Michael Zunke is DRM Chief Technology Office bij Aladdin Software

.NET – Ontwikkelen in een glazen huis?

BESCHERM JE CODE TEGEN INKIJK

Eindverantwoordelijken voor grootschalige softwareprojecten moeten rekening houden met alle mogelijke beveiligingsaspecten. Het .NET Framework biedt weliswaar een uitstekende beveiliging, toch zijn er twee mogelijke beveiligingskwesties bij .NET-ontwikkelingen; reverse engineering en ongewenste aanpassingen van code.

Alle .NET-applicaties zijn platformonafhankelijk. Gecompileerde programma's bevatten geen native processorinstructies. In plaats daarvan genereert de compiler zogeheten IL-code (Intermediate Language) die wordt geladen en gecompileerd door de runtime-omgeving van het systeem dat de code gaat gebruiken, ofwel het common language runtime (CLR)-systeem.

Hoe veilig is uw code?

Tot zover wijkt dit proces licht af van native code. Een nauwkeurige blik op de structuur van .NET-bestanden onthult veel meer. De .NET-omgeving is weliswaar voorzien van de nodige beveiliging, maar ontwikkelaars moeten zich bewust zijn van de .NET-architectuur om te voorkomen dat ze door middel van code hun eigen ruiten ingooien.IL-bestanden gebruiken de welbekende PE- (Portable Executable) bestandsstructuur die ook door Win32-bestanden wordt gebruikt. Er is voor het PE-formaat gekozen omdat het PE-mechanisme op een eenvoudige manier in oudere versies van Windows kon worden uitgebouwd. Als iemand de .NET-executable probeerde uit te voeren activeerde dit automatisch de CLR. Het is zelfs zo dat pure .NET-applicatiebestanden enige native Intel x86-code bevatten die ervoor zorgt dat de CLR wordt ingeladen en dat de controle wordt overdragen aan de core DLL (mscoree.dll); zie afbeelding 1.

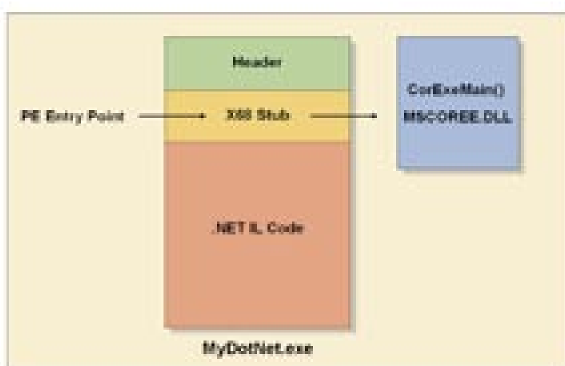
Het ware .NET-deel in het bestand wordt beschreven met behulp van een 'nieuwe' tag, bekend als MetaData. Of u het nu wilt of niet, de MetaData onthullen talloze details over uw code. Net als de PE-tables in native Win32-applicaties bevatten deze data informatie over geëxporteerde interfaces en geïmporteerde items. Verder zijn de namen van interne objecten, methoden, hun locaties, omvang, parameters, handtekeningen en alle strings hier te bekijken in een gerangschikte tabel. Deze MetaData zorgen ervoor dat de gebruiker geen informatie mist bij de disassembly van het systeem. Elke referentie naar andere objecten, zowel intern als extern, maakt gebruik van deze tabel. Voor Win32-applicaties speelt het voortdurende probleem dat de ontwikkelaar bij disassembly vaak moet raden naar het startadres van een specifieke functie en moet uitvinden of het bewuste gebied code dan wel data bevat. Bij de analyse van .NET-applicaties is echter geen sprake van al deze onzekerheid. De MetaData bieden u alle benodigde details. Het is alsof u Win32-applicaties levert met alle debugging-informatie.

Na het lezen van dit relaas kunt u zich goed voorstellen wat concurrenten en crackers met uw software kunnen doen:
- Geheime, bedrijfskritische algoritmes in de software zijn te achterhalen.
- Verificatie-informatie van licentievoorwaarden is duidelijk zichtbaar.
- Alle delen van de code zijn te decompileren en aan te passen, zodat ze zich anders gedragen en vervolgens opnieuw integreren.

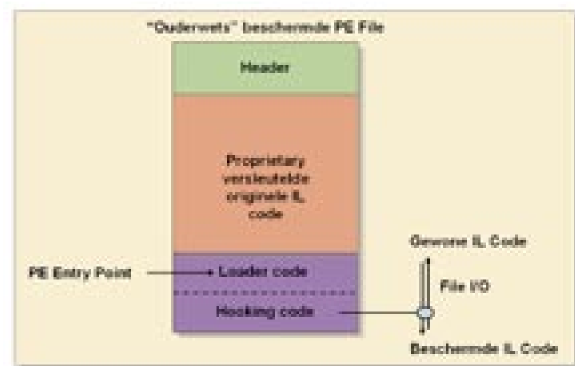
Hoe is dit te voorkomen?

Een oude en welbekende benadering van Java die nu ook in .NET wordt toegepast is obfuscation (verduistering).
Bij obfuscation gaan we uit van twee hoofdregels:
1. Computers geven niet om menselijke leesbaarheid. Nette, gemakkelijk te lezen en te doorgronden objectnamen zijn dus overbodig. Alle interne, niet-geëxporteerde objecten worden doorgaans voorzien van willekeurig gekozen herkenningspunten. In de meeste gevallen heeft de naam op zich geen enkele betekenis voor de uitvoering, zoals ook het geval is bij de meeste objecten.
2. Informatie die niet essentieel is voor correcte uitvoering wordt verwijderd.

Sommige obfuscators gebruiken aanvullende ingrediënten , zoals:
- Willekeurige toevoeging van code die geen invloed heeft op het resultaat van de functie.
- Invoering van kleine fouten in de samengestelde structuur waardoor standaardtools, zoals ILDASM, onjuiste resultaten produceren of de weg kwijt raken.



Afbeelding 1. Structuur van PE-bestand van een .NET-applicatie



Afbeelding 2. .NET-applicatie met 'ouderwetse' bescherming

Uw code uitkleden

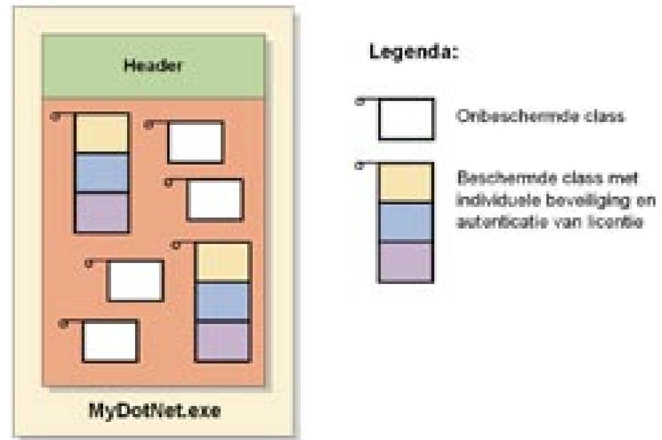
Al deze methoden verslechteren duidelijk de leesbaarheid van de code voor de gemiddelde mens. Desondanks is de uitgevoerde code uiteindelijk onbeschermd aanwezig in bestanden op de harde schijf. Alle informatie voor het disassembleren van de code met meer verfijnde tools is nog steeds aanwezig. Er zijn geen obstakels aanwezig om toegang tot de code te voorkomen. Crackers hebben al bij Win32-applicaties bewezen dat de leesbaarheid van variabelen geen noodzaak is om toegang te krijgen tot de applicatiecode en deze aan te passen. Het is dan ook duidelijk dat meer geraffineerde maatregelen nodig zijn om crack-pogingen succesvol te blokkeren.
De gangbare benadering is een erfenis van de vroegere Win32-beschermtools. Dit zijn de zogeheten 'plain loaders'. Het originele applicatiebestand wordt verpakt in een vergrendeld formaat met een loader die weet hoe de applicatie te reconstrueren is. Bij uitvoering krijgt de loader eerst de controle en wordt de originele applicatie in het geheugen geladen, of er wordt een filter geïnstalleerd dat actief wordt wanneer het besturingssysteem de applicatie aanspreekt, zodat het gedecodeerde fragment leesbaar wordt gepresenteerd. Wanneer de code aanwezig is in het geheugen wordt de loader compleet losgekoppeld van de applicatie; zie afbeelding 2.

Bij .NET-applicaties is dit schema alleen van toepassing op executables. Het is niet toepasbaar op assemblies. Tools die gebruik maken van deze techniek maken gebruik van het feit dat de .NET CLR aan te roepen is door de eerdergenoemde x86-code die aanwezig is in alle .NET-executables. Zo wordt de originele .NET-executable getransformeerd tot een regulier ogend Win32 PE-bestand. Bij uitvoering geeft het besturingssysteem de controle door aan het PE-entry point, de eigenlijke loader. Deze installeert filters om aan dit proces gerelateerde file I/O's te onderscheppen en geeft het stokje uiteindelijk door aan de mscoree.dll. Als de CLR het image-bestand van de schijf haalt, onderscheppen de filters van de loader de datastroom. De loader presenteert het originele, gedecodeerde image-bestand aan de CLR en de applicatie zal normaal opstarten. De kracht van deze benadering is dat de IL-code nooit onbeschermd wordt gelaten op de schijf, maar alleen aanwezig is in het tijdelijke geheugen tussen het laden en de eerste uitvoering. Bij de eerste uitvoering wordt de code 'just-in-time' (JIT) gecompileerd en aangepast. Als het formaat waarin de IL-code wordt opgeslagen op een schijf niet bekend is, zou de cracker het decoderingsproces volledig moeten 'reverse engineeren' om een deel van de applicatie op noemenswaardige wijze te kunnen aanpassen.

De grootste nadelen hiervan zijn dat deze methode alleen is toe te passen op executables en niet op assemblies, aangezien de x86-code zijn opstartfunctie niet kan vervullen. En als een cracker erin slaagt de aan de CLR gepresenteerde data te achterhalen, beschikt hij direct over het zuivere en onbeschermde origineel van het .NET-bestand. Bedrijven die beschermtools ontwikkelen volgens deze benadering, moeten dan ook krachtige anti-debugmaatregelen implementeren om een dergelijke aanval onmogelijk te maken.
De derde benadering van het probleem verschilt aanzienlijk van de eerdere. Deze methode is uitgevonden om de grootste zwakheden van obfuscation en de loader-benadering op te lossen. Bij obfuscation blijft de IL-code leesbaar voor de computer. Loaders werken alleen bij executables, zijn strikt gebonden aan de runtime-omgeving en zijn niet geïntegreerd in het uitvoerbare deel van de applicatie.

Class-niveau

Werken op class-niveau is dé oplossing. U kunt kiezen welke classes een individuele bescherming ontvangen die volledig is geïntegreerd in het gebruikelijke uitvoeringsproces (de execution flow, vergelijkbaar met de JIT-compilatie). Dit wordt een integraal deel van de instantiëring van de class. Elke class krijgt dus een individuele beveiliging; zie afbeelding 3. De op schijf opgeslagen data van deze class zijn daardoor onbruikbaar zonder hun eigen beveiligingsmechanisme. Dit beveiligingsmechanisme is onafhankelijk, waardoor het ook toe te passen is op .NET-assemblies. Het is zelfs toe te passen voor .NET-projecten op Mono1. In tegenstelling tot de loader-methode vergt deze technologie gedetailleerde kennis van de class-structuur en IL-code waarop de beveiliging van toepassing is. In vergelijking met obfuscation is het voornaamste verschil dat er een compleet nieuw .NET-bestand wordt gegenereerd dat in staat is tot hosting van zowel beschermde als onbeschermde classes. Door de mogelijkheid om de individuele class te selecteren, blijven de laadtijden laag en kan code automatisch op hetzelfde niveau worden toegevoegd. Bijvoorbeeld voor de bescherming van verschillende classes met afwijkende licentievoorwaarden, wat een ongekende flexibiliteit biedt voor licentiebeheer van software.



Afbeelding 3 .NET-applicatie of assembly met bescherming op class-niveau

Beveiliging van uw applicaties

Het ontwikkelen voor .NET zorgt voor nieuwe uitdagingen, met name bij het beschermen van uw applicatie tegen reverse engineering en illegaal gebruik door cracking. Moderne tools voor obfuscation kunnen reverse engineering stukken moeilijker maken. Om de beveiliging tegen crackers te verbeteren moet u zorgvuldig analyseren wat u nodig hebt. Als u geen compilatiebescherming nodig hebt, of licensering op class-niveau, kan een eenvoudige loader al voldoende zijn. Maar als u op zoek bent naar volledige flexibiliteit is de class level-methode de beste keuze.

Michael Zunke, Aladdin Software DRM CTO. http://www.aladdin.nl

Nuttige internetadressen
Informatie over Aladdin's HASP Enveloppe:
http://www.aladdin.com/HASP/HaspHL_v120.asp
http://www.aladdin.nl/solutions/Digital_Rights_Management.asp
Informatie over obfuscation:
http://www.microsoft.com/netherlands/msdn/headline/jul02_ilobfuscation.aspx
http://www.gotdotnet.com/content/featuredsite/wiseowl/default.aspx

Noot
1. Project Mono is een open source-initiatief dat met behulp van de ECMA-standaard de .NET-omgeving ontwikkelt voor onder andere Linux, Macintosh, FreeBSD. http://www.mono-project.com/