Nigel Watling is werkzaam bij Microsoft EMEA

Beveiligingscontrole - Check

VERBODEN VOOR ONBEVOEGDEN

Tegenwoordig zijn auto's voorzien van deursloten, een contactslot, startblokkering, een alarminstallatie en een tracking device, zodat de politie bij diefstal de dief snel kan vangen. Eigenlijk zou u uw toepassingen op dezelfde manier moeten beveiligen. In dit artikel geeft de auteur een overzicht van enkele beveiligingsproblemen waarmee ontwikkelaars rekening moeten houden als ze veilige toepassingen op het Windows-platform willen bouwen.

In het voorwoord van Secrets and Lies noemt Bruce Schneier een reden voor het schrijven van het boek: hij heeft het gedaan om de fouten in zijn vorige boek, Applied Cryptography, te corrigeren. Dit is toch wel schokkend, omdat Applied Cryptography een van de invloedrijkste werken op het gebied van beveiliging was en is. Wat hij echter bedoelt, is dat hoewel de wiskunde die het hart van cryptografie vormt perfect is, dat op zich geen garantie voor veiligheid is. Waarom? Omdat systemen rondom hardware, software, netwerken en mensen worden gebouwd. Door gebreken in deze factoren doet de wiskundige perfectie van cryptografie niet meer ter zake. De huidige systemen zijn zo ingewikkeld dat volledige beveiliging onmogelijk is. "Als u denkt dat u uw beveiligingsproblemen kunt oplossen met technologie, begrijpt u het probleem niet en begrijpt u de technologie niet", aldus Schneier. Als ontwikkelaars kunnen we niet gewoon op technologie vertrouwen om de toepassingen die we schrijven veilig te maken. We moeten ons bewust zijn van alle beveiligingsproblemen die van invloed zijn op onze software en de platforms waarop deze wordt gebruikt. Dan doen we wat we kunnen doen om die problemen aan te pakken. We moeten met name onze code op gebreken controleren: zowel op ontwerpgebreken als codegebreken.

Het goede nieuws is dat hoewel een veiligheidscontrole van onze code net zo aantrekkelijk is als een bezoek aan de tandarts (we weten dat we het moeten doen om grote problemen op een later tijdstip te voorkomen, maar waarschijnlijk zal het onaangenaam en misschien wel duur zijn), het toch heel interessant blijkt te zijn voor de meeste ontwikkelaars. Als je met de ogen van een hacker naar jouw toepassingen kijkt, begrijp je veel beter hoe de toepassingen eigenlijk werken. Hackers zijn vaak beter op de hoogte van systemen dan de mensen die de systemen hebben ontworpen! Ook al vindt een ontwikkelaar de ervaring niet aangenaam, het is gewoon iets dat moet gebeuren. Hopelijk biedt dit artikel een paar nuttige wenken. Hier verder volgen enkele beveiligingspunten en –problemen waarvan een Windows-ontwikkelaar volgens mij op de hoogte moet zijn. Omdat hier onvoldoende ruimte is om er gedetailleerd op in te gaan, bevat dit artikel verwijzingen om meer over deze problemen te lezen en ze grondig te bestuderen.

Veilige communicatie

Veilige communicatie betekent de bescherming van de geheimhouding en integriteit van gegevens. Voor veilige communicatie heb je de keuze uit onder meer IPSec, SSL/TLS en RPC Encryption (gebruikt door DCOM). IPSec is een versleuteling op pakketniveau en is ondertekend. Het handige hiervan is dat het programma in het platform is ingebouwd. De ontwikkelaar hoeft niets te doen omdat de systeembeheerder het proces instelt. Achter de schermen gebruiken twee computers de gebruikelijke combinatie van asymmetrische en symmetrische versleuteling met een session key. SSL (Secure Socket Layer) werkt ongeveer op dezelfde manier als IPSec en wordt meestal gebruikt tussen browserclients en websites en ook ter beveiliging van webservice-calls. SSL verifieert altijd de server en verifieert optioneel de client met digitale certificaten. Browsers en IIS hebben ingebouwde ondersteuning voor SSL, zodat ontwikkelaars zich ook nu weer niet hoeven bezig te houden met SSL. Dit is iets dat de systeembeheerder doet door een certificaat voor een website aan te schaffen of door met Certificate Services clientcertificaten op een intranet te distribueren. TLS (Transport Level Security) is in wezen SSL 3.1, maar dan in een IETF-standaard (Internet Engineering Task Force).

Identiteitbeheer

Verificatie en autorisatie worden vaak met elkaar verward. Verificatie is het controleren of mensen zijn wie ze beweren te zijn door validatie van een reeks referenties. Deze referenties kunnen iets zijn dat iemand heeft (bijvoorbeeld een smartcard), iets dat iemand weet (zoals een gebruikersnaam en een wachtwoord) of een bepaalde verificatiemethode, zoals basisverificatie, Microsoft Passport en Kerberos. Nadat de identiteit is vastgesteld door middel van verificatie, wordt autorisatie gebruikt om toegang tot bepaalde gegevens of de uitvoering van bepaalde methodes toe te staan. Aan de hand van een ACL (Access Control List) kan bijvoorbeeld worden bepaald of iemand naar een bestand mag schrijven. Bij het schrijven van code kunnen toegangscontroles doorgaans op twee verschillende manieren worden toegepast: door verplichte toegangscontroles en/of declaratieve toegangscontroles. Een verplichte controle is in codevoorbeeld 1 geschreven.

```
if (PO.Total > 10000 && !IsInRole("Manager"))
throw new SecurityException(" U moet een manager zijn");

// Een declaratieve toegangscontrole wordt uitgevoerd door middel van
een kenmerk:

[SecurityRole("Klant")]
public void Order(…)
{…}
```

Codevoorbeeld 1.
Verplichte controle

```
// cchAttribute is het aantal tekens dat door gebruiker wordt ingevoerd
WCHAR wcsAttribute[200];
if ( cchAttribute >= sizeof wcsAttribute )
    THROW( CException( DB_E_ERRORSINCOMMAND ) );
DecodeURLEscapes( (BYTE *) pszAttribute, cchAttribute,
wcsAttribute,webServer.CodePage());
...
void DecodeURLEscapes( BYTE * pIn, ULONG & l, WCHAR *
pOut, ULONG ulCodePage )
{
WCHAR * p2 = pOut;
ULONG l2 = l;
...
for( ; l2; l2-- )
{
// schrijf naar p2 op basis van pIn, maximaal 12 bytes
```

Codevoorbeeld 2.
Buffer-overrun: Code Red

Hacken van cryptografie

Cryptografische algoritmen kunnen krachtig zijn, maar ze zijn slechts zo goed als het proces waarin ze worden gebruikt. Het verleden is vervuld van beroerde codes, variërend van Mary Queen of Scots tot de Enigma-machine. Het is van essentieel belang om te beseffen hoe gemakkelijk elke methode kan worden gekraakt zie tabel 1. De volgende informatie is beschikbaar op www.cryptoapps.com/~peter/part1.pdf. Het advies dat hier wordt gegeven is dat er een getest algoritme moet worden gebruikt in plaats van een vaag algoritme of, de hemel verhoede het, een eigen algoritme! Op Windows betekent dit het gebruik van CryptoAPI, CAPICOM of System.Security.Cryptography. Keys (sleutels) worden vaak afgeleid van wachtwoorden, zodat er krachtige wachtwoorden en een initialisatievector (IV) moeten worden gebruikt als de key robuust moet zijn. IV's zijn niet geheim, ze kunnen bijvoorbeeld in een bericht worden doorgegeven, ze zorgen gewoon voor een toevallige verdeling bij het maken van gecodeerde tekst. Zelfs wanneer twee verschillende berichten een identieke platte tekst hebben, is de resulterende gecodeerde tekst hierdoor verschillend. Daardoor kan een hacker de inhoud niet meer afleiden uit terugkerende patronen in gecodeerde tekst. Vergeet tot besluit niet dat een hacker een algoritme helemaal niet hoeft te kraken als wachtwoorden onveilig zijn opgeslagen. Sla geheimen indien mogelijk op verschillende ontoegankelijke systemen op, of gebruik zoiets als DPAPI (Data Protection Application Programming Interface). Beveiliging van keys is van vitaal belang!

Buffer-overruns en invoervalidatie

De buffer-overrun is het meest voorkomende en grootste veiligheidsrisico. Buffer-overruns zijn voornamelijk een probleem in C- en C++-code. Ze treden op als gegevens omvangrijker zijn dan verwacht en daardoor andere waarden worden overschreven. Het codevoorbeeld 2 bevat een buffer-overrun. Hoe kan deze worden waargenomen en opgelost?

Het probleem wordt gevormd door wcsAttribute. Het is een WCHAR, zodat het 400 bytes kan bevatten in plaats van 200 bytes. U corrigeert deze fout in de code door de omvangcontrole te wijzigen; zie codevoorbeeld 3.

```
if ( cchAttribute >= sizeof wcsAttribute / sizeof WCHAR)
```

Codevoorbeeld 3.
Correctie

```
printf(userSuppliedString); // BUG!
printf("%s", userSuppliedString); // correct
```

Codevoorbeeld 4.
Beschadiging van de stack

Deze buffer-overrun is bekend geworden onder de naam Code Red. Door gebruik te maken van een buffer-overrun kan een hacker een van de volgende drie dingen doen: een access violation veroorzaken, instabiliteit in een toepassing veroorzaken of, erger nog, willekeurige code laten uitvoeren met alle privileges van het toepassingsproces. Bij Code Red was het proces SYSTEM. Daarom is het van cruciaal belang om toepassingen uit te voeren terwijl het geringste privilege is ingesteld en daarom wordt IIS 6.0 uitgevoerd met een speciale NetworkService-account met weinig privileges. Alleen gekken voeren hun toepassingen uit als SYSTEM of Administrator! Het voorgaande geciteerde voorbeeld is een stack overrun, maar er zijn ook andere typen buffer-overruns mogelijk, zoals heap overruns, v-table overruns, function pointer overwrites en exception handler overwrites. De stack kan al worden beschadigd of overschreven met een simple string, zie het codevoorbeeld 4 in C. Met de eerste tekenreeks kan een aanvaller willekeurige specifiers opgeven, inclusief de vage maar geldige specifier %n die naar de stack schrijft! Echt verontrustend voor de auteur van dit artikel is de inherente onveiligheid van een groot aantal routines uit de C-bibliotheek. Het is ongelofelijk gemakkelijk om gebreken in de code aan te brengen door naïef gebruik van strcpy, strcat, memcpy en sprintf. Veel ontwikkelaars gebruiken dan ook veilige versies van deze functies (bijvoorbeeld strsafe.h). Als de code in C of C++ wordt geschreven, is het aan te raden om bijlage A van Writing Secure Code te bekijken, waarin dit probleem uitgebreid aan bod komt. Het zou natuurlijk heel fijn zijn om hulpprogramma's of hardwareondersteuning te hebben voor het opsporen van buffer-overruns. Visual C++ .NET heeft een compiler-schakeloptie /GS die een cookie ter grootte van een pointer invoegt tussen een buffer en de functieafzender op de stack. Op deze cookie worden controles uitgevoerd en er wordt een foutmelding gegenereerd als de cookie is gewijzigd. Het gebruik van deze functie is beslist geen wondermiddel en is geen vervanger voor zorgvuldig programmeren. En andere typen overrun kunnen nog altijd voorkomen.

Cross-site scripting (XSS)

Met cross-site scripting kan een hacker kwaadwillig script uit-

Aanvaller	Budget	Hardware	Tijd & kosten		Veilige key-lengte	
			40-bits	56-bits	1995	2015
Alledaags	Heel klein	PC	1 week	ondoenlijk	45	59
Hacker	$400	FPGA	5 uur $0.08	38 jaar $5,000	50	64
Klein bedrijf	$10.000	FPGA	12 min $0.08	556 dagen $5,000	55	69
Bedrijfsafdeling	$300.000	FPGA	24 sec $0.08	19 dagen $5,000	60	74
		ASIC	0,18 sec $0.001	3 uur $38		
Groot bedrijf	$10.000.000	FPGA	0,7 sec $0.08	13 uur $5,000	70	84
		ASIC	0,005 sec $0,001	6 min $38		
Inlichtingendienst	$300.000.000	ASIC	0,0002 sec $0.001	12 sec $38	75	89

Tabel 1.
Versleuteling doorbreken met 'Brute Force'-technologie

```
inputUserID = Request.Form("gebruikers-id")
inputPassword = Request.Form("wachtwoord")

query = "SELECT * FROM users WHERE " & "userID = '" & _
inputUserID & "'" & "AND password = '" & inputPassword & "'"

rs.open(query, conn)
If rs.EOF Then
    Response.Write("Probeer het nogmaals" )Else
    Response.Write("Gelukt")
End If
```

Codevoorbeeld 5.
Wijziging oorspronkelijke SQL injection

```
SELECT count(*) FROM client WHERE userID=? AND password=?
```

Codevoorbeeld 6.

voeren in een browser van een gebruiker. Als alles wat door een gebruiker wordt getypt, wordt weerkaatst via een webpagina, is de site kwetsbaar, ongeacht de gebruikte technologie. De hacker kan script in de pagina insluiten of HTML ‹form›-tags in geldige hyperlinks insluiten. Het actiekenmerk van de ‹form›-tag is ingesteld naar de site van een hacker en privé-gegevens kunnen naar die site worden verzonden (zoals cookie-gegevens, gebruikersgegevens of voorkeuren). U kunt XSS voorkomen door echoën van invoer van de gebruiker en onveilige opslag van geheime informatie te vermijden. Invoer van de gebruiker mag nooit worden vertrouwd en moet altijd zorgvuldig worden gefilterd. Een beschikbare techniek die u in .NET kunt gebruiken, zijn de serverobjectmethoden HtmlEncode en URLEncode om HTML-elementen te converteren naar tekenreeksweergaven voordat deze worden weergegeven.

SQL injection en beveiliging van toegang tot gegevens

SQL injection vindt plaats als een ontwikkelaar gebruikers toestaat om de criteria van SQL-instructies te bepalen en als hackers waarden invoeren die de oorspronkelijke bedoeling van de SQL-instructie wijzigen. Vertrouw gebruikers nooit! Kijk maar wat er kan gebeuren in codevoorbeeld 5.

Wat gebeurt er als de gebruiker het volgende in het veld gebruikers-id typt?
1) administrator --
2) administrator‘ or 1=1 --
3) administrator ‘ or 1=1 drop table orders --
4) administrator ‘ or 1=1 exec master..xp_cmdshell --

Antwoorden:
1) Het wachtwoord wordt overgeslagen (-- is een SQL-opmerking).
2) Alle records worden geretourneerd (1=1 is altijd waar).
3) De tabel met orders gaat verloren.
4) Er verschijnt een opdrachtvenster. Als u het programma als SYSTEM uitvoert, bezit de hacker nu de server.

Er zijn verschillende manieren om dit soort misbruik te verzachten en deze worden in het navolgende beschreven. Maar het belangrijkste is om te voorkomen dat je informatie over de database openbaart. Standaard retourneren ASP-pagina’s OLE-DB- en ODBC-foutinformatie naar de gebruiker. Door invoerparameters te wijzigen, kan een hacker deze foutinformatie gebruiken om de database te onderzoeken en tabelnamen, kolomnamen, gegevenstypen en rijwaarden ontdekken. Er volgen nu twee manieren om dit misbruik te vermijden: invoer van aanhalingstekens en stored procedures. Bij de eerste methode wordt elk enkel aanhalingsteken in de invoer van de gebruiker vervangen door twee enkele aanhalingstekens voordat de SQL-instructie wordt uitgevoerd. Het achterliggende idee is dat er een ongeldige SQL-instructie verschijnt wanneer een hacker een aanhalingsteken probeert in

```
Bestanden
1) MijnLangeBestand.txt
2) MijnLangeBestand.txt.
3) MijnLange~1.txt
4) MijnLangeBestand.txt::$DATA

URL‘s
1) www.microsoft.com/technet/beveiliging
2) www%2emicrosoft%2ecom%2ftechnet%2fbeveiliging
3) www.microsoft.com%c0%aftechnet%c0%afbeveiliging
4) www%25%32%65microsoft.com/technet/beveiliging
5) http://172.43.122.12 = http://2888530444

Geheim bestand
1) C:\Windows\Foo\Geheim\cmd.exe
2) C:\Windows\Foo\Geheim\Bar\Temp\..\..\cmd.exe
3) C:\Windows\Foo\Geheim\Bar\..\cmd.exe
4) C:\Windows\Foo\..\Foo\Geheim\Bar\..\cmd.exe
```

Tabel 2.
Bestandsnamen

te voeren. Maar deze techniek is om twee redenen gebrekkig. Op de eerste plaats: als er invoervelden zijn waarin andere typen dan tekenreeksen zijn toegestaan, hoeft de hacker geen aanhalingsteken te gebruiken. Op de tweede plaats kan de hacker in plaats van een aanhalingsteken de functie char(0x27) gebruiken! Stored procedures kunnen helpen bij verbindingen (dat wil zeggen, als een hacker probeert “or” te gebruiken), maar een hacker kan nog steeds een SQL-opdracht zoals update of insert invoeren. Stored procedures vormen geen afdoende oplossing. Het is effectief om met het geringste privilege te werken. (Gebruik nooit system admin.) Als je de keuze hebt, moet je geïntegreerde beveiliging gebruiken in plaats van SQL-beveiliging, met een account die slechts voldoende privileges heeft om te doen wat nodig is, en niet meer. Is er bijvoorbeeld een account beschikbaar die alleen mag lezen? Het aanvalsgebied kan worden verkleind door ongebruikte stored procedures te verwijderen en geparametriseerde opdrachten te gebruiken; zie codevoorbeeld 6.
Dit heeft het geweldige neveneffect dat het sneller is dan aaneenschakeling van SQL in code! Writing Secure Code bevat een heel hoofdstuk over problemen met betrekking tot database-invoer.

Werken volgens bepaalde regels

Er zijn veel manieren om een bestandsnaam of een URL weer te geven. Zie de voorbeelden in tabel 2.
Het komt er op neer dat een beveiligingsbeslissing nooit gebaseerd mag zijn op de naam van iets. Dit gaat in tegen de manier waarop verzoeken worden gerouteerd door IIS. Het misbruik van $DATA (hiervoor) werkte op deze manier. Normale uitdrukkingen kunnen beperken wat een gebruiker in een naam zet en ze staan gebruikers niet toe om paden op te geven; aangezien een omgevingsvariabele PATH ook niet betrouwbaar is. Problemen met het doorkruisen van mappen kunnen worden vermeden door harde koppelingen naar toepassingsmappen te maken met het hulpprogramma Linkd.exe uit de Windows 2000 Resource Kit of met de API CreateHardLink. Als je een ISAPI-toepassing of –filter schrijft, is het belangrijk dat je

1. Beveiliging als een ontwerpfunctie: denk over beveiliging en implementeer deze vanaf het begin.
2. Voer het programma uit met het geringste privilege.
3. Verklein het aanvalsoppervlak van de toepassing.
4. Vertrouw invoer van gebruikers niet.
5. Laat de aanvaller raden: geef geen gratis informatie prijs.
6. Gebruik verdediging grondig.
7. Test, test, test. “Denk als een hacker. Wees kwaadaardig. Test kwaaddenkend.” Of betaal een expert om kwaadwillig te zijn.
8. Verplaats gevaarlijke C/C++-code naar managed code (zoals C#).

Beste tips voor het schrijven van veilige code

uiterst zorgvuldig te werk gaat. Het gebruik van een low level programmeertaal als C of C++ en het bewerken van bestanden kunnen tot extra problemen leiden.

Samenvatting

Toepassingen moeten standaard, qua ontwerp en qua implementatie veilig zijn. Ik hoop dat ik je wat inzicht heb gegeven in hoe jouw toepassingen kwetsbaar kunnen zijn en wat je kunt doen om een beschadigende aanval te voorkomen. Het bouwen van een veilig systeem is echter niet hetzelfde als het systeem geheel onkwetsbaar maken. Dat is gewoon niet mogelijk. Houd ook mensen en processen in gedachten. Beveiliging betreft zowel detectie en reactie als bescherming. Als je hackers wilt weren, pas je veel verschillende systemen toe die ze moeten overwinnen.

Nuttige internetadressen

Development center: msdn.microsoft.com/security/
www.microsoft.com/security/guidance/default.mspx
RSS feed: msdn.microsoft.com/security/rss.xml
www.microsoft.com/mspress/benelux/books/book19911.htm

Nigel Watling **is Senior Developer Architect bij de Developer and Platform Evangelism Group Microsoft EMEA**