Mellanskiktsprogram för Mobil Kod
Agentsystemet D'Agents

Mats Persson

18 mars 1999

1 Inledning

Det är ett känt faktum att mobil kod har många säkerhetsproblem. Dessa problem är ännu mer påfallande i mobila agentsystem där koden kan fritt vandra runt mellan datorerna, och kan exekvera var som helst och betrakta nätverket som en global interpretator. Mobil kod finns även i andra middleware-system som CORBA, DCOM, Java RMI och World Wide Web.

Ett syfte med den här undersökningen var därför att studera de system som använder sig av mobil kod, och då speciellt agentsystem, och hur de har löst säkerhetsproblemen.

För att kunna få en översikt på området så studerades flera olika middleware-system och huruvida de kan användas för mobil kod. För att kvalificera så måste de på något sätt kunna transportera en bit programkod för att exekveras på en annan dator. Här och där i rapporten har en del personliga bedömningar om dessa system lagts in.

Ett annat syfte var att göra en fungerande implementation av något slags mobil kod programvara, och tillverka ett antal agenter eller scripts som visade på olika säkerhetsproblem i det här systemet.

1.1 Definitioner

Inom detta område finns ett antal olika begrepp som bör definieras lite noggrannare. Många av begreppen används enligt följande definitioner i den här rapporten.

Mobil	förmåga att kunna förflytta sig eller bli flyttad, motsatsen till stationär
Autonom	förmåga att kunna handla självständigt
Kod	datorprogram
Objekt	innehåller data och kod i form av funktioner
Agent	autonomt objekt, något som agerar i någon annans ställe
Distribuerad	spridd över flera stycken (datorer)
Middleware	mellanskiktsprogram, kittet som knyter ihop kommunikationen i ett distribuerat system

Ofta används begreppen mobil kod, mobila objekt och mobila agenter. De är egentligen sammanblandingar av begrepp, men används oftast för samma sak. Det vill säga kod och data som mer eller mindre självständigt förflyttar sig i ett datornätverk.

2 Middleware

Det finns många olika distribuerade middleware system som på ett eller annat sätt anropar eller skickar kod över nätverk. Dessa system kan delas upp i distribuerade objektsystem, migrerande OS processer, agentsystem och world wide web. Det finns även andra metoder att skicka kod, såsom FTP och email, men dessa är inte alls lika automatiserade

Det hela handlar om objekt som skapas, anropas eller skickas genom ett nätverk till andra datorer. Objekten kan vara en bit kompilerad programkod, ett script eller bara ett virtellt objekt. De har ett väldefinierat gränssnitt och är ofta självständiga entiteter i den meningen att de inte är hårt bundna till en viss omgivning utan kan existera i enskilt. Enligt min åsikt har tyvärr många programmeringsspråk och dess programmerare en övertro på objektorientering och använder det för alla möjliga problem.

Många av de som har byggt dessa

middleware-system är i olika stor grad medvetna om säkerhetsproblemen med att fjärranropa objekt eller skicka mobil kod. I en del system har tyvärr säkerheten lagts till efteråt och i andra har det funnits säkerhet från början som sedan förbättrats.

Man kan klassificera dessa middleware eller transportsystem i typerna: distribuerade objektsystem, distribuerade operativsystem, agentsystem och World Wide Web.

2.1 Distribuerade objektsystem

Distribuerade objekt [1] är objekt som kan befinna sig på vilken annan dator som helst i ett nätverk av datorer, och i detta objekt kan en metod anropas från en annan dator i nätverket. För att utföra detta anrop behöver man ett speciellt system som håller reda på var objekten befinner sig och vilket gränsnitt de har. Det behövs även synkronisering, felhantering och säkerhetsfunktioner. Ofta finns det även ett "klister" som binder ihop objektsystemet med andra gamla icke objektorienterade system.

Distribuerade objektsystem skickar egentligen inte runt mobil kod, även om de kan fås att göra det med, utan de är mer som ett mellannivå- protokoll för att underlätta för program och applikationer att kommunicera. De skulle kunna utgöra en ryggrad åt agentsystem och ge dem en infrastruktur som till exempel registrering av namn och gränsnitt för agenterna.

2.1.1 CORBA

CORBA betyder Common Object Request Broker Architecture och är framtaget av Object Management Group. CORBA är ett system som låter olika applikationer kommunicera med varandra. De kan på ett transparent sätt anropa objekt på den egna datorn eller ett annat objekt någon annanstans i ett nätverk. Den klienten som anropar behöver inte veta var objektet befinner sig, i vilket programspråk det är skrivet eller i vilket operativsystem det befinner sig.

OpenDoc är en komponent arkitektur som baserar sig på CORBA. Det hanterar sammansatta dokument bestående av text, grafik och ljud, och har en inbyggd editor för att manipulera de olika komponenterna i dokumentet. Det kan även liknas vid ett filsystem. I OpenDoc är det meningen att man även skall kunna skriva scripts för att hantera komponenter och att dessa skall kunna vara mobila, det vill säga fria agenter. Numera är OpenDoc lite föråldrat och används knappast längre.

Som jag ser det är problemet med CORBA att det är en för stor, generell och lite klumpig lösning som tillåter säkrade och osäkrade delsystem att kommunicera med varandra. Det lär gå att låta CORBA skicka mobil kod, men det är inte direkt designat för det.

2.1.2 DCOM (ActiveX)

DCOM (Distributed Component Object Model) är Microsofts egen motsvarighet till CORBA. DCOM är hårdare integrerat i Windows arkitekturen och förutsätter att alla applikationer kör under Windows. Även ActiveX controls bygger på DCOM.

Jag ser det snarare som en nackdel att ett företag (Microsoft) äger en hel specifikation (DCOM). Det är svårt att få till en bra interoperabilitet.

2.1.3 Java RMI

Java's Remote Method Invocation är en annan variant på distribuerade objektanrop. Detta system tas inte upp närmre här utan istället hänvisas till David Lindahls rapport om Java. [2]

2.2 Distribuerade operativsystem

Dessa operativsystem kan bygga på en objektorienterad bas med migrerande processer. Ett exempel på dessa är Spring som kan låta processer vandra runt bland flera processorer. De har ungefär samma säkerhetsproblem som annan mobil kod. Distribuerade OS verkar dock ha vissa problem med robusthet för löst bundna processorer, och verkar inte vara lika framgångsrika som förväntat.

2.3 Agentsystem

Agenter är autonoma program som kan vandra runt i ett nätverk och samla information. Det finns ett antal olika utvecklade agentsystem som till exempel D'Agents, TACOMA, Telescript, IBM Aglets (Java). Agenter beskrivs mer utförligt i avsnitt 4.

2.4 WWW

På World Wide Web förekommer det ofta att websidor innehåller programkod och inte bara HTML-kod. Denna kod laddas ner automatiskt till användarens dator och exekveras i dennes webläsare. Denna nerladdning är till stor del styrd av användarens beteende och på vilka websidor denne besöker. Koden eller objekten är alltså inte fria att komma in i eller lämna webläsaren. Det går vanligtvis inte heller att skicka objekt till webservern, men däremot att starta CGI-script på den.

WWW är alltså ganska mycket en envägssituation för mobil kod. WWW och dess HTTP-protokoll är till största delen designat för att hämta och visa hypertextsidor.

2.4.1 Java

Java är ett programspråk som blivit väldigt populärt. På WWW förekommer det Java-applets men de har inte blivit lika populära där som förväntat, främst beroende på att det är för långsamt och för att JavaScript i många fall har varit lämpligare.

2.4.2 JavaScript

Däremot har Javascript blivit rätt vanligt på websidor, fast det är inte riktigt användbart till något annat. Överhuvudtaget är WWW ett ganska begränsat snitt av de möjligheter som mobil kod ger.

2.4.3 ActiveX

ActiveX är mobila objekt baserade på DCOM som laddas ner och installerar automatiskt sig själva när en websida besöks. Det är egentligen en `.dll` liknande fil som installeras. Tyvärr måste man köra Windows för att kunna använda ActiveX.

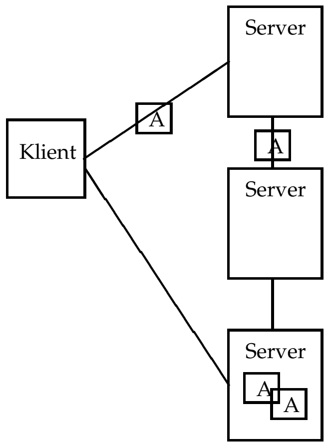
2.4.4 Övrigt

Det finns flera andra sammanhang där man skickar kod eller aktiverar fjärrstyrd kod. CGI (Common Gateway Interface) och ASP (Active Server Pages) är ett exempel på det senare. Det finns även möjligheter att skicka mobil kod via email och liknande meddelandehanteringssystem. De mest primitiva sätten att flytta och skicka kod är förstås FTP (File Transfer Protocol), eller använda det numera nästan utdöda Unix `uucp`, eller kopiera kod till diskett och flytta den på det sättet. Mobiliteten blir väldigt dålig.

3 Programspråk

Det programspråk som blivit vanligast inom området mobil kod är Java, och det har fått en marknadsledande ställning när det gäller säkerhet. Andra språk som är vanliga i distribuerade objektsystem är C och C++, och på Windows-sidan är det VisualBasic. För agenter har KQML och andra specialspråk för AI och informationsbehandling dykt upp.

Tcl har inte fått någon större genomslagskraft och Telescript har tyvärr lagts ner och ersatts av Java. Inte heller Perl och Python används i någon nämnvärd utsträckning även om man tycker att de skulle passa i dynamiska objektsystem.



Figur 1: Agenter vandrar runt i nätverket

Det finns flera sammanställningar och genomgångar över programspråk för mobil kod [3] [4].

4 Agenter

Det finns en definition av agenter:

> *Software agent: A computing entity that performs user delegated tasks autonomously.* [5]

En agent är alltså ett separat programobjekt som agerar autonomt. Enligt ovanstående definition så skulle det bara kunna agera när en användare ber om det. Det är en lite snäv definition som begränsar agenterna; vanligen kan de även agera helt självständigt utan påverkan från användaren.

4.1 Typer av agenter

Det finns både stationära och mobila agenter. De stationära stannar kvar på samma dator hela tiden och kan liknas vid daemon-processer i Unix. De mobila agenterna kan vandra runt i ett nätverk av datorer och samla information eller utföra distribuerade beräkningar. För att kunna vandra runt i nätverket så måste det finnas en agentserver på varje dator som tar emot och exekverar agenten.

Inom området agentteknologi finns det i princip två inriktningar. Dels de som sysslar med intelligenta agenter, utvecklar nya frågespråk och är inriktade mot agenternas artificiella intelligens. Dels finns det de som studerar agentsystemen och deras säkerhet och mobilitiet.

4.2 Val av system

Utav alla de middleware-system som nämnts tidigare har agentsystemen valts ut för närmare studier. Anledningen är främst att agentsystem är en bredare och mer flexibel exemplifiering av mobil kod, och att man kan skriva autonoma scripts i dessa system. Dessutom ställdes kravet på agentsystemet att agenterna skulle skrivas i ett interpreterande språk och att systemet skulle vara fokuserat på säkerhet och robusthet.

Det finns en stor mängd forskning om agenter och många av dessa projekt har utvecklat sina egna agentsystem. Exempel på dessa är Aglets, D'Agents, ARA, Concordia, Mole, Odyssey, TACOMA, Voyager och SHIP-MAI. En sammanfattning av dessa finns i en översiktsartikel [6]. Det är intressant att notera att de flesta systemen är utvecklade på någon Unix-variant.

Ett av agentsystem kallat D'Agents har valts ut för att kunna studera och testa en verklig implementation. D'Agents som tidigare kallades Agent-Tcl, använder SafeTcl som agentspråk men systemet självt är skrivet i C++. Anledningen till att detta system valdes ut är att källkoden är öppen, det kommer att kunna hantera Java, och är huvudsakligen fokuserat på säkerhet och robusthet.

4.3 Säkerhet

Säkerhetsproblemen för den typ av mobil kod som agenter utgör är många fler än i konventionella system som operativsystem och liknande. Inte minst för att mobila agenter är en ganska ny företeelse och det finns ganska lite forskning om det. Ett av de största problemen är att text och kod blandas och att hela nätverket kan ses som en gigantisk interpretator. Det kan uttryckas som i följande citat:

> *In fact, from a security perspective, there is nothing more dangerous than a global, heterogenous, general-purpose interpreter... mobile code is universally unthrustworthy...* [7]

Förutom de vanliga säkerhetsfrågorna, som sekretess, integritet och behörighetskontroll, och de övriga säkerhetsproblemen för mobil kod [8], finns det några speciella problem för agenter [9].

- Agenter är väldigt mobila och man har ingen kontroll över var de befinner sig. De kan bete sig väldigt slumpmässigt vilket kan leda till oönskade effekter som till exempel loopar där agenterna studsar fram och tillbaka mellan några agentservrar.
- De kan klona sig själv, löpa amok och fylla agentservrar med en stor mängd agenter. Detta äter snabbt upp systemets resurser.
- Agenten kan inte skyddas från påverkan från agentservern, vilket innebär att agenten själv kan råka ut för illasinnade agentservrar som ger agenten fel indata eller helt enkelt raderar den.
- Agenten kan kopieras av någon annan och användas för andra syften.

Det finns så många speciella säkerhetsproblem i agentsystem att de vanliga säkerhetsmodellerna inte längre räcker till, och knappast går att anpassa heller. Det måste till nya lösningar på problemen, och det är kanske dags att kasta bort principen om *"the information fortress model"* [10] som innebär att man bygger flera lager av skydd och murar, och kanske ersätta den med en mer flexibel eller reaktiv modell liknande biologiska immunsystem.

5 D'Agents

D'Agents [11] är ett programsystem för mobila agenter. Dessa agenter kan skrivas i Tcl, Java, eller Scheme. Systemet består av en agentserver, interpretatorer för programspråken och ett antal hjälpagenter. Agentsystemet är ett forskningsprojekt på Dartmouth College i New Hampshire, USA. D'Agents är utvecklat på Unix och kan köras på flera av dess dialekter, som till exempel Linux och Solaris.

För att kunna köra agentsystemet måste man ha en agentserver som lyssnar på en särskild port på varje dator. Dessutom måste ett antal hjälpagenter vara igång som sköter hanteringen av policies.

De programspråk som används i D'Agents har utökats med några extra funktioner för att kunna möjliggöra flyttning, kloning och meddelandesändning mellan agenter. De viktigaste är:

agent_submit	Startar en ny agent på samma eller annan dator.
agent_jump	Flyttar denna agent till en annan dator.
agent_fork	Klonar en kopia av denna agent.
agent_send	Skickar ett meddelande till en annan agent.
agent_receive	Tar emot ett meddelande.

Förutom funktioner för registrering finns det även en **agent_event**-funktion, men denna ger en ganska primitiv event-hantering där agenten själv får ta hand om det mesta.

```
proc hello {} {
  global agent
  main create -name Hello -display \
    $agent(actual-server):0
  button .button \
     -text "Hello, World!" \
     -command {set done 1}
  pack .button
  tkwait variable done
}
agent_begin
puts "Enter a nonnegative integer: "
gets stdin number
agent_submit $machine -vars number \
  -procs factorial \
  -script {factorial $number}
agent_receive code result -blocking
puts "$number! is equal to $result\n"
agent_end
```

Figur 2: Ett exempel på en agent.

5.1 Säkerhet i D'Agents

D'Agents har flera säkerhetsmekanismer, som till exempel autenticering med hjälp av PGP och har åtminstone en sandlåda baserad på Safe-Tcl för Tcl-interpretatorn. För den sandlådan kan man konfigurera hur många hopp en agent får göra, hur många kloner den kan skapa, hur mycket tid en agent får köra och mycket annat.

Varje agent som kommer till agentservern autenticeras med PGP, dess programspråk identifieras och sedan skickas koden till rätt interpretator. Vid behov kontrolleras rättigheterna genom att fråga en annan stationär agent kallad *resource_manager* som då letar upp namnet i en policyfil och som svarar "yes" eller "no" beroende på om operationen är tillåten eller inte. En operation kan då vara till exempel `agent_submit` för att starta en ny agent.

D'Agents tar hjälp av operativsystemet för hantera en del av säkerheten. Varje agent körs som en egen process och på det sättet kan agenten begränsas vad det gäller tid och minne.

I D'Agent finns även Tk-paketet med för att öppna grafiska fönster på skärmen, och detta är förstås även begränsat i policy-filen.

5.2 Implementationen

För att kunna testa D'Agents gjordes en installation av systemet på två datorer, och sedan testades en del demo-agenter som följde med. Några egna agenter utvecklades för att testa eventuella säkerhetsbrister.

För att vara ett utvecklingssystem och forskningsprojekt var det ett väldigt stabilt och säkert system, även om en del mindre brister upptäcktes. Det fanns även en del mindre problem vid installationen med buggar vid kompileringen, men det var kompatibilitetsproblem som löstes snabbt.

5.2.1 Säkerhetsbrister

Det fanns en del mindre allvarliga brister i D'Agents. Som ett rättesnöre har listan över säkerhetskrav för mobila scripts [8] använts. En del mindre demo-program har utvecklats för att visa vissa av bristerna.

- *Loggning av agenter saknas.* Det borde finnas en log över inkomna och skickade agenter, vilka som exekverats, och vilka nya som skapats.
- *Inget behörighetsystem för filer.* Det finns inget system för att hantera behörigheter, så om agenten får tillgång till filsystemet så får den i princip full behörighet till alla filer som agentsystemet har tillgång till, och kan således ändra i behörighetsfilen själv.
- *Minnesbegränsning saknas.* En agent har ingen begränsning i hur mycket minne den kan allokera och man kan ganska enkelt göra en "Denial of Service" attack på en agentserver genom att låta en agent konsumera stora mängder minne. Ett exempel på en sådan agent visas i figur 3. Problemet kan dock lösas i Unix med `ulimit`.
- *Tillåter `eval` i kod.* Om någon med höga rättigheter skriver en agent som innehåller något anrop till `eval` kan det bli ett potentiellt säkerhetshål. En enkel demo för detta konstruerades, där en `eval_server.tcl` tog emot en sträng och evaluerade den utan kontroller. Lyckligtvis innehåller inte den stationära säkerhetsagenten `resource_manager` någon `eval`.

5.2.2 Design

Förutom en del potentiella eller reella säkerhetsbrister så finns det ett antal designbrister i D'Agents som kan påverka säkerheten.

- Det saknas dokumentation över kommunikationsprotokollet till D'Agents, det vill säga det protokoll som används i agentöverföringen mellan två datorer. Det är svårt att analysera protokollet ur säkerhetssynvinkel om det saknas någon form av beskrivning av protokollet. Det saknas även dokumentation över källkoden till agentsystemet.

```
proc hog {} {
  set x 0;
  while {1} {
    incr x;
    set a($x) "test";
  };
}

agent_submit perseus -procs hog \
  -script hog
```

Figur 3: Minnesätare

- Agentservern forkar en gång för mycket när den exekverar en agent. När serverna startar så forkar den i två delar, en kallad `piped` som tar hand om pipes och en kallad `socketd` som tar emot inkommande agenter eller meddelanden. När en agent kommer in forkar `socketd` en gång och kallar sig `handle` vilken i sin tur forkar en process som startar interpretatorn. Detta verkar onödigt och genererar för många processer. Kanske de skulle fundera på att bygga en trådad server.

- D'Agents är skrivet i C++ och koden är ganska stor och svåröverskådlig, vilket gör den lite svår att analysera. Den borde kunna göras mycket mindre och enklare.

- När två datorer inte litar på varandra ömsesidigt så uppstår det problem i protokollet. Ett exempel på detta är följande händelser mellan datorerna `zorba` och `perseus`:

  1. `zorba` litar på `perseus`
  2. `perseus` litar inte på `zorba`
  3. `perseus` skickar agent till `zorba`
  4. `zorba` hänger sig när `perseus` vägrar ta emot resultatet.

- Om det går att byta ut `resource_manager` så faller hela säkerhetssystemet. Om den processen som kör `resource_manager` har försvunnit så kan någon annan starta en egen manager.

- D'Agents använder PGP för autenticering, och PGP har en mindre bra nyckeladministration eftersom den är avsedd för privatpersoner.

- Man får inga vettiga felmeddelande när rättigheter saknas för en agent. Det är överhuvudtaget svårt att skicka felmeddelanden när något går fel. De borde också autenticeras i sin tur.

- Det kunde vara vettigt att införa en separat säkerhetsnivå för lokala agenter.

5.3 Kritik av agentsystem

Det kanske verkar som om agenter har blivit det nya modet och då kommer förstås frågan, är agentsystem användbara till något? Ja, agenter används redan idag i en mängd sammanhang, men då oftast som ett litet program som ligger i bakgrunden och sköter en specifik uppgift.

Till exempel Windows 95 finns det en schemaläggaragent i bakgrunden som då och då kan defragmentera hårddisken eller sköta andra små uppgifter. På MacIntosh finns det speciella agenter i användargränsnittet som försöker lära sig användarens beteende och hjälpa denne med saker som ofta görs upprepat. För Unix kan man säga att många av dess daemons är egentligen agenter.

I övrigt finns det egentligen inte så många exempel på mobila agenter, förutom "maskar" och virusliknande scripts. I framtiden skulle agenter kunna användas för insamling och informationssökning, notifiering till användaren om vissa händelser, beräkningar av stora data distribuerat över ett stort nätverk, eller automatisering av många uppgifter.

Just nu är dock inte agentsystemen tillräckligt mogna för att användas i någon större seriös omfattning. Det finns många säkerhetsproblem som bör lösas först.

6 Problem med mobil kod

Det finns flera svåra och olösta problem inom mobil kod och det forskas en del inom detta området.

- *Proof carrying code* [12]. Det är en metod för att skicka med ett bevis på att koden är korrekt och inte bryter mot några säkerhetspolicies. Kontrollen sker statiskt och kan till exempel se efter så att inga buffertgränser överskrids. Tyvärr blir det problem med ett dynamiskt informationsflöde som kan vara svårt att förutsäga och ta med i bevisningen.

- *Komplexitetsproblem i sandlådan*. Många programsystem för mobil kod har en sandlåda där koden kan exekveras på ett säkert sätt. Tyvärr har denna sandlåda en tendens att bli ganska komplex, inte minst med ett avancerat policysystem och åtkomstkontroll.

- *Resurshantering och kontroll*. Hantering av begränsningar av CPU, minne, bildskärm och andra resurser är egentligen inte ett svårt problem, men resurskontroll saknas i många implementationer.

- *Kodskydd*. Ett av de svåraste, kanske till och med omöjliga, problemen är att skydda den mobila koden från värddatorn. Värddatorn kan i princip göra vad den vill med den mobila koden som till exempel klona den, ändra data inuti den, lägga sig i kodens kommunikation med andra datorer eller agenter. Om det var möjligt att exekvera krypterad kod så skulle det kanske lösa problemet.

7 Framtiden

Förutom problemen som nämns i sektion 6 ovan och som behöver lösas, så kan man tänka sig ett antal fortsatta forskningar om mobil kod. Dessa är här nedan ganska vagt formulerade.

- *Rekursiv evaluering*. Det uppträder ofta ett fenomen som kan kallas rekursiv evaluering. En interpretator interpreterar i sin tur en annnan interpretator. Många programmerare bygger mer eller mindre medvetet in interpretatorer i sina program för att göra dem mer flexibla och konfigurerbara. Data eller text kan börja tolkas som kod när man minst anar det. Kan man ha sandlådor inuti sandlådor? Hur kan man förebygga detta potentiella säkerhetsproblem?

- *Stack inspection*. Det finns nya idéer om att låta säkerhetspolicyn rekursivt följa med i anropsstacken. Denna anropstack kanske kunde följa med agenten i dess omgivning också? [13]

- *Flexibel accesskontroll*. Ett enkelt och flexibelt åtkomstsystem vore önskvärt, med hierarkier, anropskedjor och annat.

- *Topologier*. Hur skall en stor mängd agentservrar organiseras och hur skall nättopologin se ut? I pölar, kanaler och hav av agenter, eller i myrstackar?

- *Distribuerade system*. Hur kan agenter användas i CORBA och andra distribuerade objektsystem? CORBA är delvis förberett för mobil kod.

Referenser

[1] D. H. Robert Orfali and J. Edwards, *The Essential Distributed Objects Survival Guide*. Wiley, 1996.

[2] D. Lindahl, "Datasäkerhet och mobil kod: Säkerhetsmekanismerna i javas virtuella maskin." 1999.

[3] G. P. P. Gianpaolo Cugola, Carlo Ghezzi and G. Vigna, "Analyzing mobile code languages," in *Mobile Object Systems, Lecture Notes on Computer Science* (C. T. J. Vitek, ed.), Springer Verlag, 1997.

[4] T. Thorn, "Programming languages for mobile code," *ACM Computing Surveys*, vol. 29, September 1997.

[5] A. Caglayan and C. Harrison, *Agent Sourcebook*. Wiley, 1997.

[6] V. A. Pham and A. Karmouch, "Mobile software agents: An overview," *IEEE Communications Magazine*, pp. 26–37, July 1998.

[7] A. D. Rubin and D. E. Geer, "Mobile code security," *IEEE Internet Computing*, pp. 30–34, November-December 1998.

[8] M. Persson, "Mobile code and safe execution," User Report FOA-R–98-00807-503–SE, FOA, April 1998.

[9] J. D. G. William M. Farmer and V. Swarup, "Security for mobile agents: Issues and requirements," in *Proceedings of the 19th National Information Systems Security Conference*, pp. 591–597, October 1996.

[10] B. Blakley, "The emperor's old armor," in *Proceedings of New Security Paradigms Workshop*, 1996.

[11] R. S. Gray, "D'agents: Security in a multiple-language mobile-agent system." http://www.cs.dartmouth.edu/~agent, 1998.

[12] G. C. Necula and P. Lee, "Proof-carrying code," Technical Report CMU-CS-96-165, Carnegie Mellon University, November 1996.

[13] D. S. Wallach and E. W. Felten, "Understanding java stack inspection," in *Proceedings of Security and Privacy 1998*, May 1998.