Sikkerhed af .NET- og Java-programmer på bytekodeniveau

Arne Jørgensen, 300473-2919

30. maj 2005

Valgfagskurser:
Webtechnology
C# and Visusal Studio .NET

Vejleder:
Claus Cohn
Copenhagen Business College, Niels Brock

Indhold

1. Introduktion

Med internettets udbredelse er det i dag blevet mere og mere almindeligt at også software distribueres ad denne kanal. Det kan enten være køb/salg eller leje af software; det kan være i form af fri software/open source software; eller det kan være software der stilles tilrådighed af virksomheder eller myndigheder til servicerelaterede opgaver.

Samtidig er væksten i vira, trojanske heste, hackingforsøg og lignende stærkt stigende.

Kombinationen af dette stiller nye og skærpede krav til den sikkerhed der knytter sig til brugen af softwaren. Det bør få brugere af softwaren til at rejse spørgsmål om hvorvidt den distribuerede software nu også kommer fra den forventede og påståede leverandør og om softwaren opfører sig på den tilsigtede måde eller har skadelige bivirkninger.

Jeg finder emnet interessant netop på grund af dets øgede betydning for en sikker brug af pc'er og software i en stadig mere netværkorienteret it-virkelighed. Dele af emnet er samtidig beslægtet med emnet for min studiegruppes kommende hovedopgave om Digital Signatur.

2. Problemformulering

Jeg har valgt at tage udgangspunkt i .NET- og Java-platformene. Platformene er valgt da deres runtimesystemer netop er designede til afvikling af software fra kilder med et potentielt „ukendt" tillidsniveau.

Det vil sige at platformene er født med mekanismer der kan vejlede og sikre brugeren og de er derfor velvalgte til at belyse sikkerhedsaspektet.

I denne synopsis har jeg valgt at afdække to emner:

1. Hvordan kan man sikre sig at programmer på henholdsvis .NET- og Java-platformene kommer fra den ønskede leverandør?
2. Hvordan kan man sikre sig at programmer på henholdsvis .NET- og Java-platformene ikke opnår adgang til/får kontrol med dele af ens pc man ikke ønsker?

Der indgår naturligvis en lang række andre faktorer i spillet om at opnå et maksimalt sikkerhedsniveau for udførelsen af programmerne, men disse er ikke omfattet af synopsen af pladshensyn. Af samme årsag har det været nødvendigt at holde synopsen på et vist abstraktionsniveau hvorfor en del tekniske vil være udeladt.

Jeg vil i synopsen benytte betegnelsen *bytekode* både om det der på Java-platformen kaldes bytecode og det der på .NET-platformen kaldes assemblies (DLL- og EXE-filer).

3. Kildemateriale

I min litteratursøgning til emnet er jeg stødt på en række bøger. Fælles for dem alle har dog været at de (p.t.) er udgået fra forlagene og leveringstiden på bibliotekerne er lange. Analysen er derfor baseret på en række artikler der er tilgængelig på nettet.

Hovedkilderne har været onJava.com's artikelserie „Java vs. .NET Security" af Denis Pilipchuk [1-5], særligt artiklen „Code Protection and Code Access Security" [3].

.NET-platformen har jeg stiftet bekendtskab med i forbindelse med valgfagskurset *C# and Visual Studio .NET*, mens jeg ikke tidligere har beskæftiget mig med Java-platformen. I arbejdet med synopsen har jeg derfor også måttet studere noget baggrundsviden om de to platforme; uden at jeg dog anvender nogen litteratur eller kilder direkte her i.

4. Problemanalyse

Indledende beskriver jeg hvordan der kan opnås sikkerhed for et program kommer fra den ønskede leverandør og derefter gennemgår jeg hvilke mekanismer platformene benytter for at forhindre programmerne i at tilgå beskyttede ressourcer eller operationer.

4.1. Sikkerhed for leverandørens identitet

Som nævnt i introduktionen er det vigtigt at kunne sikre sig at programmer man ønsker at afvikle på ens pc kommer fra den leverandør man forventer og har tillid til.

På grund af internettets opbygning og der af følgende risici er det ikke nok at basere sikkerheden på at programmerne hentes fra tilsyneladende kendte URL'er. IP-adressen kan være spoofet eller hjemmesiden kan enddog være hacket hvorved man henter et andet program end det leverandøren har stillet tilrådighed.

Både .NET- og Java-platformen har løst problemet ved hjælp af PKI-baserede digitale signaturer med hvilke det er muligt at signere bytekode.

Den digitale signering sikrer både at programmet kan henføres til leverandøren via dennes offentlige certifikat og at der ikke er ændret på programmet siden det blev underskrevet af leverandøren med dennes private nøgle. Tillid til det udstedte certifikat kan højnes gennem de sædvanlige kanaler og metoder indenfor PKI-verdenen, som fx certifikatautoriteter (CA), tilbagetrækningslister (CRL), mm.

Figur 1: Javas virtuelle maskine anmoder brugeren om at bedømme tilliden til den digitale signatur i en applet (her fra http://www.optagelse.dk/)

I .NET-platformen er det muligt at signere hver enkelt assembly (bytekodefil) digitalt og dermed sikre dennes ophav og integritet. Signaturen er på .NET-platformen integreret i bytekodeformatet IL.

På Java-platformen derimod kan digitale signaturer ikke hæftes på de enkelte `class`-filer, men derimod kun på distributionsformatet `.jar`. Signaturen gemmes her i `.jar`-formatets manifestdel. Manifestdelen kan dog rumme flere signaturer der kan knytte sig til enkelte dele af `.jar`-filen.

Hvor det umiddelbart kan virke som to ret forskellige niveauer at tilføje signaturerne på finder jeg at denne forskel primært er knyttet til den tekniske implementation. Det vil således være muligt at opnå næsten ækvivalente løsninger på de to platforme. På Java-platformen kan man således opnå digital signering af en enkelt `.class`-fil ved at distribuere denne ene fil i en `.jar`-fil, mens man på .NET-platformen kan linke flere assemblies sammen i en enkelt assembly (svarende til `.jar`-filen) og derpå signere denne.

Der er dog punkter hvor platformene også adskiller sig. På Java-platformen er det muligt at knytte flere signaturer til den samme bytekode og man kan derfor have mere end én leverandør eller lignende til at stå inde for koden.

.NET-platformen har med begrebet *strong name* indført en særlig digital signatur der identificerer koden selv. Princippet og teknikken er helt tilsvarende digitale signaturer i øvrigt forstand og baserer sig på at producenten lader udstede et certifikat der skal identificere bytekoden selv. Motivationen bag indførelsen af *strong name*-teknikken lader ikke kun til at være at sikkerhedsmæssige grunde. En anden motivation er

jf. [6] at kunne identificere versioner af DLL'er mere entydigt og dermed undgå hvad artiklen omtaler som Windows' „DLL Hell".

For at de digitale signaturer skal have en effekt må kontrollen af dem ikke være overladt til applikationen selv (så ville vi have et „hønen og ægget"-problem). På begge platforme er kontrollen da også en integreret del af runtimesystemet. Runtimesystemet udfører således en kontrol at signaturens gyldighed *inden* programmet afvikles.

4.2. Adgang til og kontrol med pc'ens ressourcer

At opnå sikkerhed for leverandørens identitet behøver ikke være tilstrækkeligt til at have fuld tillid til programmet. Det vil stadig være ønskeligt at sikre sig at programmet ikke kan udføre handlinger eller tilgå informationer eller ressourcer som man ikke ønsker eller ikke er nødvendige.

Alene ud fra et forsigtighedsprincip er der ingen grund til at tillade applikationen flere rettigheder end godt er, men det er også tidligere set at selv store softwareleverandører har udsendt software med fx virus i.

Den grundlæggende tanke på både Java- og .NET-platformen er at runtimesystemet kontrollerer adgangen til beskyttede ressourcer og at en applikation kun kan opnå adgang til en ressource såfremt en række betingelser er tilstede.

Både Java-platformen og .NET-platformen opererer med begrebet *permissions* (tilladelser). En permission identificerer en beskyttet ressource som runtimesystemet enten kan tildele eller afvise en applikation adgang til.

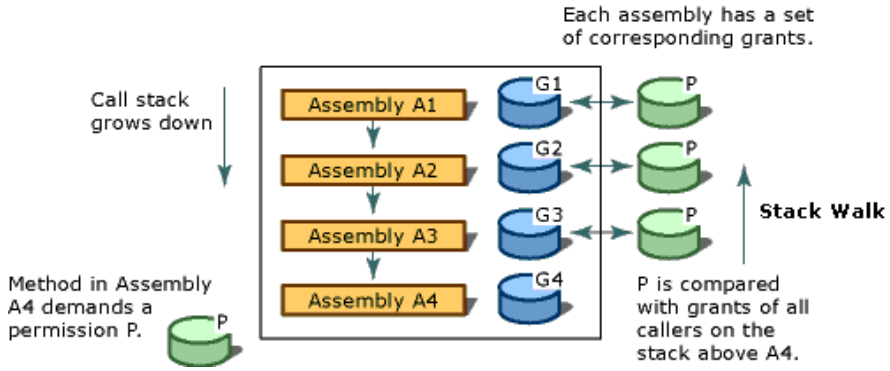
Ressourcerne der kan gives adgang er mangfoldige[1] og kan fx være læse-/skriveadgang til filer, adgang til printere eller DNS-server.

Endvidere benytter .NET-platformen sig af mængder af tilladelser, såkaldte *permission sets*, som en genvej til at give eller afvise en række permissions baseret på fx brugsscenarier. .NET opererer bl.a. med permission set'ene *Nothing*, *Execution*, *FullTrust*, *Internet*, *LocalIntranet* og *SkipVerification*.

Når runtimesystemerne skal afgøre hvilke permissions et program skal have adgang til indsamler det en lang række „beviser" (evidence) om programmet.

Beviserne er karakteristika ved applikationen som fx hvem der har signeret koden og hvor koden kommer fra (web eller placering i filsystemet). Java-platformens beviser er kun baseret på disse to, mens .NET yderligere inkluderer bl.a. *strong name*, og særlige zoner som brugeren kan gruppere koden i.

[1] Se fx .NET's på
http://msdn.microsoft.com/library/en-us/cpguide/html/cpconcodeaccesspermissions.asp?frame=true
og Java-platformens på
http://java.sun.com/j2se/1.4.2/docs/guide/security/permissions.html



Figur 2: Et *stack walk* (figur fra http://www.codeproject.com/dotnet/UB_CAS_NET.asp)

Til en installation af et runtimesystem hører en række politikker (policies) der er baseret på de indsamlede beviser om applikationen grupperer dette og tildeler herefter disse grupperinger de tilladelser det måtte skønnes nødvendigt.

Politikkerne kan udstikkes på flere niveauer. I .NET er der tre niveauer af politikker: *Enterprise*, *Machine* og *User*. I den samlede politik slår de mest restriktive permissions fra alle politikkerne igennem.

Når en applikation ønsker at tilgå en permission-beskyttet ressource foretager runtimesystemet en kontrol af de relevante politikker og beviser. I den forbindelse gennemgåes ikke blot den enkelte class-fil/assembly's tilladelser, men der foretages et såkaldt *stack walk* op gennem stakken af kaldene bytekode-enheder.

For hver bytekode-enhed indhentes beviser og udfra politikkerne tildeles permissions. Det er således den svageste permission i kæden der afgør om der gives adgang til ressourcen eller ej. Princippet i et *stack walk* fremgår af figur 2.

Generelt for begge platforme er det muligt at konfigurere mange aspekter sikkerheden gennem politikikker på flere niveauer, m.m.

4.3. Sikkerhed af runtimesystem

Som beskrevet i de foregående afsnit bygger sikkerheden og kontrollen af både leverandørens identitet og med applikationens adgang til beskyttede ressourcer på afgørelser foretaget af runtimesystemet.

Da ingen kæde som bekendt er stærkere end det svageste led forudsætter dette at vi har sikkerhed for at også runtimesystemet kommer fra den rette leverandør og at der ikke er manipuleret med dets komponenter.

Som med applikationerne er det ikke muligt at lade runtimesystemet selv afgøre hvorvidt det er troværdigt og/eller uændret.

Tilliden til runtimesystemets ophav må derfor sikres ad anden vej. Det kan fx være via fysisk distribution fra en troværdig leverandør eller elektronisk via digitale signaturer der valideres af anden software (fx operativsystemleverandøreres opdateringssystemer).

Ligeledes må sikkerheden for at runtimesystemet ikke er blevet ændret eller undermineret sikres ad anden vej. Da runtimesystemerne på både .NET- og Java-platformene afvikles som almindelige applikationer under operativsystemet baseres sikkerheden på operativsystemets mekanismer. Det kan fx være gennem adgangsrettigheder til systemets placering i filsystemet således at det kun er Administrator-rollen eller root-brugeren der har skriveadgang til runtimesystemets filer.

5. Foreløbig konklusion

.NET- og Java-platformene rummer mange ligheder på sikkerhedsområdet og platformene lader dog også til at være stærkt inspirerede af hinanden.

Til verifikation af applikationens leverandør og sikkerhed for at applikationen er uændret bygger begge platforme PKI-infrastrukturen. Valget virker overbevisende i det tilliden bygges op ud fra principper og begreber der i øvrigt kendes fra og bliver anvendt i forbindelse med digitale signaturer og digital forvaltning.

Når man ser bort fra *strong names* i .NET og muligheden for flere signaturer på et stykke Java-bytekode er mulighederne på dette punkt overvejende ækvivalente og forskellene teknisk. Et valg mellem platformene vil derfor kunne træffes ud fra andre parametre end disse.

I sidste ende vil sikkerhed baseret på digitale signaturer være et spørgsmål om tillid til den kæde af autoriteter der siger god koden.

Tildelingen af tilladelser, politikker, m.m. giver en meget høj grad af kontrol med hensyn til at styre adgangen til beskyttede ressourcer. .NET-platformen lader til at have et meget fleksibelt system til at dele disse. Desværre skaber det også en meget høj kompleksitet der kan være svær at overskue og derved kan risikere at hindre eller underminere hele sikkerhedskonceptet.

De forskellige opbygninger af Java-platformens og .NET-platformens systemer til adgangsstyring og kontrol med ressourcer er svære at veje op mod hinanden i håb om at finde en vinder. I følge [3] har .NET det mest fleksible system, men på minussiden lader han tælle at en del af de kontrollerede ressourcer ofte lægger sig meget tæt op ad det underliggende Windows-operativsystem. Jeg forestiller mig at dette kan blive

en hæmsko for .NET-platformens udbredelse til andre operativsystemer med mindre projekter som Mono[2] finder måde at emulere disse ressourcer på under fx Linux.

Sammenfattende kan jeg slå fast at begge platforme tilbyder avancerede teknikker til

- at sikre at bytekoden kommer fra den ønskede leverandør gennem digitale signaturer, og
- at sikre ens pc mod at bytekoden udfører operationer som man ikke på forhånd har ønsket.

Begge mål kan dog kun opnås såfremt der gøres en indsats for at *forstå* og *anvende* de tilrådighedværende sikkerhedsmekanismer. Det er nødvendigt at have en forståelse for PKI-opbygningen og dens tillidsmodel og det er nødvendigt at kunne formulere og konfigurere en række sikkerhedspolitikker der afspejler vores behov.

5.1. Spørgsmål

Til eksamen vil jeg lægge op til en samtale om

- de forskelle der er på sikkerhedsaspekterne i .NET- og Java-platformene og hvordan disse kan udnyttes med forskellige fordele, samt
- hvilke overvejelser og hvilken praktisk effekt sikkerhedsløsningerne har for hhv. programmører og brugere.

[2] http://www.mono-project.com/

Litteratur

[1] Denis Pilipchuk. Cryptography and communication. I *Java vs.* .NET *Security*, nr. 2. O'Reilly, december 2003. URL: http://www.onjava.com/pub/a/onjava/2003/12/10/javavsdotnet.html.

[2] Denis Pilipchuk. Security configuration and code containment. I *Java vs.* .NET *Security*, nr. 1. O'Reilly, november 2003. URL: http://www.onjava.com/pub/a/onjava/2003/11/26/javavsdotnet.html.

[3] Denis Pilipchuk. Code protection and code access security. I *Java vs.* .NET *Security*, nr. 3. O'Reilly, januar 2004. URL: http://www.onjava.com/pub/a/onjava/2004/01/28/javavsdotnet.html.

[4] Denis Pilipchuk. Epilogue: Upcoming security features. I *Java vs.* .NET *Security*, nr. 5. O'Reilly, februar 2004. URL: http://www.onjava.com/pub/a/onjava/2004/07/07/javavsdotnet.html.

[5] Denis Pilipchuk. User authentication and authorization. I *Java vs.* .NET *Security*, nr. 4. O'Reilly, februar 2004. URL: http://www.onjava.com/pub/a/onjava/2004/02/25/javavsdotnet.html.

[6] Francis Solomon. Introduction to code acces security in the microsoft .NET framework. Rapport, Pillar Technologu Group, LLC, 2004. Uddrag af artiklen tilgængelig online. URL: http://bdn.borland.com/borcon2004/article/paper/0,1963,32226,00.html.