εκτελέσιμη μορφή στον πηγαίο κώδικα.

Επομένως, ο προγραμματιστής αφού έχει ενσωματώσει το υδατογράφημα στο λογισμικό που του έχει δοθεί, έχει ελέγξει την αποτελεσματικότητά του σε πιθανές επιθέσεις κακόβουλων χρηστών και μεταγλωττίζοντάς το υδατογραφημένο πλέον λογισμικό έχει δημιουργήσει το εκτελέσιμο μέσα στο οποίο είναι ενσωματωμένο το υδατογράφημα. Αυτό λοιπόν που αναμένεται για την ολοκλήρωση της διαδικασίας είναι να εξαχθεί με επιτυχία το υδατογράφημα, δίχως να αλλοιώσει στο ελάχιστό το αρχικό πρόγραμμα.

Αρχικά, παίρνει το εκτελέσιμο αρχείο το οποίο περιέχει μονάχα μηδενικά και άσσους και με την βοήθεια της εντολής objdump –D, η οποία χρησιμοποιείται στο λειτουργικό σύστημα των Linux –os παίρνει ως αποτέλεσμα ένα αρχείο σε γλώσσα Assembly, το οποίο περιέχει όλη την διαδικασία μεταγλώττισης του λογισμικού σε αναλυτική μορφή. Με τον τρόπο αυτόν, γίνεται η άμεση μεταφορά σε ένα επίπεδο υψηλότερο και κάπως πιο κατανοητό από τον άνθρωπο. Επομένως, ο προγραμματιστής μπορεί με πολύ πιο εύκολο και γρήγορο τρόπο να εξάγει το υδατογράφημα από αυτό το επίπεδο και να αποδείξει την γνησιότητα του λογισμικού. Η εν λόγω εξαγωγή, μπορεί να πραγματοποιηθεί με ποικίλους τρόπους, ένας εκ των οποίων θα μπορούσε να είναι η δημιουργία ενός προγράμματος ενδεχομένως μικρού σε έκταση, το οποίο αυτό που θα έχει να υλοποιήσει είναι η αναζήτηση και εν συνεχεία η κατασκευή του υδατογραφήματος από τα δεδομένα που βρίσκει από το επίπεδο της Assembly (η φιλοσοφία του εν λόγο προγράμματος μοιάζει πολύ με έναν λεκτικό - συντακτικό αναλυτή).

Σε περίπτωση όπου η εξαγωγή του υδατογραφήματος πραγματοποιηθεί και το εξαγόμενο υδατογράφημα καταλήγει να είναι το σωστό υδατόσημα, με την βοήθεια βέβαια των δύο αλγορίθμων που είναι υπεύθυνοι για την αντίστροφη διαδικασία μεταφοράς από το μεταθετικό αναγώγιμο γράφημα δηλαδή στον φυσικό αριθμό από τον οποίον προήλθε, τότε η απόδειξη γνησιότητας στέφεται με επιτυχία, ενώ σε κάθε άλλη περίπτωση δηλαδή όπου στο εξαγόμενο υδατογράφημα παρατηρείται κάποια επίθεση, τότε η διαδικασία σταματά και αν αυτή η επίθεση δέχεται επιδιόρθωση τότε ο προγραμματιστής το αναφέρει για παράδειγμα στο δικαστήριο πως παρατηρήθηκε αυτή η επίθεση και συνεχίζει την εν λόγω διαδικασία ή αν δεν δέχεται επιδιόρθωση τότε σταματά την διαδικασία στο σημείο αυτό και αναφέρει πως το υδατογράφημα του λογισμικού του έχει δεχτεί ανεπανόρθωτη επίθεση ψάχνοντας διαφορετική γραμμή υπεράσπισης.

5.2 Παράδειγμα υδατογράφησης λογισμικού

Στο σημείο αυτό της παρούσας πτυχιακής εργασίας, δίνεται ένα σύντομο παράδειγμα υδατογράφησης λογισμικού (εισαγωγή και εξαγωγή) δίχως όμως αναλυτικά σχόλια και λεπτομέρειες καθώς η έκταση της εργασίας θα αυξάνονταν πολύ.

Αρχικά, δίνεται ο αλγόριθμος σε ψευδοκώδικα του προγράμματος που επρόκειτο να υδατογραφηθεί και είναι ο ακόλουθος:

- Βήμα 1. Αρχικοποίηση και γέμισμα του πίνακα που εμπεριέχει τα ονόματα των φοιτητών.

- Βήμα 2. Αρχικοποίηση και γέμισμα του πίνακα που εμπεριέχει τους βαθμούς του κάθε φοιτητή ανεξαρτήτου έτους.

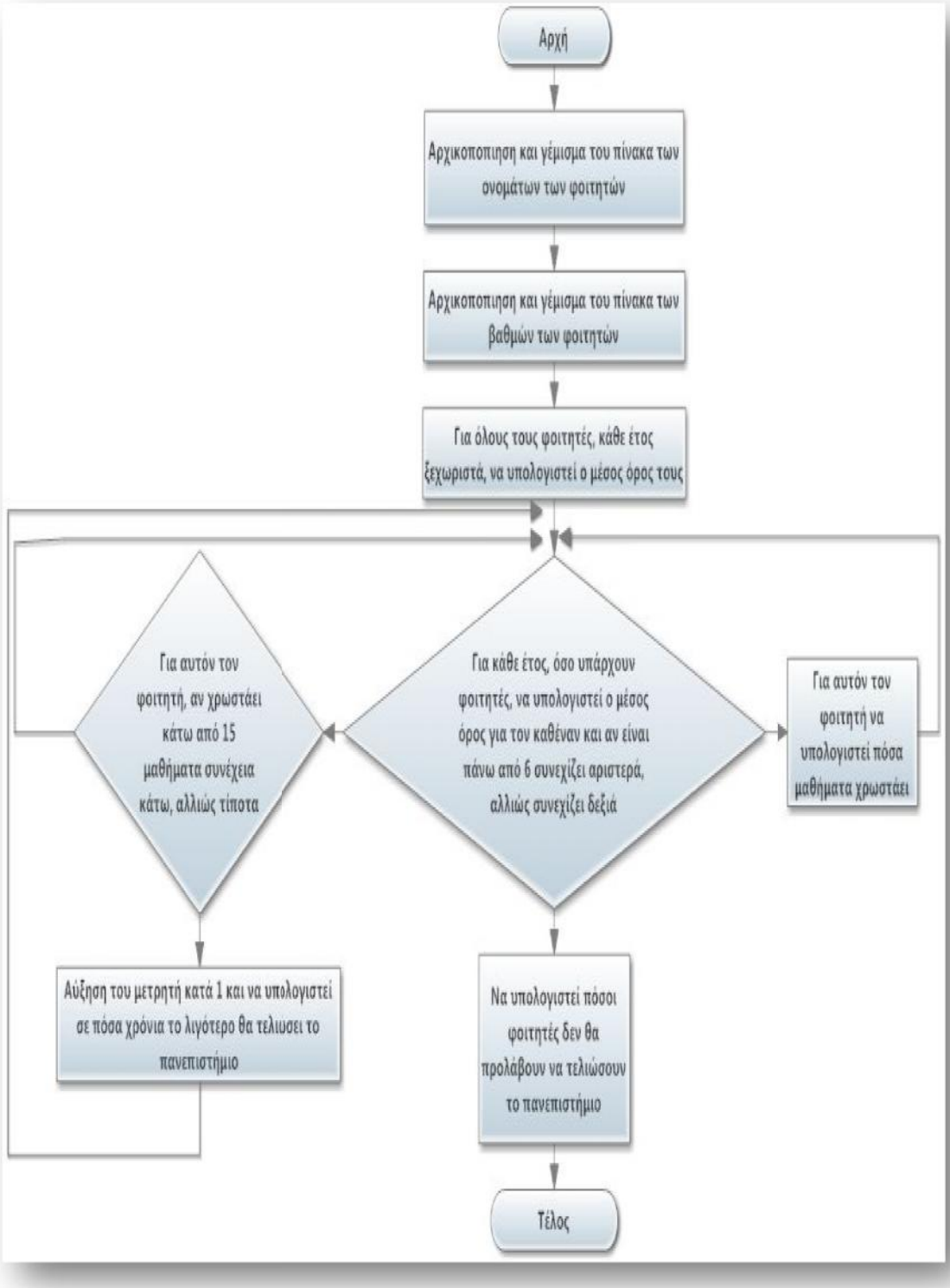
- Βήμα 3. Όσο υπάρχουν φοιτητές σε κάθε έτος να αθροιστούν οι έως τώρα βαθμοί τους ξεχωριστά και να βγει ο μέσο όρο για τον καθέναν τους.

- Βήμα 4. Όσοι από αυτούς βγάζουν μέσο όρο πάνω από 6, να υπολογιστεί για τον καθέναν τους πόσα μαθήματα χρωστάει και να βγει ο μέσος όρος χρωστούμενων μαθημάτων για τους φοιτητές ανά έτος.

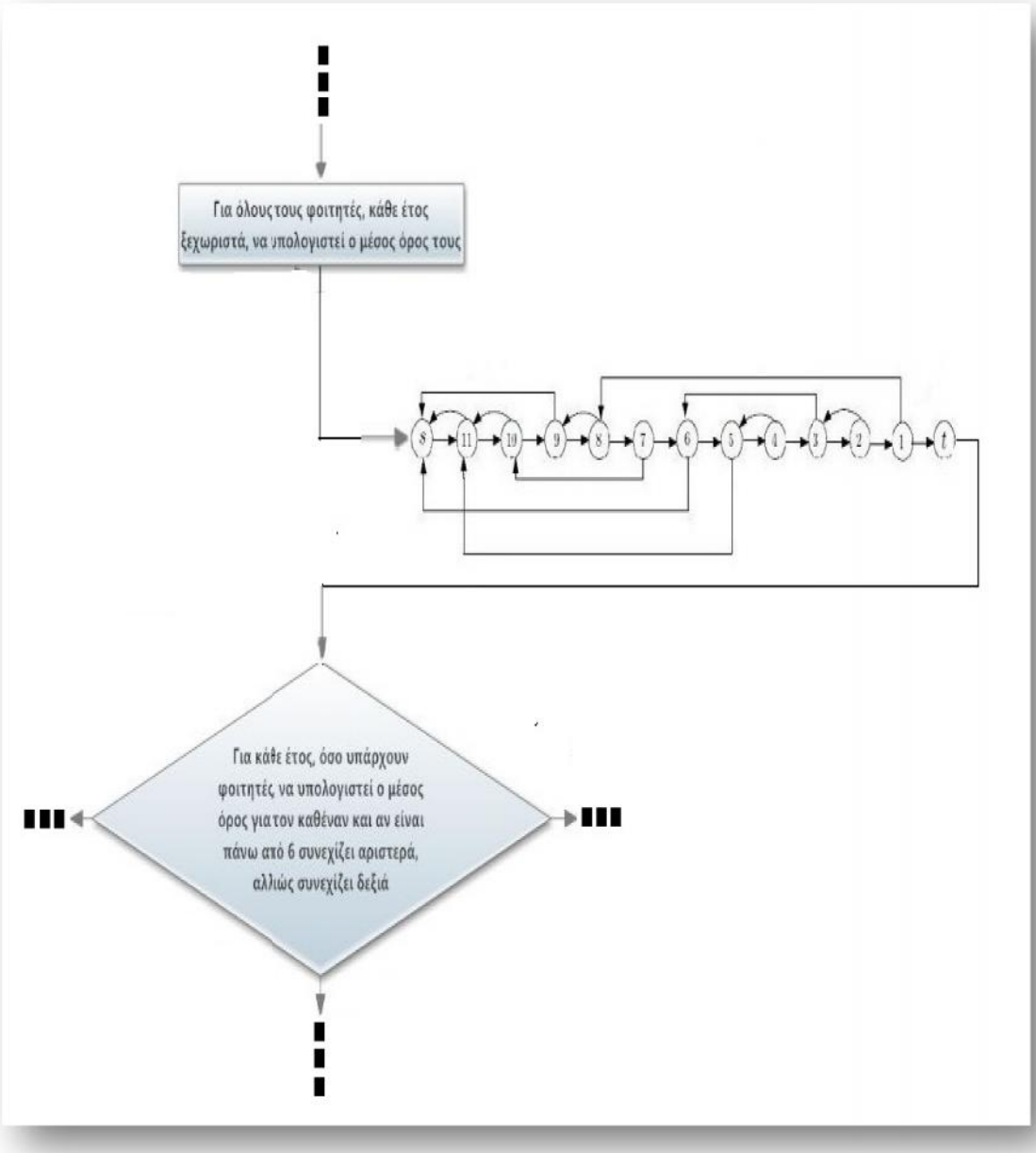
- Βήμα 5. Για τους υπόλοιπους φοιτητές, να υπολογιστεί πόσοι από αυτούς χρωστάνε κάτω από δέκα πέντε μαθήματα και να υπολογιστεί σε πόσα χρόνια το λιγότερο θα μπορέσουν να τελειώσουν τη σχολή, σύμφωνα με τον ήδη υπάρχον νόμο.

- Βήμα 6. Τέλος να υπολογιστεί το ποσοστό από τους φοιτητές που δεν θα καταφέρουν να τελειώσουν με επιτυχία τη σχολή.

Επομένως, έχοντας τον ψευδοκώδικα αυτόν, πολύ εύκολα δημιουργείται το εξής διάγραμμα ελέγχου ροής:



Με τον τρόπο αυτό, φαίνονται πλέον και οπτικά τα σημεία στα οποία θα μπορούσε να εισαχθεί το υδατογράφημα λογισμικού (είτε τμηματοποιημένο είτε όχι, αλλά στη συγκεκριμένη περίπτωση θα εισαχθεί ολόκληρο σε ένα σημείο), το οποίο δεν είναι άλλο από ένα μεταθετικό αναγώγιμο γράφημα. Με αποτέλεσμα ένα από τα ενδεχόμενα εισαγωγής ενός υδατογραφήματος με έντεκα κόμβους (το οποίο για λόγους ευχέρειας βρίσκεται και στην τελευταία εικόνα του τρίτου κατά σειρά κεφαλαίου) να είναι το ακόλουθο:



Στο σημείο όμως αυτό, είναι απαραίτητο να λάβει χώρα και η εξαγωγή του εν λόγο υδατογραφήματος, καθώς όπως ήδη έχει ειπωθεί, υδατογράφηση λογισμικού δίχως σωστή εξαγωγή δεν υφίσταται. Πιο συγκεκριμένα, έστω πως ο προηγούμενος αλγόριθμος σε ψευδοκώδικα μετατρέπεται σε μέσω της γλώσσας προγραμματισμού C, σε ένα λογισμικό και στην συνέχεια αυτό το λογισμικό μέσω των εντολών gcc και ./a.out, στο λειτουργικό σύστημα των Linux –os μεταφράζεται μετατρέπεται σε εκτελέσιμο πρόγραμμα. Με άλλα λόγια μετατρέπεται σε ένα αρχείο πλήρως κατανοητό από τον ηλεκτρονικό υπολογιστή (δυαδικό αρχείο) έχοντας μέσα του ενσωματωμένο του το υδατογράφημα, με αποτέλεσμα η εξαγωγή του από το σημείο αυτό να είναι εφικτή αλλά πολύ δύσκολη.

Για τον λόγο αυτό, χρησιμοποιείται η εντολή objdump –D προκειμένου να γίνει η μεταφορά σε ένα επίπεδο κάπως πιο κατανοητό από το προηγούμενο που βρίσκεται σε δυαδική δομή. Το νέο αυτό επίπεδο, είναι ένα στάδιο πιο ψηλά από το εκτελέσιμο αρχείο και

περιέχει κώδικα γραμμένο σε γλώσσα χαμηλού επιπέδου εν ονόματι Assembly. Αν και η αυτή η γλώσσα είναι χαμηλού επιπέδου, προσφέρεται για τον εύκολο εντοπισμό χρήσιμων πληροφοριών του αρχικού πηγαίου κώδικα. Με αποτέλεσμα, με την βοήθεια ενός λεκτικού – συντακτικού αναλυτή η εξαγωγή του κρυμμένου υδατογραφήματος να γίνεται εύκολη υπόθεση για τον δημιουργό του υδατογραφήματος.

Για λόγους οπτικοποίησης στις παρακάτω εικόνες παρατίθενται τμήματα από το αρχείο που είναι γραμμένο σε γλώσσα Assembly και εμπεριέχει το κρυμμένο υδατογράφημα (το αρχείο αυτό δημιουργείται ένα βήμα πριν δημιουργηθεί το εκτελέσιμο):

```
↓
a.out:     file format elf32-i386↓
↓
↓
Disassembly of section .interp:↓
↓
08048134 <.interp>:↓
 8048134:       2f                      das     ↓
 8048135:       6c                      insb    (%dx),%es:(%edi)↓
 8048136:       69 62 2f 6c 64 2d 6c    imul    $0x6c2d646c,0x2f(%edx),%esp↓
 804813d:       69 6e 75 78 2e 73 6f    imul    $0x6f732e78,0x75(%esi),%ebp↓
 8048144:       2e 32 00                xor     %cs:(%eax),%al↓
↓
Disassembly of section .note.ABI-tag:↓
↓
08048148 <.note.ABI-tag>:↓
 8048148:       04 00                   add     $0x0,%al↓
 804814a:       00 00                   add     %al,(%eax)↓
 804814c:       10 00                   adc     %al,(%eax)↓
 804814e:       00 00                   add     %al,(%eax)↓
 8048150:       01 00                   add     %eax,(%eax)↓
 8048152:       00 00                   add     %al,(%eax)↓
 8048154:       47                      inc     %edi↓
 8048155:       4e                      dec     %esi↓
 8048156:       55                      push    %ebp↓
 8048157:       00 00                   add     %al,(%eax)↓
 8048159:       00 00                   add     %al,(%eax)↓
 804815b:       00 02                   add     %al,(%edx)↓
 804815d:       00 00                   add     %al,(%eax)↓
 804815f:       00 06                   add     %al,(%esi)↓
 8048161:       00 00                   add     %al,(%eax)↓
 8048163:       00 0f                   add     %cl,(%edi)↓
 8048165:       00 00                   add     %al,(%eax)↓
        ...↓
```

```
↓
08048464 <main>:↓
 8048464:       55                      push   %ebp↓
 8048465:       89 e5                   mov    %esp,%ebp↓
 8048467:       83 e4 f0                and    $0xfffffff0,%esp↓
 804846a:       53                      push   %ebx↓
 804846b:       81 ec 2c 56 00 00       sub    $0x562c,%esp↓
 8048471:       c7 84 24 1c 56 00 00    movl   $0x0,0x561c(%esp)↓
 8048478:       00 00 00 00 ↓
 804847c:       c7 84 24 18 56 00 00    movl   $0x0,0x5618(%esp)↓
 8048483:       00 00 00 00 ↓
 8048487:       c7 84 24 14 56 00 00    movl   $0x0,0x5614(%esp)↓
 804848e:       00 00 00 00 ↓
 8048492:       c7 84 24 10 56 00 00    movl   $0x0,0x5610(%esp)↓
 8048499:       00 00 00 00 ↓
 804849d:       c7 84 24 08 56 00 00    movl   $0x0,0x5608(%esp)↓
 80484a4:       00 00 00 00 ↓
 80484a8:       c7 84 24 04 56 00 00    movl   $0x0,0x5604(%esp)↓
 80484af:       00 00 00 00 ↓
 80484b3:       c7 84 24 00 56 00 00    movl   $0x0,0x5600(%esp)↓
 80484ba:       00 00 00 00 ↓
 80484be:       c7 84 24 1c 56 00 00    movl   $0x0,0x561c(%esp)↓
 80484c5:       00 00 00 00 ↓
 80484c9:       e9 0d 01 00 00          jmp    80485db <main+0x177>↓
 80484ce:       c7 84 24 10 56 00 00    movl   $0x0,0x5610(%esp)↓
 80484d5:       00 00 00 00 ↓
```

```
↓
08048d6c <lala_s>:↓
 8048d6c:       55                      push   %ebp↓
 8048d6d:       89 e5                   mov    %esp,%ebp↓
 8048d6f:       83 ec 18                sub    $0x18,%esp↓
 8048d72:       b8 4c a0 04 08          mov    $0x804a04c,%eax↓
 8048d77:       89 04 24                mov    %eax,(%esp)↓
 8048d7a:       e8 c1 f7 ff ff          call   8048540 <printf@plt>↓
 8048d7f:       c7 04 24 01 00 00 00    movl   $0x1,(%esp)↓
 8048d86:       e8 15 f8 ff ff          call   80485a0 <sleep@plt>↓
 8048d8b:       a1 44 c0 04 08          mov    0x804c044,%eax↓
 8048d90:       3d e8 03 00 00          cmp    $0x3e8,%eax↓
 8048d95:       0f 85 8c 00 00 00       jne    8048e27 <lala_s+0xbb>↓
 8048d9b:       eb 46                   jmp    8048de3 <lala_s+0x77>↓
 8048d9d:       a1 9c c0 04 08          mov    0x804c09c,%eax↓
 8048da2:       8b 15 90 c0 04 08       mov    0x804c090,%edx↓
 8048da8:       c1 e2 02                shl    $0x2,%edx↓
 8048dab:       8d 14 10                lea    (%eax,%edx,1),%edx↓
 8048dae:       a1 9c c0 04 08          mov    0x804c09c,%eax↓
 8048db3:       8b 0d 90 c0 04 08       mov    0x804c090,%ecx↓
 8048db9:       c1 e1 02                shl    $0x2,%ecx↓
 8048dbc:       01 c8                   add    %ecx,%eax↓
 8048dbe:       8b 00                   mov    (%eax),%eax↓
 8048dc0:       8b 40 04                mov    0x4(%eax),%eax↓
 8048dc3:       89 02                   mov    %eax,(%edx)↓
 8048dc5:       a1 9c c0 04 08          mov    0x804c09c,%eax↓
 8048dca:       8b 15 90 c0 04 08       mov    0x804c090,%edx↓
 8048dd0:       c1 e2 02                shl    $0x2,%edx↓
 8048dd3:       01 d0                   add    %edx,%eax↓
 8048dd5:       8b 00                   mov    (%eax),%eax↓
```

```
Disassembly of section .comment:↓
↓
00000000 <.comment>:↓
   0:   47                      inc    %edi↓
   1:   43                      inc    %ebx↓
   2:   43                      inc    %ebx↓
   3:   3a 20                   cmp    (%eax),%ah↓
   5:   28 55 62                sub    %dl,0x62(%ebp)↓
   8:   75 6e                   jne    78 <_init-0x8048428>↓
   a:   74 75                   je     81 <_init-0x804841f>↓
   c:   2f                      das    ↓
   d:   4c                      dec    %esp↓
   e:   69 6e 61 72 6f 20 34    imul   $0x34206f72,0x61(%esi),%ebp↓
  15:   2e                      cs↓
  16:   34 2e                   xor    $0x2e,%al↓
  18:   34 2d                   xor    $0x2d,%al↓
  1a:   31 34 75 62 75 6e 74    xor    %esi,0x746e7562(,%esi,2)↓
  21:   75 35                   jne    58 <_init-0x8048448>↓
  23:   29 20                   sub    %esp,(%eax)↓
  25:   34 2e                   xor    $0x2e,%al↓
  27:   34 2e                   xor    $0x2e,%al↓
  29:   35                      .byte 0x35↓
        ...↓
←
```

6

Πειραματική Μελέτη Επιθέσεων

6.1 Πειράματα σε μεταθετικό αναγώγιμο γράφημα (υδατογράφημα)

Στο σημείο αυτό, έχοντας περάσει από όλα τα στάδια της δημιουργίας, της απόδειξης γνησιότητας, της ενσωμάτωσης και της εξαγωγής υδατογραφήματος (μεταθετικό αναγώγιμο γράφημα) σε λογισμικό, καθίσταται αναγκαίο να λάβουν χώρα κάποια πειράματα προκειμένου να εξαχθούν συμπεράσματα σχετικά με την ανθεκτικότητα στις εκάστοτε επιθέσεις των κακόβουλων χρηστών. Τα πειράματα αυτά θα λάβουν χώρα με την εξής σειρά επιπέδων, από το υψηλό στο χαμηλό επίπεδο και κάθε ένα από αυτά χωρίζεται σε επιμέρους επίπεδα. Στο υψηλό επίπεδο υπάρχει η εξωτερική δομή του μεταθετικού αναγώγιμου γραφήματος, η οποία αποτελείται από το επίπεδο των ακμών, των κόμβων καθώς και από το επίπεδο που σχετίζεται με τις ετικέτες των κόμβων. Ονομάζεται υψηλό επίπεδο, διότι τα επιμέρους υποεπίπεδά του είναι εμφανή με γυμνό μάτι. Από την άλλη πλευρά στο χαμηλό επίπεδο υπάρχει η εσωτερική δομή του μεταθετικού αναγώγιμου γραφήματος, η οποία στην συγκεκριμένη περίπτωση αποτελείται από την αυτοαναστρέφουσα μετάθεση, την bitonic permutation και μία δομή εν ονόματι B′ η οποία είναι σε δυαδική μορφή.

Για λόγους καλύτερης κατανόησης και παρακολούθησης της εργασίας με πιο ευχάριστο τρόπο, σημειώνεται πως τις επιθέσεις που θα ακολουθήσουν τις διεξάγει ένας εισβολέας εν ονόματι Ιωάννης. Ο εισβολέας αυτός, αφού έχει καταφέρει έχει εντοπίσει και έχει εξάγει το υδατογράφημα από το λογισμικό που έχει υποκλέψει προηγουμένως, δίχως βέβαια να αλλοιώσει στο ελάχιστο την ροή εκτέλεσης του, ξεκινά τις επιθέσεις του όπως έχει προαναφερθεί, από το υψηλό επίπεδο του υδατογραφήματος κατεβαίνοντας σιγά - σιγά στο χαμηλό επίπεδο περνώντας προφανώς από τα κατάλληλα υποεπίπεδα καθ' όλη τη διάρκεια διεξαγωγής των επιθέσεων. Στο σημείο αυτό, δίνοντας μία γεύση του τι θα ακολουθήσει, αξίζει να σημειωθεί πως τα ποσοστά από τις επιθέσεις οι οποίες δεν γίνονται αντιληπτές είναι γενικά μικρά, από 10% έως κάτι περισσότερο από 0% και όλες οι πιθανότητες έχουν υπολογιστή με πολύ μεγάλη ακρίβεια. Σημειώνεται επίσης πως το επίπεδο το οποίο είναι

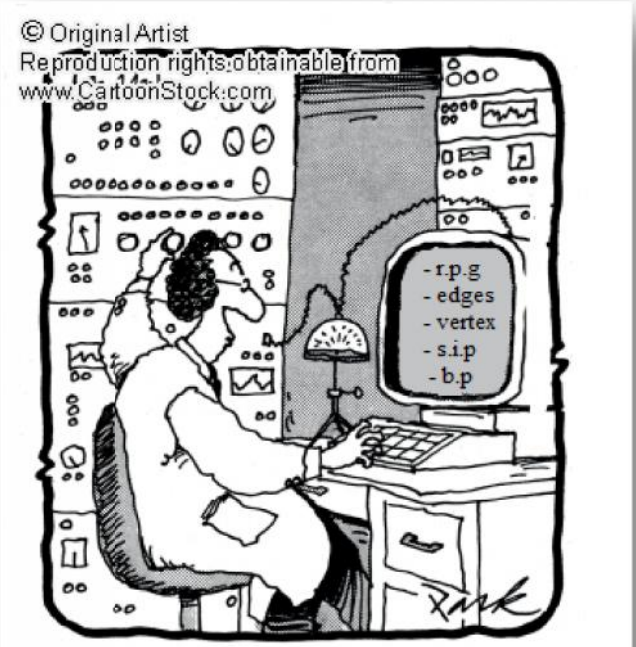
ζωτικής σημασίας για το μεταθετικό αναγώγιμο γράφημα είναι το πιο χαμηλό από όλα τα άλλα και δεν είναι άλλο από τη δυαδική δομή B′.

Στο σημείο αυτό, σημειώνονται οι υποθέσεις που κάνει σε κάθε του επίθεση ο Ιωάννης, προτού αρχίσει να τροποποιεί. Πιο συγκεκριμένα, στο υψηλό επίπεδο το οποίο εμπεριέχει τους κόμβους, τις ετικέτες των κόμβων και τις ακμές, το μόνο που γνωρίζει ο Ιωάννης είναι πως το υδατογράφημα αυτό είναι απλά ένα απλό γράφημα. Επομένως, όλες του οι ενέργειες γίνονται δίχως κάποια συγκεκριμένη γνώση. Στη συνέχεια, όταν φτάσει να διαπράξει επίθεση στο χαμηλό επίπεδο και συγκεκριμένα στο υποεπίπεδο της αυτοαναστρέφουσας μετάθεσης αρχικά, οι γνώσεις που έχει είναι πως το υδατογράφημα το οποίο έχει εξάγει είναι ένα μεταθετικό αναγώγιμο γράφημα και πως από αυτό μπορεί να προκύψει μία μετάθεση, χωρίς όμως να γνωρίζει πως αυτή είναι αυτοαναστρέφουσα και απλά επιτίθεται σε αυτή. Συνεχίζοντας όμως, στην αμέσως επόμενη επίθεση του ο Ιωάννης εξακολουθεί να επιτίθεται στη μετάθεση αυτή με την διαφορά πως τώρα γνωρίζει πως είναι αυτοαναστρέφουσα, αλλά όχι την επιπλέον ιδιότητά της (έχει μονάχα έναν μονό κύκλο και όλους τους άλλους διπλούς).Κάνοντας ένα βήμα πιο κάτω, ο Ιωάννης επιτίθεται στην bitonic permutation με το ίδιο ακριβώς σκεπτικό της αυτοαναστρέφουσας μετάθεσης. Με άλλα λόγια, αρχικά γνωρίζει πως η μετάθεση αυτή δεν έχει κάποιο επιπλέον χαρακτηριστικό και στην συνέχεια πως γνωρίζει πως είναι bitonic permutation, αλλά όχι πως αναγκαστικά στην αρχή μονοτονικά αυξάνεται και εν συνεχεία μονοτονικά μειώνεται. Τέλος, ο εν λόγω εισβολέας επιτίθεται στην δυαδική δομή (B′) και το μόνο που γνωρίζει για αυτή είναι πως είναι μία απλή μετάθεση δίχως κάποιο ιδιαίτερο χαρακτηριστικό.

Δεν πρέπει να παραληφθεί πως, στο κεφάλαιο αυτό έχουν γίνει επιθέσεις από τον Ιωάννη σε αυτοαναστρέφουσες επιθέσεις, γνωρίζοντας μονάχα πως είναι απλές μεταθέσεις. Πιο συγκεκριμένα αυτό που έγινε είναι, να παραχθούν 100000 αυτοαναστρέφουσες μεταθέσεις για κάθε κατηγορία (11 στοιχείων, 21 στοιχείων, 31 στοιχείων έως και 91 στοιχείων) και να αλλαχθούν σε κάθε μία από αυτές τέσσερα τυχαία στοιχεία με επίσης τυχαίο τρόπο.

Απώτερος σκοπός όλων αυτών των επιθέσεων είναι η μετάλλαξη του μεταθετικού αναγώγιμου γραφήματος με τέτοιον τρόπο έτσι ώστε να μην αλλοιωθεί καμία ιδιότητά του και εν συνεχεία η επανατοποθέτησή του στο υποκλεμμένο λογισμικό χωρίς βέβαια να μεταβληθεί στο ελάχιστο η ροή εκτέλεσης του καθώς και η εξαγωγή πιθανοτήτων από τις ενέργειες αυτές..

Παρακάτω παρατίθεται μία σατιρική εικόνα σχετική με τα πειράματα στα διάφορα επίπεδα των υδατογραφημάτων λογισμικού:



Αρχικά, ο εισβολέας εν ονόματι Ιωάννης διεξάγει επίθεση στις ακμές του μεταθετικού αναγώγιμου γραφήματος (υψηλό επίπεδο) με σκοπό να το μετατρέψει και να το επανεισάγει, χωρίς όμως να γίνει αντιληπτή η επίθεσή του από το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης στον έλεγχο ακμών, όπου στην συγκεκριμένη περίπτωση το πρόγραμμα αυτό μεταβαίνει από το μεταθετικό αναγώγιμο γράφημα στον φυσικό αριθμό από τον οποίο προήλθε (υδατόσημα). Άξιο υπενθύμισης είναι, πως το εν λόγω πρόγραμμα το χρησιμοποιεί ο εκάστοτε νόμιμος ιδιοκτήτης λογισμικού προκειμένου να αποδείξει πως μέσα στο λογισμικό είναι ενσωματωμένο το υδατογράφημά του, πραγματοποιώντας παράλληλα ελέγχους με την βοήθεια ενός μενού για τυχόν επιθέσεις που θα μπορούσαν να είχαν συμβεί στα διάφορα επίπεδα.

Στο σημείο αυτό σημειώνεται μία χρήσιμη πληροφορία σύμφωνα με την οποία, οποιαδήποτε επίθεση και να διαπράξει ο Ιωάννης μετακινώντας μονάχα μία ακμή διατηρώντας την ιδιότητά της και πιο συγκεκριμένα μία πίσω ακμή, είναι καταδικασμένη να εντοπιστεί καθώς με τον τρόπο αυτό μπορεί να μην χαλάει η ιδιότητα της εξωτερικής δομής του υδατογραφήματος αλλά χαλάει αυτή της αυτοαναστρέφουσας μετάθεσης, της bitonic permutation και εν συνεχεία της δυαδικής δομής με το όνομα Β′. Με την έκφραση ‘διατηρώντας την ιδιότητά της’ εννοείται πως αν μία ακμή είναι πίσω ακμή, αν αφαιρεθεί και εισαχθεί σε κάποιο άλλο σημείο θα παραμείνει πίσω ακμή. Σημειώνεται επίσης, πως καμία από τις μπροστά ακμές δεν μπορεί να αλλάξει καθώς θα χαλάσει κατευθείαν η δομή του μεταθετικού αναγώγιμου γραφήματος, πράγμα όπου στις επιθέσεις στην εξωτερική δομή του υδατογραφήματος ο εισβολέας δεν γνωρίζει. Επομένως, μία τέτοια μικρή και γρήγορη επίθεση, είτε μπροστινής, είτε όπισθεν ακμής θα ήταν εντελώς άκαρπη και η πιθανότητα εντοπισμού αυτής με το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης κάνοντας έλεγχο στην αυτοαναστρέφουσα μετάθεση αλλά και στην bitonic permutation, είναι 100% επιτυχής.

Συνεχίζοντας, αξίζει να τονιστεί πως όλες οι πιθανές επιθέσεις σε ακμές, αφαιρώντας αρχικά μόνο πίσω ακμές και εισάγοντας πάλι πίσω διατηρώντας αμετάβλητο τον αρχικό κόμβο, (για παράδειγμα από τον κόμβο x στον κόμβο y αφαιρείται και εισάγεται μία άλλη πάλι από τον κόμβο x σε έναν άλλον κόμβο z) για υδατογραφήματα με 9 κόμβους είναι 7!, για υδατογραφήματα με 11 κόμβους είναι 9!, για υδατογραφήματα με 13 κόμβους είναι 11!, για υδατογραφήματα με 15 κόμβους είναι 13! και τέλος για υδατογραφήματα με 17 κόμβους, τα οποία είναι και τα μεγαλύτερα σε έκταση υδατογραφήματα που εξετάζονται στην παρούσα πτυχιακή εργασία, είναι 15!. Από την άλλη πλευρά αφαιρώντας είτε, μπροστά ακμές είτε, πίσω ακμές και εισάγοντας εν συνεχεία πάλι είτε, μπροστά ακμές είτε, πίσω για υδατογραφήματα με 9 κόμβους όλες οι πιθανές επιθέσεις είναι 8^15, για υδατογραφήματα με 11 κόμβους είναι 10^19, για υδατογραφήματα με 13 κόμβους είναι 12^23, για υδατογραφήματα με 15 κόμβους είναι 14^27 και τέλος για υδατογραφήματα τα οποία έχουν 17 κόμβους όλες οι πιθανές επιθέσεις είναι 16^31. Επομένως από αυτά τα νούμερα και μόνο, εύκολα βγαίνει το συμπέρασμα πως ο Ιωάννης έχει την δυνατότητα να διαπράξει μία πληθώρα επιθέσεων σε ακμές του υδατογραφήματος, αλλά σύντομα θα αποδειχτεί πως μόνον ελάχιστες από αυτές θα περάσουν απαρατήρητες. Αυτό θα συμβεί καθώς στο μεγαλύτερο ποσοστό αυτών των επιθέσεων προκαλείται αλλοίωση είτε, στην δομή του μεταθετικού αναγώγιμου γραφήματος είτε, στην αυτοαναστρέφουσα μετάθεση είτε, στην bitonic permutation είτε ακόμη και στην δομή ενός ακόμα χαμηλότερου επιπέδου, το οποίο σχετίζεται με την δομή της bitonic permutation και είναι σε δυαδική μορφή το οποίο τελικά θα καταστεί και ως το πιο σημαντικό υποεπίπεδο όλου του υδατογραφήματος. Η δομή αυτή όπως έχει ήδη αναφερθεί ονομάζεται B′.

Στο σημείο αυτό, ο Ιωάννης επιθυμεί να διαπράξει επίθεση στις ακμές του υδατογραφήματος, εξάγοντας μία πίσω ακμή και εισάγοντας μία άλλη εξίσου πίση ακμή κρατώντας το αρχικό σημείο (σημείο εκκίνησης) αμετάβλητο. Ο σκοπός του είναι να αλλάξει το υδατογράφημα, δίχως να αλλοιώσει την εξωτερική δομή του και δίχως να γίνει αντιληπτή η επίθεσή του από το πρόγραμμα το οποίο εκτελεί την αντίστροφη διαδικασία υδατογράφησης με έλεγχο στις ακμές.

Πιο συγκεκριμένα, ο Ιωάννης διαπράττοντας αυτού του είδους την επίθεση για υδατογράφημα με 9 κόμβους, έχει πιθανότητα να μην γίνει αντιληπτή η κίνησή του 2,304526748971%. Αυτή η πιθανότητα εξάγεται με τον ακόλουθο τρόπο. Αρχικά η πιθανότητα ο Ιωάννης να βγάλει μία πίσω ακμή είναι 7/15 και αυτό διότι όλες οι ακμές είναι 15 στον αριθμό και μόνο 7 από αυτές είναι πίσω ακμές. Στην συνέχεια, η πιθανότητα να μην αλλάξει ο αρχικός κόμβος της εξαχθείσας ακμής είναι 1/9 καθώς 9 είναι στον αριθμό όλοι οι κόμβοι του υδατογραφήματος και τέλος η πιθανότητα ο τελικός κόμβος της νέας ακμής να είναι μεγαλύτερος του αρχικού καθώς πρέπει να είναι πίσω ακμή 28/63. Η τελευταία πιθανότητα εξάγεται ακολουθώντας τον εξής τρόπο. Πρώτον από τον κόμβο με την ετικέτα 7 η πιθανότητα η πίσω ακμή του να αφαιρεθεί και να επανεισαχθεί, καθώς είναι γνωστό πως ο μεγαλύτερος κόμβος έχει πάντα πίσω ακμή στον κόμβο με την ετικέτα s, είναι 1/9, δεύτερον από τον κόμβο με την ετικέτα 6 η πιθανότητα η πίσω του ακμή να αφαιρεθεί και να εισαχθεί μία καινούρια εξίσου πίσω ακμή είναι 2/9 καθώς οι μόνοι κόμβοι οι οποίοι θα μπορούσαν να αποτελέσουν τελικό κόμβο αυτής της ακμής διατηρώντας την πίσω ακμή είναι οι κόμβοι με τις ετικέτες 7 και s, άρα δύο στον αριθμό. Με τον ίδιο τρόπο υπολογίζονται και όλες οι άλλες πιθανότητες μέχρι τον κόμβο 1 και τέλος αφού έχουν υπολογιστεί όλες αυτές οι πιθανότητες προστίθενται και εν συνεχεία διαιρούνται με τον αριθμό 7 καθώς πρέπει να βγει ο μέσος όρος

των υπολογισθέντων πιθανοτήτων. Η διαίρεση αυτή λαμβάνει χώρα καθώς όλα τα ενδεχόμενα έχουν την ίδια ακριβώς πιθανότητα να συμβούν.

Στη συνέχεια ο Ιωάννης επιθυμεί να διαπράξει επίθεση σε υδατογράφημα με 9 κόμβους πάλι, άλλα αυτήν την φορά αλλάζοντας με τον ίδιο ακριβώς τρόπο δύο ακμές. Καθώς η όλη διαδικασία εξαγωγής των πιθανοτήτων έχει περιγραφεί στην αμέσως προηγούμενη παράγραφο θα ήταν κάπως φλύαρο να επαναληφθεί, επομένως δίνεται αμέσως η τελική πιθανότητα ο εισβολέας αυτός να δημιουργήσει ένα καινούριο μεταθετικό αναγώγιμο γράφημα και είναι 0,05310843536722846 5%. Εν συνεχεία ο Ιωάννης επιθυμεί να επιτεθεί μετακινώντας τρείς πίσω ακμές, επομένως η πιθανότητα επιτυχίας του γίνεται 0,0012238980899775490961 1% και τέλος επιθυμεί να επιτεθεί μετακινώντας τέσσερις πίσω ακμές με αποτέλεσμα η πιθανότητα επιτυχίας του να κατέβει στο 0,0000282050590715508323527369 18%.

Με το ίδιο ακριβώς σκεπτικό των τριών προηγουμένων παραγράφων ο Ιωάννης επιτίθεται σε υδατογραφήματα με 11, 13, 15 και 17 κόμβους. Για λόγους πληρότητας και κατανόησης της δυσκολίας που θα αντιμετωπίσει ο Ιωάννης και ο εκάστοτε εισβολέας προκειμένου να μην γίνει αντιληπτή η επίθεσή του από το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης δίνονται και οι πιθανότητες αυτών των επιθέσεων.

Αρχικά στα υδατογραφήματα με 11 κόμβους η πιθανότητα με μία μονάχα επίθεση να το κρατήσει στη σωστή δομή του μεταθετικού αναγώγιμου γραφήματος με σκοπό αυτή του η ενέργεια να μην γίνει αντιληπτή από το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης με έλεγχο στις ακμές είναι 1,95737277077%. Συνεχίζοντας πάνω στο υδατογραφήματα με τον ίδιο αριθμό κόμβων ο Ιωάννης επιτίθεται σε αυτά μετακινώντας δύο ακμές. Έτσι, η πιθανότητα τώρα να το κρατήσει στην σωστή δομή η οποία περνά από τους ελέγχους ακμών απαρατήρητη είναι 0,03831308163751827%. Στη συνέχεια διαπράττει επίθεση μετακινώντας τρείς ακμές με αποτέλεσμα η πιθανότητα, επιτυχίας του να πέφτει στο 0,0007499298276156634493 6% και τέλος μετακινώντας τέσσερις ακμές η πιθανότητά του πέφτει στο 0,0000146789222456313962852559 9%.

Στη συνέχεια ο Ιωάννης, μεταβαίνει επιτίθεται σε υδατογραφήματα τα οποία έχουν 13 κόμβους και διαπράττει την πρώτη του επίθεση σε αυτά μετακινώντας μονάχα μία πίσω ακμή με τον τρόπο που έχει αναφερθεί στις προηγούμενες παραγράφους. Με αποτέλεσμα, η πιθανότητά αυτή του η κίνηση να περάσει απαρατήρητη είναι 1,697967584255%. Από την άλλη πλευρά αυξάνοντας τον αριθμό των μετακινούμενων ακμών στις δύο, η πιθανότητα αυτή πέφτει στο 0,028830939171807605%, ενώ αν οι ακμές αυτές γίνουν τρείς στον αριθμό τότε η πιθανότητα γίνεται 0,0004895400013735700946 35%. Τέλος αν οι ακμές αυτές γίνουν τέσσερις, η πιθανότητα γίνεται 0,0000095821226889133450198252 11%.

Με τον τρόπο αυτό, ο Ιωάννης προχωρά στα υδατογραφήματα με 15 κόμβους στα οποία αρχικά θα διαπράξει επίθεση μετακινώντας μονάχα μία πίσω ακμή με τον γνωστό τρόπο. Έτσι με την κίνησή του αυτή έχει πιθανότητα επιτυχίας 1,358598908987%. Συνεχίζοντας, διαπράττει επίθεση αλλάζοντας δύο ακμές, με αποτέλεσμα η πιθανότητά του αυτή την φορά να είναι 0,018457909955006667%. Κάνοντας ακόμα ένα βήμα και αλλάζοντας ακόμα μία ακμή η πιθανότητά του πέφτει στο 0,0002507689632705234404 45%, ενώ την φορά που αλλάζει τέσσερις ακμές η πιθανότητά του γίνεται 0,0000034069443990713422152498 67%.

Τέλος ο Ιωάννης επιτίθεται στα μεγαλύτερα υδατογραφήματα με τα οποία ασχολείται η περούσα πτυχιακή εργασία και έχουν συνολικό αριθμό από κόμβους 17. Έτσι, ξεκινά τις επιθέσεις του μετακινώντας μονάχα μία ακμή, με αποτέλεσμα η πιθανότητά του να είναι 1,339435204822%. Στην συνέχεια, μετακινεί δύο ακμές και η πιθανότητά του γίνεται αμέσως 0,017940866679165531%, ενώ μετακινώντας τρεις ακμές η πιθανότητα του πέφτει ακόμα περισσότερο και φτάνει στο 0,000240306284350922779 75%. Τέλος, μετακινώντας τέσσερις ακμές η πιθανότητα αγγίζει το 0.000003218746971995920268191 47%.

Προχωρώντας στο ίδιο επίπεδο γίνεται η μετάβαση στις επιθέσεις κόμβων του μεταθετικού αναγώγιμου γραφήματος. Αρχικά, σημειώνεται πως οποιοδήποτε υδατογράφημα και να δημιουργηθεί με βάση τους αρχικούς δύο αλγορίθμους δημιουργίας υδατογραφήματος θα έχει αναγκαστικά περιττό πλήθος από κόμβους (είτε δημιουργείται από υδατόσημα μέχρι το 127 τα οποία εξετάζονται σε αυτήν την πτυχιακή εργασία λεπτομερώς, είτε είναι μεγαλύτερα από αυτόν τον φυσικό αριθμό), με αποτέλεσμα σε περίπτωση που ο Ιωάννης επιχειρήσει να εισάγει ή ακόμα και να διαγράψει έναν ή περισσότερους κόμβους καθιστώντας το τελικό υδατογράφημα να έχει άρτιο πλήθος από κόμβους τότε απευθείας γίνεται αντιληπτή η επίθεση ακόμα και δια γυμνού οφθαλμού και το ποσοστό επιτυχίας του προγράμματος σε αυτήν την περίπτωση είναι 100%. Στην περίπτωση όμως που το τελικό υδατογράφημα έχει περιττό αριθμό από κόμβους, τότε αν ο Ιωάννης δεν έχει φροντίσει να προσθέσει ακμές σε αυτούς τους κόμβους τότε πάλι η επίθεση γίνεται αντιληπτή δια γυμνού οφθαλμού και το ποσοστό συνεχίζει να είναι 100%. Αν όμως, αυτός έχει προνοήσει και έχει προσθέσει και ακμές δημιουργώντας ένα καινούριο μεταθετικό αναγώγιμο γράφημα τότε η επίθεση αυτή θα πρέπει να ελεγχθεί με βάση τις ακμές του υδατογραφήματος και όχι τους κόμβους αυτού.

Βαδίζοντας ακόμη στο υψηλό επίπεδο του μεταθετικού αναγώγιμου γραφήματος, ο Ιωάννης διεξάγει επίθεση αυτήν την φορά στις ετικέτες των κόμβων και όχι στους κόμβους αυτούς καθ’ αυτούς . Πιο συγκεκριμένα με αυτήν την επίθεση, αυτό που στοχεύει είναι να ανακατέψει τις ετικέτες των κόμβων αναμεταξύ τους δημιουργώντας μία σύγχυση στο υδατογράφημα, αλλά και στο εκάστοτε πρόγραμμα το οποίο είναι υπεύθυνο για την απόδειξη γνησιότητας (στην παρούσα πτυχιακή εργασία το πρόγραμμα αυτό εκτελεί την αντίστροφη διαδικασία υδατογράφησης). Πάρα ταύτα, λόγω μίας σημαντικής ιδιότητας του μεταθετικού αναγώγιμου γραφήματος, η οποία είναι το μοναδικό Hamiltonian path που έχει, όπως και να μπερδέψει τις ετικέτες των κόμβων ο εισβολέας, το πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης είναι σε θέση να ξαναδημιουργήσει το αρχικό υδατογράφημα πολύ εύκολα και γρήγορα. Με αποτέλεσμα το ποσοστό επιτυχούς εντοπισμού επίθεσης του προγράμματος σε αυτού του είδους την επίθεση να είναι 100%.

Κλείνοντας με τις επιθέσεις στο υψηλό επίπεδο του υδατογραφήματος σημειώνεται πως κάποιες από αυτές τις επιθέσεις είναι όχι μόνο εύκολες στο να εντοπιστούν, αλλά με την βοήθεια του προγράμματος που εκτελεί την αντίστροφη διαδικασία υδατογράφησης είναι και εύκολες στην επιδιόρθωσή τους, πράγμα το οποίο και έχει υλοποιηθεί στο συγκεκριμένο πρόγραμμα. Από την άλλη πλευρά όμως, υπάρχουν και κάποιες άλλες επιθέσεις οι οποίες, είτε εμφανίζονται στο υψηλό επίπεδο, είτε στο χαμηλό προκαλούν ανεπανόρθωτη ζημία στο υδατογράφημα. Επίσης, όπως ίσως έχει γίνει φανερό και λίγες παραγράφους πριν, τις ανεπανόρθωτες ζημίες το πρόγραμμα το οποίο εκτελεί την αντίστροφη διαδικασία υδατογράφησης ως επί το πλείστον τις εντοπίζει και εν συνεχεία τερματίζει την λειτουργία του, αλλά υπάρχουν όπως θα γίνει καλύτερα αντιληπτό και στην συνέχεια και κάποιες επιθέσεις οι οποίες περνούν απαρατήρητές απ’ όλους του ελέγχους και το ποσοστό αυτών για

μικρά υδατογραφήματα είναι μεταξύ του 2% και του 10%, ενώ για μεγάλα είναι πολύ πιο κάτω από το 1%, τείνοντας στο 0% καθώς το πλήθος των κόμβων αυξάνεται.

Προχωρώντας στην πειραματική μελέτη επιθέσεων, γίνεται η μετάβαση από το υψηλότερο επίπεδο του υδατογραφήματος στο χαμηλότερο, στο οποίο οι επιθέσεις δεν είναι ούτε τόσο ευδιάκριτες, ούτε τόσο εύκολα υλοποιήσιμες. Πριν αρχίσουν να παρατίθενται τα αποτελέσματα των πειραμάτων στις εκάστοτε επιθέσεις, σημειώνεται πως σε αυτό το επίπεδο έχουν γίνει πειράματα για όλα τα υδατογραφήματα τα οποία περιέχουν μέχρι 13 κόμβους σαρώνοντας όλες τις πιθανές επιθέσεις που θα μπορούσαν να λάβουν χώρα σε αυτά, εξάγοντας με τον τρόπο αυτόν τις καλύτερες δυνατές πιθανότητες και εν συνεχεία τα καλύτερα δυνατά συμπεράσματα από αυτές. Ο λόγος που έγιναν σαρωτικές επιθέσεις μέχρι αυτό το μέγεθος υδατογραφημάτων είναι διότι από εκεί και μετά τα δεδομένα είναι της τάξης των terabytes και πάνω, επομένως ο χειρισμός τους είναι ανέφικτος.

Αρχικά, ο εισβολέας Ιωάννης επιθυμεί να επιτεθεί στην αυτοαναστρέφουσα μετάθεση, αλλά με βάση την υπόθεση η οποία έχει γίνει στην αρχή αυτού του υποκεφαλαίου αυτού, η επίθεση αυτή θα διεξαχθεί δίχως ο εισβολέας να γνωρίζει δύο κύρια σημεία. Τα σημεία αυτά, όπως θα γίνει φανερό και στην συνέχεια αποτελούν μοναδικό μονοπάτι στην μετάβαση από την αυτοαναστρέφουσα μετάθεση στην bitonic permutation και σε περίπτωση παραβίασής τους αλλοιώνεται η ιδιότητα της bitonic permutation τόσο η γνωστή όσο και η κρυφή (αρχικά μονοτονικά αυξάνεται και τελικά μονοτονικά μειώνεται).

Επομένως, ο εισβολέας αυτός έχοντας στα χέρια του το μεταθετικό αναγώγιμο γράφημα μεταβαίνει στην αυτοαναστρέφουσα μετάθεση (μετά από τα απαραίτητα βήματα φυσικά). Αφού έχει πραγματοποιηθεί αυτή η μετάβαση με τον σωστό τρόπο, η πιθανότητα, να την αλλάξει και στη συνέχεια να την ξαναμετατρέψει σε μεταθετικό αναγώγιμο γράφημα έτοιμο για εισαγωγή χωρίς να την αλλοιώσει για υδατογράφημα με 9 κόμβους είναι 4,603174603175%, για υδατογράφημα με 11 κόμβους είναι 0,722001763668%, για υδατογράφημα με 13 κόμβους είναι 0,089426006093% και για υδατογραφήματα με 15 και 17 κόμβους οι πιθανότητες είναι ακόμα μικρότερες, καθώς με την αύξηση των στοιχείων της μετάθεσης η πιθανότητα δημιουργίας μίας σωστής μειώνεται με γοργούς ρυθμούς.

Στο σημείο αυτό αξίζει να σημειωθεί πως στο υποεπίπεδο της αυτοαναστρέφουσας μετάθεσης έχει λάβει χώρα και ένα άλλο πείραμα με την βοήθεια του προγράμματος το οποίο βρίσκεται στο παράρτημα 5. Στα πλαίσια του συγκεκριμένου πειράματος παράγονται 100 000 αυτοαναστρέφουσες μεταθέσεις με την βοήθεια της συνάρτησης των τυχαίων αριθμών οι οποίες έχουν μήκος 11, 21, 31, 41, 51, 61, 71, 81 και 91 στοιχείων (δηλαδή τα μεταθετικά γραφήματα που δημιουργούνται από αυτές έχουν 13, 23, 33, 43, 53, 63, 73, 83 και 93 κόμβους αντίστοιχα). Σε αυτές λοιπόν τις μεταθέσεις ο εν λόγω εισβολέας διαλέγει με τυχαίο τρόπο τέσσερα στοιχεία (καθώς έχει φτάσει στο υποεπίπεδο αυτό αλλά δεν γνωρίζει πως η μετάθεση αυτή είναι αυτοαναστρέφουσα) και τους αλλάζει θέση. Με τον τρόπο αυτό οι πιθανότητες που έχουν αυτές του οι κινήσεις να μην γίνει αντιληπτές στο υποεπίπεδο αυτό ή για αλλαγή αυτή την φορά οι πιθανότητες που έχει ο νόμιμος ιδιοκτήτης να αντιληφθεί την εκάστοτε επίθεση για αυτοαναστρέφουσα μετάθεση με 11 στοιχεία και εν συνεχεία υδατογράφημα με 13 κόμβους είναι 96,972%, αντίστοιχα για 23 κόμβους είναι 99,24%, για 33 είναι 99,72%, για 43 είναι 99,809%, για 53 είναι 99,872%, για 63 είναι 99,914%, για 73 είναι 99,947%, για 83 είναι 99,95% και τέλος για 93 είναι 99,964% (από τις 100000 αυτοαναστρέφουσες μεταθέσεις έγιναν αντιληπτές οι 96972 έχοντας συνολοικό αριθμό στοιχείων 11, αντίστοιχα έγιναν αντιληπτές οι 99240 με συνολικό αριθμό στοιχείων 21, εν

συνεχεία οι 99720 με συνολικό αριθμό στοιχείων 31, οι 99809 με συνολικό αριθμό στοιχείων 41, οι 99872 με συνολικό αριθμό στοιχείων 51, οι 99914 με συνολικό αριθμό 61, οι 99947 με συνολικό αριθμό 71, οι 99950 με συνολικό αριθμό 81 και τέλος οι 99964 με συνολικό αριθμό στοιχείων 91) και στο υποεπίπεδο του μεταθετικού αναγώγιμου γραφήματος να είναι 100%. Με άλλα λόγια στο κρυφό υποεπίπεδο ο εισβολέας έχει πολύ μικρή πιθανότητα επιτυχίας, ενώ στο ορατό υποεπίπεδο του γραφήματος να έχει πλήρης επιτυχία.

Για λόγους συνοχής, στο σημείο αυτό δίνεται πως για υδατογραφήματα με 9 κόμβους η αυτοαναστρέφουσα μετάθεση έχει 7 στοιχεία και όλες οι πιθανές μεταθέσεις με 7 στοιχεία είναι 5040 και από αυτές τις 5040 μόνον οι 232 είναι αυτοαναστρέφουσες μεταθέσεις με την γενική έννοια, για υδατογραφήματα με 11 κόμβους η αυτοαναστρέφουσα μετάθεση έχει 9 στοιχεία και όλες οι πιθανές μεταθέσεις με 9 στοιχεία είναι 362880 και από αυτές τις 362880 μόνον οι 2620 είναι αυτοαναστρέφουσες μεταθέσεις και τέλος για υδατογραφήματα με 13 κόμβους η αυτοαναστρέφουσα μετάθεση έχει 11 στοιχεία και όλες οι πιθανές μεταθέσεις με 11 στοιχεία είναι 39916800 και από αυτές τις 39916800 μόνον οι 35696 είναι αυτοαναστρέφουσες μεταθέσεις. Συνοψίζοντας, όσο αυξάνονται οι κόμβοι του μεταθετικού αναγώγιμου γραφήματος τόσο αυξάνονται και τα στοιχεία της αυτοαναστρέφουσας μετάθεσης, με αποτέλεσμα να αυξάνονται παραγοντικά όλες οι πιθανές μεταθέσεις με τα εκάστοτε πλήθη στοιχείων ρίχνοντας πάρα πολύ χαμηλά το ποσοστό να μην γίνει αντιληπτή μία επίθεση, με αποτέλεσμα οι εκάστοτε εισβολείς και στην συγκεκριμένη περίπτωση ο Ιωάννης να έχει τη δυνατότητα να επιλέξει μέσα από ένα πραγματικό χάος μεταθέσεων, την σωστή για αυτόν.

Συνεχίζοντας με τα πειράματα επιθέσεων, ο Ιωάννης επιθυμεί να διεξάγει επίθεση στην αυτοαναστρέφουσα μετάθεση, γνωρίζοντας αυτήν την φορά πως είναι αυτοαναστρέφουσα, αλλά όχι ότι πρέπει να περιέχει μονάχα έναν μονό κύκλο και όλους τους άλλους διπλούς. Πιο συγκεκριμένα, η πιθανότητα ο Ιωάννης να επιτεθεί στην αυτοαναστρέφουσα μετάθεση δεδομένου πως γνωρίζει πλέον τι είναι ακριβώς αυτή η μετάθεση για υδατογράφημα με 9 κόμβους είναι 0,396825396825%, για υδατογράφημα με 11 κόμβους είναι 0,019290123457%, για υδατογράφημα με 13 κόμβους είναι 0,002284752285% και καθώς οι κόμβοι αυξάνονται σε αριθμό τόσο μειώνεται η πιθανότητα να μην γίνει αντιληπτή η επίθεση.

Κλείνοντας με τις επιθέσεις στο μεταθετικό αναγώγιμο γράφημα, αξίζει να σημειωθεί πως λίγο πριν τελειώσει το πρόγραμμα μετατροπής ενός μεταθετικού αναγώγιμου γραφήματος σε έναν φυσικό αριθμό από τον οποίο και προήλθε δημιουργείται μία δομή (μετάθεση) με το όνομα Β′, η οποία περιέχει τόσα στοιχεία όσα και η αυτοαναστρέφουσα μετάθεση καθώς και η bitonic permutation του εκάστοτε υδατογραφήματος. Αυτή η δομή, όπως προαναφέρθηκε είναι η πιο σημαντική, καθώς όπως θα γίνει φανερό στο τέλος δίνει το απόλυτο ποσοστό επιτυχίας στο πρόγραμμα που εκτελεί την αντίστροφη διαδικασία υδατογράφησης καθιστώντας την ως θεμέλιο λίθο του μεταθετικού αναγώγιμου γραφήματος.

Πιο συγκεκριμένα αν το υδατόσημα είναι ο φυσικός αριθμός 12, τότε η δομή Β′ περιέχει 7 στοιχεία εκ των οποίων τα πρώτα κάτω ακέραιο μέρος του 7 είναι πάντοτε μηδενικά, τα υπόλοιπα κάτω ακέραιο μέρος του 7 είναι ο αριθμός 7 στη δυαδική μορφή του και το τελευταίο στοιχείο που μένει είναι πάντοτε άσσος. Έτσι, εξασφαλίζεται μία ένα προς ένα αντιστοιχία της αυτοαναστρέφουσας μετάθεσης με το μεταθετικό αναγώγιμο γράφημα, πράγμα ιδιαίτερα σημαντικό για τον εντοπισμό επιθέσεων. Με αποτέλεσμα οποιαδήποτε επίθεση και να λάβει χώρα στην αυτοαναστρέφουσα μετάθεση και στην bitonic permutation

να γίνεται αντιληπτή, καθώς θα αλλοιώνεται η μορφή της δομής Β', εκτός αν η επίθεση αυτή αλλάζει την αυτοαναστρέφουσα μετάθεση και την bitonic permutation με τέτοιον τρόπο έτσι ώστε να παράγουν άλλο υδατόσημα, εντός των εκάστοτε ορίων φυσικά, για παράδειγμα αν το υδατόσημα είναι όπως και πριν το 12 τότε οι επιθέσεις που δεν θα γίνουν αντιληπτές καθόλου είναι αυτές που αλλάζουν το 12 σε 8, 9, 10, 11, 13, 14 και 15. Με άλλα λόγια, ο μόνος τρόπος να επιτεθεί ο Ιωάννης με επιτυχία είναι να διαπράξει μία επίθεση είτε, στην αυτοαναστρέφουσα μετάθεση είτε, στην bitonic permutation και να τις κάνουν να παράγουν διαφορετικό υδατόσημα εντός των ορίων που προαναφέρθηκαν.

Με τον τρόπο αυτόν, η πιθανότητα να λάβει χώρα μία επίθεση και να μην γίνει αντιληπτή στο χαμηλότερο επίπεδο (δυαδική δομή Β') με την προϋπόθεση πως είναι γνωστά στο εισβολέα τα πάντα εκτός από την τελευταία δομή και εν συνεχεία να περάσει απαρατήρητη από τον νόμιμο ιδιοκτήτη για υδατογραφήματα με 9 κόμβους η πιθανότητα είναι 0,059523809524%, για υδατογραφήματα με 11 κόμβους η πιθανότητα είναι 0,001929012346%, για υδατογραφήματα με 13 κόμβους η πιθανότητα είναι 0,000037578163%, για υδατογραφήματα με 15 κόμβους η πιθανότητα είναι 0,00000049783%, ενώ για τα μεγαλύτερα υδατογραφήματα της παρούσας πτυχιακής εργασίας η πιθανότητα καταλήγει να είναι 0,0000000048177% και καθώς αυξάνονται οι κόμβοι η πιθανότητες τείνουν να μηδενιστούν.

6.2 Απεικόνιση αποτελεσμάτων σε γραφικές παραστάσεις

Παρακάτω παρατίθεται μία σειρά από εικόνες στις οποίες φαίνονται όλα τα προηγούμενα αποτελέσματα του υποκεφαλαίου αυτού (τα όρια των αποτελεσμάτων κυμαίνονται από 0% έως 100%). Σημειώνεται πως πριν από κάθε σύνολο εικόνων υπάρχει παράγραφος στην οποία γίνεται λόγος για το περιεχόμενο τους (με ποιον τρόπο βγήκαν οι εν λόγω πιθανότητες):

- **Πίνακας όλων των πιθανοτήτων του παρόντος κεφαλαίου**

Στην αμέσως παρακάτω εικόνα απεικονίζονται όλες οι πιθανότητες που εξήχθησαν στο υποκεφάλαιο αυτό, είτε με σαρωτικό τρόπο, είτε με τυχαίο. Πιο συγκεκριμένα, οι πιθανότητες των επιθέσεων που λαμβάνουν χώρα στο μεταθετικό αναγώγιμο γράφημα (επίθεση σε μία ακμή του γραφήματος, σε δύο, σε τρεις, σε τέσσερις και στις ετικέτες των κόμβων) πηγάζουν από το γεγονός πως ο εισβολέας, σε αυτό το στάδιο γνωρίζει μόνο πως το υδατογράφημα αυτό είναι απλά ένα γράφημα. Στη συνέχεια οι πιθανότητες των επιθέσεων στην αυτοαναστρέφουσα μετάθεση στην γενική περίπτωση, πηγάζουν από το γεγονός πως ο εισβολέας γνωρίζει μία επιπλέον πληροφορία (πως το υδατογράφημα είναι μεταθετικό αναγώγιμο γράφημα) καθώς και τον τρόπο σύμφωνα με τον οποίον θα μεταβεί από το γράφημα αυτό στην αυτοαναστρέφουσα μετάθεση δίχως όμως να γνωρίζει καμία πληροφορία για αυτή (απλά γνωρίζει πως είναι μία μετάθεση). Εν αντιθέσει, στην αμέσως επόμενη επίθεση (στην αυτοαναστρέφουσα μετάθεση στην ειδική περίπτωση), η πιθανότητα της επίθεσης πηγάζει από το γεγονός πως ο εισβολέας ξέρει επιπλέον πως αυτή η μετάθεση είναι αυτοαναστρέφουσα, αλλά όχι πως πρέπει αναγκάστηκα να έχει μονάχα ένα μονό κύκλο. Εν συνεχεία, οι πιθανότητες των επιθέσεων στην bitonic permutation εξάγονται με τον ίδιο

Σαρωτικές επιθέσεις	r.p.g με 9 κόμβους	r.p.g με 11 κόμβους	r.p.g με 13 κόμβους
Επίθεση σε μία ακμή του r.p.g	2,304526748971	1,957372770770	1,697967584255
Επίθεση σε δύο ακμές του r.p.g	0,0531084353672284	0,0383130816375182	0,0288309391718076
Επίθεση σε τρεις ακμές του r.p.g	0,001223898098997750	0,00074929827615663	0,00048954000137357 0
Επίθεση σε τέσσερις ακμές του r.p.g	0,000028205059071550 80	0,000014678922245631 30	0,000009582122688913 34
Επίθεση σε ετικέτες κόμβων του r.p.g	0,00	0,00	0,00
Επίθεση σε s.i.p (γενικά)	4,603174603175	0,722001763668	0,089426006093
Επίθεση σε s.i.p (με έναν μονό κύκλο)	0,396825396825	0,019290123457	0,002284752285
Επίθεση σε b.p (γενικά)	0,396825396825	0,019290123457	0,002284752285
Επίεθση σε b.p (μ. αυξ μ. μειών)	0,396825396825	0,019290123457	0,002284752285
Επίθεση σε Β′	0,059523809524	0,001929012346	0,000037578163

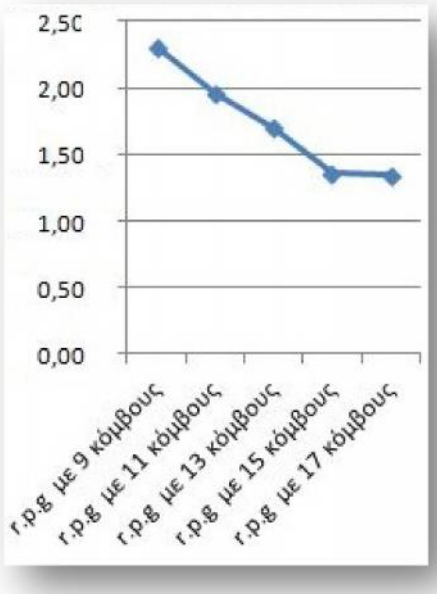
Σαρωτικές επιθέσεις	r.p.g με 15 κόμβους	r.p.g με 17 κόμβους
Επίθεση σε μία ακμή του r.p.g	1,358598908987	1,339435204822
Επίθεση σε δύο ακμές του r.p.g	0,0184579099550066	0,0179408666791655
Επίθεση σε τρεις ακμές του r.p.g	0,000250768963270523	0,000240306284350922
Επίθεση σε τέσσερις ακμές του r.p.g	0,000003406944399071 34	0,000003218746971995 92
Επίθεση σε ετικέτες κόμβων του r.p.g	0,00	0,00
Επίθεση σε s.i.p (γενικά)		
Επίθεση σε s.i.p (με έναν μονό κύκλο)		
Επίθεση σε b.p (γενικά)		
Επίεθση σε b.p (μ. αυξ μ. μειών)		
Επίθεση σε Β′	0,000000497830 00	0,000000004817 70

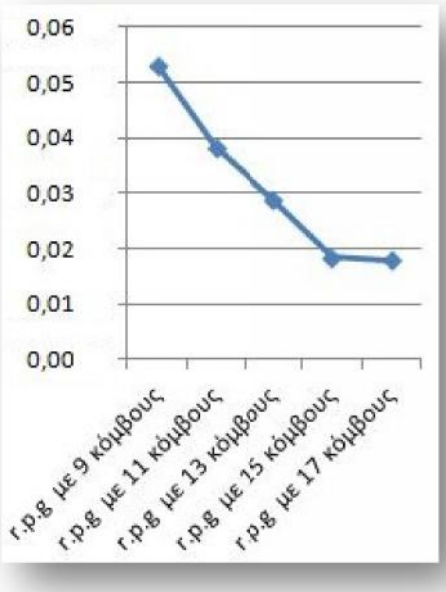
Τυχαίες επιθέσεις	r.p.g με 13 κόμβους	r.p.g με 23 κόμβους	r.p.g με 33 κόμβους
Επίθεση σε 100000 s.i.p αλλάζοντας τέσσερα στοιχεία την φορά	96,972	99,24	99,72
Τυχαίες επιθέσεις	r.p.g με 53 κόμβους	r.p.g με 63 κόμβους	r.p.g με 73 κόμβους
Επίθεση σε 100000 s.i.p αλλάζοντας τέσσερα στοιχεία την φορά	99,872	99,914	99,947
Τυχαίες επιθέσεις	r.p.g με 43 κόμβους	r.p.g με 83 κόμβους	r.p.g με 93 κόμβους
Επίθεση σε 100000 s.i.p αλλάζοντας τέσσερα στοιχεία την φορά	99,809	99,95	99,964

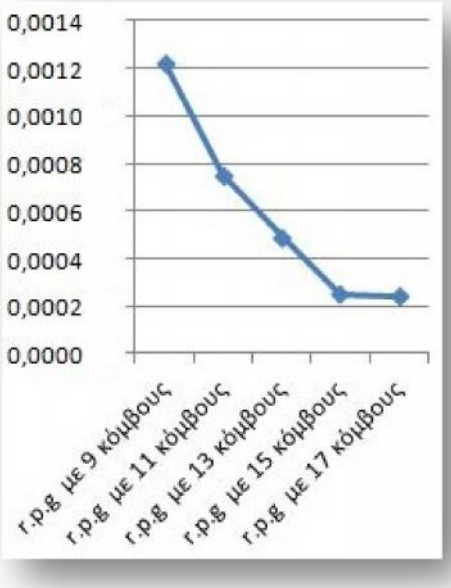
ακριβώς τρόπο (στην γενική περίπτωση της bitonic permutation ο εισβολέας γνωρίζει ότι είναι απλά μία μετάθεση, ενώ στην ειδική της περίπτωση ξέρει επιπλέον πως είναι bitonic permutation αλλά όχι πως πρέπει αναγκάστηκα στην αρχή να αυξάνεται μονοτονικά και μετά να μειώνεται μονοτονικά).Συνεχίζοντας, οι πιθανότητες στην δυαδική δομή (Β′), η οποία είναι και το χαμηλότερο υποεπίπεδο στο οποίο ο εισβολέας μπορεί να διαπράξει κάποια επίθεση, εξάγονται με την προϋπόθεση πως ο εισβολέας δεν γνωρίζει την εσωτερική δομή της, την βλέπει δηλαδή ως μία απλή μετάθεση. Τέλος, οι πιθανότητες οι οποίες αναφέρονται σε 100000 επιθέσεις, προκύπτουν από την δημιουργία 100000 αυτοαναστρέφουσων ακολουθιών με τυχαίο τρόπο, στις οποίες σε έχουν αλλάξει θέση τέσσερα στοιχεία σε κάθε μία από αυτές.

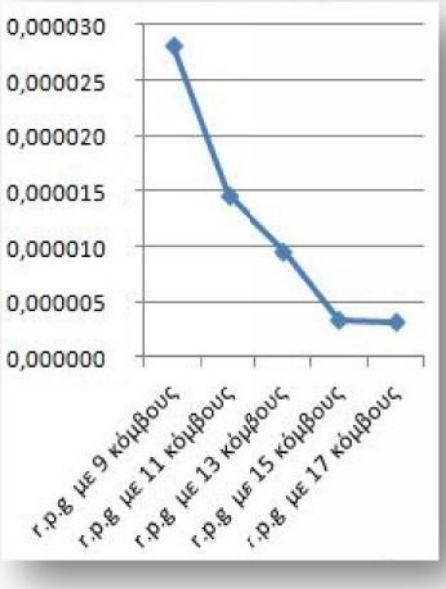
- **Γραφικές παραστάσεις πιθανοτήτων αναφερόμενων σε πιθανότητες επιτυχίας του εισβολέα ύστερα από επίθεση στις ακμές του υδατογραφήματος**

Στις επόμενες τέσσερις εικόνες εμφανίζονται οι γραφικές παραστάσεις οι οποίες απεικονίζουν τις πιθανότητες που έχει κάποιος εισβολέας να επιτεθεί σε ένα μεταθετικό αναγώγιμο γράφημα μετακινώντας στην πρώτη εικόνα μία ακμή, στην δεύτερη δύο ακμές, στην τρίτη τρεις ακμές και στην τέταρτη τέσσερις, έτσι ώστε να μην γίνει αντιληπτός από το πρόγραμμα το οποίο εκτελεί την αντίστροφη διαδικασία υδατογράφησης στο επίπεδο του γραφήματος. Σημειώνεται επίσης πως ο εισβολέας στις επιθέσεις αυτές γνωρίζει μόνο πως το υδατογράφημα αυτό είναι μονάχα ένα απλό γράφημα.



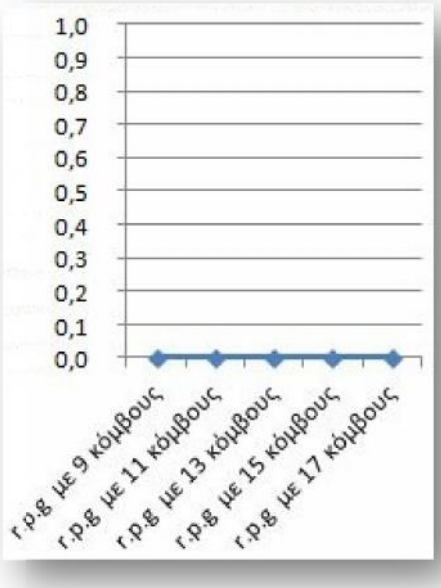






- **Γραφική παράσταση πιθανοτήτων αναφερόμενη στην επιτυχία του εισβολέα ύστερα από επίθεση στις ετικέτες των κόμβων του υδατογραφήματος**

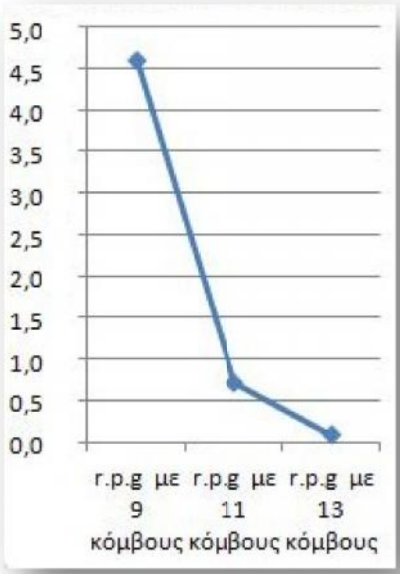
Στην επόμενη εικόνα εμφανίζεται η γραφική παράσταση η οποία είναι όλη πάνω στον οριζόντιο άξονα (όλες οι πιθανότητες είναι μηδενικές). Η γραφική παράσταση αυτή σχετίζεται με την πιθανότητα που έχει κάποιος εισβολέας να επιτεθεί σε ένα μεταθετικό αναγώγιμο γράφημα στο υποεπίπεδο των ετικετών των κόμβων και η επίθεσή του να μην γίνει αντιληπτή στο υποεπίπεδο αυτό. Σημειώνεται επίσης πως ο εισβολέας στις επιθέσεις αυτές γνωρίζει μόνο πως το υδατογράφημα αυτό είναι μονάχα ένα απλό γράφημα, όπως ακριβώς και στις επιθέσεις στις ακμές.

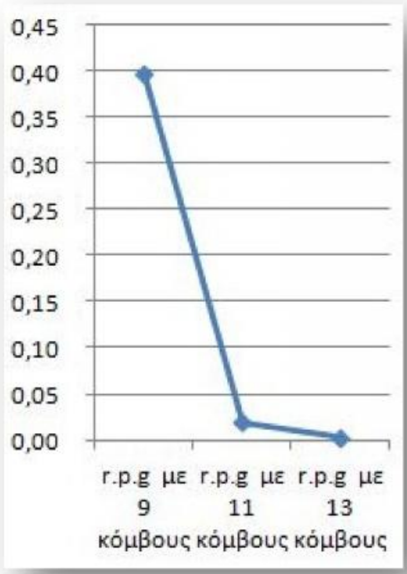


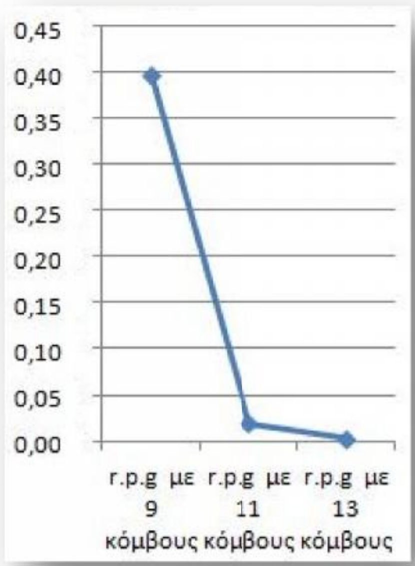
- **Γραφικές παραστάσεις αναφερόμενες σε πιθανότητες επιτυχίας του εισβολέα ύστερα από επιθέσεις σε αυτοαναστρέφουσα μετάθεση και bitonic permutation**

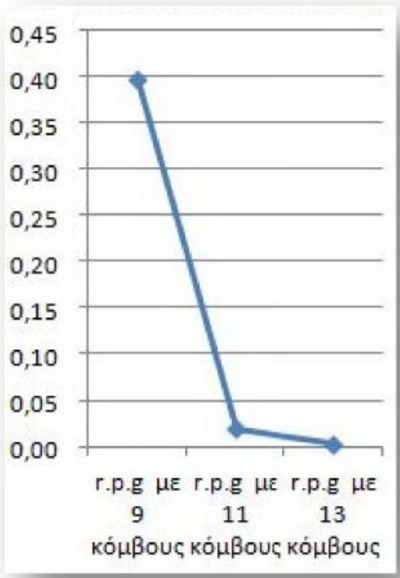
Συνεχίζοντας οι παρακάτω τέσσερις εικόνες σχετίζονται με τις πιθανότητες που έχει κάποιος κακόβουλος χρήστης να επιτεθεί στα κρυφά υποεπίπεδα του γραφήματος και η επίθεσή του να μην γίνει αντιληπτή στο υποεπίπεδο της αυτοαναστρέφουσας μετάθεσης. Πιο συγκεκριμένα η πρώτη εικόνα απεικονίζει την πιθανότητα ο εισβολέας να επιτεθεί στο υποεπίπεδο της μετάθεσης δίχως να γνωρίζει πως είναι γενικά αυτοαναστρέφουσα και επίθεση να μην γίνει αντιληπτή. Η δεύτερη εικόνα απεικονίζει την πιθανότητα ο εισβολέας να επιτεθεί στο υποεπίπεδο της μετάθεσης και να γνωρίζει πως είναι γενικά αυτοαναστρέφουσα, αλλά όχι την επιπλέον πληροφορία του μονού κύκλου (δηλαδή πως η μετάθεση αυτή έχει μονάχα έναν μονό κύκλο και όλους τους άλλους διπλούς). Η τρίτη απεικονίζει την πιθανότητα ο εισβολέας να επιτεθεί στο υποεπίπεδο της bitonic permutation, δίχως να γνωρίζει την πληροφορία πως η μετάθεση αυτή είναι b.p, βλέποντάς την σαν μία απλή μετάθεση. Τέλος, η τέταρτη εικόνα απεικονίζει τις πιθανότητες ο εισβολέας να επιτεθεί στην bitonic permutation γνωρίζοντας την αμέσως προηγούμενη πληροφορία, όχι όμως την

πληροφορία πως η b.p αυτή πρέπει αρχικά να αυξάνεται μονοτονικά και μετά να μειώνεται μονοτονικά.



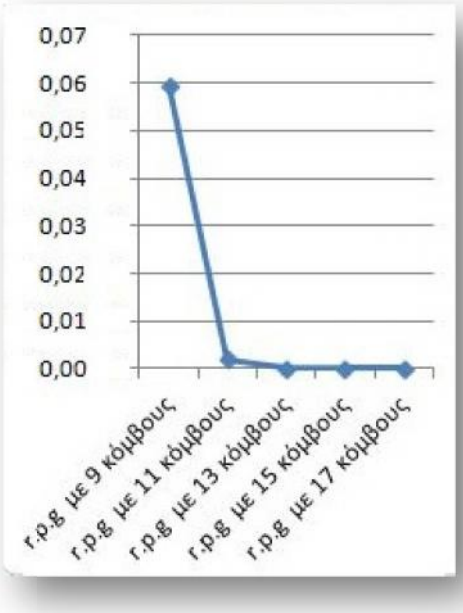






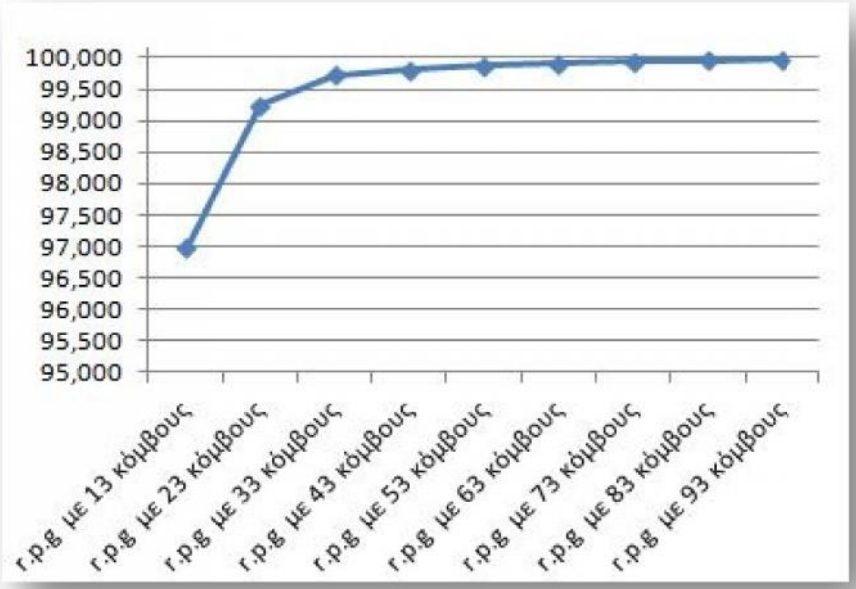
- **Γραφική παράσταση πιθανοτήτων αναφερόμενων στην επιτυχία του εισβολέα ύστερα από επίθεση στην δυαδική δομή**

Πλησιάζοντας στο τέλος με τις εικόνες των γραφικών παραστάσεων, παρακάτω υπάρχει μία εικόνα στην οποία εμφανίζονται οι πιθανότητες που έχει κάποιος εισβολέας να επιτεθεί στο μεταθετικό αναγώγιμο γράφημα στο πιο κρυφό υποεπίπεδό του και η επίθεσή του να γίνει αντιληπτή στο ίδιο υποεπίπεδο. Το υποεπίπεδο αυτό ονομάζεται δυαδική δομή (Β′) και ο κακόβουλος χρήστης δεν γνωρίζει τον τρόπο κατασκευής του (το βλέπει σαν μία μετάθεση απλή).



- **Γραφική παράσταση αναφερόμενη στην επιτυχία του νόμιμου ιδιοκτήτη στον εντοπισμό επίθεσης στις αυτοαναστρέφουσες μεταθέσεις**

Με τον τρόπο αυτό, έρχεται και η σειρά της γραφικής παράστασης στην οποία απεικονίζονται οι πιθανότητες που έχει ο νόμιμος ιδιοκτήτης του λογισμικού να εντοπίσει τις εκάστοτε επιθέσεις στο υποεπίπεδο της αυτοαναστρέφουσας μετάθεσης (δίχως να χρησιμοποιήσει την επιπλέον πληροφορία, πως οι αυτοαναστρέφουσες μεταθέσεις των υδατογραφημάτων αυτής της πτυχιακής εργασίας έχουν αναγκαστικά μονάχα έναν μονό κύκλο). Επομένως αυτό που γίνεται είναι να παραχθούν για κάθε είδος υδατογραφήματος τη φορά 100000 αυτοαναστρέφουσες μεταθέσεις να αλλαχθούν τέσσερα στοιχεία σε κάθε μία και να γίνει ο έλεγχος για το πόσες από αυτές έγιναν αντιληπτές από τον νόμιμο ιδιοκτήτη σε κάθε είδος. Σημειώνεται πως με το όρο είδος εννοείται, πόσους κόμβους έχει το υδατογράφημα κάθε φορά.



6.3 Συμπεράσματα επιθέσεων

Έχοντας ολοκληρώσει με επιτυχία την πειραματική μελέτη επιθέσεων πάνω στα υδατογραφήματα με τα οποία σχετίζεται η παρούσα πτυχιακή εργασία σε όλα τους τα επίπεδα θα ήταν ωφέλιμο να αναφερθούν και τα ανάλογα συμπεράσματα.

Το πρώτο και κυριότερο συμπέρασμα που εξάγεται είναι πως οι εκάστοτε εισβολείς σε περίπτωση που επιθυμούν να υποκλέψουν ένα λογισμικό υδατογραφημένο με το υδατογράφημα της παρούσας πτυχιακής εργασίας πρέπει να εργαστούν με τον εξής τρόπο. Αρχικά είναι υποχρεωμένοι να το εντοπίσουν στο εκτελέσιμο του λογισμικού (καθώς μόνο αυτό έχουν στα χέρια τους), στη συνέχεια να το εξάγουν με επιτυχία δίχως δηλαδή να το αλλοιώσουν ούτε αυτό ούτε το λογισμικό και τέλος να επιτεθούν στο εν λόγω υδατογράφημα. Σε αντίθετη περίπτωση μπορούν απλά να υποκλέψουν το λογισμικό κινδυνεύοντας ανά πάσα στιγμή να πάνε στα δικαστήρια για αυτή τους την κίνηση από τον εκάστοτε νόμιμο ιδιόκτητη. Πάρα ταύτα, οι εισβολείς γενικά έχουν πολύ μικρές πιθανότητες η κίνησή τους αυτή να μην γίνει εν τέλει αντιληπτή είτε, από τον ίδιο τον ιδιοκτήτη του λογισμικού είτε, από το πρόγραμμα που χρησιμοποιεί αυτός προκειμένου να αποδείξει την γνησιότητα του λογισμικού προστατεύοντας ταυτόχρονα τα πνευματικά του δικαιώματα. Πιο συγκεκριμένα, σύμφωνα με το προηγούμενο υποκεφάλαιο, ο εισβολέας έχει πάντα πιθανότητα κάτω από 10%, οποιοδήποτε είδος επίθεσης και να διαπράξει και σε οποιοδήποτε υδατογράφημα ανεξαρτήτου μεγέθους άνω των εννέα κόμβων.

Εν συνεχεία, όπως γίνεται αντιληπτό βλέποντας τις γραφικές παραστάσεις αλλά και τον πινάκα με όλες τις πιθανότητες, εξάγεται αμέσως το συμπέρασμα πως καθώς το πλήθος των κόμβων στα μεταθετικά αναγώγιμα γραφήματα αυξάνεται, η πιθανότητα ο εισβολέας να επιτεθεί σε αυτά και να μην γίνει αντιληπτός πλησιάζει όλο και περισσότερο στο μηδέν καθώς το πλήθος των κινήσεων που έχει να κάνει σε κάθε του επίθεση αυξάνεται παραγοντικά. Από την άλλη πλευρά όμως, όσο μεγαλύτερο είναι ένα υδατογράφημα, τόσο περισσότερες πιθανότητες έχει ο εισβολέας να το εντοπίσει μέσα στο λογισμικό. Οπότε είναι στο χέρι του δημιουργού του υδατογραφήματος να επιλέξει το μέγεθος του υδατογραφήματος που θα ενσωματώσει στο εκάστοτε λογισμικό, λαμβάνοντας υπόψη κάθε φορά το μέγεθος του λογισμικού που θα επρόκειτο να υδατογραφήσει, το μέγεθος προστασίας που επιθυμεί να

δώσει στο λογισμικό καθώς και διάφορες άλλες υποθέσεις οι οποίες εξαρτώνται από ποικίλους φορείς.

Τέλος, αξίζει να σημειωθεί πως ως υπόθεση στο πειράματα που έλαβαν χώρα στο προηγούμενο υποκεφάλαιο, είναι πως ο Ιωάννης προκειμένου να επιτεθεί στο χαμηλό επίπεδο, το οποίο εμπεριέχει τα υποεπίπεδα της αυτοαναστρέφουσας μετάθεσης, της bitonic permutation και της δυαδικής δομής Β′ πρέπει να γνωρίζει με κάθε λεπτομέρεια τον τρόπο λειτουργίας του υψηλού επιπέδου. Το γεγονός αυτό είναι απολύτως φυσιολογικό καθώς δεν δύναται ο κακόβουλος χρήστης να διαπράξει επίθεση, για παράδειγμα στην αυτοαναστρέφουσα μετάθεση, δίχως να γνωρίζει τις ιδιότητες του μεταθετικού αναγώγιμου γραφήματος καθώς και τον αλγόριθμο μεταφοράς από το γράφημα αυτό στην μετάθεση αυτή. Επομένως γίνεται φανερό πως ο εκάστοτε εισβολέας πρέπει να γνωρίζει επακριβώς όλους τους αλγορίθμους (και τους τέσσερις) δημιουργίας υδατογραφήματος προκειμένου να επιτεθεί στο χαμηλό επίπεδο. Με αποτέλεσμα το υδατογράφημα αυτού του είδους να καθίστανται μία πολύ καλή λύση σε υδατογραφήσεις λογισμικού προστατεύοντας τα πνευματικά δικαιώματα των εκάστοτε ιδιοκτητών με μία αποτελεσματικότατη και ταυτόχρονα έξυπνη μέθοδο.

7

Συμπεράσματα και Επεκτάσεις

7.1 Συμπεράσματα από την παρούσα πτυχιακή εργασία

Τα συμπεράσματα που αναδύονται από αυτήν την πτυχιακή εργασία είναι πολλά και διαφόρου είδους καθώς λαμβάνουν χώρα αντικείμενα από διάφορους τομείς.

Ένα πρώτο συμπέρασμα είναι πως δεν αρκεί η δημιουργία ενός πολύ καλού λογισμικού (μικρό σε έκταση, γρήγορο, βελτιστοποιημένο, φιλικό στον χρήστη) για να κερδίσει την αγορά, αλλά χρειάζεται και την απαραίτητη θωράκιση. Η θωράκιση αυτή, λαμβάνει χωρά εξαιτίας των εκάστοτε κακόβουλων χρηστών που επιδιώκουν να επωφεληθούν κάποιο κέρδος με το να υποκλέψουν κάποιο λογισμικό και στην ουσία είναι η προστασία του λογισμικού από δύο πλευρές. Η μία πλευρά είναι η υδατογράφησή του με κάποια έξυπνη και εύρωστη μέθοδο η οποία αντιστέκεται όσο το δυνατόν περισσότερο σε επιθέσεις εισβολέων. Αυτή είναι χρήσιμη όταν οι εισβολείς μπορούν να επέμβουν στο εκτελέσιμο ενός λογισμικού αλλάζοντάς το και η οποία είναι και το βασικό αντικείμενο αυτής της πτυχιακής εργασίας, ενώ η δεύτερη είναι το κλείδωμα του λογισμικού από κάθε ‘οπτική γωνία’, όπως είναι για παράδειγμα όλα τα εμπορικά λογισμικά στα οποία δεν είναι δυνατόν να δει κάποιος τον πηγαίο κώδικά τους.

Πάρα ταύτα όμως, αξίζει να σημειωθεί πως η θωράκιση των λογισμικών δεν συνεπάγεται και σε παρεμπόδιση της κλοπής τους, καθώς ως επί το πλείστον υπάρχει πάντα κάποιο μικρό ‘παραθυράκι’ που βοηθά τους εκάστοτε εισβολείς να δράσουν. Το ζήτημα είναι με τις θωρακίσεις που λαμβάνουν χώρα στα λογισμικά, να αποθαρρυνθούν οι εκάστοτε εισβολείς και το παραθυράκι αυτό να μικρύνει τόσο πολύ, έτσι ώστε οι υποκλοπές να ελαττωθούν στο ελάχιστο.

Συνεχίζοντας την αναφορά των συμπερασμάτων, αξίζει να σημειωθεί πως είναι πολύ πιθανόν να δημιουργηθεί κάποιο λογισμικό, εν συνεχεία να υδατογραφηθεί με κάποια μέθοδο και κάποιος εισβολέας να καταφέρει να το υποκλέψει. Τότε, όταν αυτή η υποκλοπή γίνει αντιληπτή στους ιδιοκτήτες του λογισμικού, η υπόθεση καταλήγει στα δικαστήρια και αμφότερες οι δύο πλευρές καλούνται να εξάγουν το υδατογράφημα από το λογισμικό (προφανώς από την εκτελέσιμη μορφή του). Αν όμως ο εν λόγω εισβολέας έχει καταφέρει και

έχει εντοπίσει το υδατογράφημα αλλάζοντάς το τότε υπάρχουν τρεις πιθανές εκδοχές. Είτε με την αλλαγή αυτή το πρόγραμμα του αληθινού ιδιοκτήτη που υλοποιεί την αντίστροφη διαδικασία υδατογράφησης (απόδειξη γνησιότητας λογισμικού) μπορεί να εντοπίσει την επίθεση και να την διορθώσει, είτε μπορεί να εντοπίσει την επίθεση και να μην μπορεί να την διορθώσει λόγω ανεπανόρθωτης αλλαγής, είτε ακόμα χειρότερα η επίθεση να περάσει απαρατήρητη και ο αληθινός ιδιοκτήτης να μην είναι σε θέση να υποστηρίξει την γνησιότητα του λογισμικού του. Με αποτέλεσμα ο ιδιοκτήτης είτε να δικαιωθεί αυτομάτως αν ισχύει η πρώτη εκδοχή και να είναι στην ευχάριστη θέση να ζητήσει κάποιου είδους αποζημίωση, είτε να σταματήσει την διαδικασία απόδειξης στο σημείο επίθεσης εν συνεχεία να την αναφέρει και τέλος να χρειαστεί προσκομίσει περεταίρω αποδεικτικά στοιχεία, είτε να φύγει άπραγος καθώς η προστασία που είχε το λογισμικό του αποδείχτηκε εύθραυστη.

Επιπλέον, καθίσταται προφανές πως δεν υπάρχει καμία μαγική συνταγή υδατογράφησης λογισμικού που χρησιμοποιεί κάποιο τέλειο υδατογράφημα και προστατεύει όλα τα λογισμικά, αντιθέτως κάθε λογισμικό είναι ξεχωριστό και πρέπει να αντιμετωπίζεται ως μοναδική οντότητα. Η οντότητα αυτή προφανώς δέχεται αποτελεσματικά μόνο ορισμένα υδατογραφήματα ανάλογα με το είδος της, την έκτασή της και διάφορα άλλα χαρακτηριστικά που αποτελούν την ταυτότητά της, αφού βέβαια προηγηθεί η διεξοδική μελέτη της. Αυτό συμβαίνει διότι, προκειμένου να υδατογραφηθεί κάποιο λογισμικό, πρέπει πρώτα να κατανοηθεί πλήρως από αυτόν που το θωρακίζει και μετά να ληφθούν οι απαραίτητες επεμβάσεις σε αυτό.

Άλλο ένα συμπέρασμα είναι, πως υπάρχουν διάφορα στάδια στα οποία μπορεί να ενσωματωθεί ένα υδατογράφημα σε κάποιο λογισμικό με τις απαραίτητες τροποποιήσεις βέβαια, όπως είναι για παράδειγμα ο πηγαίος κώδικας (υψηλότερο επίπεδο), ή ο κώδικας Assembly σε προγράμματα γραμμένα σε C (ένα επίπεδο πάνω από το χαμηλότερο, το οποίο είναι η δυαδική μορφή), ή το bytecode που δημιουργείται μόνο σε Java προγράμματα και είναι επίσης ένα επίπεδο πάνω από το χαμηλότερο, ή ακόμα και στο χαμηλότερο επίπεδο, το οποίο αποτελεί την πλήρως κατανοητή μορφή από τον υπολογιστή και εμπεριέχει αποκλειστικά μηδενικά και άσσους. Σημειώνεται επίσης πως μετά το πέρας της υδατογράφησης, ανάλογα σε ποιο στάδιο έγινε η ενσωμάτωση υπάρχουν ειδικά προγράμματα ή ακόμα και εντολές, όπου κάνουν την μεταφορά από εκείνο το σημείο εισαγωγής στον πηγαίο κώδικα. Επομένως αφού γίνει η μεταφορά αυτή, το καινούριο και υδατογραφημένο πλέον λογισμικό επαναμεταγλωττίζεται με την ίδια ακριβώς διαδικασία προκειμένου να δημιουργηθεί το υδατογραφημένο εκτελέσιμο.

Αξίζει να σημειωθεί στο σημείο αυτό, πως από το κεφάλαιο έξι το οποίο αναφέρεται στην πειραματική μελέτη επιθέσεων, βγαίνει το συμπέρασμα πως αν κάποιος κακόβουλος χρήστης έχει εντοπίσει με επιτυχία το υδατογράφημα και το έχει εξάγει από το σημείο ή τα σημεία που βρίσκονταν και εν συνεχεία επέμβει σε αυτό οι πιθανότητες που έχει να μην γίνει εν τέλει αντιληπτός από τον νόμιμο ιδιοκτήτη είναι ελάχιστες ακόμα και αν γνωρίζει κομβικά σημεία των αλγορίθμων υδατογράφησης. Στην περίπτωση βέβαια που το μόνο που έχει στα χέρια του ο εισβολέας είναι μονάχα το υδατογράφημα και καμία άλλη πληροφορία, τότε οι πιθανότητές του να καταφέρει και να υποκλέψει το λογισμικό χωρία να γίνει αντιληπτός τείνουν να μηδενιστούν καθώς το πλήθος των κόμβων αυξάνεται στα εκάστοτε υδατογραφήματα.

Τέλος, σημειώνεται πως όσο πιο ευέλικτο είναι ένα υδατογράφημα τόσο δυσκολότερο καθίσταται το έργο των κακόβουλων χρηστών να το εξάγουν από το λογισμικό,

με αποτέλεσμα οι δυναμικές υδατογραφήσεις να έχουν ένα αξιοσημείωτο προβάδισμα εξέλιξης από τις στατικές.

7.2 Επεκτάσεις

Οι επεκτάσεις που θα μπορούσαν να λάβουν χώρα στην παρούσα πτυχιακή εργασία είναι ποικίλες και αφορούν ξεχωριστά τμήματα στην όλη διαδικασία υδατογράφησης λογισμικού. Τα τμήματα αυτά αφορούν πρώτον την διαδικασία παραγωγής ενός υδατογραφήματος από έναν φυσικό αριθμό σε ένα μεταθετικό αναγώγιμο γράφημα και δεύτερον την αντίστροφη από αυτή διαδικασία, τρίτον τις επιθέσεις που δέχεται το υδατογράφημα από τους εκάστοτε εισβολείς εφόσον εντοπιστεί καθώς επίσης και τον τρόπο με τον οποίο αυτές αντιμετωπίζονται και τέταρτον και τελευταίο την μέθοδο υδατογράφησης στην οποία διεξάγεται τόσο η ενσωμάτωση όσο και η εξαγωγή του υδατογραφήματος.

Πιο συγκεκριμένα, οι πρώτες δύο επεκτάσεις σχετίζονται με την δημιουργία ενός ακόμα πιο ισχυρού υδατογραφήματος λογισμικού το οποίο με την ενσωμάτωσή του θα θωρακίζει σε ακόμα μεγαλύτερο ποσοστό τα εκάστοτε λογισμικά έναντι των κακόβουλων χρηστών. Για παράδειγμα, μία ισχυρή επέκταση του μεταθετικού αναγώγιμου γραφήματος θα μπορούσε να είναι η ενσωμάτωσή του μέσα σε ένα πιο γενικό γράφημα το οποίο και θα επεκτείνεται σε ολόκληρο το λογισμικό και δεν θα καταλαμβάνει συγκεκριμένα σημεία του όπως τώρα, με την αναγκαία προϋπόθεση όμως πως ο χρόνος εκτέλεσης του υδατογραφημένου προγράμματος να παραμένει στα αρχικά επίπεδα. Αυτό το γενικό γράφημα, θα μπορούσε να είναι ένα μεγαλύτερο από το αρχικό μεταθέσιμο αναγώγιμο γράφημα, το οποίο θα προσδιορίζεται από επιπλέον ιδιότητες, προκειμένου να είναι και αυτό εξίσου δυνατό και ανθεκτικό σε επιθέσεις όπως το αρχικό αλλά και ακόμα περισσότερο. Από την άλλη πλευρά όμως θα μπορούσε να διατηρηθεί το εν λόγο γράφημα προσθέτοντάς του κάποιες επιπλέον ιδιότητες, κάνοντας το ακόμη πιο αποδοτικό και βέλτιστο.

Συνεχίζοντας, με την τρίτη κατά σειρά επέκταση, έρχεται η σειρά των επιθέσεων οι οποίες θα μπορούσαν να λάβουν χώρα ταυτόχρονα σε περισσότερα του ενός τμήματα του μεταθετικού αγώγιμου γραφήματος είτε αυτά είναι ορατά απευθείας με το μάτι (ετικέτες κόμβων, κόμβοι, ακμές), είτε όχι (αυτοαναστρέφουσα μετάθεση, bitonic permutation, δυαδική δομή). Επίσης οι επιδιορθώσεις υδατογραφημάτων, στα υδατογραφήματα όπου έχει εντοπιστεί κάποιου είδους επίθεση θα μπορούσαν να γίνονται σε μεγαλύτερο εύρος επιθέσεων, με εξίσου σωστά αποτελέσματα.

Τέλος, μία άλλη επέκταση εξίσου σημαντική και ενδιαφέρον με τις προηγούμενες θα μπορούσε να είναι η τροποποίηση της μεθόδου υδατογράφησης λογισμικού με τέτοιον τρόπο έτσι ώστε το εκάστοτε παραγόμενο υδατογράφημα να γίνει περισσότερο δυνατό και παράλληλα περισσότερο κρυφό από το αρχικό. Αυτό θα μπορούσε να πραγματοποιηθεί, είτε με εισαγωγή στις εκάστοτε συναρτήσεις (κόμβους του μεταθετικού αναγώγιμου γραφήματος) δεδομένων καλύτερων των ως τώρα, είτε με την τροποποίηση της διαδικασίας ενσωμάτωσης τοποθετώντας το υδατογράφημα σε διαφορετικό στάδιο μεταγλώττισης ή αλλάζοντας τον τρόπο επιλογής των σημείων προς ενσωμάτωση στο διάγραμμα ελέγχου ροής. Σημειώνεται επίσης πως με τον όρο καλύτερα δεδομένα εννοείται, δεδομένα τα οποία δεν δίνουν στόχο σε προγράμματα βελτιστοποίησης και δεν αυξάνουν τον χρόνο εκτέλεσης του αρχικού προγράμματος σε σημείο επικίνδυνο, τόσο για τον ιδιοκτήτη, όσο και για τον εισβολέα.

Βιβλιογραφία

[1] A. Larkin, F. Balado, N. Hurley and G. Silvestre, "Dither modulation watermarking of dynamic memory traces", *Lecture Notes in Computer Science, Information Hiding 3727 (NA), 372-386, 2005.*

[2] A. Monden, H. Iida, K. Matsumoto, K. Inoue, and K. Torii, "A practical method for watermarking Java programs", *24th Computer Software and Applications Conference (COMPSAC'00), 191–197, 2000.*

[3] B. Anckaert, B. Sutter, and K. Bosschere, "Steganography for executables and code transformation signatures", *7th International Conference on Information Security and Cryptology, 7-10, 2005.*

[4] B. Anckaert, B. Sutter, and K. Bosschere, "Cover communication through executables", *Manuscript, 2004*.

[5] B. Feng, M. Zhang, G. Xu, X. Niu, and Z. Hu, "Dynamic data flow graph based software watermarking", *IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC'09), 1029-1033, 2009.*

[6] B. Lu, F. Liu, X. Ge, and P. Wang, "Feature n - gram set based software zero watermarking", *International Symposiums on Information Processing (ISIP'08), 607-611, 2008.*

[7] C. Collberg, A. Huntwork, E. Carter, and G. Townsend, "Graph theoretic software watermarks: implementation, analysis and attacks", *6th Workshop on Information Hiding (IH'04), 192-207, 2004.*

[8] C. Collberg, and C. Thomborson, "Watermarking, tamper proofing, and obfuscation tools for software protection", *IEEE Transactions on Software Engineering, 735-746, 2004.*

[9] C. Collberg, and C. Thomborson, "Software watermarking: models and dynamic embedding", *26th ACM SIGPLAN-SIGACT on Principles of Programming Languages (POPL'99), 311–324, 1999.*

[10] C. Collberg, and C. Thomborson, "On the limits of software watermarking", *Department of Computer Science, University of Auckland, Technical Report No 164, 1998.*

[11] C. Collberg, and G. Myles, "Software watermarking via opaque predicates: implementation, analysis and attacks", *Electronic Commerce Research 6, 155–171, 2006.*

[12] C. Collberg, and J. Nagra, "Surreptitious Software: obfuscation, watermarking and tamperproofing for software protection", *Book: Addison-Wesley, 2009.*

[13] C. Collberg, and M. Myles, "Software watermarking through register allocation: implementation, analysis, and attacks", *Journal Electronic Commerce Research archive Volume 6 Issue 2, 274-293, 2003.*

[14] C. Collberg, C. Thomborson, and G. Townsend, "Dynamic graph based software fingerprinting", *ACM Transactions on Programming Languages and Systems 29, 35:1-67, 2007.*

[15] C. Collberg, C. Thomborson, and L. Douglas, "Manufacturing cheap resilient and stealthy opaque constructors", *In Principles of Programming Languages (POPL'98), 184–196, 1998.*

[16] C. Collberg, C. Tomborson, S. Kouborov, and E. Carter, "Error correcting graphs for software watermarking", *29th Workshop on Graphs in Computer Science (WG'03), LNCS 2880, 156–167, 2003.*

[17] C. Collberg, E. Carter, S. Debray, A. Huntwork, J. Kececioglu, C. Linn and M. Stepp, "Dynamic path based software watermarking", *ACM SIGPLAN Conference on Programming Language Design and Implementation, ACM SIG PLAN 39, 107-118, 2004.*

[18] C. Collberg, G. Myles, Z. Heidepriem, and A. Navabi, "The evaluation of two software watermarking algorithms", *Journal Software—Practice & Experience archive Volume 35 Issue 10, 923-938, 2005.*

[19] C. Collberg, J. Nagra, and C. Thomborson, "A functional taxonomy for software watermarking", *25th Australasian conference on Computer science (ACSC'02), Volume 4, 177-186, 2002.*

[20] C. Colleberg, and T. Sahoo, "Software watermarking in the frequency of domain: implementation, analysis and attacks", *Journal of Computer security, 2004.*

[21] C. Thomborson, J. Nagra, R. Somaraju, and C. He, "Tamper proofing software watermarks", *2th workshop on Australasian information security, Data Mining and Web Intelligence, and Software Internationalisation (ACSW Frontiers'04), Volume 32, 27-36, 2004.*

[22] C. Zhang, H. Peng, X. Long, Z. Pan, and Y. Wu, "A fragile software watermarking for tamper proof", *5th International Conference on Information Assurance and Security (IAS'09), 309-312, 2009.*

[23] C. Zhang, J. Wang, C. Thomborson, C. Wang, and C. Collberg, "A semi dynamic multiple watermarking scheme for Java applications", *9th ACM workshop on Digital rights management ACM New York* (*DRM'09), 144-151, 2009.*

[24] D. Curran, J. Hurley, and M. Cinneide, "Securing Java through software watermarking", *Int'l Conference on Principles and Practice of Programming in Java (PPPJ'07), 145–148, 2003.*

[25] D. Gong, F. Liu, B. Lu, P. Wang, and L. Ding, "Hiding information on Java class file", *International Symposium on Computer Science and Computational Technology (ISCSCT'08), 160-164, 2008.*

[26] F. Liu, B. Lu, and X. Luo, "A Chaos-Based Robust Software Watermarking", *(ISPEC'06), 355-366, 2006.*

[27] F. Wang, J. Zhao, and A. Saddik, "Copyright protection of J2EE web applications through watermarking", *Canadian Conference on Electrical and Computer Engineering, 1782-1785, 2005.*

[28] G. Arboit, "A method for watermarking Java programs via opaque predicates", *5th International Conference on Electronic Commerce Research (ICECR-5'02), 2002.*

[29] G. Grover, "The protection of computer software its technology and applications", *Cambridge University Press, New York, 263, 1992.*

[30] G. Gupta, and J. Pieprzyk, "Source code watermarking based on function dependency oriented sequencing", *International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIHMSP'08), 965-968, 2008.*

[31] G. Gupta, and J. Pieprzyk, "Software watermarking resilient to debugging attacks", *Journal of Multimedia, Vol 2, No 2, 10-16, 2007.*

[32] G. Gupta, and J. Pieprzyk, "A low cost attack on branch based on software watermarking schemes", *(IWDW'06), 282-293, 2006*

[33] G. Myles, and H. Jin, "Self validating branch based software watermarking", *Information Hiding, 3727, 342-356, 2005.*

[34] H. Lee, and K. Kaneko, "Two new algorithms for software watermarking by register allocation and their empirical evaluation", *6th International Conference on Information Technology: New Generations (ITNG'09), 217-222, 2009.*

[35] H. Lee, and K. Kaneko, "New approaches for software watermarking by register allocation", *9th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'08), 63-68, 2008.*

[36] J. Ai, X. Sun, Y. Liu, I. Cox, G. Sun, and Y. Luo, "A stern based collusion secure software watermarking algorithm and its implementation", *International Conference on Multimedia and Ubiquitous Engineering (MUE'07), 813-818, 2007.*

[37] J. Nagra, and C. Thomborson 2004, "Threading software watermarking", *6th Int'l Workshop on Information Hiding (IH'04), LNCS 3200, 208-223, 2004.*

[38] J. Palsberg, S. Krishnaswamy, M. Kwon, D. Ma, G. Shao, and Y. Zhang, "Experience with software watermarking", *16th Annual Conference on Computer Security Applications (ACSAC '00) , 308-316, 2000.*

[39] J. Stern, G. Hachez, F. Koeune, J. Quisquarter, Ucl Crypto Group, B. Maxwell, and P. Levant, "Robust object watermarking: application to code", *3rd Int'l Workshop on Information Hiding (IH'99), LNCS 1768, 368–378, 1999.*

[40] J. Zhu, K. Yin, and Y. Liu, "A novel dgw scheme based on 2d_ppct and permutation", *International Conference on Multimedia Information Networking and Security (MINES'09), 109-113, 2009.*

[41] J. Zhu, Y. Liu, and K. Yin, "A novel dynamic graph software watermark scheme", *1st International Workshop on Education Technology and Computer Science (ETCS'09), Volume 03, 775-780, 2009.*

[42] J.L. Wong, Q. Gang, and P. Miodrag, "Optimization intensive watermarking techniques for decision problem", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 119-127, 1999.*

[43] K. Fukushima, and K. Sakurai, "A software fingerprinting scheme for Java using classfiles obfuscation", *(WISA'03), 303-316, 2003.*
[44] K. Hattanda, N. Ichikawa, S. Ichikawa, "The evaluation of Davidson's digital signature scheme", *Manuscript, 2004*.
[45] L. Wang 2006, "A constant encoding algorithm which is tamper proofs the CT watermark", *Master thesis, University of Auckland.*
[46] M. Amarnath, and Y. Srikant, "SWuS: software watermarking using slices", *IISc-CSA-TR-2004-7, 2004.*
[47] M. Chroni, and S. Nikolopoulos, "Encoding numbers as reducible permutation graphs for software watermarking", *3rd Int'l Conference on Software, Services & Semantic Technologies (S3T'11), poster, 2011.*
[48] M. Madou, B. Anckaert, B. Sutter, and K. Bosschere, "Hybrid static dynamic attacks against software protection mechanisms", *5th ACM workshop on Digital rights management (DRM'05) ACM New York, 75-82, 2005.*
[49] M. Preda, M. Madou, K. Bosschere, and R. Giacobazzi, "Opaque predicates detection by abstract interpretation", *Book: Springer Berlin, 2006.*
[50] M. Preda, R. Giacobazzi, and E. Visentini, "Hiding software watermarks in loop structures", *15th International Static Analysis Symposium (SAS'08), Volume 5079 of Lecture Notes in Computer Science, 174-188, 2008.*
[51] M. Shirali-Shahreza, and S. Shirali-Shahreza, "Software watermarking by equation reordering", *3rd International Conference on Information and Communication Technologies: From Theory to Applications (ICTTA'08), 1-4, 2008.*
[52] M. Sikandar, H. Khiyal, A. Khan, S. Amjad, and M. Khalil, "Evaluating effectiveness of tamper proofing on dynamic graph software watermarks", *International Journal of Computer Science and Information Security (IJCSIS'09), Vol. 6, No. 3, 57-63, 2009.*
[53] N. Jasvir, "Threading software watermark", *Master thesis, University of Auckland, 2007.*
[54] O. Esparza, M. Fernadez, M. Soriano, J. Munoz, and J. Forne, "Mobile agent watermarking and fingerprinting: tracing malicious hosts", *Database and Expert Systems Applications (DEXA'03), Lecture Notes in Computer Science, 927-936, 2003.*
[55] P. Cousot, and R. Cousot, "An abstract interpretation based framework for software watermarking", *31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'04), 173–185, 2004.*
[56] P. Samson, "Apparatus and method for serializing and validating copies of computer software", *US Patent 5.287.408, 1994.*
[57] Q. Gang, and P. Miodrag, "Hiding signatures in graph coloring solutions", *3th International Workshop on Information Hiding (IH'99), 348-367, 1999.*
[58] Q. Gang, and P. Miodrag, "Analysis of watermarking techniques for graph coloring problem", *IEEE/ACM Int'l Conference on Computer-aided Design (ICCAD '98), ACM Press, 190–193, 1998.*
[59] R. Davidson, and N. Myhrvold, "Method and system for generating and auditing a signature for a computer program", *US Patent 5.559.884, Microsoft Corporation, 1994.*
[60] R. Giacobazzi, "Hiding information in completeness holes", *6th IEEE International Conference on Software engineering and Formal Methods (SEFM'08), IEEE Press, 7-20, 2008.*
[61] R. Giacobazzi, "Hiding information in complete holes: New perspective in code obfuscation and watermarking", *6th IEEE International Conference on Software Engineering and Formal Methods (SEFM'08), 7-18, 2008.*
[62] R. Giacobazzi, M. Preda, and E. Visentini, "Hiding software watermarks in loop structures", *15th international symposium on Static Analysis (SAS'08), 174-188, 2008.*
[63] R. Khalil, and A. Keromytis, "Hydan: hiding information in program binaries", *International Conference on Information and Communications Security (ICICS'04*), 187-199, 2004.
[64] R. Tu, F. Wang, J. Zhao, and A. Saddik, "Copyright protection of web applications through watermarking", *1st International Conference on Innovative Computing Information and Control (ICICIC'06), 78-82, 2006.*
[65] R. Venkatesan, V. Vazirani and S. Sihna, "A graph theoretic approach to software watermarking", *4th Int'l Workshop on Information Hiding (IH'01), LNCS 2137, 157–168, 2001.*

[66] S. Moscowitz, "Method for stegachiper protection of computer code", *US Patent 5.745.569, 1996.*
[67] S. Thacker, "Software watermarking via Assembly code transformations", *Master thesis, University of San Jose State, 2004.*
[68] V. Len, and Y. Desmedt, "Cryptanalysis of UCLA watermarking schemes for intellectual property protection", *Revised Papers from the 5th International Workshop on Information Hiding (IH'02), 213-225, 2002.*
[69] V. Ramarathnam, and V. Vijay, "Technique for producing through watermarking highly tamper-resistant executable code and resulting "watermarked" code so formed", *US Patent 6829710, 2006.*
[70] V. Yarmolik, and S. Partsianka, "Software ip protection based on watermarking techniques", *Springer US, 227-234, 2005.*
[71] W. Zhu, "Concepts and techniques in software watermarking and obfuscation", *(PhD--Computer Science) Thesis, University of Auckland, 2007.*
[72] W. Zhu, "Informed recognition in software watermarking", *Pacific Asia conference on Intelligence and security informatics* (*PAISI'07), 257-261, 2007.*
[73] W. Zhu, and C. Thomborson, "Algorithms to watermark software through register allocation", *(DRMTICS'05), 180-191, 2005.*
[74] W. Zhu, and C. Thomborson, "Extraction in software watermarking"*, 8th workshop on Multimedia and security (MM&Sec'06), ACM New York, 175-181, 2006.*
[75] W. Zhu, and C. Thomborson, "Recognition in software watermarking"*, 4th ACM international workshop on Contents protection and security (MCPS'06), ACM New York, 29-36, 2006.*
[76] W. Zhu, and C. Thomborson, "On the QP Algorithm in Software Watermarking", *Intelligence and Security Informatics, 3495/2005, 646-647, 2005.*
[77] W. Zhu, C. Thomborson, and F. Wang, "A survey of software watermarking", *IEEE Int'l Conference on Intelligence and Security Informatics (ISI'05), LNCS 3495, 454–458, 2005.*
[78] X. Chen, D. Fang, J. Shen, F. Chen, W. Wang, and L. He, "A dynamic graph watermark scheme of tamper resistance", *5th International Conference on Information Assurance and Security (IAS'09), 3-6, 2009.*
[79] X. Deng, G. Xu, G. Sun, and J. Man, "Software watermarking based on dynamic program slicing"*, International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP'08), 461-464, 2008.*
[80] X. Zhang, F. He, and W. Zuo, "Hash function based on software watermarking", *Advanced Software Engineering and Its Applications (ASEA'08), 95-98, 2008.*
[81] Y. He, "Tamperproofing a software watermark by encoding constants", *Master thesis, University of Auckland, 2002.*
[82] Y. KeXin, Y. Ke, and Z. JianQi, "A robust dynamic software watermarking", *Information Technology and Computer Science (ITCS'09), International Conference on, 15-18, 2009.*
[83] Y. Zhou, A. Main, Y. Gu, and H. Johnson, "Information hiding in software with mixed boolean arithmetic transforms"*, 8th international conference on Information security applications (WISA'07), 61-75, 2007.*
[84] Y.X. Luo, J.H. Cheng, and D.Y. Fang "Dynamic graph watermark algorithm based on the threshold scheme", *Manuscript, 2008.*
[85] Z. Jiang, P. Zhong, and B. Zheng, "A software watermarking method based on public key cryptography and graph coloring", *3rd International Conference on Genetic and Evolutionary Computing (WGEC'09), 433-437, 2009.*
[86] Z. JianQi, L. YanHeng, Y. Ke, and Y. KeXin, "A robust dynamic watermarking scheme based on STBDW", *World Congress on Computer Science and Information Engineering (WRI'09), 602-606, 2009.*
[87] Z. Jianqi, L. YanHeng, Y. Ke, and Y. KeXin, "A novel method based software watermarking scheme"*, International Conference on Information Technology: New Generations (ITNG'09), 388-392, 2009.*
[88] Z. Pervez, Y. Mahmood, and H. Ahmad, "Semblance based disseminated software watermarking algorithm", *23rd International Symposium on Computer and Information Sciences (ISCIS'08), 1-4, 2008.*
[89] Z. Sha, H. Jiang, and A. Xuan, "Software watermarking algorithm by coefficients of equation", *3rd International Conference on Genetic and Evolutionary Computing, (WGEC'09) , 410-413, 2009.*

Παράρτημα 1

Κώδικας Δημιουργίας Υδατογραφήματος Από Φυσικό Αριθμό - Ορθή Διαδικασία

- Στον παρακάτω κώδικα, μετατρέπεται ένας φυσικός αριθμός από το 4 έως και το 127 σε ένα μεταθετικό αναγώγιμο γράφημα.
- Η μετατροπή αυτή στηρίζεται στην αυτοαναστρέφουσα μετάθεση.
- Με τον τρόπο αυτόν δημιουργείται μία δομή (υδατογράφημα) έτοιμη προς ενσωμάτωση σε κάποιο λογισμικό.

```
/* --Grafthke apo Iwannhs Xionhs A.M:1430-- */



/* --Edw einai oi dhlwseis twn includes kai twn defines-- */
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
/* --Mexri edw-- */



/* --Edw einai h dhlwshs ths apla sundedemenhs listas-- */
struct list_elements{
  int value;
  struct list_elements *next;
};
typedef struct list_elements item;
/* --Mexri edw-- */



/* --Edw einai oi dhlwseis twn global metablhtwn ths prwths diadikasias-- */
int w;
int length=0, *B, *B_tonos, *B_asteraki, *X, *Y, *BP, *sip;
int *Z, **COUPLES; /* --Z:= bohthhtikos pinakas gia thn s.i.p-- */
int megethos_X=0, megethos_Y=0;
int pos_1=0, pos_2=0;
int flow=1;
/* --Mexri edw-- */



/* --Edw einai oi dhlwseis twn global metablhtwn ths deuterhs diadikasias-- */
item **dag, **final_graph;
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou kanei thn prosfwnhsh mias diadikasias metatrophs-- */
void prosfwnhsh(){

  if(flow==1){
        /* --Edw einai h prosfwnhsh ths prwths diadikasias-- */
        fprintf(stdout, "\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Edw ksekinaei h diadikasia metatrophs tou fusikou arithmou sthn s.i.p#\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
```

```
        /* --Mexri edw-- */
 }
 else{
        /* --Edw einai h prosfwnhsh thw deuterhs diadikasias-- */
        fprintf(stdout, "\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Edw ksekinaei h diadikasia metatrophs ths s.i.p sto r.p.g            #\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
 /* --Mexri edw-- */
 }

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou dhlwnei to telos mias diadikasias metatrophs-- */
void telos(){

 if(flow==1){
        /* --Edw einai to telos ths prwths diadikasias-- */
        fprintf(stdout, "\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Edw teliwnei h diadikasia metatrophs tou fusikou arithmou sthn s.i.p #\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
        /* --Mexri edw-- */
 }
 else{
        /* --Edw einai to telos ths deuterhs diadikasias-- */
        fprintf(stdout, "\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Edw teliwnei h diadikasia metatrophs ths s.i.p sto r.p.g             #\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
        /* --Mexri edw-- */
 }

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou diabazei to udatoshma-- */
void watermark(){

 /* --Edw diabazoume apo to plhktrologio thn timh pros udatografhsh-- */
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "Plhktrologiste ton fusiko arithmo pros udatografhsh(w>=4 kai w<=127):#\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n");
 fscanf(stdin, "%d", &w);
 fprintf(stdout, "\n");
 while((w<4) || (w>127)){
        fprintf(stdout, "Lathos, parakalw plhktrologiste ksana ton fusiko arithmo:\n");
        fprintf(stdout, "\n");
        fscanf(stdin, "%d", &w);
        fprintf(stdout, "\n");
 }
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou kanei kapoia apo ta malloc kai tis antistoixes arxikopoihseis-- */
void memory(){

 /* --Edw einai oi metablhtes ths sunarthshs memory-- */
 int i=0;
 /* --Mexri edw-- */
```

```
 /* --Edw kanoume malloc ton pinaka B kai kratame to mhkos tou-- */
 if((w>=4) && (w<=7)){
        B=(int*)malloc(3* sizeof(int));
        if(B==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
        length=3;
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Briskomaste sto sunolo N%d twn fusikwn arithmwn!!!!!!!!!!            #\n",
((2*length)+1));
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
 }
 else if((w>=8) && (w<=15)){
        B=(int*)malloc(4* sizeof(int));
        if(B==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
        length=4;
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Briskomaste sto sunolo N%d twn fusikwn arithmwn!!!!!!!!!!            #\n",
((2*length)+1));
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
 }
 else if((w>=16) && (w<=31)){
        B=(int*)malloc(5* sizeof(int));
        if(B==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
        length=5;
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Briskomaste sto sunolo N%d twn fusikwn arithmwn!!!!!!!!!!            #\n",
((2*length)+1));
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
 }
 else if((w>=32) && (w<=63)){
        B=(int*)malloc(6* sizeof(int));
        if(B==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
        length=6;
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Briskomaste sto sunolo N%d twn fusikwn arithmwn!!!!!!!!!!            #\n",
((2*length)+1));
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
 }
 else if((w>=64) && (w<=127)){
        B=(int*)malloc(7* sizeof(int));
        if(B==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
        length=7;
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Briskomaste sto sunolo N%d twn fusikwn arithmwn!!!!!!!!!!            #\n",
((2*length)+1));
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
 }
 /* --Mexri edw-- */

 /* --Edw arxikopoioume ton B-- */
 for(i=0; i<length; i++){
        B[i]=0;
 }
 /* --Mexri edw-- */

 /* --Edw kanoume malloc gia ton B_tonos kai ton B_asteraki-- */
 B_tonos=(int*)malloc(((2*length)+1)* sizeof(int));
 if(B_tonos==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 B_asteraki=(int*)malloc(((2*length)+1)* sizeof(int));
 if(B_asteraki==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
```

```
 }
 /* --Mexri edw-- */

 /* --Edw arxikopoioume ton B_tonos kai ton B_asteraki-- */
 for(i=0; i<((2*length)+1); i++){
        B_tonos[i]=0;
 }
 for(i=0; i<((2*length)+1); i++){
        B_asteraki[i]=0;
 }
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou diabazei to w kai to metatrepei se duadikh morfh-- */
void w_to_binary(){

 /* --Edw einai oi metablhtes ths sunarthshs w_to_binary-- */
 int i=0, puliko_int=0;
 float puliko_float=0.0;
 /* --Mexri edw-- */

 /* --Edw metatrepoume to w se duadikh morfh-- */
 i=length-1;
 puliko_int=w;
 puliko_float=(float)w;
 while(puliko_int!=0.0){
        puliko_float=puliko_int/2.0;
        puliko_int=puliko_int/2;
        if((float)puliko_int<puliko_float){
               B[i]=1;
        }
        else{
               B[i]=0;
        }
        i--;
 }
 /* --Mexri edw-- */

 /* --Edw tupwnoume ton pinaka B-- */
 fprintf(stdout, "##############################################################################\n");
 fprintf(stdout, "B:                                                                          #\n");
 fprintf(stdout, "##############################################################################\n");
 fprintf(stdout, "\n(");
 for(i=0; i<length-1; i++){
        fprintf(stdout, "%d, ", B[i]);
 }
 fprintf(stdout, "%d)\n\n", B[length-1]);
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou ftiaxnei tous B_tonos kai B_asteraki-- */
void B_tonos_and_B_asteraki(){

 /* --Edw einai oi metablhtes ths sunarthshs B_tonos_and_B_asteraki-- */
 int i=0, ii=0;
 /* --Mexri edw-- */

 /* --Edw ftiaxnoume tous B_tonos kai B_asteraki-- */
 for(i=length; i<2*length; i++){
        B_tonos[i]=B[ii];
        ii++;
 }
 B_tonos[2*length]=1;
 fprintf(stdout, "##############################################################################\n");
 fprintf(stdout, "B_tonos:                                                                    #\n");
 fprintf(stdout, "##############################################################################\n");
 fprintf(stdout, "\n(");
```

```
 for(i=0; i<(2*length); i++){
        fprintf(stdout, "%d, ", B_tonos[i]);
 }
 fprintf(stdout, "%d)\n\n", B_tonos[2*length]);
 for(i=0; i<((2*length)+1); i++){
        if(B_tonos[i]==0){
               B_asteraki[i]=1;
        }
        else{
               B_asteraki[i]=0;
        }
 }
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "B_asteraki:                                                             #\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n(");
 for(i=0; i<(2*length); i++){
        fprintf(stdout, "%d, ", B_asteraki[i]);
 }
 fprintf(stdout, "%d)\n\n", B_asteraki[2*length]);
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou dhmiourgei tous X kai Y pinakes-- */
void X_and_Y(){

 /* --Edw einai oi metablhtes ths sunarthshs X_and_Y-- */
 int i=0, ii=0, iii=0, sum_1=0, sum_2=0;
 /* --Mexri edw-- */

 /* --Edw prwta brisk poses theseis xreiazontai autoi oi pinakes (X kai Y) kai meta kanoume to malloc-- */
 for(i=0; i<((2*length)+1); i++){
        if(B_asteraki[i]==0){
               sum_1++;
               megethos_X++;
        }
        else{
               sum_2++;
               megethos_Y++;
        }
 }
 X=(int*)malloc(sum_1* sizeof(int));
 if(X==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<sum_1; i++){
        X[i]=0;
 }
 Y=(int*)malloc(sum_2* sizeof(int));
 if(Y==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<sum_2; i++){
        Y[i]=0;
 }
 for(i=0; i<((2*length)+1); i++){
        if(B_asteraki[i]==0){
               X[ii]=i+1;
               ii++;
        }
        else{
               Y[iii]=i+1;
               iii++;
        }
 }
 /* --Mexri edw-- */

 /* --Edw tupwnoume tous pinakes X kai Y-- */
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "X:                                                                      #\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n(");
```

```
 for(i=0; i<ii-1; i++){
        fprintf(stdout, "%d, ", X[i]);
 }
 fprintf(stdout, "%d)\n\n", X[ii-1]);
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "Y:                                                                      #\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n(");
 for(i=0; i<iii-1; i++){
        fprintf(stdout, "%d, ", Y[i]);
 }
 fprintf(stdout, "%d)\n\n", Y[iii-1]);
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou dhmiourgei thn bitoc permutation-- */
void bitonic_p(){

 /* --Edw einai oi metablhtes ths sunarthshs bitonic_p-- */
 int i=0, ii=megethos_Y-1;
 /* --Mexri edw-- */

 /* --Edw kanoume malloc gia ton BP-- */
 BP=(int*)malloc(((2*length)+1)* sizeof(int));
 if(BP==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 /* --Mexri edw-- */

 /* --Edw briskoume ta stoixeia tou BP-- */
 for(i=0; i<megethos_X; i++){
        BP[i]=X[i];
 }
 for(i=megethos_X; i<((2*length)+1); i++){
        BP[i]=Y[ii];
        ii--;
 }
 /* --Mexri edw-- */

 /* --Edw tupwnoume ta stoixeia tou BP-- */
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "BP:                                                                     #\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n(");
 for(i=0; i<2*length; i++){
        fprintf(stdout, "%d, ", BP[i]);
 }
 fprintf(stdout, "%d)\n\n", BP[2*length]);
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou dhmiourgei thn s.i.p-- */
void s_i_p(){

 /* --Edw einai oi metablhtes ths sunarthshs s_i_p-- */
 int i=0, j=0, ii=0, mhkos=0;
 float mhkoss=0.0;
 /* --Mexri edw-- */

 /* --Edw kanoume to malloc gia ton pin Z kai sip pou tha mas xreishmeusoun sth dhmiourgia ths s.i.p-- */
 Z=(int*)malloc(((2*length)+1)* sizeof(int));
 if(Z==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 sip=(int*)malloc(((2*length)+1)* sizeof(int));
 if(sip==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
```

```
}
/* --Mexri edw-- */

/* --Edw arxikopoioume ton Z kai ton sip-- */
for(i=0; i<((2*length)+1); i++){
	Z[i]=0;
	sip[i]=0;
}
/* --Mexri edw-- */

/* --Edw gemizoume ton Z-- */
for(i=0; i<((2*length)+1); i++){
	Z[i]=BP[i];
}
/* --Mexri edw-- */

/* --Edw tupwnoume ton Z-- */
fprintf(stdout, "#########################################################################\n");
fprintf(stdout, "Z (bohthitikos):                                                        #\n");
fprintf(stdout, "#########################################################################\n");
fprintf(stdout, "\n(");
for(i=0; i<(2*length); i++){
	fprintf(stdout, "%d, ", Z[i]);
}
fprintf(stdout, "%d)\n\n", Z[2*length]);
/* --Mexri edw-- */

/* --Edw elegxoume an to mhkos tou Z einai artio h peritto kai gemizoume ton sip-- */
mhkos=(2*length)+1;
mhkoss=mhkos/2.0;
mhkos=mhkos/2;
if((float)mhkos==mhkoss){

	/* --Edw kanoume malloc ton pinaka me ta zeugraia kai ton arxik otan to (2*length): artio-- */
	COUPLES=(int **)malloc(length*sizeof(int *));
		if(COUPLES==NULL){
		fprintf(stdout, "Den egine swsta to malloc!!!!\n");
	}
		for(i=0; i<length; i++){
		COUPLES[i]=(int *)malloc(2*sizeof(int));
			if(COUPLES[i]==NULL){
			fprintf(stdout, "Den egine swsta to malloc!!!!\n");
		}
	}
	for(i=0; i<length; i++){
		for(j=0; j<2; j++){
			COUPLES[i][j]=0;
		}
	}
	/* --Mexri edw-- */

	/* --Edw ftiaxnoume ta zeugaria-- */
	for(i=0; i<length; i++){
		COUPLES[i][0]=Z[i];
		COUPLES[i][1]=Z[(2*length)-i];
	}
	/* --Mexri edw-- */

	/* --Edw ektupwnoume ta zeugaria pou ftiaksame-- */
	fprintf(stdout, "#########################################################################\n");
	fprintf(stdout, "Ta zeugraia:                                                            #\n");
	fprintf(stdout, "#########################################################################\n");
	fprintf(stdout, "\n");
	for(i=0; i<length; i++){
		fprintf(stdout, "(%d, %d)\n", COUPLES[i][0], COUPLES[i][1]);
	}
	fprintf(stdout, "\n");
	/* --Mexri edw-- */

	/* --Edw ftiaxnoume kai ektupwnoume thn s.i.p-- */
	for(i=0; i<length; i++){
		pos_1=COUPLES[i][0];
		pos_2=COUPLES[i][1];
		sip[pos_2-1]=pos_1;
		sip[pos_1-1]=pos_2;
	}
	fprintf(stdout, "#########################################################################\n");
```

```
        fprintf(stdout, "S.i.p:                                                                 #\n");
        fprintf(stdout, "#######################################################################\n");
        fprintf(stdout, "\n");
        fprintf(stdout, "------\n(");
        for(i=0; i<(2*length)-1; i++){
                fprintf(stdout, "%d, ", sip[i]);
        }
        fprintf(stdout, "%d)\n", sip[(2*length)-1]);
        fprintf(stdout, "------\n");
        fprintf(stdout, "\n");
        /* --Mexri edw-- */

}
else{

        /* --Edw kanoume malloc ton pinaka me ta zeugraia kai ton arxik otan (2*length)+1: peritto-- */
        COUPLES=(int **)malloc((length+1)*sizeof(int *));
                if(COUPLES==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
                for(i=0; i<(length+1); i++){
                COUPLES[i]=(int *)malloc(2*sizeof(int));
                        if(COUPLES[i]==NULL){
                        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
                }
        }
        for(i=0; i<(length+1); i++){
                for(j=0; j<2; j++){
                        COUPLES[i][j]=0;
                }
        }
        /* --Mexri edw-- */

        /* --Edw ftiaxnoume ta zeugaria-- */
        for(i=0; i<length; i++){
                COUPLES[i][0]=Z[i];
                COUPLES[i][1]=Z[(2*length)-i];
        }
        COUPLES[length][0]=Z[length];
        COUPLES[length][1]=Z[length]; /* --Edw baz authn thn timh gia na gin zeugari h teleutaia timh-- */
        /* --Mexri edw-- */

        /* --Edw ektupwnoume ta zeugaria pou ftiaksame-- */
        fprintf(stdout, "#######################################################################\n");
        fprintf(stdout, "Ta zeugraia:                                                           #\n");
        fprintf(stdout, "#######################################################################\n");
        fprintf(stdout, "\n");
        for(i=0; i<(length+1); i++){
                fprintf(stdout, "(%d, %d)\n", COUPLES[i][0], COUPLES[i][1]);
        }
        fprintf(stdout, "\n");
        /* --Mexri edw-- */

        /* --Edw ftiaxnoume kai ektupwnoume thn s.i.p-- */
        for(i=0; i<length; i++){
                pos_1=COUPLES[i][0];
                pos_2=COUPLES[i][1];
                sip[pos_2-1]=pos_1;
                sip[pos_1-1]=pos_2;
        }
        pos_1=COUPLES[length][0];
        sip[pos_1-1]=pos_1;
        fprintf(stdout, "#######################################################################\n");
        fprintf(stdout, "S.i.p:                                                                 #\n");
        fprintf(stdout, "#######################################################################\n");
        fprintf(stdout, "\n");
        fprintf(stdout, "------\n(");
        for(i=0; i<(2*length); i++){
                fprintf(stdout, "%d, ", sip[i]);
        }
        fprintf(stdout, "%d)\n", sip[2*length]);
        fprintf(stdout, "------\n");
        fprintf(stdout, "\n");
        /* --Mexri edw-- */

}
/* --Mexri edw-- */
```

```
}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou dhmiourgei to d.a.g-- */
void create_dag(){

 /* --Edw einai dhlwseis twn metablhtwn ths create_dag-- */
 int i=0, k=0, temp=0, flag=0;
 item **current_pointer;
 /* --Mexri edw-- */

 length=(2*length)+1;

 /* --Edw dhmiourgoume ton monodiastato pinaka pou kathe stoixeio tou einai (deikths) kai ena-- */
 /* --stoixeio tou d.a.g to opoio me th seira tou deixnei tous geitwnes tou sto d.a.g-- */
 /* --Edw kanoume malloc gia ton pinaka dag gia na dextei ta stoixeia ths sip kai gia ton
 /* --current_pointer ton opoio tha xrhsimopoioume gia na gurizoume opote theloume sthn arxh tou d.a.g-- */

 /* --Einai length+2 gia na kanoume malloc kai gia tous s kai t kombous tou d.a.g-- */
 dag=(item **)malloc((length+2)* sizeof(item*));
 if(dag==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<length+2; i++){
        dag[i]=(item *)malloc(1* sizeof(item));

        if(dag[i]==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
 }

 /* --Einai length+2 gia na kanoume malloc kai gia tous s kai t kombous tou-- */
 current_pointer=(item **)malloc(length+2* sizeof(item*));
 if(current_pointer==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<length+2; i++){
        current_pointer[i]=(item *)malloc(1* sizeof(item));

        if(current_pointer[i]==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
 }
 /* --Mexri edw-- */

 /* --Edw antigrafoume thn sip ston pinaka dag kai to dag ston current_pointer-- */
 /* --gia na mporoume na gurizoume sthn arxh opote theloume-- */
 for(i=0; i<length; i++){
        dag[i]->value=sip[i];
        dag[i]->next=NULL;
 }
 dag[length]->value=1000;
 dag[length]->next=NULL;
 dag[length+1]->value=-1000;
 dag[length+1]->next=NULL;
 for(i=0; i<length+2; i++){
        current_pointer[i]=dag[i];
 }
 /* --Mexri edw-- */

 /* --Edw briskoume thn didomination sxesh gia kathe stoixeio-- */
 for(k=0; k<length-1; k++){
        i=k;
        temp=0;
        while((i+1)<length){
                if((sip[k]>sip[i+1]) && (sip[i+1]>=temp)){
                        dag[k]->next=(item *)malloc(1* sizeof(item));
                        dag[k]=dag[k]->next;
                        dag[k]->value=sip[i+1];
                        dag[k]->next=NULL;
                        temp=sip[i+1];
                }
                i++;
```

```
        }
    }
    /* --Mexri edw-- */

    /* --Edw briskoume tous geitones twn s kai t kombwn tou d.a.g-- */
    for(i=0; i<length+2; i++){
        dag[i]=current_pointer[i];
    }
    for(k=0; k<length; k++){
        for(i=0; i<length+2; i++){
            if(i!=length){
                dag[i]=current_pointer[i];
            }
        }
        flag=0;
        i=0;
        while((i<length) && (flag==0)){
            dag[i]=dag[i]->next;
            while(dag[i]!=NULL){
                if(dag[i]->value!=sip[k]){
                    dag[i]=dag[i]->next;
                }
                else{
                    flag=1;
                    break;
                }
            }
            if(flag==0){
                i++;
            }
        }
        if(flag==0){
            dag[length]->next=(item *)malloc(1* sizeof(item));
            dag[length]=dag[length]->next;
            dag[length]->value=sip[k];
            dag[length]->next=NULL;
        }
    }
    for(i=0; i<length+2; i++){
        dag[i]=current_pointer[i];
    }
    for(i=0; i<length; i++){
        if(dag[i]->next==NULL){
            dag[i]->next=(item *)malloc(1* sizeof(item));
            dag[i]=dag[i]->next;
            dag[i]->value=-1000;
            dag[i]->next=NULL;
        }
    }
    /* --Mexri edw-- */

    for(i=0; i<length+2; i++){
        dag[i]=current_pointer[i];
    }

    /* --Edw tupwnoume to d.a.g pou dhmiourghthhke-- */
    fprintf(stdout, "#########################################################################\n");
    fprintf(stdout, "D.a.g (opou 1000 einai o s kombos kai -1000 einai o t kombos):       #\n");
    fprintf(stdout, "#########################################################################\n");
    fprintf(stdout, "\n");
    for(i=0; i<length+2; i++){
        while(dag[i]!=NULL){
            fprintf(stdout, "(%d)---->", dag[i]->value);
            dag[i]=dag[i]->next;
        }
        fprintf(stdout, "NULL\n");
    }
    fprintf(stdout, "\n");
    /* --Mexri edw-- */

    /* --Mexri edw-- */

    for(i=0; i<length+2; i++){
        dag[i]=current_pointer[i];
    }

}
```

```
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou dhmiourgei to teliko grafhma-- */
void create_final_graph(){

 /* --Edw einai dhlwseis twn metablhtwn ths create_final_graph-- */
 int i=0, ii=0, iii=0, k=0, temp=0, temp_2=0, sum=0, sum_2=0, maximum=0, maximum_2=0, stop=0, flag=0;
 int *temp_buffer, temp_buffer_2[100], max_element[100];

 /* --O prwtos einai gia na mporoume na gurizoume sthn arxh tou final_graph opote theloume-- */
 /* --kai deuteros einai gia na kanoume thn idia diadikasia sto d.a.g-- */
 item **current_pointer, **current_pointer_2;
 /* --Mexri edw-- */

 /* --Edw tha antigrapsoume thn s.i.p se enan prosorino pinaka kai meta tha thn taksinomhsoume-- */
 temp_buffer=(int *)malloc((length+2)*sizeof(int));
 if(temp_buffer==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<length; i++){
        temp_buffer[i]=sip[i];
 }
 temp_buffer[length]=1000;
 temp_buffer[length+1]=-1000;
 for(k=1; k<length+2; k++){
        for(i=length+1; i>=k; i--){
                if(temp_buffer[i]>temp_buffer[i-1]){
                        temp=temp_buffer[i];
                        temp_buffer[i]=temp_buffer[i-1];
                        temp_buffer[i-1]=temp;
                }
        }
 }
 /* --Mexri edw-- */

 /* --Edw dhmiourgoume ton monodiastato pinaka pou kathe stoixeio tou einai (deikths) kai-- */
 /* --ena stoixeio tou final_graph to opoio me th seira tou deixnei tous geitwnes tou sto final_graph-- */

 /* --Edw kan malloc gia ton pinaka final_graph gia na dextei ta stoixeia kai gia tous current_pointer-- */
 /* --kai current_pointer_2 tous opoious tha xrhsimopoioume gia na gurizoume opote theloume-- */
 /* --sthn arxh tou final_graph kai tou d.a.g-- */

 /* --Einai length+2 gia na kanoume malloc kai gia tous s kai t kombous tou final_graph-- */
 final_graph=(item **)malloc((length+2)* sizeof(item*));
 if(final_graph==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<length+2; i++){
        final_graph[i]=(item *)malloc(1* sizeof(item));
        if(final_graph[i]==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
 }

 /* --Einai length+2 gia na kanoume malloc kai gia tous s kai t kombous tou-- */
 current_pointer=(item **)malloc(length+2* sizeof(item*));
 if(current_pointer==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<length+2; i++){
        current_pointer[i]=(item *)malloc(1* sizeof(item));
        if(current_pointer[i]==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
 }

 /* --Einai length+2 gia na kanoume malloc kai gia tous s kai t kombous tou-- */
 current_pointer_2=(item **)malloc(length+2* sizeof(item*));
 if(current_pointer_2==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<length+2; i++){
        current_pointer_2[i]=(item *)malloc(1* sizeof(item));
        if(current_pointer_2[i]==NULL){
```

```
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
}
/* --Mexri edw-- */

for(i=0; i<length+2; i++){
        current_pointer[i]=final_graph[i];
}
for(i=0; i<length+2; i++){
        current_pointer_2[i]=dag[i];
}
for(i=0; i<length+2; i++){
        final_graph[i]->value=temp_buffer[i];
        final_graph[i]->next=NULL;
}

/* --Edw arxikopoioume ton temp_buffer_2 o opoios tha exei ola ta stoixeia pou tha deixnoun kathe-- */
/* --fora se kathe ena stoixeio tou d.a.g-- */
for(i=0; i<100; i++){
        temp_buffer_2[i]=0;
        max_element[i]=0;
}
/* --Mexri edw-- */

/* --Edw dhmiourgoume to final graph-- */
for(i=0; i<length+1; i++){
        final_graph[i]->next=(item *)malloc(sizeof(item));
        final_graph[i]=final_graph[i]->next;
                final_graph[i]->value=temp_buffer[i+1];
                final_graph[i]->next=NULL;
}
for(i=0; i<length+2; i++){
        final_graph[i]=current_pointer[i];
}
for(i=0; i<length+2; i++){
        dag[i]=current_pointer_2[i];
}
for(ii=0; ii<length; ii++){
        maximum=0;
        sum=0;
        for(i=0; i<length+2; i++){
                while(dag[i]->next!=NULL){
                        max_element[sum_2]=dag[i]->value;
                        sum_2++;
                        if(dag[i]->next->value==sip[ii]){
                                flag=1;
                                break;
                        }
                        dag[i]=dag[i]->next;
                }
                if(flag==1){
                        for(iii=0; iii<sum_2; iii++){
                                if(max_element[iii]>maximum_2){
                                        maximum_2=max_element[iii];
                                }
                        }
                        temp_buffer_2[sum]=maximum_2;
                        sum++;
                        flag=0;
                        maximum_2=0;
                }
                for(iii=0; iii<100; iii++){
                        max_element[i]=0;
                }
                sum_2=0;
        }
        for(i=0; i<sum; i++){
                if(temp_buffer_2[i]>maximum){
                        maximum=temp_buffer_2[i];
                }
        }
        if(sum!=0){
                i=0;
                stop=0;
                while(i<length+2){
                        if(final_graph[i]->value==sip[ii]){
                                while(final_graph[i]->next!=NULL){
```

```
                                        final_graph[i]=final_graph[i]->next;
                                }
                                if(final_graph[i]->next==NULL){
                                        final_graph[i]->next=(item *)malloc(sizeof(item));
                                        final_graph[i]=final_graph[i]->next;
                                                final_graph[i]->value=maximum;
                                                final_graph[i]->next=NULL;
                                        break;
                                }
                        }
                        i++;
                }
        }
        for(i=0; i<length+2; i++){
                final_graph[i]=current_pointer[i];
        }
        for(i=0; i<length+2; i++){
                dag[i]=current_pointer_2[i];
        }
        for(i=0; i<100; i++){
                temp_buffer_2[i]=0;
                max_element[i]=0;
        }
 }
 /* --Mexri edw-- */

 /* --Edw tupwnoume to teliko grafhma pou dhmiourghthhke-- */
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "Final_graph (opou 1000 einai o s kombos kai -1000 einai o t kombos): #\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n");
 fprintf(stdout, "------\n");
 for(i=0; i<length+2; i++){
        while(final_graph[i]!=NULL){
                fprintf(stdout, "(%d)---->", final_graph[i]->value);
                final_graph[i]=final_graph[i]->next;
        }
        fprintf(stdout, "NULL\n");
 }
 fprintf(stdout, "------\n");
 fprintf(stdout, "\n");
 /* --Mexri edw-- */

 for(i=0; i<length+2; i++){
        final_graph[i]=current_pointer[i];
 }
 for(i=0; i<length+2; i++){
        dag[i]=current_pointer_2[i];
 }
}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh main-- */
int main(){

 /* --Edw einai oi metablhtes ths sunarthshs main-- */
 int i=0;
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh gia thn prosf ths diad metatrophs enos fusikou arithmous se mia s.i.p-- */
 prosfwnhsh();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou diabazei to udatoshma-- */
 watermark();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh gia na kanoume kapoia apo ta aparaithta malloc-- */
 memory();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh gia na feroume se duadikh morfh ton akeraio w-- */
 w_to_binary();
 /* --Mexri edw-- */
```

```
 /* --Edw kaloume thn sunarthsh pou dhmiourgei ton pinaka B_tonos kai ton B_asteraki-- */
 B_tonos_and_B_asteraki();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou ftiaxnei tous pinakes X kai Y-- */
 X_and_Y();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou ftiaxnei thn bitonic permutation-- */
 bitonic_p();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou ftiaxnei ta zeugraria kai thn self inverting permutation (s.i.p)-- */
 s_i_p();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou dhlwnei to telos diad metatrop enos fusikou arithmou se mia s.i.p-- */
 telos();
 /* --Mexri edw-- */

 flow=2;

 /* --Edw kaloume thn sunarthsh gia thn prosfwnhsh ths diadikasias metatrophs mias s.i.p se ena r.p.g-- */
 prosfwnhsh();
 /* --Mexri edw-- */

 /* --Edw einai h klhsh ths sunarthshs pou dhmiourgei to d.a.g-- */
 create_dag();
 /* --Mexri edw-- */

 /* --Edw einai h klhsh ths sunarthshs pou dhmiourgei to teliko grafhma-- */
 create_final_graph();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou dhlwnei to telos ths diad metatrop mias s.i.p se ena r.p.g-- */
 telos();
 /* --Mexri edw-- */

 /* --Edw kanoume tis apodesmeuseis mnhmhs stous deiktes-- */
 free(B);
 free(B_tonos);
 free(B_asteraki);
 free(BP);
 free(X);
 free(Y);
 free(Z);
 for(i=0; i<2; i++){
        free(COUPLES[i]);
 }
 free(COUPLES);
 free(sip);
 for(i=0; i<length+2; i++){
        free(dag[i]);
        free(final_graph[i]);
 }
 free(dag);
 free(final_graph);
 /* --Mexri edw-- */

}
/* --Mexri edw-- */
```

Παράρτημα 2

Κώδικας Δημιουργίας Φυσικού Αριθμού Από Υδατογράφημα - Αντίστροφη Διαδικασία

- Στον παρακάτω κώδικα, μετατρέπεται ένα μεταθετικό αναγώγιμο γράφημα (η δομή που έχει ενσωματωθεί σε κάποιο λογισμικό) σε έναν φυσικό αριθμό από το 4 έως και το 127.
- Η μετατροπή αυτή στηρίζεται στην αυτοαναστρέφουσα μετάθεση.
- Με τον τρόπο αυτόν, αποδεικνύεται πως όντως το μεταθετικό αναγώγιμο γράφημα το οποίο ήταν ενσωματωμένο σε κάποιο λογισμικό κρύβει έναν και μοναδικό φυσικό αριθμό.

```
/* --Grafthke apo Iwannhs Xionhs A.M:1430-- */



/* --Edw einai oi dhlwseis twn includes kai twn defines-- */
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
/* --Mexri edw-- */



/* --Edw einai h dhlwshs ths apla sundedemenhs listas-- */
struct list_elements{
  int value;
  struct list_elements *next;
};
typedef struct list_elements item;
/* --Mexri edw-- */



/* --Edw einai oi dhlwseis twn global metablhtwn ths prwths diadikasias-- */
item **final_graph, **current_pointer, **temp_final_graph, **temp_current_pointer;
int flow=1, line_number=0;
/* --Mexri edw-- */



/* --Edw einai oi dhlwseis twn global metablhtwn ths deuterhs diadikasias-- */
int BP_2_temp[100], *BP_2, **COUPLES, *BP, *X, *Y, *B_tonos, *B_asteraki, sum_1=1, sum_2=0;
int w=0, choice=0, is=0, sip_bp=0;
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou kanei thn prosfwnhsh mias diadikasias metatrophs-- */
void prosfwnhsh(){

  if(flow==1){
         /* --Edw einai h prosfwnhsh ths prwths diadikasias-- */
         fprintf(stdout, "\n");
         fprintf(stdout, "#########################################################################\n");
         fprintf(stdout, "Edw ksekinaei h diadikasia metatrophs tou r.p.g sthn s.i.p           #\n");
         fprintf(stdout, "#########################################################################\n");
         fprintf(stdout, "\n");
         /* --Mexri edw-- */
  }
  else{
         /* --Edw einai h prosfwnhsh ths deuterhs diadikasias-- */
         fprintf(stdout, "\n");
         fprintf(stdout, "#########################################################################\n");
```

```
        fprintf(stdout, "Edw ksekinaei h diadikasia metatrophs ths s.i.p ston fusiko arithmo  #\n");
        fprintf(stdout, "#################################################################################\n");
        fprintf(stdout, "\n");
        /* --Mexri edw-- */
  }

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou dhlwnei to telos mias diadikasias metatrophs-- */
void telos(){

  if(flow==1){
        /* --Edw einai to telos ths prwths diadikasias-- */
        fprintf(stdout, "\n");
        fprintf(stdout, "#################################################################################\n");
        fprintf(stdout, "Edw teliwnei h diadikasia metatrophs tou r.p.g sthn s.i.p            #\n");
        fprintf(stdout, "#################################################################################\n");
        fprintf(stdout, "\n");
        /* --Mexri edw-- */
  }
  else{
        /* --Edw einai to telos ths deuterhs diadikasias-- */
        fprintf(stdout, "\n");
        fprintf(stdout, "#################################################################################\n");
        fprintf(stdout, "Edw teliwnei h diadikasia metatrophs ths s.i.p ston fusiko arithmo   #\n");
        fprintf(stdout, "#################################################################################\n");
        fprintf(stdout, "\n");
        /* --Mexri edw-- */
  }

}
/* --Mexri edw-- */



/* --Edw anoigei to arxeio pou exei to udatografhma-- */
int open_file_for_read(){
  char *inname="my_file";
  FILE *infile;
  char line_buffer[BUFSIZ];

  infile = fopen(inname, "r");
  if(!infile){
        printf("Den mporei na anoiksei to arxeio %s gia diabasma!!!!\n", inname);
        return(0);
  }
  printf("To arxeio %s anoikse gia diabasma.\n\n", inname);

  line_number=0;
  while (fgets(line_buffer, sizeof(line_buffer), infile)) {
        ++line_number;
        printf("%4d: %s", line_number, line_buffer);
  }
  printf("\nO sunolikos arithmos apo grammes tou arxeiou: %s einai isos me: %d.\n\n", inname, line_number);
  fclose(infile);

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh me to menu twn epithesewn pou lambanoun xwra sto r.p.g-- */
void menu(){

  fprintf(stdout, "#################################################################################\n");
  fprintf(stdout, "Menu epithesewn sto udatografhma:                                      #\n");
  fprintf(stdout, "#################################################################################\n\n");
  fprintf(stdout, "Gia elegxo epitheshs se kombous plhktrologhste 1.....................\n");
  fprintf(stdout, "Gia elegxo epitheshs se akmes plhktrologhste 2.....................\n");
  fprintf(stdout, "Gia elegxo epitheshs sthn s.i.p plhktrologhste 3.....................\n");
  fprintf(stdout, "Gia elegxo epitheshs sthn bitonic permutation plhktrologhste 4........\n");
```

```
 fprintf(stdout, "Gia elegxo epitheshs sta labels plhktrologhste 5.....................\n");
 fprintf(stdout, "Xwris elegxo epitheshs plhktrologhste 0.............................\n\n");
 fscanf(stdin, "%d", &choice);

 while((choice!=0) && (choice!=1) && (choice!=2) && (choice!=3) && (choice!=4) && (choice!=5)){
        fprintf(stdout, "\nLathos, parakalw plhktrologiste ksana to noumero pou epithumeite.\n\n");
        fscanf(stdin, "%d", &choice);
 }

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou elegxei gia epitheseis stis akmes tou final_graph-- */
void attack_to_labels(){
 int i=0, ii=0, iii=0, k=0, *old_permutation, temp=0, epithesh=0, flag=0, tp_1=0, tp_2=0, tp_3=0, pos=0, pos_1=0,
pos_2=0;

 /* --Edw ginetai elegxow an exei ginei epithesh se lables-- */
 for(ii=0; ii<line_number; ii++){
                final_graph[ii]=current_pointer[ii];
 }

 if(final_graph[0]->next->value!=line_number-2){
        epithesh=1;
 }
 if(final_graph[line_number-1]->next!=NULL){
        epithesh=1;
 }

 k=0;
 while(k<line_number-2){
        if(final_graph[k]->next->value!=line_number-2-k){
                epithesh=1;
                break;
        }
        k++;
 }

 k=1;
 while(k<line_number-1){
        if(final_graph[k]->value!=line_number-2-k+1){
                epithesh=1;
                break;
        }
        k++;
 }
 /* --Mexri edw-- */

 if(epithesh==1){
        fprintf(stdout, "??????????  To grafhma exei upostei epithesh sta labels.\n\n");

        /* --Edw kanoume malloc ton pinaka ths allagmenhs metatheshs-- */
        old_permutation=(int *)malloc(line_number* sizeof(int));
        /* --Mexri edw-- */

        /* --Edw kanoume arxikopoihsh ton pinaka ths allagmenhs metatheshs-- */
        for(i=0; i<line_number; i++){
                old_permutation[i]=0;
        }
        /* --Mexri edw-- */

        for(ii=0; ii<line_number; ii++){
                final_graph[ii]=current_pointer[ii];
        }

        /* --Edw ftiaxnoume to swsto r.p.g-- */
        i=0;
        while(i<line_number){
                if((final_graph[i]->next->next==NULL) && (final_graph[i]->next!=NULL)){
                        temp=final_graph[i]->value;
                        old_permutation[iii]=temp;
                        iii++;
                        break;
                }
```

```
		i++;
	}

	for(ii=0; ii<line_number; ii++){
		final_graph[ii]=current_pointer[ii];
	}

	while(iii<line_number){
		if(flag==0){
			i=0;
			while(i<line_number){
				if(final_graph[i]->value==temp){
					temp=final_graph[i]->next->value;
					old_permutation[iii]=temp;
					iii++;
					break;
				}
				i++;
			}
			flag=1;
		}

		for(ii=0; ii<line_number; ii++){
			if(final_graph[ii]->value==temp){
				temp=final_graph[ii]->next->value;
				old_permutation[iii]=temp;
				iii++;
				break;
			}
		}

		temp=old_permutation[iii-2];

		for(ii=0; ii<line_number; ii++){
			final_graph[ii]=current_pointer[ii];
		}

		for(ii=0; ii<line_number; ii++){
			final_graph[ii]=final_graph[ii]->next;
			while(final_graph[ii]!=NULL){
				if(final_graph[ii]->value==temp){
					/*final_graph[i]->value=0;*/
				}
				final_graph[ii]=final_graph[ii]->next;
			}
		}

		for(ii=0; ii<line_number; ii++){
			final_graph[ii]=current_pointer[ii];
		}

		temp=old_permutation[iii-1];
	}
	/* --Mexri edw-- */

	fprintf(stdout, "H kainouria akolouthia pou prokuptei einai: (");
	for(i=0; i<line_number-1; i++){
		fprintf(stdout, "(%d), ", old_permutation[i]);
	}
	fprintf(stdout, "(%d)).\nMe bash authn tha ginoun oi antikatastaseis.\n", old_permutation[line_number-
1]);
	fprintf(stdout, "Epomenws tha dhmioyrghthei kainourio final graph.\n\n");

	for(i=0; i<line_number; i++){
		final_graph[i]=current_pointer[i];
	}

	/* --Edw ginontai oi antikatastaseis-- */
	temp_final_graph=(item **)malloc(line_number* sizeof(item*));
	if(temp_final_graph==NULL){
		fprintf(stdout, "Den egine swsta to malloc!!!!\n");
	}

	temp_current_pointer=(item **)malloc(line_number* sizeof(item*));
	if(temp_current_pointer==NULL){
		fprintf(stdout, "Den egine swsta to malloc!!!!\n");
	}
```

```
for(i=0; i<line_number; i++){
        temp_final_graph[i]=(item *)malloc(1* sizeof(item));
        temp_final_graph[i]->next=NULL;

        temp_current_pointer[i]=(item *)malloc(1* sizeof(item));
        temp_current_pointer[i]->next=NULL;

        temp_current_pointer[i]=temp_final_graph[i];

        final_graph[i]=final_graph[i]->next;
        while(final_graph[i]!=NULL){
                temp_final_graph[i]->next=(item *)malloc(1* sizeof(item));
                temp_final_graph[i]=temp_final_graph[i]->next;
                temp_final_graph[i]->next=NULL;

                final_graph[i]=final_graph[i]->next;
        }
}

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
        temp_final_graph[i]=temp_current_pointer[i];
}

k=1;
for(ii=1; ii<line_number-1; ii++){
        for(i=0; i<line_number; i++){
                while(final_graph[i]!=NULL){
                        if(final_graph[i]->value==old_permutation[k]){
                                temp_final_graph[i]->value=line_number-2-k+1;
                        }
                        final_graph[i]=final_graph[i]->next;
                        temp_final_graph[i]=temp_final_graph[i]->next;
                }
        }

        for(i=0; i<line_number; i++){
                final_graph[i]=current_pointer[i];
                temp_final_graph[i]=temp_current_pointer[i];
        }

        k++;
}

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
        temp_final_graph[i]=temp_current_pointer[i];
}

for(i=0; i<line_number; i++){
        while(final_graph[i]!=NULL){
                if(final_graph[i]->value==old_permutation[0]){
                        temp_final_graph[i]->value=1000;
                }
                final_graph[i]=final_graph[i]->next;
                temp_final_graph[i]=temp_final_graph[i]->next;
        }
}

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
        temp_final_graph[i]=temp_current_pointer[i];
}

for(i=0; i<line_number; i++){
        while(final_graph[i]!=NULL){
                if(final_graph[i]->value==old_permutation[line_number-1]){
                        temp_final_graph[i]->value=-1000;
                }
                final_graph[i]=final_graph[i]->next;
                temp_final_graph[i]=temp_final_graph[i]->next;
        }
}

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
```

```
        temp_final_graph[i]=temp_current_pointer[i];
}

for(i=0; i<line_number; i++){
        if(temp_final_graph[i]->value==1000){
                while(final_graph[i]!=NULL){
                        final_graph[i]->value=temp_final_graph[i]->value;

                        final_graph[i]=final_graph[i]->next;
                        temp_final_graph[i]=temp_final_graph[i]->next;
                }
        }
}

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
        temp_final_graph[i]=temp_current_pointer[i];
}

for(k=0; k<line_number-2; k++){
        for(i=0; i<line_number; i++){
                if(temp_final_graph[i]->value==line_number-2-k){
                        while(final_graph[i]!=NULL){
                                final_graph[i]->value=temp_final_graph[i]->value;

                                final_graph[i]=final_graph[i]->next;
                                temp_final_graph[i]=temp_final_graph[i]->next;
                        }
                }
        }
        for(i=0; i<line_number; i++){
                final_graph[i]=current_pointer[i];
                temp_final_graph[i]=temp_current_pointer[i];
        }
}

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
        temp_final_graph[i]=temp_current_pointer[i];
}

for(i=0; i<line_number; i++){
        if(temp_final_graph[i]->value==-1000){
                while(final_graph[i]!=NULL){
                        final_graph[i]->value=temp_final_graph[i]->value;

                        final_graph[i]=final_graph[i]->next;
                        temp_final_graph[i]=temp_final_graph[i]->next;
                }
        }
}

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
        temp_final_graph[i]=temp_current_pointer[i];
}

/* --Edw antigrafoume ton temp_final_graph ston final graph me ton swsto tropo-- */
for(i=0; i<line_number; i++){
        while(final_graph[i]!=NULL){
                temp_final_graph[i]->value=final_graph[i]->value;
                final_graph[i]->value=0;

                final_graph[i]=final_graph[i]->next;
                temp_final_graph[i]=temp_final_graph[i]->next;
        }
}

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
}

for(i=0; i<line_number; i++){
        k=0;
        while(final_graph[i]!=NULL){
                k++;
                final_graph[i]=final_graph[i]->next;
```

```
        }
        if(k==1){
                pos_1=i;
        }
        if(k==2){
                pos_2=i;
        }
}


for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
}

final_graph[pos_1]->next=(item *)malloc(1* sizeof(item));
final_graph[pos_1]=final_graph[pos_1]->next;
final_graph[pos_1]->value=0;
final_graph[pos_1]->next=NULL;

final_graph[pos_1]->next=(item *)malloc(1* sizeof(item));
final_graph[pos_1]=final_graph[pos_1]->next;
final_graph[pos_1]->value=0;
final_graph[pos_1]->next=NULL;

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
}

final_graph[pos_2]=final_graph[pos_2]->next;
final_graph[pos_2]->next=(item *)malloc(1* sizeof(item));
final_graph[pos_2]=final_graph[pos_2]->next;
final_graph[pos_2]->value=0;
final_graph[pos_2]->next=NULL;

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
}

final_graph[0]=final_graph[0]->next;
final_graph[0]->next=NULL;

final_graph[line_number-1]->next=NULL;

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
        temp_final_graph[i]=temp_current_pointer[i];
}

for(i=0; i<line_number; i++){
        if(temp_final_graph[i]->value==1000){
                while(temp_final_graph[i]!=NULL){
                        final_graph[0]->value=temp_final_graph[i]->value;

                        final_graph[0]=final_graph[0]->next;
                        temp_final_graph[i]=temp_final_graph[i]->next;
                }
        }
}

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
        temp_final_graph[i]=temp_current_pointer[i];
}

for(i=0; i<line_number; i++){
        if(temp_final_graph[i]->value==-1000){
                while(temp_final_graph[i]!=NULL){
                        final_graph[line_number-1]->value=temp_final_graph[i]->value;

                        final_graph[line_number-1]=final_graph[line_number-1]->next;
                        temp_final_graph[i]=temp_final_graph[i]->next;
                }
        }
}

for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
```

```
                temp_final_graph[i]=temp_current_pointer[i];
        }

        k=1;
        while(k<=line_number-2){
                for(i=0; i<line_number; i++){
                        if(temp_final_graph[i]->value==k){
                                while(temp_final_graph[i]!=NULL){
                                        final_graph[line_number-1-k]->value=temp_final_graph[i]->value;

                                        final_graph[line_number-1-k]=final_graph[line_number-1-k]->next;
                                        temp_final_graph[i]=temp_final_graph[i]->next;
                                }
                        }
                }

                for(i=0; i<line_number; i++){
                        final_graph[i]=current_pointer[i];
                        temp_final_graph[i]=temp_current_pointer[i];
                }

                k++;
        }
        /* --Mexri edw-- */

        /* --Mexri edw-- */

        for(i=0; i<line_number; i++){
                final_graph[i]=current_pointer[i];
        }

        /* --Edw tupwnoume to teliko grafhma pou dhmiourghthhke-- */
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Final_graph (opou 1000 einai o s kombos -1000 einai o t kombos):     #\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
        for(i=0; i<line_number; i++){
                while(final_graph[i]!=NULL){
                        fprintf(stdout, "(%d)---->", final_graph[i]->value);
                        final_graph[i]=final_graph[i]->next;
                }
                fprintf(stdout, "NULL\n");
        }
        fprintf(stdout, "\n");
        /* --Mexri edw-- */
  }
}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou elegxei gia epitheseis stis akmes tou final_graph-- */
void attack_to_edges(){
  int i=0, out_degree=0, ok_1=0, ok_2=0, temp_vertex=0, pos=-100, correction=0, insert_1=0, insert_2=0;
  int insert_3=0, insert_4=0;

  for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
  }

  /* --Edw elegxetai an xalaei h idiothta tou grafhmatos pou kathe kombos exei ekserxomeno-- */
  /* --arithmo apo akmes iso me 2-- */
  for(i=1; i<line_number-1; i++){
        out_degree=0;
        temp_vertex=final_graph[i]->value;
        while(final_graph[i]!=NULL){
                final_graph[i]=final_graph[i]->next;
                out_degree++;
        }
        out_degree--;
        if(out_degree<2){
                fprintf(stdout, "??????????  To grafhma exei upostei epithesh stis akmes.\n");
                fprintf(stdout, "??????????  Exei xalasei h idiothta tou grafhmatos, pou kathe kombos exei out
degree iso me 2, me afairesh akmwn.\n");
                fprintf(stdout, "??????????  O kombos: %d exei lathos arithmo apo ekserxomenes akmes.\n\n",
temp_vertex);
```

```
                insert_1=1;
        }
        if(out_degree>2){
                fprintf(stdout, "??????????  To grafhma exei upostei epithesh stis akmes.\n");
                fprintf(stdout, "??????????  Exei xalasei h idiothta tou grafhmatos, pou kathe kombos exei out
degree iso me 2, me prosthesh akmwn.\n");
                fprintf(stdout, "??????????  O kombos: %d exei lathos arithmo apo ekserxomenes akmes.\n\n",
temp_vertex);
                insert_1=1;
                }
 }

 /* --Edw elegxoume an theloume na ginei epidiorthwsh ths epitheshs h na termatistei to programma-- */
 if(insert_1==1){
        fprintf(stdout, "Tha thelate na ginei epidiorthwsh ths epitheshs, h na termatisei edw to programma?\n");
        fprintf(stdout, "Gia sunexeia plhktrologhste 1, enw gia termatismo plhktrologhste 0.\n\n");
        fscanf(stdin, "%d", &correction);
        fprintf(stdout, "\n");

        if(correction==0){
                abort();
        }
 }
 /* --Mexri edw-- */

 /* --Mexri edw-- */

 /* --Edw elegxetai an o s kombos exei ekserxomeno arithmo apo akmes iso me 1 kai o t iso me 0-- */
 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }
 out_degree=0;
 pos=-100;
 for(i=0; i<line_number; i++){
        if(final_graph[i]->value==1000){
                pos=i;
                break;
        }
 }
 if(pos!=-100){
        while(final_graph[pos]!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                out_degree++;
        }
        out_degree--;
        if(out_degree>1){
                fprintf(stdout, "??????????  To grafhma exei upostei epithesh stis akmes.\n");
                fprintf(stdout, "??????????  O s kombos exei arithmo apo ekserxomenes akmes megalutero apo
1.\n\n");
                insert_3=1;
        }
        if(out_degree==0){
                fprintf(stdout, "??????????  To grafhma exei upostei epithesh stis akmes.\n");
                fprintf(stdout, "??????????  O s kombos exei arithmo apo ekserxomenes akmes iso me 0.\n\n");
                insert_3=1;
        }

        /* --Edw elegxoume an theloume na ginei epidiorthwsh ths epitheshs h na termatistei to programma-- */
        if(insert_3==1){
                fprintf(stdout, "Tha thelate na ginei epidiorthwsh ths epitheshs, h na termatisei edw to
programma?\n");
                fprintf(stdout, "Gia sunexeia plhktrologhste 1, enw gia termatismo plhktrologhste 0.\n\n");
                fscanf(stdin, "%d", &correction);
                fprintf(stdout, "\n");

                if(correction==0){
                        abort();
                }
        }
        /* --Mexri edw-- */
 }

 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }
 out_degree=0;
 pos=-100;
```

```
  for(i=0; i<line_number; i++){
          if(final_graph[i]->value==-1000){
                  pos=i;
                  break;
          }
  }
  if(pos!=-100){
          while(final_graph[pos]!=NULL){
                  final_graph[pos]=final_graph[pos]->next;
                  out_degree++;
          }
          out_degree--;
          if(out_degree>0){
                  fprintf(stdout, "??????????  To grafhma exei upostei epithesh stis akmes.\n");
                  fprintf(stdout, "??????????  O t kombos exei arithmo apo ekserxomenes akmes megalutero tou
0.\n\n");
                  insert_4=1;

                  /* --Edw elegxoume an theloume na ginei epidiorthwsh ths epitheshs h na termatistei to programma-
- */
                  if(insert_4==1){
                          fprintf(stdout, "Tha thelate na ginei epidiorthwsh ths epitheshs, h na termatisei edw to
programma?\n");
                          fprintf(stdout, "Gia sunexeia plhktrologhste 1, enw gia termatismo plhktrologhste
0.\n\n");
                          fscanf(stdin, "%d", &correction);
                          fprintf(stdout, "\n");

                          if(correction==0){
                                  abort();
                          }
                  }
                  /* --Mexri edw-- */
          }
  }
  /* --Mexri edw-- */

  /* --Edw elegxoume ton arithmo twn front edges kai twn back edges-- */
  for(i=0; i<line_number; i++){
          final_graph[i]=current_pointer[i];
  }

  for(i=1; i<line_number-2; i++){
          ok_1=0;
          ok_2=0;
          temp_vertex=final_graph[i]->value;
          while(final_graph[i]->next!=NULL){
                  if(temp_vertex-(final_graph[i]->next->value)==1){
                          ok_1=1;
                  }
                  if(temp_vertex-(final_graph[i]->next->value)<0){
                          ok_2=1;
                  }
                  final_graph[i]=final_graph[i]->next;
          }
          if(ok_1!=1){
                  fprintf(stdout, "??????????  To grafhma exei upostei epithesh stis akmes.\n");
                  fprintf(stdout, "??????????  Ston kombo: %d kamia akmh den deixnei se kombo me timh kata mia
monada mikroterh (front edge).\n\n", temp_vertex);
                  insert_2=1;

                  /* --Edw elegxoume an theloume na ginei epidiorthwsh ths epitheshs h na termatistei to programma-
- */
                  if(insert_2==1){
                          fprintf(stdout, "Tha thelate na ginei epidiorthwsh ths epitheshs, h na termatisei edw to
programma?\n");
                          fprintf(stdout, "Gia sunexeia plhktrologhste 1, enw gia termatismo plhktrologhste
0.\n\n");
                          fscanf(stdin, "%d", &correction);
                          fprintf(stdout, "\n");

                          if(correction==0){
                                  abort();
                          }
                  }
                  /* --Mexri edw-- */
          }
```

```
        if(ok_2!=1){
                fprintf(stdout, "??????????  To grafhma exei upostei epithesh stis akmes.\n");
                fprintf(stdout, "??????????  Ston kombo: %d kamia akmh den deixnei se kombo me megaluterh timh
(back edge).\n\n", temp_vertex);
                abort();
        }
 }
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou elegxei gia epitheseis stous kombous tou final_graph-- */
void attack_to_vertexes(){
 int i=0, uparxei_s=0, uparxei_t=0, maximum_kombos=0, insert=0, correction=0, temp_number=0;

 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }

 for(i=0; i<line_number; i++){
        while(final_graph[i]!=NULL){
                if(final_graph[i]->value==1000){
                        uparxei_s=1;
                }
                if(final_graph[i]->value==-1000){
                        uparxei_t=1;
                }
                final_graph[i]=final_graph[i]->next;
        }
 }

 if(uparxei_s==0){
        fprintf(stdout, "??????????  To grafhma exei upostei epithesh stous kombous.\n");
        fprintf(stdout, "??????????  Exei afairethei o kombos s.\n\n");
        abort();
 }
 if(uparxei_t==0){
        fprintf(stdout, "??????????  To grafhma exei upostei epithesh stous kombous.\n");
        fprintf(stdout, "??????????  Exei afairethei o kombos t.\n\n");
        insert=1;

        /* --Edw elegxoume an theloume na ginei epidiorthwsh ths epitheshs h na term to programma-- */
        if(insert==1){
                fprintf(stdout, "Tha thelate na ginei epidiorthwsh ths epitheshs, h na termatisei edw to
programma?\n");
                fprintf(stdout, "Gia sunexeia pathste to 1, enw gia termatismo plhktrologhste 0.\n\n");
                fscanf(stdin, "%d", &correction);
                fprintf(stdout, "\n");

                if(correction==0){
                        abort();
                }
                else{
                        for(i=0; i<line_number; i++){
                                final_graph[i]=current_pointer[i];
                        }

                        if(final_graph[line_number-1]->next!=NULL){
                                line_number++;

                                final_graph=(item **)realloc(final_graph, line_number* sizeof(item*));
                                if(final_graph==NULL){
                                        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
                                }
                                current_pointer=(item **)realloc(current_pointer, line_number* sizeof(item*));
                                if(current_pointer==NULL){
                                        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
                                }

                                final_graph[line_number-1]=(item *)malloc(1* sizeof(item));
                                if(final_graph[line_number-1]==NULL){
                                        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
                                }
```

```
                                current_pointer[line_number-1]=(item *)malloc(1* sizeof(item));
                                if(current_pointer[line_number-1]==NULL){
                                        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
                                }

                                for(i=0; i<line_number; i++){
                                        final_graph[i]=current_pointer[i];
                                }

                                temp_number=final_graph[line_number-2]->next->value;
                                final_graph[line_number-2]->next->value=-1000;
                                final_graph[line_number-2]=final_graph[line_number-2]->next;
                                if(final_graph[line_number-2]->next==NULL){
                                        final_graph[line_number-2]->next=(item *)malloc(1* sizeof(item));
                                        if(final_graph[line_number-2]==NULL){
                                                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
                                        }
                                        final_graph[line_number-2]=final_graph[line_number-2]->next;
                                        final_graph[line_number-2]->value=temp_number;
                                        final_graph[line_number-2]->next=NULL;
                                }

                                final_graph[line_number-1]->value=-1000;
                                final_graph[line_number-1]->next=NULL;

                                for(i=0; i<line_number; i++){
                                        final_graph[i]=current_pointer[i];
                                }

                                /* --Edw tupwnoume to teliko grafhma pou dhmiourghthhke-- */
                                fprintf(stdout,
"#########################################################################\n");
                                fprintf(stdout, "Final_graph (opou 1000 einai o s kombos -1000 einai o t kombos):
#\n");
                                fprintf(stdout,
"#########################################################################\n");
                                fprintf(stdout, "\n");
                                for(i=0; i<line_number; i++){
                                        while(final_graph[i]!=NULL){
                                                fprintf(stdout, "(%d)---->", final_graph[i]->value);
                                                final_graph[i]=final_graph[i]->next;
                                        }
                                        fprintf(stdout, "NULL\n");
                                }
                                fprintf(stdout, "\n");
                                /* --Mexri edw-- */
                        }
                        else{
                                fprintf(stdout, "??????????  Sto grafhma o kombos me thn etiketa 1 den exei kamia
akmh.\n");
                                abort();
                        }
                }
        }
        /* --Mexri edw-- */
 }

 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }

 for(i=0; i<line_number; i++){
        while(final_graph[i]!=NULL){
                if((final_graph[i]->value>15) && (final_graph[i]->value!=1000) || (final_graph[i]->value<1) &&
(final_graph[i]->value!=-1000)){
                        fprintf(stdout, "??????????  To grafhma exei upostei epithesh stous kombous.\n");
                        fprintf(stdout, "??????????  Yparxei kombos me mh egkurh timh.\n");
                        fprintf(stdout, "??????????  Autos einai o: %d.\n\n", final_graph[i]->value);
                        abort();
                }

                final_graph[i]=final_graph[i]->next;
        }
 }

 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
```

```
  }

  for(i=0; i<line_number; i++){
          while(final_graph[i]!=NULL){
                  if((final_graph[i]->value>maximum_kombos) && (final_graph[i]->value!=1000) && (final_graph[i]-
>value<=15)){
                          maximum_kombos=final_graph[i]->value;
                  }

                  final_graph[i]=final_graph[i]->next;
          }
  }

  if(maximum_kombos-(line_number-2)==1){
          fprintf(stdout, "??????????  Kapoios apo tous kombous den exei ekserxomenes akmes.\n\n");
          abort();
  }
  if(maximum_kombos-(line_number-2)>1){
          fprintf(stdout, "??????????  Kapoioi apo tous kombous den exoun ekserxomenes akmes.\n\n");
          abort();
  }

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou diabazei to teliko grafhma apo to arxeio-- */
int scan_final_graph(){

  /* --Edw einai dhlwseis twn metablhtwn ths scan_final_graph-- */
  int i=0, number=0;
  FILE *infile;
  char *inname="my_file";
  /* --Mexri edw-- */

  /* --Edw anoigei to arxeio pou exei to udatografhma gia anagnwsh-- */
  infile = fopen(inname, "r");
  if(!infile){
          printf("Den mporei na anoiksei to arxeio %s gia diabasma!!!!\n", inname);
          return(0);
  }
  /* --Mexri edw-- */

  /* --Edw kan malloc to current pointer op mas bohtha na guriz opote thel sthn arxh tou final_graph-- */
  current_pointer=(item **)malloc(line_number* sizeof(item*));
  if(current_pointer==NULL){
          fprintf(stdout, "Den egine swsta to malloc!!!!\n");
  }
  for(i=0; i<line_number; i++){
          current_pointer[i]=(item *)malloc(1* sizeof(item));
          if(current_pointer[i]==NULL){
                  fprintf(stdout, "Den egine swsta to malloc!!!!\n");
          }
  }
  fprintf(stdout, "\n");
  /* --Mexri edw-- */

  /* --Edw kanoume to malloc gia ton pinaka tou r.p.g-- */
  final_graph=(item **)malloc(line_number* sizeof(item*));
  if(final_graph==NULL){
          fprintf(stdout, "Den egine swsta to malloc!!!!\n");
  }
  /* --Mexri edw-- */

  /* --Edw antigrafoume ta stoixeia tou arxeiou ston pinaka pou exei to teliko grafhma-- */
  i=0;
  while(1){
          final_graph[i]=(item *)malloc(1* sizeof(item));
          if(final_graph[i]==NULL){
                  fprintf(stdout, "Den egine swsta to malloc!!!!\n");
          }
          if(fscanf(infile, "%d", &number)==EOF){
                  break;
          }
```

```
        final_graph[i]->value=number;
        final_graph[i]->next=NULL;
        current_pointer[i]=final_graph[i];

        while(1){
                fscanf(infile, "%d", &number);
                if(number==0){
                        i++;
                        break;
                }
                final_graph[i]->next=(item *)malloc(1* sizeof(item));
                final_graph[i]=final_graph[i]->next;
                final_graph[i]->value=number;
                final_graph[i]->next=NULL;
        }
 }
 /* --Mexri edw-- */

 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }

 /* --Edw tupwnoume to teliko grafhma pou dhmiourghthhke-- */
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "Final_graph (opou 1000 einai o s kombos -1000 einai o t kombos):     #\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n");
 for(i=0; i<line_number; i++){
        while(final_graph[i]!=NULL){
                fprintf(stdout, "(%d)---->", final_graph[i]->value);
                final_graph[i]=final_graph[i]->next;
        }
        fprintf(stdout, "NULL\n");
 }
 fprintf(stdout, "\n");
 /* --Mexri edw-- */

 /* --Edw elegxoume gia ephtheseis se kombous, akmes h labels-- */
 if(choice==1){
        attack_to_vertexes();
 }
 if(choice==2){
        attack_to_edges();
 }
 if(choice==5){
        attack_to_labels();
 }
 /* --Mexri edw-- */

 fclose(infile);

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou diagrafei tis akmes apo to vi+1 sto vi-- */
void delete_right_edges(){

 /* --Edw einai dhlwseis twn metablhtwn ths delete_right_edges-- */
 int i=0, temp_variable=0;
 /* --Mexri edw-- */

 /* --Edw sbhnoume tis akmes pou theloume-- */
 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }
 for(i=1; i<line_number-1; i++){
        final_graph[i]->next=final_graph[i]->next->next;
 }

 final_graph[0]->next=NULL;
 final_graph[line_number-1]->next=NULL;
 /* --Mexri edw-- */

 for(i=0; i<line_number; i++){
```

```
        final_graph[i]=current_pointer[i];
 }

 /* --Edw tupwnoume to endiameso grafhma pou dhmiourghthhke sbhnontas tis akmes apo to vi+1 sto vi-- */
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "Endiameso grafhma (sbhnontas tis akmes apo to vi+1 sto vi):          #\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n");
 for(i=0; i<line_number; i++){
        while(final_graph[i]!=NULL){
                fprintf(stdout, "(%d)---->", final_graph[i]->value);
                final_graph[i]=final_graph[i]->next;
        }
        fprintf(stdout, "NULL\n");
 }
 fprintf(stdout, "\n");
 /* --Mexri edw-- */

 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }

 /* --Edw ftiaxnoume to grafhma pou dhm anapodogurizontas oles tis enapomhnantes kateuthunomenes akmes-- */
 for(i=0; i<line_number; i++){
        if(final_graph[i]->next!=NULL){
                temp_variable=final_graph[i]->value;
                final_graph[i]->value=final_graph[i]->next->value;
                final_graph[i]->next->value=temp_variable;
        }
 }
 /* --Mexri edw-- */

 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }

 /* --Edw tupwnoume to endiameso grafhma pou dhmiourghthhke anapodogurizntas tis enapomhnantes akmes (dentro)--
*/
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "Endiameso grafhma(anapodogurizntas tis enapomhnantes akmes (dentro)):#\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n");
 for(i=0; i<line_number; i++){
        while(final_graph[i]!=NULL){
                fprintf(stdout, "(%d)---->", final_graph[i]->value);
                final_graph[i]=final_graph[i]->next;
        }
        fprintf(stdout, "NULL\n");
 }
 fprintf(stdout, "\n");
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou elegxei gia epitheseis sthn s.i.p-- */
void attack_to_sip(){
 int i=0, not_ok=0, temp_1=0, temp_2=0, lala_1=0, lala_2=0;

 for(i=0; i<line_number-2; i++){
        temp_1=BP_2[i];
        temp_2=i;

        lala_1=BP_2[temp_1-1];
        lala_2=temp_1-1;

        if((temp_1!=lala_2+1) || (temp_2+1!=lala_1)){
                fprintf(stdout, "??????????  Exei alloiwthei h idiothta ths s.i.p!!!!\n\n");
                abort();
                not_ok=1;
                break;
        }
 }
```

```
 if(not_ok==0){
        is=1;

        //fprintf(stdout, "(");
        //for(i=0; i<line_number-2-1; i++){
        //      fprintf(stdout, "%d, ", BP_2[i]);
        //}
        //fprintf(stdout, "%d)\t", BP_2[line_number-2-1]);
        //fprintf(stdout, "Nai s.i.p!!!!\t");
 }
}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou briskei kai tupwnei thn s.i.p-- */
void find_and_print_sip(){

 /* --Edw einai dhlwseis twn metablhtwn ths delete_right_edges-- */
 int i=0, ii=0, k=0, temp_variable=0, minimum=2000, maximum=0, in=0, *buffer_sip, mphke=0, max_sip=0;
 /* --Mexri edw-- */

 /* --Edw kanoume malloc ton pinaka pou tha baloume thn s.i.p-- */
 buffer_sip=(int *)malloc((line_number-2)* sizeof(int));
 /* --Mexri edw-- */

 /* --Edw arxikopoioume ton buffer_sip-- */
 for(i=0; i<line_number-2; i++){
        buffer_sip[i]=0;
 }
 /* --Mexri edw-- */

 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }

 /* --Edw briskoume thn s.i.p-- */
 temp_variable=final_graph[1]->value;
 while(k<line_number-2){
        minimum=2000;
        maximum=0;
        for(i=1; i<line_number-1; i++){
                if(temp_variable==final_graph[i]->value){
                        if(final_graph[i]->next->value<minimum){
                                minimum=final_graph[i]->next->value;
                                in=1;
                        }
                }
        }
        for(i=0; i<line_number; i++){
                final_graph[i]=current_pointer[i];
        }
        temp_variable=minimum;
        if(in==1){
                buffer_sip[k]=temp_variable;

                for(i=0; i<line_number-2; i++){
                        if(max_sip<buffer_sip[i]){
                                max_sip=buffer_sip[i];
                        }
                }

                k++;
                in=0;
                for(i=1; i<line_number-1; i++){
                        if(final_graph[i]->next->value==temp_variable){
                                final_graph[i]->value=2000;
                                final_graph[i]->next->value=2000;
                        }
                }
                for(i=0; i<line_number; i++){
                        final_graph[i]=current_pointer[i];
                }
        }
        else{
                for(i=1; i<line_number-1; i++){
```

```
                        if(final_graph[i]->value==max_sip){
                                maximum=final_graph[i]->value;
                                mphke=1;
                        }
                }

                if(mphke==0){
                        for(i=1; i<line_number-1; i++){
                                if((final_graph[i]->value>maximum) && (final_graph[i]->value!=2000)){
                                        maximum=final_graph[i]->value;
                                }
                        }
                }

                temp_variable=maximum;
                mphke=0;
                for(i=0; i<line_number; i++){
                        final_graph[i]=current_pointer[i];
                }
        }
 }
 /* --Mexri edw-- */

 /* --Edw tupwnoume thn s.i.p pou brikame-- */
 fprintf(stdout, "##############################################################################\n");
 fprintf(stdout, "S.i.p:                                                                      #\n");
 fprintf(stdout, "##############################################################################\n");
 fprintf(stdout, "\n");
 fprintf(stdout, "------\n");
 fprintf(stdout, "(");
 for(i=0; i<line_number-2; i++){
        if(i!=line_number-3){
                fprintf(stdout, "%d, ", buffer_sip[i]);
        }
        else{
                fprintf(stdout, "%d)\n", buffer_sip[i]);
        }
 }
 fprintf(stdout, "------\n");
 /* --Mexri edw-- */

 /* ---------Edw arxikopoioume ton BP_2_temp, o opoios einai bohthitikos kai exei thn s.i.p-- */
 for(i=0; i<100; i++){
        BP_2_temp[i]=0;
 }
 /* --Mexri edw-- */

 /* --Edw antigrafoume thn s.i.p pou edwse h prwth diadikasia se enan bohthhtiko pinaka BP_2_temp-- */
 for(i=0; i<line_number-2; i++){
        BP_2_temp[i]=buffer_sip[i];
 }
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw h sunarthsh dhmiourgei ta zeugaria apo thn s.i.p-- */
void sip(){
 int i=0, current_number=0;

 /* --Edw antigrafoume ton bohthitiko pinaka BP_2_temp ston BP_2 kanontas prwta malloc ston BP_2-- */
 BP_2=(int *)malloc((line_number-2)*sizeof(int));
 if(BP_2==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<line_number-2; i++){
        BP_2[i]=BP_2_temp[i];
 }
 /* --Mexri edw-- */

 /* --Edw elegxoume gia ephtheseis sthn s.i.p-- */
 if(choice==3){
        attack_to_sip();
 }
 /* --Mexri edw-- */
```

```
}
/* --Mexri edw-- */



/* --Edw h sunarthsh dhmiourgei ta zeugaria apo thn s.i.p-- */
void couples(){

 /* --Edw einai oi metablhtes ths sunarthshs couples-- */
 int i=0, j=0, mhkos=0, temp_variable=0, jeugos=0;
 float mhkoss=0.0;
 /* --Mexri edw-- */

 /* --Edw kanoume malloc gia ton pinaka pou tha exei ta zeugaria-- */
 mhkos=(2*(line_number-2))+1;
 mhkoss=mhkos/2.0;
 mhkos=mhkos/2;

 /* --Edw kanoume to malloc gia ton artio arithmo apo noumera kai sthn sunexeia kanoume arxikopihsh-- */
 if((float)mhkos==mhkoss){
        COUPLES=(int **)malloc((line_number-2)*sizeof(int *));
                if(COUPLES==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
                for(i=0; i<line_number-2; i++){
                COUPLES[i]=(int *)malloc(2*sizeof(int));
                        if(COUPLES[i]==NULL){
                        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
                }
        }
        for(i=0; i<line_number-2; i++){
                for(j=0; j<2; j++){
                        COUPLES[i][j]=0;
                }
        }

        /* --Edw dhmiourgoume ta epithumhta zeugaria kai ta tupwnoume-- */
        for(i=0; i<line_number-2; i++){
                temp_variable=BP_2[i];
                if((BP_2[i]!=0) && (BP_2[temp_variable-1]!=0)){
                        COUPLES[jeugos][0]=BP_2[temp_variable-1];
                        COUPLES[jeugos][1]=BP_2[i];
                        jeugos++;
                        BP_2[i]=0;
                        BP_2[temp_variable-1]=0;
                }
        }
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "Zeugaria:                                                              #\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
        for(i=0; i<(int)((line_number-2)/2); i++){
                fprintf(stdout, "(%d, %d)\n", COUPLES[i][0], COUPLES[i][1]);
        }
        fprintf(stdout, "\n");
        /* --Mexri edw-- */

 }
 /* --Mexri edw-- */

 /* --Mexri edw-- */

 /* --Edw kanoume to malloc gia ton peritto arithmo apo noumera kai sthn sunexeia kanoume arxikopihsh-- */
 else{
        COUPLES=(int **)malloc((line_number-1)*sizeof(int *));
                if(COUPLES==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
                for(i=0; i<(line_number-1); i++){
                COUPLES[i]=(int *)malloc(2*sizeof(int));
                        if(COUPLES[i]==NULL){
                        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
                }
        }
        for(i=0; i<(line_number-1); i++){
```

```
                for(j=0; j<2; j++){
                        COUPLES[i][j]=0;
                }
        }

        /* --Edw dhmiourgoume ta epithumhta zeugaria kai ta tupwnoume-- */
        for(i=0; i<line_number-2; i++){
                temp_variable=BP_2[i];
                if((BP_2[i]!=0) && (BP_2[temp_variable-1]!=0)){
                        COUPLES[jeugos][0]=BP_2[temp_variable-1];
                        COUPLES[jeugos][1]=BP_2[i];
                        jeugos++;
                        BP_2[i]=0;
                        BP_2[temp_variable-1]=0;
                }
        }
        fprintf(stdout, "##############################################################################\n");
        fprintf(stdout, "Zeugaria:                                                                   #\n");
        fprintf(stdout, "##############################################################################\n");
        fprintf(stdout, "\n");
        for(i=0; i<(int)((line_number-2)/2)+1; i++){
                fprintf(stdout, "(%d, %d)\n", COUPLES[i][0], COUPLES[i][1]);
        }
        fprintf(stdout, "\n");
        /* --Mexri edw-- */

 }
 /* --Mexri edw-- */

 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou elegxei gia epitheseis sthn bp-- */
void attack_to_bp(){
 int i=0, allagh_sth_monotonia=0, bad=0, ok=0;

 while(i<(line_number-2)/2){
        if(COUPLES[i][0]==COUPLES[i][1]){
                ok++;
        }
        i++;
 }
 if(ok>1){
        bad=1;
 }


 i=0;

 if(bad==0){
        while((i<line_number-2-1) && (allagh_sth_monotonia==0)){
                if(BP[i]>BP[i+1]){
                        allagh_sth_monotonia=1;
                        break;
                }
                i++;
        }

        while((i<line_number-2-1) && (allagh_sth_monotonia==1)){
                if(BP[i]<BP[i+1]){
                        bad=1;
                        break;
                }
                i++;
        }

        if(bad==1){
                fprintf(stdout, "??????????  Exei alloiwthei h idiothta ths b.p!!!!\n\n");
                abort();
        }

 }
```

```
 if(is==1){
        if(bad==1){
                fprintf(stdout, "??????????  Exei alloiwthei h idiothta ths b.p!!!!\n\n");
                abort();
        }
        else{
                //fprintf(stdout, "Nai b.p!!!!\n");
                //sip_bp++;
        }
 }
}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou dhmiourgei thn bitoc permutation apo ta zeugaria-- */
void bitonic_p(){

 /* --Edw einai oi metablhtes ths sunarthshs bitonic_p-- */
 int i=0, j=0, mhkos=0, temp_variable=0, temp_variable_2=0;
 float mhkoss=0.0;
 /* --Mexri edw-- */

 /* --Edw kanoume malloc gia ton pinaka pou ths bitonic permutation-- */
 mhkos=(2*(line_number-2))+1;
 mhkoss=mhkos/2.0;
 mhkos=mhkos/2;

 /* --Edw kanoume to malloc gia ton artio arithmo apo noumera kai sthn sunexeia kanoume-- */
 /* --arxikopihsh kai gemisma-- */
 if((float)mhkos==mhkoss){
        BP=(int*)malloc((line_number-2)* sizeof(int));
        if(BP==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
        for(i=0; i<(line_number-2); i++){
                BP[i]=0;
        }
        for(i=0; i<(int)((line_number-2)/2); i++){
                BP[i]=COUPLES[i][1];
                BP[((line_number-2)-1)-i]=COUPLES[i][0];
        }
        BP[(int)((line_number-2)/2)]=COUPLES[(int)((line_number-2)/2)][0];

        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "BP:                                                                    #\n");
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
        fprintf(stdout, "(");
        for(i=0; i<(line_number-2)-1; i++){
                fprintf(stdout, "%d,  ", BP[i]);
        }
        fprintf(stdout, "%d)", BP[line_number-3]);
        fprintf(stdout, "\n\n");
 }
 /* --Mexri edw-- */

 /* --Edw kanoume to malloc gia ton peritto arithmo apo noumera kai sthn sunexeia kanoume-- */
 /* --arxikopihsh kai gemisma-- */
 else{
        BP=(int*)malloc((line_number-2)* sizeof(int));
        if(BP==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
        for(i=0; i<(line_number-2); i++){
                BP[i]=0;
        }
        for(i=0; i<(int)((line_number-2)/2); i++){
                BP[i]=COUPLES[i][1];
                BP[((line_number-2)-1)-i]=COUPLES[i][0];
        }
        BP[(int)((line_number-2)/2)]=COUPLES[(int)((line_number-2)/2)][0];

        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "BP:                                                                    #\n");
```

```
        fprintf(stdout, "#########################################################################\n");
        fprintf(stdout, "\n");
        fprintf(stdout, "(");
        for(i=0; i<(line_number-2)-1; i++){
                fprintf(stdout, "%d,  ", BP[i]);
        }
        fprintf(stdout, "%d)", BP[line_number-3]);
        fprintf(stdout, "\n\n");
 }
 /* --Mexri edw-- */

 /* --Mexri edw-- */

 /* --Edw elegxoume gia ephtheseis sthn b.p-- */
 if(choice==4){
        attack_to_bp();
 }
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou dhmiourgei tous X kai Y pinakes-- */
void X_and_Y(){

 /* --Edw einai oi metablhtes ths sunarthshs X_and_Y-- */
 int i=0, ii=0;
 /* --Mexri edw-- */

 /* --Edw prwta briskoume poses theseis xreiazontai autoi oi pinakes kai meta kanoume to malloc-- */
 while(BP[i]<BP[i+1]){
        sum_1++;
        i++;
 }
 sum_2=(line_number-2)-sum_1;
 /* --Mexri edw-- */

 /* --Edw kanoume ta duo malloc pou xreiazontai stous X kai Y pinakes kai tis arxikopoihseis autwn--*/
 X=(int*)malloc(sum_1* sizeof(int));
 if(X==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<sum_1; i++){
        X[i]=0;
 }
 Y=(int*)malloc(sum_2* sizeof(int));
 if(Y==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<sum_2; i++){
        Y[i]=0;
 }
 /* --Mexri edw-- */

 /* --Edw gemizoume tous X kai Y pinakes-- */
 for(i=0; i<sum_1; i++){
        X[i]=BP[i];
 }
 for(i=sum_1; i<(line_number-2); i++){
        Y[ii]=BP[i];
        ii++;
 }
 /* --Mexri edw-- */

 /* --Edw tupwnoume tous pinakes X kai Y-- */
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "X:                                                                      #\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n(");
 for(i=0; i<sum_1-1; i++){
        fprintf(stdout, "%d, ", X[i]);
 }
 fprintf(stdout, "%d)\n\n", X[sum_1-1]);
 fprintf(stdout, "#########################################################################\n");
```

```
 fprintf(stdout, "Y:                                                                    #\n");
 fprintf(stdout, "#######################################################################\n");
 fprintf(stdout, "\n(");
 for(i=0; i<sum_2-1; i++){
        fprintf(stdout, "%d, ", Y[i]);
 }
 fprintf(stdout, "%d)\n\n", Y[sum_2-1]);
 /* --Mexri edw-- */

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou ftiaxnei tous B_tonos kai B_asteraki-- */
void B_tonos_and_B_asteraki(){

 /* --Edw einai oi metablhtes ths sunarthshs B_tonos_and_B_asteraki-- */
 int i=0, temp_variable=0;
 /* --Mexri edw-- */

 /* --Edw kanoume malloc gia ton B_tonos kai ton B_asteraki-- */
 B_tonos=(int*)malloc((line_number-2)* sizeof(int));
 if(B_tonos==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 B_asteraki=(int*)malloc((line_number-2)* sizeof(int));
 if(B_asteraki==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 /* --Mexri edw-- */

 /* --Edw arxikopoioume ton B_tonos kai ton B_asteraki-- */
 for(i=0; i<line_number-2; i++){
        B_tonos[i]=0;
 }
 for(i=0; i<line_number-2; i++){
        B_asteraki[i]=0;
 }
 /* --Mexri edw-- */

 /* --Edw gemizoume me thn seira ton B_asteraki kai ton B_tonos kai tous ektupwnoume-- */
 for(i=0; i<sum_1; i++){
        temp_variable=X[i]-1;
        B_asteraki[temp_variable]=0;
 }
 for(i=0; i<sum_2; i++){
        temp_variable=Y[i]-1;
        B_asteraki[temp_variable]=1;
 }
 fprintf(stdout, "#######################################################################\n");
 fprintf(stdout, "B_asteraki:                                                           #\n");
 fprintf(stdout, "#######################################################################\n");
 fprintf(stdout, "\n(");
 for(i=0; i<line_number-3; i++){
        fprintf(stdout, "%d, ", B_asteraki[i]);
 }
 fprintf(stdout, "%d)\n\n", B_asteraki[line_number-3]);
 for(i=0; i<line_number-2; i++){
        if(B_asteraki[i]==1){
                B_tonos[i]=0;
        }
        else{
                B_tonos[i]=1;
        }
 }
 fprintf(stdout, "#######################################################################\n");
 fprintf(stdout, "B_tonos:                                                              #\n");
 fprintf(stdout, "#######################################################################\n");
 fprintf(stdout, "\n(");
 for(i=0; i<line_number-3; i++){
        fprintf(stdout, "%d, ", B_tonos[i]);
 }
 fprintf(stdout, "%d)\n\n", B_tonos[line_number-3]);
 /* --Mexri edw-- */
```

```
}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh upswshs se dunamh-- */
int power(int base, int n){

 /* --Edw einai oi metablhtes ths sunarthshs power-- */
 int i=0, p=0;
 /* --Mexri edw-- */

 p = 1;
        for(i=1; i<=n; ++i){
        p*=base;
 }
        return(p);
}
/* --Mexri edw-- */



/* --Edw kataskeuazoume to udatoshma apo duadikh morfh se dekadikh-- */
void binary_to_w(){

 /* --Edw einai oi metablhtes ths sunarthshs binary_to_w-- */
 int i=0, ii=0, pow=0;
 /* --Mexri edw-- */

 for(i=(line_number-2)-2; i>=(int)((line_number-2)/2); i--){
        if(B_tonos[i]==1){
                w=w+power(2, pow);
        }
        pow++;
 }
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "Briskomaste sto sunolo N%d twn fusikwn arithmwn!!!!!!!!!!            #\n", line_number-2);
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "To udatoshma einai:                                                    #\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n");
 fprintf(stdout, "------\n");
 fprintf(stdout, "(%d)\n", w);
 fprintf(stdout, "------\n");
 fprintf(stdout, "\n");
}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh main-- */
int main(){

 /* --Edw einai dhlwseis twn metablhtwn ths main-- */
 /* item *current=NULL, *start=NULL; */
 int i=0;
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh gia thn prosfwnhsh ths diadikasias metatrophs enos r.p.g se mia s.i.p-- */
 prosfwnhsh();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh h opoia anoigei to arxeio pou emperiexei to udatografhma-- */
 open_file_for_read();
 /* --Mexri edw-- */

 /* --Edw einai hsunarthsh me to menu twn epithesewn-- */
 menu();
 /* --Mexri edw-- */

 /* --Edw einai h klhsh ths sunarthshs opou diabazei to teliko grafhma-- */
 scan_final_graph();
```

```
 /* --Mexri edw-- */

 /* --Edw einai h klhsh ths sunarthshs opou diagrafei tis akmes apo to vi+1 sto vi-- */
 delete_right_edges();
 /* --Mexri edw-- */

 /* --Edw einai h klhsh ths sunarthshs opou briskei kai tupwnei thn s.i.p-- */
 find_and_print_sip();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh gia to telos ths diadikasias metatrophw mias enos r.p.g se mia s.i.p-- */
 telos();
 /* --Mexri edw-- */

 /* --Edw einai mia aplh upolopoihsh gia to pws leitourgei mia apla sundedemenh lista-- */
 /*current=(item *)malloc(sizeof(item));
 current->value=1;
 current->next=NULL;
 start=current;

 for(i=2;i<=10;i++) {
        current->next=(item *)malloc(sizeof(item));
        current=current->next;
               current->value=i;
               current->next=NULL;
 }
 current=start;
        while(current!=NULL) {
               fprintf(stdout, "%d---->", current->value);
               current=current->next ;
        }
 fprintf(stdout, "NULL\n");*/
 /* --Mexri edw-- */

 flow=2;

 /* --Edw kaloume thn sunarthsh gia thn prosfwnhsh ths diadikasias metatrophs enos r.p.g se mia s.i.p-- */
 prosfwnhsh();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou diabazei thn s.i.p-- */
 sip();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou ftiaxnei ta zeugaria-- */
 couples();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou ftiaxnei thn bitonic permutation-- */
 bitonic_p();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou ftiaxnei tous pinakes X kai Y-- */
 X_and_Y();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou dhmiourgei ton pinaka B_tonos kai ton B_asteraki-- */
 B_tonos_and_B_asteraki();
 /* --Mexri edw-- */

 /* --Edw kaloume thn sunarthsh pou kata ton fusiko arithmo w (udatoshma) apo thn duadiakh tou morfh-- */
 binary_to_w();
 /* --Mexri edw-- */

 is=0;

 /* --Edw kaloume thn sunarthsh gia to telos ths diadikasias metatrophw mias enos r.p.g se mia s.i.p-- */
 telos();
 /* --Mexri edw-- */

 /* --Edw kanoume tis apodesmeuseis mnhmhs stous deiktes-- */

 /* --Mexri edw-- */

}
/* --Mexri edw-- */
```

Παράρτημα 3

Κώδικας Ενσωμάτωσης Υδατογραφήματος σε Λογισμικό

- Στον παρακάτω κώδικα, ένα μεταθετικό αναγώγιμο γράφημα (υδατογράφημα) μετατρέπεται σε μορφή κώδικα.
- Ο κώδικας αυτός είναι έτοιμος προς ενσωμάτωση σε ένα ή περισσότερα σημεία εντός του προς υδατογράφηση λογισμικού.

```
/* --Grafthke apo Iwannhs Xionhs A.M:1430-- */



/* --Edw einai oi dhlwseis twn includes kai twn defines-- */
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
/* --Mexri edw-- */



/* --Edw einai oi dhlwseis twn sunarthshewn-- */
void lala_s();
void lala_15();
void lala_14();
void lala_13();
void lala_12();
void lala_11();
void lala_10();
void lala_9();
void lala_8();
void lala_7();
void lala_6();
void lala_5();
void lala_4();
void lala_3();
void lala_2();
void lala_1();
void lala_t();
void lala_null();
void start_proof();
void end_proof();
int scan_final_graph();
int open_file_for_read();
void call();
/* --Mexri edw-- */



/* --Edw einai h dhlwshs ths apla sundedemenhs listas-- */
struct list_elements{
  int value;
  struct list_elements *next;
};
typedef struct list_elements item;
/* --Mexri edw-- */



/* --Edw einai oi dhlwseis twn global metablhtwn-- */
int length=0, pos=0, pointer_for_next_block=1000, line_number=0, trexon_thesh=1000;
```

```
item **final_graph, **current_pointer;
/* --Mexri edw-- */



/* --Edw anoigei to arxeio pou exei to udatografhma-- */
int open_file_for_read(){
 char *inname="my_file";
 FILE *infile;
 char line_buffer[BUFSIZ];

 infile = fopen(inname, "r");
 if(!infile){
        printf("Den mporei na anoiksei to arxeio %s gia diabasma!!!!\n", inname);
        return(0);
 }
 printf("To arxeio %s anoikse gia diabasma!!!!\n\n", inname);

 line_number=0;
 while (fgets(line_buffer, sizeof(line_buffer), infile)) {
        ++line_number;
        printf("%4d: %s", line_number, line_buffer);
 }
 printf("\nO sunolikos arithmos apo grammes tou arxeiou %s einai isos me %d\n\n", inname, line_number);
 fclose(infile);

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou diabazei to teliko grafhma-- */
int scan_final_graph(){

 /* --Edw einai dhlwseis twn metablhtwn ths scan_final_graph-- */
 int i=0, number=0;
 FILE *infile;
 char *inname="my_file";
 /* --Mexri edw-- */

 /* --Edw anoigei to arxeio pou exei to udatografhma gia anagnwsh-- */
 infile = fopen(inname, "r");
 if(!infile){
        printf("Den mporei na anoiksei to arxeio %s gia diabasma!!!!\n", inname);
        return(0);
 }
 /* --Mexri edw-- */

 /* --Edw kanoume malloc to current pointer opoios tha mas bohtha na gurizoume opote theloume-- */
 /* --sthn arxh tou final_graph-- */
 current_pointer=(item **)malloc(line_number* sizeof(item*));
 if(current_pointer==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<line_number; i++){
        current_pointer[i]=(item *)malloc(1* sizeof(item));
        if(current_pointer[i]==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
 }
 fprintf(stdout, "\n");
 /* --Mexri edw-- */

 /* --Edw kanoume to malloc gia ton pinaka tou r.p.g-- */
 final_graph=(item **)malloc(line_number* sizeof(item*));
 if(final_graph==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 /* --Mexri edw-- */

 /* --Edw antigrafoume ta stoixeia tou arxeiou ston pinaka pou exei to teliko grafhma-- */
 i=0;
 while(1){
        final_graph[i]=(item *)malloc(1* sizeof(item));
        if(final_graph[i]==NULL){
```

```
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
        if(fscanf(infile, "%d", &number)==EOF){
                break;
        }

        final_graph[i]->value=number;
        final_graph[i]->next=NULL;
        current_pointer[i]=final_graph[i];

        while(1){
                fscanf(infile, "%d", &number);
                if(number==0){
                        i++;
                        break;
                }
                final_graph[i]->next=(item *)malloc(1* sizeof(item));
                final_graph[i]=final_graph[i]->next;
                final_graph[i]->value=number;
                final_graph[i]->next=NULL;
        }
 }
 /* --Mexri edw-- */

 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }

 /* --Edw tupwnoume to teliko grafhma pou dhmiourghthhke-- */
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "Final_graph (opou 1000 einai o s kombos -1000 einai o t kombos):     #\n");
 fprintf(stdout, "#########################################################################\n");
 fprintf(stdout, "\n");
 for(i=0; i<line_number; i++){
        while(final_graph[i]!=NULL){
                fprintf(stdout, "(%d)---->", final_graph[i]->value);
                final_graph[i]=final_graph[i]->next;
        }
        fprintf(stdout, "NULL\n");
 }
 fprintf(stdout, "\n");
 /* --Mexri edw-- */

 for(i=0; i<line_number; i++){
        final_graph[i]=current_pointer[i];
 }

 fclose(infile);

}
/* --Mexri edw-- */



/* --Edw einai h sunarthsh pou kanei tous elegxous gia to poia tha einai h epomenh sunarthsh-- */
/* --pou tha kalestei-- */
void call(){

 if(pointer_for_next_block==1000){
                lala_s();
        }
        else if(pointer_for_next_block==1){
                lala_1();
        }
        else if(pointer_for_next_block==2){
                lala_2();
        }
        else if(pointer_for_next_block==3){
                lala_3();
        }
        else if(pointer_for_next_block==4){
                lala_4();
        }
        else if(pointer_for_next_block==5){
                lala_5();
        }
```

```
        else if(pointer_for_next_block==6){
                lala_6();
        }
        else if(pointer_for_next_block==7){
                lala_7();
        }
        else if(pointer_for_next_block==8){
                lala_8();
        }
        else if(pointer_for_next_block==9){
                lala_9();
        }
        else if(pointer_for_next_block==10){
                lala_10();
        }
        else if(pointer_for_next_block==11){
                lala_11();
        }
        else if(pointer_for_next_block==12){
                lala_12();
        }
        else if(pointer_for_next_block==13){
                lala_13();
        }
        else if(pointer_for_next_block==14){
                lala_14();
        }
        else if(pointer_for_next_block==15){
                lala_15();
        }
        else if(pointer_for_next_block==-1000){
                lala_t();
        }
}
/* --Mexri edw-- */



/* --Edw einai oi sunarthseis pou dhmiourgountai mesa apo to final graph-- */
void lala_s(){
 printf("Ok lala_s!!!!  ->  ");
 sleep(1);

 if(trexon_thesh==1000){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh=line_number-2;
                lala_null();
        }
 }
}

void lala_15(){
 printf("Ok lala_15!!!!  ->  ");
 sleep(1);

 if(trexon_thesh==15){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
 }
}

void lala_14(){
 printf("Ok lala_14!!!!  ->  ");
 sleep(1);
```

```
 if(trexon_thesh==14){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
 }
}

void lala_13(){
 printf("Ok lala_13!!!!  ->  ");
 sleep(1);

 if(trexon_thesh==13){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
 }
}

void lala_12(){
 printf("Ok lala_12!!!!  ->  ");
 sleep(1);

 if(trexon_thesh==12){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
 }
}

void lala_11(){
 printf("Ok lala_11!!!!  ->  ");
 sleep(1);

 if(trexon_thesh==11){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
 }
}

void lala_10(){
 printf("Ok lala_10!!!!  ->  ");
 sleep(1);

 if(trexon_thesh==10){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
```

```
                lala_null();
        }
  }
}

void lala_9(){
  printf("Ok lala_9!!!!  ->  ");
  sleep(1);

  if(trexon_thesh==9){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
  }
}

void lala_8(){
  printf("Ok lala_8!!!!  ->  ");
  sleep(1);

  if(trexon_thesh==8){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
  }
}

void lala_7(){
  printf("Ok lala_7!!!!  ->  ");
  sleep(1);

  if(trexon_thesh==7){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
  }
}

void lala_6(){
  printf("Ok lala_6!!!!  ->  ");
  sleep(1);

  if(trexon_thesh==6){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
  }
}

void lala_5(){
  printf("Ok lala_5!!!!  ->  ");
  sleep(1);
```

```
  if(trexon_thesh==5){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
  }
}

void lala_4(){
  printf("Ok lala_4!!!!  ->  ");
  sleep(1);

  if(trexon_thesh==4){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
  }
}

void lala_3(){
  printf("Ok lala_3!!!!  ->  ");
  sleep(1);

  if(trexon_thesh==3){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
  }
}

void lala_2(){
  printf("Ok lala_2!!!!  ->  ");
  sleep(1);

  if(trexon_thesh==2){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh--;
                lala_null();
        }
  }
}

void lala_1(){
  printf("Ok lala_1!!!!  ->  ");
  sleep(1);

  if(trexon_thesh==1){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                trexon_thesh=-1000;
                lala_null();
```

```
        }
  }
}

void lala_t(){
  printf("Ok lala_t!!!!  ->  ");
  sleep(1);

  if(trexon_thesh==-1000){
        while(final_graph[pos]->next!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                pointer_for_next_block = final_graph[pos]->value;
                call();
        }
        if(final_graph[pos]->next==NULL){
                lala_null();
        }
        trexon_thesh--;
  }
}

void lala_null(){
  printf("Null!!!!\n");
  sleep(1);
  pos++;
  if(pos>=line_number){
        end_proof();
  }
  else{
        pointer_for_next_block=final_graph[pos]->value;
        call();
  }
}
/* --Mexri edw-- */



void start_proof(){
  printf("\nArxh apodeikshs gnhsiothtas!!!!\n\n");
  lala_s();
}



void end_proof(){
  printf("\nTelos apodeikshs gnhsiothtas!!!!\n\n");
}



int main(){
  open_file_for_read();
  scan_final_graph();

  start_proof();
}
```

Παράρτημα 4

Κώδικας για Έλεγχο Πολλαπλών Μεταθέσεων για s.i.p και b.p

- Στον παρακάτω κώδικα, ελέγχονται επαναληπτικά μεταθέσεις προκειμένου να διαπιστωθεί αν είναι απλές μεταθέσεις, αν είναι αυτοαναστρέφουσες μεταθέσεις, αν είναι bitonic permutations ή αν είναι και αυτοαναστρέφουσες και bitonic..

```
/* --Grafthke apo Iwannhs Xionhs A.M:1430-- */
```

```
/* --Edw einai oi dhlwseis twn includes kai twn defines-- */
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
/* --Mexri edw-- */
```

```
/* --Edw einai oi dhlwseis twn global metablhtwn-- */
int BP_2_temp[100], *BP_2, length=0, **COUPLES, *BP, *X, *Y, *B_tonos, *B_asteraki, sum_1=1;
int sum_2=0, is=0, sip_bp=0;
int w=0; /* --!!!!!!!!!!Timh udatografhshs!!!!!!!!!!-- */
char *inname="all_permutations_sip_11_8", *inname_2="results_of_all_permutations_sip_11_8";
FILE *infile, *write_to_file;
/* --Mexri edw-- */
```

```
/* --Edw einai h sunarthsh pou elegxei gia epitheseis sthn s.i.p-- */
```

```
void attack_to_sip(){
  int i=0, not_ok=0, temp_1=0, temp_2=0, lala_1=0, lala_2=0;

  for(i=0; i<length; i++){
          temp_1=BP_2[i];
          temp_2=i;

          lala_1=BP_2[temp_1-1];
          lala_2=temp_1-1;

          if((temp_1!=lala_2+1) || (temp_2+1!=lala_1)){
                  //fprintf(write_to_file, "Oxi s.i.p!!!!\t");
                  not_ok=1;
                  break;
          }
  }

  if(not_ok==0){
          is=1;

          fprintf(write_to_file, "(");
          for(i=0; i<length-1; i++){
                  fprintf(write_to_file, "%d, ", BP_2[i]);
          }
          fprintf(write_to_file, "%d)\t", BP_2[length-1]);
          fprintf(write_to_file, "Nai s.i.p!!!!\t");
  }
}
/* --Mexri edw-- */




/* --Edw h sunarthsh dhmiourgei ta zeugaria apo thn s.i.p-- */
```

```
void sip(){
  int i=0, current_number=0;

  /* ---------Edw arxikopoioume ton BP_2_temp, o opoios einai bohthitikos kai exei thn s.i.p-- */
  for(i=0; i<100; i++){
          BP_2_temp[i]=0;
  }
  /* --Mexri edw-- */

  /* --Edw diabazoume apo to plhktrologio thn s.i.p kai thn apothkeuoume prwta se ena bohthitiko-- */
  /* --pinaka, ton BP_2_temp-- */
  //fprintf(stdout, "#########################################################################\n");
  //fprintf(stdout, "Plhktrologiste thn s.i.p dinontas apo 7 ews 15 arithmous apo to 1 ews to
  //15 (dinontas to 0 gia na teliwsei):\n");
  //fprintf(stdout, "#########################################################################\n");
  //fprintf(stdout, "\n");
  i=0;
  fscanf(infile, "%d", &current_number);
  while(current_number!=-100){
          BP_2_temp[i]=current_number;

          fscanf(infile, "%d", &current_number);
          i++;
          length++;
  }
  /* --Mexri edw-- */

  /* --Edw antigrafoume ton bohthitiko pinaka BP_2_temp ston BP_2 kanontas prwta malloc ston BP_2-- */
  BP_2=(int *)malloc((length)*sizeof(int));
  if(BP_2==NULL){
          fprintf(stdout, "Den egine swsta to malloc!!!!\n");
  }
  for(i=0; i<length; i++){
          BP_2[i]=BP_2_temp[i];
```

```
 }
 /* --Mexri edw-- */

 attack_to_sip();
}
/* --Mexri edw-- */




/* --Edw h sunarthsh dhmiourgei ta zeugaria apo thn s.i.p-- */
void couples(){

 /* --Edw einai oi metablhtes ths sunarthshs couples-- */
 int i=0, j=0, mhkos=0, temp_variable=0, jeugos=0;
 float mhkoss=0.0;
 /* --Mexri edw-- */

 /* --Edw kanoume malloc gia ton pinaka pou tha exei ta zeugaria-- */
 mhkos=(2*length)+1;
 mhkoss=mhkos/2.0;
 mhkos=mhkos/2;

 /* --Edw kanoume to malloc gia ton artio arithmo apo noumera kai sthn sunexeia kanoume arxikopihsh-- */
 if((float)mhkos==mhkoss){
        COUPLES=(int **)malloc(length* sizeof(int *));
                if(COUPLES==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
                for(i=0; i<length; i++){
                COUPLES[i]=(int *)malloc(2* sizeof(int));
                        if(COUPLES[i]==NULL){
                        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
                }
```

```
	}
	for(i=0; i<length; i++){
		for(j=0; j<2; j++){
			COUPLES[i][j]=0;
		}
	}

	/* --Edw dhmiourgoume ta epithumhta zeugaria kai ta tupwnoume-- */
	for(i=0; i<length; i++){
		temp_variable=BP_2[i];
		if((BP_2[i]!=0) && (BP_2[temp_variable-1]!=0)){
			COUPLES[jeugos][0]=BP_2[temp_variable-1];
			COUPLES[jeugos][1]=BP_2[i];
			jeugos++;
			BP_2[i]=0;
			BP_2[temp_variable-1]=0;
		}
	}
	//fprintf(stdout, "#########################################################################\n");
	//fprintf(stdout, "Ta zeugaria pou dhmiourghsame einai:\n");
	//fprintf(stdout, "#########################################################################\n");
	//fprintf(stdout, "\n");
	//for(i=0; i<(int)(length/2); i++){
	//	fprintf(stdout, "%d\t%d\n", COUPLES[i][0], COUPLES[i][1]);
	//}
	//fprintf(stdout, "\n");
	/* --Mexri edw-- */

}
/* --Mexri edw-- */

/* --Mexri edw-- */

/* --Edw kanoume to malloc gia ton peritto arithmo apo noumera kai sthn sunexeia kanoume arxikopihsh-- */
```

```
else{
        COUPLES=(int **)malloc((length+1)* sizeof(int *));
                if(COUPLES==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
                for(i=0; i<(length+1); i++){
                COUPLES[i]=(int *)malloc(2* sizeof(int));
                        if(COUPLES[i]==NULL){
                        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
                }
        }
        for(i=0; i<(length+1); i++){
                for(j=0; j<2; j++){
                        COUPLES[i][j]=0;
                }
        }

        /* --Edw dhmiourgoume ta epithumhta zeugaria kai ta tupwnoume-- */
        for(i=0; i<length; i++){
                temp_variable=BP_2[i];
                if((BP_2[i]!=0) && (BP_2[temp_variable-1]!=0)){
                        COUPLES[jeugos][0]=BP_2[temp_variable-1];
                        COUPLES[jeugos][1]=BP_2[i];
                        jeugos++;
                        BP_2[i]=0;
                        BP_2[temp_variable-1]=0;
                }
        }
        //fprintf(stdout, "#########################################################################\n");
        //fprintf(stdout, "Ta zeugaria pou dhmiourghsame einai:\n");
        //fprintf(stdout, "#########################################################################\n");
        //fprintf(stdout, "\n");
        //for(i=0; i<(int)(length/2)+1; i++){
        //      fprintf(stdout, "%d\t%d\n", COUPLES[i][0], COUPLES[i][1]);
```

```
        //}
        //fprintf(stdout, "\n");
        /* --Mexri edw-- */

 }
 /* --Mexri edw-- */

 /* --Mexri edw-- */

}
/* --Mexri edw-- */




/* --Edw einai h sunarthsh pou elegxei gia epitheseis sthn bp-- */
void attack_to_bp(){
 int i=0, allagh_sth_monotonia=0, bad=0, ok=0;

 while(i<length/2){
        if(COUPLES[i][0]==COUPLES[i][1]){
                ok++;
        }
        i++;
 }
 if(ok>1){
        bad=1;
 }


 i=0;

 if(bad==0){
        while((i<length-1) && (allagh_sth_monotonia==0)){
```

```
                if(BP[i]>BP[i+1]){
                        allagh_sth_monotonia=1;
                        break;
                }
                i++;
        }

        while((i<length-1) && (allagh_sth_monotonia==1)){
                if(BP[i]<BP[i+1]){
                        bad=1;
                        break;
                }
                i++;
        }
 }

 if(is==1){
        if(bad==1){
                fprintf(write_to_file, "Oxi b.p!!!!\n");
                //abort();
        }
        else{
                fprintf(write_to_file, "Nai b.p!!!!\n");
                sip_bp++;
        }
 }
}
/* --Mexri edw-- */




/* --Edw einai h sunarthsh pou dhmiourgei thn bitoc permutation apo ta zeugaria-- */
void bitonic_p(){
```

```
/* --Edw einai oi metablhtes ths sunarthshs bitonic_p-- */
int i=0, j=0, mhkos=0, temp_variable=0, temp_variable_2=0;
float mhkoss=0.0;
/* --Mexri edw-- */

/* --Edw kanoume malloc gia ton pinaka pou ths bitonic permutation-- */
mhkos=(2*length)+1;
mhkoss=mhkos/2.0;
mhkos=mhkos/2;

/* --Edw kanoume to malloc gia ton artio arithmo apo noumera kai sthn sunexeia kanoume-- */
/* --arxikopihsh kai gemisma-- */
if((float)mhkos==mhkoss){
        BP=(int*)malloc(length* sizeof(int));
        if(BP==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
        for(i=0; i<length; i++){
                BP[i]=0;
        }
        //fprintf(stdout, "#########################################################################\n");
        //fprintf(stdout, "O pinakas BP einai:\n");
        //fprintf(stdout, "#########################################################################\n");
        ////fprintf(stdout, "\n");
        //for(i=0; i<length; i++){
        //      fprintf(stdout, "%d\t", BP[i]);
        //}
        //fprintf(stdout, "\n\n");
}
/* --Mexri edw-- */

/* --Mexri edw-- */
```

```
 /* --Edw kanoume to malloc gia ton peritto arithmo apo noumera kai sthn sunexeia kanoume-- */
 /* --arxikopihsh kai gemisma-- */
 else{
        BP=(int*)malloc(length* sizeof(int));
        if(BP==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
        for(i=0; i<length; i++){
                BP[i]=0;
        }
        for(i=0; i<(int)(length/2); i++){
                BP[i]=COUPLES[i][1];
                BP[(length-1)-i]=COUPLES[i][0];
        }
        BP[(int)(length/2)]=COUPLES[(int)(length/2)][0];

        //fprintf(stdout, "##########################################################################\n");
        //fprintf(stdout, "O pinakas BP einai:\n");
        //fprintf(stdout, "##########################################################################\n");
        //fprintf(stdout, "\n");
        //for(i=0; i<length; i++){
        //      fprintf(stdout, "%d\t", BP[i]);
        //}
        //fprintf(stdout, "\n\n");
 }
 /* --Mexri edw-- */

 /* --Mexri edw-- */

 attack_to_bp();
}
/* --Mexri edw-- */
```

```
/* --Edw einai h sunarthsh pou dhmiourgei tous X kai Y pinakes-- */
void X_and_Y(){

 /* --Edw einai oi metablhtes ths sunarthshs X_and_Y-- */
 int i=0, ii=0;
 /* --Mexri edw-- */

 /* --Edw prwta briskoume poses theseis xreiazontai autoi oi pinakes kai meta kanoume to malloc-- */
 while(BP[i]<BP[i+1]){
        sum_1++;
        i++;
 }
 sum_2=length-sum_1;
 /* --Mexri edw-- */

 /* --Edw kanoume ta duo malloc pou xreiazontai stous X kai Y pinakes kai tis arxikopoihseis autwn--*/
 X=(int*)malloc(sum_1* sizeof(int));
 if(X==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<sum_1; i++){
        X[i]=0;
 }
 Y=(int*)malloc(sum_2* sizeof(int));
 if(Y==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<sum_2; i++){
        Y[i]=0;
 }
 /* --Mexri edw-- */
```

```
 /* --Edw gemizoume tous X kai Y pinakes-- */
 for(i=0; i<sum_1; i++){
        X[i]=BP[i];
 }
 for(i=sum_1; i<length; i++){
        Y[ii]=BP[i];
        ii++;
 }
 /* --Mexri edw-- */

 /* --Edw tupwnoume tous pinakes X kai Y-- */
 //fprintf(stdout, "##############################################################################\n");
 //fprintf(stdout, "O pinakas X einai:\n");
 //fprintf(stdout, "##############################################################################\n");
 //fprintf(stdout, "\n");
 //for(i=0; i<sum_1; i++){
 //      fprintf(stdout, "%d\t", X[i]);
 //}
 //fprintf(stdout, "\n\n");
 //fprintf(stdout, "##############################################################################\n");
 //fprintf(stdout, "O pinakas Y einai:\n");
 //fprintf(stdout, "##############################################################################\n");
 //fprintf(stdout, "\n");
 //for(i=0; i<sum_2; i++){
 //      fprintf(stdout, "%d\t", Y[i]);
 //}
 //fprintf(stdout, "\n\n");
 /* --Mexri edw-- */

}
/* --Mexri edw-- */
```

```
/* --Edw einai h sunarthsh pou ftiaxnei tous B_tonos kai B_asteraki-- */
void B_tonos_and_B_asteraki(){

 /* --Edw einai oi metablhtes ths sunarthshs B_tonos_and_B_asteraki-- */
 int i=0, temp_variable=0;
 /* --Mexri edw-- */

 /* --Edw kanoume malloc gia ton B_tonos kai ton B_asteraki-- */
 B_tonos=(int*)malloc(length* sizeof(int));
 if(B_tonos==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 B_asteraki=(int*)malloc(length* sizeof(int));
 if(B_asteraki==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 /* --Mexri edw-- */

 /* --Edw arxikopoioume ton B_tonos kai ton B_asteraki-- */
 for(i=0; i<length; i++){
        B_tonos[i]=0;
 }
 for(i=0; i<length; i++){
        B_asteraki[i]=0;
 }
 /* --Mexri edw-- */

 /* --Edw gemizoume me thn seira ton B_asteraki kai ton B_tonos kai tous ektupwnoume-- */
 for(i=0; i<sum_1; i++){
        temp_variable=X[i]-1;
        B_asteraki[temp_variable]=0;
 }
 for(i=0; i<sum_2; i++){
```

```
        temp_variable=Y[i]-1;
        B_asteraki[temp_variable]=1;
 }
 //fprintf(stdout, "#########################################################################\n");
 //fprintf(stdout, "O pinakas B_asteraki einai:\n");
 //fprintf(stdout, "#########################################################################\n");
 //fprintf(stdout, "\n");
 //for(i=0; i<length; i++){
 //      fprintf(stdout, "%d\t", B_asteraki[i]);
 //}
 //fprintf(stdout, "\n\n");
 for(i=0; i<length; i++){
        if(B_asteraki[i]==1){
                B_tonos[i]=0;
        }
        else{
                B_tonos[i]=1;
        }
 }
 //fprintf(stdout, "#########################################################################\n");
 //fprintf(stdout, "O pinakas B_tonos einai:\n");
 //fprintf(stdout, "#########################################################################\n");
 //fprintf(stdout, "\n");
 //for(i=0; i<length; i++){
 //      fprintf(stdout, "%d\t", B_tonos[i]);
 //}
 //fprintf(stdout, "\n\n");
 /* --Mexri edw-- */

}
/* --Mexri edw-- */
```

```
/* --Edw einai h sunarthsh upswshs se dunamh-- */
int power(int base, int n){

  /* --Edw einai oi metablhtes ths sunarthshs power-- */
  int i=0, p=0;
  /* --Mexri edw-- */

  p = 1;
          for(i=1; i<=n; ++i){
          p*=base;
  }
          return(p);
}
/* --Mexri edw-- */




/* --Edw kataskeuazoume to udatoshma apo duadikh morfh se dekadikh-- */
void binary_to_w(){

  /* --Edw einai oi metablhtes ths sunarthshs binary_to_w-- */
  int i=0, ii=0, pow=0;
  /* --Mexri edw-- */

  for(i=length-2; i>=(int)(length/2); i--){
          if(B_tonos[i]==1){
                  w=w+power(2, pow);
          }
          pow++;
  }
  //fprintf(stdout, "#########################################################################\n");
  //fprintf(stdout, "!!!!Briskomaste sto sunolo N%d twn fusikwn arithmwn!!!!\n", length);
```

```
  //fprintf(stdout, "#########################################################################\n");
  //fprintf(stdout, "\n");
  //fprintf(stdout, "#########################################################################\n");
  //fprintf(stdout, "To udatoshma einai:\n");
  //fprintf(stdout, "#########################################################################\n");
  //fprintf(stdout, "\n");
  //fprintf(stdout, "-------------------------\n");
  //fprintf(stdout, "%d\n", w);
  //fprintf(stdout, "-------------------------\n");
  //fprintf(stdout, "\n");
}
/* --Mexri edw-- */




/* --Edw einai h sunarthsh main-- */
int main(){

  /* --Edw einai oi metablhtes ths sunarthshs main-- */
  unsigned int i=0;
  /* --Mexri edw-- */

  infile = fopen(inname, "r");
  if(!infile){
          printf("Den mporei na anoiksei to arxeio %s gia diabasma!!!!\n", inname);
          return(0);
  }

  write_to_file = fopen(inname_2, "w+");
  if(!write_to_file){
          printf("Den mporei na anoiksei to arxeio %s gia diabasma!!!!\n", inname);
          return(0);
  }
```

```
while(i<4916800){
        //printf("i= %d\n", i);
        /* --Edw kaloume thn sunarthsh pou diabazei thn s.i.p-- */
        sip();
        /* --Mexri edw-- */

        /* --Edw kaloume thn sunarthsh pou ftiaxnei ta zeugaria-- */
        couples();
        /* --Mexri edw-- */

        /* --Edw kaloume thn sunarthsh pou ftiaxnei thn bitonic permutation-- */
        bitonic_p();
        /* --Mexri edw-- */

        /* --Edw kaloume thn sunarthsh pou ftiaxnei tous pinakes X kai Y-- */
        X_and_Y();
        /* --Mexri edw-- */

        /* --Edw kaloume thn sunarthsh pou dhmiourgei ton pinaka B_tonos kai ton B_asteraki-- */
        B_tonos_and_B_asteraki();
        /* --Mexri edw-- */

        /* --Edw kaloume thn sunarthsh pou kataskeuazei ton fusiko arithmo w (udatoshma-- */
        /* --apo thn duadiakh tou morfh-- */
        binary_to_w();
        /* --Mexri edw-- */

        length=0;
        i++;

        /*free(BP_2);
        free(BP);
        free(COUPLES[0]);
```

```
            free(COUPLES[1]);
            free(COUPLES);
            free(X);
            free(Y);
            free(B_tonos);
            free(B_asteraki);*/

            sum_1=1;
            sum_2=0;

            is=0;
  }

  fprintf(stdout, "S.i.p kai b.p tautoxrona einai: %d permutations!!!!\n", sip_bp);

  fclose(infile);
  fclose(write_to_file);
}
/* --Mexri edw-- */
```

Παράρτημα 5

Κώδικας για Δημιουργία και Επίθεση σε Τυχαίες s.i.p

- Στον παρακάτω κώδικα, υλοποιείται η δημιουργία αυτοαναστρέφουσων μεταθέσεων επαναληπτικά.
- Στη συνέχεια γίνονται επιθέσεις σε κάθε μία από αυτές τις μεταθέσεις.
- Ενώ στο τέλος ελέγχονται μία προς μία αν έχουν παραμείνει αυτοαναστρέφουσες ή όχι.

```
/* --Grafthke apo Iwannhs Xionhs A.M:1430-- */


#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <time.h>


void make_sip();

void attack_to_permutation();

void find_the_attack_in_sip();

void create_dag();

void create_final_graph();

void attack_to_edges();


/* --Edw einai h dhlwshs ths apla sundedemenhs listas-- */

struct list_elements{

 int value;

 struct list_elements *next;

};

typedef struct list_elements item;

/* --Mexri edw-- */


item **dag, **final_graph;


time_t seconds;
```

```
int *sip, cycles_of_sip[15][2], random_number=0;
int L=1, H=29, attacks_detect_on_sip=0, attacks_detect_on_rpg=0, length=29;
char *name="sip_with_29_elements", *attacked_name="attacked_sips", *n="attacked_rpg";
FILE *write_to_file, *write_to_attacked_file, *write_attacked_rpg;

void make_sip(){
 int i=0, ii=0, k=0, temp=0;

 sip=(int *)malloc(29* sizeof(int));

 random_number=rand()%(H-L+1)+L;

 for(i=0; i<29; i++){
        sip[i]=0;
        sip[i]=i+1;
 }

 //fprintf(stdout, "H s.i.p einai:\n");
 //for(i=0; i<H-1; i++){
 //      fprintf(stdout, "%d, ", sip[i]);
 //}
 //fprintf(stdout, "%d\n\n", sip[H-1]);

 for(ii=0; ii<14; ii++){
        cycles_of_sip[i][0]=0;
        cycles_of_sip[i][1]=0;
 }

 for(ii=0; ii<14; ii++){
        random_number=rand()%(H-L+1)+L;
        cycles_of_sip[ii][0]=sip[random_number-1];

        random_number=rand()%(H-L+1)+L;
        while(sip[random_number-1]==cycles_of_sip[ii][0]){
```

```
                random_number=rand()%(H-L+1)+L;
        }
        cycles_of_sip[ii][1]=sip[random_number-1];

        for(i=0; i<H; i++){
                if((cycles_of_sip[ii][0]==sip[i]) || (cycles_of_sip[ii][1]==sip[i])){
                        sip[i]=0;
                }
        }

        for(k=0; k<2; k++){
                for(i=0; i<H-1; i++){
                        if(sip[i]==0){
                                temp=sip[i];
                                sip[i]=sip[i+1];
                                sip[i+1]=temp;
                        }
                }
        }

        H=H-2;
        sip=(int*)realloc(sip, H * sizeof(int));

        //fprintf(stdout, "H mikroterh s.i.p einai:\n");
        //for(i=0; i<H-1; i++){
        //      fprintf(stdout, "%d, ", sip[i]);
        //}
        //fprintf(stdout, "%d\n\n", sip[H-1]);
}

cycles_of_sip[14][0]=sip[0];
cycles_of_sip[14][1]=sip[0];

//fprintf(stdout, "Oi kukloi einai:\n");
```

```
 //for(i=0; i<15; i++){
 //      fprintf(stdout, "%d\t%d\n", cycles_of_sip[i][0], cycles_of_sip[i][1]);
 //}
 //fprintf(stdout, "\n\n");

 sip=(int*)realloc(sip, 29 * sizeof(int));

 for(i=0; i<14; i++){
        sip[cycles_of_sip[i][0]-1]=cycles_of_sip[i][1];
        sip[cycles_of_sip[i][1]-1]=cycles_of_sip[i][0];
 }
 sip[cycles_of_sip[14][0]-1]=cycles_of_sip[14][1];

 //fprintf(stdout, "H telikh s.i.p einai:\n");
 fprintf(write_to_file, "(");
 for(i=0; i<28; i++){
        fprintf(write_to_file, "%d, ", sip[i]);
 }
 fprintf(write_to_file, "%d)\n\n", sip[28]);

 H=29;
}

void attack_to_permutation(){
 int i=0, k=0, flag=0, temporary=0, pos_of_sip_element[4]={0, 0, 0, 0};

 while(i<4){
        flag=0;
        random_number=rand()%(29-L+1)+L;
        for(k=0; k<4; k++){
                if(random_number==pos_of_sip_element[k]){
                        flag=1;
                        break;
                }
```

```
        }
        if(flag==0){
                pos_of_sip_element[i]=random_number;
                i++;
        }
 }

 //for(i=0; i<4; i++){
 //      fprintf(stdout, "%d\t", pos_of_sip_element[i]);
 //}
 //fprintf(stdout, "\n");

 temporary=sip[pos_of_sip_element[0]-1];
 sip[pos_of_sip_element[0]-1]=sip[pos_of_sip_element[1]-1];
 sip[pos_of_sip_element[1]-1]=temporary;

 temporary=sip[pos_of_sip_element[2]-1];
 sip[pos_of_sip_element[2]-1]=sip[pos_of_sip_element[3]-1];
 sip[pos_of_sip_element[3]-1]=temporary;

 fprintf(write_to_attacked_file, "(");
 for(i=0; i<28; i++){
        fprintf(write_to_attacked_file, "%d, ", sip[i]);
 }
 fprintf(write_to_attacked_file, "%d)\n\n", sip[28]);
}

/* --Edw einai h sunarthsh pou elegxei gia epitheseis sthn s.i.p-- */
void find_the_attack_in_sip(){
 int i=0, temp_1=0, temp_2=0, lala_1=0, lala_2=0;

 for(i=0; i<29; i++){
        temp_1=sip[i];
        temp_2=i;
```

```
		lala_1=sip[temp_1-1];
		lala_2=temp_1-1;


		if((temp_1!=lala_2+1) || (temp_2+1!=lala_1)){
			attacks_detect_on_sip++;
			//abort();
			break;
		}
	}
}
/* --Mexri edw-- */


/* --Edw einai h sunarthsh pou dhmiourgei to d.a.g-- */
void create_dag(){


	/* --Edw einai dhlwseis twn metablhtwn ths create_dag-- */
	int i=0, k=0, temp=0, flag=0;
	item **current_pointer;
	/* --Mexri edw-- */


	length=29;


	/* --Edw dhmiourgoume ton mo pinaka pou kathe st tou einai deikths kai ena stoixeio tou d.a.g to opoio me th
seira tou deixnei tous geitwnes tou sto d.a.g-- */
	/* --Edw kanoume malloc gia ton pinaka dag gia na dextei ta stoixeia ths sip kai gia ton current_pointer ton
opoio tha xrhsimopoioume gia na gurizoume opote */
	/* --Theloume sthn arxh tou d.a.g-- */
	dag=(item **)malloc((length+2)* sizeof(item*)); /* --Einai length+2 gia na kanoume malloc kai gia tous s kai t
kombous tou d.a.g-- */
	if(dag==NULL){
		fprintf(stdout, "Den egine swsta to malloc!!!!\n");
	}
	for(i=0; i<length+2; i++){
		dag[i]=(item *)malloc(1* sizeof(item));
```

```
        if(dag[i]==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
 }

 current_pointer=(item **)malloc(length+2* sizeof(item*)); /* --Einai length+2 gia na kanoume malloc kai gia tous
s kai t kombous tou-- */
 if(current_pointer==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<length+2; i++){
        current_pointer[i]=(item *)malloc(1* sizeof(item));

        if(current_pointer[i]==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
 }
 /* --Mexri edw-- */

 /* --Edw antigrafoume thn sip ston pinaka dag kai to dag ston current_pointer gia na mporoume na gurizoume sthn
arxh opote theloume-- */
 for(i=0; i<length; i++){
        dag[i]->value=sip[i];
        dag[i]->next=NULL;
 }
 dag[length]->value=1000;
 dag[length]->next=NULL;
 dag[length+1]->value=-1000;
 dag[length+1]->next=NULL;
 for(i=0; i<length+2; i++){
        current_pointer[i]=dag[i];
 }
 /* --Mexri edw-- */

 /* --Edw briskoume thn didomination sxesh gia kathe stoixeio-- */
```

```
for(k=0; k<length-1; k++){
        i=k;
        temp=0;
        while((i+1)<length){
                if((sip[k]>sip[i+1]) && (sip[i+1]>=temp)){
                        dag[k]->next=(item *)malloc(1* sizeof(item));
                        dag[k]=dag[k]->next;
                        dag[k]->value=sip[i+1];
                        dag[k]->next=NULL;
                        temp=sip[i+1];
                }
                i++;
        }
}
/* --Mexri edw-- */

/* --Edw briskoume tous geitones twn s kai t kombwn tou d.a.g-- */
for(i=0; i<length+2; i++){
        dag[i]=current_pointer[i];
}
for(k=0; k<length; k++){

        for(i=0; i<length+2; i++){
                if(i!=length){
                        dag[i]=current_pointer[i];
                }
        }

        flag=0;
        i=0;
        while((i<length) && (flag==0)){
                dag[i]=dag[i]->next;
                while(dag[i]!=NULL){
                        if(dag[i]->value!=sip[k]){
```

```
                        dag[i]=dag[i]->next;
                }
                else{
                        flag=1;
                        break;
                }
        }
        if(flag==0){
                i++;
        }
    }

    if(flag==0){
        dag[length]->next=(item *)malloc(1* sizeof(item));
        dag[length]=dag[length]->next;
        dag[length]->value=sip[k];
        dag[length]->next=NULL;
    }
}
for(i=0; i<length+2; i++){
    dag[i]=current_pointer[i];
}
for(i=0; i<length; i++){
    if(dag[i]->next==NULL){
        dag[i]->next=(item *)malloc(1* sizeof(item));
        dag[i]=dag[i]->next;
        dag[i]->value=-1000;
        dag[i]->next=NULL;
    }
}
/* --Mexri edw-- */

for(i=0; i<length+2; i++){
    dag[i]=current_pointer[i];
```

```
  }


  /* --Edw tupwnoume to d.a.g pou dhmiourghthhke-- */

  /*fprintf(write_attacked_rpg, "#########################################################################\n");

  fprintf(write_attacked_rpg, "D.a.g (opou 1000 einai o s kombos kai -1000 einai o t kombos):       #\n");

  fprintf(write_attacked_rpg, "#########################################################################\n");

  fprintf(write_attacked_rpg, "\n");

  for(i=0; i<length+2; i++){

          while(dag[i]!=NULL){

                  fprintf(write_attacked_rpg, "(%d)---->", dag[i]->value);

                  dag[i]=dag[i]->next;

          }

          fprintf(write_attacked_rpg, "NULL\n");

  }

  fprintf(write_attacked_rpg, "\n");*/

  /* --Mexri edw-- */


  /* --Mexri edw-- */


  for(i=0; i<length+2; i++){

          dag[i]=current_pointer[i];

  }

}

/* --Mexri edw-- */


/* --Edw einai h sunarthsh pou dhmiourgei to teliko grafhma-- */

void create_final_graph(){


  /* --Edw einai dhlwseis twn metablhtwn ths create_final_graph-- */

  int  i=0,  ii=0,  iii=0,  k=0,  temp=0,  temp_2=0,  sum=0,  sum_2=0,  maximum=0,  maximum_2=0,  stop=0,  flag=0,
*temp_buffer, temp_buffer_2[100], max_element[100];

  /* --O prwtos einai gia na mp na gu sthn arxh tou final_graph opote th kai de einai gia na kanoume thn idia
diadikasia sto d.a.g-- */

  item **current_pointer, **current_pointer_2;

  /* --Mexri edw-- */
```

```
 length=29;


 /* --Edw tha antigrapsoume thn s.i.p se enan prosorino pinaka kai meta tha thn taksinomhsoume-- */

 temp_buffer=(int *)malloc((length+2)*sizeof(int));

 if(temp_buffer==NULL){

        fprintf(stdout, "Den egine swsta to malloc!!!!\n");

 }

 for(i=0; i<length; i++){

        temp_buffer[i]=sip[i];

 }

 temp_buffer[length]=1000;

 temp_buffer[length+1]=-1000;

 for(k=1; k<length+2; k++){

        for(i=length+1; i>=k; i--){

                if(temp_buffer[i]>temp_buffer[i-1]){

                        temp=temp_buffer[i];

                        temp_buffer[i]=temp_buffer[i-1];

                        temp_buffer[i-1]=temp;

                }

        }

 }

 /* --Mexri edw-- */


 /* --Edw dh ton mono pinaka pou kathe stoixeio tou einai deikths kai ena st tou final_graph to opoio me th seira
tou deixnei tous geitwnes tou sto final_graph-- */

 /* --Edw kanoume malloc gia ton pinaka final_graph gia na dextei ta stoixeia kai gia tous current_pointer kai
current_pointer_2 tous opoious tha xrhsimopoioume-- */

 /* --Gia na gurizoume opote theloume sthn arxh tou final_graph kai tou d.a.g-- */

 final_graph=(item **)malloc((length+2)* sizeof(item*)); /* --Einai length+2 gia na kanoume malloc kai gia tous s
kai t kombous tou final_graph-- */

 if(final_graph==NULL){

        fprintf(stdout, "Den egine swsta to malloc!!!!\n");

 }

 for(i=0; i<length+2; i++){

        final_graph[i]=(item *)malloc(1* sizeof(item));
```

```
        if(final_graph[i]==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
 }

 current_pointer=(item **)malloc(length+2* sizeof(item*)); /* --Einai length+2 gia na kanoume malloc kai gia tous
s kai t kombous tou-- */
 if(current_pointer==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<length+2; i++){
        current_pointer[i]=(item *)malloc(1* sizeof(item));
        if(current_pointer[i]==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
 }

 current_pointer_2=(item **)malloc(length+2* sizeof(item*)); /* --Einai length+2 gia na kanoume malloc kai gia
tous s kai t kombous tou-- */
 if(current_pointer_2==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
 }
 for(i=0; i<length+2; i++){
        current_pointer_2[i]=(item *)malloc(1* sizeof(item));
        if(current_pointer_2[i]==NULL){
                fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
 }
 /* --Mexri edw-- */

 for(i=0; i<length+2; i++){
        current_pointer[i]=final_graph[i];
 }
 for(i=0; i<length+2; i++){
        current_pointer_2[i]=dag[i];
```

```
 }
 for(i=0; i<length+2; i++){
        final_graph[i]->value=temp_buffer[i];
        final_graph[i]->next=NULL;
 }

 /* --Edw arxikopoioume ton temp_buffer_2 o opoios tha exei ola ta stoixeia pou tha deixnoun kathe fora se kathe
ena stoixeio tou d.a.g-- */
 for(i=0; i<100; i++){
        temp_buffer_2[i]=0;
        max_element[i]=0;
 }
 /* --Mexri edw-- */

 /* --Edw dhmiourgoume to final graph-- */
 for(i=0; i<length+1; i++){
        final_graph[i]->next=(item *)malloc(sizeof(item));
        final_graph[i]=final_graph[i]->next;
                final_graph[i]->value=temp_buffer[i+1];
                final_graph[i]->next=NULL;
 }
 for(i=0; i<length+2; i++){
        final_graph[i]=current_pointer[i];
 }
 for(i=0; i<length+2; i++){
        dag[i]=current_pointer_2[i];
 }
 for(ii=0; ii<length; ii++){
        maximum=0;
        sum=0;
        for(i=0; i<length+2; i++){
                while(dag[i]->next!=NULL){
                        max_element[sum_2]=dag[i]->value;
                        sum_2++;
                        if(dag[i]->next->value==sip[ii]){
```

```
                        flag=1;
                        break;
                }
                dag[i]=dag[i]->next;
        }
        if(flag==1){
                for(iii=0; iii<sum_2; iii++){
                        if(max_element[iii]>maximum_2){
                                maximum_2=max_element[iii];
                        }
                }
                temp_buffer_2[sum]=maximum_2;
                sum++;
                flag=0;
                maximum_2=0;
        }
        for(iii=0; iii<100; iii++){
                max_element[i]=0;
        }
        sum_2=0;
}
for(i=0; i<sum; i++){
        if(temp_buffer_2[i]>maximum){
                maximum=temp_buffer_2[i];
        }
}
if(sum!=0){
        i=0;
        stop=0;
        while(i<length+2){
                if(final_graph[i]->value==sip[ii]){
                        while(final_graph[i]->next!=NULL){
                                final_graph[i]=final_graph[i]->next;
                        }
```

```
                                if(final_graph[i]->next==NULL){
                                        final_graph[i]->next=(item *)malloc(sizeof(item));
                                        final_graph[i]=final_graph[i]->next;
                                                final_graph[i]->value=maximum;
                                                final_graph[i]->next=NULL;
                                        break;
                                }
                        }
                        i++;
                }
        }
        for(i=0; i<length+2; i++){
                final_graph[i]=current_pointer[i];
        }
        for(i=0; i<length+2; i++){
                dag[i]=current_pointer_2[i];
        }
        for(i=0; i<100; i++){
                temp_buffer_2[i]=0;
                max_element[i]=0;
        }
}
/* --Mexri edw-- */

/* --Edw tupwnoume to teliko grafhma pou dhmiourghthhke-- */
fprintf(write_attacked_rpg, "#########################################################################\n");
fprintf(write_attacked_rpg, "Final_graph (opou 1000 einai o s kombos kai -1000 einai o t kombos): #\n");
fprintf(write_attacked_rpg, "#########################################################################\n");
fprintf(write_attacked_rpg, "\n");
fprintf(write_attacked_rpg, "------\n");
for(i=0; i<length+2; i++){
        while(final_graph[i]!=NULL){
                fprintf(write_attacked_rpg, "(%d)---->", final_graph[i]->value);
                final_graph[i]=final_graph[i]->next;
```

```
        }
        fprintf(write_attacked_rpg, "NULL\n");
  }
  fprintf(write_attacked_rpg, "------\n");
  fprintf(write_attacked_rpg, "\n");
  /* --Mexri edw-- */

  for(i=0; i<length+2; i++){
        final_graph[i]=current_pointer[i];
  }
  for(i=0; i<length+2; i++){
        dag[i]=current_pointer_2[i];
  }
}
/* --Mexri edw-- */

/* --Edw einai h sunarthsh pou elegxei gia epitheseis stis akmes tou final_graph-- */
void attack_to_edges(){
  int  i=0,  out_degree=0,  ok_1=0,  ok_2=0,  temp_vertex=0,  pos=-100,  correction=0,  insert_1=0,  insert_2=0,
insert_3=0, insert_4=0, check=0;
  item **current_pointer;

  current_pointer=(item **)malloc(length+2* sizeof(item*)); /* --Einai length+2 gia na kanoume malloc kai gia tous
s kai t kombous tou-- */
  if(current_pointer==NULL){
        fprintf(stdout, "Den egine swsta to malloc!!!!\n");
  }
  for(i=0; i<length+2; i++){
        current_pointer[i]=(item *)malloc(1* sizeof(item));
        if(current_pointer[i]==NULL){
              fprintf(stdout, "Den egine swsta to malloc!!!!\n");
        }
  }

  for(i=0; i<length+2; i++){
```

```
        current_pointer[i]=final_graph[i];
  }


  /* --Edw elegxetai an xalaei h idiothta tou grafhmatos pou kathe kombos exei ekserxomeno arithmo apo akmes iso
me 2-- */
  for(i=1; i<31-1; i++){
        out_degree=0;
        temp_vertex=final_graph[i]->value;
        while(final_graph[i]!=NULL){
                final_graph[i]=final_graph[i]->next;
                out_degree++;
        }
        out_degree--;
        if(out_degree<2){
                if(check==0){
                        attacks_detect_on_rpg++;
                        check=1;
                }
                insert_1=1;
        }
        if(out_degree>2){
                if(check==0){
                        attacks_detect_on_rpg++;
                        check=1;
                }
                insert_1=1;
                }
  }
  /* --Mexri edw-- */


  /* --Edw elegxetai an o s kombos exei ekserxomeno arithmo apo akmes iso me 1 kai o t iso me 0-- */
  for(i=0; i<31; i++){
        final_graph[i]=current_pointer[i];
  }
  out_degree=0;
```

```
pos=-100;
for(i=0; i<31; i++){
        if(final_graph[i]->value==1000){
                pos=i;
                break;
        }
}
if(pos!=-100){
        while(final_graph[pos]!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                out_degree++;
        }
        out_degree--;
        if(out_degree>1){
                if(check==0){
                        attacks_detect_on_rpg++;
                        check=1;
                }
                insert_3=1;
        }
        if(out_degree==0){
                if(check==0){
                        attacks_detect_on_rpg++;
                        check=1;
                }
                insert_3=1;
        }
}

for(i=0; i<31; i++){
        final_graph[i]=current_pointer[i];
}
out_degree=0;
pos=-100;
```

```
for(i=0; i<31; i++){
        if(final_graph[i]->value==-1000){
                pos=i;
                break;
        }
}
if(pos!=-100){
        while(final_graph[pos]!=NULL){
                final_graph[pos]=final_graph[pos]->next;
                out_degree++;
        }
        out_degree--;
        if(out_degree>0){
                if(check==0){
                        attacks_detect_on_rpg++;
                        check=1;
                }
                insert_4=1;
        }
}
/* --Mexri edw-- */

/* --Edw elegxoume ton arithmo twn front edges kai twn back edges-- */
for(i=0; i<31; i++){
        final_graph[i]=current_pointer[i];
}

for(i=1; i<31-2; i++){
        ok_1=0;
        ok_2=0;
        temp_vertex=final_graph[i]->value;
        while(final_graph[i]->next!=NULL){
                if(temp_vertex-(final_graph[i]->next->value)==1){
                        ok_1=1;
```

```
                }
                if(temp_vertex-(final_graph[i]->next->value)<0){
                        ok_2=1;
                }
                final_graph[i]=final_graph[i]->next;
        }
        if(ok_1!=1){
                if(check==0){
                        attacks_detect_on_rpg++;
                        check=1;
                }
                insert_2=1;
        }
        if(ok_2!=1){
                if(check==0){
                        attacks_detect_on_rpg++;
                        check=1;
                }
        }
  }
  /* --Mexri edw-- */

  for(i=0; i<31; i++){
        final_graph[i]=current_pointer[i];
  }

}
/* --Mexri edw-- */

int main(){
  int t=0, l=0, i=0;
  item *temp_1, *temp_2;

  temp_1=(item *)malloc(1* sizeof(item));
```

```
temp_1->next=NULL;


temp_2=(item *)malloc(1* sizeof(item));
temp_2->next=NULL;



time(&seconds);
srand((unsigned int) seconds);


write_to_file=fopen(name, "w+");
if(!write_to_file){
        printf("Den mporei na anoiksei to arxeio %s gia diabasma!!!!\n", name);
        return(0);
}


write_to_attacked_file=fopen(attacked_name, "w+");
if(!write_to_attacked_file){
        printf("Den mporei na anoiksei to arxeio %s gia diabasma!!!!\n", name);
        return(0);
}


write_attacked_rpg=fopen(n, "w+");
if(!write_attacked_rpg){
        printf("Den mporei na anoiksei to arxeio %s gia diabasma!!!!\n", n);
        return(0);
}


for(t=0; t<100000; t++){
        make_sip();


        attack_to_permutation();


        find_the_attack_in_sip();
```

```
        create_dag();


        create_final_graph();


        attack_to_edges();


        free(sip);

        free(dag);

        free(final_graph);


        //sleep(1);

 }


 fprintf(stdout, "Apo tis 100000 allagmenes s.i.p, antiliptes eginan oi %d sto epipedo ths s.i.p!!!!\n\n",
attacks_detect_on_sip);

 fprintf(stdout, "Apo tis 100000 allagmenes s.i.p, antiliptes eginan oi %d sto epipedo tou r.p.g!!!!\n",
attacks_detect_on_rpg);


 fclose(write_to_file);

 fclose(write_to_attacked_file);

}

/* --Mexri edw-- */
```