Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης
και Υπολογιστων

Θέματα Ασφάλειας στις Γλώσσες Προγραμματισμού Βιομηχανική και Ακαδημαϊκή Προσέγγιση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΗΤΡΗΣ ΒΑΡΔΟΥΛΑΚΗΣ

Επιβλέπων : Νικόλαος Παπασπύρου
Λέκτορας Ε.Μ.Π.

Αθήνα, Ιούλιος 2005

Εθνικο Μετσοβιο Πολυτεχνειο
Σχολη Ηλεκτρολογων Μηχανικων
και Μηχανικων Υπολογιστων
Τομεας Τεχνολογιας Πληροφορικης
και Υπολογιστων

Θέματα Ασφάλειας στις Γλώσσες Προγραμματισμού Βιομηχανική και Ακαδημαϊκή Προσέγγιση

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΔΗΜΗΤΡΗΣ ΒΑΡΔΟΥΛΑΚΗΣ

Επιβλέπων : Νικόλαος Παπασπύρου
Λέκτορας Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 11η Ιουλίου 2005.

Νικόλαος Παπασπύρου Λέκτορας Ε.Μ.Π.	Ευστάθιος Ζάχος Καθηγητής Ε.Μ.Π.	Ανδρέας-Γεώργιος Σταφυλοπάτης Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούλιος 2005

Δημήτρης Βαρδουλάκης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Ο σκοπός της παρούσας εργασίας είναι η συγκριτική μελέτη της βιομηχανικής και της ακαδημαϊκής προσέγγισης στο θέμα της ασφάλειας εκτελέσιμου κώδικα, όπως αυτό αντιμετωπίζεται από τις σύγχρονες γλώσσες προγραμματισμού, και η υλοποίηση κάποιων αντιπροσωπευτικών εφαρμογών χρησιμοποιώντας και τις δύο προσεγγίσεις.

Τα τελευταία χρόνια, εξαιτίας της εξάπλωσης του διαδικτύου, τα ογκώδη, συγκεντρωμένα σε μια τοποθεσία υπολογιστικά συστήματα έχουν αντικατασταθεί από τα πιο ευέλικτα κατανεμημένα συστήματα. Είναι πλέον πολύ σύνηθες για τους χρήστες να εκτελούν άμεσα ή και έμμεσα κώδικα που έχουν κατεβάσει από το διαδίκτυο. Σε ένα τέτοιο κατανεμημένο περιβάλλον η ασφάλεια και η αξιοπιστία του εκτελέσιμου κώδικα αποτελεί μείζον θέμα, αφού πολλές ετερόκλητες εφαρμογές (π.χ. τηλεπικοινωνίες, ηλεκτρονικό εμπόριο, μάθηση εξ αποστάσεως) εξαρτώνται από την ποιότητα κώδικα που έχει κατέβει από το διαδίκτυο.

Παρουσιάζουμε πρώτα το σύστημα ασφαλείας της πλατφόρμας Java και του πλαισίου .NET, που είναι οι πιο δημοφιλείς πλατφόρμες στους προγραμματιστές διαδικτυακών εφαρμογών. Στη συνέχεια, παρουσιάζουμε κάποιες προτάσεις της ακαδημαϊκής κοινότητας για την αντιμετώπιση του θέματος της ασφάλειας των γλωσσών προγραμματισμού. Για την επίδειξη όλων αυτών των προσεγγίσεων, υλοποιήσαμε τρεις εφαρμογές με αυξημένες απαιτήσεις ασφάλειας:

- το κρυπτογραφικό πρωτόκολλο δημοσίου κλειδιού Needham-Schroeder,
- μια εφαρμογή ηλεκτρονικών τραπεζικών συναλλαγών, και
- μια εφαρμογή με αμοιβαίο αποκλεισμό διεργασιών που δρουν σε κοινά δεδομένα.

Οι εφαρμογές αυτές υλοποιήθηκαν στις γλώσσες Java, C# και Apollo. Αφού περιγράψουμε τα πλεονεκτήματα και τα μειονεκτήματα κάθε υλοποίησης, συνοψίζουμε επιχειρώντας μια γενική σύγκριση μεταξύ της βιομηχανικής και της ακαδημαϊκής προσέγγισης στο θέμα της ασφάλειας.

Λέξεις κλειδιά

Ασφάλεια εκτελέσιμου κώδικα, γλώσσες προγραμματισμού, νοητή μηχανή, αντικειμενοστρεφής προγραμματισμός, συστήματα τύπων.

Abstract

The purpose of this diploma dissertation is the comparative study of the software-industry approach and the academic approach to the subject of security of executable code, in the way this is addressed by modern programming languages, and the implementation of some representative applications using both approaches.

In our days, due to the spreading of internet, the monolithic computer systems of the past that were concentrated in one location are replaced by more flexible distributed systems. It is common practise for the users of these systems to execute (directly or indirectly) code downloaded from the internet. In such a distributed environment, security and integrity of executable code is a matter of major significance, as various applications (e.g. telecommunications, electronic commerce, distance learning) depend on the quality of code that is downloaded from the internet.

In this dissertation, we first present the security system of the Java platform and the .NET framework, the two most popular platforms among programmers of internet applications. Subsequently, we present some solutions to the problem of language-based security proposed by the academic community. To demonstrate all these approaches, we have implemented three applications of high security requirements:

- the Needham-Schroeder public key protocol,
- a distributed banking application, and
- an application requiring mutual exclusion of processes acting on the same data.

These applications were implemented using three different languages: Java, C# and Apollo. We describe the advantages and disadvantages of each implementation and conclude with an overall comparison between the software-indusrty approach and the academic approach to the subject of language-based security.

Key words

Security of executable code, programming languages, virtual machine, object-oriented programming, type systems.

Ευχαριστίες

Κατ' αρχήν θέλω να ευχαριστήσω τον επιβλέποντα καθηγητή της διπλωματικής μου, κ. Νίκο Παπασπύρου για το χρόνο που αφιέρωσε σε αυτή την εργασία και γιατί μου έδειξε σε ποια κατεύθυνση κινείται η έρευνα στις γλώσσες προγραμματισμού σήμερα. Επίσης, θέλω να ευχαριστήσω το Stephen Tse για τη βοήθειά του όσον αφορά την Apollo. Τέλος, θέλω να ευχαριστήσω τους φίλους-συμφοιτητές Μιχάλη Δημάκο και Ανδρέα Μενύχτα για τη συνεργασία που είχαμε όλα αυτά τα χρόνια.

Δημήτρης Βαρδουλάκης,

Αθήνα, 11η Ιουλίου 2005.

Η εργασία αυτή είναι επίσης διαθέσιμη ως Τεχνική Αναφορά CSD-SW-TR-2-05, Εθνικό Μετσόβιο Πολυτεχνείο, Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Τομέας Τεχνολογίας Πληροφορικής και Υπολογιστών, Εργαστήριο Τεχνολογίας Λογισμικού, Ιούλιος 2005.

URL: `http://www.softlab.ntua.gr/techrep/`
FTP: `ftp://ftp.softlab.ntua.gr/pub/techrep/`

Ο κώδικας των εφαρμογών που υλοποιήθηκαν για τις ανάγκες της παρούσας εργασίας είναι διαθέσιμος από τις ίδιες ηλεκτρονικές διευθύνσεις.

Περιεχόμενα

Πίνακες

Σχήματα

Κεφάλαιο 1

Εισαγωγή

1.1 Σκοπός

Σκοπός της παρούσας διπλωματικής εργασίας είναι η συγκριτική μελέτη της βιομηχανικής και της ακαδημαϊκής προσέγγισης στο θέμα της ασφάλειας εκτελέσιμου κώδικα, όπως αυτό αντιμετωπίζεται από τις σύγχρονες γλώσσες προγραμματισμού. Για το σκοπό αυτό υλοποιήθηκαν τρεις αντιπροσωπευτικές εφαρμογές με αυξημένες απαιτήσεις ασφάλειας, χρησιμοποιώντας και τις δύο προσεγγίσεις.

1.2 Ασφάλεια εκτελέσιμου κώδικα

Στην εποχή της κοινωνίας της πληροφορίας, το θέμα της αξιοπιστίας και της ασφάλειας έχει μεγάλη βαρύτητα. Οι όροι αυτοί αφορούν τεχνολογίες, υποδομές, υπηρεσίες και εφαρμογές οι οποίες απευθύνονται στις εξελισσόμενες κοινωνικές και οικονομικές ανάγκες. Από την οπτική γωνία ενός μηχανικού, ο εκτελέσιμος κώδικας είναι ένα σημαντικό τμήμα όλων αυτών των τεχνολογιών, υποδομών, υπηρεσιών και εφαρμογών, γι' αυτό η αξιοπιστία και η ασφάλειά του είναι μείζονος σημασίας.

Η αιχμή της τεχνολογίας στην ανάπτυξη λογισμικού μπορεί σύντομα να περιγραφεί ως εξής: Τα συστήματα λογισμικού είναι οργανωμένα σε ομάδες από μικρές συνεργαζόμενες δομικές μονάδες με δημόσια διαθέσιμες διαπροσωπείες. Αυτές τυπικά ορίζουν το πρωτόκολλο διαλειτουργίας, του οποίου όμως η σημασιολογία συνήθως περιγράφεται με έναν μη-τυπικό τρόπο. Τα δομικά στοιχεία αναπτύσσονται ανεξάρτητα το ένα από το άλλο και πιθανώς σε διαφορετικές γλώσσες προγραμματισμού. Η ανεξαρτησία πλατφόρμας και η διαλειτουργησιμότητα είναι δύο βασικές μη-λειτουργικές απαιτήσεις από τέτοια στοιχεία. Επιπλέον, καθώς η τεχνολογία στους υπολογιστές και στα δίκτυα ολοκληρώνεται στο καθημερινό περιβάλλον μας με τη μορφή μικρών "έξυπνων" συσκευών, αυτές οι απαιτήσεις γίνονται όλο και πιο σημαντικές και νέες απαιτήσεις έρχονται στο προσκήνιο.

Όπως κάθε μορφή δεδομένων μπορεί να μεταφερθεί μέσω δικτύου από μια υπολογιστική συσκευή σε μια άλλη, έτσι μπορεί και ο εκτελέσιμος κώδικας. Υποθέτοντας ότι ο μεταφερόμενος κώδικας είναι κατάλληλος προς εκτέλεση στη συσκευή-παραλήπτη, αυτή η διαδικασία μπορεί να αποδειχτεί πολύτιμη καθώς αποτελεί τη βάση ενός κατανεμημένου συστήματος λογισμικού. Η απαίτηση για ανεξαρτησία πλατφόρμας στα πλαίσια αυτά επιβάλλει την τυποποίηση μιας ενδιάμεσης μορφής κώδικα, η οποία θα μεταφράζεται σε κώδικα μηχανής ώστε να εκτελεστεί στην συσκευή-παραλήπτη. Η απαίτηση για διαλειτουργησιμότητα γίνεται ολοένα και πιο σημαντική καθώς ο μεταφερόμενος κώδικας αναμένεται να μπορεί να ολοκληρωθεί με κώδικα που υπάρχει ήδη στη συσκευή-παραλήπτη. Επιπλέον είναι φυσικό να απαιτείται η δυνατότητα για δυναμική αντικατάσταση ή ενημέρωση των δομικών στοιχείων ενός συστήματος λογισμικού χωρίς να διακόπτονται αισθητά οι δραστηριότητες του συστήματος.

Σε ένα τέτοιο κατανεμημένο σύστημα λογισμικού, δεν είναι δυνατόν να υποθέσει κανείς ότι ο εκτελέσιμος κώδικας είναι αξιόπιστος. Ο μεταφερόμενος κώδικας μπορεί να δημιουργήσει προβλήματα, είτε σκόπιμα (π.χ. ιοί, επιθέσεις), είτε χωρίς πρόθεση, αν περιέχει σφάλματα

που θέτουν σε κίνδυνο τη σταθερότητα και την ασφάλεια της συσκευής-παραλήπτη. Καθώς ένα πλήθος εφαρμογών από διαφορετικές περιοχές (π.χ. τηλεπικοινωνίες, ηλεκτρονικό εμπόριο, τηλεϊατρική, μάθηση εξ αποστάσεως) σε μεγάλο βαθμό εξαρτάται από την ποιότητα του κώδικα, είναι προφανές ότι, για να εγγυηθεί κανείς την ασφάλεια στην κοινωνία της γνώσης και της πληροφορίας, πρεπεί να εξασφαλιστεί η χωρίς προβλήματα συμπεριφορά του εκτελέσιμου κώδικα.

Τα τελευταία χρόνια έχουν γίνει σημαντικά βήματα προόδου όσον αφορά την ασφάλεια στις εμπορικές γλώσσες προγραμματισμού. Οι γλώσσες Java και C#, που είναι οι πλέον διαδεδομένες αντικειμενοστρεφείς γλώσσες, έχουν ασφαλή συστήματα τύπων (σε αντίθεση με προηγούμενες γλώσσες όπως η C) και επιπλέον, χρησιμοποιώντας τους συλλέκτες σκουπιδιών, απαλλάσουν τον προγραμματιστή από τη διαδικασία διαχείρισης της μνήμης (η οποία αποτελεί μία από τις πιο συνηθισμένες πηγές λαθών που θέτουν σε κίνδυνο την ασφάλεια ενός συστήματος). Οι νοητές μηχανές που αναλαμβάνουν την εκτέλεση των προγραμμάτων σε αυτές τις γλώσσες παρέχουν ένα επιπλέον επίπεδο ασφάλειας πραγματοποιώντας ελέγχους κατά το χρόνο εκτέλεσης. Επίσης, σημαντικές καινοτομίες έχουν προταθεί και από την ακαδημαϊκή κοινότητα για την ασφάλεια των γλωσσών προγραμματισμού. Έχουν αναπτυχθεί διάφορα θεωρητικά μοντέλα για τη συνολική αντιμετώπιση του ζητήματος και παράλληλα, με τη συνεχή ανάπτυξη της θεωρίας τύπων, έχουν αναπτυχθεί ισχυρά συστήματα τύπων που μπορούν να ενσωματώσουν περιορισμούς ασφάλειας.

1.3 Σύνοψη της εργασίας

Η παρούσα διπλωματική εργασία έχει την ακόλουθη δομή:

Κεφάλαιο 2. Στο κεφάλαιο αυτό βλέπουμε πώς αντιμετωπίζουν το θέμα της ασφάλειας η Java και η C#, οι δύο ευρέως χρησιμοποιούμενες γλώσσες στο διαδίκτυο.

Κεφάλαιο 3 Στο κεφάλαιο αυτό παρουσιάζουμε διάφορες λύσεις που έχουν προταθεί από την ακαδημαϊκή κοινότητα.

Κεφάλαια 4, 5 και 6. Στα κεφάλαια αυτά παρουσιάζουμε την υλοποίηση τριών εφαρμογών που έχουν διαφορετικές απαιτήσεις για την ασφάλεια η καθεμία.

Κεφάλαιο 7. Στο κεφάλαιο αυτό επιχειρούμε μια σύγκριση μεταξύ της βιομηχανικής και της ακαδημαϊκής προσέγγισης στο θέμα της ασφάλειας εκτελέσιμου κώδικα, όπως αντιμετωπίζεται από τις γλώσσες προγραμματισμού.

Παράρτημα. Περιέχει τον πλήρη κώδικα των εφαρμογών που υλοποιήθηκαν.

Για την ανάπτυξη των εφαρμογών σε Java χρησιμοποιήθηκε το περιβάλλον Eclipse και για την ανάπτυξη των εφαρμογών σε C# χρησιμοποιήθηκε το περιβάλλον #Develop.

Κεφάλαιο 2

Ασφάλεια στις εμπορικές γλώσσες προγραμματισμού

Η ραγδαία εξάπλωση του διαδικτύου τα τελευταία χρόνια έχει επιφέρει τεράστιες αλλαγές στους τομείς της αρχιτεκτονικής συστημάτων και της λογισμικής μηχανικής (software engineering). Τα ογκώδη, συγκεντρωμένα σε μια τοποθεσία υπολογιστικά συστήματα έχουν αντικατασταθεί από τα πιο ευέλικτα κατανεμημένα συστήματα (distributed systems), με τους υπολογιστές να συνδέονται μεταξύ τους στο τοπικό δίκτυο ή μέσω διαδικτύου (internet).

Εξαιτίας των παραπάνω εξελίξεων, είναι πλέον πολύ σύνηθες για τους χρήστες να εκτελούν άμεσα ή και έμμεσα κώδικα που έχουν κατεβάσει από το διαδίκτυο. Για να γίνεται αυτό αποτελεσματικά, ο κώδικας οφείλει να 'ναι ανεξάρτητος της πλατφόρμας υλοποίησης, ώστε να είναι συμβατός με όλες τις (διαφορετικές μεταξύ τους) πλατφόρμες που κυκλοφορούν (Windows, Linux, Solaris, Macintosh κ.λπ.). Πρέπει να διανέμεται γραμμένος όχι σε γλώσσα μηχανής αλλά σε μια ενδιάμεση γλώσσα κοινή για όλους τους επεξεργαστές. Επιπλέον, στη μορφή της ενδιάμεσης γλώσσας είναι ευκολότερος ο έλεγχος και η πιστοποίηση του κώδικα προτού αρχίσει να εκτελείται.

Έχοντας τα προαναφερθέντα χαρακτηριστικά υπόψιν, η Sun Microsystems παρουσίασε το 1995 τη Java, η οποία έφερε επανάσταση στο χώρο των γλωσσών προγραμματισμού και γρήγορα εξαπλώθηκε σε όλο το διαδίκτυο. Τα προγράμματα Java μεταφράζονται σε μια ενδιάμεση γλώσσα που ονομάζεται *Java-bytecode* και εκτελούνται σε οποιαδήποτε πλατφόρμα που έχει εγκατεστημένο ένα υποθετικό επεξεργαστή που ονομάζεται *Νοητή Μηχανή Java* (Java Virtual Machine, JVM). Η "γλώσσα μηχανής" αυτού του "επεξεργαστή" είναι το Java-bytecode.

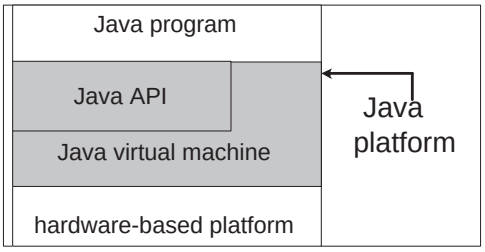
Ακολουθώντας το δρόμο της Sun, η Microsoft παρουσίασε κάποια χρόνια αργότερα το πλαίσιο .NET, το οποίο ακολουθεί την ίδια λογική με τη Java. Η βασική διαφορά είναι ότι το περιβάλλον εκτέλεσης του .NET, που ονομάζεται *Κοινό Περιβάλλον Εκτέλεσης για Όλες τις Γλώσσες* (Common Language Runtime, CLR), σε αντίθεση με τη Νοητή Μηχανή Java, υποστηρίζει πολλές γλώσσες προγραμματισμού και όχι μόνο μία (C#, C++, Visual Basic). Οι γλώσσες αυτές μεταφράζονται σε μια κοινή ενδιάμεση γλώσσα την οποία εκτελεί το CLR. Η σημαντικότερη απ' τις γλώσσες αυτές είναι ασφαλώς η C#, μιας και σχεδιάστηκε ειδικά για το πλαίσιο .NET ενώ οι υπόλοιπες προϋπήρχαν.

Η Java και η C# έχουν κάνει σημαντικά βήματα στον τομέα της ασφάλειας του εκτελέσιμου κώδικα. Οι αρχιτέκτονες του .NET και της Java είχαν στο μυαλό τους την ασφάλεια ως ένα από τα βασικά χαρακτηριστικά των συστημάτων τους και όχι σαν αμελητέο παράγοντα που θα μπορούσε να προστεθεί αργότερα. Παρ' όλα αυτά όμως, είναι χαρακτηριστικό των εμπορικών συστημάτων ότι χρησιμοποιούν ειδικές λύσεις ανάλογα με το εκάστοτε πρόβλημα και δεν υπάρχει μια πιο γενική-θεωρητική αντιμετώπιση στο θέμα της ασφάλειας. Παρακάτω θα δούμε τα βασικά χαρακτηριστικά ασφαλείας στη Java και στη C# (και στο .NET γενικότερα).

2.1 Το μοντέλο ασφάλειας στην πλατφόρμα Java

2.1.1 Σύντομη περιγραφή της Java

Όπως είδαμε (σελ. 19), ο πηγαίος κώδικας της Java μεταφράζεται σε bytecode που εκτελείται από τη JVM. Η JVM μαζί με τη Διαπροσωπεία Προγραμματιστικών Εφαρμογών (Application Programming Interface, API) συνθέτουν την πλατφόρμα Java (σχήμα 2.1).



Σχήμα 2.1: Δόμηση της πλατφόρμας Java.

Ένας πολύ συνηθισμένος τύπος εφαρμογών στη Java είναι τα applets. *Applet* ονομάζεται ένα σχετικά μικρό σε μέγεθος πρόγραμμα που χρησιμοποιείται στην κατασκευή δυναμικών ιστοσελίδων. Το applet στέλνεται από τον εξυπηρετητή στον περιηγητή (browser) του χρήστη, ο οποίος το εκτελεί. Λόγω του ότι τα applets κατεβαίνουν από το διαδίκτυο, είναι πιο επικίνδυνα για την ασφάλεια ενός υπολογιστικού συστήματος απ' ότι ο κώδικας που είναι αποθηκευμένος τοπικά.

Η Java ακολουθεί το *αντικειμενοστρεφές μοντέλο προγραμματισμού* (object oriented programming). Ο κώδικας δομείται σε μέρη που ονομάζονται *αντικείμενα*. Κάθε αντικείμενο είναι μια ξεχωριστή οντότητα στο πρόγραμμα, έχει τις δικές του μεταβλητές (πεδία) και συναρτήσεις (μεθόδους). Τα αντικείμενα επικοινωνούν μεταξύ τους ανταλλάσοντας μηνύματα, δηλάδη καλώντας το ένα τις μεθόδους του άλλου.

Τα ομοειδή αντικείμενα οργανώνονται σε κλάσεις. Η *κλάση* είναι ένα "καλούπι" απ' το οποίο κατασκευάζονται αντικείμενα, που γι' αυτό το λόγο λέγονται και στιγμιότυπα (instances) της κλάσης. Με άλλα λόγια, δύο αντικείμενα που κατασκευάστηκαν από την ίδια κλάση έχουν το ίδιο πλήθος και τύπο πεδίων και μεθόδων, αλλά μπορούν να διαφέρουν οι τιμές των πεδίων τους. Οι κλάσεις ομαδοποιούνται σε πακέτα (packages) που το καθένα έχει το δικό του χώρο ονομάτων (namespace). Για παράδειγμα, η κλάση SimpleClass του πακέτου Package1 έχει το πλήρες όνομα Package1.SimpleClass, άρα διαφέρει από την κλάση Package2.SimpleClass του πακέτου Package2. Έτσι αποφεύγονται οι συγκρούσεις ονομάτων.

Το αντικειμενοστρεφές μοντέλο επιτρέπει καλή οργάνωση και λειτουργικότητα στον κώδικα, γιατί η δομή του μοιάζει με τη δομή των πραγμάτων στο φυσικό κόσμο. Τα βασικά χαρακτηριστικά στον αντικειμενοστρεφή προγραμματισμό είναι:

Κληρονομικότητα. Κάθε κλάση μπορεί να κληρονομεί τα πεδία και τις μεθόδους μιας άλλης κλάσης και να προσθέτει δικά της. Για παράδειγμα, μπορεί να υπάρχει μια κλάση "εργαζόμενος" απ' την οποία να κληρονομούν οι κλάσεις "καθηγητής", "οδηγός ταξί", "αγρότης". Έτσι δημιουργείται μια ιεραρχία κλάσεων.

Πολυμορφισμός. Οι κλάσεις είναι και αυτές τύποι της γλώσσας. Μέσω της ιεραρχίας κλάσεων μπορούμε να έχουμε πολυμορφισμό στα αντικείμενα, δηλαδή αν μια μέθοδος παίρνει σαν όρισμα ένα αντικείμενο της κλάσης "εργαζόμενος", τότε θα δέχεται και αντικείμενα από τις κλάσεις "καθηγητής", "οδηγός ταξί", "αγρότης" καθώς και από κάθε άλλη υποκλάση της κλάσης "εργαζόμενος".

Ενθυλάκωση. Ο κώδικας μπορεί να δομηθεί έτσι ώστε ένα αντικείμενο από μια κλάση να μη βλέπει κανένα από τα πεδία ενός αντικειμένου από άλλη κλάση. Η επικοινωνία τους

γίνεται μόνο μέσω μηνυμάτων, δηλαδή με κλήσεις των μεθόδων τους. Με αυτό τον τρόπο μπορούμε να αποκρύπτουμε τις λεπτομέρειες υλοποίησης μιας κλάσης, από τον "έξω κόσμο", δηλαδή τις υπόλοιπες κλάσεις.

Είναι φανερό ότι τα παραπάνω χαρακτηριστικά καθώς και το ισχυρό σύστημα τύπων της Java συμβάλλουν αποφασιστικά στην ασφάλεια του εκτελέσιμου κώδικα. Επιπλέον, Η Java παρέχει το μηχανισμό των εξαιρέσεων για ενέργειες που θεωρούνται μη-αποδεκτές από τη JVM. Όταν πάει να εκτελεστεί μια τέτοια ενέργεια, η JVM εγείρει μια εξαίρεση (exception) και αν δεν έχει γραφτεί από τον προγραμματιστή κατάλληλος κώδικας για το χειρισμό της τότε το πρόγραμμα τερματίζει. Για μια πλήρη περιγραφή της γλώσσας δείτε τα [1, 2].

2.1.2 Το Αμμοδοχείο

Το μοντέλο ασφάλειας της Java μπορεί να περιγραφεί χρησιμοποιώντας μεταφορικά τη λέξη *αμμοδοχείο* (Sandbox). Η Java επιτρέπει στους χρήστες να κατεβάσουν και να εκτελέσουν "ύποπτο" κώδικα (που δεν ξέρουν ή δεν εμπιστεύονται την πηγή προέλευσής του) χωρίς κίνδυνο, απομονώνοντας τον κώδικα στο ξεχωριστό του αμμοδοχείο. Ένα applet μπορεί να προκαλέσει καταστροφές στο αμμοδοχείο του, αλλά δεν μπορεί να παρέμβει και να επηρεάσει άλλα αμμοδοχεία. Επιπλέον, μπορούν να τεθούν περιορισμοί για το τί μπορεί να κάνει μια εφαρμογή μέσα στο αμμοδοχείο της, πετυχαίνοντας έτσι την εκτέλεση ύποπτου κώδικα σε έμπιστο και ασφαλές περιβάλλον.

Το αμμοδοχείο συνθέτουν συνεργαζόμενα τμήματα του συστήματος, από διαχειριστές ασφάλειας που εκτελούνται παράλληλα με την εφαρμογή μέχρι περιορισμούς ασφαλείας ενσωματωμένους στην πλατφόρμα Java και στην ίδια τη γλώσσα. Παρακάτω περιγράφεται πώς το μοντέλο ασφάλειας της Java κατασκευάζει το αμμοδοχείο.

Φορτωτής Κλάσεων (The Class Loader): Για να φορτωθεί ένα applet, ο περιηγητής καλεί το φορτωτή κλάσεων. Αυτό αποτελεί την πρώτη γραμμή άμυνας στο μοντέλο ασφάλειας της Java, γιατί ο φορτωτής κλάσεων αποφασίζει αν και πότε μπορούν τα applets να φορτώσουν κλάσεις (κώδικα). Οι βασικές λειτουργίες του είναι:

- Φέρνει τον κώδικα του applet από κάποιο άλλο υπολογιστή.
- Δημιουργεί ένα καινούριο χώρο ονομάτων για κάθε applet. Έτσι, τα applets έχουν πρόσβαση μόνο στις κλάσεις τους και τις κλάσεις του Java API, όχι στις κλάσεις του τοπικού χρήστη ή στις κλάσεις άλλων applets. Επίσης, ο φορτωτής εξασφαλίζει ότι τα applets δε θα αντικαταστήσουν τμήματα του συστήματος, αφού τους απαγορεύει να δημιουργήσουν δικό τους φορτωτή.
- Απαγορεύει στα applets να καλέσουν μεθόδους που ανήκουν στο φορτωτή κλάσεων του συστήματος.

Στο περιβάλλον εκτέλεσης της Java επιτρέπεται να υπάρχουν πολλοί φορτωτές που είναι ενεργοί ταυτόχρονα (ο καθένας με το δικό του χώρο ονομάτων). Οι χώροι ονομάτων επιτρέπουν στη JVM να ομαδοποιεί τις κλάσεις ανάλογα με την προέλευσή τους (τοπικές ή απομακρυσμένες). Με αυτό τον τρόπο, τα ύποπτα applets έχουν περιορισμένο χώρο πρόσβασης στο περιβάλλον εκτέλεσης και δεν μπορούν να επέμβουν σε ευαίσθητα δεδομένα, τοπικά αρχεία κ.λπ. Με άλλα λόγια, οι χώροι ονομάτων αποτελούν την οχύρωση σε ένα αμμοδοχείο.

Πιστοποιητής Ενδιάμεσου Κώδικα (The Byte-Code Verifier): Όπως αναφέρθηκε, ο πηγαίος κώδικας της Java μεταφράζεται σε bytecode (ανεξάρτητο της πλατφόρμας υλοποίησης). Προτού δώσει ο φορτωτής άδεια για εκτέλεση σε κάποιο applet, ο κώδικάς του πρέπει να πιστοποιηθεί απ' τον Πιστοποιητή Ενδιάμεσου Κώδικα. Ο πιστοποιητής επιβεβαιώνει ότι το bytecode του applet, που μπορεί να προέρχεται από άλλη πηγή και όχι από μεταφρασμένο

πηγαίο κώδικα Java, πληροί τις προδιαγραφές ασφαλείας και δεν παραβιάζει τους κανόνες της γλώσσας. Ο πιστοποιητής θεωρεί δεδομένο (εκτός αν αποδειχθεί το αντίθετο) ότι κάθε τμήμα κώδικα προσπαθεί να διασπάσει τα μέσα ασφαλείας του συστήματος.

Γι' αυτό το λόγο, ο πιστοποιητής εκτελεί αρκετούς ελέγχους επικύρωσης του κώδικα. Αρχικά επιβεβαιώνει ότι ο κώδικας συμμορφώνεται με τις προδιαγραφές της γλώσσας. Έπειτα εφαρμόζει (σε χρόνο μεταγλώττισης) ένα ενσωματωμένο σύστημα αποδείξεων (theorem prover) στον κώδικα για να εξετάσει αν το applet προσπαθεί να πλαστογραφήσει δείκτες ή να παρακάμψει περιορισμούς πρόσβασης για να αποκτήσει πρόσβαση σε τοπικό κώδικα. Ο πιστοποιητής εξασφαλίζει ότι:

- Ο μεταφρασμένος κώδικας είναι σωστά δομημένος.
- Οι στοίβες δε θα υπερχειλίσουν/υποχειλίσουν, γιατί αν συμβεί αυτό ο ύποπτος κώδικας αποκτά πρόσβαση σε άσχετα με το applet δεδομένα με απρόβλεπτες συνέπειες. Αυτός ο τύπος επίθεσης έχει επιφέρει πολλές διαρροές ασφάλειας στο παρελθόν.
- Δε θα συμβούν μη-επιτρεπτές μετατροπές τύπων. Για παράδειγμα, δε θα επιτραπεί σε ακέραιους να μετατραπούν σε δείκτες, για να μην υπάρξει πρόσβαση σε απαγορευμένες περιοχές της μνήμης.
- Οι εντολές του bytecode έχουν το σωστό τύπο στις παραμέτρους τους (για τον ίδιο λόγο όπως παραπάνω).
- Όλες οι προσβασεις σε πεδία αντικειμένων είναι νόμιμες, δηλαδή τα ιδιωτικά πεδία ενός αντικειμένου παραμένουν όντως ιδιωτικά.

Με τη χρησιμοποίηση του πιστοποιητή ενδιάμεσου κώδικα η Java πιστοποιεί κάθε ύποπτο τμήμα κώδικα πριν του επιτρέψει να εκτελεστεί στο χώρο ονομάτων του. Έτσι, ενώ οι χώροι ονομάτων διασφαλίζουν ότι κανένα applet δεν μπορεί να επηρεάσει το υπόλοιπο περιβάλλον εκτέλεσης, ο πιστοποιητής διασφαλίζει ότι κανένα applet δεν προκαλεί καταστροφές εντός του χώρου ονομάτων του. Τελικά η JVM θα εκτελέσει μόνο τον κώδικα που πέρασε επιτυχώς από τον πιστοποιητή.

Διαχειριστής Ασφάλειας (The Security Manager): Το πιο σημαντικό τμήμα του μοντέλου ασφάλειας της Java είναι ο διαχειριστής ασφάλειας. Η δουλειά του είναι να εκτελεί ελέγχους κατά το χρόνο εκτέλεσης. Ελέγχει δηλαδή μεθόδους που θέλουν να χρησιμοποιήσουν το σύστημα εισόδου-εξόδου, να έχουν πρόσβαση στο δίκτυο ή να ορίσουν καινούριο φορτωτή κλάσεων. Στις περιπτώσεις αυτές, ο διαχειριστής ασφάλειας έχει το δικαίωμα να ασκήσει βέτο. Για παράδειγμα, αν ένα applet καλέσει μια μέθοδο read, η JVM συμβουλεύεται το διαχειριστή ασφάλειας. Αυτός θα δώσει την άδεια για την ενέργεια μόνο αν εμπιστεύεται το applet, αλλιώς θα απορρίψει την αίτηση για διάβασμα. Από τα παραπάνω είναι φανερό ότι οι ενσωματωμένες κλάσεις της γλώσσας έχουν λιγότερους περιορισμούς από τον κώδικα που κατεβαίνει μέσω διαδικτύου. Παρατηρούμε ότι η δουλειά του διαχειριστή ασφάλειας είναι να επιβλέπει τα σύνορα των αμμοδοχείων κατά το χρόνο εκτέλεσης. Μερικά από τα καθήκοντά του είναι:

- να διαχειρίζεται τις λειτουργίες μέσω sockets.
- να απαγορεύει την πρόσβαση σε εμπιστευτικά δεδομένα όπως τα αρχεία, τα προσωπικά στοιχεία χρηστών κ.λπ.
- να απαγορεύει την πρόσβαση στις υπάρχουσες διεργασίες του λειτουργικού συστήματος και τη δημιουργία καινούριων.
- να επιβλέπει την αλληλεπίδραση των νημάτων εκτέλεσης (threads).

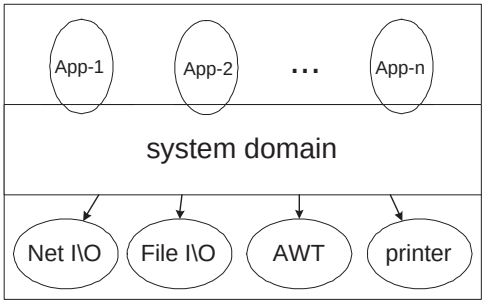
Για να εξασφαλιστεί η συμμόρφωση με την πολιτική ασφαλείας, ακόμα και οι μέθοδοι που ανήκουν στις ενσωματωμένες στη γλώσσα βιβλιοθήκες ρωτούν το διαχειριστή ασφάλειας προτού εκτελέσουν μια ενδεχομένως επικίνδυνη λειτουργία. Επιπλέον, ο διαχειριστής ασφάλειας μπορεί να χαλαρώσει/ενδυναμώσει την πολιτική ασφαλείας ανάλογα με τον κώδικα που εξετάζει. Έτσι, ο διαχειριστής ενός υπολογιστικού συστήματος μπορεί να δημιουργήσει βαθμίδες ασφαλείας· κώδικας από έμπιστη τοποθεσία απολαμβάνει περισσότερα δικαιώματα από κώδικα ύποπτης προέλευσης.

2.1.3 Πεδία Προστασίας της Java

Ένα άλλο βασικό συστατικό του μοντέλου ασφάλειας της Java είναι τα πεδία προστασίας. *Πεδίο προστασίας* (Protection Domain) είναι το τμήμα του συστήματος που είναι απευθείας προσβάσιμο από το σύνολο αντικειμένων που διαχειρίζεται ένα principal. *Principal* ονομάζουμε κάθε αυτόνομη οντότητα στην οποία επιτρέπεται/απαγορεύεται η πρόσβαση στους πόρους του συστήματος από το διαχειριστή ασφάλειας. Το αμμοδοχείο που αναλύσαμε στην προηγούμενη ενότητα είναι στην ουσία ένα πεδίο προστασίας με προκαθορισμένο σύνορο.

Τα πεδία προστασίας αποτελούν ένα αποτελεσματικό μηχανισμό για ομαδοποίηση/απομόνωση μεταξύ ενοτήτων προστασίας. Για παράδειγμα, μπορούμε να απομονώσουμε τα διαφορετικά πεδία προστασίας, έτσι ώστε κάθε επιτρεπτή αλληλεπίδραση να γίνεται είτε μέσω έμπιστου κώδικα συτήματος είτε αφού εγκριθεί από όλα τα πεδία προστασίας που αλληλεπιδρούν. Εδώ πρέπει να σημειωθεί ότι όλοι οι μηχανισμοί προστασίας που αναπτύχθηκαν σε προηγούμενες ενότητες ισχύουν και αν προστεθεί ο μηχανισμός των πεδίων προστασίας στο σύστημα.

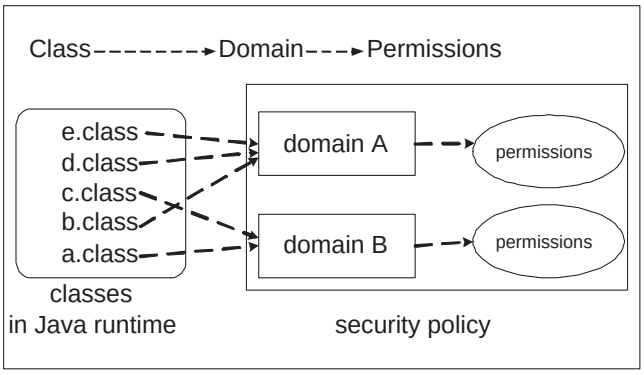
Τα πεδία προστασίας κατατάσσονται σε δύο ξένες μεταξύ τους κατηγορίες: *πεδίο συστήματος* και *πεδίο εφαρμογών*. Οι πόροι του συστήματος όπως το σύστημα αρχείων, η πρόσβαση στο δίκτυο, η οθόνη, το πληκτρολόγιο κ.α. πρέπει να είναι προσβάσιμοι μόνο μέσω πεδίων συστήματος. Στο σχήμα 2.2 φαίνεται η διάταξη των πεδίων προστασίας στο περιβάλλον εκτέλεσης της Java. Ένα πεδίο προστασίας περιέχει τις κλάσεις που τα αντικείμενά τους έχουν το ίδιο



Σχήμα 2.2: Διάταξη Πεδίων Προστασίας.

σύνολο αδειών πρόσβασης. Τα σύνορα των πεδίων προστασίας καθορίζονται από την πολιτική ασφαλείας στο εκάστοτε υπολογιστικό σύστημα. Το περιβάλλον εκτέλεσης αντιστοιχίζει τις κλάσεις στα πεδία προστασίας και ακολούθως στις άδειες πρόσβασης, όπως φαίνεται και στο σχήμα 2.3.

Ένα νήμα εκτέλεσης μπορεί να εμπίπτει εξ' ολοκλήρου σε ένα πεδίο προστασίας ή μπορεί να εκτείνεται σε κάποιο πεδίο εφαρμογών και στο πεδίο συστήματος. Για παράδειγμα, μια εφαρμογή που εμφανίζει ένα μήνυμα πρέπει αναγκαστικά να αλληλεπιδράσει με το πεδίο συστήματος γιατί είναι ο μόνος τρόπος πρόσβασης σε μια συσκευή εξόδου. Σε αυτή την περίπτωση, είναι σημαντικό να μην αυξηθούν οι άδειες πρόσβασης του πεδίου εφαρμογών όταν καλεί το πεδίο συστήματος για να μην υπάρξουν επιπλοκές στην ασφάλεια. Το ίδιο πρέπει να συμβαίνει και στην αντίστροφη περίπτωση, όταν δηλαδή το πεδίο συστήματος καλεί ένα πεδίο εφαρμογών. Με άλλα λόγια, δεν πρέπει να αυξάνονται τα δικαιώματα ενός λιγότερο ισχυρού πεδίου προστασίας όταν καλεί ή όταν καλείται από ένα πιο ισχυρό πεδίο προστασίας.



Σχήμα 2.3: Αντιστοίχιση των κλάσεων στα πεδία προστασίας.

Στη γενική περίπτωση που ένα νήμα εκτέλεσης διασχίζει περισσότερα από ένα πεδία προστασίας πρέπει να ισχύουν τα παρακάτω:

- Το σύνολο αδειών ενός τέτοιου νήματος βρίσκεται από την τομή των συνόλων αδειών των πεδίων προστασίας που διασχίζει.
- Όταν ένα τμήμα κώδικα καλεί τη μέθοδο doPrivileged (περιγράφεται παρακάτω), το σύνολο αδειών του νήματος εκτέλεσης περιλαμβάνει μια άδεια μόνο αν αυτή δίνεται από όλα τα πεδία προστασίας που καλούνται.

Η μέθοδος doPrivileged επιτρέπει σε ένα έμπιστο τμήμα κώδικα να αποκτήσει προσωρινή πρόσβαση σε περισσότερους πόρους του συστήματος από όσους αρχικά διατέθηκαν στην εφαρμογή που το περιέχει. Αυτό είναι απαραίτητο σε κάποιες περιπτώσεις. Για παράδειγμα, μια εφαρμογή δεν έχει πρόσβαση στα αρχεία που περιέχουν γραμματοσειρές, αλλά μια διεργασία συστήματος πρέπει να τις χρησιμοποιήσει εκ μέρους του χρήστη για να τυπώσει ένα κείμενο. Η μέθοδος doPrivileged χρησιμοποιείται από το πεδίο συστήματος για να λυθεί το πρόβλημα· μάλιστα η μέθοδος είναι διαθέσιμη σε όλα τα πεδία προστασίας.

Στο χρόνο εκτέλεσης, όταν γίνεται αίτηση για χρήση κάποιου πόρου του συστήματος, ο κώδικας που τη χειρίζεται καλεί μια μέθοδο που αποτιμά την αίτηση και αποφασίζει αν πρέπει να εγκριθεί ή να απορριφθεί. Οι δύο κανόνες για την αποτίμηση τέτοιων αιτήσεων αναφέρθηκαν παραπάνω. Αυτό που γίνεται συνήθως είναι να εξετάζονται οι άδειες που δόθηκαν σε παρεμφερείς κλήσεις στο παρελθόν και αν εγκριθεί η αίτηση τότε η μέθοδος επιστρέφει κανονικά, αλλιώς καλεί μια εξαίρεση ασφαλείας.

Τέλος, αναφέρουμε ότι κάθε πεδίο προστασίας μπορεί να υλοποιεί επιπρόσθετους περιορισμούς ασφαλείας μέσα στα σύνορά του. Για παράδειγμα, μια εφαρμογή τραπεζικών συναλλαγών πρέπει να έχει μηχανισμούς προστασίας για τους λογαριασμούς, τους πελάτες κ.λπ. Η σημασιολογία τέτοιων προβλημάτων είναι δύσκολο να οριστεί και να αντιμετωπιστεί από το Java2SDK. Παρ' όλα αυτά, υπάρχουν έτοιμες συναρτήσεις στις βιβλιοθήκες της γλώσσας που μπορούν να φανούν χρήσιμες στους προγραμματιστές τέτοιων εφαρμογών.

2.1.4 Ασφαλής επικοινωνία μεταξύ απομακρυσμένων χρηστών

Στις παραγράφους 2.1.2 και 2.1.3 περιγράψαμε τους μηχανισμούς της Java για την ασφαλή εκτέλεση ύποπτου κώδικα. Εκτός από αυτό, είναι πολύ σημαντικό να γίνεται πιστοποίηση της πηγής προέλευσης του κώδικα. Ένας χρήστης πρέπει να γνωρίζει ότι αυτός που βρίσκεται στην άλλη άκρη της γραμμής είναι όντως αυτός με τον οποίο θέλει να επικοινωνήσει και όχι κάποιος εισβολέας που παρεμβάλλεται στη γραμμή. Επίσης πρέπει να γνωρίζει ότι ο κώδικας που κατέβασε είναι όντως αυτός που ήθελε να κατεβάσει και δεν αντικαταστάθηκε από κάποιο αντίγραφο.

Η Java παρέχει πακέτα όπως το `java.security` και το `javax.crypto` για την ασφαλή επικοινωνία μεταξύ απομακρυσμένων χρηστών. Λεπτομερή περιγραφή των πακέτων θα βρείτε στο [1]. Στα πακέτα αυτά παρέχονται κλάσεις για παραγωγή ψηφιακών υπογραφών, συναρτήσεις κατακερματισμού και για συμμετρική και ασύμμετρη κρυπτογράφηση. Υποστηρίζονται αλγόριθμοι όπως Diffie-Hellman για την προκατανομή κλειδιών, RSA για την παραγωγή ψηφιακών υπογραφών, DES, DSA, RC2 και άλλοι. Ένας αποστολέας μπορεί να συμπιέσει τον κώδικά του και τα απαραίτητα μεταδεδομένα για ένα πρόγραμμα σε ένα JAR (Java ARchive) αρχείο και μετά να το υπογράψει χρησιμοποιώντας μια ψηφιακή υπογραφή. Ο παραλήπτης του κώδικα μπορεί να διαπιστώσει την αυθεντικότητα της πηγής εξετάζοντας τη γνησιότητα της υπογραφής. Τέλος, η Java παρέχει κλάσεις για τη δημιουργία Λιστών Ελεγχόμενης Προσβασης (Access Control Lists). Οι λίστες αυτές χρησιμοποιούνται για να επιτρέπουν/απαγορεύουν στους χρήστες ενός συστήματος την πρόσβαση στους πόρους του. Για περισσότερες πληροφορίες για το μοντέλο ασφάλειας της Java δείτε τα [5, 21].

2.2 Το μοντέλο ασφάλειας στο πλαίσιο .NET και τη C#

2.2.1 Σύντομη περιγραφή του πλαισίου .NET και της C#

Το πλαίσιο .NET είναι ένα περιβάλλον ανάπτυξης λογισμικού που δημιουργήθηκε απ' τη Microsoft. Όπως και στην πλατφόρμα Java, τα προγράμματα που εκτελούνται στην πλατφόρμα .NET μεταφράζονται σε μια ενδιάμεση γλώσσα, την MSIL (Microsoft Intermediate Language) και μετά εκτελούνται από ένα υποθετικό επεξεργαστή που ονομάζεται Κοινό Περιβάλλον Εκτέλεσης για Όλες τις Γλώσσες. Η πλατφόρμα .NET υποστηρίζει πολλές γλώσσες προγραμματισμού, και όλες μεταφράζονται στην MSIL πριν εκτελεστούν. Με αυτό τον τρόπο επιτρέπεται η "αφομοίωση" μεταξύ των γλωσσών και υπάρχει δυνατότητα ανάπτυξης μιας εφαρμογής χρησιμοποιώντας πάνω από μία γλώσσες. Το πλαίσιο .NET παρέχει ένα κοινό σύστημα τύπων για όλες τις γλώσσες που τρέχουν στο CLR, που ονομάζεται Common Type System (CTS). Το CTS ορίζει τα μεγέθη των βασικών τύπων δεδομένων και όλες οι γλώσσες που τρέχουν στο CLR πρέπει να ακολουθούν τις προδιαγραφές του. Ορίζοντας τα μεγέθη των τύπων, το CTS βοηθάει στην ανεξαρτησία του κώδικα από την πλατφόρμα υλοποίησης. Η πλατφόρμα .NET, όπως και η πλατφόρμα Java, υποστηρίζει αυτόματη διαχείρηση μνήμης χρησιμοποιώντας *συλλογή σκουπιδιών* (garbage collection). Τέλος, δίνει τη δυνατότητα της κλήσης unmanaged κώδικα, δηλαδή κώδικα που δεν τρέχει στο CLR, όπως DLL βιβλιοθήκες και κώδικα που ακολουθεί το COM μοντέλο.

Η C# είναι η μόνη γλώσσα που κατασκευάστηκε ειδικά για το πλαίσιο .NET. Έχει πάρει αρκετά χαρακτηριστικά από τη C++ και τη Java, ακολουθεί δηλαδή το αντικειμενοστρεφές μοντέλο προγραμματισμού. Πολλές μέθοδοι της C# και της Java έχουν ακριβώς το ίδιο όνομα, με τη διαφορά ότι στη C# τα ονόματα των μεθόδων ξεκινάνε με κεφαλαίο γράμμα ενώ στη Java με μικρό. Οι κλάσεις της C# ομαδοποιούνται σε χώρους ονομάτων (namespaces). Επίσης, η C# υποστηρίζει και αυτή το μηχανισμό των εξαιρέσεων. Υπάρχουν όμως και αρκετές διαφορές από τη Java. Μερικές απ' αυτές είναι:

- Μπορεί να υπάρχει μεταβλητός αριθμός παραμέτρων σε μια συνάρτηση.
- Υποστηρίζεται το πέρασμα παραμέτρων κατ' αναφορά με τη χρήση των τελεστών ref, out.
- Η εντολή foreach.
- Η εντολή goto.
- Οδηγίες στον προεπεξεργαστή.
- Οριζόμενες από το χρήστη μετατροπές τύπων.

Λεπτομερή περιγραφή της C# και των έτοιμων βιβλιοθηκών του πλαισίου .NET θα βρείτε στα [3, 10].

2.2.2 Ασφάλεια Βασισμένη σε Αποδείξεις

Το μοντέλο ασφάλειας του πλαισίου .NET βασίζεται σε ένα τροποποιήσιμο σύνολο κανόνων που ονομάζεται πολιτική ασφαλείας. Έτσι, ένας χρήστης ή ο διαχειριστής ενός υπολογιστικού συστήματος μπορεί να ορίσει πότε εμπιστεύεται ένα τμήμα κώδικα και αναλόγως να του δίνει/απαγορεύει την πρόσβαση σε τοπικούς πόρους ή, στη χειρότερη περίπτωση, να μην το αφήνει καν να εκτελεστεί.

Ο κώδικας που προορίζεται για εκτέλεση στο CLR οργανώνεται σε ενότητες που λέγονται *assemblies*. Όταν φορτώνει τον κώδικα, το CLR εξετάζει τις *αποδείξεις* που περιλαμβάνονται σε ένα assembly, όπως την ψηφιακή υπογραφή του δημιουργού του ή την προέλευσή του. Ανάλογα με τις αποδείξεις, το CLR κατατάσσει το assembly στην κατάλληλη *ομάδα κώδικα* (code group). Κάθε ομάδα κώδικα είναι ορισμένη έτσι ώστε να εξετάζει για την ύπαρξη συγκεκριμένων αποδείξεων και να αντιστοιχεί στις ανάλογες *άδειες πρόσβασης* (permissions). Έτσι, όταν ένα assembly κατατάσσεται σε μια ομάδα κώδικα απολαμβάνει τις αντίστοιχες άδειες που αναλογούν στην ομάδα αυτή.

Οι άδειες είναι *αντικείμενα* που καθορίζουν την πρόσβαση σε προστατευόμενους πόρους του συστήματος. Κάθε τέτοιο αντικείμενο έχει παραμέτρους που προσαρμόζονται έτσι ώστε να μπορεί να παραστήσει πολλές διαφορετικές άδειες πρόσβασης. Για παράδειγμα, το αντικείμενο `FileIOPermission` αντιπροσωπεύει τα δικαιώματα δημιουργίας/ανάγνωσης/εγγραφής/τροποποίησης ενός αρχείου· ανάλογα με τις παραμέτρους του μπορεί να δίνει άδεια ανάγνωσης ενός αρχείου, άδεια ανάγνωσης/εγγραφής ενός αρχείου ή και άδεια ανάγνωσης όλων των αρχείων ενός φακέλου. Οι παράμετροι αυτές ρυθμίζονται από το διαχειριστή του συστήματος. Έτσι, παρόλο που κάθε εφαρμογή μπορεί να δημιουργήσει και να ρυθμίσει όπως θέλει αντικείμενα-άδειες, η άδεια πρόσβασης σε πόρους του συστήματος δίνεται μόνο απ' την πολιτική ασφαλείας του, άρα απ' το διαχειριστή του συστήματος.

Υπάρχουν τέσσερα επίπεδα στο μοντέλο ασφάλειας του πλαισίου .NET, τα οποία περιγράφονται στον πίνακα 2.1. Το επίπεδο επιχειρησιακής πολιτικής είναι το υψηλότερο στην ιεραρχία (αυτό που εφαρμόζεται πρώτο) και το επίπεδο πεδίου εφαρμογών το χαμηλότερο. Κάθε επίπεδο ασφάλειας περιλαμβάνει μια ιεραρχία ομάδων κώδικα. Οι διαχειριστές κάθε επι-

Πίνακας 2.1: Επίπεδα Πολιτικής Ασφαλείας.

Επίπεδο Πολιτικής Ασφαλείας	Περιγραφή
Επιχειρησιακή Πολιτική	Ορίζεται από διαχειριστές που διαμορφώνουν την πολιτική ασφαλείας ολόκληρης επιχείρησης.
Πολιτική Μηχανής	Ορίζεται από διαχειριστές που διαμορφώνουν την πολιτική ασφαλείας ενός υπολογιστή.
Πολιτική Χρήστη	Ορίζεται από χρήστες που διαμορφώνουν την πολιτική ασφαλείας του λογαριασμού τους.
Πολιτική Πεδίου Εφαρμογών	Ορίζεται από τη διεργασία που τρέχει το CLR για πολιτική ασφαλείας κατά το χρόνο φόρτωσης. Αυτό το επίπεδο δεν είναι διαχειρίσιμο.

πέδου είναι ελεύθεροι να φτιάξουν τις δικές τους ομάδες κώδικα και τα αντίστοιχα σύνολα αδειών πρόσβασης. Στο χρόνο φόρτωσης, το σύστημα ασφαλείας εξετάζει όλα τα επίπεδα και οι τελικές άδειες υπολογίζονται από την τομή των αδειών όλων των επιπέδων. Ο διαχειριστής ενός χαμηλότερου επιπέδου δεν μπορεί να χαλαρώσει την πολιτική ασφαλείας που ορίστηκε σε υψηλότερο επίπεδο, μπορεί όμως να την κάνει ακόμα πιο αυστηρή. Στις προεπιλεγμένες επιλογές ασφάλειας, τα επίπεδα επιχείρησης και χρήστη δε θέτουν κανένα περιορισμό, αυτό όμως

δε σημαίνει ότι οι άδειες που δίνονται σε ένα assembly είναι απεριόριστες, γιατί στο επίπεδο μηχανής υπάρχουν περιορισμοί. Αφού οι τελικές άδειες υπολογίζονται από την τομή όλων των επιπέδων, οι προεπιλεγμένες άδειες ισούνται με τις άδειες που δίνονται στο επίπεδο μηχανής.

Η κατάταξη ενός assembly σε μια απ' τις ομάδες κώδικα γίνεται με την εξέταση μιας συνθήκης μέλους. Το assembly παρουσιάζει αποδείξεις στο CLR το οποίο κοιτάζει τις συνθήκες μέλους των διαφόρων ομάδων κώδικα και ανάλογα με τις αποδείξεις κατατάσσει το assembly στη σωστή ομάδα. Ένα assembly μπορεί να ανήκει σε περισσότερες από μια ομάδες κώδικα. Οι ακόλουθες αποδείξεις μπορούν να χρησιμοποιηθούν ως συνθήκες μέλους:

- Ο φάκελος εγκατάστασης της εφαρμογής.
- Η κρυπτογραφική συνάρτηση κατακερματισμού ενός assembly.
- Η ψηφιακή υπογραφή του διανομέα ενός assembly.
- Η ιστοσελίδα απ' όπου προέρχεται ένα assembly.
- Η ζώνη προέλευσης ενός assembly: απ' το συγκεκριμένο υπολογιστή, απ' το τοπικό δίκτυο, απ' το διαδίκτυο, από έμπιστες ιστοσελίδες, από πιθανώς επικίνδυνες ιστοσελίδες.
- Το κρυπτογραφικό ισχυρό όνομα[1] ενός assembly.

Οι παραπάνω αποδείξεις μπορούν και να συνδυαστούν μεταξύ τους για την κατασκευή πιο αυστηρών συνθηκών μέλους. Οι προεπιλεγμένες επιλογές ασφάλειας αρκούν για τις περισσότερες περιπτώσεις αφού παρέχουν ικανοποιητικό επίπεδο προστασίας. Κώδικας που δεν προέρχεται από το συγκεκριμένο υπολογιστή θεωρείται λιγότερο έμπιστος και έχει περιορισμένη πρόσβαση στους πόρους του υπολογιστή. Ειδικότερα, στις προεπιλεγμένες επιλογές ασφάλειας ο κώδικας που προέρχεται απ' το τοπικό δίκτυο ή το διαδίκτυο αντιμετωπίζεται ως εξής:

- Δεν έχει άδεια ανάγνωσης/εγγραφής σε τοπικό δίσκο.
- Δεν έχει άδεια ανάγνωσης/εγγραφής στο registry του συστήματος.
- Έχει άδεια επικοινωνίας με την ιστοσελίδα προέλευσής του.
- Ο κώδικας απ' το τοπικό δίκτυο έχει απεριόριστη πρόσβαση στα τμήματα της διαπροσωπείας χρήστη ενώ ο κώδικας απ' το διαδίκτυο μόνο σε κάποια απ' αυτά.

Σε μερικές περιπτώσεις, οι προεπιλεγμένες επιλογές πρέπει να αλλαχθούν. Για παράδειγμα, μπορεί να υπάρχει περισσότερη εμπιστοσύνη στον κώδικα ενός συγκεκριμένου διανομέα από τον υπόλοιπο κώδικα του διαδικτύου. Αυτός ο κώδικας θα έχει πρόσβαση σε κάποιους πόρους ανεξάρτητα από τον υπολογιστή στον οποίο εκτελείται. Επίσης, μπορεί να χρειάζεται να περιοριστούν τα δικαιώματα του κώδικα που τρέχει στον τοπικό υπολογιστή. Αυτό γίνεται όταν ο διαχειριστής ενός συστήματος θέλει να απαγορεύσει στους χρήστες την εγκατάσταση και την εκτέλεση ύποπτων εφαρμογών.

Το χαμηλότερο επίπεδο πολιτικής ασφαλείας ορίζεται στο χρόνο εκτέλεσης από τη διεργασία του συστήματος που τρέχει το CLR και όχι από καποιον χρήστη/διαχειριστή όπως τα τρία υψηλότερα επίπεδα. Η διεργασία αυτή έχει ένα προεπιλεγμένο πεδίο εφαρμογών όπου φορτώνει όλες τις εφαρμογές που εκτελούνται από το CLR. Μπορεί όμως να δημιουργήσει πολλά πεδία εφαρμογών ώστε κάθε μια να τρέχει στο ξεχωριστό της πεδίο και να μην μπορεί να προκαλέσει καταστροφές σε άλλα πεδία εφαρμογών. Αυτή η πολιτική οχύρωσης του κώδικα σε διαφορετικά πεδία εφαρμογών έχει την ίδια λογική με τα αμμοδοχεία της πλατφόρμας Java. Για να λειτουργήσει όμως αποτελεσματικά πρέπει οι εφαρμογές να μην έχουν απεριόριστες άδειες, αλλιώς μπορούν να επηρεάσουν όποιο πεδίο εφαρμογών θέλουν. Περισσότερες πληροφορίες γι' αυτό το θέμα υπάρχουν στο [15].

[1] Ισχυρό όνομα (strong name) ονομάζεται ένα κρυπτογραφημένο αναγνωριστικό που χαρακτηρίζει με μοναδικό τρόπο ένα assembly.

2.2.3 Εργαλεία για τη διαμόρφωση πολιτικής ασφαλείας με αποδείξεις

Το πλαίσιο .NET παρέχει το εργαλείο `mscorcfg.msc` για τη διαμόρφωση της επιθυμητής πολιτικής ασφαλείας. Χρησιμοποιώντας το, ο διαχειριστής οποιουδήποτε επιπέδου μπορεί να προσαρμόσει τις άδειες ανάλογα με τη ζώνη προέλευσης ενός assembly.

Μια εφαρμογή που χρειάζεται περισσότερες άδειες απ' αυτές που τις παρέχονται από την ισχύουσα πολιτική ασφαλείας δε θα εκτελεστεί καν ή θα παρουσιάσει προβλήματα κατά την εκτέλεση. Σε περίπτωση που ο διαχειριστής του συστήματος εμπιστεύεται την εφαρμογή μπορεί να αυξήσει τις άδειές της χωρίς να αυξηθούν οι άδειες των υπόλοιπων εφαρμογών από την ίδια ζώνη προέλευσης:

1. Χρησιμοποιώντας το εργαλείο `permview.exe` βρίσκει το ελάχιστο σύνολο αδειών που απαιτεί η εφαρμογή για τη σωστή εκτέλεσή της.
2. Βρίσκει ένα χαρακτηριστικό γνώρισμα της εφαρμογής όπως π.χ. το ισχυρό της όνομα.
3. Φτιάχνει μια καινούρια ομάδα κώδικα που χρησιμοποιεί ως συνθήκη μέλους το παραπάνω χαρακτηριστικό.
4. Χρησιμοποιώντας το `mscorcfg.msc` δημιουργεί ένα καινούριο σύνολο αδειών που περιλαμβάνει όλες τις άδειες που απαιτεί η εφαρμογή και μόνο αυτές.
5. Αντιστοιχίζει το νέο σύνολο αδειών στη νέα ομάδα κώδικα.

Είδαμε παραπάνω(σελ. 26) ότι η συνολική πολιτική ασφαλείας βρίσκεται από την τομή των πολιτικών όλων των επιπέδων. Σε ορισμένες περιπτώσεις όμως, ο διαχειριστής ενός επιπέδου θέλει να μη μειώνονται οι άδειες μιας εφαρμογής από πολιτικές χαμηλότερων επιπέδων. Αυτό γίνεται αν χαρακτηρίσει την ομάδα κώδικα στην οποία ανήκει η εφαρμογή ως `LevelFinal`. Για παράδειγμα, αν δώσει το γνώρισμα `LevelFinal` σε μια ομάδα κώδικα στο επίπεδο επιχείρησης, κάθε ομάδα κώδικα χαμηλότερου επιπέδου στην οποία ανήκει η εφαρμογή δε θα επηρεάσει τις άδειες της εφαρμογής. Εκτός από το `LevelFinal` υπάρχει και το γνώρισμα `Exclusive`. Χρησιμοποιείται όταν ο διαχειριστής ενός επιπέδου θέλει να μην αλλάζουν οι άδειες μιας εφαρμογής που ανήκει σε περισσότερες από μια ομάδες κώδικα. Χαρακτηρίζοντας τη μια από αυτές ως `Exclusive`, απαγορεύει στις άλλες ομάδες *του ίδιου επιπέδου* να αλλάξουν τις άδειες μιας εφαρμογής. Το `Exclusive` δεν επηρεάζει πολιτικές που γίνονται σε άλλα επίπεδα. Περισσότερες πληροφορίες για την ασφάλεια που βασίζεται σε αποδείξεις[2] θα βρείτε στο [6].

2.2.4 Προστακτική και Δηλωτική Σύνταξη στη C#

Είδαμε στην παράγραφο 2.2.2 ότι για να εκτελεστεί μια εφαρμογή ζητάει απ' το CLR τις άδειες που χρειάζεται. Αν αυτές μπορούν να της δοθούν τότε εκτελείται κανονικά αλλιώς δεν εκτελείται καθόλου. Μια εφαρμογή ζητάει τις απαιτούμενες άδειες χρησιμοποιώντας *προστακτική ή δηλωτική σύνταξη*. Οι δηλωτικές κλήσεις γίνονται χρησιμοποιώντας *γνωρίσματα* (attributes). Οι προστακτικές κλήσεις γίνονται δημιουργώντας νέα αντικείμενα-άδειες.

Η χρήση της προστακτικής και της δηλωτικής σύνταξης θα περιγραφεί με ένα παράδειγμα. Η κλάση NativeMethods καλεί κώδικα που δεν τρέχει στο CLR (unmanaged code).

```
class NativeMethods {
   [DllImport("msvcrt.dll")]
   public static extern int puts(string str);
   [DllImport("msvcrt.dll")]
   internal static extern int _flushall();
}
```

[2] Για αρχεία που περιέχουν πολιτικές ασφαλείας πρέπει να χρησιμοποιείται το σύστημα αρχείων NTFS, όχι FAT. Το NTFS παρέχει περιορισμένη πρόσβαση στα αρχεία πολιτικής ασφαλείας και μόνο οι διαχειριστές ενός επιπέδου μπορούν να τροποποιούν τα αρχεία ασφαλείας για το επίπεδο αυτό.

Για να κληθούν τα `dll` αρχεία πρέπει να υπάρχει η άδεια για εκτέλεση unmanaged κώδικα. Ένας χρήστης που κατεβάζει ύποπτο κώδικα απ' το διαδίκτυο στον υπολογιστή του και θέλει να του απαγορεύσει την πρόσβαση σε unmanaged κώδικα μπορεί να γράψει τα παρακάτω χρησιμοποιώντας προστακτική σύνταξη:

```
class WellMeaningCode{
   public void CallPlugIn(EvilCode plugin) {
     SecurityPermission perm =
       new SecurityPermission(SecurityPermissionFlag.UnmanagedCode);
     perm.Deny();
     plugin.DoWork();
   }
}
```

Ο χρήστης δημιουργεί το αντικείμενο-άδεια και μετά καλεί τη συνάρτηση `Deny` που απαγορεύει την πρόσβαση σε unmanaged κώδικα. Αν όμως έχει κατατάξει το plugin σε ομάδα κώδικα με πλήρη εμπιστοσύνη άρα απεριόριστες άδειες, ένας επιτιθέμενος θα μπορούσε εύκολα να κάνει το εξής:

```
class EvilCode {
   void DoWork(){
     SecurityPermission perm =
       new SecurityPermission(SecurityPermissionFlag.UnmanagedCode);
     perm.Assert();
     try {
       NativeMethods.puts("Hello World!");
       NativeMethods._flushall();
     }
     catch (SecurityException) { }
   }
}
```

Με την `Assert` ο εισβολέας ακυρώνει τη `Deny` του χρήστη. Για να έχει δικαίωμα μια εφαρμογή να καλέσει την `Assert` πρέπει να έχει τη σχετική άδεια από το σύστημα ασφαλείας. Την άδεια αυτή φυσικά την έχουν οι εφαρμογές που θεωρούνται πλήρως έμπιστες από το διαχειριστή ασφαλείας. Γι' αυτό, είναι σημαντικό ο διαχειριστής ασφάλειας να είναι απόλυτα σίγουρος για τις εφαρμογές στις οποίες παρέχει πλήρη εμπιστοσύνη. Ο ίδιος κώδικας μπορεί να γραφτεί χρησιμοποιώντας δηλωτική σύνταξη. Για παράδειγμα, η `WellMeaningCode` γράφεται:

```
[SecurityPermission(SecurityAction.Deny, Flags =
   SecurityPermissionFlag.UnmanagedCode)]
class WellMeaningCode{
   public void CallPlugIn(EvilCode plugin) {
     plugin.DoWork();
   }
}
```

Περισσότερα για τη δηλωτική και την προστακτική σύνταξη και για τους κινδύνους που υπάρχουν όταν μια εφαρμογή έχει απεριόριστες άδειες θα βρείτε στα [4, 15].

2.2.5 Ασφάλεια Βασισμένη σε Ρόλους - Κρυπτογραφία

Εκτός απ' τους τρόπους προφύλαξης από ύποπτο κώδικα, το πλαίσιο .NET παρέχει ένα μοντέλο για την πιστοποίηση της ταυτότητας των χρηστών που ονομάζεται *ασφάλεια βασισμένη σε ρόλους* (Role Based Security). Στο μοντέλο αυτό, κάθε χρήστης αντιπροσωπεύεται από ένα principal. Αφού γίνει πιστοποίηση της ταυτότητας ενός principal, του ανατίθεται ένας ή περισσότεροι ρόλοι ανάλογα με τις ενέργειες που θέλει να εκτελέσει. Ρόλος ονομάζεται το σύνολο των χρηστών που έχουν τις ίδιες άδειες πρόσβασης από το σύστημα.

Ο πιο κατάλληλος τρόπος για την πιστοποίηση της ταυτότητας ενός principal είναι με τη χρησιμοποίηση κρυπτογραφικών μεθόδων. Το πλαίσιο .NET παρέχει συναρτήσεις για συμμετρική/ασύμμετρη κρυπτογράφηση, ψηφιακές υπογραφές, συναρτήσεις κατακερματισμού και γεννήτριες τυχαίων αριθμών. Υλοποιούνται αλγόριθμοι όπως RSA, DSA, DES, Triple DES, Rijndael/AES, MD5, SHA-1, SHA-512 κ.α. Επιπλέον πληροφορίες για την ασφάλεια που βασίζεται σε ρόλους και για τις κρυπτογραφικές συναρτήσεις του πλαισίου .NET θα βρείτε στα [30, 3].

Κεφάλαιο 3

Η ακαδημαϊκή προσέγγιση στην ασφάλεια

Στο κεφάλαιο 2 είδαμε πως αντιμετωπίζεται το θέμα της ασφάλειας από τις δύο πιο διαδεδομένες πλατφόρμες του εμπορίου, την πλατφόρμα Java και το πλαίσιο .NET. Το θέμα της ασφάλειας του εκτελέσιμου κώδικα έχει απασχολήσει αρκετά και την ακαδημαϊκή κοινότητα και έχουν προταθεί διάφορες ενδιαφέρουσες λύσεις, οι οποίες δεν έχουν υιοθετηθεί ακόμα από τις ευρέως χρησιμοποιούμενες γλώσσες προγραμματισμού. Είναι γνωστό εξάλλου ότι η έρευνα στην πληροφορική γίνεται κατά κύριο λόγο στα πανεπιστήμια και τα αποτελέσματά της περνούν στη βιομηχανία αρκετά χρόνια μετά.[1] Στο κεφάλαιο αυτό θα περιγράψουμε τις κυριότερες λύσεις που προτάθηκαν από την ακαδημαϊκή κοινότητα τα τελευταία χρόνια.

Κάθε λογισμικό σύστημα που πρόκειται να εκτελέσει ύποπτο κώδικα πρέπει να περιλαμβάνει τα εξής τμήματα:

1. Μια *ασφαλή γλώσσα προγραμματισμού* με ορθό σύστημα τύπων για να εντοπίζει όσο περισσότερους κινδύνους γίνεται κατά το χρόνο μεταγλώττισης.

2. Ένα *περιβάλλον εκτέλεσης* που θα εντοπίζει τους κινδύνους που δεν μπορούν να ελεγχθούν στατικά.

Οι λύσεις που προτείνονται από τη βιομηχανία λογισμικού έχουν δύο σημαντικά μειονεκτήματα. Πρώτον, η νοητή μηχανή πρέπει να παρεμβάλλεται στην εκτέλεση του πιθανώς αναξιόπιστου κώδικα και αυτό οδηγεί σε αναπόφευκτη απώλεια στην απόδοση. Ένας αριθμός από απαιτούμενους ελέγχους μπορούν να γίνουν στατικά και εισάγουν για μία μόνο φορά μια μικρή καθυστέρηση. Άλλοι έλεγχοι όμως, μπορούν να γίνουν μόνο δυναμικά και έχουν μεγάλες επιπτώσεις στην απόδοση, αν συμβαίνουν μέσα σε βρόχους. Από την άλλη πλευρά υπάρχει πιθανώς ένα ακόμη πρόβλημα: Η εγγύηση για την ασφάλεια είναι έγκυρη μόνο όταν η νοητή μηχανή είναι καλά σχεδιασμένη και σωστά υλοποιημένη. Αυτό σημαίνει ότι δεν πρέπει να υπάρχουν σφάλματα σε κανένα υποσύστημα της νοητής μηχανής (φορτωτής, ελεγκτής, μεταφραστής ή μεταγλωττιστής τελευταίας στιγμής και βιβλιοθήκη χρόνου εκτέλεσης) και ότι δεν πρέπει να υπάρχουν προβλήματα στο σύστημα προδιαγραφής της ασφάλειας. Ακόμη και αν το τελευταίο μπορεί να μην έχει προβλήματα, δεν μπορεί κανείς εύκολα να εγγυηθεί τα πρώτα, καθώς η υλοποίηση των υποσυστημάτων της νοητής μηχανής συνήθως περιέχει πολλές χιλιάδες γραμμές κώδικα.

Η συνολική ακαδημαϊκή προσέγγιση έθεσε σαν στόχο της να ξεπεράσει τα προαναφερθέντα μειονεκτήματα. Για να γίνει αυτό εφικτό, πρέπει να πληρούνται οι παρακάτω προϋποθέσεις:

- Δε θα πρέπει να επιτρέπεται απώλεια απόδοσης, αν μπορεί να αποφευχθεί.

- Ελάχιστα πράγματα θα πρέπει να θεωρούνται αξιόπιστα χωρίς έλεγχο (συμπεριλαμβανομένης της πολιτικής ασφάλειας και της ίδιας της νοητής μηχανής) και αυτά τα πράγματα θα πρέπει να είναι όσο πιο απλά γίνεται.

[1] Χαρακτηριστικό παράδειγμα είναι ότι η συλλογή σκουπιδιών (για την αυτόματη διαχείριση της μνήμης) που χρησιμοποιείται σήμερα είχε προταθεί από τον H. Baker ήδη από το 1977-78 (βλ. [11, 12]).

- Η γλώσσα προγραμματισμού θα πρέπει να είναι αρκετά εκφραστική, ώστε να επιτρέπει την αποδοτική υλοποίηση δομικών μονάδων ενός συστήματος λογισμικού, χωρίς να παρακάμπτονται περιορισμοί ασφάλειας.

- Εκφραστική δύναμη: Η προτεινόμενη λύση πρέπει να μπορεί να εκφράζει και να πιστοποιεί τυχαίες ιδιότητες των δομικών μονάδων ενός προγράμματος.

- Ευελιξία: Σε διαφορετικές περιοχές εφαρμογών, διαφορετικοί αποδέκτες κώδικα ορίζουν την ασφάλεια με διαφορετικό τρόπο ο καθένας. Θα ήταν κατα συνέπεια καλό, ο εκτελέσιμος κώδικας να μπορεί να πιστοποιηθεί όχι μόνο ότι καλύπτει τις ακριβείς προδιαγραφές ασφαλείας που περιμένει ένας παραλήπτης, αλλά και ένα σύνολο προδιαγραφών πιο ισχυρό από αυτό.

- Κλιμάκωση: Η προτεινόμενη λύση πρέπει να είναι αποδοτική όταν το μέγεθος των αναπτυσσόμενων εφαρμογών αυξάνει.

Το υπόλοιπο κεφάλαιο είναι δομημένο ως εξής: Στις παραγράφους 3.1, 3.2, 3.4 παρουσιάζονται τεχνικές που μπορούν να ενσωματωθούν στη γλώσσα προγραμματισμού ενώ στην παράγραφο 3.3 περιγράφονται οι επόπτες (monitors), που είναι προγράμματα που ελέγχουν προδιαγραφές ασφαλείας κατά το χρόνο εκτέλεσης.

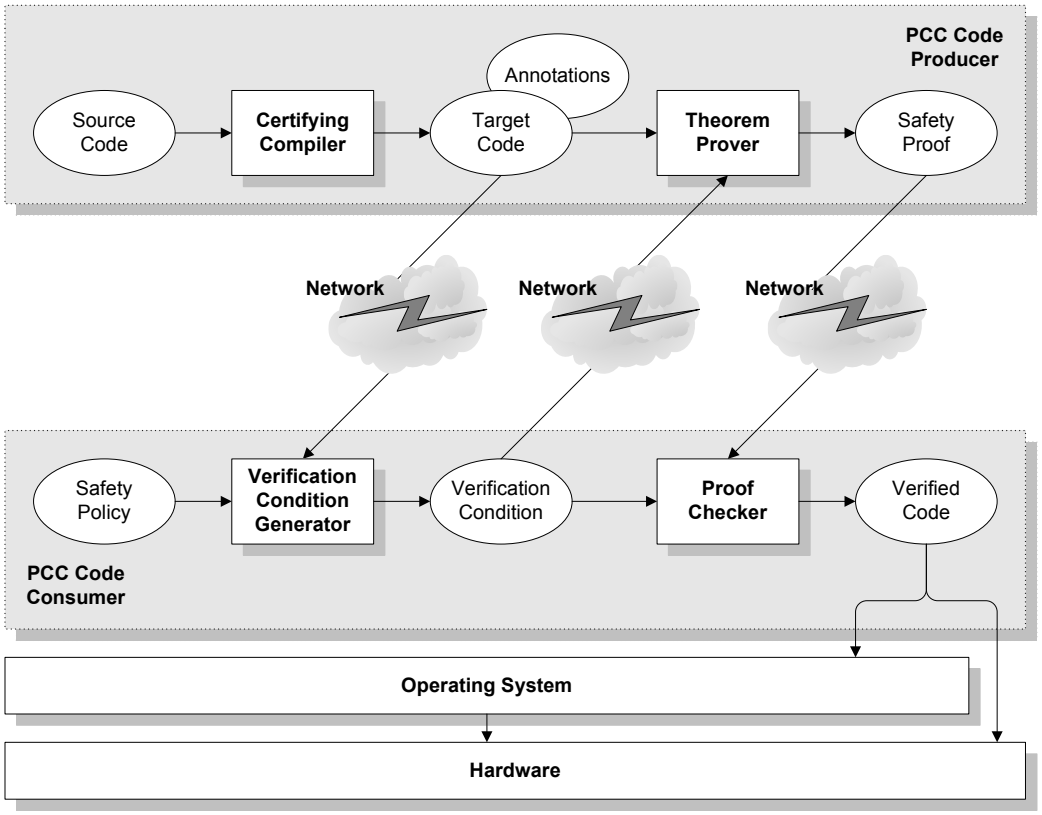
3.1 Κώδικας που φέρει αποδείξεις και παρεμφερή συστήματα

3.1.1 Proof-Carrying Code

Το σύστημα *Proof-Carrying Code* (PCC), που προτάθηκε από τον Necula, είναι ένα γενικό πλαίσιο για το χειρισμό της ακεραιότητας του συστήματος και προβλημάτων ασφάλειας, χρησιμοποιώντας τεχνικές από τη μαθηματική λογική και τη σημασιολογία των γλωσσών προγραμματισμού [26, 27, 28]. Μια τυπική χρήση του εκτελέσιμου κώδικα στο πλαίσιο PCC φαίνεται στο σχήμα 3.1.

Αρχικά, τα δύο εμπλεκόμενα μέρη, ο παραγωγός και ο καταναλωτής κώδικα πρέπει να συμφωνήσουν πάνω στην πολιτική ασφάλειας, γύρω από την οποία εστιάζεται όλη η διαδικασία. Η πολιτική ασφάλειας, εκφρασμένη σε μια κατάλληλη τυπική λογική, είναι ένα σύνολο από κανόνες ασφάλειας και απαιτήσεις ορθότητας οι οποίες εξασφαλίζουν την ασφαλή συμπεριφορά του εκτελέσιμου κώδικα. Από την πλευρά του παραγωγού, ο πηγαίος κώδικας περνά από ένα μεταγλωττιστή ο οποίος πιστοποιεί τον κώδικα, εμπλουτίζοντας τον με κατάλληλη πληροφορία. Αυτή η πληροφορία περιέχει προδιαγραφές συναρτήσεων και αναλλοίωτες βρόχων, οι οποίες βοηθούν ένα αυτόματο εργαλείο για απόδειξη θεωρημάτων να κατασκευάσει μια τυπική απόδειξη ότι ο κώδικας είναι συνεπής με την πολιτική ασφάλειας. Ο πιστοποιημένος κώδικας στο πλαίσιο PCC είναι ένα πακέτο, που περιέχει τον κώδικα μαζί με μια αναπαράσταση αυτής της απόδειξης. Από την πλευρά του καταναλωτή, η ασφάλεια της εκτέλεσης του πιστοποιημένου εκτελέσιμου εξασφαλίζεται ελέγχοντας την επισυναπτόμενη απόδειξη. Εν συντομία, το πλαίσιο PCC απαιτεί μια σχετικά απλή υποδομή για αξιόπιστη μεταφορά και δεν επιφέρει ποινές σε χρόνο εκτέλεσης εξαιτίας ελέγχων ασφαλείας.

Το πλαίσιο PCC χρησιμοποιεί μια γενικής χρήσης πρώτης τάξης κατηγορηματική λογική και μπορεί να εκφράσει σύνθετες ιδιότητες. Από την άλλη πλευρά, το PCC χρησιμοποιεί ειδικές αποδείξεις ασφάλειας που δεν μπορούν στη γενική περίπτωση να παραχθούν αυτόματα. Η λογική στην οποία εκφράζονται οι αποδείξεις αυτές περιέχει μια εγγενή γνώση για την υποκείμενη γλώσσα προγραμματισμού και το σύστημα τύπων της. Αυτό σημαίνει ότι, για να εγγυηθεί κανείς την ασφάλεια, όχι μόνο δεν πρέπει να υπάρχουν σφάλματα στην πολιτική ασφάλειας, τον ελεγκτή τύπων και το μεταφραστή σε γλώσσα μηχανής, αν απαιτείται, αλλά πρέπει επίσης να μην υπάρχουν σφάλματα στους κανόνες τύπων και τα λογικά αξιώματα που σχετίζονται με τους κατασκευαστές τύπων. Ευτυχώς όλα αυτά ειναι μια τάξη μεγέθους πιο



Σχήμα 3.1: Διάρθρωση του πλαισίου PCC.

απλά από την υλοποίηση της JVM ή του CLR και η ορθότητά τους μπορεί να επιβεβαιωθεί σε μια μέτα-θεωρία.

3.1.2 Foundational Proof-Carrying Code

Το σύστημα *Foundational Proof-Carrying Code* (FPCC) που προτάθηκε από τους Appel και Felty, στοχεύει να ξεπεράσει κάποια από τα προαναφερθέντα προβλήματα με την ελαχιστοποίηση του μεγέθους των δομικών μονάδων που πρέπει να εμπιστευτεί κανείς τυφλά (ή να επιβεβαιώσει σε μια μετα-θεωρία) σε ένα σύστημα PCC [7, 8]. Το FPCC χρησιμοποιεί μια γενικής χρήσης κατηγορηματική λογική υψηλής τάξης, και μερικά αξιώματα από την αριθμητική, τα οποία μπορούν να χρησιμοποιηθούν σαν θεμέλια των σύγχρονων μαθηματικών. Χρησιμοποιώντας αυτή τη λογική, είναι δυνατό να ορίσουμε την πολιτική ασφαλείας, καθώς και το σύστημα τύπων και τη σημασιολογία της υποκείμενης γλώσσας προγραμματισμού. Στο πλαίσιο FPCC, ο πιστοποιημένος κώδικας είναι πάλι ένα πακέτο που περιέχει τον προς εκτέλεση κώδικα μαζί με μια αναπαράσταση μιας απόδειξης ασφάλειας. Αυτές οι αποδείξεις είναι *θεμελιώδεις* (foundational), δηλαδή πρέπει ρητά να ορίζουν, μέχρι τα θεμέλια των μαθηματικών, όλες τις απαιτούμενες έννοιες και πρέπει ρητά να αποδεικνύουν όλες τις απαιτούμενες ιδιότητες αυτών των εννοιών.

Σε σύγκριση με το PCC, το FPCC είναι περισσότερο ευέλικτο γιατί δεν περιορίζεται σε μια συγκεκριμένη υποκείμενη γλώσσα προγραμματισμού, ούτε σε ένα δοσμένο υποσύνολο της λογικής ή των μαθηματικών. Οι θεμελιώδεις αποδείξεις μπορεί να περιέχουν τον ορισμό ενός νέου συστήματος τύπων για τη συγκεκριμένη γλώσσα και νέα θεωρήματα για να αποδεικνύουν την πολιτική ασφαλείας. Επιπλέον, το πλαίσιο FPCC είναι περισσότερο ασφαλές καθώς μειώνει περαιτέρω το μέγεθος των εμπιστευόμενων μερών. Για να εγγυηθεί κανείς την ασφάλεια, δεν πρέπει να υπάρχουν σφάλματα στην πολιτική ασφαλείας, στον ορισμό της target γλώσσας

και στον foundational ελεγκτή αποδείξεων, ο οποίος μπορεί να γίνει αρκετά μικρός. Παρ' όλα αυτά, σαν συνέπεια αυτών των πλεονεκτημάτων, το πρόβλημα της αυτόματης κατασκευής foundational αποδείξεων ασφάλειας γίνεται ακόμη πιο δύσκολο.

3.1.3 Η τυποθεωρητική προσέγγιση

Μια προσπάθεια να συνδυαστούν τα πλεονεκτήματα των γλωσσών με ισχυρά συστήματα τύπων (βλ. παρ. 3.2) για την αυτόματη κατασκευή πιστοποιημένου κώδικα με την απλότητα και την εκφραστικότητα του συστήματος FPCC έγινε από τον Shao *et al.* σαν μέρος του έργου FLINT στο πανεπιστήμιο Yale [32]. Η δουλειά τους προτείνει ένα τυποθεωρητικό πλαίσιο για κατασκευή, σύνθεση και reasoning σε πιστοποιημένα εκτελέσιμα, βασισμένο στην αρχή των "προτάσεων ως τύπων" [23]. Οι προτάσεις και οι αποδείξεις εκφράζονται σε ένα γενικό και ισχυρό σύστημα τύπων που εμπεριέχει κατηγορηματική λογική υψηλής τάξης. Αυτό το σύστημα τύπων χρησιμοποιείται επίσης για να ορίσει τους τύπους μιας ή περισσοτέρων υποκείμενων γλωσσών προγραμματισμού, ώστε να παράσχει ένα σύνδεσμο μεταξύ των προγραμμάτων και των ιδιοτήτων τους όπως αυτές θα εκφράζονταν σε μια τυπική λογική. Έτσι επιτυγχάνεται η ενοποίηση των αποδείξεων και των προγραμμάτων με έναν κομψό τρόπο και το τυπικό reasoning που παραδοσιακά γινόταν σε μια μετα-θεωρία, μπορεί τώρα να εκφραστεί μέσα στην ίδια τη γλώσσα. Μια προσέγγιση πολύ παρόμοια στο πνεύμα, αλλά που διαφέρει στις λεπτομέρειες της λογικής προτάθηκε ανεξάρτητα από τους Crary και Vanderwaart [19]. Ο κοινός παρονομαστής ανάμεσα στις δύο δουλειές, δηλαδή η ενσωμάτωση μιας λογικής κατάλληλης να εκφράσει ιδιότητες στο σύστημα τύπων της γλώσσας, θα αναφέρεται ως η τυποθεωρητική προσέγγιση.

Η τυποθεωρητική προσέγγιση χτίζει πάνω σε ένα μεγάλο όγκο δουλειάς στην περιοχή της λογικής και της θεωρίας τύπων και επιχειρεί να ενσωματώσει αυτή τη δουλειά σε εμπορικής εμβέλειας μεταγλωττιστές για γενικής χρήσης γλώσσες προγραμματισμού. Παρ' όλο που αρκετά και δύσκολα προβλήματα, θεωρητικά και πρακτικά, πρέπει να λυθούν πριν πραγματικοί προγραμματιστές μπορέσουν να χρησιμοποιήσουν τα αποτελέσματα αυτής της προσπάθειας, είναι πιθανό αυτή η προσέγγιση να αποτελέσει ένα σημαντικό βήμα προς την επίλυση ενός προβλήματος πιο γενικού από αυτό της ασφάλειας: Παραγωγή και εκτέλεση αποδοτικού εκτελέσιμου κώδικα που αποδεικνύεται ότι ικανοποιεί τυχαίες επιθυμητές ιδιότητες.

3.2 Ενδιάμεσες και τελικές γλώσσες με τύπους

Η *ενδιάμεση γλώσσα με τύπους* (Typed Intermediate Language, TIL) και η *τελική γλώσσα με τύπους* (Typed Assembly Language, TAL) (βλ. [22, 25]) ήταν δύο από τις πρώτες γενικές απόπειρες να επεκταθούν οι ενδιάμεσες ή τελικές γλώσσες με ισχυρά συστήματα τύπων, τα οποία θα μπορούσαν να χρησιμοποιηθούν για επαλήθευση του εκτελέσιμου κώδικα. Παρ' όλο που τα πλεονεκτήματα από τη χρήση μιας ενδιάμεσης ή τελικής γλώσσας με ισχυρό σύστημα τύπων σε ένα μεταγλωττιστή δεν περιορίζονται μόνο σε θέματα ασφάλειας, οι τύποι που ανατίθενται σε προγράμματα γραμμένα σε αυτές τις γλώσσες μπορούν να χρησιμοποιηθούν για να εγγυηθεί κανείς ότι κάποιες, συνήθως απλές ιδιότητες ασφάλειας ισχύουν κατά τη διάρκεια της εκτέλεσης του κώδικα. Για να γίνει αυτό δυνατό, ένας κατάλληλος ελεγκτής τύπων πρέπει να επιβεβαιώνει την ορθότητα των τύπων του προς εκτέλεση κώδικα πρίν την εκτέλεσή του. Επιπλέον, επειδή η πολιτική ασφαλείας είναι ενσωματωμένη στο σύστημα τύπων της γλώσσας χαμηλού επιπέδου, οι αποδείξεις για την ασφάλεια μπορούν να κατασκευαστούν αυτόματα από το μεταγλωττιστή. Παρακάτω περιγράφεται η χρήση μιας ενδιάμεσης γλώσσας με τύπους για την αυτόματη και ασφαλή δέσμευση/αποδέσμευση μνήμης.

3.2.1 Διαχείριση μνήμης βασισμένη σε περιοχές – Capability Language

Η βασισμένη σε περιοχές διαχείριση μνήμης (region-based memory management) είναι μία, εναλλακτική της συλλογής σκουπιδιών, τεχνική αυτόματης διαχείρισης της μνήμης του σωρού

από ένα πρόγραμμα. Η Capability Language (CL), που προτάθηκε από τον D. Walker *et al.* στο [34], είναι μια TIL γλώσσα που χρησιμοποιεί αυτή την τεχνική. Στόχος της είναι η αντιμετώπιση του εξής προβλήματος:

> Επειδή τα πλαίσια TAL και PCC δεν ασχολούνται με τη διαχείριση μνήμης, η ορθότητά τους βασίζεται σε κάποιον έμπιστο συλλέκτη σκουπιδιών. Οι συλλέκτες σκουπιδιών όμως είναι συνήθως μεγάλες, μη-πιστοποιημένες εφαρμογές και έτσι ο βαθμός ασφάλειας όλου του συστήματος μειώνεται.

Η CL καταφέρνει να ενσωματώσει τη διαχείριση της μνήμης του σωρού στο σύστημα τύπων της με αποτέλεσμα τα προγράμματα που γράφονται σε CL να έχουν έλεγχο στη διαχείριση της μνήμης και επιπλέον να είναι πιστοποιημένα ασφαλή. Επιπλέον, τα προγράμματα δεν πάσχουν από τις απώλειες απόδοσης λόγω των διακοπών στη ροή εκτέλεσης που προκαλεί ένας συλλέκτης σκουπιδιών.

Στο σύστημα αυτό, η μνήμη χωρίζεται σε τμήματα που ονομάζονται *περιοχές*. Κάθε δεδομένο που απαιτεί δυναμική δέσμευση μνήμης αποθηκεύεται σε μία από τις ενεργές περιοχές και η διάρκεια ζωής του είναι το πολύ όση και η διάρκεια ζωής της περιοχής. Κάθε περιοχή αντιπροσωπεύεται απο ένα capability, που δηλώνει το δικαίωμα πρόσβασης σε αυτή την περιοχή. Έτσι, για να χρησιμοποιηθεί ένα δεδομένο της περιοχής `r` σε κάποιο σημείο της ροής ελέγχου του προγράμματος, το πρόγραμμα πρέπει να παρουσιάσει το capability της `r`.

Η μνήμη θεωρείται μια πεπερασμένη αντιστοίχιση από ονόματα περιοχών (ν) σε περιοχές, όπου περιοχή είναι ένα block μνήμης όπου αποθηκεύονται δεδομένα. Οι περιοχές δημιουργούνται κατά το χρόνο εκτέλεσης με τη δήλωση `newrgn` ρ, x η οποία προσθέτει στη μνήμη μια νέα περιοχή (ν), δεσμεύει το ρ σαν όνομα αυτής της περιοχής και το x σαν το χειριστή (handle) αυτής της περιοχής. Η χρησιμοποίηση ονομάτων και χειριστών για τις περιοχές δεν είναι πλεονασμός, μιας και τα ονόματα αποτελούν εκφράσεις που χρειάζονται μόνο κατά το στάδιο μεταγλώττισης ενώ οι χειριστές χρειάζονται και κατά το χρόνο εκτέλεσης. Η περιοχή αποδεσμεύεται με την εντολή `freergn` x, όπου x ο χειριστής της περιοχής.

Οι περιοχές στη CL αντιμετωπίζονται ως τύποι, και έτσι ο προγραμματιστής μπορεί να ορίσει μεταβλητές τύπου περιοχής. Επιτρέπεται δύο μεταβλητές περιοχής, π.χ. r_1, r_2 να αρχικοποιηθούν με την ίδια περιοχή. Για να αποδεσμευτεί όμως η περιοχή r_1 με ασφάλεια, πρέπει να ξέρουμε ότι καμία άλλη μεταβλητή περιοχής δεν δείχνει στην ίδια περιοχή με τη r_1. Γι' αυτό η CL παρέχει δύο τελεστές: το r_1^1 εννοεί ότι η περιοχή είναι μονάδικη ενώ το r_1^+ εννοεί ότι μπορεί να υπάρχει συνωνυμία (aliasing), δηλαδή να υπάρχει και άλλη μεταβλητή που να δείχνει στην ίδια περιοχή, π.χ. η r_2. Έτσι, μια περιοχή μπορεί να αποδεσμευτεί μόνο αν είναι της μορφής r^1. Πλήρης περιγραφή του συστήματος τύπων και της σημασιολογίας της CL υπάρχει στο [34]. Ο αναγνώστης που δεν είναι εξοικειωμένος με τα συστήματα τύπων και τον λ-λογισμό μπορεί να ενημερωθεί από τα [37, 13, 16, 17].

3.3 Επόπτες

Επόπτης (monitor) λέγεται ένα πρόγραμμα που τρέχει παράλληλα με μια ύποπτη εφαρμογή και παρακολουθεί τη ροή εκτέλεσής της. Αν αυτή ξεφύγει από την επιθυμητή πολιτική ασφαλείας ο επόπτης ή την τροποποιεί ή τερματίζει την εφαρμογή. Η λειτουργία ενός επόπτη περιγράφεται σε γενικές γραμμές από τον παρακάτω μηχανισμό:
Όταν η ύποπτη εφαρμογή πρόκειται να εκτελέσει μια πιθανώς επικίνδυνη κλήση συνάρτησης, η ροή ελέγχου του προγράμματος περνάει στον επόπτη ο οποίος αποφασίζει τί από τα παρακάτω θα γίνει.

i. Επιτρέπει στην εφαρμογή να καλέσει τη συνάρτηση.

ii. Τερματίζει την εφαρμογή.

iii. Δεν της επιτρέπει να καλέσει τη συνάρτηση, αλλά την αφήνει να συνεχίσει.

iv. Ο ίδιος ο επόπτης εκτελεί κάποιους υπολογισμούς εκ μέρους της εφαρμογής και έπειτα διαλέγει μια από τις παραπάνω επιλογές.

3.3.1 Η γλώσσα Polymer

Προηγούμενη έρευνα έχει δείξει ότι οι επόπτες αποτελούν ένα πιο γενικό μηχανισμό από τις Λίστες Ελεγχόμενης Πρόσβασης (σελ. 25) και από τον έλεγχο της στοίβας (stack inspection). Παρ' όλα αυτά, υπάρχει έλλειψη ακριβούς σημασιολογίας στον τομέα αυτό. Οι L. Bauer, J. Ligatti και D. Walker κάλυψαν αυτό το κενό σχεδιάζοντας μια γλώσσα για επόπτες, την Polymer, η οποία περιγράφεται παρακάτω. Το πλήρες σύστημα τύπων και η σημασιολογία της υπάρχει στο [14].

Απλές Πολιτικές

Πολιτική καλείται το ζευγάρι {`actions: A; whatToDo: E`}, όπου A ένα σύνολο ενεργειών της ύποπτης εφαρμογής και E ένα σύνολο ενεργειών του επόπτη. Το A ονομάζεται *διαχειριζόμενο σύνολο*. Όταν η εφαρμογή θέλει να εκτελέσει μια ενέργεια που ανήκει στο διαχειριζόμενο σύνολο, ο επόπτης παρεμβαίνει και εκτελεί το E για να διατηρηθεί η επιθυμητή πολιτική ασφαλείας του συστήματος.

Για παράδειγμα, μια πολιτική που επιτρέπει στις εφαρμογές να δεσμεύσουν μνήμη μέχρι κάποιο ανώτατο όριο, έστω `q0`, μπορεί να γραφτεί ως εξής:

```
fun mpol(q:int).
{
  actions: malloc();
  policy:
    next ->
      case * of
        malloc(n) ->
          if ((q-n) > 0) then
            ok; run (mpol (q-n))
          else
            halt
      end
    done -> ()
}
```

Η δομή (next $\rightarrow E_1$ — done $\rightarrow E_2$) αναστέλλει τη λειτουργία του επόπτη μέχρι να συναντήσει κάποια εντολή malloc(). Τότε ο επόπτης κοιτάζει αν η εφαρμογή, δεσμεύοντας την επιπλέον μνήμη, ξεπεράσει την `q0`. Αν ναι, τότε διακόπτει την εφαρμογή, αν όχι την αφήνει να συνεχίσει. Η mpol είναι αναδρομική. Μετά το `ok;` καλεί τον εαυτό της και "παγώνει" στο επόμενο `next` μέχρι την επόμενη εντολή του διαχειριζόμενου συνόλου (εδώ μόνο η malloc() ανήκει στο A). Αν η εφαρμογή τερματίσει χωρίς να βρεθεί ξανά εντολή που να ανήκει στο A, τότε εκτελείται η E_2 για να καθαρίσει πιθανά ίχνη που άφησε η εφαρμογή. Έδω αυτό δεν είναι απαραίτητο, οπότε η E_2 επιστρέφει unit. Η πολιτική καλείται αρχικά ως εξής: `memlimit = mpol q0`. Κάθε πολιτική έχει τύπο $\mathcal{M}(\tau)$ όπου τ είναι ο τύπος που επιστρέφει η E_2. Εδώ, η `memlimit` έχει τύπο $\mathcal{M}(unit)$.

Συνδυάζοντας Πολιτικές

Οι πολιτικές στην Polymer είναι αντικείμενα πρώτης τάξης. Αυτό σημαίνει ότι μια συνάρτηση μπορεί να είναι πολυμορφική ως προς τις πολιτικές και ότι μια πολιτική μπορεί να είναι φωλιασμένη μέσα σε μια άλλη πολιτική. Επιπλέον, υπάρχουν τελεστές που επιτρέπουν να συνδυάζουμε πολιτικές για να φτιάχνουμε πιο σύνθετες πολιτικές. Ας υποθέσουμε ότι υπάρχει

μια πολιτική, η `fileAccess` που ελέγχει την πρόσβαση των εφαρμογών στα τοπικά αρχεία. Επιτρέπεται να τρέχει παράλληλα με τη `memlimit` ώστε να έχουμε μια πολιτική που ελέγχει το όριο της μνήμης και την πρόσβαση στα αρχεία:

`RM = fileAccess` $\wedge$ `memlimit`

Η malloc() αγνοείται από την `fileAccess` ενώ ενέργειες όπως η fopen() αγνοούνται από τη `memlimit`. Αν μια από τις δύο πολιτικές αποφασίσει να σταματήσει την εφαρμογή, τότε τη σταματάει χωρίς την άδεια της άλλης. Ο τυπός της συνολικής πολιτικής είναι $\mathcal{M}(\tau_1 \times \tau_2)$ δηλαδή εδώ είναι $\mathcal{M}(unit \times unit)$. Ορίζεται και η ουδέτερη πολιτική $\top$ για τον τελεστή $\wedge$. Έτσι, η πολιτική $M \wedge \top$ είναι η ίδια με τη M. Με την ίδια λογική, ο τελεστής $\vee_\tau$ αντιπροσωπεύει τη διάζευξη δύο πολιτικών που τρέχουν παράλληλα. Δηλαδή μια εφαρμογή γίνεται δεκτή όταν τη δέχεται έστω η μία από τις δύο πολιτικές, ενώ για να σταματήσει να εκτελείται πρέπει να συμφωνήσουν και οι δύο πολιτικές. Ο τύπος της συνολικής πολιτικής είναι $\mathcal{M}(\tau_1 + \tau_2)$.

Συνθέτοντας Πολιτικές

Καμιά φορά ο συνδυασμός δύο πολιτικών μπορεί να οδηγήσει σε άλλα αποτελέσματα από τα επιθυμητά. Για παράδειγμα, έστω η `liberalFilePolicy` που επιτρέπει στις εφαρμογές πρόσβαση σε αρκετά αρχεία, ενώ η `stricterFilePolicy` είναι μια πολιτική που απαγορεύει την πρόσβαση στο σύστημα αρχείων. Η σύζευξη αυτών των πολιτικών θα έχει απρόβλεπτα αποτελέσματα, αφού οι ενέργειες που απαγορεύονται από τη μία επιτρέπονται (και πιθανώς απαιτούνται) από την άλλη. Γι' αυτές τις περιπτώσεις η Polymer παρέχει ένα σύστημα τύπων που θέτει περιορισμούς για το ποιες πολιτικές μπορούν να συνδυάζονται και επιπλέον παρέχει τελεστές για την ακολουθιακή εκτέλεση δύο πολιτικών αντί της παράλληλης όπου χρειάζεται.

Η Polymer χρησιμοποιεί ένα σύστημα τύπων και αποτελεσμάτων (type and effect system). Κάθε πολιτική έχει τύπο $\mathcal{M}^{A_r}_{A_c}$ όπου A_r είναι το διαχειριζόμενο σύνολό της και A_c είναι τα αποτελέσματα που επιστρέφει, δηλαδή οι ενέργειες που η πολιτική απαγορεύει ή τις εκτελεί η ίδια εκ μέρους της εφαρμογής. Έτσι, για τους τελεστές που ορίσαμε παραπάνω έχουμε τους κανόνες τύπων:

$$\frac{\Gamma \vdash M_1 : \mathcal{M}^{A_1}_{A_2}(\tau_1) \qquad \Gamma \vdash M_2 : \mathcal{M}^{A_3}_{A_4}(\tau_2) \qquad A_1 \cap A_4 = A_2 \cap A_3 = \emptyset}{\Gamma \vdash M_1 \wedge M_2 : \mathcal{M}^{A_1 \cup A_3}_{A_2 \cup A_4}(\tau_1 \times \tau_2)}$$

$$\frac{\Gamma \vdash M_1 : \mathcal{M}^{A_1}_{A_2}(\tau_1) \qquad \Gamma \vdash M_2 : \mathcal{M}^{A_3}_{A_4}(\tau_2) \qquad A_1 \cap A_4 = A_2 \cap A_3 = \emptyset}{\Gamma \vdash M_1 \vee_{\tau_1+\tau_2} M_2 : \mathcal{M}^{A_1 \cup A_3}_{A_2 \cup A_4}(\tau_1 + \tau_2)}$$

Παρατηρούμε ότι για να συνδυάζονται δύο πολιτικές πρέπει το διαχειριζόμενο σύνολο της μίας να μην έχει κοινές ενέργειες με τα αποτελέσματα της άλλης. Το διαχειριζόμενο σύνολο της συνολικής πολιτικής είναι η ένωση των διαχειριζόμενων συνόλων των επιμέρους πολιτικών. Το ίδιο ισχύει και για τα αποτελέσματα της συνολικής πολιτικής.

Επίσης, ορίζονται δύο τελεστές για ακολουθιακή εκτέλεση πολιτικών. Η συζευκτική ακολουθιακή πολιτική $M_1 \triangle M_2$ λειτουργεί ως εξής: Η M_1 εφαρμόζεται στην εφαρμογή-στόχο και έχει το δικαίωμα να απαγορεύσει ενέργειες ή να προσθέσει δικές της ενέργειες στο στόχο. Έτσι δημιουργεί ένα ρεύμα εξόδου στο οποίο εφαρμόζεται η M_2 και ενεργεί όπως θα ενεργούσε κανονικά αν εφαρμοζόταν κατευθείαν στο στόχο. Για να τερματιστεί ο στόχος αρκεί να το αποφασίσει μια εκ των δύο πολιτικών. Η διαζευκτική ακολουθιακή πολιτική $M_1 \nabla_\tau M_2$ είναι παρόμοια, με τη διαφορά ότι τώρα πρέπει να συμφωνήσουν και οι δύο για να τερματιστεί ο στόχος. Αφού οι πολιτικές εκτελούνται ακολουθιακά, είναι στην ευθύνη του προγραμματιστή να αποφασίσει πώς θα επιλύσει τις συγκρούσεις. Έτσι, οι κανόνες τύπων για τους δύο αυτούς τελεστές είναι λιγότερο περιοριστικοί από τους προηγούμενους.

$$\frac{\Gamma \vdash M_1 : \mathcal{M}^{A_1}_{A_2}(\tau_1) \quad \Gamma \vdash M_2 : \mathcal{M}^{A_3}_{A_4}(\tau_2)}{\Gamma \vdash M_1 \vartriangle M_2 : \mathcal{M}^{A_1 \cup A_3}_{A_2 \cup A_4}(\tau_1 \times \tau_2)}$$

$$\frac{\Gamma \vdash M_1 : \mathcal{M}^{A_1}_{A_2}(\tau_1) \quad \Gamma \vdash M_2 : \mathcal{M}^{A_3}_{A_4}(\tau_2)}{\Gamma \vdash M_1 \triangledown_{\tau_1+\tau_2} M_2 : \mathcal{M}^{A_1 \cup A_3}_{A_2 \cup A_4}(\tau_1 + \tau_2)}$$

Κάθε πολιτική που σχηματίζεται με τους παράλληλους τελεστές σχηματίζεται και με τους ακολουθιακούς, αφού δέχονται περισσότερες πολιτικές. Από την άλλη όμως, οι παράλληλοι τελεστές εξασφαλίζουν την (συχνά επιθυμητή) ιδιότητα της μη-παρεμβολής μεταξύ των πολιτικών.

3.4 Συστήματα τύπων με ροή πληροφορίας – Η γλώσσα $\lambda_{\mathbf{RP}}$

Τα *συστήματα τύπων με ροή πληροφορίας* (information-flow type systems [31]) είναι ένας καλός τρόπος για να ενσωματωθούν πολιτικές ασφαλείας μέσα σε μια γλώσσα. Οι τύποι σε αυτά τα συστήματα έχουν ετικέτες (labels), οι οποίες χρησιμοποιούνται για να εκφράσουν ιδιότητες ασφάλειας των δεδομένων. Συνήθως, οι ερευνητές που χρησιμοποίησαν αυτά τα συστήματα απαιτούσαν να καθορίζεται η διαθεσιμότητα κάθε δεδομένου (δημόσιο ή ιδιωτικό) κατά τη διάρκεια της ανάπτυξης ενός προγράμματος. Στην πράξη όμως αυτό δεν είναι πάντα δυνατό και έτσι περιορίζονται οι πολιτικές που μπορούν να εκφραστούν. Γι' αυτό το λόγο, οι S. Tse και S. Zdancewic δημιούργησαν τη γλώσσα λ_{RP} που χρησιμοποιεί μεν σύστημα τύπων με ροή πληροφορίας αλλά έχει και ένα μηχανισμό, τα run-time principals για να εκφράζει δυναμικά ιδιότητες ασφάλειας στο πρόγραμμα. Τα principals είναι τιμές πρώτης τάξης στην λ_{RP} και αντιπροσωπεύουν χρήστες, ομάδες χρηστών και γενικά οντότητες που αλληλεπιδρούν με το περιβάλλον εκτέλεσης του προγράμματος. Η σύνταξη και οι κανόνες τύπων που θα παρουσιάσουμε εδώ είναι το υποσύνολο της γλώσσας που χρησιμοποιήθηκε στην υλοποίηση της παρούσας εργασίας. Η πλήρης γλώσσα περιγράφεται στο [33].

3.4.1 Ετικέτες και Principals

Οι πολιτικές στην λ_{RP} εκφράζονται με σύνολα από principals. Θα χρησιμοποιούμε αναγνωριστικά με κεφαλαίο γράμμα για τα ονόματα των principals όπως Alice, Bob κ.λπ. και τη μετα-μεταβλητή X για αυτά τα ονόματα. Τα σύνολα s από principals είναι μη-διατεταγμένες λίστες από principals που χωρίζονται με κόμμα. Το άδειο σύνολο συμβολίζεται με '$\cdot$' και συχνά θα παραλείπεται. Έτσι έχουμε:

$$p ::= X \mid \alpha \qquad\qquad s ::= \cdot \mid p, s$$

όπου α είναι μια principal μεταβλητή επειδή η ταυτότητα ενός principal μπορεί να μην είναι γνωστή στατικά. Κάθε ετικέτα l αποτελείται από *τμήματα* c της μορφής $p : s$ όπου p είναι ο ιδιοκτήτης ενός δεδομένου και s είναι τα principals που τους επιτρέπει το p να διαβάσουν το δεδομένο. Τα principals που τελικά επιτρέπεται να διαβάσουν το δεδομένο βρίσκονται από την τομή των τμημάτων, δηλαδή ένα δεδομένο με ετικέτα $Alice : Bob, Charles$ και $Bob : Charles, Eve$ θα διαβάζεται μόνο από τους Bob και Charles.

$$c ::= p : s \qquad\qquad d ::= \cdot \mid c; d \qquad\qquad l ::= \{d\}$$

Ιεραρχία δράσης (acts-for hierarchy) στην λ_{RP} ονομάζεται ένα σύνολο από περιορισμούς της μορφής $p \preceq q$. Ο συμβολισμός $p \preceq q$ εννοεί ότι το q δρα για το p, ή αντίστοιχα ότι το p αντιπροσωπεύεται από το q. Όταν ισχύει αυτό τότε το q μπορεί να διαβάσει ότι διαβάζει το p. Η σχέση $\preceq$ είναι ανακλαστική και μεταβατική και σχηματίζεται έτσι μια μερική διάταξη

στα principals. Αν μια ιεραρχία δράσης δεν περιέχει principal μεταβλητές είναι κλειστή και συμβολίζεται με $\mathcal{A}$, αλλιώς συμβολίζεται με Δ. Γράφουμε $\Delta \vdash p \preceq q$ όταν το ανακλαστικό μεταβατικό κλείσιμο της Δ περιέχει το $p \preceq q$. Η σχέση γενικεύεται και σε σύνολα από principals, $\Delta \vdash s_1 \preceq s_2$. Το σύνολο s_2 δρα για το s_1 όταν γιά κάθε $p \in s_1$ υπάρχει ένα $q \in s_2$ ώστε $p \preceq q$. Τέλος, ορίζεται το ισχυρότερο principal $\top$ που έχει την ιδιότητα $\Delta \vdash p \preceq \top$ για κάθε ιεραρχία Δ και κάθε principal p.

Η ένωση ετικετών στην λ_{RP} ορίζεται από τη σχέση $\{d_1\} \sqcup \{d_2\} \stackrel{\mathrm{def}}{=} \{d_1 \cup d_2\}$. Μια ετικέτα $l_1 = \{d_1\}$ είναι λιγότερο περιοριστική από μια ετικέτα $l_2 = \{d_2\}$ όταν ισχύουν

$$\frac{\Delta \vdash p_1 \preceq p_2 \qquad \forall p_2' \in s_2.\, \exists p_1' \in s_1.\, \Delta \vdash p_1' \preceq p_2'}{\Delta \vdash p_1 : s_1 \sqsubseteq p_2 : s_2}$$

$$\frac{\forall c_1 \in d_1.\, \exists c_2 \in d_2.\, \Delta \vdash c_1 \sqsubseteq c_2}{\Delta \vdash \{d_1\} \sqsubseteq \{d_2\}}$$

και γράφουμε $\Delta \vdash l_1 \sqsubseteq l_2$. Όταν αυτό δεν ισχύει γράφουμε $\Delta \vdash l_1 \not\sqsubseteq l_2$. Παρατηρούμε ότι στη διάταξη που σχηματίζουν οι ετικέτες λόγω της σχέσης $\sqsubseteq$, ισχύει πάντα ότι $\Delta \vdash l \sqsubseteq \{\top\}$ και ότι $\Delta \vdash \{\cdot\} \sqsubseteq l$ για κάθε ετικέτα l και κάθε ιεραρχία Δ.

3.4.2 Η σύνταξη και το σύστημα τύπων της λ_{RP}

Η λ_{RP} είναι μια παραλλαγή του λ-λογισμού με τύπους, με πολιτικές ασφαλείας που εκφράζονται από τη διάταξη των ετικετών που περιγράψαμε παραπάνω. Όλα τα προγράμματα της γλώσσας τερματίζουν. Οι τύποι στην λ_{RP} ονομάζονται τύποι ασφαλείας t και είναι βασικοί τύποι (base types) u που επιπλέον έχουν μια ετικέτα l, δηλαδή $t \stackrel{\mathrm{def}}{=} u_l$. Ο τύπος μονάδας (unit) συμβολίζεται με `1` και η μόνη τιμή για μεταβλητές αυτού του τύπου είναι το $*$. Αν παραλειφθεί η ετικέτα ενός βασικού τύπου υπονοείται ότι έχει τη χαμηλότερη ετικέτα, $\{\cdot\}$. Ένα principal p έχει μοναδικό τύπο[2] P_p. Αυτό συνεπάγεται ότι η μόνη τιμή του τύπου P_{Alice} είναι η σταθερά *Alice*. Οι μοναδικοί τύποι έχουν ξαναχρησιμοποιηθεί στο παρελθόν για να αναπαραστήσουν δυναμική πληροφορία [18]. Η λ_{RP} υποστηρίζει επίσης καθολικούς $\forall \alpha \preceq p.\, t$ και υπαρξιακούς τύπους $\exists \alpha \preceq p.\, t$. Για παράδειγμα, ο τύπος $t_0 = \forall \alpha \preceq Alice.\, \mathrm{bool}_{\{\alpha\}}$ περιγράφει boolean μεταβλητές που ανήκουν σε κάποιο principal για το οποίο δρα η *Alice*. Ένα πάνω όριο $\top$ σε ένα τύπο θα παραλείπεται γιατί όπως είδαμε για κάθε principal ισχύει $p \preceq \top$. Έτσι, για συντομία θα γράφουμε: $\forall \alpha.\, t \stackrel{\mathrm{def}}{=} \forall \alpha \preceq \top.\, t$ και $\Lambda \alpha.\, e \stackrel{\mathrm{def}}{=} \Lambda \alpha \preceq \top.\, e$. Το πρόγραμμα μπορεί να κάνει έλεγχο στη σχέση αντιπροσώπευσης μεταξύ *Alice* και *Bob* κατά το χρόνο εκτέλεσης χρησιμοποιώντας την έκφραση `if` $(Alice \preceq Bob)\ e_1\ e_2$. Η σύνταξη της λ_{RP} φαίνεται στον πίνακα 3.1.

3.4.3 Δικαιώματα και Capabilities

Η λ_{RP} ορίζει δύο *δικαιώματα* (privileges), τα declassify και delegate που επιτρέπουν την υπό συνθήκες χαλάρωση της πολιτικής ασφαλείας. Η έκφραση `declassify` $e\ t$ μετατρέπει τον τύπο της e στον τύπο t, το οποίο μπορεί να χαλαρώσει κάποιες ετικέτες που υπάρχουν στην e. Για παράδειγμα, δείτε την έκφραση `declassify` $(int_{Alice:})$ $(int_{Alice:Bob})$. Η έκφραση αυτή παίρνει έναν ακέραιο που τον διαβάζει μόνο η *Alice* και προσθέτει και τον *Bob* στα principals που μπορούν να τον διαβάσουν. Για να γίνονται τα declassifications χωρίς να παραβιάζεται η ασφάλεια της γλώσσας, η λ_{RP} απαιτεί να υπάρχει η έγκριση του ιδιοκτήτη του δεδομένου (εδώ της Alice) για να χαλαρώσει η πολιτική ασφαλείας. Το delegate επιτρέπει να αλλάξει η ιεραρχία δράσης κατά το χρόνο εκτέλεσης. Με άλλα λόγια, αν ισχύει η έκφραση `let`$(p_1 \preceq p_2)$`in` e τότε, εντός της εμβέλειας της e, το p_2 μπορεί να δράσει για το p_1. Φυσικά για να επιτραπεί αυτό χρειάζεται η έγκριση του p_1 (ή κάποιου principal που δρα για το p_1).

[2] Να μη συγχέεται με τον τύπο μονάδας. Οι μοναδικοί τύποι (singleton types) προτάθηκαν από τον D. Aspinall στο [9].

Πίνακας 3.1: Σύνταξη της λ_{RP}.

$t ::= u_l$	`Secure types`
$u ::=$	`Base types`
`1`	unit
t+t	sum
$t \to t$	function
P_p	principal
$\forall\alpha \preceq p.\, t$	universal
$\exists\alpha \preceq p.\, t$	existential
`C`	capability
$e ::=$	`Terms`
v	value
x	variable
`inl` e	left injection
`inr` e	right injection
`case` $e\ v\ v$	sum case
$e\ e$	application
`if` $(e \preceq e)\ e\ e$	if delegation
$e[p]$	instantiation
`open` $(\alpha, x) = e$ `in` e	opening
`if` $(e \Rightarrow e \triangleright i)\, e\ e$	if certify
`declassify` $e\ t$	declassify
`let`$(e \preceq e)$ `in` e	let delegate
`acquire` $e \triangleright i$	acquire
$v ::=$	`Values`
`*`	unit
`inl` v	left injection
`inr` v	right injection
$\lambda x{:}t.\, e$	function
X	principal
$\Lambda\alpha \preceq p.\, e$	polymorphism
`pack` $(p \preceq q, e)$	packing
`let` $(X_1 \preceq X_2)$ `in` v	let delegate
$X\{i\}$	capability
$i ::=$	`Privileges`
`declassify`	declassification
`delegate`$_{p \preceq p}$	delegation

Για να υλοποιήσει τις εγκρίσεις των principals κατά το χρόνο εκτέλεσης, η λ_{RP} χρησιμοποιεί το μηχανισμό των capabilities. Έτσι ξεχωρίζει τις στατικές άδειες από τις αναπαραστάσεις τους στο χρόνο εκτέλεσης. Η στατική άδεια γράφεται $p \triangleright i$ και δηλώνει ότι το πρόγραμμα χρειάζεται την έγκριση του p για να χρησιμοποιήσει το δικαίωμα i. Η έγκριση κατά το χρόνο εκτέλεσης γράφεται $X\{i\}$ και είναι ένα capability που δημιουργήθηκε από το run-time principal X και παραχωρεί το δικαίωμα i. Ένα capability έχει τύπο `C`. Ένα πρόγραμμα μπορεί να εξετάσει ένα capability χρησιμοποιώντας το δυναμικό έλεγχο `if` $(e_1 \Rightarrow e_2 \triangleright i)\, e_3\ e_4$, όπου το e_2 αποτιμάται σε run-time principal και το e_1 σε capability. Αν το capability υπονοεί ότι το principal επιτρέπει το i τότε αποτιμάται το e_3, αλλιώς αποτιμάται το e_4.

Ονομάζουμε π το σύνολο των στατικών αδειών, δηλαδή $\pi ::= \cdot \mid \pi, p \triangleright i$. Ανάλογα με το δικαίωμα i, ορίζουμε το $\pi(i) \stackrel{\text{def}}{=} \{p \mid p \triangleright i \in \pi\}$ που είναι το σύνολο των principals που δίνουν

το δικαίωμα i. Ο κανόνας $\Delta \vdash t_2 - t_1 = s$ σημαίνει ότι ο τύπος t_2 μπορεί να γίνει declassified στον τύπο t_1 χρησιμοποιώντας την έγκριση των principals στο s. Το s αποτελείται από τους ιδιοκτήτες στις ετικέτες του t_2. Χρησιμοποιώντας αυτούς τους ορισμούς, ο κανόνας τύπων για το declassify γράφεται:

$$\frac{\Delta;\Gamma;\pi \vdash e : t_2 \qquad \Delta \vdash t_2 - t_1 = s \qquad \Delta \vdash s \preceq \pi(\texttt{declassify})}{\Delta;\Gamma;\pi \vdash \texttt{declassify}\, e\, t_1 : t_1}$$

Ο κανόνας $\Delta \vdash s \preceq \pi(\texttt{declassify})$ σημαίνει ότι το σύνολο των principals που επιτρέπουν στατικά το declassify αρκεί για να δράσει για το s. Με παρόμοια λογική γράφεται και ο κανόνας τύπων για το δικαίωμα delegate:

$$\frac{\begin{array}{cc}\Delta;\Gamma;\pi \vdash e_1 : \mathrm{P}_p & \Delta;\Gamma;\pi \vdash e_2 : \mathrm{P}_q \\ \Delta, p \preceq q;\Gamma;\pi \vdash e_3 : t & \Delta \vdash p \preceq \pi(\texttt{delegate}_{p \preceq q})\end{array}}{\Delta;\Gamma;\pi \vdash \texttt{let}(e_1 \preceq e_2)\ \texttt{in}\ e_3 : t}$$

Όπως φαίνεται στον παραπάνω κανόνα, το σώμα του let αποτιμάται χρησιμοποιώντας την επεκτεταμένη ιεραρχία δράσης αλλά μετά αποκαθίσταται η αρχική ιεραρχία. Με αυτό τον τρόπο εξασφαλίζεται ότι η αντιπροσώπευση γίνεται τοπικά, μόνο στην e_3.

Ως τώρα είδαμε τι χρειάζεται για να δοθούν τα δικαιώματα σε κάποια principals αλλά δεν εξετάσαμε πως αποκτά τα απαιτούμενα capabilities το περιβάλλον εκτέλεσης. Όπως είπαμε, τα principals αντιπροσωπεύουν χρήστες ή γενικά οντότητες που αλληλεπιδρούν με το περιβάλλον εκτέλεσης. Τα capabilities λοιπόν, μπορούν να δοθούν με διάφορους τρόπους: όταν κάνει ένας χρήστης login, μετά την πιστοποίηση της ταυτότητας κάποιου απομακρυσμένου χρήστη κ.λπ. Η λ_{RP} μοντελοποιεί το εξωτερικό περιβάλλον $\mathcal{E}$ σαν μαύρο κουτί και ορίζει την έκφραση acquire για να εξετάζει δυναμικά αν το $\mathcal{E}$ παρέχει τα απαιτούμενα capabilities. Γράφοντας $\texttt{acquire}\ e \triangleright i$, όπου η e αποτιμάται σε κάποιο run-time principal, το πρόγραμμα ρωτά το $\mathcal{E}$ αν υπάρχει το capability $X\{i\}$. Η έκφραση αυτή επιστρέφει το $X\{i\}$ αν είναι διαθέσιμο, αλλιώς επιστρέφει `*`. Ο κανόνας τύπων της είναι:

$$\frac{\Delta;\Gamma;\pi \vdash e : \mathrm{P}_p}{\Delta;\Gamma;\pi \vdash \texttt{acquire}\, e \triangleright i : (\mathtt{C} + \mathtt{1})}$$

Η λ_{RP}, χρησιμοποιώντας τον αποτελεσματικό μηχανισμό των run-time principals, μπορεί να εκφράσει ισχυρές πολιτικές ασφάλειας. Επίσης, οι ιδιότητες αυτές αποδεικνύονται ορθές θεωρητικά (στο [33] υπάρχουν οι αποδείξεις προόδου και διατήρησης των τύπων) σε αντίθεση με τις πλατφόρμες Java και .NET όπου είναι δύσκολη η θεωρητική επιβεβαίωση του συστήματος ασφαλείας τους. Στα επόμενα κεφάλαια θα δούμε κάποιες εφαρμογές της λ_{RP}, όπως την υλοποίηση κρυπτογραφικού πρωτοκόλλου δημοσίου κλειδιού κ.α. Η Apollo, η παραλλαγή της λ_{RP} που χρησιμοποιήσαμε υπάρχει στο `http://www.cis.upenn.edu/~stse/apollo`.

Κεφάλαιο 4

Υλοποίηση του κρυπτογραφικού πρωτοκόλλου δημοσίου κλειδιού Needham-Schroeder

Στο κεφάλαιο αυτό παρουσιάζεται η υλοποίηση του κρυπτογραφικού πρωτοκόλλου δημοσίου κλειδιού Needham-Schroeder (Needham-Schroeder Public Key Protocol, NSPKP). Αφού περιγράψουμε συνοπτικά τις βασικές έννοιες της κρυπτογραφίας δημοσίου κλειδιού, θα παρουσιάσουμε το NSPKP και τις υλοποιήσεις του στη Java, στη C# και στην Apollo. Τέλος, θα γίνει μια συγκριτική μελέτη των πλεονεκτημάτων και των μειονεκτημάτων κάθε υλοποίησης.

4.1 Κρυπτογραφία δημοσίου κλειδιού

Γενικά, ένα κρυπτοσύστημα αποτελείται από δύο αλγορίθμους: τον αλγόριθμο κρυπτογράφησης ή κωδικοποίησης E και τον αλγόριθμο αποκρυπτογράφησης ή αποκωδικοποίησης D. Το αρχικό κείμενο είναι το κείμενο προς κρυπτογράφηση. Χρησιμοποιώντας το σαν είσοδο του αλγορίθμου κρυπτογράφησης παίρνουμε σαν έξοδο το κρυπτοκείμενο. Αντίστοιχα, το κρυπτοκείμενο είναι η είσοδος του αλγορίθμου αποκρυπτογράφησης και η έξοδός του είναι το αρχικό κείμενο. Τα κρυπτοσυστήματα, ανάλογα με τον αριθμό των κλειδιών που χρησιμοποιούν κατατάσσονται στις παρακάτω κατηγορίες:

Χωρίς κλειδί. Οι αλγόριθμοι κρυπτογράφησης και αποκρυπτογράφησης δε χρησιμοποιούν παραμέτρους. Για να είναι ασφαλές το κρυπτοσύστημα, οι αλγόριθμοι πρέπει να κρατιούνται μυστικοί και να είναι γνωστοί μόνο στα πρόσωπα που ανταλλάσουν τα κρυπτογραφημένα μηνύματα.

Συμμετρική κρυπτογραφία. Οι αλγόριθμοι E και D χρησιμοποιούν μια παράμετρο k που ονομάζεται κλειδί. Οι αλγόριθμοι μπορούν να κοινοποιηθούν αλλά το κλειδί πρέπει να μείνει μυστικό.

Ασύμμετρη κρυπτογραφία ή κρυπτογραφία δημοσίου κλειδιού. Οι E και D χρησιμοποιούν διαφορετικά κλειδιά k και k'. Οι αλγόριθμοι και το k κοινοποιούνται αλλά το k' μένει μυστικό.

Παρατηρούμε ότι στη συμμετρική κρυπτογραφία το k πρέπει να κρατηθεί κρυφό. Αν θέλουν δύο άνθρωποι να επικοινωνήσουν πρέπει να συμφωνήσουν για το κλειδί μέσω ενός ασφαλούς καναλιού, με άλλα λόγια η ασφάλεια μετριάζεται αφού η έναρξη της επικοινωνίας προϋποθέτει την ύπαρξη ενός έμπιστου μέσου. Στην κρυπτογραφία δημοσίου κλειδιού αυτό δε χρειάζεται. Ο κάθε χρήστης γνωστοποιεί το δημόσιο κλειδί του k. Αν κάποιος θέλει να του στείλει μήνυμα το κρυπτογραφεί με το k και ο χρήστης χρησιμοποιεί το k' για την αποκρυπτογράφηση που το γνωρίζει μόνο ο ίδιος.

Η ιδέα της κρυπτογραφίας δημοσίου κλειδιού παρουσιάστηκε από τους Diffie και Hellman το 1976 [20]. Στηρίζεται στις συναρτήσεις μονής κατεύθυνσης. *Μονής κατεύθυνσης* είναι μια συνάρτηση $f(x)$ όταν είναι εύκολο (σε χαμηλό πολυωνυμικό χρόνο, ακόμα και γραμμικό) να υπολογιστεί το $f(x)$ από το x, αλλά είναι υπολογιστικά απρόσιτο να υπολογιστεί το x από το $f(x)$. Ένα πρόβλημα λέγεται απρόσιτο όταν δεν υπάρχει γνωστός πολυωνυμικός αλγόριθμος

που να το λύνει. Ένα παράδειγμα είναι ο τηλεφωνικός κατάλογος. Είναι πολύ εύκολο να βρεθεί το τηλέφωνο ενός ατόμου όταν είναι γνωστό το όνομά του, αλλά είναι πολύ δύσκολο να βρεθεί το όνομα ενός ατόμου αν είναι γνωστό το τηλέφωνό του. Η ασύμμετρη κρυπτογραφία βασίζεται σε απρόσιτα προβλήματα όπως το πρόβλημα της παραγοντοποίησης ενός αριθμού, δηλαδή της ανάλυσής του σε γινόμενο πρώτων παραγόντων. Έτσι, για να επικοινωνήσει ένας χρήστης Β με ένα χρήστη Α αρκεί να γνωρίζει το δημόσιο κλειδί του Α. Αν ένας χρήστης Γ παρεμβληθεί και υποκλέψει το κρυπτογραφημένο μήνυμα του Β δεν μπορεί να το αποκρυπτογραφήσει χωρίς να χρειαστεί ένας πολύ χρονοβόρος υπολογισμός. Για την υλοποίηση του NSPKP στη Java και στη C# χρησιμοποιήσαμε το κρυπτοσύστημα δημοσίου κλειδιού RSA. Η περιγραφή του RSA ξεφεύγει από το σκοπό αυτής της εργασίας. Πληροφορίες για το RSA και γενικά για την κρυπτογραφία δημοσίου κλειδιού υπάρχουν στο [35].

4.2 Περιγραφή του πρωτοκόλλου Needham-Schroeder

Το NSPKP προτάθηκε από τους Roger Needham και Michael Schroeder [29]. Σκοπός του είναι η πιστοποιημένη επικοινωνία μεταξύ δύο χρηστών σε διαφορετικούς υπολογιστές, όπου η αποστολή μηνυμάτων γίνεται εναλλάξ και προς τις δύο κατευθύνσεις. Προτού επικοινωνήσουν δύο χρήστες A και B χρησιμοποιούν το πρωτόκολλο για να βεβαιωθούν ότι όντως ανταλλάσουν μηνύματα μεταξύ τους και όχι με κάποιον εισβολέα που προσποιείται ότι είναι ο A ή ο B.

Κάθε χρήστης χρησιμοποιεί σαν όνομά του ένα μοναδικό αναγνωριστικό (A, B, ..., N). Θα χρησιμοποιούμε τις συντομογραφίες PK και SK για τα δημόσια και ιδιωτικά κλειδιά των χρηστών αντίστοιχα, π.χ. γράφοντας PK(A) εννοούμε το δημόσιο κλειδί του A. Επίσης, γράφοντας $\{msg\}^{key}$ εννοούμε ότι το μήνυμα msg είναι κρυπτογραφημένο με το κλειδί key. Υποθέτουμε ότι οι A και B δε γνωρίζουν ο ένας το δημόσιο κλειδί του άλλου και έτσι πρέπει να επικοινωνήσουν με έναν εξυπηρετητή πιστοποίησης (authentication server, AS) για να του ζητήσουν τα κλειδιά. Δείτε τον πίνακα 4.1: Αρχικά, ο A στέλνει μήνυμα στον AS ζητώντας του το δημόσιο κλειδί του B (βήμα 1). Ο AS απαντάει στον A, στέλνοντάς του το PK(B) και το όνομα του B κρυπτογραφημένα με το ιδιωτικό του κλειδί (βήμα 2). Ο A μπορεί να διαβάσει το μήνυμα αποκρυπτογραφώντας το με το δημόσιο κλειδί του AS. Αυτό γίνεται για να βεβαιωθεί ο A ότι αυτός που του έστειλε το μήνυμα είναι ο AS. Έπειτα, ο A στέλνει μήνυμα στον B που περιέχει το όνομά του και ένα τυχαία επιλεγμένο αναγνωριστικό N_A κρυπτογραφημένο με το PK(B) (βήμα 3). Αυτό το μήνυμα, που μπορεί να διαβαστεί μόνο από τον B, δηλώνει ότι κάποιος χρήστης ονόματι A προσπαθεί να επικοινωνήσει μαζί του στέλνοντας το N_A. Ο B τότε αποκτά το PK(A) με παρόμοιο τρόπο από τον AS (βήματα 4,5). Τώρα ο B στέλνει πίσω στον A το N_A και ένα τυχαίο αναγνωριστικό N_B (βήμα 6). Η παρουσία του N_A δηλώνει στον A ότι όντως μιλάει με τον B, αφού κανείς άλλος δεν θα μπορούσε να έχει διαβάσει το N_A. Τότε στέλνει πίσω στον B το N_B για να βεβαιωθεί και αυτός ότι όντως μιλάει με τον A (βήμα 7). Σ' αυτό το σημείο ολοκληρώνεται η πιστοποίηση της ταυτότητας των χρηστών και μπορούν να αρχίσουν τη συνομιλία τους. Στην περίπτωση που οι χρήστες γνωρίζουν τα

Πίνακας 4.1: Τα βήματα του πρωτοκόλλου Needham-Schroeder.

1.	$A \rightarrow AS:$	A, B
2.	$AS \rightarrow A:$	$\{\mathrm{PK}(B), B\}^{\mathrm{SK}(AS)}$
3.	$A \rightarrow B:$	$\{N_A, A\}^{\mathrm{PK}(B)}$
4.	$B \rightarrow AS:$	B, A
5.	$AS \rightarrow B:$	$\{\mathrm{PK}(A), A\}^{\mathrm{SK}(AS)}$
6.	$B \rightarrow A:$	$\{N_A, N_B\}^{\mathrm{PK}(A)}$
7.	$A \rightarrow B:$	$\{N_B\}^{\mathrm{PK}(B)}$

δημόσια κλειδιά των υπολοίπων χρηστών, η επικοινωνία με τον εξυπηρετητή πιστοποίησης δεν

είναι απαραίτητη και τα βήματα του πρωτοκόλλου μειώνονται σε τρία (3, 6, 7).

Ο Gavin Lowe βρήκε ένα ελάττωμα στο πρωτόκολλο και πρότεινε μια επίθεση που μπορεί να παραβιάσει την ασφάλεια της επικοινωνίας [24]. Δείτε τον πίνακα 4.2: Αν ο A επικοινωνήσει με ένα κακόβουλο χρήστη I, τότε ο I μπορεί να παραστήσει τον A για να επικοινωνήσει με τον B. Παρατηρούμε ότι ο I δεν μπορεί να αποκρυπτογραφήσει τα μηνύματα του B, έτσι τα στέλνει

Πίνακας 4.2: Η επίθεση του Lowe στο πρωτόκολλο Needham-Schroeder.

1.	$A \to I:$	$\{N_A, A\}^{\mathrm{PK}(I)}$
I.	$I \to B:$	$\{N_A, A\}^{\mathrm{PK}(B)}$
II.	$B \to I:$	$\{N_A, N_B\}^{\mathrm{PK}(A)}$
2.	$I \to A:$	$\{N_A, N_B\}^{\mathrm{PK}(A)}$
3.	$A \to I:$	$\{N_B\}^{\mathrm{PK}(I)}$
III.	$I \to B:$	$\{N_B\}^{\mathrm{PK}(B)}$

στον A. Τα μηνύματα που δέχεται ο A από τον I είναι βάση του πρωτοκόλλου και έτσι δεν μπορεί να υποψιαστεί ότι ο I είναι εισβολέας. Τελικά, ο B πιστεύει ότι επικοινωνεί με τον A ενώ στην πραγματικότητα επικοινωνεί με τον I. Ο Lowe πρότεινε μια αλλαγή στο πρωτόκολλο που διορθώνει αυτό το ελάττωμα: στο δεύτερο βήμα επικοινωνίας, ο B να στέλνει στον A το $\{N_A, N_B, B\}^{\mathrm{PK}(A)}$. Με αυτό τον τρόπο, ένας χρήστης I δεν μπορεί να παραστήσει τον A γιατί θα πρέπει να στείλει στον A το μήνυμα $\{N_A, N_B, B\}^{\mathrm{PK}(A)}$ για αποκρυπτογράφηση ενώ ο A θα περίμενε το $\{N_A, N_B, I\}^{\mathrm{PK}(A)}$. Το πρωτόκολλο χωρίς εξυπηρετητή πιστοποίησης και μετά τη διόρθωση του Lowe φαίνεται στον πίνακα 4.3. Αυτή είναι η παραλλαγή του πρωτοκόλλου που υλοποιείται στην παρούσα εργασία.

Πίνακας 4.3: Τα βήματα του απλοποιημένου Needham-Schroeder, με τη διόρθωση του Lowe.

1.	$A \to B:$	$\{N_A, A\}^{\mathrm{PK}(B)}$
2.	$B \to A:$	$\{N_A, N_B, B\}^{\mathrm{PK}(A)}$
3.	$A \to B:$	$\{N_B\}^{\mathrm{PK}(B)}$

4.3 Υλοποίηση και σχολιασμός των αποτελεσμάτων

4.3.1 Java

Η έκδοση της Java που χρησιμοποιήσαμε είναι η Java 1.4. Η έκδοση αυτή παρέχει κλάσεις με κατάλληλα πεδία για την υλοποίηση του RSA αλλά δεν παρέχει τις μεθόδους που το υλοποιούν. Η υλοποίηση του RSA που χρησιμοποιήθηκε είναι το αρχείο `bcprov-jdk14-120.jar` που υπάρχει στο `http://www.bouncycastle.org/latest_releases.html`.

Τα αναγνωριστικά που αποτελούν τις ταυτότητες των χρηστών που επικοινωνούν είναι Alice και Bob (όπως συνηθίζεται σε πολλές κρυπτογραφικές εφαρμογές). Οι χρήστες υλοποιούνται ως αντικείμενα. Τα μηνύματα που στέλνουν ο ένας στον άλλο υλοποιούνται με κλήσεις μεθόδων των δύο αντικειμένων. Η Alice είναι ένα στιγμιότυπο της κλάσης `Initiator` και ο Bob της κλάσης `Responder`. Κάποια από τα πεδία της κλάσης `Initiator` είναι:

```
private String     id;    // αναγνωριστικό-ταυτότητα του χρήστη
private PrivateKey priv;  // ιδιωτικό κλειδί του χρήστη
public  PublicKey  publ;  // δημόσιο κλειδί του χρήστη
```

```
private Nonce anonce;     // τυχαίο αναγνωριστικό που στέλνει στο Bob
```

Παρατηρήστε εδώ ότι τα πεδία **id**, **priv** και **anonce** πρέπει να είναι ιδιωτικά για να μην μπορούν να προσπελαστούν από αντικείμενα άλλων κλάσεων. Αντίθετα, το πεδίο **publ** είναι δημόσιο για να φαίνεται από άλλα αντικείμενα που θέλουν να επικοινωνήσουν με την Alice. Παρόμοια πεδία έχει και η κλάση **Responder**. Στο βήμα 1 του πρωτοκόλλου, η Alice στέλνει στο Bob το τυχαίο nonce της N_A και το όνομά της κρυπτογραφημένα με το δημόσιο κλειδί του Bob. Αυτά γίνονται στη μέθοδο **initiate**.

```
// Η μέθοδος Initiator.initiate
public void initiate (Responder b)
   throws GeneralSecurityException
{
      bid = b.getId();
      System.out.println(id + ":\tinitiating communication with " + bid);
      anonce = new Nonce();
      System.out.println(id + ":\tmy nonce is " + anonce.toString());
      Message clearSnd = new Message(Nonce.SIZE + id.length());
      clearSnd.append(anonce.getBytes());
      clearSnd.append(id.getBytes());
      System.out.println(id + ":\tsending (" +
                           anonce.toString() + ", " + id + ")");
      Message encryptedSnd = clearSnd.encrypt(rsaCipher, b.publ);
      b.message1(this, encryptedSnd);
}
```

Η αποστολή του μηνύματος γίνεται καλώντας τη μέθοδο **message1** του Bob. Ο Bob αποκρυπτογραφεί το μήνυμα και στέλνει πίσω στην Alice το N_A μαζί με το N_B και το όνομά του.

```
// Η μέθοδος Responder.message1
public void message1 (Initiator a, Message encryptedRcv)
   throws GeneralSecurityException
{
      Message clearRcv = encryptedRcv.decrypt(rsaCipher, priv);
      Nonce anonce = new Nonce(clearRcv.read(Nonce.SIZE));
      String aid = new String(clearRcv.read());
      System.out.println(id + ":\treceived (" +
                              anonce.toString() + ", " + aid + ")");
      bnonce = new Nonce();
      System.out.println(id + ":\tmy nonce is " + bnonce.toString());
      Message clearSnd = new Message(2*Nonce.SIZE + id.length());
      clearSnd.append(anonce.getBytes());
      clearSnd.append(bnonce.getBytes());
      clearSnd.append(id.getBytes());
      System.out.println(id + ":\tsending (" + anonce.toString() + ", "
       + bnonce.toString() + ", " + id + ")");
      Message encryptedSnd = clearSnd.encrypt(rsaCipher, a.publ);
      a.message2(this, encryptedSnd);
}
```

Η αποστολή γίνεται καλώντας τη μέθοδο **message2** της Alice. Εδώ η Alice συγκρίνει το nonce που έλαβε από το Bob με το N_A που του είχε στείλει πρωτύτερα. Αν είναι ίδια, τότε βεβαιώνεται ότι μιλάει με το Bob και του στέλνει πίσω το N_B για να πιστοποιήσει και αυτός την ταυτότητα της Alice.

```
// Η μέθοδος Initiator.message2
public void message2 (Responder b, Message encryptedRcv)
   throws GeneralSecurityException
```

```
{
      Message clearRcv = encryptedRcv.decrypt(rsaCipher, priv);
      Nonce anonce = new Nonce(clearRcv.read(Nonce.SIZE));
      Nonce bnonce = new Nonce(clearRcv.read(Nonce.SIZE));
      String bid = new String(clearRcv.read());
      System.out.println(id + ":\treceived (" + anonce.toString() + ", "
       + bnonce.toString() + ", " + bid + ")");
      if (!anonce.equals(this.anonce)) {
       String msg = id + ": the nonce you sent me back is not mine!";
        throw new GeneralSecurityException(msg);
      }
      else
         System.out.println(id + ":\teverything ok for me");
        Message clearSnd = new Message(Nonce.SIZE);
        clearSnd.append(bnonce.getBytes());
        System.out.println(id + ":\tsending (" + bnonce.toString() + ")");
        Message encryptedSnd = clearSnd.encrypt(rsaCipher, b.publ);
        b.message3(this, encryptedSnd);
}
```

Η αποστολή γίνεται καλώντας τη μέθοδο message3 του Bob. Ο Bob πιστοποιεί την ταυτότητα της Alice και τα βήματα του πρωτοκόλλου έχουν ολοκληρωθεί. Οι δύο χρήστες μπορούν να αρχίσουν τη συνομιλία τους. Παρατηρείστε ότι σε περίπτωση που ένας εκ των δύο χρηστών αποτύχει να πιστοποιήσει την ταυτότητα του άλλου, εγείρει μια εξαίρεση ασφάλειας και το πρόγραμμα τερματίζει.

```
// Η μέθοδος Responder.message3
public void message3 (Initiator a, Message encryptedRcv)
   throws GeneralSecurityException
{
      Message clearRcv = encryptedRcv.decrypt(rsaCipher, priv);
      Nonce bnonce = new Nonce(clearRcv.read(Nonce.SIZE));
      System.out.println(id + ":\treceived (" + bnonce.toString() + ")");
      if (!bnonce.equals(this.bnonce)) {
         String msg = id + ": the nonce you sent me back is not mine!";
         throw new GeneralSecurityException(msg);
      }
      else
         System.out.println(id + ":\teverything ok for me");
}
```

4.3.2 C#

Το πλαίσιο .NET 2.0 Beta που χρησιμοποιήσαμε παρέχει υλοποιήση του RSA στις συναρτήσεις βιβλιοθήκης του [3]. Κατά τα άλλα, η υλοποίηση του NSPKP στη C# δεν έχει ουσιαστικές διαφορές από αυτή στη Java. Διατηρήθηκε η ίδια ακριβώς μεθοδολογία και οι μόνες διαφορές ήταν στη σύνταξη από γλώσσα σε γλώσσα.

4.3.3 Apollo

Στην Apollo τα πράγματα είναι διαφορετικά απ' ότι στη Java και τη C#. Η Apollo ακολουθεί το συναρτησιακό προγραμματιστικό στυλ και έτσι ένα πρόγραμμα αποτελείται από διαδοχικές κλήσεις συναρτήσεων. Πρέπει να τονίσουμε ότι η Apollo είναι στην ουσία μια γλώσσα-εξομοίωση της λ_{RP} και δεν μπορεί να χρησιμοποιηθεί για πραγματικές εφαρμογές. Σε αυτό το πλαίσιο θα τη συγκρίνουμε με τις δύο άλλες γλώσσες.

Οι χρήστες Alice και Bob υλοποιούνται ως run-time principals. Η γλώσσα δεν παρέχει κρυπτογραφικές συναρτήσεις αλλά μπορούν εύκολα να προσομοιωθούν ως εξής: για να κρυπτογραφηθεί ένα μήνυμα με το δημόσιο κλειδί της Alice αρκεί να έχει ετικέτα {Alice}, αφού όπως είδαμε μόνο η Alice μπορεί να διαβάσει δεδομένα με αυτή την ετικέτα. Ομοίως, τα μηνύματα προς τον Bob θα 'χουν ετικέτα {Bob}. Τα μηνύματα που στέλνουν οι χρήστες θα είναι πλειάδες με κάποια κατάλληλη ετικέτα.[1] Οι ετικέτες στην Apollo γράφονται διαφορέτικα απ' ότι στη λ_{RP}: το δεδομένο με ετικέτα {R P P1 P2... Pn} έχει ιδιοκτήτη το principal P και αναγνώστες τα principals από P1 ως Pn. Δεν υπονοείται ότι και το P ανήκει στο σύνολο των αναγνωστών, π.χ. μια ετικέτα που η Alice είναι ιδιοκτήτης και αναγνώστης πρέπει να αναφέρει την Alice δύο φορές {R Alice Alice}. Ο κώδικας του NSPKP σε Apollo φαίνεται παρακάτω.

```
let AliceToBob1 =
   (fun AliceNonce:int.<AliceNonce, 'alice>{R bob bob}) 42 ,
```

Κατά τα γνωστά, η Alice στέλνει στο Bob ένα τυχαίο nonce και το όνομά της. Τα nonces θα παριστάνονται με ακεραίους. Ο αριθμός 42 επιλέχθηκε από μας, αφού δεν υπάρχει γεννήτρια τυχαίων αριθμών στην Apollo.

```
BobToAlice =
   (fun BobNonce:int, c1:cert, c2:cert, message: <int, 'alice>{R bob bob}.
       if c1 => 'bob #> '(delegate alice) then
          delegate 'bob <: 'alice in (
             if c2 => 'bob #> '(declassify alice) then
                declassify (bind msg2 = message in
                                 <msg2.1, BobNonce, 'bob>{R bob bob})
                as <int, int, 'bob>{R alice alice}
             else
                error "fail"
             )
       else
          error "fail") 7,
```

Ο Bob λαμβάνει το μήνυμα της Alice και της στέλνει το δικό του, το οποίο πρέπει να περιέχει και το AliceNonce, δηλαδή το πρώτο στοιχείο της πλειάδας που έλαβε. Αυτή η πλειάδα έχει ετικέτα {R bob bob} και πρέπει να μετατραπεί σε {R alice alice} πριν σταλεί. Εδώ χρησιμοποιούνται τα δικαιώματα. Για να προστεθεί η Alice στους αναγνώστες της πλειάδας πρέπει να γίνει declassified η ετικέτα της πλειάδας σε {R bob alice}. Το delegate χρησιμοποιείται για να παραχωρήσει ο Bob την ιδιοκτησία της πλειάδας στην Alice, δηλαδή για να μετατραπεί η ετικέτα σε {R alice alice}

```
AliceToBob2 =
   (fun AliceNonce:int,c1:cert,c2:cert,message:<int,int,'bob>{R alice alice}.
       if c1 => 'alice #> '(delegate bob) then
           delegate 'alice <: 'bob in  (
              if c2 => 'alice #> '(declassify bob) then
                 declassify (bind msg2 = message in
                                 if msg2.1 = AliceNonce then
                                    <msg2.2>{R alice alice}
                                 else
                                    error "You are not Bob!!")
                 as <int>{R bob bob}
              else
                 error "fail"
        )
       else
           error "fail") 42,
```

[1] Η Apollo έχει μικρές διαφορές στη σύνταξη από τη λ_{RP}. Επίσης παρέχει κάποιες επιπλέον εκφράσεις όπως δομές-let, πλειάδες κ.α. που δεν υπάρχουν στη λ_{RP} αλλά μπορούν να προστεθούν χωρίς ιδιαίτερες αλλαγές.

Η Alice λαμβάνει το μήνυμα του Bob. Ελέγχει αν το AliceNonce που της έστειλε είναι το σωστό και μετά του στέλνει πίσω το BobNonce για να πιστοποιήσει και αυτός την ταυτότητά της. Χρησιμοποιούμε και εδώ τα ίδια δικαιώματα για τη μετατροπή της ετικέτας, μόνο που τώρα ο Bob δρα για την Alice.

```
BobReads2 =
   (fun BobNonce:int, message:<int>{R bob bob}.
       bind msg2 = message in
          if msg2.1 = BobNonce then
              "Successful Communication"
          else
              error "You are not Alice!!") 7
```

Τέλος, αν ο Bob πιστοποιήσει την Alice επιτυχώς τότε μπορεί να αρχίσει η συνομιλία. Αυτές οι συναρτήσεις σε Apollo που είδαμε παραπάνω πρέπει να καλούν η μία την άλλη για να υπάρχει η ροή πληροφορίας στο πρόγραμμα. Θα έχουμε δηλαδή:

```
BobReads2
    (AliceToBob2  (acquire 'alice #> '(delegate bob))
                  (acquire 'alice #> '(declassify bob))
                  (BobToAlice (acquire 'bob #> '(delegate alice))
                              (acquire 'bob #> '(declassify alice))
                               AliceToBob1))
```

Παρατηρήστε ότι τα ορίσματα στις συναρτήσεις γράφονται σε παράθεση (όπως η εφαρμογή στο λ-λογισμό). Είπαμε ότι η Apollo είναι στην ουσία εξομοίωση και όχι πλήρης γλώσσα προγραμματισμού. Δε δίνεται η δυνατότητα της εισόδου τιμών από το πληκτρολόγιο, διάβασμα αρχείων κ.λπ. Γι' αυτό το λόγο, η έκφραση acquire που στη λ_{RP} μπορεί να επιστρέψει *unit*, στην Apollo επιστρέφει πάντα το ζητούμενο capability· υποθέτουμε δηλαδή ότι τα capabilities δίνονται μετά από επικοινωνία με το χρήστη (principal) που τα παραχωρεί. Το `'alice #> '(declassify bob)` δε γίνεται να έχει δημιουργηθεί από τον Bob. Τα capabilities στην Apollo έχουν τύπο `cert`, δηλαδή οι μεταβλητές `c1` και `c2` στη συνάρτηση BobToAlice παίρνουν τιμές από τις εκφράσεις `(acquire 'bob #> '(delegate alice))` και `(acquire 'bob #> '(declassify alice))` αντίστοιχα.

4.3.4 Σχόλια

Ενώ οι υλοποιήσεις στη Java και τη C# μοιάζουν πολύ μεταξύ τους, η υλοποίηση στην Apollo είναι πολύ διαφορετική. Παρατηρούμε ότι γλώσσες με ισχυρό σύστημα τύπων μπορούν να εκφράσουν με αυτό πολύπλοκους μηχανισμούς όπως η κρυπτογραφία δημοσίου κλειδιού με αρκετά απλό και αποτελεσματικό τρόπο. Αυτό βοηθά και στη μείωση του μεγέθους του κώδικα.

Από την άλλη, η οργάνωση των χρηστών σε αντικείμενα μοιάζει πιο φυσική απ' ότι στην Apollo που δεν μπορούμε να έχουμε κάποιο τμήμα κώδικα μόνο για τον Bob ή μόνο για την Alice. Οι συναρτήσεις AliceToBob1, BobToAlice κ.λπ. είναι ορατές σε όλο το πρόγραμμα, μόνο τα δεδομένα μπορούν να προστατεύονται μέσω των ετικετών. Επίσης, στην Apollo δεν μπορούμε να έχουμε μεταβλητές τύπου $Ref(t)$, δηλαδή μεταβλητές όπως στις κλασικές γλώσσες προγραμματισμού, που αντιστοιχούν σε μια θέση μνήμης και η τιμή τους αλλάζει όσες φορές θέλουμε στο πρόγραμμα. Έτσι, αναγκαζόμαστε να περνάμε κάθε φορά σαν όρισμα το AliceNonce και το BobNonce.

Μια παρατήρηση όσον αφορά τη Java και τη C#: Θα ήταν καλό να κληρονομούσαν από μια ίδια κλάση (έστω `NspkpUsers`) οι `Initiator` και `Responder`, στην οποία θα υπήρχαν τα κοινά τους πεδία. Έτσι, θα είχαμε δυνατότητα επαναχρησιμοποίησης του κώδικα. Αυτό όμως δεν μπορεί να γίνει γιατί τα protected πεδία μιας κλάσης φαίνονται από τις υποκλάσεις της. Αν δηλαδή στην `NspkpUsers` υπήρχαν τα πεδία

```
protected PrivateKey priv;
protected Nonce any_nonce;
```

οι κλάσεις που κληρονομούν από την `NspkpUsers` βλέπουν η μια το ιδιωτικό κλειδί της άλλης. Παρατηρήστε ότι *οι access modifiers (private, protected, ...) αφορούν κλάσεις και όχι στιγμιότυπα κλάσεων*. Με άλλα λόγια, ένα ιδιωτικό πεδίο δεν είναι ορατό από αντικείμενα άλλης κλάσης αλλά τα αντικείμενα της ίδιας κλάσης βλέπουν το ένα τα ιδιωτικά πεδία του άλλου. Αν οι access modifiers ήταν στο επίπεδο των αντικειμένων και όχι των κλάσεων, η Alice και ο Bob θα μπορούσαν ακόμα και να είναι αντικείμενα της ίδιας κλάσης(!)

Κεφάλαιο 5

Υλοποίηση ηλεκτρονικών τραπεζικών συναλλαγών

5.1 Περιγραφή του προβλήματος

Στις μέρες μας ένα μεγάλο ποσοστό των εμπορικών και οικονομικών συναλλαγών γίνεται ηλεκτρονικά. Αριθμοί πιστωτικών καρτών, αριθμοί τραπεζικών λογαριασμών και άλλα ευαίσθητα δεδομένα μεταφέρονται μέσω διαδικτύου. Είναι λογικό σε τέτοιου είδους εφαρμογές η ασφάλεια να έχει καθοριστικό ρόλο. Χαρακτηριστικό παράδειγμα είναι οι ηλεκτρονικές τραπεζικές συναλλαγές. Κάθε πελάτης μιας τράπεζας αποκαλύπτει στο μηχάνημα ανάληψης χρημάτων (ATM) τον προσωπικό του κωδικό για να διεξαγάγει τη συναλλαγή που επιθυμεί. Τα στοιχεία αυτά μεταφέρονται μέσω δικτύου στη βάση δεδομένων της τράπεζας όπου γίνεται η επαλήθευσή τους. Κατά τη μεταφορά των δεδομένων πρέπει οι περιορισμοί ασφαλείας να είναι τέτοιοι ώστε να μην μπορεί κάποιος τρίτος να τα υποκλέψει.

Μια τέτοια συναλλαγή αναφέρεται ως παράδειγμα από τους Tse και Zdancewic στο [33] για να δείξουν πως μπορεί να επιτευχθεί η απαιτούμενη ασφάλεια χρησιμοποιώντας run-time principals. Στο παράδειγμά τους, χρησιμοποιείται η παραδοχή ότι το ATM και η τράπεζα επικοινωνούν μέσω ασφαλούς καναλιού. Αυτό μπορεί να επιτευχθεί με κρυπτογράφηση. Τα δεδομένα στο κανάλι πρέπει να μπορούν να διαβαστούν μόνο από αυτόν που θα τα παραλάβει. Παρακάτω περιγράφουμε την υλοποίηση μιας εφαρμογής τραπεζικών συναλλαγών στις τρεις γλώσσες και πως πληροί τις απαραίτητες προδιαγραφές.

5.2 Υλοποίηση και σχολιασμός των αποτελεσμάτων

5.2.1 Apollo

Το κυρίως πρόγραμμα φαίνεται στη σελ. 52.

```
let
   login =
     (fun input:<>. <pack 'alice with alice as some user.'user,
                     acquire 'alice #> '(delegate atm)
                    >{R atm atm})
   ,
```

Ο πελάτης κάνοντας login αποκαλύπτει την ταυτότητά του και επιτρέπει στο ATM να δρα για αυτόν (delegate). Ο πελάτης υλοποιείται ως run-time principal. Μετά πληκτρολογεί στο ATM την επιθυμητή συναλλαγή, π.χ. ανάληψη 100€. Αυτή η ενέργεια παριστάνεται από το capability `creq = acquire userid #> 'withdraw`. Παρατηρήστε ότι το δικαίωμα withdraw το ορίσαμε εμείς, αφού η Apollo επιτρέπει τον ορισμό νέων δικαιωμάτων εκτός των declassify, delegate. Το ATM βάζει τα στοιχεία του χρήστη και τα απαραίτητα capabilities για τη συναλλαγή σε ένα μήνυμα και τα στέλνει στην τράπεζα (αυτό γίνεται με την κλήση της συνάρτησης request). Όπως προαναφέραμε, τα δεδομένα στο κανάλι πρέπει να μπορούν να διαβαστούν μόνο από τον παραλήπτη τους και γι' αυτό το μήνυμα παίρνει την ετικέτα {R bank bank}.

```
def atm,alice,bank,prt.
def withdraw.
// ...
// ο κώδικας των δομών let για τις συναρτήσεις login, request κ.λπ.
in
< (
  fun u:<>.
  if (acquire 'atm #> '(declassify bank))
    => 'atm #> '(declassify bank) then
    bind x = login <> in
    let <y,cdel> = x in
    open userid = y with user in
    let creq = acquire userid #> 'withdraw in
    let msg = <'atm, userid, cdel, creq> in
    let dmsg = msg{R bank bank} in
    let balance = request [atm,user] dmsg in
    if (acquire 'atm #> '(declassify prt))
      => 'atm #> '(declassify prt) then
      let dbalance = declassify balance as int{R atm prt} in
      print dbalance
    else <>
  else error "insecure network"
  )<>
,
  (
  fun u:<>.
  bind x = listen <> in
  open y = x with agent, user in
  let <agentid,userid,cdel,creq,reply> = y in
  if cdel => userid #> '(delegate agent) then
    delegate userid <: agentid in
    if creq => userid #> 'withdraw then
      bind old = get [user] userid in
      let balance = old - 100 in
      let y = set [user] userid balance in
      let dbalance = declassify balance as int{R user user} in
      reply dbalance
    else <>
  else error "insecure network"
  )<>
>
```

Εδώ φαίνεται το σώμα της συνάρτησης request. Με τα c1 και c2 ελέγχεται αν ο χρήστης επιτρέπει στο ATM να δράσει γι' αυτόν και μετά ελέγχεται η επιθυμητή συναλλαγή. Αν παρέχονται αυτά τα capabilities, η τράπεζα επιστρέφει την κατάσταση του λογαριασμού σε ένα μήνυμα με ετικέτα {R atm atm}.

```
request =
     (poly agent, user. fun msg:<'agent,'user,cert,cert>{R bank bank}.
        if (acquire 'bank #> '(delegate atm)) => 'bank #> '(delegate atm) then
            delegate 'bank<:'atm in (
               if (acquire 'bank #> '(declassify atm))=>
                      'bank #> '(declassify atm) then
               declassify
                   bind x=msg in
                      let <a, u, c1, c2>=x in
                          if c1=> u #> '(delegate agent) then
```

```
                    if c2=> u #> 'withdraw then
                        1100{R bank bank}
                        else error "fail"
                         else error "fail"
                as int{R atm atm}
                else error "fail"
            )
        else error "fail")
    ,
```

Αν ο χρήστης έχει ζητήσει να τυπωθεί απόδειξη, το ATM κάνει declassify τα στοιχεία για να διαβάζονται από το principal prt και καλείται η συνάρτηση print.[1]

Το κεντρικό μηχάνημα της τράπεζας έχει πρόσβαση στο κανάλι και λαμβάνει το μήνυμα του ATM (συνάρτηση listen). Επιβεβαιώνει τη σύνδεση του πελάτη με το δίκτυο από το cdel και μετά κοιτάζει τι συναλλαγή επιθυμεί ο χρήστης (capability creq). Καλώντας τη συνάρτηση get παίρνει το τρέχον υπόλοιπο του χρήστη, αφαιρεί 100€ και ενημερώνει τη βάση δεδομένων για το καινούριο υπόλοιπο (συνάρτηση set). Μετά κάνει declassify το μήνυμα σε ετικέτα {R user user} για να το στείλει στο ATM. Το user είναι ένα run-time principal που θα πάρει τιμή κατά το χρόνο εκτέλεσης με το ATM στο οποίο έγινε το login.

5.2.2 Java

Στη Java τα μηχανήματα αυτόματης ανάληψης τα υλοποιήσαμε ως αντικείμενα, στιγμιότυπα της κλάσης ATM. Όλα τα πεδία και οι μέθοδοι της κλάσης είναι ιδιωτικά, εκτός από τη μέθοδο transaction που καλείται για να ξεκινήσει μια καινούρια συναλλαγή. Οι δυνατές συναλλαγές είναι δύο: ανάληψη χρημάτων και ερώτηση υπολοίπου (στην Apollo υλοποιήσαμε μόνο την ανάληψη χρημάτων). Το ρόλο του πελάτη στην εφαρμογή παίζει ο χρήστης, που εισάγει τα δεδομένα για τη συναλλαγή από το πληκτρολόγιο.

```
public static void main(String[] args) throws Exception{
       ATM atm=new ATM();
       while (true) {
            atm.transaction();
            if (wantsAnother()) continue; else break;
       }
       System.exit(0);
}
```

Όταν ξεκινάει η εφαρμογή, η συνάρτηση main δημιουργεί ένα αντικείμενο ATM και καλεί τη μέθοδο transaction. Αυτή καλεί τη μέθοδο login που ζητάει από το χρήστη να εισαγάγει τον τετραψήφιο κωδικό του. Μετά καλείται η μέθοδος request που ρωτάει το χρήστη αν θέλει να πάρει χρήματα ή να μάθει το υπόλοιπό του. Το ποσό που ζητάει ο χρήστης αποθηκεύεται στο πεδίο sum, το οποίο αν είναι 0 σημαίνει ότι έγινε ερώτηση υπολοίπου.

```
public class ATM {
private int password,sum;

public void transaction() throws IOException{
password = login();
sum = request();
System.out.println(Bank.message(password,sum));
}
// ... ο υπόλοιπος κώδικας της κλάσης
```

[1] Στην Apollo, δεν υπάρχει συνάρτηση που να τυπώνει δεδομένα στην οθόνη δηλαδή η print θα είναι κάπως έτσι: `(fun money : intR atm prt.<>)` απλά για να μην έχει πρόβλημα ο ελεγκτής τύπων. Θυμίζουμε ότι είναι απλά μια γλώσσα εξομοίωσης και όχι για πραγματικές εφαρμογές.

Η μέθοδος request δεν αντιστοιχεί στη request που είχαμε στην Apollo, αλλά στο capability creq. Η request της Apollo αντιπροσωπεύει την επικοινωνία με την τράπεζα, δηλαδή τη μέθοδο Bank.message εδώ. Το ATM περνάει ως παραμέτρους στην Bank.message τον κωδικό του χρήστη και το είδος της συναλλαγής που επιθυμεί. Η τράπεζα κάνει την απαιτούμενη επεξεργασία και το αποτέλεσμα που επιστρέφει τυπώνεται στην οθόνη (αντιστοιχία με την print της Apollo).

```
public static String message(int password, int sum){
        int user;
        if ((user = validUser(password))==5) return "There is no such user";
        if (sum==0) return ("You have " + accounts[user] + " dollars left.");
        return setBalance(sum, user);
}
```

Όπως και τα υπόλοιπα πεδία και μέθοδοι της κλάσης Bank, έτσι και η message είναι static, γιατί η τράπεζα είναι μία οπότε δε θα έχουμε περισσότερα του ενός στιγμιότυπα της κλάσης. Η προϋπόθεση που τέθηκε αρχικά για την εφαρμογή, να διαβάζονται τα μηνύματα μόνο από τον παραλήπτη υλοποιείται στη Java καλώντας τις μεθόδους του κατάλληλου αντικειμένου. Η message αρχικά καλεί τη μέθοδο validUser για να διαπιστώσει αν υπάρχει χρήστης με αυτό τον κωδικό. Παρατηρήστε την αντιστοιχία με την υλοποίηση της Apollo, που στη συνάρτηση της τράπεζας ελέγχεται το cdel προτού ολοκληρωθεί η συναλλαγή. Γι' αυτό το λόγο δεν κάναμε τον έλεγχο του κωδικού στη login και τον αφήσαμε για τη message. Μετά την επιτυχή πιστοποίηση του πελάτη διεξάγεται η συναλλαγή. Αν είναι ερώτηση υπολοίπου απλά επιστρέφεται το υπόλοιπο στο ATM που κάλεσε τη message. Αν είναι ανάληψη χρημάτων, καλείται η μέθοδος setBalance για να ενημερωθεί η βάση και το αποτέλεσμα επιστρέφεται πάλι στο κατάλληλο ATM.

5.2.3 C#

Η υλοποίηση της εφαρμογής στη C# δεν έχει ουσιαστικές διαφορές από αυτή στη Java. Υπάρχει απόλυτη αντιστοιχία στις κλάσεις και τις μεθόδους που υλοποιήσαμε. Ενδεικτικά παραθέτουμε τη μέθοδο ValidUser γραμμένη σε C#.

```
private static int ValidUser(int password){
       int i;
       for (i=0;i<5;i++)
       if (pelates[i]==password) break;
       return i;
}
```

Παρατηρήστε ότι επειδή το βάρος της εφαρμογής δίνεται στην επικοινωνία μεταξύ ATM και τράπεζας και στην ορθή πιστοποίηση του χρήστη, η βάση δεδομένων της τράπεζας είναι απλά ένας μικρός πίνακας πέντε ακεραίων που είναι οι κωδικοί των πελατών. Δείτε επίσης ότι, χωρίς να απαιτείται από την υλοποίηση των γλωσσών, τα ονόματα των μεθόδων στη C# αρχίζουν με κεφαλαίο ενώ στη Java με μικρό. Είναι μια σύμβαση που υιοθετείται από τους περισσότερους προγραμματιστές.

5.2.4 Σχόλια

Βλέπουμε ότι η υλοποίση στην Apollo παρουσιάζει αρκετές διαφορές σε σχέση με τις C# και Java. Η Apollo εκμεταλεύεται το μηχανισμό των principals για να παραστήσει όλες τις οντότητες που παίρνουν μέρος στη συναλλαγή. Η ταυτότητα ενός χρήστη είναι ένα principal (στο παράδειγμα έχουμε μόνο ένα χρήστη, την alice, αλλά θα μπορούσαν να υπάρχουν περισσότεροι) ενώ στις C# και Java είναι ένας ακέραιος που παριστάνει τον τετραψήφιο κωδικό του πελάτη. Από τις τιμές των run-time principals user και agent και τα capabilities που παρέχονται εξαρτάται η διεξαγωγή της συναλλαγής. Με άλλα λόγια, οι έλεγχοι του προγράμματος γίνονται

στο επίπεδο των τύπων σε αντίθεση με τις άλλες δύο γλώσσες που χρησιμοποιούμε δομές for, if, πίνακες κ.λπ. Ένα άλλο στοιχείο που διαφοροποιεί τις υλοποιήσεις είναι η χρήση υπαρξιακών τύπων στην Apollo. Οι υπαρξιακοί τύποι προσφέρουν ένα επιπλέον επίπεδο αφαίρεσης και έτσι αποκρύπτονται αχρείαστες λεπτομέρειες από αυτούς που χρησιμοποιούν τα δεδομένα με αυτό τον τύπο. Τέλος, πρέπει να πούμε ότι στην υλοποίηση σε Apollo δεν υπάρχει πραγματική επικοινωνία μεταξύ τράπεζας και ATM. Δείτε ότι τα δύο μέρη του κώδικα είναι στοιχεία μιας πλειάδας που σε κανένα σημείο δεν υπάρχει ροή πληροφορίας από το ένα στο άλλο. Αυτή όμως είναι η δομή που επιλέχθηκε στο [33] και υπάρχει στην ιστοσελίδα της Apollo. Γι' αυτό το λόγο διατηρήθηκε η ίδια δομή και εδώ.

Κεφάλαιο 6

Αμοιβαίος αποκλεισμός

6.1 Περιγραφή του προβλήματος

Η ακεραιότητα των δεδομένων όταν σε αυτά ενεργούν πολλοί χρήστες ταυτόχρονα είναι ένα πρόβλημα που το συναντάμε συχνά σε καθημερινές εφαρμογές. Σκεφτείτε τη διαδικασία κράτησης αεροπορικών εισητηρίων. Η εταιρία έχει πολλά διαφορετικά υποκαταστήματα που επικοινωνούν με το κεντρικό σύστημα για να κρατήσουν θέσεις. Έτσι, είναι σημαντικό σε μια τέτοια διαδικασία να υπάρχει συγχρονισμός για να αποφευχθούν φαινόμενα όπως η κράτηση μιας θέσης για δύο διαφορετικούς πελάτες κ.λπ.

Όταν πολλές διεργασίες που εκτελούνται παράλληλα προσπαθούν να προσπελάσουν ένα κοινό δεδομένο πρέπει να υπάρχει *αμοιβαίος αποκλεισμός* μεταξύ τους, για να διατηρηθεί η ακεραιότητα του δεδομένου. Αυτό σημαίνει ότι δε γίνεται μια διεργασία να αρχίσει να επεξεργάζεται ένα κοινό δεδομένο πριν τελείωσει η προηγούμενη. Στην περίπτωση που οι διεργασίες εκτελούνται στον ίδιο επεξεργαστή εφαρμόζεται κάποιος *αλγόριθμος χρονοδρομολόγησης* ο οποίος αποφασίζει ποια θα καταλαμβάνει την κεντρική μονάδα επεξεργασίας (ΚΜΕ). Συνήθως μια διεργασία δεν εκτελείται αδιαίρετα, αλλά τμηματικά. Κάνει κάποιες ενέργειες στο χρόνο ΚΜΕ που της αναλογεί και μετά περιμένει τις υπόλοιπες διεργασίες να κάνουν το ίδιο. Όταν ξαναέρθει η σειρά της συνεχίζει από κει που σταμάτησε. Για να επιτευχθεί ο αμοιβαίος αποκλεισμός όμως, πρέπει το τμήμα κώδικα της διεργασίας που προσπελαύνει το κοινό δεδομένο να εκτελείται αδιαίρετα. Τέτοια τμήματα κώδικα ονομάζονται *κρίσιμα τμήματα*. Έχουν προταθεί διάφοροι τρόποι για να επιτευχθεί ο αμοιβαίος αποκλεισμός μεταξύ παράλληλων διεργασιών όπως οι σημαφόροι και οι επόπτες [38].

Στο παρόν κεφάλαιο ασχολούμαστε με την εξής εφαρμογή: έχουμε ένα δεδομένο το οποίο προσπελαύνουν τέσσερις διεργασίες, τρεις που το διαβάζουν (readers) και μία που το ενημερώνει (writer). Οι readers μπορούν να διαβάζουν το δεδομένο ταυτόχρονα αλλά όταν το ενημερώνει ο writer πρέπει να έχει αποκλειστική πρόσβαση. Στις επόμενες παραγράφους παρουσιάζονται οι υλοποιήσεις σε καθεμιά απ' τις γλώσσες και οι μηχανισμοί που χρησιμοποιεί κάθε γλώσσα για να πετύχει τον αμοιβαίο αποκλεισμό.

6.2 Υλοποίηση και σχολιασμός των αποτελεσμάτων

6.2.1 Java

Στη Java, n διεργασίες που εκτελούνται παράλληλα καταλαμβάνουν n *νήματα* (threads). Κάθε τέτοια διεργασία είναι στιγμιότυπο μιας υποκλάσης της `java.lang.Thread` και υλοποιεί τη μέθοδο run. Οι run μέθοδοι μπορούν να εκτελούνται παράλληλα και έτσι επιτυγχάνεται η ταυτόχρονη εκτέλεση πολλών διεργασιών.[1] Όταν ένα νήμα A που τρέχει θέλει να γίνει κάποια ενέργεια από άλλα νήματα πριν συνεχίσει την εκτέλεσή του, καλεί τη μέθοδο wait. Τότε σταματάει η εκτέλεση του νήματος A μέχρι κάποιο άλλο νήμα B να καλέσει τη μέθοδο notify για το A ή τη notifyAll για όλα τα νήματα που περιμένουν.

[1] Ο άλλος τρόπος επίτευξης παραλληλίας είναι όταν οι κλάσεις υλοποιούν τη διαπροσωπεία Runnable.

Σε περίπτωση που τα παράλληλα νήματα ενεργούν σε κοινά δεδομένα χρησιμοποιείται η λέξη-κλειδί **synchronized**. Για παράδειγμα, ας υποθέσουμε ότι δύο νήματα Α και Β καλούν μεθόδους ενός αντικειμένου Γ. Κάθε μέθοδος του Γ που χαρακτηρίζεται synchronized μπορεί να εκτελείται από ένα μόνο νήμα κάθε χρονική στιγμή. Επιπλέον, όταν το Α καλεί μια synchronized μέθοδο, τότε το Γ κλειδώνεται και το Β δεν μπορεί να καλέσει καμία synchronized μέθοδο του Γ πριν ξεκλειδωθεί. Έτσι λοιπόν, τα κρίσιμα τμήματα μιας εφαρμογής που περιλαμβάνει πρόσβαση σε κοινά δεδομένα μπορούν να μπουν σε synchronized μεθόδους για να εξασφαλιστεί η ακεραιότητά τους.

```
public class Multi_Access {

        public static void main(String[] args) throws Exception{
               Sensitive_Data the_data = new Sensitive_Data();
               MA_Writer writer  = new MA_Writer(the_data);
               MA_Reader reader1 = new MA_Reader(the_data, 1);
               MA_Reader reader2 = new MA_Reader(the_data, 2);
               MA_Reader reader3 = new MA_Reader(the_data, 3);
               writer.start();
               reader1.start();
               reader2.start();
               reader3.start();
               Thread.sleep(10000);
               System.exit(0);
        }
}
```

Εδώ φαίνεται η συνάρτηση main της εφαρμογής. Αρχικά δημιουργεί ένα αντικείμενο της κλάσης Sensitive_Data που είναι η κλάση που περιέχει το κοινό δεδομένο (ένα αλφαριθμητικό που λέγεται data). Μετά δημιουργεί τρεις reader και ένα writer. Οι κλάσεις MA_Reader και MA_Writer είναι υποκλάσεις της Thread. Καλώντας τη μέθοδο start η main ξεκινάει τα τέσσερα νήματα. Τέλος, περιμένει δέκα δευτερόλεπτα και η εφαρμογή τερματίζει. Παρατηρήστε ότι η main είναι και αυτή ένα ξεχωριστό νήμα εκτέλεσης. Για να περιμένει όσο τα άλλα νήματα δρουν στο αντικείμενο the_data, η main καλεί τη μέθοδο sleep η οποία αδρανοποιεί το καλούν νήμα για όσα milliseconds λέει το όρισμά της.

Παρακάτω φαίνονται οι μέθοδοι run των reader και του writer. Το μόνο που κάνουν είναι να περιμένουν για ένα τυχαίο χρονικό διάστημα προτού ξαναπροσπαθήσουν να διαβάσουν ή να γράψουν. Μετά τυπώνουν στην οθόνη την τρέχουσα τιμή του δεδομένου.

```
// Η μέθοδος MA_Reader.run
public void run() {
      try {
         while (true){
             // περιμένει ένα τυχαίο διάστημα μεταξύ διαβασμάτων
             sleep((long)(500*Math.random()));
             // διαβάζει και τυπώνει στην οθόνη ποιος
             // reader είναι και τι διάβασε
             data.read(whatIread);
             System.out.println("Id " + myId + ": " + whatIread);
         }
      }
      catch (InterruptedException e){System.exit(-1);}
}

// Η μέθοδος MA_Writer.run
public void run() {
      try {
         while (true){
```

```
                // περιμένει ένα τυχαίο διάστημα μεταξύ γραψιμάτων
                sleep((long)(500*Math.random()));
                data.write("I am writing");
            }
        }
        catch (InterruptedException e){System.exit(-1);}
    }
```

Οι μέθοδοι Sensitive_Data.read και Sensitive_Data.write δεν είναι synchronized. Αυτό συμβαίνει γιατί:

α) Θέλουμε οι reader να μπορούν να διαβάζουν ταυτόχρονα, άρα η read δεν μπορεί να είναι synchronized.

β) Ο writer πρέπει να δρα μόνος του στο δεδομένο αλλά αυτό δεν μπορεί να επιτευχθεί όταν η write είναι synchronized. Αυτό συμβαίνει γιατί κατά τη διάρκεια μιας synchronized μεθόδου ναι μεν δεν μπορούν να κληθούν άλλες synchronized μέθοδοι του ίδιου αντικειμένου, αλλά οι υπόλοιπες μέθοδοι, άρα και η read, μπορούν να κληθούν.

Για να πετύχουμε τον αμοιβαίο αποκλεισμό χρησιμοποιούμε τη synchronized μέθοδο mayI, την οποία καλούν οι read και η write. Βάση της τιμής της μεταβλητής status η mayI επιστρέφει true αν μπορεί να γίνει η ζητούμενη ενέργεια και false αν δεν μπορεί. Η τιμή της status αλλάζει μόνο μέσα στη mayI έτσι η ακεραιότητα των δεδομένων είναι εξασφαλισμένη.

```
// Η μέθοδος Sensitive_Data.mayI
private synchronized boolean mayI(String action){
       if (action.compareTo("read")==0) {
            if (status > -1) {status++; return true;}
            else {return false;}
       }
       else {
            if (status > 0) {return false;}
            else {status--; return true;}
       }
}

// Η μέθοδος Sensitive_Data.read
public void read(StringBuffer data)throws InterruptedException{
       if (!mayI("read")) {
            data = data.delete(0,data.length()).append("could not read");
            return;
       }
       Thread.sleep(125);
       data = data.delete(0,data.length()).append(Sensitive_Data.data);
       status--;
}

// Η μέθοδος Sensitive_Data.write
public void write(String newdata) throws InterruptedException{
       if (!mayI("write")) return;
       data = data.delete(0,data.length()).append(newdata);
       Thread.sleep(250);
       data = data.delete(0,data.length()).append("OK");
       status++;
}
```

6.2.2 C#

Η υλοποίηση στη C# έχει την ίδια δομή και οργάνωση κλάσεων και μεθόδων με αυτή στη Java. Ο χειρισμός των νημάτων όμως στη C# είναι λίγο διαφορετικός από τη Java. Όπως είδαμε, στη Java για να φτιαχτεί ένα καινούριο νήμα πρέπει να δημιουργηθεί μια κλάση που κληρονομεί από τη Thread και υλοποιεί τη μέθοδο run. Στη C#, η κλάση Thread είναι *σφραγισμένη* (sealed) δηλαδή δεν μπορεί να έχει υποκλάσεις. Έτσι, για να φτιάξουμε ένα καινούριο νήμα πρέπει να δημιουργήσουμε ένα αντικείμενο της κλάσης Thread και να περάσουμε σαν όρισμα τη μέθοδο που θέλουμε να εκτελεί το νήμα. Καλώντας τη Start, το νήμα θα αρχίσει να εκτελεί τη μέθοδο που περάσαμε σαν όρισμα κατά τη δημιουργία του. Για παράδειγμα, δείτε πως δημιουργείται το Thread του writer.

```
SensitiveData the_data = new SensitiveData();
Writer writer  = new Writer(the_data);
Thread threadW = new Thread(new ThreadStart(writer.StartWriting));
threadW.Start();
```

Παρακάτω φαίνεται η StartWriting, που αντιστοιχεί στην MA_Writer.run της Java.

```
public void StartWriting() {
     try {
        while (true){
             Thread.Sleep(MultiAccessMainThread.RandomWaiting.Next(0,500));
             data.Write("I am writing");
        }
     }
     catch (Exception e) {Console.WriteLine("Writer " + e.Message);}
}
```

Με παρόμοιο τρόπο υλοποιήθηκαν η κλάση Reader και η μέθοδος StartReading. Για να επιτευχθεί ο αμοιβαίος αποκλεισμός, η C# χρησιμοποιεί τη λέξη-κλειδί `lock`, που αντιστοιχεί στη synchronized της Java. Για παράδειγμα, η μέθοδος MayI σε C# θα ήταν κάπως έτσι:

```
private bool MayI(String action){
   lock(this)
   {
      if (action.CompareTo("read")==0) {
         if (status > -1) {status++; return true;}
         else return false;
      }
      else {
         if (status>0) return false;
         else {status--; return true;}
      }
   }
}
```

Εκτός απ' αυτό όμως, η C# παρέχει την κλάση `Interlocked`, που οι μέθοδοί της μπορούν να τροποποιούν μια μεταβλητή έχοντας αποκλειστική πρόσβαση σε αυτή. Έτσι, δε χρειαζόμαστε τη μέθοδο MayI αφού οι μέθοδοι SensitiveData.Read και SensitiveData.Write μπορούν να αλλάζουν την τιμή του status αποκλείοντας αμοιβαία η μία την άλλη, καλώντας μεθόδους από την κλάση Interlocked.

```
public void Read(out String data){
     if (Interlocked.CompareExchange(ref status,-2,0) == -1)
     {
        data = "could not read";
        return;
```

```
        }
        // αύξηση της status
        if (Interlocked.CompareExchange(ref status,1,-2) != -2)
           Interlocked.Increment(ref status);
        Thread.Sleep(125);
        data = SensitiveData.data.ToString();
        Interlocked.Decrement(ref status);
    }

    public void Write(String newData){
        if (Interlocked.CompareExchange(ref status,-1,0) != 0)
           return;
        SensitiveData.data.Replace(SensitiveData.data.ToString(),newData);
        Thread.Sleep(250);
        SensitiveData.data.Replace(SensitiveData.data.ToString(),"OK");
        Interlocked.Increment(ref status);
    }
```

Η μέθοδος Increment (Decrement) αυξάνει (μειώνει) την τιμή του status έχοντας αποκλειστική πρόσβαση σε αυτό. Η CompareExchange συγκρίνει την πρώτη παράμετρο με την τρίτη και αν είναι ίσες τότε η τιμή της δεύτερης παραμέτρου ανατίθεται στην πρώτη παράμετρο. Η τιμή που επιστρέφει είναι η τιμή της πρώτης παραμέτρου πριν από την αλλαγή. Παρατηρήστε την περίπτωση που το status είναι 0: Ο writer ή κάποιος reader θα προλάβει να εκτελέσει την CompareExchange.[2] Αν προλάβει ο writer τότε το status θα γίνει -1 και οι readers δε θα μπορέσουν να διαβάσουν. Αν προλάβει κάποιος reader, τότε το status θα γίνει -2 και ο writer θα επιστρέψει χωρίς να γράψει. Ο λόγος που το status δε γίνεται 1 είναι γιατί μετά τη γραμμή `// αύξηση της status` δε θα ξέραμε αν έχει ήδη ενημερωθεί η τιμή του ή όχι και δε θα μπορούσαμε να καλέσουμε την Increment.

6.2.3 Apollo

Η Apollo μπορεί να έχει ένα μόνο νήμα εκτέλεσης. Για να υπάρχει μια ψευδο-παραλληλία χρησιμοποιήσαμε την εξής μέθοδο: Κάθε reader έχει μια συνάρτηση για να ξεκινάει το διάβασμα και μια για να σταματάει. Το ίδιο γίνεται και με τον writer. Έτσι, μπορεί να έχει κληθεί η StartRead από κάποιον reader και μετά να κληθεί η StartWrite πριν κληθεί η StopRead. Το κοινό δεδομένο είναι μια πλειάδα με τύπο <string,int> όπου το πρώτο στοιχείο είναι το αλφαριθμητικό που διαβάζεται και γράφεται και το δεύτερο στοιχείο είναι η μεταβλητή ελέγχου (αντιστοιχία με τη status). Όταν η τιμή της είναι 0 τότε κανένας χρήστης δεν είναι ενεργός, όταν είναι από 1 ως 3 είναι ενεργοί οι readers και όταν είναι 4 είναι ενεργός ο writer. Η κύρια συνάρτηση του προγράμματος είναι:

```
Main =
poly user. fun data:<string,int>, StartOrStop:bool, AnyNewData:string.
if 'user<:'reader then
  if StartOrStop then
    StartRead (MayI data 0) data
  else
    StopRead data
else if 'user<:'writer then
  if StartOrStop then
    StartWrite (MayI data 1) data AnyNewData
  else
    StopWrite data
else error "unknown user"
```

[2] αφού, όπως συμβαίνει και με τις υπόλοιπες μεθόδους της Interlocked, όποιο νήμα την εκτελεί έχει αποκλειστική πρόσβαση σε αυτή.

Οι χρήστες προφανώς υλοποιούνται ως principals. Η Main είναι πολυμορφική συνάρτηση και αρχικοποιείται με τον τύπο του principal που την καλεί. Αν το StartOrStop είναι true τότε ο καλών χρήστης θέλει να αρχίσει διάβασμα ή γράψιμο ενώ όταν είναι false τότε θέλει να σταματήσει. Το AnyNewData λαμβάνεται υπόψη μόνο όταν ο καλών χρήστης είναι ο writer. Η Main καλεί τη συνάρτηση MayI που επιστρέφει ένα capability το οποίο επιτρέπει ή απαγορεύει στο χρήστη να επεξεργαστεί το δεδομένο.

```
MayI=
fun data:<string,int>, action:int.
            // action=0 για διάβασμα, action=1 για γράψιμο
if action=0 then
  if data.2 = 4 then
     acquire 'reader #> 'IllegalAction
  else
     acquire 'reader #> 'PermissionToRead
else if action=1 then
  if data.2 = 0 then
     acquire 'writer #> 'PermissionToWrite
  else
     acquire 'writer #> 'IllegalAction
else error "unknown action"
```

Παραπάνω φαίνεται ότι η MayI επιστρέφει το κατάλληλο capability ανάλογα με την τιμή της μεταβλητής ελέγχου. Τα δικαιώματα IllegalAction, PermissionToRead και PermissionToWrite τα ορίσαμε εμείς· η σημασιολογία τους φαίνεται από το όνομά τους. Όπως φαίνεται στη Main, το capability που επιστρέφει η MayI περνιέται σαν όρισμα στις StartRead και StartWrite:

```
StartRead =
fun c:cert, data:<string,int>.
if c => 'reader #> 'PermissionToRead then
  <data.1, data.2 + 1>
else data

StartWrite =
fun c:cert, data:<string,int>, newValue:string.
if c => 'writer #> 'PermissionToWrite then
  <newValue, 4>
else data
```

Η StartRead ελέγχει το capability και αν επιτρέπεται το διάβασμα τότε απλά αυξάνει το πλήθος των readers κατά ένα. Η StartWrite αντικαθιστά το παλιό αλφαριθμητικό με το καινούριο και αλλάζει κατάλληλα την τιμή της μεταβλητής ελέγχου. Όταν δεν επιτρέπεται η πρόσβαση στο δεδομένο και οι δυο συναρτήσεις επιστρέφουν το data αναλλοίωτο. Οι StopRead και StopWrite το μόνο που κάνουν είναι να αλλάζουν την τιμή της μεταβλητής ελέγχου. Χρησιμοποιώντας τις παραπάνω συναρτήσεις, μια πιθανή ακολουθία εκτέλεσης είναι:

```
Main [read2] (
   Main [read1] (
         Main [read2] (
               Main [writer]
                     (Main [read1] <"first value",0> true "does not matter")
               true "ok")
         true "does not matter")
   false "does not matter")
false "does not matter"
```

Πρώτα αρχίζει να διαβάζει ο reader 1. Μετά ο writer προσπαθεί να αρχίσει γράψιμο αλλά αποτυγχάνει γιατί ο reader 1 δεν έχει τελείωσει το διάβασμα. Έπειτα αρχίζει διάβασμα και

ο reader 2 παράλληλα με τον πρώτο και τέλος οι δύο readers σταματούν το δίαβασμα και η τιμή "first value" του δεδομένου παραμένει η ίδια, αφού δεν ανανεώθηκε ποτέ από το writer. Η Apollo δεν περιλαμβάνει αναδρομή και προγράμματα που δεν τερματίζουν, ούτε γεννήτρια τυχαίων αριθμών. Έτσι δεν μπορούμε να έχουμε μεγάλες τυχαίες ακολουθίες εκτέλεσης όπως στις Java και C#. Παρ' όλα αυτά, χρησιμοποιώντας σύνταξη όπως την παραπάνω μπορούμε να εκφράσουμε (ντετερμινιστικά) οποιαδήποτε πεπερασμένη ακολουθία εκτέλεσης.

6.2.4 Σχόλια

Οι Java και C# χρησιμοποιώντας τα νήματα μπορούν να πετύχουν παράλληλη εκτέλεση διεργασιών που είτε είναι ανεξάρτητες μεταξύ τους ή δρουν σε κοινά δεδομένα. Αυτό δε συμβαίνει στην Apollo, όπου έπρεπε να "εφεύρουμε" ένα μηχανισμό για να προσομοιώσουμε την παραλληλία. Το γεγονός ότι δεν μπορούμε να έχουμε ξεχωριστά νήματα στην Apollo μας απαγορεύει να εκφράσουμε πιο πολύπλοκες περιπτώσεις παραλληλίας όπως όταν ένα νήμα αναστέλλει τη λειτουργία του (μέθοδος wait) μέχρι κάποιο άλλο νήμα να ολοκληρώσει μια ενέργεια. Από την άλλη, χρησιμοποιώντας μόνο τα principals και τα δικαιώματα στην Apollo καταφέραμε να υλοποιήσουμε τρεις εφαρμογές διαφορετικής φύσης ενώ στις άλλες δυο γλώσσες χρησιμοποιήσαμε άλλες κλάσεις σε κάθε εφαρμογή. Τέλος, πρέπει να αναφέρουμε στα υπέρ της C# έναντι της Java ότι παρέχει την κλάση Interlocked η οποία διευκολύνει την υλοποίηση του αμοιβαίου αποκλεισμού.

Κεφάλαιο 7

Συμπεράσματα

Στην παρούσα εργασία έγινε μια προσπάθεια συγκριτικής αποτίμησης των μηχανισμών ασφάλειας που παρέχονται από τις εμπορικές γλώσσες προγραμματισμού και των αντίστοιχων μηχανισμών που έχουν προταθεί από την ακαδημαϊκή κοινότητα.

Γλώσσες όπως η Java και η C# έχουν συμβάλει στη βελτίωση του επιπέδου ασφαλείας των εφαρμογών που χρησιμοποιούνται σήμερα στο διαδίκτυο. Το σύστημα ασφαλείας που υποστηρίζουν μπορεί χονδρικά να διαχωριστεί σε τρία επίπεδα:

- πιστοποίηση της πηγής προέλευσης του εκτελέσιμου κώδικα (μέσω κρυπτογραφικών μηχανισμών κ.λπ.).
- παραχώρηση κατάλληλων δικαιωμάτων στον κώδικα ανάλογα με το βαθμό εμπιστοσύνης που έχουμε για την πηγή προέλευσής του και απομόνωση των διαφορετικών εφαρμογών που εκτελούνται παράλληλα βάση της λογικής του αμμοδοχείου.
- γλώσσα με ασφαλές σύστημα τύπων για να αποφευχθούν φαινόμενα όπως η υπερχείλιση/υποχείλιση στοίβας, η πλαστογράφηση δεικτών κ.λπ.

Παρ' όλο που τα επίπεδα αυτά παρέχουν ικανοποιητικό βαθμό ασφάλειας, οι γλώσσες αυτές αφήνουν μεγάλο μέρος των ελέγχων να γίνει από τη νοητή μηχανή σε χρόνο εκτέλεσης με αναπόφευκτη μείωση στην απόδοση. Επιπλέον, είναι εξαιρετικά δύσκολο να επαληθευτούν θεωρητικά οι ιδιότητες ασφάλειας, αφού πρέπει να εμπιστευτούμε τα μεγάλα μη-πιστοποιημένα κομμάτια κώδικα που αποτελούν τη νοητή μηχανή (φορτωτής, ελεγκτής, μεταφραστής ή μεταγλωττιστής τελευταίας στιγμής και βιβλιοθήκη χρόνου εκτέλεσης).

Από την άλλη, η ακαδημαϊκή κοινότητα έχει προτείνει λύσεις που μπορούν να επαληθευτούν θεωρητικά και επιπλέον παρέχουν μεγαλύτερη απόδοση. Αυτό συμβαίνει γιατί, χρησιμοποιώντας ισχυρά συστήματα τύπων, οι γλώσσες αυτές καταφέρνουν να:

- ενσωματώνουν μεγάλο μέρος των ελέγχων στην ίδια τη γλώσσα και όχι να τους αφήνουν για κάποιο εξωτερικό σύστημα όπως η νοητή μηχανή.
- πραγματοποιούν ελέγχους στο επίπεδο των τύπων, πράγμα που σημαίνει ότι οι έλεγχοι αυτοί θα γίνουν από τον ελεγκτή των τύπων *σε χρόνο μεταγλώττισης*, επιτρέποντας έτσι την αύξηση της απόδοσης.
- παρέχουν ένα σύστημα ασφάλειας που αποδεικνύεται ορθό θεωρητικά.

Οι γλώσσες προγραμματισμού που χρησιμοποιήθηκαν στην υλοποίηση των εφαρμογών μας είναι οι Java, C# και Apollo. Συγκρίνοντας τη Java με τη C#, παρατηρούμε ότι έχουν αρκετές ομοιότητες, από τη σύνταξή τους μέχρι το σύστημα ασφαλείας των νοητών μηχανών που τις εκτελούν. Ένα πλεονέκτημα του πλαισίου .NET σε σχέση με την πλατφόρμα Java είναι ότι υποστηρίζει πολλές γλώσσες προγραμματισμού, διευκολύνοντας έτσι τους προγραμματιστές αφού μπορούν να υλοποιούν τα τμήματα μιας εφαρμογής σε διαφορετικές γλώσσες και να εκμεταλλεύονται τα προτερήματα της καθεμιάς. Η Apollo, χρησιμοποιώντας σύστημα τύπων με ροή πληροφορίας, παρουσιάζει αρκετές διαφορές από τις δύο παραπάνω γλώσσες. Οι

έλεγχοι γίνονται στατικά σε μεγάλο βαθμό και σε χρόνο εκτέλεσης γίνονται μόνο οι έλεγχοι που αφορούν τις τιμές των run-time principals και τα capabilities που παρέχονται. Επίσης υπάρχει διαφορά στην οργάνωση και τη διάταξη του κώδικα αφού οι Java και C# είναι αντικειμενοστρεφείς γλώσσες ενώ η Apollo συναρτησιακή.

Συμπερασματικά, παρατηρούμε ότι είναι δυνατή η περαιτέρω βελτίωση των εμπορικών συστήματων αν ενσωματώσουν στοιχεία των ακαδημαϊκών γλωσσών. Για παράδειγμα, μια εμπορική γλώσσα υψηλού επιπέδου θα μπορούσε να μεταγλωττίζεται σε μια TIL γλώσσα όπως η CL, καταργώντας την ανάγκη για συλλέκτη σκουπιδιών και πραγματοποιώντας τη διαχείριση της μνήμης στατικά. Επίσης, αν ενσωματωθεί ένα σύστημα τύπων με ροή πληροφορίας σε μια εμπορική γλώσσα προγραμματισμού, οι τύποι των δεδομένων (με ετικέτες) θα αντιπροσωπεύουν και τις άδειες πρόσβασης σε αυτά. Έτσι, θα μειωθεί η ανάγκη για παραχώρηση δικαιωμάτων από το σύστημα ανάλογα με την εμπιστοσύνη που έχουμε στην πηγή προέλευσης του κώδικα. Τέλος, η χρήση μιας γλώσσας για επόπτες όπως η Polymer επιτρέπει να εφαρμόζουμε σύνθετες πολιτικές ασφαλείας σε χρόνο εκτέλεσης συνδυάζοντας απλούστερες πολιτικές, οι οποίες επαληθεύονται θεωρητικά.

Απομένει να δούμε κατά πόσον η βιομηχανική και η ακαδημαϊκή προσέγγιση στο θέμα της ασφάλειας, που για την ώρα είναι ανεξάρτητες και ριζικά διαφορετικές, θα συγκλίνουν στο μέλλον. Ο συγκερασμός τους σε μία κοινή πλατφόρμα ανάπτυξης εφαρμογών θα ήταν ιδιαίτερα ωφέλιμος, καθώς θα συνδύαζε τα πλεονεκτήματα των δύο προσεγγίσεων.

Παράρτημα Α

Πλήρης κώδικας των εφαρμογών

Α.1 Κώδικας Java

Α.1.1 Needham-Schroeder public key protocol

```
// class nspk
import java.security.*;

public class nspk {
           public static void communicate (Initiator alice, Responder bob)
              throws GeneralSecurityException
           {
              alice.initiate(bob);
           }

           public static void main (String[] args)
           {
              try {
                 Initiator alice = new Initiator("Alice");
                 Responder bob = new Responder("Bob");
                 communicate(alice, bob);
              }
              catch (GeneralSecurityException exc) {
                 System.err.println(exc.getMessage());
              }
              catch (RuntimeException exc) {
                 System.err.println(exc.getMessage());
              }
           }
}

// class Nonce
public class Nonce{
           public static final int SIZE = 8;
           private byte[] data;

           public Nonce ()
           {
              data = new byte[SIZE];
              for (int i = 0; i < SIZE; i++)
                 data[i] = (byte) (256 * Math.random());
           }

           public Nonce (byte[] data)
           {
              this.data = data;
           }
```

```
            public boolean equals (Nonce n)
            {
               for (int i = 0; i < SIZE; i++)
                  if (data[i] != n.data[i])
                     return false;
               return true;
            }

            protected byte[] getBytes ()
            {
               return data;
            }

            public String toString ()
            { // το κάθε byte έχει τιμή από 0 ως 255 οπότε χρειάζεται 2 δεκαεξαδικά
              // ψηφία για να περιγραφεί.
               byte[] hex = new byte[2*SIZE];
               int len = 0;
               for (int i = 0; i < SIZE; i++) {
                  int b = data[i];
                  if (b < 0) b += 256;
                  hex[len++] = (byte) Character.forDigit(b / 16, 16);
                  hex[len++] = (byte) Character.forDigit(b % 16, 16);
               }
               return new String(hex);
            }
}

// class Message
import java.io.*;
import java.security.*;
import javax.crypto.*;

public class Message extends Nonce{
            private byte[] data;
            private int    written;
            private int    read;

            public Message (int size)
            {
               data = new byte[size];
               written = 0;
               read = 0;
            }

            public Message (byte[] data)
            {
               this.data = data;
               written = data.length;
               read = 0;
            }

            // παραθέτει στο τέλος του Message.data το όρισμα της συνάρτησης
            public void append (byte[] data)
               throws RuntimeException
            {
               if (written + data.length <= this.data.length)
```

```
            for (int i = 0; i < data.length; i++)
               this.data[written++] = data[i];
         else
            throw new RuntimeException("Message buffer overflow");
      }

      // επιστρέφει το data αν read==0 ή μέρος του data αν read>0
      public byte[] read ()
      {
         int size = written - read;
         byte[] result = new byte[size];
         for (int i = 0; i < size; i++)
            result[i] = data[read++];
         return result;
      }

      // επιστρέφει "size" αδιάβαστα bytes του data
      public byte[] read (int size)
         throws RuntimeException
      {
         if (read + size <= written) {
            byte[] result = new byte[size];
            for (int i = 0; i < size; i++)
               result[i] = data[read++];
            return result;
         }
         else
            throw new RuntimeException("Message buffer underflow");
      }

      public Message encrypt (Cipher cipher, Key key)
         throws GeneralSecurityException
      {
         cipher.init(Cipher.ENCRYPT_MODE, key);
         return new Message(cipher.doFinal(data));
      }

      public Message decrypt (Cipher cipher, Key key)
         throws GeneralSecurityException
      {
         cipher.init(Cipher.DECRYPT_MODE, key);
         return new Message(cipher.doFinal(data));
      }
}

// class Initiator
import java.io.*;
import java.security.*;

import javax.crypto.*;

public class Initiator
{
   private String     id;
   private PrivateKey priv;
   public  PublicKey  publ;
   private Cipher     rsaCipher;
   private Nonce anonce;
```

```
   private String bid;

   public Initiator (String id)
      throws GeneralSecurityException
   {
        this.id = id;
    KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
    keyGen.initialize(1024);
    KeyPair pair = keyGen.generateKeyPair();
    priv = pair.getPrivate();
    publ = pair.getPublic();
    rsaCipher = Cipher.getInstance("RSA");
    System.out.println(id + ":\tcreated and initialized successfully");
   }

   public void initiate (Responder b)
      throws GeneralSecurityException
   {
      bid = b.getId();
      System.out.println(id + ":\tinitiating communication with " + bid);
      anonce = new Nonce();
      System.out.println(id + ":\tmy nonce is " + anonce.toString());
      Message clearSnd = new Message(Nonce.SIZE + id.length());
      clearSnd.append(anonce.getBytes());
      clearSnd.append(id.getBytes());
      System.out.println(id + ":\tsending (" + anonce.toString() + ", " + id + ")");
      Message encryptedSnd = clearSnd.encrypt(rsaCipher, b.publ);
      b.message1(this, encryptedSnd);
   }

         public void message2 (Responder b, Message encryptedRcv)
            throws GeneralSecurityException
         {
      Message clearRcv = encryptedRcv.decrypt(rsaCipher, priv);
      Nonce anonce = new Nonce(clearRcv.read(Nonce.SIZE));
      Nonce bnonce = new Nonce(clearRcv.read(Nonce.SIZE));
      String bid = new String(clearRcv.read());
      System.out.println(id + ":\treceived (" + anonce.toString() + ", "
                + bnonce.toString() + ", " + bid + ")");
      if (!anonce.equals(this.anonce)) {
        String msg = id + ": the nonce you sent me back is not mine!";
        throw new GeneralSecurityException(msg);
      }
      else
         System.out.println(id + ":\teverything ok for me");
         Message clearSnd = new Message(Nonce.SIZE);
         clearSnd.append(bnonce.getBytes());
         System.out.println(id + ":\tsending (" + bnonce.toString() + ")");
         Message encryptedSnd = clearSnd.encrypt(rsaCipher, b.publ);
         b.message3(this, encryptedSnd);
   }
}

// class Responder
import java.io.*;
import java.security.*;

import javax.crypto.*;
```

```
public class Responder
{
   private String     id;
   private PrivateKey priv;
   public    PublicKey  publ;
   private Cipher     rsaCipher;
   private Nonce bnonce;


   public Responder (String id)
      throws GeneralSecurityException
   {
        this.id = id;
        KeyPairGenerator keyGen = KeyPairGenerator.getInstance("RSA");
        keyGen.initialize(1024);
        KeyPair pair = keyGen.generateKeyPair();
        priv = pair.getPrivate();
        publ = pair.getPublic();
        rsaCipher = Cipher.getInstance("RSA");
        System.out.println(id + ":\tcreated and initialized successfully");
   }

   public String getId ()
   {
      return id;
   }

   public void message1 (Initiator a, Message encryptedRcv)
      throws GeneralSecurityException
   {
      Message clearRcv = encryptedRcv.decrypt(rsaCipher, priv);
      Nonce anonce = new Nonce(clearRcv.read(Nonce.SIZE));
      String aid = new String(clearRcv.read());
      System.out.println(id + ":\treceived (" + anonce.toString() + ", " + aid + ")");
      bnonce = new Nonce();
      System.out.println(id + ":\tmy nonce is " + bnonce.toString());
      Message clearSnd = new Message(2*Nonce.SIZE + id.length());
      clearSnd.append(anonce.getBytes());
      clearSnd.append(bnonce.getBytes());
      clearSnd.append(id.getBytes());
      System.out.println(id + ":\tsending (" + anonce.toString() + ", "
                + bnonce.toString() + ", " + id + ")");
      Message encryptedSnd = clearSnd.encrypt(rsaCipher, a.publ);
      a.message2(this, encryptedSnd);
   }

   public void message3 (Initiator a, Message encryptedRcv)
            throws GeneralSecurityException
   {
      Message clearRcv = encryptedRcv.decrypt(rsaCipher, priv);
      Nonce bnonce = new Nonce(clearRcv.read(Nonce.SIZE));
      System.out.println(id + ":\treceived (" + bnonce.toString() + ")");
      if (!bnonce.equals(this.bnonce)) {
         String msg = id + ": the nonce you sent me back is not mine!";
         throw new GeneralSecurityException(msg);
      }
```

```
      else
         System.out.println(id + ":\teverything ok for me");
   }
}
```

A.1.2 Ηλεκτρονικές Τραπεζικές Συναλλαγές

```
// class Money_app
public class Money_app {

        public static void main(String[] args) throws Exception{
                ATM atm=new ATM();
                while (true) {
                        atm.transaction();
                        if (wantsAnother()) continue; else break;
                }
                System.exit(0);
        }

        private static boolean wantsAnother() throws Exception{
                int[] input = new int[3];
                int i;
                System.out.println("If you want another transaction " +
                                        "press \'y\' and then \"enter\"");
                for (i=0;i<3;i++)
                        input[i] = System.in.read();
                if (input[0]=='y' && input[1]==13 && input[2]==10) return true;
                else return false;
        }
}

// class ATM
import java.io.IOException;

public class ATM {
        private int password,sum;

        public void transaction() throws IOException{
                password = login();
                sum = request();
                System.out.println(Bank.message(password,sum));
        }


        private int login()throws IOException{
                byte[] code = new byte[20];
                int i,input,flag;

                System.in.skip(80);
                while (true) {
                        i=0; flag=0;
                        System.out.println("Please enter your code and" +
                                                "press \"enter\" to login.");
                        while ((input = System.in.read())!= '\n') {
                                if (!Character.isDigit((char)input) && (input!=10)
                                        && (input!=13)) flag=1;
                                code[i++]= (byte)input;
                        }
```

```
                    System.in.skip(80);
                    if (flag==1) {System.out.println("Give digits only, try again.");
                                  continue;}
                    if (i==5) break;
                    System.out.println("You must enter a 4-digit code," +
                         " please try again");
                }
                return 1000*(code[0]-48)+100*(code[1]-48)+10*(code[2]-48)+code[3]-48;
        }

        private int request() throws IOException{
            int i,input,flag,sum;

            System.in.skip(80);
            while (true){
                System.out.println("Press 1 to withdraw money and 2 to " +
                        "check your account balance and press \"enter\"");
                flag=0; i=0;
                while ((input = System.in.read())!= 10 && input!=13) {
                     i++;
                     if (input!=49 && input!=50)
                         flag=1;
                     else if (input==49)
                         flag=2;
                     else if (input==50)
                         flag=3;
                }
                System.in.skip(80);
                if (flag==1 || i!=1) {
                     System.out.println("Invalid entry, try again");
                     continue;
                }
                else break;
            }
            sum=0;
            while (flag==2) {
                System.out.println("Type the amount of money using 4 digits" +
                           " (type 0's at start if needed) and then \"enter\"");
                sum=0; i=1000; flag=0;
                while ((input = System.in.read())!=10 && input!=13)     {
                    if (Character.isDigit((char)input) && i>=1) {
                       sum += i*(input-48);
                       i/=10;
                    }
                    else flag=1;
                }
                System.in.skip(80);
                if (flag==1 || sum==0 || i>0)
                    {System.out.println("Invalid amount, try again");
                     flag=2; continue;}
                else { break;}
            }
            return sum;
        }
}

// class Bank
public class Bank{
```

```
        private static int[] pelates = {1234,4534,7808,4325,7654};
        private static int[] accounts = {1000,0,15789,9999,350};

        public static String message(int password, int sum){
                int user;
                if ((user = validUser(password))==5) return "There is no such user";
                if (sum==0) return ("You have " + accounts[user] + " dollars left.");
                return setBalance(sum, user);
        }

        private static int validUser(int password){
                int i;
                for (i=0;i<5;i++)
                        if (pelates[i]==password) break;
                return i;
        }

        private static String setBalance(int accountBalance, int user){
                if ((accounts[user]-accountBalance)>=0)
                        {accounts[user]-=accountBalance; return ("Successful. You have "
                                        + accounts[user] + " dollars left.");}
                else return ("Sorry, you only have " + accounts[user] + " dollars left.");
        }
}
```

A.1.3 Αμοιβαίος Αποκλεισμός

```
// class Multi_Access
public class Multi_Access {

        public static void main(String[] args) throws Exception{
               Sensitive_Data the_data = new Sensitive_Data();
               MA_Writer writer  = new MA_Writer(the_data);
               MA_Reader reader1 = new MA_Reader(the_data, 1);
               MA_Reader reader2 = new MA_Reader(the_data, 2);
               MA_Reader reader3 = new MA_Reader(the_data, 3);
               writer.start();
               reader1.start();
               reader2.start();
               reader3.start();
               Thread.sleep(10000);
               System.exit(0);
        }
}

// class MA_Reader
public class MA_Reader extends Thread{
        private Sensitive_Data data;
        private int myId;
        private StringBuffer whatIread;

        public MA_Reader(Sensitive_Data data, int id){
                this.data = data;
                myId = id;
                whatIread = new StringBuffer();
        }
        public void run() {
                try {
```

```
                        while (true){
                                sleep((long)(500*Math.random()));
                                data.read(whatIread);
                                System.out.println("Id " + myId + ": " + whatIread);
                        }
                }
                catch (InterruptedException e){System.exit(-1);}
        }
}

// class MA_Writer
public class MA_Writer extends Thread{
        private Sensitive_Data data;

        public MA_Writer(Sensitive_Data data){
                this.data = data;
        }
        public void run() {
                try {
                        while (true){
                                sleep((long)(500*Math.random()));
                                data.write("I am writing");
                        }
                }
                catch (InterruptedException e){System.exit(-1);}
        }
}

// class Sensitive_Data
public class Sensitive_Data{
        private static StringBuffer data;
        private static int status;

        public Sensitive_Data(){
                data = new StringBuffer("OK");
                status = 0;
        }
        private synchronized boolean mayI(String action){
                if (action.compareTo("read")==0) {
                        if (status > -1) {status++; return true;}
                        else {return false;}
                }
                else {
                        if (status > 0) {return false;}
                        else {status--; return true;}
                }
        }
        public void read(StringBuffer data)throws InterruptedException{
                if (!mayI("read")) {
                        data = data.delete(0,data.length()).append("could not read");
                        return;
                }
                Thread.sleep(125);
                data = data.delete(0,data.length()).append(Sensitive_Data.data);
                status--;
        }
        public void write(String newdata) throws InterruptedException{
                if (!mayI("write")) return;
```

```
                data = data.delete(0,data.length()).append(newdata);
                Thread.sleep(250);
                data = data.delete(0,data.length()).append("OK");
                status++;
        }
}
```

A.2 Κώδικας C#

A.2.1 Needham-Schroeder public key protocol

```
#region Using directives

using System;
using System.Security.Cryptography;

#endregion

namespace NSPK
{
    public class NSPK
    {
        private static void communicate(Initiator alice, Responder bob)
        {
            alice.initiate(bob);
        }

        public static void Main(string[] args)
        {
            try
            {
                Initiator alice = new Initiator("Alice");
                Responder bob = new Responder("Bob");
                communicate(alice, bob);
            }
            catch (ApplicationException exc)
            {
                Console.WriteLine(exc.Message);
            }
        }
    }

    class Nonce
    {
        public const int SIZE = 8;
        private static Random rg = new Random();

        private byte[] data;

        public Nonce()
        {
            data = new byte[SIZE];
            for (int i = 0; i < SIZE; i++)
                data[i] = (byte)(rg.Next(256));
        }

        public Nonce(byte[] data)
        {
```

```
        this.data = data;
    }

    public bool Equals(Nonce n)
    {
        for (int i = 0; i < SIZE; i++)
            if (data[i] != n.data[i])
                return false;
        return true;
    }

    internal byte[] getBytes()
    {
        return data;
    }

    public String toString()
    {
        String result = String.Empty;
        for (int i = 0; i < SIZE; i++)
        {
            int b = data[i];
            if (b < 0) b += 256;
            result += b.ToString("x");
        }
        return result;
    }
}

class Message
{
    private byte[] data;
    private int written;
    private int read;

    public Message(int size)
    {
        data = new byte[size];
        written = 0;
        read = 0;
    }

    public Message(byte[] data)
    {
        this.data = data;
        written = data.Length;
        read = 0;
    }

    public void append(byte[] data)
    {
        if (written + data.Length <= this.data.Length)
            foreach (byte b in data)
                this.data[written++] = b;
        else
            throw new ApplicationException("Message buffer overflow");
    }
```

```
    public void append(char[] data)
    {
        if (written + data.Length <= this.data.Length)
            foreach (char c in data)
                this.data[written++] = (byte) c;
        else
            throw new ApplicationException("Message buffer overflow");
    }

    public char[] extract()
    {
        int size = written - read;
        char[] result = new char[size];
        for (int i = 0; i < size; i++)
            result[i] = (char) data[read++];
        return result;
    }

    public byte[] extract(int size)
    {
        if (read + size <= written)
        {
            byte[] result = new byte[size];
            for (int i = 0; i < size; i++ )
                result[i] = data[read++];
            return result;
        }
        else
            throw new ApplicationException("Message buffer underflow");
    }

    public Message encrypt(RSAParameters key)
    {
        RSACryptoServiceProvider cipher = new RSACryptoServiceProvider();
        cipher.ImportParameters(key);
        return new Message(cipher.Encrypt(data, false));
    }

    public Message decrypt(RSAParameters key)
    {
        RSACryptoServiceProvider cipher = new RSACryptoServiceProvider();
        cipher.ImportParameters(key);
        return new Message(cipher.Decrypt(data, false));
    }
}

class Initiator
{
    private String id;
    private RSAParameters priv;
    public RSAParameters publ;
    private Nonce anonce;
    private String bid;

    public Initiator(String id)
    {
        this.id = id;
        RSACryptoServiceProvider cipher = new RSACryptoServiceProvider(1024);
```

```
        priv = cipher.ExportParameters(true);
        publ = cipher.ExportParameters(false);
        Console.WriteLine("{0}:\tcreated and initialized successfully", id);
    }

    public void initiate(Responder b)
    {
        bid = b.getId();
        Console.WriteLine("{0}:\tinitiating communication with {1}", id, bid);
        anonce = new Nonce();
        Console.WriteLine("{0}:\tmy nonce is {1}", id, anonce.toString());
        Message clearSnd = new Message(Nonce.SIZE + id.Length);
        clearSnd.append(anonce.getBytes());
        clearSnd.append(id.ToCharArray());
        Console.WriteLine("{0}:\tsending ({1}, {2})", id, anonce.toString(), id);
        Message encryptedSnd = clearSnd.encrypt(b.publ);
        b.message1(this, encryptedSnd);
    }

    public void message2(Responder b, Message encryptedRcv)
    {
        Message clearRcv = encryptedRcv.decrypt(priv);
        Nonce anonce = new Nonce(clearRcv.extract(Nonce.SIZE));
        Nonce bnonce = new Nonce(clearRcv.extract(Nonce.SIZE));
        String bid = new String(clearRcv.extract());
        Console.WriteLine("{0}:\treceived ({1}, {2}, {3})",
                          id, anonce.toString(), bnonce.toString(), bid);
        if (!anonce.Equals(this.anonce))
        {
            String msg = id + ": the nonce you sent me back is not mine!";
            throw new ApplicationException(msg);
        }
        else
        Console.WriteLine("{0}:\teverything ok for me", id);
        Message clearSnd = new Message(Nonce.SIZE);
        clearSnd.append(bnonce.getBytes());
        Console.WriteLine("{0}:\tsending ({1})", id, bnonce.toString());
        Message encryptedSnd = clearSnd.encrypt(b.publ);
        b.message3(this, encryptedSnd);
    }
}

class Responder
{
    private String id;
    private RSAParameters priv;
    public RSAParameters publ;
    private Nonce bnonce;

    public Responder(String id)
    {
        this.id = id;
        RSACryptoServiceProvider cipher = new RSACryptoServiceProvider(1024);
        priv = cipher.ExportParameters(true);
        publ = cipher.ExportParameters(false);
        Console.WriteLine("{0}:\tcreated and initialized successfully", id);
    }
```

```
        public String getId()
        {
            return id;
        }

        public void message1(Initiator a, Message encryptedRcv)
        {
            Message clearRcv = encryptedRcv.decrypt(priv);
            Nonce anonce = new Nonce(clearRcv.extract(Nonce.SIZE));
            String aid = new String(clearRcv.extract());
            Console.WriteLine("{0}:\treceived ({1}, {2})", id, anonce.toString(), aid);
            bnonce = new Nonce();
            Console.WriteLine("{0}:\tmy nonce is {1}", id, bnonce.toString());
            Message clearSnd = new Message(2 * Nonce.SIZE + id.Length);
            clearSnd.append(anonce.getBytes());
            clearSnd.append(bnonce.getBytes());
            clearSnd.append(id.ToCharArray());
            Console.WriteLine("{0}:\tsending ({1}, {2}, {3})",
                        id, anonce.toString(), bnonce.toString(), id);
            Message encryptedSnd = clearSnd.encrypt(a.publ);
            a.message2(this, encryptedSnd);
        }

        public void message3(Initiator a, Message encryptedRcv)
        {
            Message clearRcv = encryptedRcv.decrypt(priv);
            Nonce bnonce = new Nonce(clearRcv.extract(Nonce.SIZE));
            Console.WriteLine("{0}:\treceived ({1})", id, bnonce.toString());
            if (!bnonce.Equals(this.bnonce))
            {
                String msg = id + ": the nonce you sent me back is not mine!";
                throw new ApplicationException(msg);
            }
            else
                Console.WriteLine("{0}:\teverything ok for me", id);
        }
    }
}
```

A.2.2 Ηλεκτρονικές Τραπεζικές Συναλλαγές

```
// class MoneyApp
#region Using directives

using System;
using System.Text;

#endregion

namespace AtmSimulator
{
    class MoneyApp
    {

        public static void Main(String[] args)
        {
            Atm atm = new Atm();
```

```
                while (true)
                {
                        atm.Transaction();
                        if (WantsAnother()) continue; else break;
                }
                System.Environment.Exit(0);
        }

        private static bool WantsAnother() {
                int[] input = new int[10];
            int i;
                Console.WriteLine("If you want another transaction " +
                        "press \'y\' and then \"enter\"");
            input[0] = 0;
            for (i=0; i<3; i++)
                input[i] = Console.Read();
            if (input[0] == 'y' && input[1] == 13 && input[2] == 10)
                return true;
            else return false;
        }
    }
}

// class Atm
#region Using directives

using System;
using System.Text;

#endregion

namespace AtmSimulator
{
    public class Atm
    {
     private int password,sum;
     public void Transaction() {
        password = Login();
        sum = Request();
        Console.WriteLine(Bank.Message(password,sum));
     }

     private int Login(){
         byte[] code = new byte[20];
         int i,input,flag;
         while (true) {
              i=0; flag=0;
              Console.WriteLine("Please enter your code and press \"enter\"");
              while ((input = Console.Read())!= '\n') {
                    if (!Char.IsDigit((char)input) && (input!=10)
                            && (input!=13)) flag=1;
                        code[i++]= (byte)input;
                    }
              if (flag==1) {Console.WriteLine("Give digits only, try again.");
                            continue;}
              if (i==5) break;
              Console.WriteLine("You must enter a 4-digit code, please try again");
         }
```

```
        return 1000*(code[0]-48)+100*(code[1]-48)+10*(code[2]-48)+code[3]-48;
    }

    private int Request(){
        int i,input,flag,sum;

        while (true){
            Console.WriteLine("Press 1 to withdraw money and 2 to " +
                 "check your account balance and press \"enter\"");
            flag=0; i=0;
            while ((input = Console.Read())!= 10 && input!=13) {
                  i++;
                  if (input!=49 && input!=50)
                         flag=1;
                  else if (input==49)
                         flag=2;
                  else if (input==50)
                         flag=3;
            }
            if (flag==1 || i!=1) {
                  Console.Read();
                  Console.WriteLine("Invalid entry, try again");
                  continue;
            }
            else {Console.Read(); break;}
        }
        sum=0;
        while (flag==2) {
            Console.WriteLine("Type the amount of money using 4 digits" +
                  " (type 0's at start if needed) and then \"enter\"");
            sum=0; i=1000; flag=0;
            while ((input = Console.Read())!=10 && input!=13)       {
                  if (Char.IsDigit((char)input) && i>=1) {
                      sum += i*(input-48);
                      i/=10;
                  }
                  else flag=1;
            }
            if (flag==1 || sum==0 || i>0)
            {
                  Console.Read();
                  Console.WriteLine("Invalid amount, try again");
                  flag=2;
                  continue;
            }
            else { Console.Read(); break;}
        }
        return sum;
    }
   }
}

// class Bank
using System;

namespace AtmSimulator
{
        public class Bank{
```

```
        private static int[] pelates = {1234,4534,7808,4325,7654};
        private static int[] accounts = {1000,0,15789,9999,350};

        public static String Message(int password, int sum){
                int user;
                if ((user=ValidUser(password))==5) return "There is no such user";
                if (sum==0) return ("You have " + accounts[user] + " dollars left.");
                return SetBalance(sum, user);
        }



        private static int ValidUser(int password){
                int i;
                for (i=0;i<5;i++)
                        if (pelates[i]==password) break;
                return i;
        }

        private static String SetBalance(int accountBalance, int user)
        {
           if ((accounts[user]-accountBalance)>=0)
                {accounts[user]-=accountBalance; return ("Successful. You have "
                       + accounts[user] + " dollars left.");}
           else return ("Sorry, you only have " + accounts[user]
                         + " dollars left.");
        }
}
}
```

A.2.3 Αμοιβαίος Αποκλεισμός

```
// class MultiAccessMainThread
using System;
using System.Threading;

namespace MultiAccess
{
        class MultiAccessMainThread
        {
                public static Random RandomWaiting;
                public static void Main(string[] args)
                {
                        RandomWaiting = new Random();
                        SensitiveData the_data = new SensitiveData();
                Writer writer  = new Writer(the_data);
                Reader reader1 = new Reader(the_data, 1);
                Reader reader2 = new Reader(the_data, 2);
                Reader reader3 = new Reader(the_data, 3);

                Thread threadW = new Thread(new ThreadStart(writer.StartWriting));
                Thread threadR1 = new Thread(new ThreadStart(reader1.StartReading));
                Thread threadR2 = new Thread(new ThreadStart(reader2.StartReading));
                Thread threadR3 = new Thread(new ThreadStart(reader3.StartReading));

                threadW.Start();
                threadR1.Start();
                threadR2.Start();
```

```
                threadR3.Start();

                Thread.Sleep(10000);
                System.Environment.Exit(0);
                }
        }
}

// class Reader
#region
using System;
using System.Text;
using System.Threading;
#endregion

namespace MultiAccess
{
        public class Reader
        {
                private SensitiveData data;
                private int myId;
                private String whatIread;

                public Reader(SensitiveData data, int id){
                        this.data = data;
                        myId = id;
                }
                public void StartReading() {
                   try {
                      while (true){
                         Thread.Sleep(MultiAccessMainThread.RandomWaiting.Next(0,500));
                         data.Read(out whatIread);
                         Console.WriteLine("Id " + myId + ": " + whatIread);
                      }
                   }
                   catch (Exception e) {Console.WriteLine("Reader " + e.Message);}
                }
        }
}

// class Writer
using System;
using System.Threading;

namespace MultiAccess
{
        public class Writer
        {
                private SensitiveData data;

                public Writer(SensitiveData data){
                        this.data = data;
                }
                public void StartWriting() {
                   try {
                        while (true){
                           Thread.Sleep(MultiAccessMainThread.RandomWaiting.Next(0,500));
                           data.Write("I am writing");
```

```
                        }
                    }
                    catch (Exception e) {Console.WriteLine("Writer " + e.Message);}

                }
        }
}

// class SensitiveData
#region
using System;
using System.Text;
using System.Threading;
#endregion

namespace MultiAccess
{

        public class SensitiveData
        {
                private static StringBuilder data;
                private static int status;

                public SensitiveData(){
                        data = new StringBuilder("OK");
                        status = 0;
                }

                public void Read(out String data){

                if (Interlocked.CompareExchange(ref status,-2,0) == -1)
                        {
                                data = "could not read";
                                return;
                        }
                if (Interlocked.CompareExchange(ref status,1,-2) != -2)
                        Interlocked.Increment(ref status);
                Console.WriteLine("Read " + status);
                Thread.Sleep(125);
                data = SensitiveData.data.ToString();
                Interlocked.Decrement(ref status);
                }

                public void Write(String newData){

                if (Interlocked.CompareExchange(ref status,-1,0) != 0)
                                return;
                SensitiveData.data.Replace(SensitiveData.data.ToString(),newData);
                Thread.Sleep(250);
                SensitiveData.data.Replace(SensitiveData.data.ToString(),"OK");
                if (status!=-1) throw new Exception("Mutual Exclusion Violated!!");
                Interlocked.Increment(ref status);
                }
        }
}
```

A.3 Κώδικας Apollo

A.3.1 Needham-Schroeder public key protocol

```
def alice, bob.

  let AliceToBob1 =
        (fun AliceNonce:int.
           <AliceNonce, 'alice>{R bob bob}) 42,

      BobToAlice =
         (fun BobNonce:int, c1:cert, c2:cert, message: <int, 'alice>{R bob bob}.
            if c1 => 'bob #> '(delegate alice) then
               delegate 'bob <: 'alice in (
                  if c2 => 'bob #> '(declassify alice) then
                     declassify (bind msg2 = message in <msg2.1, BobNonce,
                                                         'bob>{R bob bob})
                        as <int, int, 'bob>{R alice alice}
                  else
                     error "fail"
               )
            else
               error "fail") 7,

      AliceToBob2 =
         (fun AliceNonce:int, c1:cert, c2:cert, message: <int,
                                  int, 'bob>{R alice alice}.
            if c1 => 'alice #> '(delegate bob) then
               delegate 'alice <: 'bob in  (
                  if c2 => 'alice #> '(declassify bob)then
                     declassify (bind msg2 = message in
                                  if msg2.1 = AliceNonce then
                                     <msg2.2>{R alice alice}
                                  else
                                     error "You are not Bob!!"
                                )
                       as <int>{R bob bob}
                  else
                     error "fail"
               )
            else
               error "fail") 42,

      BobReads2 =
         (fun BobNonce:int, message:<int>{R bob bob}.
            bind msg2 = message in
               if msg2.1 = BobNonce then
                  "Successful Communication"
               else
                  error "You are not Alice!!") 7

  in  BobReads2
          (AliceToBob2  (acquire 'alice #> '(delegate bob))
                        (acquire 'alice #> '(declassify bob))
                        (BobToAlice (acquire 'bob #> '(delegate alice))
                                    (acquire 'bob #> '(declassify alice))
                                     AliceToBob1))
```

A.3.2 Ηλεκτρονικές Τραπεζικές Συναλλαγές

```
def atm,alice,bank,prt.
def withdraw.

let
   login =
     (fun input:<>. <pack 'alice with alice as some user.'user,
                     acquire 'alice #> '(delegate atm)
                    >{R atm atm})
   ,
   request =
     (poly agent, user. fun msg:<'agent,'user,cert,cert>{R bank bank}.
        if (acquire 'bank #> '(delegate atm)) => 'bank #> '(delegate atm) then
            delegate 'bank<:'atm in (
               if (acquire 'bank #> '(declassify atm))=>
                      'bank #> '(declassify atm) then
                   declassify
                       bind x=msg in
                          let <a, u, c1, c2>=x in
                              if c1=> u #> '(delegate agent) then
                                 if c2=> u #> 'withdraw then
                                     1100{R bank bank}
                                 else error "fail"
                              else error "fail"
                   as int{R atm atm}
               else error "fail"
            )
        else error "fail")
   ,
   listen =
     (fun listen : <>. (pack pack
          <'atm, 'alice, acquire 'alice #> '(delegate atm),
           acquire 'alice #> 'withdraw,
           fun money:int{R atm atm}.<>
          >
         with alice as
            some user.<'atm,'user,cert,cert,(int{R atm atm} -> <>)>
       with atm as
          some agent,user.<'agent,'user,cert,cert,
                           (int{R agent agent} -> <>)>){R bank bank})
   ,
   print =
     (fun money : int{R atm prt}.<>)
   ,
   get =
     (poly user. fun human:'user.
        if human<:'alice then
           1200{R bank bank}
        else
           error "no such client")
   ,
   set =
     (poly user. fun human:'user, balance:int.<>)

in
< (
  fun u:<>.
  if (acquire 'atm #> '(declassify bank))
```

```
    => 'atm #> '(declassify bank) then
    bind x = login <> in
    let <y,cdel> = x in
    open userid = y with user in
    let creq = acquire userid #> 'withdraw in
    let msg = <'atm, userid, cdel, creq> in
    let dmsg = msg{R bank bank} in
    let balance = request [atm,user] dmsg in
    if (acquire 'atm #> '(declassify prt))
      => 'atm #> '(declassify prt) then
      let dbalance = declassify balance as int{R atm prt} in
      print dbalance
    else <>
  else error "insecure network"
  )<>
,
  (
  fun u:<>.
  bind x = listen <> in
  open y = x with agent, user in
  let <agentid,userid,cdel,creq,reply> = y in
  if cdel => userid #> '(delegate agent) then
    delegate userid <: agentid in
    if creq => userid #> 'withdraw then
      bind old = get [user] userid in
      let balance = old - 100 in
      let y = set [user] userid balance in
      let dbalance = declassify balance as int{R user user} in
      reply dbalance
    else <>
  else error "insecure network"
  )<>
>
```

A.3.3 Αμοιβαίος Αποκλεισμός

```
def reader, read1<:reader, read2<:reader, read3<:reader, writer.
def PermissionToRead, PermissionToWrite, IllegalAction.

// data.1 είναι η πληροφορία,
// data.2 ποιοι την επεξεργάζονται, 0 κανένας, 1-3 οι readers, 4 ο writer

let MayI=
fun data:<string,int>, action:int. // action=0 για δίαβασμα, action=1 για γράψιμο
if action=0 then
  if data.2 = 4 then
     acquire 'reader #> 'IllegalAction
  else
     acquire 'reader #> 'PermissionToRead
else if action=1 then
  if data.2 = 0 then
     acquire 'writer #> 'PermissionToWrite
  else
     acquire 'writer #> 'IllegalAction
else error "unknown action"
,

StartRead =
```

```
fun c:cert, data:<string,int>.
if c => 'reader #> 'PermissionToRead then
  <data.1, data.2 + 1>
else data
,

StopRead =
fun data:<string,int>.
if data.2=4 then
        data
else
        <data.1, data.2 - 1>
,

StartWrite =
fun c:cert, data:<string,int>, newValue:string.
if c => 'writer #> 'PermissionToWrite then
  <newValue, 4>
else data
,

StopWrite =
fun data:<string,int>.
if data.2=4 then
        <data.1, 0>
else
        data
,

Main =
poly user. fun data:<string,int>, StartOrStop:bool, AnyNewData:string.
if 'user<:'reader then
  if StartOrStop then
    StartRead (MayI data 0) data
  else
    StopRead data
else if 'user<:'writer then
  if StartOrStop then
    StartWrite (MayI data 1) data AnyNewData
  else
    StopWrite data
else error "unknown user"

in

        Main [read2] (
                Main [read1] (
                        Main [read2] (
                                Main [writer]
                                   (Main [read1] <"first value",0>
                                     true "does not matter")
                                true "ok")
                        true "does not matter")
                false "does not matter")
        false "does not matter"
```

Βιβλιογραφία

[1] Java 2SE Platform v1.4.2, API Specification, http://java.sun.com/j2se/1.4.2/docs/api/

[2] The Java Tutorial, http://java.sun.com/docs/books/tutorial/

[3] .NET Framework Class Library, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpref/html/cpref_start.asp

[4] .NET Framework Developer's Guide, Securing Applications, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconsecuringyourapplication.asp

[5] Secure Computing with Java: Now and the Future, http://java.sun.com/security/javaone97-whitepaper.html

[6] Security Policy Best Practices, http://www.gotdotnet.com/team/clr/SecurityPolicyBestPractices.htm

[7] A. W. Appel and A. P. Felty, A Semantic Model of Types and Machine Instructions for Proof-Carrying Code, Proceedings of the 27th Annual Symposium on Principles of Programming Languages (POPL 2000), ACM Press, 243–253.

[8] A. W. Appel, Foundational Proof-Carrying Code, Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science, June 2001, 247–258.

[9] D. Aspinall, Subtyping with Singleton Types, in Computer Science Logic, 1994.

[10] B. Bagnall, P. Chen, S. Goldberg, J. Faircloth, H. Cabrera, C# for Java Programmers, Syngress Publishing, Inc. 2002.

[11] Henry G. Baker, Jr. and Carl Hewitt, The Incremental Garbage Collection of Processes, August 1977, Proceedings of the 1977 symposium on Artificial intelligence and programming languages, Volume 12 , Issue 8 , 64.

[12] Henry G. Baker, Jr. List Processing in Real Time on a Serial Computer, April 1978, Communications of the ACM 21, 4, 280–294.

[13] H. P. Barendregt, Lambda Calculi with Types, in S. Abramsky, D.M. Gabbay and T.S.E. Maibaum, editors, Handbook of Logic in Computer Science, vol.2, Oxford Science Publications, 1992.

[14] L. Bauer, J. Ligatti and D. Walker, Types and Effects for Non-Interfering Program Monitors, Princeton University, 2002.

[15] Keith Brown, Security Briefs, Beware of Fully Trusted Code, MSDN Magazine, http://msdn.microsoft.com/msdnmag/issues/04/04/SecurityBriefs/

[16] Luca Cardelli, Type Systems, Handbook of Computer Science and Engineering, Chapter 103, CRC Press, 1997.

[17] Luca Cardelli and Peter Wegner, On Understanding Types, Data Abstraction and Polymorphism, ACM Computing Surveys 17(4), 471–522, 1985.

[18] K. Crary, S. Weirich and G. Morrisett, Intensional polymorphism in type erasure semantics, Journal of Functional Programming 12, Nov. 2002, 567–600.

[19] K. Crary and J. C. Vanderwaart, An Expressive, Scalable Type Theory for Certified Code, Proceedings of the 7th ACM SIGPLAN International Conference on Functional Programming, 191–205, 2002, Pittsburgh, PA.

[20] W. Diffie and M. E. Hellman, New directions in cryptography, IEEE Transactions on Information Theory, Vol. 22:pp 644–654, 1976.

[21] Li Gong, Java 2 Platform Security Architecture, http://java.sun.com/j2se/1.4.2/docs/guide/security/spec/security-spec.doc.html

[22] R. Harper and G. Morrisett, Compiling Polymorphism Using Intensional Type Analysis, Proceedings of the 22nd Annual Symposium on Principles of Programming Languages (POPL 1995), 130–141, ACM Press, 1995.

[23] W. A. Howard, The Formulae-as-Types Notion of Constructions, To H. B. Curry: Essays on Computation Logic, Lambda Calculus and Formalism, Academic Press, 1980.

[24] Gavin Lowe, Breaking and fixing the Needham-Schroeder public-key protocol using CSP and FDR, In T. Margaria and B. Steffen, editors, Tools and Algorithms for the Construction and Analysis of Systems, Second International Workshop, TACAS 1996, LNCS 1055, p. 147–166.

[25] G. Morrisett and D. Walker and K. Crary and N. Glew, From System F to Typed Assembly Language, Proceedings of the 25th Annual Symposium on Principles of Programming Languages (POPL 1998), 85–97, ACM Press, 1998.

[26] G. Necula and P. Lee, Safe Kernel Extensions without Run-Time Checking, Proceedings of the 2nd USENIX Symposium on Operating System Design and Implementation, 1996, USENIX Association, 229–243.

[27] G. Necula, Proof-Carrying Code, Proceedings of the 24th Annual Symposium on Principles of Programming Languages (POPL 1997), 106–119, ACM Press, New York.

[28] G. Necula, Compiling with Proofs, PhD thesis, Carnegie Mellon University, September 1998, CMU-CS-98-154.

[29] R. M. Needham and M. D. Schroeder, Using encryption for authentication in large networks of computers, Communications of the ACM, 21(2):993–999, December 1978.

[30] Vijay Patil, Role Based Security in .NET, http://www.csharphelp.com/archives/archive195.html

[31] A. Sabelfeld and A. C. Myers, Language-based information-flow security, IEEE Journal on Selected Areas in Communications 21, 5–19, 2003.

[32] Z. Shao and B. Saha and V. Trifonov and N. Papaspyrou, A Type System for Certified Binaries, Proceedings of the 29th Annual Symposium on Principles of Programming Languages (POPL 2002), 217–232, Portland, OR, USA.

[33] Stephen Tse and Steve Zdancewic, Run-time principals in information-flow type systems, ACM Transactions on Programming Languages and Systems, 1–39, 2004.

[34] David Walker, Karl Crary and Greg Morrisett, Typed Memory Management via Static Capabilities, Transactions on Programming Languages and Systems, 1–71, ACM Press, 2000.

[35] Ε. Ζάχος, Σημειώσεις στη Θεωρία Αριθμών και την Κρυπτογραφία, 2004.

[36] Νικόλαος Σ. Παπασπύρου, Programming with Proofs, draft version, last revised on May 31, 2005.

[37] Νικόλαος Σ. Παπασπύρου και Πάνος Ροντογιάννης, Σημειώσεις για το μάθημα Γλώσσες Προγραμματισμού ΙΙ, 2003-04.

[38] Ανδρέας-Γεώργιος Ν. Σταφυλοπάτης, Γλώσσες Προγραμματισμού, έκδοση Εθνικού Μετσοβίου Πολυτεχνείου, Αθήνα 1999.