2.5 Αξιοποίηση προγραμματιστικών σφαλμάτων

Σε πολλές περιπτώσεις, προγραμματιστικά σφάλματα σε εφαρμογές ή σε λειτουργικά συστήματα επιτρέπουν σε επίδοξους εισβολείς να υποβαθμίσουν την ασφάλεια των υπολογιστικών συστημάτων. Με τον όρο *προγραμματιστικά σφάλματα*

4.3 Ασφάλεια στον προγραμματισμό

Στον τομέα της ανάπτυξης λογισμικού είναι αρκετά διαδεδομένη η αντίληψη ότι ο κώδικας των προγραμμάτων δεν σχετίζεται με την ασφάλεια, η οποία θα πρέπει να είναι μέλημα πρόσθετων εξειδικευμένων μηχανισμών που συμπληρώνουν το λογισμικό. Η αντίληψη αυτή είναι εσφαλμένη καθώς συγκεκριμένοι τρόποι συγγραφής κώδικα δίνουν έδαφος σε επίδοξους εισβολείς να υποβαθμίσουν την ασφάλεια του συστήματος. Πέραν της άγνοιας, η παράβλεψη των πρακτικών ασφαλείας κατά τη συγγραφή κώδικα μπορεί να οφείλεται και σε άλλους λόγους, π.χ. η ασφάλεια να θεωρείται πάρεργο του προγραμματιστή, σε σχέση με την «πιο ενδιαφέρουσα» αντιμετώπιση των λειτουργικών προδιαγραφών ή η έμφαση σε χρονικούς περιορισμούς παράδοσης του λογισμικού που δημιουργεί πίεση από υψηλότερα εταιρικά κλιμάκια. Η μειωμένη ασφάλεια όμως ενός λογισμικού δημιουργεί σοβαρές παρενέργειες στους οίκους ανάπτυξης λογισμικού καθώς, όταν ανακαλυφθεί το κενό στην ασφάλεια, θα πρέπει:

- Οι προγραμματιστές να σταματήσουν την τρέχουσα εργασία τους και να ασχοληθούν με την επιδιόρθωση του προβλήματος
- Να ειδοποιηθούν οι πελάτες/χρήστες
- Να αποσταλεί η νέα έκδοση
- Να εγκατασταθεί η νέα έκδοση (με πιθανή συνδρομή τεχνικών της εταιρείας)

Όλες οι ανωτέρω διαδικασίες ενέχουν σημαντικό οικονομικό και διαχειριστικό κόστος για την εταιρεία, πέραν βέβαια της βλάβης που προκαλείται στην εικόνα της, ειδικότερα αν η ασφάλεια πρόκειται για κρίσιμο παράγοντα στις υπηρεσίες που παρέχει (π.χ. χρηματοπιστωτικοί οργανισμοί, διαχείριση ευαίσθητων δεδομένων κ.λπ.).

Βάσει των ανωτέρω, θα πρέπει η ασφάλεια να ενταχθεί ως απαραίτητο συστατικό της διαδικασίας διασφάλισης ποιότητας του λογισμικού. Δύο κρίσιμες διαστάσεις του θέματος είναι οι ακόλουθες:

- Γνώση από τους προγραμματιστές όλων των πιθανών ευπαθειών ασφάλειας της γλώσσας και του περιβάλλοντος προγραμματισμού, ώστε αυτές να αποφεύγονται ή να αντιμετωπίζονται κατάλληλα
- Επιθεωρήσεις ασφαλείας για τον κώδικα, δηλ. εντοπισμός των πιθανών σημείων όπου διακυβεύεται η ασφάλεια του λογισμικού. Οι επιθεωρήσεις μπορούν να γίνονται:
    - Από προγράμματα ανίχνευσης ευπαθειών
    - Από εξειδικευμένους προγραμματιστές (security audits).

Στο σχήμα που ακολουθεί φαίνεται ενδεικτικά πώς σε ένα τετριμμένο πρόγραμμα γλώσσας C μπορούν να εντοπισθούν αρκετά σημεία με προβληματική ασφάλεια από ένα εργαλείο επιθεώρησης .ασφάλειας. Τα αναφερόμενα προβλήματα επεξηγούνται στη συνέχεια του εδαφίου.

```
#include <stdio.h>
#include <string.h>

int main(void) {
char s1[100], s2[80];

puts("Enter a string: ");
gets(s1);
strcpy(s2, s1);
printf(s2);
return 0;
}
```

(α) Πρόγραμμα probs.c

```
its4 probs.c
probs.c:8:(Urgent) gets
The input buffer can almost always be
overflowed.
Use fgets(buf,size,stdin) instead.
---------------
probs.c:10:(Urgent) printf
Non-constant format strings can often be
attacked.
Use a constant format string.
---------------
probs.c:9:(Very Risky) strcpy
This function is high risk for buffer
overflows
Use strncpy instead.
---------------
```

(β) Ανάλυση ευπαθειών

4.3.1 Ευπάθειες στις γλώσσες C/C++

Οι γλώσσες προγραμματισμού C και C++ παρουσιάζουν το πλουσιότερο ιστορικό αναφερόμενων ευπαθειών σε προγράμματα που έχουν αναπτυχθεί μ' αυτές για δύο κυρίως λόγους:

1. πρόκειται για τις γλώσσες που κατά κύριο λόγο χρησιμοποιούνται στην ανάπτυξη λογισμικού συστήματος και λογισμικού εφαρμογών, συμπεριλαμβάνοντας λειτουργικά συστήματα, βάσεις δεδομένων, παραθυρικά περιβάλλοντα, εξυπηρέτες διαδικτύου, κ.ο.κ.
2. οι γλώσσες αυτές έχουν σχεδιαστεί με τη φιλοσοφία ότι ο προγραμματιστής *γνωρίζει καλά τι κάνει*, επιτρέποντας λειτουργίες που σε άλλες γλώσσες θα απαγορευόταν συνολικά – π.χ. η αυθαίρετη μετατροπή δεικτών από έναν τύπο δεδομένων σε έναν άλλο, η χρήση αρνητικών δεικτών σε έναν πίνακα, η κλήση συναρτήσεων με μεταβλητό αριθμό παραμέτρων κ.ο.κ. Η κακή ή απρόσεκτη χρήση των δυνατοτήτων αυτών οδηγεί πολλές φορές σε προβλήματα ασφάλειας.

4.3.1.1 Υπερχείλιση μνήμης

Ένα μεγάλο σύνολο από προβλήματα ασφάλειας συσχετίζεται με την υπερχείλιση μνήμης. Με τον όρο αυτό αναφερόμαστε στις περιπτώσεις όπου δεσμεύεται ένα ποσό μνήμης για κάποια λειτουργία, αλλά κατά την εκτέλεση της λειτουργίας απαιτείται μεγαλύτερο ποσό από αυτό που δεσμεύτηκε. Οι αναγκαίες συνθήκες για να έχουμε υπερχείλιση ενδιάμεσης μνήμης είναι:

1. Λήψη δεδομένων από μη αξιόπιστη πηγή (πληκτρολόγιο, δίκτυο, αρχείο, διαδιεργασιακή επικοινωνία κ.λπ.)
2. Αποθήκευση των δεδομένων σε ενδιάμεση μνήμη περιορισμένου μεγέθους χωρίς κατάλληλο έλεγχο αν τα δεδομένα όντως χωράνε στη μνήμη

Το αποτέλεσμα είναι να επικαλυφθούν τιμές από γειτονικές μεταβλητές ή/και διευθύνσεις επιστροφής από συναρτήσεις, τιμές αποθηκευμένων καταχωρητών κ.ο.κ.

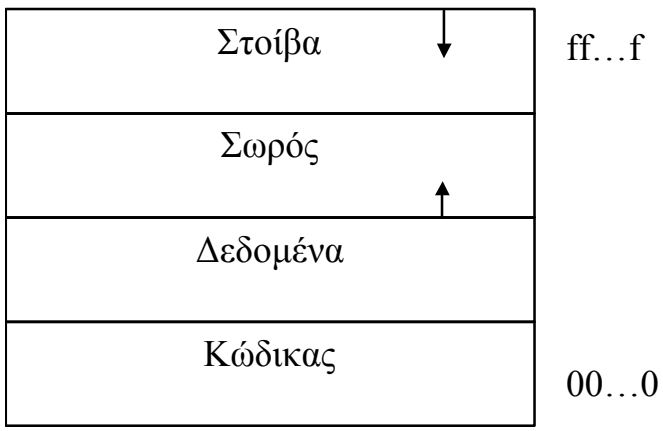
Στις επόμενες παραγράφους αναλύονται οι διάταξη της μνήμης και οι επιπτώσεις στην ασφάλεια που έχουν τα συνηθέστερα λάθη.

Δομή μνήμης διεργασίας

Στα περισσότερα λειτουργικά συστήματα η μνήμη της διεργασίας οργανώνεται σε τμήματα με τα κάτωθι χαρακτηριστικά:

1. *Τμήμα κώδικα*. Περιέχει τον εκτελέσιμο κώδικα της διεργασία, δηλ. τις εντολές που εκτελεί ο επεξεργαστής. Τις περισσότερες φορές είναι ανάγνωσης μόνο.
2. *Τμήμα δεδομένων*. Περιέχει τα *καθολικά δεδομένα* της διεργασίας, καθώς και στατικές μεταβλητές συναρτήσεων (και κλάσεων στη C++). Μερικές φορές διαχωρίζεται σε *τμήμα αρχικοποιημένων δεδομένων* και *τμήμα μη αρχικοποιημένων δεδομένων*, ενώ μπορεί να περιλαμβάνει και *τμήμα σταθερών* που μπορεί να χαρακτηρίζεται ως *ανάγνωσης μόνο*.
3. *Τμήμα στοίβας*. Το τμήμα στοίβας περιέχει τα δεδομένα που δημιουργούνται και καταστρέφονται δυναμικά κατά την εκτέλεση της διεργασίας. Τέτοια δεδομένα είναι οι χώροι που δεσμεύονται με τη malloc, καθώς και οι τιμές παραμέτρων συναρτήσεων και οι τοπικές τους μεταβλητές (που δεν είναι static). Στη στοίβα επίσης αποθηκεύονται και οι *διευθύνσεις επιστροφής* από συναρτήσεις, π.χ. όταν η main() καλεί μία συνάρτηση f(), στη στοίβα αποθηκεύεται η διεύθυνση της εντολής της main() στην οποία θα συνεχιστεί η εκτέλεση μετά το πέρας της f(). Το τμήμα στοίβας διαχωρίζεται συνήθως στον *σωρό*, που περιέχει τα δυναμικά δεσμευόμενα τμήματα και τη *στοίβα* που περιέχει τα σχετιζόμενα με κλήση συναρτήσεων δεδομένα. Η στοίβα επεκτείνεται από τις υψηλότερες διευθύνσεις προς τις χαμηλότερες, ενώ ο σωρός αντιστρόφως.

Το σχήμα που ακολουθεί απεικονίζει τη διάταξη μνήμης μιας τυπικής διεργασίας.



Μηχανισμός κλήσης συναρτήσεων

Όταν μία συνάρτηση f1() καλεί κάποια άλλη f2(), ο μηχανισμός κλήσης των γλωσσών C/C++ προβαίνει στις ακόλουθες ενέργειες:

1. Τοποθετεί στη στοίβα τις τιμές των παραμέτρων που μεταβιβάζονται από την f1() στην f2(). Η διάταξη τοποθέτησης ορίζει ότι πρώτα θα τοποθετηθεί η *τελευταία παράμετρος* και *τελευταία η πρώτη*, έτσι ώστε στο άκρο της στοίβας να βρίσκεται πάντα η πρώτη παράμετρος. Αυτό είναι ουσιώδες για να υποστηρίζονται παράμετροι με μεταβλητό πλήθος παραμέτρων, όπως η

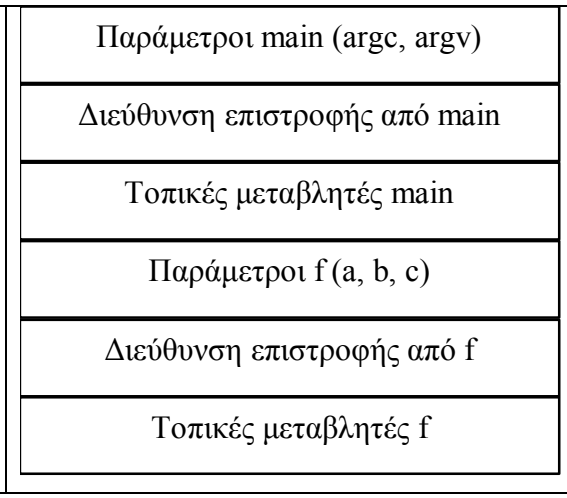
printf(). Ειδικότερα αν ο τύπος επιστροφής της f2 δεν χωράει σε έναν καταχωρητή (ή σε ζεύγος καταχωρητών, κατά περίπτωση) στη στοίβα δεσμεύεται επαρκής χώρος για την τιμή του αποτελέσματος.

2. Εκτελείται μία εντολή γλώσσας μηχανής «CALL f2» με αποτέλεσμα στη στοίβα να τοποθετούνται τα περιεχόμενα του δείκτη εντολών προγράμματος και –πιθανώς- άλλων καταχωρητών, όπως ορίζεται από την αρχιτεκτονική του επεξεργαστή.
3. Οι πρώτες εντολές της συνάρτησης f2() (εισάγονται από τον μεταγλωττιστή κατά την παραγωγή του κώδικα μηχανής) φροντίζουν να δεσμευτεί στη στοίβα επαρκής χώρος για τις τοπικές μεταβλητές.

Κατά την επιστροφή από την f2(), αυτή πρώτα θα καθαρίσει τη στοίβα από τις τοπικές της μεταβλητές και εν συνεχεία θα εκτελέσει μία εντολή γλώσσας μηχανής RETURN, προκειμένου να φορτώσει στον δείκτη εντολών προγράμματος την αποθηκευμένη τιμή (πιθανώς να φορτώνεται και η αποθηκευμένη τιμή άλλων καταχωρητών, αν αυτό προβλέπεται από το υλικό). Τέλος η f1() θα καθαρίσει τη στοίβα από τις τιμές των παραμέτρων που είχε τοποθετήσει εκεί.

Στο ακόλουθο σχήμα απεικονίζεται ένα απόσπασμα προγράμματος C καθώς και η κατάσταση της στοίβας όταν εκτελείται το σώμα της συνάρτησης f(), η οποία συνάρτηση έχει κληθεί άμεσα από τη main.

```
int status = 0; int tbl[5000];
void f(int a, char *b, int c[1000]) {
    int i, j; double sum; …; return; }
int main(int argc, char *argv[]) {
    int x = 7, y = 2; int t[2000];
    …
    f(x + y, "str", &t[999]);
}
```



Υπερχείλιση μνήμης για τοπικές μεταβλητές

Έχοντας υπόψη τη διάταξη της μνήμης και τον μηχανισμό κλήσεων συναρτήσεων, ας θεωρήσουμε την περίπτωση του κώδικα που απεικονίζεται στο ακόλουθο σχήμα:

```
int testPassword(void) {
int password_correct = 0, attempts = 0;
char buf[16];

while ((attempts < 3) && (password_correct == 0)) {
        printf("Enter password: ");
        gets(buf);
        if (strcmp(buf, "secret") == 0)
                password_correct = 1;
        else
                attempts++;
}
if (password_correct) return 1;
else return 0;
}
```

Ο κώδικας ζητάει από τον χρήστη ένα συνθηματικό και αποθηκεύει την απάντησή του στον πίνακα buff μεγέθους 16 bytes. Το βασικό ερώτημα είναι τι θα συμβεί αν ο χρήστης δώσει *πάνω από 16 χαρακτήρες* ως συνθηματικό. Βάσει του μηχανισμού κλήσης συναρτήσεων, οι τοπικές μεταβλητές password_correct, attempts και buf αποθηκεύονται στη στοίβα, ας υποθέσουμε στις εξής διευθύνσεις (σημειώστε ότι η στοίβα μεγαλώνει *προς τα κάτω*):

Μεταβλητή	Διεύθυνση
password_correct	1000-1003
attempts	996-999
buff	980-995

Αν τώρα ο χρήστης εισάγει ένα συνθηματικό μεγέθους 24 χαρακτήρων, έστω 24 επαναλήψεις του κεφαλαίου λατινικού Α, τα 16 πρώτα Α θα αποθηκευθούν στα όρια του buff, τα 4 επόμενα στη μεταβλητή attempts και τα 4 επόμενα στη μεταβλητή password_correct (επίσης θα αποθηκευθεί και ένα μηδενικό στη θέση 1004 – σε αυτό θα επανέλθουμε στη συνέχεια). Οι δύο ακέραιες μεταβλητές έχοντας κάθε ένα από τα 4 bytes τους να ισούνται με 65 (ο ASCII κωδικός του Α) διερμηνεύονται από τη C ως έχουσες την τιμή 1094795585 – έτσι η συνθήκη της εντολής "if (password_correct)" είναι αληθής και η συνάρτηση επιστρέφει 1!

Αντίστοιχο πρόβλημα θα αντιμετωπίσουμε και με τον κώδικα του ακόλουθου σχήματος:

```
int testPassword(void) {
        char thePass[] = "secret";
        char userPass[7];
        int passlen = strlen(thePass);

        printf("Enter password:");
        gets(userPass);
        if (strncmp(thePass, userPass, passlen) == 0)
                return 1;
        else
                return 0;
}
```

Εδώ αν ο χρήστης εισάγει «aaaaaaaaaaaaaaaa» (πάνω από 14 επαναλήψεις του a) η συνάρτηση και πάλι επιστρέφει 1.

Και στις δύο περιπτώσεις το πρόβλημα δημιουργήθηκε γιατί έχουμε δεσμεύσει κάποιο χώρο για την αποθήκευση της εισόδου τους χρήστη, αλλά η συνάρτηση gets() δεν κάνει οποιονδήποτε έλεγχο για το αν οι περιορισμοί χώρου πληρούνται κατά την αποθήκευση της εισόδου στη μεταβλητή.

Υπερχείλιση μνήμης για τοπικές μεταβλητές

Στα προηγούμενα παραδείγματα η επικάλυψη των δεδομένων της στοίβας σταμάτησε στα όρια των τοπικών μεταβλητών της συνάρτησης. Αυτό δεν είναι πάντα απαραίτητο: αν η υπερχείλιση είναι αρκετά μεγάλη, τότε η επικάλυψη των δεδομένων θα συνεχιστεί και πέρα από αυτά, στον χώρο όπου βρίσκεται η *διεύθυνση επιστροφής* από τη συνάρτηση και τυχόν άλλοι καταχωρητές. Κατά συνέπεια, όταν εκτελεστεί η εντολή *return* της συνάρτησης ο έλεγχος θα μεταφερθεί όχι στη συνάρτηση που την κάλεσε, αλλά όπου υποδεικνύουν τα bytes που έχουν αντικαταστήσει την αρχική διεύθυνση επιστροφής.

Στην «καλή» περίπτωση, τα bytes αυτά θα σχηματίζουν μία άκυρη διεύθυνση, οπότε το πρόγραμμα θα τερματιστεί άμεσα από το λειτουργικό σύστημα, ή κάποια έγκυρη διεύθυνση με τυχαία bytes, οπότε το συνηθέστερο είναι να εκτελούνται λίγες εντολές πριν την οριστική κατάρρευση.. Στην «κακή» περίπτωση τα bytes είναι κατάλληλα επιλεγμένα από τον εισβολέα ώστε:

- είτε να δείχνουν σε κώδικα μηχανής που ο ίδιος έδωσε ως δεδομένα (και έχει αποθηκευθεί στη στοίβα!)
- είτε να καλούν επιλεγμένες κλήσεις συστήματος εφοδιασμένες με κατάλληλες παραμέτρους (τις οποίες ο εισβολέας παρέχει στα *επόμενα bytes* από τη διεύθυνση επιστροφής) – π.χ. exec("sh", "/bin/sh", NULL);

Οι συνηθέστερες αιτίες υπερχείλισης μνήμης

Η γλώσσα C παρέχει αρκετές συναρτήσεις βιβλιοθήκης που είναι επιρρεπείς σε εμφάνιση υπερχείλιση μνήμης. Μερικές από αυτές φαίνονται στον ακόλουθο πίνακα μαζί με σχετικά σχόλια και τους επικρατέστερους «ασφαλείς αντικαταστάτες».

Συνάρτηση	Σχόλια	Αντικαταστάτης
gets(myStr);	Κανένας έλεγχος στο πλήθος των αποθηκευόμενων bytes	fgets(myStr, 80, stdin);
scanf("%s", mystr);	Κανένας έλεγχος στο πλήθος των αποθηκευόμενων bytes	scanf("%80s", mystr);
sprintf(myStr, "%d%s%f", …);	Δεν διασφαλίζεται ότι τα bytes που θα αποθηκευθούν χωράνε στο mystr	snprintf(myStr, 80, "%d%s%f", …); sprintf(myStr, "%5d%.65s%8.2f", …);
strcpy(prayItFits, bigOne);	Δεν διασφαλίζεται ότι το μήκος της bigOne είναι μικρότερο από τη χωρητικότητα του prayItFits	strncpy(prayItFits, bigOne, 80);
strcat(smallStr, possiblyBig);	Δεν διασφαλίζεται ότι το μήκος της bygone *συν το τρέχον μήκος του* prayItFits είναι μικρότερο από τη χωρητικότητα του prayItFits	strncat(smallStr, possiblyBig, 80 – strlen(smallStr));
while ((c = getchar()) != '\n') str[idx++] = c;	Η getchar είναι ασφαλής, αλλά στον βρόχο δεν γίνεται ο κατάλληλος έλεγχος χωρητικότητας	while ((idx < 80) && ((c = getchar()) != '\n')) str[idx++] = c;
char s[256]; getcwd(s, PATH_MAX);	Η getcwd είναι ασφαλής, αλλά έχουμε δεσμεύσει (δυνητικά) λίγο χώρο.	char s[PATH_MAX + 1]; getcwd(s, PATH_MAX);

Υπερχείλιση μνήμης στον σωρό

Η υπερχείλιση μνήμης στον σωρό είναι συνήθως συνυφασμένη με αναποτελεσματικό έλεγχο της υπερχείλισης των πράξεων ακεραίων. Θεωρήστε το ακόλουθο πρόγραμμα:

```
printf("Enter array size: ");
scanf("%d", &numElements);
numBytes = numElements * sizeof(int);
if ((myArray = malloc(numBytes)) == NULL) {
        perror("Out of memory");
        exit(1);
}
for (i = 0; i < numElements; i++)
        myArray[numElements – 1] = 0;
```

που δεσμεύει και αρχικοποιεί έναν πίνακα ακεραίων. Το πρόβλημα με το πρόγραμμα είναι ότι δεν ελέγχει αποτελεσματικά την περίπτωση που το εισαχθέν από το χρήστη πλήθος στοιχείων πολλαπλασιαζόμενο με το μέγεθος του ακεραίου ξεπερνά την μέγιστη τιμή του μη προσημασμένου ακεραίου, οπότε έχουμε ή αναδίπλωση, ή επιστροφή του μέγιστου μη προσημασμένου ακεραίου, ανάλογα με την υλοποίηση. Ο σωστός κώδικας απεικονίζεται στο σχήμα που ακολουθεί:

```
printf("Enter array size: ");
scanf("%d", &numElements);
if (numElements > UINT_MAX / 4) {
        fprintf(stderr, "Array too large!\n");
        exit(1);
}
numBytes = numElements * sizeof(int);
if ((myArray = malloc(numBytes)) == NULL) {
        perror("Out of memory");
        exit(1);
}
for (i = 0; i < numElements; i++)
        myArray[numElements – 1] = 0;
```

Το τελικό αποτέλεσμα από τον πλημμελή έλεγχο ορίων είναι ότι και πάλι μπορεί να επικαλυφθεί η στοίβα ή άλλα δεδομένα στον σωρό.

4.3.1.2 Παράλειψη προσδιοριστή μορφής στην printf

Όταν η printf/fprintf κ.λπ. τυπώνει μία συμβολοσειρά και μόνο, πολλοί προγραμματιστές παραλείπουν το *"%s"* ως προσδιοριστή μορφής και παραθέτουν μόνο τη συμβολοσειρά. Η συμβολοσειρά τυπώνεται στη γενική περίπτωση, αλλά αν περιέχει ενδείξεις εκτύπωσης παραμέτρων (π.χ. %s, %d κ.λπ), η printf θα αναζητήσει τις τιμές τους στη στοίβα. Αντί για τιμές παραμέτρων της printf (που δεν υπάρχουν) εκεί βρίσκονται άλλα δεδομένα, όπως διευθύνσεις επιστροφής, τοπικές μεταβλητές, παράμετροι προς τη συνάρτηση που καλεί την printf κ.ο.κ., με αποτέλεσμα να έχουμε πιθανή διαρροή πληροφοριών. Έτσι, για να τυπώσουμε μία συμβολοσειρά s θα πρέπει *πάντα* να γράφουμε printf("%s", s); (ή να την τυπώνουμε με εναλλακτικό τρόπο, π.χ. puts(s)).

4.3.2 Συνθήκες ανταγωνισμού

Ο όρος *συνθήκες ανταγωνισμού* αναφέρεται στις περιπτώσεις όπου δύο διεργασίες αλληλεπιδρούν με μη προβλεπόμενο τρόπο, κυρίως λόγω χρονισμού ενεργειών που

αφορούν το ίδιο αντικείμενο. Η βασική αιτία πίσω από τις συνθήκες ανταγωνισμού είναι είτε το γεγονός ότι ένα αντικείμενο ελέγχεται σε κάποια στιγμή και χρησιμοποιείται αργότερα ενώ στο μεσοδιάστημα έχει αλλάξει, είτε ότι μία εννοιολογικά αδιαίρετη λειτουργία διασπάται σε μικρότερα κομμάτια, δίνοντας έδαφος για εξωτερικές παρεμβάσεις. Συγκεκριμένες περιπτώσεις συνθηκών ανταγωνισμού μπορεί να έχουν σοβαρές επιπτώσεις στην ασφάλεια του συστήματος.

Μία από τις πρώτες συνθήκες ανταγωνισμού που ανακαλύφθηκαν (και αξιοποιήθηκαν από τους εισβολείς) ήταν η πρώτη υλοποίηση της δημιουργίας καταλόγων στο Unix που γινόταν με τον ακόλουθο κώδικα:

```
mknod(path, S_IFDIR, dev);
/* Υπάρχει ο κατάλογος, ιδιοκτησία του root και είναι κενός */
chown(path, uid, gid); /* Αλλαγή ιδιοκτήτη*/
chdir(path);
link(path, "."); /* Δημιουργία καταχώρησης. */
link(parent, ".."); /* Δημιουργία καταχώρησης .. */}
```

Η αξιοποίηση του κενού ασφάλειας στην υλοποίηση αυτή (που εκτελείται με τα προνόμια του υπερχρήστη) συνίσταται στην εκτέλεση του κάτωθι κώδικα:

```
while true
do
nice -39 mkdir foo &
rm -rf foo; ln /etc/passwd foo
rm -fr foo &
ls -l /etc/passwd
done
```

Ο στόχος του κακόβουλου κώδικα είναι η επίτευξη της εκτέλεσης της γραμμής

```
rm -rf foo; ln /etc/passwd foo
```

μετά την mknod και *πριν* την chown. Το αποτέλεσμα είναι να αλλάξει η ιδιοκτησία του αρχείου /etc/passwd και να περιέλθει στον κακόβουλο χρήστη.

Μια πιο γενική περίπτωση συνθήκης ανταγωνισμού είναι η χρήση της συνάρτησης access για έλεγχο του αν σε προγράμματα παραχώρησης ταυτότητας χρήστη ο *πραγματικός χρήστης* έχει δικαιώματα πρόσβασης σε ένα αρχείο ή όχι· αν διαπιστωθεί ότι ο πραγματικός χρήστης έχει το σχετικό δικαίωμα τότε το πρόγραμμα το ανοίγει. Ο έλεγχος αυτός συνήθως κωδικοποιείται ως ακολούθως:

```
if(access(theFile, R_OK) == 0) {
        /* User has read permission */
        if((fd = open(theFile, O_RDONLY)) < 0){
        perror(theFile);
                exit(-1);
        }
/* print the file */
}
else perror(theFile);
```

Το πρόβλημα με τον κώδικα αυτό συνίσταται ότι τα δικαιώματα *του πραγματικού χρήστη* ελέγχονται με την *access*, ενώ το άνοιγμα πραγματοποιείται με την *open* (με δικαιώματα υπερχρήστη) σε μεταγενέστερο χρονικό σημείο, και είναι έτσι πιθανό στο μεσοδιάστημα το αρχείο να έχει αλλάξει. Ως παράδειγμα, ας θεωρήσουμε την

περίπτωση όπου ο ανωτέρω κώδικας ενσωματώνεται στην εντολή εκτύπωσης lpr, και ο χρήστης εκτελεί σε ένα παράθυρο ένα πρόγραμμα C με το εξής περιεχόμενο:

```
while(1) {
        creat("harmless", 0644);
        unlink("harmless");
        symlink("/etc/shadow", "harmless");
}
```

και σε ένα άλλο παράθυρο την εντολή

```
repeat 10000 lpr harmless
```

με στόχο να επιτευχθεί η εκτέλεση των «unlink("harmless"); symlink("/etc/shadow", "harmless");» αμέσως μετά την access αλλά πριν την open. Λόγω του μεγάλου πλήθους των επαναλήψεων είναι βέβαιο ότι κάποτε το αποτέλεσμα θα επιτευχθεί, και έτσι θα διαβαστεί το αρχείο που κανονικά είναι προστατευμένο. Αντίστοιχες περιστάσεις μπορούν να προκύψουν και με εγγραφή, οπότε έχουμε αλλοίωση αρχείων.

Η αντιμετώπιση των συνθηκών ανταγωνισμού είναι ιδιαίτερα δύσκολη, διότι δεν είναι εύκολο να αναπαραχθούν. Ως γενικά μέτρα αντιμετώπισης ενδείκνυνται:

1. Η κωδικοποίηση των ατομικών λειτουργιών με κατάλληλους μηχανισμούς που εξασφαλίζουν τη μη διαιρετότητά τους π.χ. η δημιουργία καταλόγων έχει πλέον υλοποιηθεί ως κλήση συστήματος (mkdir()).
2. Να προσπελαύνεται το αντικείμενο και να γίνεται ο έλεγχος πάνω στο αντικείμενο που έχει αποκτηθεί, όταν αυτό είναι δυνατόν. Επί παραδείγματι, πολλά συστήματα υποστηρίζουν τη συνάρτηση faccess() που ελέγχει αν ο *πραγματικός χρήστης* έχει δικαιώματα πάνω στο *ανοικτό αρχείο*. Έτσι μία ασφαλής κωδικοποίηση του ελέγχου θα ήταν:

   ```
   if((fd = open(theFile, O_RDONLY)) < 0){
           perror(theFile);
           exit(1);
   }
   if(faccess(fd, R_OK) == 0) {
           /* User has read permission */
   /* print the file */
   }
   else {
           perror(theFile);
           exit(1);
   }
   ```

   Στο απόσπασμα αυτό κώδικα, ακόμη και αν μετακινηθεί το αρχείο δεν αντιμετωπίζεται πρόβλημα διότι ο περιγραφέας αρχείου fd αναφέρεται στο αρχείο που έχει ανοιχθεί.
3. Αποκλεισμός των συμβολικών συνδέσμων, έλεγχος δηλαδή ότι τα αρχεία που δίνονται ως παράμετροι στο πρόγραμμα δεν είναι συμβολικοί σύνδεσμοι. Η λύση αυτή δεν είναι ιδιαίτερα αποτελεσματική διότι (α) καταργεί τη λειτουργικότητα των συμβολικών συνδέσμων για συγκεκριμένες λειτουργίες αίροντας την ομοιομορφία χειρισμού τους (β) μπορεί να ξεπεραστεί χρησιμοποιώντας «κανονικούς» συνδέσμους και (γ) δεν υποστηρίζεται από όλα τα συστήματα.

4. Μετάβαση προσωρινά σε κατάσταση ελαττωμένων δικαιωμάτων για το άνοιγμα του αρχείου. Το χαρακτηριστικό αυτό εισήχθη στο πρότυπο POSIX μετά την πρώτη του έκδοση, οπότε ΄δεν υποστηρίζεται από κάποια συστήματα. Βάσει αυτής της λύσης, η εφαρμογή θέτει ως ενεργό ταυτότητα χρήστη την *πραγματική ταυτότητα χρήστη* προκειμένου για να ανοίξει το αρχείο και στη συνέχεια επανέρχεται σε καθεστώς αυξημένων προνομίων. Η λύση σκιαγραφείται στο ακόλουθο απόσπασμα κώδικα. Τα συστήματα που υποστηρίζουν τη λειτουργικότητα αυτή έχουν ορισμένη τη σταθερά _POSIX_SAVED_IDS κατά τη μεταγλώττιση.

```
uid_t euid, ruid;
gid_t egid, rgid;

euid = geteuid(); /* Αποθήκευση τρέχουσας ενεργού ταυτότητας χρήστη */
egid = getegid(); /* Αποθήκευση τρέχουσας ενεργού ταυτότητας ομάδας */
ruid = getuid(); /* Αποθήκευση τρέχουσας πραγματικής ταυτότητας χρήστη */
rgid = getgid(); /* Αποθήκευση τρέχουσας πραγματικής ταυτότητας ομάδας */

if(setegid(rgid) < 0) /* Η πραγματική ταυτότητα χρήστη τίθεται ως ενεργός */
        exit(1);
if(seteuid(ruid) < 0) /* Η πραγματική ταυτότητα χρήστη τίθεται ως ενεργός */
        exit(1);

open("...", ...);

if(setegid(egid) < 0) /* Αποκατάσταση αρχικής ενεργού ταυτότητας ομάδας */
        exit(1);
if(seteuid(euid) < 0) /* Αποκατάσταση αρχικής ενεργού ταυτότητας χρήστη */
        exit(1);
```

5. Δημιουργία μιας θυγατρικής διεργασίας η οποία απεκδύεται τα πρόσθετα προνόμια και ανοίγει το αρχείο με την ταυτότητα του χρήστη. Στη συνέχεια η διεργασία αυτή μεταβιβάζει στη γονική της τον *προσδιοριστή αρχείου* που έχει ανοίξει.

4.3.3 Προσωρινά αρχεία

Τα προσωρινά αρχεία είναι εξαιρετικά ύποπτα για διάφορα προβλήματα, συμπεριλαμβάνοντας δυσλειτουργία προγραμμάτων, διαρροή πληροφοριών κ.ο.κ.

Το πρώτο πρόβλημα που αρχικά αντιμετωπίσθηκε ήταν ότι τα προσωρινά αρχεία δημιουργούνταν σε καταλόγους που ήταν εγγράψιμοι από όλους (/tmp, /var/tmp). Βάσει της αρχικής σημασιολογίας του Unix, αν ένας χρήστης είχε δικαίωμα εγγραφής σε έναν κατάλογο, μπορούσε να διαγράψει οποιοδήποτε αρχείο εντός αυτού, ανεξαρτήτως δικαιωμάτων και ιδιοκτησίας του αρχείου. Έτσι, οποιοσδήποτε χρήστης θα μπορούσε να διαγράψει το προσωρινό αρχείο μίας διεργασίας δημιουργώντας της προβλήματα, ή να το αντικαταστήσει με κάποιο που θα μπορούσε ο ίδιος να διαβάσει, προκειμένου να υποκλέψει δεδομένα. Το ζήτημα αυτό αντιμετωπίσθηκε με τον εμπλουτισμό της σημασιολογίας των δικαιωμάτων καταλόγων, στον οποίο προστέθηκε ο χειρισμός του «save text» bit. Όταν το bit αυτό είναι ενεργό *για κατάλογο* και ο χρήστης έχει δικαίωμα εγγραφής σ' αυτόν, επιτρέπεται η δημιουργία νέων αρχείων στον κατάλογο, αλλά η διαγραφή περιορίζεται στα αρχεία των οποίων έχει την ιδιοκτησία (εκτός φυσικά αν είναι ο υπερχρήστης).

Ένα δεύτερο ζήτημα με τα προσωρινά αρχεία είναι ότι η διαδικασία δημιουργίας τους συνήθως περιλαμβάνει τη γέννηση ενός μοναδικού ονόματος αρχείου και εν συνεχεία τη δημιουργία του ίδιου του αρχείου:

```
char *tmp_name;
int tmpfd;
tmp_name = tmpnam(NULL);
if( (tmpfd = open(tmp_name, O_RDWR | O_CREAT | 0600)) < 0)
        exit(1);
unlink(tmp_name); /* Αδύνατη η αναφορά στο αρχείο, αυτόματη
διαγραφή του όταν τερματίσει το πρόγραμμα */
```

Εδώ βλέπουμε ότι τίθεται ξανά ένα θέμα συνθήκης ανταγωνισμού, καθώς το όνομα tmp_nam είναι *σίγουρα μοναδικό* κατά την παραγωγή του (όταν δηλ. εκτελείται η συνάρτηση tmpnam), αλλά δεν υπάρχει η εγγύηση ότι δεν έχει δημιουργηθεί μέχρι να εκτελεσθεί η open().

Μία λύση στο ζήτημα αυτό θα ήταν η χρήση της mkstemp, η οποία δημιουργεί *και ανοίγει* ένα προσωρινό αρχείο ως ατομική ενέργεια.

```
fd = mkstemp("/var/tmp/atempFile.XXXXXX");
fchmod(fd, 0600);
```

Η εντολή fchmod επιχειρεί να εξασφαλίσει ότι κανείς άλλος δεν έχει δικαίωμα πρόσβασης στο αρχείο, αλλά δυστυχώς ελλοχεύει ο κίνδυνος να προλάβει ο επιτιθέμενος να ανοίξει το αρχείο *αμέσως μετά* την mkstemp αλλά *πριν* την fchmod (οι περισσότερες υλοποιήσεις της mkstemp δημιουργούσαν αρχείο με δικαιώματα 0666 – οι νεώτερες λαμβάνουν υπόψη το umask που σπανίως όμως έχει κατάλληλη τιμή π.χ. 077). Στην αξιοποίηση του umask στηρίζεται έτσι η τρίτη λύση:

```
umask(066);
fd = mkstemp("/var/tmp/atempFile.XXXXXX");
```

Με τη ρύθμιση αυτή το νέο αρχείο θα έχει εξ αρχής δικαιώματα πρόσβασης που θα απαγορεύουν στους χρήστες της ομάδας και τους λοιπούς χρήστες οποιαδήποτε πρόσβαση. Μία εναλλακτική λύση είναι η αξιοποίηση της ένδειξης O_EXCL στην κλήση συστήματος open για τη δημιουργία αρχείου, η οποία υποδεικνύει ότι το άνοιγμα του αρχείου πρέπει να αποτύχει *αν το αρχείο ήδη υπάρχει.* Ο κώδικας έτσι διαμορφώνεται ως ακολούθως:

```
char *tmp_name;
int tmpfd;
tmp_name = tmpnam(NULL);
if( (tmpfd = open(tmp_name, O_RDWR | O_CREAT | O_EXCL | 0600)) < 0)
        exit(1);
unlink(tmp_name); /* Αδύνατη η αναφορά στο αρχείο, αυτόματη διαγραφή του
όταν τερματίσει το πρόγραμμα */
```

4.3.4 Μεταβλητές περιβάλλοντος

Κάθε πρόγραμμα έχει μεταβλητές περιβάλλοντος που περιέχουν συγκεκριμένες πληροφορίες, όπως π.χ. το όνομα του χρήστη, τη διαδρομή αναζήτησης εντολών, διαδρομή αναζήτησης βιβλιοθηκών κ.λπ. Οι μεταβλητές είναι προσπελάσιμες μέσω της μεταβλητής environ, η οποία είναι ένας πίνακας συμβολοσειρών με την ακόλουθη μορφή:

environ[0] = "USER=thomas";
environ[1] = "PATH=/usr/bin:/sbin:/usr/local/bin";
environ[2] = "LD_LIBRARY_PATH=/oracle/lib:/usr/lib";

Τα προγράμματα (ειδικότερα τα πιο προνομιούχα) ΔΕΝ πρέπει να βασίζονται στις τιμές αυτές, καθώς μπορούν να τροποποιηθούν από τους χρήστες π.χ.

USER=root; export USER

Αντί για τις μεταβλητές περιβάλλοντος θα πρέπει τα προγράμματα να βασίζονται σε *κλήσεις συστήματος* οι οποίες είναι αξιόπιστες. Έτσι η ακολουθία κλήσεων συστήματος

getpwuid(getuid())->pw_name

θα μας δώσει εγγυημένα αξιόπιστα αποτελέσματα, σε αντίθεση με την επισφαλή

getenv("USER");

Επίσης πρέπει να δίνεται ιδιαίτερη μέριμνα σε ειδικές μεταβλητές περιβάλλοντος όπως η PATH που ορίζει τους καταλόγους όπου αναζητούνται εντολές και η LD_LIBRARY_PATH που ορίζει τους καταλόγους όπου αναζητούνται βιβλιοθήκες, ειδικότερα κατά την εκτέλεση προνομιούχων προγραμμάτων. Για τον σκοπό αυτό τα περισσότερα συστήματα παρέχουν τη ρύθμιση SUPATH η οποία θέτει αυτόματα τη μεταβλητή PATH για προνομιούχα προγράμματα. Ο διαχειριστής του συστήματος οφείλει να εξασφαλίσει ότι μόνο αξιόπιστοι κατάλογοι μνημονεύονται στη μεταβλητή αυτή – ειδικότερα ο τρέχον κατάλογος (.) *δεν* πρέπει να μνημονεύεται. Επίσης, ο *φορτωτής δυναμικών βιβλιοθηκών* του συστήματος πρέπει να *αγνοεί* επανορισμούς κλήσεων συστήματος ή βασικών διαδικασιών βιβλιοθήκης που βρίσκονται σε μη «εγκεκριμένους» καταλόγους, ανεξάρτητα με τη διάταξη των καταλόγων στη μεταβλητή LD_LIBRARY_PATH. Για παράδειγμα, αν ο χρήστης δώσει την τιμή

LD_LIBRARY_PATH=/home/pika/libs

όπου υπάρχει η βιβλιοθήκη *mylib.so* που περιέχει την υλοποίηση συνάρτησης

uid_t geteuid(void) {return 0;}

ο φορτωτής πρέπει να την αγνοήσει και να χρησιμοποιήσει κανονικά την υλοποίηση που παρέχεται στη στάνταρ βιβλιοθήκη της C.

4.3.5 Περιορισμός πόρων

Η C και η C++ μας δίνουν τη δυνατότητα να περιορίσουμε ποσοτικά τους πόρους στους οποίους έχει πρόσβαση το πρόγραμμά μας. Οι περιορισμοί αυτοί μπορούν να συμβάλλουν σε δύο κατευθύνσεις αναφορικά με την επαύξηση της ασφάλειας:

1. *αποτροπή ή περιορισμός των επιπτώσεων επιθέσεων τύπου άρνησης παροχής υπηρεσιών*. Μία τέτοια επίθεση έχει ως στόχο συνήθως μία διεργασία, η οποία καταλαμβάνει όλους τους πόρους του συστήματος (μνήμη, πίνακας διεργασιών, πίνακας ανοικτών αρχείων κ.λπ.) για την εξυπηρέτηση ανούσιων αιτημάτων, καθιστώντας έτσι το σύστημα μη λειτουργικό. Περιορίζοντας τους πόρους στους οποίους έχει πρόσβαση η κάθε διεργασία, η οποιαδήποτε τέτοια επίθεση θα έχει περιορισμένα αποτελέσματα, μια και το λειτουργικό σύστημα δεν θα επιτρέψει στη διεργασία να καταλάβει όλους τους πόρους του συστήματος.

2. *αποτροπή διαρροής πληροφοριών*. Σε πολλά συστήματα όταν τερματίζεται κατά «μη κανονικό τρόπο» μία διεργασία δημιουργείται ένα *αρχείο αποτύπωσης μνήμης*, που στόχος του είναι να διευκολύνει τον εντοπισμό σφαλμάτων. Το αρχείο όμως αυτό δυνητικά περιέχει ευαίσθητες πληροφορίες π.χ. συνθηματικά, που δεν πρέπει να διαρρεύσουν. Ένας κακόβουλος χρήστης θα μπορούσε για παράδειγμα να προσπαθήσει να υποκλέψει το αρχείο κρυπτογραφημένων συνθηματικών προκαλώντας τον απότομο τερματισμό της εντολής passwd:

   passwd → pid = 2078
   kill –ABRT 2078 → stop and core dump
   strings core | grep root → extract root entry

   Η πιο αποτελεσματική άμυνα είναι να μην δημιουργείται καθόλου το αρχείο αυτό, περιορίζοντας το μέγιστο μέγεθός του στο μηδέν.

5.1 Φιλτράρισμα πακέτων

Η πρώτη βασική λειτουργία των firewalls είναι να «φιλτράρουν» τα δικτυακά πακέτα, επιτρέποντας επιλεκτικά την πρόσβαση σε συγκεκριμένους μόνο υπολογιστές και υπηρεσίες. Ας θεωρήσουμε ένα εταιρικό δίκτυο που διαθέτει τους ακόλουθους εξυπηρέτες: WWW, ηλεκτρονικού ταχυδρομείου, μεταφοράς αρχείων (FTP), εκτυπώσεων, αρχείων για Windows (CIFS) και αρχείων για UNIX (NFS). Πέρα από τους εξυπηρέτες, υπάρχουν και αρκετοί εξυπηρετούμενοι για χρήση του προσωπικού της εταιρίας. Σε ένα τέτοιο περιβάλλον θα μπορούσαμε να περιμένουμε ότι ισχύουν οι κάτωθι κανόνες πρόσβασης:

1. Οι εξωτερικοί ως προς το εταιρικό δίκτυο χρήστες μπορούν να προσπελάσουν:
   a. Τον εξυπηρέτη WWW στη θύρα 80, όπου προσφέρεται η υπηρεσία WWW
   b. Τον εξυπηρέτη ηλεκτρονικού ταχυδρομείου στη θύρα 25, όπου παρέχεται η υπηρεσία διακίνησης μηνυμάτων.
   c. Τον εξυπηρέτη μεταφοράς αρχείων στις θύρες 20 και 21.

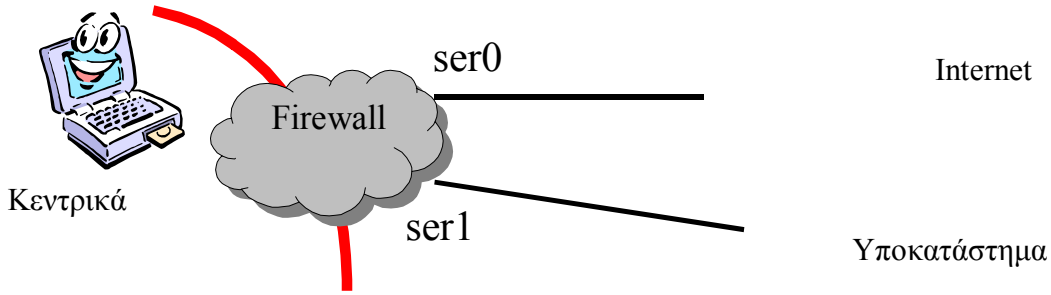
   Οι εξωτερικοί ως προς το εταιρικό δίκτυο χρήστες δεν μπορούν να προσπελάσουν *καθόλου* τους υπόλοιπους υπολογιστές (συμπεριλαμβανομένων των εξυπηρετών εκτυπώσεων, αρχείων για Windows και αρχείων για UNIX) ούτε τις υπόλοιπες θύρες εξυπηρέτησης στους τρεις ανωτέρω εξυπηρέτες.
2. Οι εσωτερικοί χρήστες μπορούν απρόσκοπτα να προσπελάσουν όλες τις υπηρεσίες σε όλους τους υπολογιστές του εταιρικού δικτύου και όλες τις υπηρεσίες σε υπολογιστές του διαδικτύου. (Κατά περίπτωση βέβαια, είναι δυνατόν να περιορισθεί και η πρόσβαση των εσωτερικών χρηστών στο διαδίκτυο είτε για λόγους ασφάλειας είτε για λόγους παραγωγικότητας.)

Προκειμένου να μπορέσει το firewall να περιορίσει την κυκλοφορία των πακέτων στο δίκτυο έχει στη διάθεσή του *κανόνες* οι οποίοι εξετάζουν διάφορα χαρακτηριστικά των δικτυακών πακέτων και υποδεικνύουν αν το πακέτο πρέπει να προωθηθεί στον προορισμό του ή να απορριφθεί. Οι κανόνες των firewalls εξετάζουν συνήθως τα ακόλουθα χαρακτηριστικά των πακέτων δικτύου:

- *Διεύθυνση αφετηρίας (IP, Port)*. Η διεύθυνση IP μπορεί να αντιστοιχεί σε ένα σύνολο υπολογιστών, καθορίζοντας έτσι ένα ολόκληρο *υποδίκτυο* αντί για κάποιον μεμονωμένο υπολογιστή. Επίσης, το Port μπορεί να καθορίζει μία περιοχή θυρών σύνδεσης αντί για μία συγκεκριμένη.
- *Διεύθυνση προορισμού (IP, Port)*. Όμοια με τη διεύθυνση αφετηρίας τα τμήματα IP και Port μπορεί να καθορίζουν ένα υποδίκτυο ή μια περιοχή θυρών σύνδεσης, αντίστοιχα.
- *Φυσική συσκευή λήψης του πακέτου*. Σε δρομολογητές που έχουν πάνω από μία δικτυακές διασυνδέσεις, η φυσική συσκευή λήψης του πακέτου εξετάζεται σε συνδυασμό με την αναγραφόμενη στο πακέτο διεύθυνση αφετηρίας, προκειμένου να εντοπισθούν πακέτα με *πλαστογραφημένη διεύθυνση αφετηρίας*. Η πιο συνηθισμένη περίπτωση τέτοιων πακέτων αφορά πακέτα που προέρχονται στην πραγματικότητα από το διαδίκτυο (συνεπώς λαμβάνονται από τη φυσική συσκευή που συνδέει το εταιρικό δίκτυο με το διαδίκτυο) αλλά αναφέρουν ψευδώς ως διεύθυνση αφετηρίας κάποια εσωτερική διεύθυνση του διαδικτύου, προκειμένου να τύχουν καλύτερης μεταχείρισης.
- *Πρωτόκολλο ανώτερου επιπέδου (TCP/UDP)*.
- *Δικτυακές ενδείξεις* (π.χ. αίτηση εγκαθίδρυσης σύνδεσης)

Αν τα χαρακτηριστικά του πακέτου του δικτύου ταιριάζουν με αυτά που περιγράφονται στον κανόνα, τότε το πακέτο τυγχάνει της μεταχείρισης που προσδιορίζει ο κανόνας (προώθηση ή απόρριψη).

Οι (απλοποιημένοι) κανόνες που θα μπορούσαν να ισχύουν για ένα firewall μιας τράπεζας (βλ. σχήμα) όπου το κεντρικό κατάστημα διαθέτει έναν δρομολογητή με δύο εξωτερικές συνδέσεις, μία προς το Internet και μία άλλη προς ένα υποκατάστημα έχουν ως εξής:



Φυσική συσκευή	Αφετηρία	Προορισμός	Πρωτόκολλο	Ενδείξεις	Ενέργεια
ser0	branch1/*	*/*	*	*	Deny
*	*/*	mailsrv/25	TCP	*	Allow
*	*	namesrv/143	TCP/UDP	*	Allow
ser1	branch1/*	oraclesrv/1525	TCP	*	Allow
*	*	*	*	*	Deny

Ο πρώτος κανόνας παρέχει προστασία έναντι πλαστογραφημένων πακέτων που έρχονται στην πραγματικότητα από το Internet αλλά εμφανίζονται να έχουν

διεύθυνση που αντιστοιχεί στο υποκατάστημα. Οι δύο επόμενες γραμμές επιτρέπουν σε όλους την πρόσβαση στον εξυπηρέτη ηλεκτρονικού ταχυδρομείου και στον εξυπηρέτη ονοματολογίας, ενώ η τέταρτη γραμμή δίνει στους υπολογιστές του υποκαταστήματος δικαίωμα πρόσβασης στον εξυπηρέτη βάσεων δεδομένων των κεντρικών. Η τελευταία γραμμή φροντίζει για την απόρριψη όλων των πακέτων που δεν ταιριάζουν με κάποιον από τους προηγούμενους κανόνες, υλοποιώντας ουσιαστικά την πολιτική «ό,τι δεν επιτρέπεται ρητώς, απαγορεύεται».

Το φιλτράρισμα πακέτων παρουσιάζει τα εξής σημαντικά πλεονεκτήματα:

1. Είναι πολύ απλό στην υλοποίηση
2. Ενσωματωμένο σε δικτυακές διατάξεις, όπως δρομολογητές, παρέχει εξαιρετικές επιδόσεις.

Από την άλλη πλευρά ωστόσο, το φιλτράρισμα πακέτων δεν δίνει τη δυνατότητα τήρησης αρχείων καταγραφής (τουλάχιστον όχι χωρίς σημαντικές επιπτώσεις στην απόδοση), είναι δύσκολο να ρυθμιστεί και να ελεγχθεί, –ιδίως για πολύπλοκες περιπτώσεις, ενώ δεν είναι ιδιαίτερα ευέλικτο ή άμεσα επεκτάσιμο. Τέλος μπορεί να παρακαμφθεί μέσω της τεχνικής που ονομάζεται «tunneling», δηλαδή της μεταφοράς πακέτων μιας απαγορευμένης υπηρεσίας (π.χ. *telnet*) μέσα από πακέτα μιας επιτρεπόμενης υπηρεσίας (π.χ. WWW). Η τεχνική αυτή απαιτεί και τη χρήση ειδικού λογισμικού στα δύο άκρα της επικοινωνίας, το οποίο όμως είναι εύκολο να βρεθεί και να εγκατασταθεί.

Στο σχήμα που ακολουθεί παρουσιάζεται ένα παράδειγμα κανόνων φιλτραρίσματος, βασισμένο στο ευρέως διαδεδομένο πακέτο του Linux, iptables

```
iptables -s 10.30.19.0/24 –i ! eth0 –j DENY –l
iptables -s 195.134.65.128/26 –i ! eth2 –j DENY –l

iptables -A forward -s 10.30.19.0/24 –i eth0 –o eth2 -j good-dmz
iptables -A forward -s 10.30.19.0/24 -i eth0 –o eth1 -j good-bad
iptables -A forward -s 195.134.65.128/26 -i eth2 –o eth1 -j dmz-bad
iptables -A forward -s 195.134.65.128/26 -i eth2 –o eth0 -j dmz-good
iptables -A forward -i eth0 -j bad-dmz
iptables -A forward -i eth2 -j bad-good
iptables -A forward -j DENY -l

iptables -A good-dmz -p tcp -d 195.134.65.183 smtp -j ACCEPT
iptables -A good-dmz -p tcp -d 195.134.65.183 pop3 -j ACCEPT
iptables -A good-dmz -j DENY -l

iptables -A dmz-good -p tcp ! -y -s 195.134.65.183 smtp -j ACCEPT
iptables -A dmz-good -p tcp ! -y -s 195.134.65.183 pop3 -j ACCEPT
iptables -A dmz-good -j DENY –l

iptables -A bad-dmz -p tcp -d 195.134.65.183 smtp -j ACCEPT
iptables -A bad-dmz -p tcp ! -y --sport 25 -d 195.134.65.183 -j ACCEPT
iptables -A bad-dmz -j DENY

iptables -A dmz-bad -p tcp -s 195.134.65.183 --dport smtp -j ACCEPT
iptables -A dmz-bad -p tcp ! –y -s 195.134.65.183 smtp -j ACCEPT
iptables -A dmz-bad -j DENY –l

iptables -A good-bad -p tcp --dport www -j MASQ
iptables –A good-bad -j DENY –l

iptables –A bad-good -j DENY
```

Η πρώτη ομάδα εντολών ασχολείται με την προστασία από πακέτα με παραποιημένες διευθύνσεις αφετηρίας: διευθύνσεις αφετηρίας της μορφής 10.30.19.0/24 μπορούν να

εμφανίζονται *μόνο* στη συσκευή *eth0*, ενώ διευθύνσεις αφετηρίας της μορφής 195.134.65.128/26 μπορούν να εμφανίζονται *μόνο* στη συσκευή *eth2*.

Οι έξι πρώτες εντολές της δεύτερης ομάδας διαχωρίζουν τα πακέτα που πρόκειται να δρομολογηθούν σε έξι κατηγορίες, ανάλογα με (α) τη διεύθυνση αφετηρίας (-s) και τη φυσική συσκευή προς την οποία θα προωθηθούν (-i). Για κάθε μία από αυτές, ορίζεται το σύνολο κανόνων που θα εφαρμοσθεί (good-dmz, good-bad κ.λπ.). Η τελευταία εντολή της δεύτερης ομάδας ορίζει ότι απορρίπτονται όλα τα υπόλοιπα υπό προώθηση πακέτα.

Η τρίτη ομάδα εντολών ρυθμίζει την κυκλοφορία που δρομολογείται από τη φυσική συσκευή eth0 προς τη φυσική συσκευή eth2 ($1^{η}$ εντολή $2^{ης}$ ομάδας). Ο πρώτος κανόνας είναι ότι επιτρέπεται η δρομολόγηση αν η διεύθυνση προορισμού είναι η 195.134.65.183 και η θύρα προορισμού είναι η smtp (25 - αποστολή μηνυμάτων ηλ. ταχυδρομείου). Ο δεύτερος κανόνας είναι ότι επιτρέπεται η δρομολόγηση αν η διεύθυνση προορισμού είναι η 195.134.65.183 και η θύρα προορισμού είναι η pop3 (110 – ανάγνωση μηνυμάτων ηλεκτρονικού ταχυδρομείου με το πρωτόκολλο Post Office Protocol έκδοση 3). Ο τρίτος κανόνας της ομάδας υποδεικνύει ότι οποιαδήποτε άλλη κυκλοφορία απαγορεύεται, υλοποιώντας έτσι την πολιτική «ό,τι δεν επιτρέπεται ρητώς, απαγορεύεται».

Η τέταρτη ομάδα εντολών ρυθμίζει την κυκλοφορία που δρομολογείται από τη φυσική συσκευή eth2 προς τη φυσική συσκευή eth0. Ο πρώτος κανόνας ορίζει ότι επιτρέπεται η δρομολόγηση πακέτων που αποστέλλονται από τη θύρα smtp του υπολογιστή 195.134.65.183 *μόνο εφ' όσον δεν πρόκειται για πακέτα εγκαθίδρυσης σύνδεσης* (! –y), επιτρέποντας έτσι ουσιαστικά μόνο στον εξυπηρέτη ηλεκτρονικού ταχυδρομείου να απαντήσει σε πακέτα που έχει δεχθεί λόγω του $1^{ου}$ κανόνα της δεύτερης ομάδας. Ο ίδιος ο εξυπηρέτης ηλεκτρονικού ταχυδρομείου, δεν μπορεί να αιτηθεί σύνδεση προς υπολογιστή πίσω από τη δικτυακή συσκευή eth0 (αυτό δεν περιλαμβάνεται στην κανονική λειτουργία του).

Η έβδομη ομάδα εντολών ρυθμίζει την κυκλοφορία που δρομολογείται από τη φυσική συσκευή eth0 προς τη φυσική συσκευή eth1. Η πρώτη εντολή της ομάδας αυτής ορίζει ότι αιτήσεις για τέτοια δρομολόγηση που προορίζονται για θύρα www (80) θα *μεταμφιέζονται*, δηλ. θα αποκτούν ως διεύθυνση αφετηρίας τη διεύθυνση του firewall και θα προωθούνται, ενώ ο επόμενος κανόνας ορίζει ότι υπόλοιπες αιτήσεις θα απορρίπτονται.

Τέλος, η όγδοη ομάδα εντολών αποτελείται από έναν μόνο κανόνα που ορίζει ότι όλες οι δρομολογήσεις από τη φυσική συσκευή eth1 προς τη φυσική συσκευή eth0 απορρίπτονται. Σημειώνουμε ότι αυτό δεν επηρεάζει τις αιτήσεις που έχουν μεταμφιεστεί δυνάμει της $7^{ης}$ ομάδας εντολών, καθώς αυτές τυγχάνουν επεξεργασίας απ' ευθείας από το υποσύστημα μεταμφίεσης.

Μία επέκταση της τεχνικής φιλτραρίσματος είναι το *δυναμικό φιλτράρισμα πακέτων*, το οποίο παράλληλα με τους κανόνες πρόσβασης που έχουν τεθεί στο firewall εξετάζονται και οι κανόνες του *δικτυακού πρωτοκόλλου* (π.χ. TCP) που χρησιμοποιείται στην επικοινωνία. Για παράδειγμα, θα μπορούσαν να απορρίπτονται πακέτα που καταφθάνουν χωρίς να έχει εγκαθιδρυθεί σχετική σύνδεση, καθώς και πακέτα που καταφθάνουν μετά την καταστροφή της σχετικής σύνδεσης ή πακέτα που απαντούν σε ερωτήσεις που ποτέ δεν έγιναν. Το μοναδικό μειονέκτημα αυτής της πρακτικής είναι ότι πρέπει να παρακολουθεί την κατάσταση όλων των συνδέσεων μεταξύ του διαδικτύου και του εταιρικού δικτύου, κάτι που απαιτεί πολλή μνήμη. Η

απαίτηση αυτή αξιοποιείται μερικές φορές από τους επιτιθέμενους, οι οποίοι εξαπολύουν επιθέσεις που εγκαθιδρύουν και διατηρούν ενεργές πολλές συνδέσεις, με αποτέλεσμα να εξαντλούνται οι πόροι του μηχανήματος που υλοποιεί το φιλτράρισμα και έτσι αυτό να καθίσταται ανενεργό.

5.4 Χρήση «περιτυλιγμάτων» υπηρεσιών

Τα περιτυλίγματα υπηρεσιών είναι ανεξάρτητα προγράμματα που χρησιμοποιούνται για να ελέγχουν την πρόσβαση σε άλλα προγράμματα ή υπηρεσίες. Η πραγματοποίηση των σχετικών ελέγχων με την τεχνική του περιτυλίγματος αποσκοπεί στον διαχωρισμό των λειτουργιών ελέγχου πρόσβασης από τις καθ' εαυτό λειτουργίες των προγραμμάτων ή υπηρεσιών που επιθυμούμε να προστατευθούν, ενώ επίσης μπορούμε να προστατεύσουμε και προγράμματα ή υπηρεσίες που είχαν αναπτυχθεί χωρίς να έχουν κατά νου την ασφάλεια και έτσι δεν ενσωματώνουν χαρακτηριστικά ελέγχου πρόσβασης. Ο διαχωρισμός επιτρέπει επίσης την ανεξάρτητη ανάπτυξη, έλεγχο και συντήρηση του περιτυλίγματος, ενώ τέλος ένα μόνο περιτύλιγμα μπορεί να χρησιμοποιηθεί για να ελέγχει την πρόσβαση σε πολλές υπηρεσίες, εξοικονομώντας έτσι πόρους συστήματος και μειώνοντας τη διαχειριστική επιβάρυνση.

Τα περιτυλίγματα μπορούν να υλοποιούν καταγραφή συνδέσεων, ενώ παράλληλα να εξετάζουν διάφορες παραμέτρους πριν επιτρέψουν κάποια χρήση προγράμματος ή υπηρεσίας. Οι έλεγχοι μπορούν να περιλαμβάνουν διάφορα χαρακτηριστικά, όπως η διεύθυνση και το όνομα του υπολογιστή απ' όπου προέρχεται η αίτηση, η ταυτότητα του χρήστη που επιθυμεί να χρησιμοποιήσει το πρόγραμμα ή την υπηρεσία κ.λπ. Το αν η χρήση της υπηρεσίας επιτρέπεται ή απαγορεύεται τελικά αποφασίζεται βάσει ενός συνόλου κανόνων.

Το πιο διαδεδομένο πρόγραμμα περιτύλιξης υπηρεσιών είναι το πακέτο *tcpd*, το οποίο χρησιμοποιείται συνήθως σε υπολογιστές τύπου Unix. Το πακέτο tcpd υποδέχεται όλες τις αιτήσεις για χρήση υπηρεσιών στον υπολογιστή (π.χ. telnet, ftp, printer κ.λπ.) και διενεργεί τους σχετικούς ελέγχους βάσει των κανόνων που περιγράφονται σε δύο αρχεία, τα */etc/hosts.allow* και */etc/hosts.deny*. Αν οι έλεγχοι είναι επιτυχής, ο έλεγχος μεταβιβάζεται στο σχετικό πρόγραμμα παροχής υπηρεσίας, αν όμως όχι η σύνδεση κλείνεται άμεσα και το πρόγραμμα παροχής υπηρεσίας δεν καλείται καθόλου.

Ένα παράδειγμα αρχείων hosts.allow και hosts.deny παρατίθεται στα σχήματα :

```
imapd: .mycom.com .affliatecom.com LOCAL
in.telnetd: .mycom.com .affliatecom.com
in.telnetd: 195.170.21.138 192.168.3.9/24
sshd: 60.70.80. 10.0.0.0/255.0.0.0
ALL: .mycom.com .affliatecom.com goodgye.somewhere.org
```

Παράδειγμα αρχείου hosts.allow

```
ALL: UNKNOWN: (/usr/sbin/safe_finger -l @%a | \
     /usr/ucb/mail -s %H-%d-%h root)&
ALL: 143.233.160.99: (/usr/sbin/safe_finger -l @%a | \
     /usr/ucb/mail -s %H-%d-%h root)&
ALL: PARANOID
ALL: .hackers.org .rivals.com 195.134.79.73
imapd: ALL
```

Παράδειγμα αρχείου hosts.deny

Παρατηρούμε ότι το αρχείο hosts.allow περιέχει γραμμές της ακόλουθης μορφής:

Όνομα_υπηρεσίας: λίστα_επιτρεπόμενων_διευθύνσεων

Το *Όνομα_υπηρεσίας* είναι στην ουσία το όνομα του προγράμματος παροχής της συγκεκριμένης υπηρεσίας, ενώ η *λίστα_επιτρεπόμενων_διευθύνσεων* παραθέτει δικτυακές ή συμβολικές διευθύνσεις υπολογιστών που επιτρέπεται να χρησιμοποιήσουν την συγκεκριμένη υπηρεσία. Η κάθε διεύθυνση μπορεί να καθορίζεται ως:

Τρόπος	***Παράδειγμα***	***Σχόλια***
Μοναδική συμβολική διεύθυνση	`goodgye.somewhere.org`	Ο υπολογιστής με το συγκεκριμένο συμβολικό όνομα
Περιοχή συμβολικών ονομάτων	`.mycom.com`	Όλοι οι υπολογιστές των οποίων το συμβολικό όνομα τελειώνει με την παρατιθέμενη συμβολοσειρά
Μοναδική δικτυακή διεύθυνση	`195.170.21.138`	Ο υπολογιστής με τη συγκεκριμένη δικτυακή διεύθυνση
Περιοχή δικτυακών διευθύνσεων	`60.70.80.`	Οι υπολογιστές που τα τρία πρώτα ψηφία της δικτυακής τους διεύθυνσης είναι `60.70.80.`
Περιοχή δικτυακών διευθύνσεων	`192.168.3.9/24`	Οι υπολογιστές που τα 24 πρώτα bits της δικτυακής τους διεύθυνσης είναι ίσα με τα 24 πρώτα bits της διεύθυνσης `192.168.3.9`
Περιοχή δικτυακών διευθύνσεων	`10.0.0.0/255.0.0.0`	Οι υπολογιστές με διεύθυνση a.b τέτοια ώστε (x & `255.0.0.0`) == `10.0.0.0` & `255.0.0.0`)

Τόσο το όνομα υπηρεσίας όσο και οποιοδήποτε μέλος της λίστας διευθύνσεων μπορεί να είναι η λέξη `ALL` που δηλώνει *όλες τις υπηρεσίες* ή *όλες τις διευθύνσεις* αντίστοιχα.

Ειδικότερα για τα μέλη της λίστας διευθύνσεων μπορούν να χρησιμοποιηθούν οι κάτωθι συμβολισμοί:

Συμβολισμός	***Σημασία***
`UNKNOWN`	Οι υπολογιστές των οποίων η συμβολική διεύθυνση δεν είναι γνωστή. Όταν γίνεται η σύνδεση, το περιτύλιγμα έχει στη διάθεσή του τη δικτυακή διεύθυνση του απομακρυσμένου υπολογιστή και προσπαθεί *πάντα* να βρει από αυτή τη

	συμβολική διεύθυνση, συνήθως με ένα ερώτημα αντίστροφης επίλυσης (reverse resolution). Αν αυτό δεν δώσει αποτέλεσμα, ο υπολογιστής χαρακτηρίζεται *UNKNOWN*.
PARANOID	Αν για μία σύνδεση βρεθεί το συμβολικό όνομα ενός υπολογιστή, τότε επιχειρείται και ένα ερώτημα ευθείας επίλυσης για τη συμβολική διεύθυνση για την εύρεση της δικτυακής. Υπό κανονικές συνθήκες, το ερώτημα αυτό θα πρέπει να έχει ως απάντηση τη δικτυακή διεύθυνση του υπολογιστή από τον οποίο έγινε η σύνδεση. Αν το αποτέλεσμα είναι διαφορετικό (κάτι που πολλές φορές οφείλεται είτε σε κακή ρύθμιση ή σε σκόπιμη προσπάθεια εξαπάτησης), ο υπολογιστής χαρακτηρίζεται ως *PARANOID*.

Οι συμβολισμοί UNKNOWN και PARANOID χρησιμοποιούνται συνηθέστερα στο hosts.deny.

Η σειρά επεξεργασίας των κανόνων στα δύο αρχεία, τέλος, είναι:

1. Πρώτα εξετάζεται το αρχείο *hosts.allow* και, αν βρεθεί ταίριασμα, η πρόσβαση επιτρέπεται.
2. Ακολούθως εξετάζεται το αρχείο *hosts.deny* και, αν βρεθεί ταίριασμα, η πρόσβαση απαγορεύεται.
3. Αν δεν βρεθεί ταίριασμα σε κανένα αρχείο, η πρόσβαση επιτρέπεται.

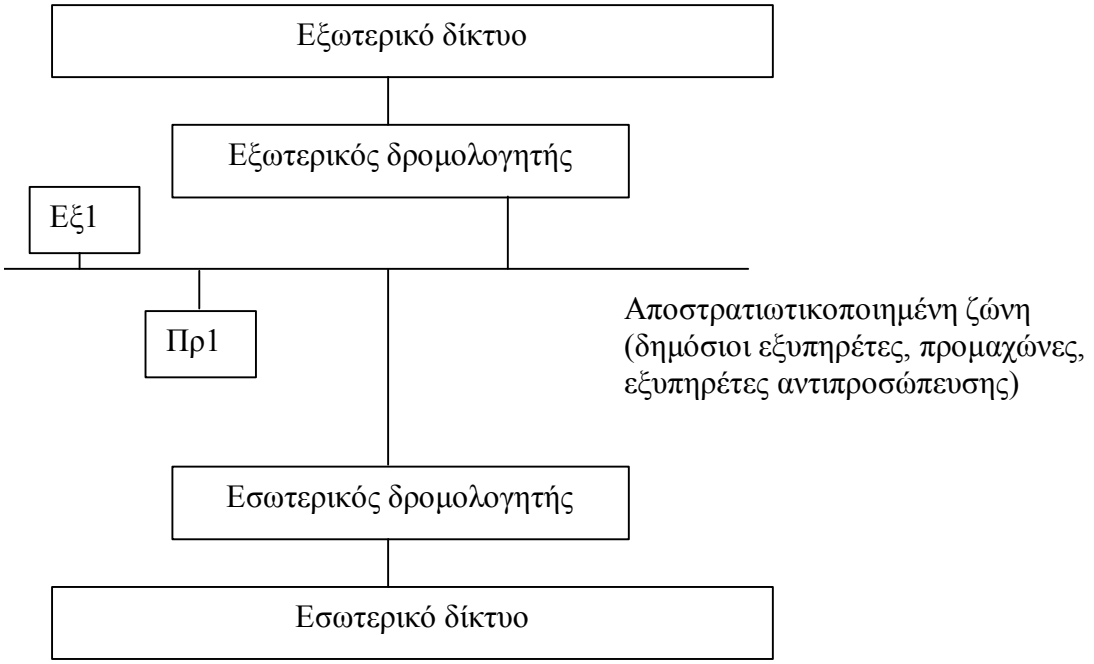
5.5 Σχεδιασμός τοπολογίας δικτύου

Έχοντας στη διάθεσή μας το φιλτράρισμα πακέτων, ένα ζήτημα που συνήθως τίθεται είναι η διαμόρφωση της τοπολογίας του δικτύου προκειμένου να μεγιστοποιείται η ασφάλεια. Η πιο διαδεδομένη προσέγγιση προς το ζήτημα αυτό είναι ο διαχωρισμός των υπολογιστών/ενεργών συσκευών σε τρεις κατηγορίες:

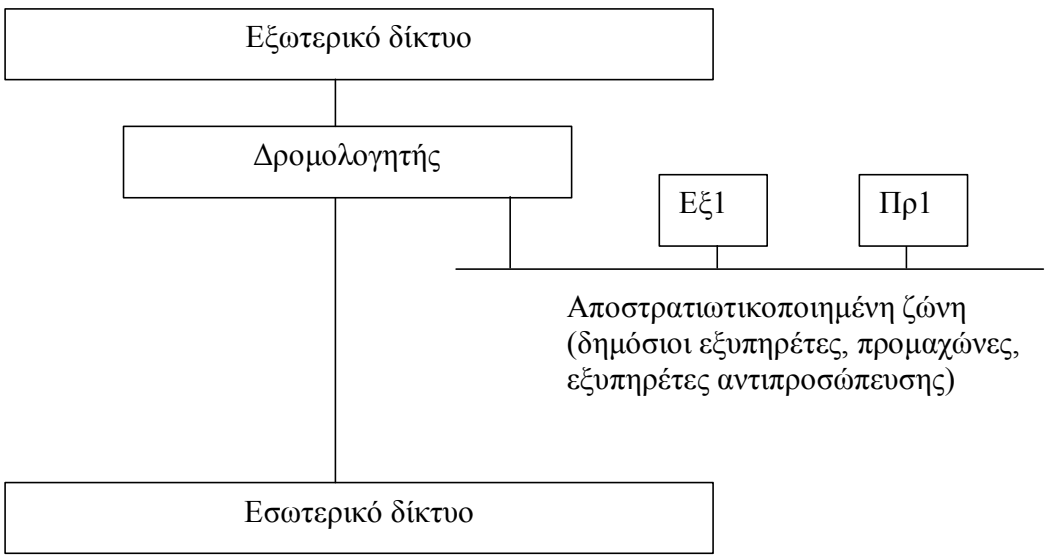
1. *Εξωτερικό δίκτυο*. Είναι οι υπολογιστές/συσκευές που βρίσκονται εκτός του εταιρικού δικτύου, οι οποίοι θεωρούνται και οι πιο επικίνδυνοι.
2. *Εσωτερικό δίκτυο*. Είναι οι υπολογιστές/συσκευές που ανήκουν στο εταιρικό δίκτυο και δεν παρέχουν καμία υπηρεσία προς τους εξωτερικούς χρήστες – με άλλα λόγια, κανένας υπολογιστής του εξωτερικού δικτύου δεν μπορεί να συνδεθεί προς οποιονδήποτε υπολογιστή του εσωτερικού δικτύου. Ανάλογα με την πολιτική ασφάλειας *είναι πιθανό* να επιτρέπεται στους εσωτερικούς χρήστες να προσπελαύνουν υπηρεσίες σε εξωτερικούς εξυπηρέτες, π.χ. να προσπελαύνουν εξυπηρέτες ΠΠΠ (WWW). Στην περίπτωση αυτή επιτρέπεται στους εξωτερικούς εξυπηρέτες να επιστρέψουν απαντήσεις στα αιτήματα που τους έχουν γίνει.
3. *Αποστρατιωτικοποιημένη ζώνη*. Είναι οι υπολογιστές /συσκευές που ανήκουν στο εταιρικό δίκτυο και παρέχουν κάποια υπηρεσία προς τους εξωτερικούς χρήστες – π.χ. οι εξυπηρέτες WWW, DNS, ηλεκτρονικού ταχυδρομείου της εταιρείας, οι οποίοι προφανώς πρέπει να είναι προσπελάσιμοι από τους υπολογιστές του εξωτερικού δικτύου. Σημειώνεται ότι εξυπηρέτες που έμμεσα υποστηρίζουν τις υπηρεσίες αυτές, π.χ. ένας εξυπηρέτης βάσεων δεδομένων που έχει υλικό για διαμόρφωση σελίδων www, *δεν* τοποθετούνται στην αποστρατιωτικοποιημένη ζώνη, αλλά στο *εσωτερικό δίκτυο*. Το ίδιο ισχύει και στην περίπτωση που έχουμε αντιπροσώπευση υπηρεσιών, όπου ο πραγματικός

εξυπηρέτης τοποθετείται στο εσωτερικό δίκτυο και ο αντιπρόσωπός του στην αποστρατιωτικοποιημένη ζώνη.

Η δικτυακή διασύνδεση των τριών αυτών κατηγοριών υλοποιείται με τη χρήση *δύο δρομολογητών*, όπως φαίνεται στο σχήμα που ακολουθεί:



Επιτρέπεται σε μία τέτοια διάταξη να συγχωνεύονται οι εξωτερικός και ο εσωτερικός δρομολογητής, υπό την προϋπόθεση ότι ο δρομολογητής που θα χρησιμοποιηθεί (α) έχει διαφορετικές φυσικές συσκευές για κάθε υποδίκτυο και (β) μπορεί να εκτελεί φιλτράρισμα λαμβάνοντας υπόψη τη φυσική συσκευή εισόδου και εξόδου για κάθε πακέτο. Η εναλλακτική προσέγγιση φαίνεται στο ακόλουθο σχήμα:



6.2.4 Διακρίβωση δημόσιων κλειδιών

Ακόμη ένα θεμελιώδες πρόβλημα που ανακύπτει σε ανοικτά περιβάλλοντα με μεγάλο πλήθος εταίρων που δεν γνωρίζονται μεταξύ τους είναι η αυθεντικότητα των δημόσιων κλειδιών. Κατά την επαλήθευση μιας ψηφιακής υπογραφής, ο επαληθευτής πρέπει να είναι βέβαιος ότι το δημόσιο κλειδί που χρησιμοποιεί για την επαλήθευση της υπογραφής είναι πραγματικά το δημόσιο κλειδί του υποτιθέμενα υπογράφοντος. Με άλλα λόγια πρέπει να υπάρχει εμπιστοσύνη στην αντιστοιχία μεταξύ δημόσιου κλειδιού και οντότητας. Χωρίς πρόσθετα μέτρα, θα πρέπει κάθε χρήστης να διακριβώνει εξωσυστημικά την αυθεντικότητα κάθε δημόσιου κλειδιού πριν επιλέξει να το εμπιστευθεί. Η πολυπλοκότητα του ζητήματος μπορεί να μειωθεί εισάγοντας τη δυνατότητα διακρίβωσης για τα δημόσια κλειδιά μέσω μιας τρίτης οντότητας την οποία εμπιστεύονται και τα δύο μέρη. Η τρίτη οντότητα, που καλείται επίσης *αρχή πιστοποίησης*, υπογράφει με το δικό της ιδιωτικό κλειδί τα δημόσια κλειδιά και τα αντίστοιχα ονόματα, προσθέτοντας επίσης κάποια επιπλέον στοιχεία, π.χ. περίοδο εγκυρότητας. Το κομμάτι αυτό δεδομένων που έχει υπογραφεί από την αρχή πιστοποίησης καλείται *πιστοποιητικό*. Το πιστοποιητικό μπορεί να επαληθευτεί χρησιμοποιώντας το δημόσιο κλειδί της αρχής πιστοποίησης. Με τον τρόπο αυτό ο κάθε εταίρος μπορεί να διακριβώσει την ταυτότητα του άλλου, επαληθεύοντας πρώτα την ψηφιακή υπογραφή του εταίρου με το δημόσιο κλειδί του εταίρου και κατόπιν επαληθεύοντας την αυθεντικότητα του δημόσιου κλειδιού του εταίρου μέσα από την ψηφιακή υπογραφή του πιστοποιητικού, χρησιμοποιώντας το δημόσιο κλειδί της αρχής πιστοποίησης. Ο κάθε ένας έτσι αρκεί να έχει στην κατοχή του και να εμπιστεύεται μόνο το δημόσιο κλειδί της αρχής πιστοποίησης (και φυσικά το δικό του ιδιωτικό κλειδί), μειώνοντας έτσι δραστικά το πλήθος των κλειδιών που πρέπει κανείς να εμπιστεύεται. Στη γενική περίπτωση ένα πιστοποιητικό περιέχει τις ακόλουθες πληροφορίες:

- Έκδοση και αριθμό σειράς
- Το όνομα του εκδότη
- Το όνομα του υποκειμένου και άλλες τυχόν επεκτάσεις (διεύθυνση οικίας, εργασίας, αριθμό ταυτότητας κ.λπ.)
- Το σκοπό του πιστοποιητικού (ένδειξη αν το υποκείμενο δρα και ως αρχή πιστοποίησης)
- Το δημόσιο κλειδί του υποκειμένου
- Την περίοδο εγκυρότητας του πιστοποιητικού
- Την υπογραφή της αρχής διαχείρισης πιστοποιητικών

Σε πολυπληθείς κοινότητες οντοτήτων, μία και μόνη αρχή πιστοποίησης δεν είναι επαρκής, καθώς θα αποτελούσε σημείο συμφόρησης για όλο το δίκτυο και σημείο συνολικής αποτυχίας. Επίσης, βάσει των κανόνων της «ελεύθερης αγοράς» η κάθε εταιρεία ή άτομο θα πρέπει να μπορεί να επιλέγει ποιος θα είναι ο υπεύθυνος για τις δικές του λειτουργίες σε σχέση με τα δημόσια κλειδιά δηλ. (α) ποιος θα εκδώσει το δικό του πιστοποιητικό και (β) σε ποιον θα απευθύνει τις ερωτήσεις του σχετικά με το αν κάποιο πιστοποιητικό είναι έγκυρο. Σε ένα σχήμα με πολλαπλές αρχές πιστοποίησης, αν θεωρήσουμε την περίπτωση που το υποκείμενο Χ πιστοποιείται από και εμπιστεύεται για πιστοποίηση την αρχή Α και το υποκείμενο Υ πιστοποιείται από και εμπιστεύεται για πιστοποίηση την αρχή Β, τίθεται το θέμα του ποια αξία έχουν

για τον Υ τα πιστοποιητικά του Α (δηλ. πιστοποιητικά που έχουν εκδοθεί από αρχή *διαφορετική* από αυτή που έχει επιλέξει ο Υ).

Η πρώτη προσέγγιση πάνω στο ζήτημα αυτό είναι να πρέπει κάθε υποκείμενο να δηλώνει ρητώς ποιες αρχές πιστοποίησης εμπιστεύεται ως προς την έκδοση πιστοποιητικών. Η προσέγγιση αυτή είναι μη διαχειρίσιμη, λόγω του μεγάλου πλήθους των αρχών πιστοποίησης (θεωρητικά μπορούν να υπάρχουν μερικές δεκάδες έως εκατοντάδες σε κάθε χώρα) καθώς και της ανεπάρκειας γνώσεων των χρηστών (ο κάθε χρήστης δεν μπορεί να γνωρίζει όλες τις παραμέτρους αδειοδότησης και λειτουργίας όλων των αρχών).

Η δεύτερη προσέγγιση είναι αυτή της *ομότιμης διασταυρούμενης πιστοποίησης*. Βάσει της προσέγγισης αυτής οι αρχές πιστοποίησης εγκαθιστούν μεταξύ τους μονόδρομες ή αμφίδρομες σχέσεις εμπιστοσύνης σε ομότιμη βάση. Η μονόδρομη σχέση εμπιστοσύνης ορίζει ότι η αρχή πιστοποίησης Α πιστοποιεί τη Β ως έγκυρη αρχή πιστοποίησης, ενώ η αμφίδρομη καλύπτει και την αντίστροφη κατεύθυνση (και η Β πιστοποιεί την Α). Οι χρήστες και πάλι εμπιστεύονται τις επιμέρους αρχές πιστοποίησης, ωστόσο κατά τη διακρίβωση εγκυρότητας των πιστοποιητικών αξιοποιούνται οι σχέσεις εμπιστοσύνης. Για να θεωρηθεί από τον χρήστη Χ που εμπιστεύεται την αρχή Α έγκυρο ένα πιστοποιητικό που έχει εκδοθεί από την αρχή Β θα πρέπει να υπάρχει μία *αλυσίδα εμπιστοσύνης που να ξεκινάει από την Α και να καταλήγει στη Β*. Με άλλα λόγια θα πρέπει η αρχή Α να πιστοποιεί άμεσα τη Β ως «έγκυρη αρχή πιστοποίησης» ή να πιστοποιεί την αρχή Γ που με τη σειρά της πιστοποιεί τη Β κ.ο.κ. Κατά περίπτωση είναι δυνατόν να τίθενται και όρια στο μήκος της αλυσίδας πιστοποίησης, π.χ. να μην περιλαμβάνει πάνω από τρεις ενδιάμεσους κόμβους.

Μία τρίτη προσέγγιση στο ζήτημα είναι αυτή της *ιεραρχικής διασταυρούμενη πιστοποίησης*. Βάσει της προσέγγισης αυτής, οι αρχές πιστοποίησης οργανώνονται σε ιεραρχίες, με κάθε μία να πιστοποιεί τις υφιστάμενές της ως αρχές πιστοποίησης και ο κάθε χρήστης εμπιστεύεται την *πρωταρχική αρχή πιστοποίησης*, στη ρίζα της ιεραρχίας. Τα ίδια τα πιστοποιητικά εκδίδονται όχι (κατ' ανάγκην) από την πρωταρχική αρχή πιστοποίησης, αλλά από (κυρίως) από τις αρχές πιστοποίησης χαμηλότερα στην ιεραρχία. Οι χρήστες εμπιστεύονται ένα πιστοποιητικό διότι έχει εκδοθεί από μία αρχή για την οποία εγγυάται (άμεσα ή έμμεσα) η πρωταρχική αρχή πιστοποίησης (την οποία και εμπιστεύονται). Η διακρίβωση ενός πιστοποιητικού πρακτικά συνίσταται στην προσπάθεια να βρούμε αν υπάρχει μονοπάτι πιστοποίησης που να ξεκινάει από τη ρίζα και να φτάνει ως την αρχή που εξέδωσε το πιστοποιητικό.

Μία τελική προσέγγιση είναι το υβριδικό μοντέλο, όπου έχουμε στην ένα πλήθος *πρωταρχικών αρχών πιστοποίησης* που σχηματίζουν ανεξάρτητα μεταξύ τους δένδρα αρχών πιστοποίησης., παράλληλα όμως οι πρωταρχικές αρχές πιστοποίησης εγκαθιδρύουν μεταξύ τους σχέσεις εμπιστοσύνης.

Ανεξάρτητα από την προσέγγιση, είναι απαραίτητο ο κάθε χρήστης να εμπιστεύεται την αρχή πιστοποίησης που έχει επιλέξει και για να είναι αποτελεσματικό αυτό είναι απαραίτητο να έχει το δημόσιο κλειδί της. Η συνήθης προσέγγιση είναι να *εγκαθίσταται* αυτό το κλειδί στον υπολογιστή είτε με εξωσυστημική μεταφορά είτε από ασφαλές κανάλι επικοινωνίας, που συνήθως συμπληρώνεται και από τον οπτικό έλεγχο του χειριστή σε διάφορα στοιχεία του κλειδιού (άθροισμα ελέγχου, ονόματα, διευθύνσεις κ.λπ.). Παράλληλα, στον υπολογιστή εγκαθίσταται και το *ιδιωτικό κλειδί της οντότητας*, εφ' όσον αυτό υπάρχει. Το ιδιωτικό κλειδί πρέπει να είναι

προστατευμένο όχι μόνο από τροποποίηση αλλά και από διαρροή, καθ' όσον αν διαρρεύσει οποιοσδήποτε κάτοχός του μπορεί να αντιποιηθεί την οντότητα.

6.2.5 Ανάκληση κλειδιών

Μολονότι τα πιστοποιητικά που εκδίδονται από τις αρχές πιστοποίησης εμπεριέχουν περίοδο ισχύος, είναι δυνατόν να χρειασθεί κάποιο πιστοποιητικό να *ανακληθεί* προ της λήξεως της ισχύος του, για ένα πλήθος λόγων:

1. αν το ιδιωτικό κλειδί του χρήστη έχει διαρρεύσει, το δημόσιο πρέπει να ανακληθεί διότι μπορούν να κυκλοφορούν έγγραφα με πλαστή υπογραφή.
2. κάποιο από τα στοιχεία που περιέχονται στο πιστοποιητικό, έχει αλλάξει, καθιστώντας άκυρο το πιστοποιητικό συνολικά.
3. το πιστοποιητικό της αρχής πιστοποίησης θεωρείται ελαττωμένης ασφάλειας.
4. ο χρήστης παραβίασε τους κανόνες της αρχής πιστοποίησης.

Η αρχή πιστοποίησης ακυρώνει ένα πιστοποιητικό τοποθετώντας το σε μία λίστα *ανακληθέντων πιστοποιητικών*, η οποία δημοσιοποιείται για να είναι προσπελάσιμη στους ενδιαφερόμενους. Για να έχει αποτέλεσμα η λίστα ανάκλησης κλειδιών θα πρέπει το λογισμικό που χρησιμοποιείται να *μην* χρησιμοποιεί αποθηκευμένα πιστοποιητικά (cached certificates) χωρίς να επιβεβαιώνει ότι τα πιστοποιητικά αυτά δεν έχουν ήδη ανακληθεί.

6.2.6 Διαχείριση κλειδιών

Η διαχείριση κλειδιών είναι η διαδικασία παραγωγής κατάλληλων κλειδιών για ίδια χρήση και διάθεσής τους στους εταίρους με τρόπο ώστε να είναι δυνατόν να επιβεβαιωθούν και να χρησιμοποιηθούν με τον προτιθέμενο τρόπο, ενώ παράλληλα κανείς δεν μπορεί να τα καταχρασθεί. Αυτό σημαίνει ότι τα συμμετρικά κλειδιά, καθώς και τα μυστικά κλειδιά της ασύμμετρης κρυπτογραφίας πρέπει να μεταδίδονται εμπιστευτικά (με κρυπτογράφηση ή εξωσυστημική μετάδοση). Για τα δημόσια κλειδιά της ασύμμετρης κρυπτογραφίας πρέπει να είναι δυνατόν να επαληθευτεί η αυθεντικότητα του κλειδιού.

Οι ίδιες απαιτήσεις ισχύουν για την επικοινωνία μεταξύ μιας αρχής πιστοποίησης και ενός χρήστη για τη διακρίβωση ενός δημόσιου κλειδιού. Είναι απαραίτητη η ανταλλαγή κλειδιών μεταξύ του χρήστη και της αρχής πιστοποίησης. Υπάρχουν δύο κύριες εναλλακτικές προσεγγίσεις για τη λειτουργία της διακρίβωσης δημόσιων κλειδιών, οι οποίες έχουν διαφορετικά χαρακτηριστικά ασφάλειας:

1. Η αρχή πιστοποίησης δημιουργεί το ζεύγος κλειδιών ασύμμετρης κρυπτογραφίας, πιστοποιεί το δημόσιο κλειδί και δίνει στον χρήστη το μυστικό κλειδί, το πιστοποιητικό (που περιλαμβάνει το δημόσιο κλειδί) και ενδεχομένως άλλα πιστοποιητικά που συνδέουν το σύνολο των κλειδιών με το κλειδί ρίζας.
2. Ο χρήστης δημιουργεί μόνος του το ζεύγος των κλειδιών και αποστέλλει το δημόσιο κλειδί στην αρχή πιστοποίησης, η οποία το πιστοποιεί και δίνει στον χρήστη το πιστοποιητικό (που περιλαμβάνει το δημόσιο κλειδί) και ενδεχομένως άλλα πιστοποιητικά που συνδέουν το σύνολο των κλειδιών με το κλειδί ρίζας.

Στην πρώτη περίπτωση υφίσταται η ανάγκη για εμπιστευτική επικοινωνία, στη δεύτερη όχι. Η εμπιστευτική επικοινωνία από την αρχή πιστοποίησης προς τον

χρήστη είναι εύκολη, με τη δημιουργία μιας έξυπνης κάρτας που περιέχει τα απαραίτητα στοιχεία και που παραδίδεται αυτοπροσώπως στον χρήστη. Αν χρησιμοποιείται ηλεκτρονική μετάδοση, πρέπει να ανταλλαχθεί μεταξύ χρήστη και αρχής πιστοποίησης ένα συμμετρικό κλειδί κρυπτογράφησης με εξωσυστημικό τρόπο. Στην πρώτη περίπτωση η αρχή πιστοποίησης έχει πλήρη έλεγχο για την ποιότητα των παραγόμενων κλειδιών, στη δεύτερη ο κάθε χρήστης είναι υπεύθυνος για το δικό του ζεύγος κλειδιών. Σε κάθε περίπτωση, είναι απαραίτητη μια εξωσυστημική ανταλλαγή πληροφορίας, διαμέσου της οποίας η αρχή πιστοποίησης θα βεβαιωθεί ότι η πιστοποιούμενη οντότητα είναι πραγματικά αυτή που ισχυρίζεται ότι είναι.

6.2.7 Δημόσιοι κατάλογοι

Η χρήση της κρυπτογραφίας δημόσιου κλειδιού καθιστά αναγκαία τη γνώση των δημόσιων κλειδιών των άλλων και την ύπαρξη εμπιστοσύνης προς αυτά. Οι δημόσιοι κατάλογοι, όπως οι κατάλογοι τύπου X500, μπορούν να υποστηρίξουν αυτόν τον στόχο. Οι χρήστες και οι αρχές πιστοποίησης, όταν ενέχονται, έχουν τις ακόλουθες απαιτήσεις όταν χρησιμοποιούν υπηρεσίες ασφάλειας με βάση την κρυπτογραφία δημόσιου κλειδιού:

- Οι χρήστες θέλουν να δημοσιοποιούν τα δικά τους δημόσια κλειδιά, να μπορούν να προσπελάσουν τα πιστοποιητικά των άλλων και τις λίστες ανάκλησης πιστοποιητικών, προκειμένου να διακριβώνουν ότι κάποιο πιστοποιητικό είναι έγκυρο και δεν έχει ανακληθεί.
- Οι αρχές πιστοποίησης επιθυμούν να δημοσιοποιούν τα δημόσια κλειδιά και τις λίστες ανάκλησης με τρόπο που να τα καθιστά διαθέσιμα στην ενδιαφερόμενη κοινότητα.
- Οι αρχές πιστοποίησης επιθυμούν να ανταλλάσσουν κλειδιά αμοιβαίας πιστοποίησης με άλλες αρχές πιστοποίησης και να δημοσιοποιούν τα δικά τους δημόσια κλειδιά.

Οι δημόσιοι κατάλογοι βοηθούν σ' αυτή την κατεύθυνση ως παροχείς πληροφορίας στη βασική υποδομή ασφάλειας, αποθηκεύοντας δηλαδή τη σχετική πληροφορία και επιτρέποντας την ευέλικτη αναζήτησή της. Παράλληλα δε, οι κατάλογοι είναι και *χρήστες μηχανισμών ασφάλειας,* καθώς ενσωματώνουν τεχνικές για την προστασία της πληροφορίας του κατάλογου, των επικοινωνιών με άλλες οντότητες καθώς και των πόρων των υπολογιστικών συστημάτων που τους φιλοξενούν. Για τους λόγους αυτούς, οι προδιαγραφές του δημόσιου κατάλογου τύπου X500 έχουν πλαισιωθεί από το *Πλαίσιο Αυθεντικότητας X509*, το οποίο συμπεριλαμβάνει ισχυρούς μηχανισμούς διακρίβωσης ταυτότητας.

6.2.8 Ψηφιακές υπογραφές

Οι ψηφιακές υπογραφές είναι μία τεχνική που επιτρέπει την εξακρίβωση ότι ένα συγκεκριμένο κείμενο προέρχεται από έναν συγκεκριμένο αποστολέα, υποστηρίζοντας έτσι τη διάσταση της *αυθεντικότητας*. Προκειμένου να εξακριβωθεί η αυθεντικότητα ενός εγγράφου λαμβάνουν χώρα τα κάτωθι βήματα:

1. Αρχικά στο μήνυμα εφαρμόζεται μία *συνάρτηση κερματισμού*, η οποία υπολογίζει ένα πλήθος από bits βάσει του περιεχομένου του μηνύματος. Τα bits αυτά ονομάζονται digest του μηνύματος.

2. Το digest του μηνύματος κρυπτογραφείται με το ιδιωτικό κλειδί του αποστολέα παράγοντας την *ψηφιακή υπογραφή του μηνύματος*.
3. Η ψηφιακή υπογραφή επισυνάπτεται στο μήνυμα και αποστέλλεται μαζί με αυτό.
4. Ο παραλήπτης του μηνύματος αρχικά διαχωρίζει την ψηφιακή υπογραφή από το καθ' εαυτό μήνυμα και εφαρμόζει στο καθ' εαυτό μήνυμα την ίδια συνάρτηση κερματισμού, παράγοντας και αυτός ένα digest.
5. Ακολούθως αποκρυπτογραφεί την ψηφιακή υπογραφή με το δημόσιο κλειδί του υπογράψαντος παράγοντας ένα digest'.
6. Τέλος, τα digest και digest' των βημάτων 5 και 6 συγκρίνονται. Μόνο αν αυτά είναι ίσα το μήνυμα θεωρείται αυθεντικό.

Σημειώνεται ότι καθώς η ανωτέρω διαδικασία περιλαμβάνει τόσο τα κλειδιά (δημόσιο και ιδιωτικό) του αποστολέα όσο και το ίδιο το περιεχόμενο του μηνύματος (βάσει του οποίου υπολογίζεται το digest), η τεχνική αυτή διακριβώνει τόσο το γεγονός ότι το μήνυμα εστάλη από τον συγκεκριμένο αποστολέα όσο και ότι το περιεχόμενό του παραμένει αναλλοίωτο από τη στιγμή που υπεγράφη ψηφιακά.