Application security assessment

De webapplicatie: even schudden voor gebruik!

Ing. R.J.A. Stouthart RE CISA

Bij een application security assessment worden door middel van diverse soorten tests kwetsbaarheden in webapplicaties geïdentificeerd. Met deze kennis kunnen webapplicaties gerichter worden gepantserd tegen aanvallen. In dit artikel worden de noodzaak en inhoud van een application security assessment aan de hand van theorie en praktijkvoorbeelden toegelicht. De doelgroep voor dit artikel wordt gevormd door ICT-auditors en management met verantwoordelijkheden inzake ICT-beveiliging. Deze beide doelgroepen zullen in application security assessments een probaat middel vinden om inzicht te krijgen in de status van de beveiliging van webapplicaties.

Inleiding

Application security assessments zijn ontstaan vanuit een concrete marktvraag. De hieraan ten grondslag liggende behoefte wordt als eerste beschreven, gevolgd door de probleemstelling: waarom eigenlijk een application security assessment. Vervolgens worden de oorzaken van kwetsbaarheden in webapplicaties toegelicht, aangevuld met mogelijke tegenmaatregelen. Daaropvolgend worden de soorten application security assessments verder toegelicht in combinatie met testdimensies die kunnen worden onderkend in de uitvoering van assessments.

Teneinde de lezer een gevoel te geven voor de meest voorkomende soorten kwetsbaarheden is een top-5 opgenomen. Daarnaast bevat dit artikel een uitgewerkt praktijkvoorbeeld van een veelvoorkomende kwetsbaarheid met de bijbehorende remedie. Als laatste is voorzien in een korte checklist met controlepunten voor de auditor die eens aan de slag wil met een application security assessment.

Aanleiding

Steeds meer organisaties bieden diensten aan via het internet. Waar vroeger slechts een enkeling verder ging dan een statische site, bieden tegenwoordig steeds meer organisaties diensten aan met een dynamisch karakter. Deze dynamiek wordt bereikt door gebruik te maken van webapplicaties die bedrijfsregels en koppelingen met databases bevatten. Zo verandert een marketingzuil in een e-businessapplicatie.

Om deze applicaties te realiseren richten organisaties in hoog tempo ontwikkelstraten in, scholen zij medewerkers bij of laten zij extern webapplicaties ontwikkelen. Kenmerkend hierbij is de korte time-to-market die wordt nagestreefd en de andersoortige tooling en ontwikkelmethodieken die worden toegepast in vergelijking met traditionele systeemontwikkelingstrajecten.

Tevens valt op dat grotere organisaties vaak een scheiding hebben doorgevoerd tussen de webdevelopmentafdeling en de staande traditionele systeemontwikkelingsafdelingen.

Voor het aanbieden van actuele en gepersonaliseerde informatie aan de gebruiker kan niet worden volstaan met statische HTML-pagina's, maar wordt gebruikgemaakt van programma's om de pagina's aan te maken die passen bij een klantprofiel. Dit kan de inhoud van de pagina betreffen, maar ook de opmaak of het aanpassen van de pagina aan het gebruikte browsertype (Web, Wap, GPRS e.d.). Daarbij vindt toepassing plaats van relatief eenvoudige programmeertalen zoals JavaScript, Java en Visual Basic, maar ook van complete op bijvoorbeeld MySAP of Siebel gebaseerde ontwikkelomgevingen.

Beveiliging heeft in de internetwereld een eigen plaats veroverd. Echter, tot op heden heeft de aandacht voor het treffen van beveiligingsmaatregelen zich met name toegespitst op beveiliging van de verbindingen met de boze buitenwereld. Hierbij moet worden gedacht aan bijvoorbeeld de obligatoire firewalls, intrusion detection systemen, public key infrastructuren en andere mogelijkheden om veilige verbindingen te realiseren.

De grenzen van de firewall

De voortdurende stroom van inbraken op webomgevingen maakt duidelijk dat voornoemde beveiligingsmaatregelen beperkingen kennen. De essentie van de beperkingen is gelegen in het feit dat voornoemde beveiligingsmaatregelen bedoeld zijn om verkeer door te laten. Met andere woorden, de beveiligingsmaatregelen die we normaal in infrastructuurschema's tegenkomen als een muurtje (de firewall en alles wat daarbij hoort), vormen in wezen geen muurtje maar een tunnel. Weliswaar zal een tunnel ervoor zorgen dat al te afwijkende verkeerssoorten (zoals vliegtuigen en olietankers) geen doorgang krijgen, maar de intenties van de inzittenden van toelaatbaar geachte verkeerssoorten (zoals tanks en vrachtauto's) blijven een onbekende factor en daarmee een risico.

Steeds meer inbraken en compromittaties van webomgevingen maken gebruik van het feit dat de firewall en consorten verkeer dienen toe te laten naar een achterliggende applicatie. De applicatie (en de daarin opgeslagen data) behoort juist tot de kroonjuwelen van een organisatie. Door de dreiging voor de applicatie ontstaat de noodzaak om ook op webapplicatieniveau beveiliging adequaat te regelen. Daartoe dient een stelsel van maatregelen te worden getroffen in en rondom de webapplicatie.

Vaak wordt gedacht dat het testen van deze maatregelen een regulier onderdeel is van een penetratietest of dat commerciële vulnerability scanners[1] zoals ISS en Nmap dit aspect meenemen. Dit is echter niet het geval omdat de standaard-penetratietest en een application security assessment andere aandachtsgebieden bestrijken.

Bij een penetratietest wordt gepoogd in te breken op een computer of netwerk door misbruik te maken van externe en interne netwerkdiensten om controle te krijgen over een computer, netwerk, de hierop draaiende applicaties en/of opgeslagen data. Bij een application security assessment wordt ditzelfde gepoogd maar dan door misbruik te maken van mogelijkheden in de applicatiearchitectuur en de applicatiefunctionaliteit.

Wat maakt een applicatie kwetsbaar?

Een applicatie is per definitie kwetsbaar omdat deze steunt op de integriteit van de omgeving waarbinnen de applicatie draait. Indien in de onderliggende lagen (zoals het netwerk en het besturingssysteem) zwakheden bestaan, zal de applicatie daardoor risico lopen. De onderliggende lagen vormen voor de applicatie de general IT controls waaraan moet zijn voldaan. Veel applicatiehacks slagen omdat binnen het besturingssysteem onvoldoende controles worden uitgevoerd op de wijze waarop applicaties worden aangeroepen. Dit is echter maar een deel van het verhaal.

Een andere bron van kwetsbaarheden is een gevolg van de ontwikkeling dat applicaties niet meer uit één geheel bestaan maar uit een groot aantal losse componenten. Het gebruik van componenten is eigenlijk een terugkeer naar de tijden van het mainframe, waar traditioneel reeds met kleine servermodules werd gewerkt in plaats van met onhandelbare grote brokken software op pc's, zoals gebruikelijk werd in het client-servertijdperk.

De belangrijkste reden waarom componenten binnen webomgevingen tot kwetsbaarheden leiden is gelegen in het ontbreken van vaste toegangspaden. De toegangspaden zijn weliswaar gedefinieerd vanuit de proceslogica, maar de applicatie en de omgeving dwingen de toegangspaden niet meer af. Als gevolg hiervan kunnen gebruikers zelf proberen om componenten aan te roepen in een afwijkende volgorde. Het gevolg hiervan is dat een gebruiker de context waarbinnen een component is bedoeld te functioneren, kan manipuleren. Wanneer de applicatie hier niet op bedacht is, kan dit leiden tot onvoorspelbaar en daarmee risicovol gedrag.

Een webapplicatie maakt gebruik van parameters om een context te creëren bij een aanroep van een applicatieonderdeel. De parameter kan bijvoorbeeld een taalkeuze zijn, een rentepercentage of de gebruikersnaam waarmee een gebruiker is ingelogd. Als niet alle parameters en (dus) mogelijke gebruikersinvoer worden gecontroleerd, bestaat de mogelijkheid dat onverwachte parameters leiden tot fouten bij de verwerking van de parameters, zoals buffer-overflows, uitvoeren van oneigenlijke commando's, enz. In wezen kan hier worden gesproken van tekortkomingen in de programmatuur. Binnen de programmatuur dient daarom voortdurend controle plaats te vinden op lengte, vorm, inhoud en samenhang van parameters.

Een risicofactor uit een heel andere hoek is de relatieve onvolwassenheid van de technologie die wordt ingezet bij het realiseren en draaien van webapplicaties. Deze relatieve onvolwassenheid is een gevolg van de onstuimige ontwikkelingen op dit gebied, het tekort aan ervaren webontwikkelaars en de inzet van technologie en standaarden die nog niet uitgekristalliseerd en/of 'proven technology' zijn.

De relatieve onvolwassenheid van de technologie die wordt ingezet bij het realiseren en draaien van een webapplicatie, maakt zo'n applicatie kwetsbaar.

Tegenmaatregelen

Om voorgaande risico's en zwakheden het hoofd te bieden is een diversiteit aan weloverwogen maatregelen nodig. De belangrijkste maatregelen worden hierna beschreven en vormen zoals altijd een combinatie van organisatie en techniek. Opgemerkt wordt dat een veilige webapplicatie een utopie is, een veiligere webapplicatie is het maximum van wat men zal bereiken.

De belangrijkste te treffen maatregel betreft het bouwen onder architectuur. Bouwen onder architectuur staat hierbij voor het ontwikkelen vanuit een integrale visie op zowel het product als de randvoorwaarden waarbinnen het product moet functioneren. Eén van die randvoorwaarden is beveiliging. Door de wereldwijde toegang staat een webapplicatie bloot aan risico's die bij programmatuur in besloten omgevingen niet bestaan of van een andere dimensie zijn. Om deze reden dient beveiliging nog nadrukkelijker te worden geïncorporeerd in het ontwerp-, ontwikkel- en test- en acceptatietraject.

Een mogelijke maatregel voor het incorporeren van beveiliging in het ontwerp- en ontwikkeltraject is het uitvoeren van een intern gerichte application security assessment. Hierin wordt de opzet van de applicatie onderzocht betreffende de wijze waarop de te verwachten risico's worden onderkend en adequaat gemitigeerd. Deze sterk op risicoanalyse gerichte maatregel zou idealiter deel moeten uitmaken van een governancestrategie ten aanzien van systeemontwikkeling. Denk hierbij aan application security assessments als onderdeel van quality assurance tijdens een ontwikkeltraject.

1) Vulnerability scanners zijn tools die op basis van bibliotheken met bekende kwetsbaarheden netwerkcomponenten op afstand onderzoeken op het voorkomen van bekende kwetsbaarheden.

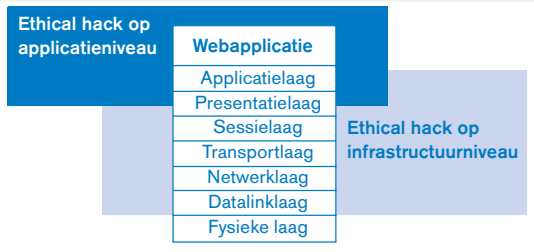
Ook kan een application security assessment worden ingezet als onderdeel van een testproces, liefst nog voor de gebruikersacceptatietest en zeker voor de lifetests. Dit is mogelijk doordat de externe application security assessment niet ingaat op de onderliggende infrastructuur en dus niet hoeft te worden uitgevoerd (en dus te wachten) op een productieomgeving.

Doelstelling application security assessment

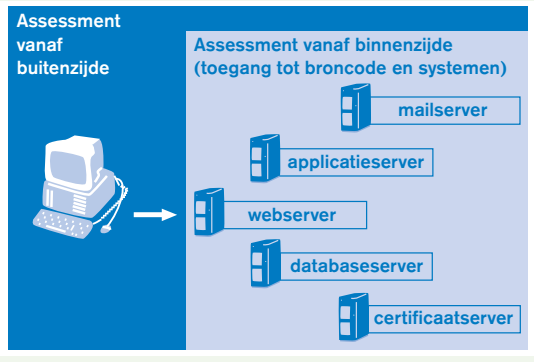
Een application security assessment heeft als doelstelling te onderzoeken in hoeverre de op webapplicatieniveau getroffen beveiligingsmaatregelen daadwerkelijk het gewenste niveau van beveiliging bereiken.

Om te onderzoeken in hoeverre getroffen beveiligingsmaatregelen daadwerkelijk het gewenste niveau van beveiliging realiseren, wordt gezocht naar kwetsbaarheden in de applicatie. Deze kwetsbaarheden kunnen het gevolg zijn van tekortkomingen in de applicatielogica, in de applicatiearchitectuur of in de onderliggende infrastructuur. Vaak treden op meerdere niveaus kwetsbaarheden op die, in combinatie, zorgen voor exploiteerbare gaten in de beveiliging.

Een application security assessment is in wezen een uitbreiding op de standaard-penetratietest. Deze uitbreiding betreft de toespitsing bij een application security assessment op het logicagedeelte van een webomgeving, terwijl de standaard-penetratietest zich voornamelijk richt op de infrastructuur. Een application security assessment wordt daarom in de volksmond vaak een applicatie-hack genoemd. Het verschil in scope tussen de standaard-penetratietest en een application security assessment kan worden toegelicht aan de hand van het welbekende OSI-model.



Figuur 1. Ethical hacking en het OSI-model.



Figuur 2. Twee uitvoeringsvarianten van de application security assessment.

In figuur 1 is schematisch weergegeven hoe aanvallen op webapplicatie- en infrastructureel niveau zich tot elkaar verhouden. Kenmerkend hierbij is dat een aanval op het niveau van de webapplicatie zich concentreert op de mogelijkheden om de aangeboden functionaliteit te manipuleren en te compromitteren vanuit de applicatie (de functionaliteit). Een aanval op het niveau van de webapplicatie zal dus zijn gericht op de specifieke functionaliteit van de applicatie en is steeds uniek.

Bij een penetratietest daarentegen wordt getest op basis van (combinaties van) bekende zwakheden. De aanval op infrastructureel niveau richt zich op zwakheden onder de applicatielaag. Hierbij wordt gepoogd om bijvoorbeeld een firewall, webserver, router en dergelijke te manipuleren en compromitteren.

In wezen maakt het voor de uitvoering van de application security assessment niet uit of het een webapplicatie betreft, een traditionele client-serverimplementatie of een 3-tier ERP-implementatie; waar functionaliteit wordt aangeboden, ligt een mogelijkheid voor misbruik en waar gebruikers kunnen kiezen, dient adequaat te worden gecontroleerd. De reden dat application security assessments voornamelijk in webomgevingen worden uitgevoerd, is gelegen in het besef dat in deze omgeving een organisatie maar beperkt controle kan uitoefenen over de webapplicatie die zij aanbiedt. In interne omgevingen is deze controle in grotere mate aanwezig en zullen gebruikers in mindere mate via trucs proberen om applicaties naar hun hand te zetten. De ervaring leert ons dat gebruikers dat heel vaak al kunnen dankzij inadequate (lees te ruime) autorisatieprofielen; trucs zijn dan niet meer nodig.

Uitvoering

Een application security assessment kan worden uitgevoerd van binnen uit en van buiten af. Wanneer van buiten af wordt gewerkt, is in wezen sprake van een ethical application hack waarbij wordt gepoogd om door gebruik te maken van tekortkomingen in de applicatie een omgeving te compromitteren.

Een intern gerichte application security assessment beziet de binnenkant van een webapplicatie en maakt het mogelijk de opzet van de applicatiearchitectuur en de gebruikte ontwikkelomgevingen te evalueren op de mate waarin de gemaakte keuzen bijdragen aan een adequate beveiliging van de webapplicatie. Ook de mogelijkheid om broncode te reviewen draagt hieraan bij. Daarnaast kan een intern gerichte application security assessment, doordat wordt beschikt over broncode, documentatie en medewerking van de ontwikkelaars, ook andere kwaliteitskenmerken dan beveiliging onderkennen.

In figuur 2 zijn de twee uitvoeringsvarianten schematisch weergegeven. Links wordt het perspectief van de externe application security assessment geschetst, waarbij met trucs, tools zoals password-crackers en andere middelen wordt gepoogd de webapplicatie te manipuleren. Aan de rechterzijde wordt het perspectief geschetst van de interne application security assessment, waarbij openheid wordt gegeven ten aanzien van de opzet en implementatie van de webapplicatie.

Een extern gerichte application security assessment kan het best worden vergeleken met een standaard-penetratietest waarbij wordt gepoogd om van buiten af een omgeving binnen te dringen met behulp van trucs, tools en brute kracht. Bij een extern gerichte application security assessment wordt ditzelfde gepoogd op het niveau van een webapplicatie in plaats van op het niveau van de infrastructuur.

Testdimensies

Om een vraag als 'is onze webapplicatie veilig' genuanceerd te kunnen beantwoorden kan niet worden volstaan met een 'ja, we kwamen er niet doorheen' of een 'nee, volgende keer beter'. De opdrachtgever dient inzicht te krijgen in de mate van risico die hij loopt wat betreft de veiligheid van de webapplicatie. Dit inzicht kan worden bewerkstelligd door testdimensies te introduceren waartegen gemeten en gerapporteerd wordt. Door middel van testdimensies kan het abstracte begrip veiligheid worden geconcretiseerd naar bijvoorbeeld mensen, middelen en tijd.

Mate van autorisatie

Verschillende soorten gebruikers zullen verschillende soorten toegangsprofielen hebben in een webapplicatie. Bij het uitvoeren van application security assessments worden eerst de diverse soorten toegangsprofielen in kaart gebracht en wordt in samenwerking met de klant besloten welke toegangsprofielen worden getest. Hieronder worden de meest voorkomende toegangsprofielen beschreven.

Onbekende gebruiker
Hierbij wordt uitgegaan van een situatie waarbij de application security assessment-specialist niet de beschikking heeft over een gebruikersnaam, cookie of token waaraan de applicatie hem kan herkennen. In deze situatie dient de webapplicatie te waarborgen dat de gebruiker geen toegang krijgt tot de webapplicatie of alleen tot het publieke gedeelte.

Bekende klant
Hierbij wordt uitgegaan van een situatie waarbij een application security assessment-specialist wél de beschikking heeft over een gebruikersnaam, cookie of token en dat de applicatie hem herkent als een geldige gebruiker. In deze situatie dient de webapplicatie te waarborgen dat de gebruiker slechts toegang krijgt tot de gedeelten van de webapplicatie die voor het betreffende toegangsprofiel geautoriseerd zijn.

Binnen de situatie van een bekende klant worden de volgende subscenario's onderkend:
* geen beschikking over transactie-autorisaties;
* wel beschikking over transactie-autorisaties;
* beschikking over beheerautorisaties.

Voor ieder (sub)scenario worden de zwakheden in het identificatie-, autorisatie- en authenticatiemechanisme en de (achterliggende) technische infrastructuur vastgesteld en waar mogelijk uitgebuit. Tevens wordt vastgesteld of een situatie kan worden gecreëerd waarbij een gebruiker zich voordoet als een andere gebruiker om daarmee toegang te krijgen tot gegevens die tot het domein van deze andere gebruiker behoren.

Door het identificeren van toegangsprofielen kan een onderscheid worden gemaakt naar de soorten functionaliteit en de bijbehorende gebruikers, en naar de risico's die de bewuste onderdelen van de webapplicatie met zich meebrengen.

Mate van kennis en vaardigheden

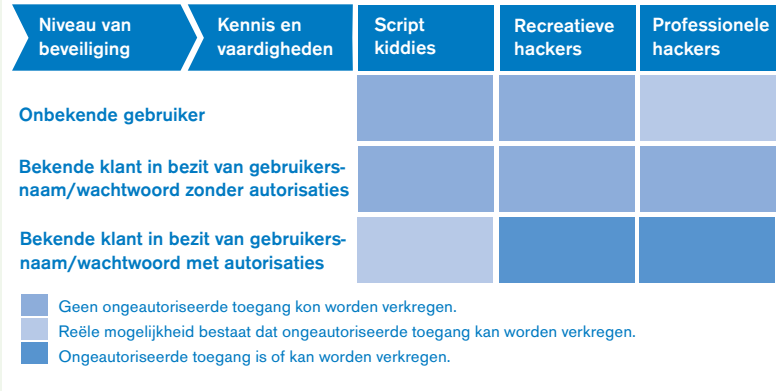
Niet iedereen heeft dezelfde kennis, vaardigheden, tools en drijfveren. KPMG maakt bij haar application security assessments meestal onderscheid in verschillende soorten aanvallers om helder te kunnen maken welke mate van weerbaarheid een webapplicatie biedt tegen aanvallen en aanvallers. KPMG onderscheidt binnen dit type van opdrachten drie soorten aanvallers:
* Een *script kiddy* is iemand die over het algemeen zal trachten via bekende fouten zwakheden in de programmatuur uit te buiten via een beperkt aantal tools van derden. Script kiddies zijn als gauwdieven: een probleem als de deur openstaat, maar een willekeurig slot houdt hen tegen.
* Een *recreatieve hacker* zal zich niet beperken tot reguliere toegangswegen, maar zal trachten zelfstandig alternatieve toegangswegen te vinden en kleinere zwakheden te combineren tot een gapend gat. Een recreatieve hacker is op zoek naar uitdagingen maar niet naar zwaar werk.
* Een *professionele hacker* is iemand die, op grond van mogelijkheden die hij of zij waarneemt, zal trachten zich toegang te verschaffen tot een site in het bijzonder. Daarbij wordt een groot arsenaal aan wapens ingezet, gecombineerd met grondige kennis en ruime ervaring.

Door het identificeren van soorten aanvallers kan een onderscheid worden gemaakt naar de kans op een aanval en de soort aanval, en naar de inspanningen die gepaard gaan met beveiliging tegen de onderscheidene aanvallen op de webapplicatie.

Een ICT-auditor die een application security assessment uitvoert, kan overigens niet met een hoger niveau van kennis en vaardigheden toetsen dan hij zelf bezit.

Rapportagemogelijkheden

Wanneer de testdimensies worden meegenomen in de opdrachtuitvoering kunnen deze ook worden gehanteerd binnen de application security assessment-rapportage. Figuur 3 geeft een rapportagevoorbeeld waarin de testdimensie van autorisatie en de testdimensie van kennis en vaardigheden tegen elkaar worden afgezet in een matrix.



Figuur 3. Rapportagevoorbeeld van de testdimensies autorisatie versus kennis en vaardigheden.

Top-5

Hoewel iedere webapplicatie haar eigen kwetsbaarheden kent en op een andere manier moet worden gehackt, kan wel een top-5 van veelvoorkomende kwetsbaarheden worden onderkend. Hieronder volgen van hoog naar laag de vijf meest voorkomende kwetsbaarheden die KPMG benut om een webapplicatie te manipuleren en compromitteren. Daar waar voorbeelden zijn gegeven, is gebruikgemaakt van de fictieve site www.InteractiveLifestyle.nl.

1. De webserver is onvoldoende beveiligd
Hoewel strikt genomen kwetsbaarheden in de webserver niet tot de webapplicatie behoren maar tot de infrastructuur waarop de webapplicatie draait en waarvan zij afhankelijk is, vormt deze kwetsbaarheid toch de onbetwiste nummer één. Bijna dagelijks worden nieuwe kwetsbaarheden ontdekt in de gangbare webservers waarmee hackers zich toegang kunnen verschaffen tot deze webservers. Vandaar uit kunnen zij dan proberen broncode van webapplicaties van binnen te analyseren op kwetsbaarheden dan wel broncode proberen aan te passen om vervolgens te werken aan de totale overname van een omgeving.

2. Gebruikersinvoer wordt onvoldoende gecontroleerd
Eén van de grootste risico's van webprogrammatuur wordt gevormd door het niet of onvoldoende controleren van gebruikersinvoer. Door slimme combinaties van valide invoer en speciale tekens kunnen via de gebruikersinvoer in bepaalde gevallen commando's worden gegeven aan het besturingssysteem, de webserver en/of de webapplicatie. Zie ook het praktijkvoorbeeld dat in bijlage 1 is gegeven.

3. Parameters worden onvoldoende gecontroleerd
Pagina's geven vaak gegevens aan elkaar door om op deze wijze de context waarbinnen een gebruiker zich bevindt, te bewaren. Hieronder ziet u een voorbeeld waarin een pagina wordt opgeroepen op een site waarin de context wordt bepaald door twee variabelen: één voor het klantnummer en één voor de naam van de opgeroepen pagina:

```
http://www.InteractiveLifestyle.nl/VerkoopEffect.asp?
UserID=3232+Pagina=KiesRekening
```

In dit voorbeeld is een pagina gebruikt die is bedoeld voor directe aanroep door de gebruiker vanuit diens browser. Daarnaast bevat een webapplicatie vaak nog talloze scripts en pagina's die alleen zouden moeten worden aangeroepen als onderdeel van een andere pagina: via een gecontroleerd toegangspad. In dat geval is het toegangspad vertrouwd en kan in bepaalde mate worden gesteund op de integriteit van de aanroepende partij, in casu een pagina op de webserver. Wanneer een gebruiker echter zo'n script of pagina direct aanroept vanuit de browser, kan de gebruiker zelf bepalen met welke parameters hij een script of pagina wil aanroepen. In dat geval zal een gebrek aan parameterinvoercontroles direct leiden tot zwakheden in de webapplicatie.

4. De opvang en afhandeling van uitzonderingssituaties en fouten is niet adequaat
Veel webapplicaties houden onvoldoende rekening met uitzonderingssituaties en fouten in de verwerking. Met het oog op performance kan een webapplicatie bij het verwerken van uitzonderingssituaties en fouten zelfs besluiten om controles achterwege te laten. Een andere veelvoorkomende mogelijkheid is om door middel van fouten systeeminformatie te achterhalen.

Hieronder volgt een voorbeeld van de informatie die één van de meest gebruikte webservers verstrekt wanneer een pagina wordt aangeroepen die een numerieke waarde verwacht maar deze niet krijgt. In onderstaand voorbeeld wordt in plaats van een productnummer een compleet databasecommando meegegeven. Door de combinatie van het niet voldoende controleren van de invoer en het niet voldoende opvangen van een fout geeft de webserver informatie over de achterliggende database opeens bloot[2].

Voor de volledigheid, de fout in kwestie is dat de database bij het zoeken met een ProductID verwacht dat het ProductID numeriek is. Het resultaat (de uitvoer) van het opgegeven databasecommando achter ProductID is echter tekst (versiedetails van het merk en type database), waardoor de database een mismatch constateert tussen een numeriek veld en een tekstwaarde.
Invoer van de gebruiker:

```
http://www.InteractiveLifestyle.nl/items/detail.asp?Pro-
ductID=(select%20@@version)
```

Uitvoer van de webapplicatie:

```
[Microsoft][ODBC SQL Server Driver][SQL Server]
Syntax error converting the nvarchar value 'Microsoft
SQL Server 7.00 - 7.00.623 (Intel X86) Nov 27 1998
22:20:07 Copyright (c) 1988-1998 Microsoft Corpo-
ration Small Business Server Edition on Windows NT
4.0 (Build 1381: Service Pack 5)' to a column of data-
type int.
```

Met deze informatie kan vervolgens weer een volgend aanvalsscenario worden uitgewerkt, bijvoorbeeld door het zoeken in kwetsbaarheidsbibliotheken op internet (zoals www.securityfocus.com) naar kwetsbaarheden voor deze specifieke versie van de databaseserver.

2) Remko Stouthart, *Soluties voor lekke websites*, 1998.

Indien men bekend is met een type database (of een handboek downloadt op internet), kan men op deze manier ook achterhalen wat de gebruikersnaam is waarmee de verbinding tussen de webserver en de databaseserver wordt gemaakt. Dit gegeven is dan weer voldoende om het wachtwoord van de betreffende gebruiker te wijzigen, met een vastgelopen website als resultaat. Voor wie echt wat wil bereiken, bestaat dan nog de mogelijkheid om direct thuis commando's uit te gaan voeren op de webserver.

Voor de Microsoft SQL-database in bovenstaand voorbeeld is het 'xp_cmdshell'-commando daarvoor een goede optie. Met dit commando kan de databaseserver een opdracht laten uitvoeren door het besturingssysteem waarop de databaseserver draait. Binnen webapplicaties draaien de database- en de webserver vaak op dezelfde server. Met andere woorden, in dat geval kan een gebruiker door het intypen van een URL via de database een commando uitvoeren op het besturingssysteem van de webserver.

5. De programmatuur bevat nog elementen uit de bouw- en testfase

Vaak bevat een webapplicatie nog sporen uit de bouw- en testfase of, erger nog, voorbeelden van de leverancier van de webserver of de ontwikkeltools. Deze voorbeelden zijn bekend, opvraagbaar en bevatten talloze zwakheden (vaak gerelateerd aan te ruime privileges) omdat ze zodanig zijn ontwikkeld dat ze altijd werken en zoveel mogelijk functionaliteit gelijktijdig willen demonstreren: veiligheid is geen issue voor een voorbeeld van functionaliteit.

Afsluitend

In dit artikel is beschreven wat een application security assessment is en waarom het kan worden beschouwd als een krachtig middel ter controle van de veiligheid van een webapplicatie tegen de verfijndere aanvalssoorten. Het is in de praktijk zowel een zinvolle aanvulling geworden op de traditionele penetratietests als een zelfstandige opdrachtsoort die graag door internetdienstverleners wordt ingezet.

Een application security assessment is gericht op het identificeren van kwetsbaarheden, en geeft een betrouwbaar, maar geen absoluut beeld omtrent het volledige scala van zwakheden in de beveiliging van een webapplicatie. Daarnaast wordt de volledigheid en diepgang in grote mate bepaald door de factoren tijd en kennis. Dagelijks worden nieuwe zwakheden in producten en standaardapplicaties gevonden die weer leiden tot nieuwe aanvalsmogelijkheden. Daarmee is de zeggingskracht van een externe application security assessment sterk tijdgebonden.

Een externe application security assessment komt te laat in het applicatieontwikkeltraject om nog actief te kunnen sturen in het proces. Om als webapplicatie goed te scoren bij een application security assessment moeten de maatregelen al in het totstandkomingsproces zijn ingebakken in de organisatie en de techniek. De interne application security assessment kan wel die rol spelen als onderdeel van een governancestrategie tijdens het totstandkomingsproces.

De externe application security assessment kan dan worden ingezet als 'proof-of-the-pudding' voordat de webapplicatie live gaat en is dan te vergelijken met de laatste controle in een televisiefabriek. Flink bonken op en schudden met de tv en kijken of het beeld niet verstoord raakt; pas dan gaat de tv de wereld in. Kortom: even schudden voor gebruik.

Ing. R. (Roeland) J.A. Stouthart RE CISA
is als senior manager werkzaam bij KPMG Information Risk Management in Amstelveen en houdt zich binnen de unit e-Business Security bezig met beveiligingsarchitecturen en de problematiek rondom grootschalige IT-omgevingen.

Bijlage 1: Praktijkvoorbeeld

Om de risico's rondom een webapplicatie toe te lichten wordt een eenvoudig voorbeeld gegeven om kwetsbaarheden en de wijze van exploitatie te verduidelijken. Dit voorbeeld is gebaseerd op de dagelijkse praktijk van broncodeanalyse en soortgelijke tests behoren tot het standaardrepertoire voor application security assessments.

Men neme een fictieve uitgeverij genaamd 'InteractiveLifestyle'. Natuurlijk heeft 'InteractiveLifestyle' een website: www.InteractiveLifestyle.nl. Via die site kan men een abonnement aanvragen op haar glossy en fictieve magazine genaamd 'Connect2Vision'.

Door het klikken op de link 'Abonnement' activeert de gebruiker de volgende URL:

```
https://www.InteractiveLifestyle.nl/Eformulieren/
p_Eform_Generator.asp?FormID=323
```

De browser maakt daarbij een met SSL beveiligde verbinding naar de InteractiveLifestyle webserver. Nu kunnen de heen en weer gaande sessiegegevens niet meer zomaar worden gebruikt door derden. Over de betrouwbaarheid van het verkeer dat wij over de lijn gaan sturen, zegt dit echter niets!

De complete URL naar de abonnementspagina wordt niet getoond op het scherm, de statusbalk of de URL-regel boven in de browser. Blijkbaar heeft InteractiveLifestyle dit afgeschermd omdat ze de opbouw van haar site niet zomaar bekend wil maken. Echter, door middel van de history-dropdownlist is de volledige URL eenvoudig te achterhalen.

Aan de URL te zien maakt InteractiveLifestyle gebruik van een Microsoft internetserver (.asp is een typische Microsoft-extensie) en werkt de webserver met webpagina's met een dynamische inhoud die wordt bepaald door de parameter FormID. FormID zal dus staan voor een sleutel waarmee een bepaalde context kan worden teruggevonden in een database of zo.

Als FormID 323 kan, dan kan FormID 322 ook vast wel. Daarom typen we de URL weer in en veranderen het laatste cijfer in een 2.

```
https://www.InteractiveLifestyle.nl/Eformulieren/
p_Eform_Generator.asp?FormID=322
```

We kunnen dus eenvoudig zelf bepalen wat de waarde van de parameter wordt die de webserver gaat verwerken. Daarmee krijgen we de mogelijkheid om:

* allerlei (typen) waarden uit te proberen.

Hierbij kan worden gedacht aan extreem grote getallen, negatieve getallen, breuken en formules. Hierbij wordt getest hoe robuust de fout- en exceptieafhandeling van de webapplicatie is.

* allerlei schermen direct aan te roepen, de normale toegangspaden daarmee omzeilend.

Hiermee kan snel een inzicht worden verkregen in de gehele functionaliteit die achter een webapplicatie schuilt. Daarnaast kan een gebruiker door het direct aanroepen van een pagina zelf een context creëren voor de pagina, een context die afwijkt van wat de applicatie verwacht, waardoor bijvoorbeeld applicatiecontroles worden gefrustreerd.

* de wijze waarop de parameter in de webapplicatie wordt verwerkt verder te onderzoeken.

Wanneer we kunnen uitvinden hoe een parameter aan de binnenzijde van een applicatie wordt verwerkt, dan maximaliseren we de kans om de parameter zodanig te manipuleren dat we een achterliggend proces naar onze hand kunnen zetten. Kennis is en blijft nu eenmaal macht.

Als we kiezen voor het onderzoeken van de parameter FormID, dan dienen we te weten hoe deze parameter wordt gebruikt en wat de mogelijkheden ervan zijn. De manier waarop een dergelijke parameter normaal wordt gebruikt in programmatuur waarbij een gegeven moet worden gevonden in een database, is ongeveer als volgt:

```
SQL = ''SELECT Form FROM Forms WHERE
FormID = '' + FormID
```

waarbij SQL een zoekopdracht is in een database en het FormID het – zelf opgegeven – nummer van de pagina is. Dus bij de opdracht:

```
https://www.InteractiveLifestyle.nl/Eformulieren/
p_Eform_Generator.asp?FormID=322
```

wordt – waarschijnlijk – de volgende opdracht uitgevoerd in de webserver:

```
/* Bouw de zoekopdracht */

SQL = ''SELECT Form FROM Forms WHERE
FormID = '' + FormID

/* Doe de zoekopdracht */

EXECUTE SQL
```

Wat is hier, afgezien van het verkeerde nummer, eigenlijk mis mee? Wat er mis mee is, is dat de webapplicatie niet adequaat controleert of FormID een valide waarde bevat. Daardoor kunnen we via een slimme manipulatie van de parameter FormID nog andere opdrachten geven aan de database. Dit kan doordat de programmeertalen binnen databases de mooie eigenschap hebben om SQL-opdrachten te scheiden met een puntkomma (;).

Dit wetende kunnen we de URL aanpassen met de volgende opdracht:

```
https://www.InteractiveLifestyle.nl/Eformulieren/
p_Eform_Generator.asp?FormID=322;
DELETE * FROM FORMS
```

Dan wordt waarschijnlijk de volgende opdracht uitgevoerd:

```
/* Bouw de zoekopdracht */

SQL = ''SELECT Form FROM Forms WHERE
FormID = 322; DELETE * FROM FORMS''

/* Doe de aangepaste zoekopdracht */

EXECUTE SQL
```

Het is duidelijk dat als deze opdracht daadwerkelijk wordt uitgevoerd, alle form-definities uit de database verwijderd zijn en dat de inhoud van de site daardoor waarschijnlijk een stuk minder interessant wordt voor de gemiddelde gebruiker!

In wezen is het voorgaande een voorbeeld van de combinatie van het kenmerk dat in webomgevingen toegangspaden niet vaststaan met het gegeven van een inadequate parameterafhandeling bij het aanroepen van een applicatieonderdeel. Om de parameterafhandeling onder controle te krijgen kan de volgende validatieconstructie worden gebruikt:

```
/* geen parameter of een niet-numerieke parame-
ter */

IF FormID = '' '' OR NOT Isnumeric (FormID)
THEN

   /* doorverwijzen naar een pagina met een
passende melding */

   Response.redirect (''p_waarzijnwijmeebezig.asp'')

ELSE

   /* wel een numerieke parameter */

   SQL = ''SELECT Form FROM Forms
WHERE FormID = '' + FormID

   /* Doe de zoekopdracht */

   EXECUTE SQL

END IF
```

Door het toevoegen van een klein stukje code is de webapplicatie weer iets veiliger geworden. Echter, iedere parameter moet op deze wijze worden beveiligd. Daarnaast zijn er veel meer soorten manipulaties mogelijk op de parameter FormID dan alleen het sturen van een commando-separator (de ;). Andere mogelijkheden zijn bijvoorbeeld de tekst '/0' waardoor sommige webapplicaties denken dat het einde van een string is bereikt en dus de controle stoppen. Al met al legio mogelijkheden voor een onderzoekende geest. En iedere dag worden gaten gedicht en nieuwe mogelijkheden ontdekt.

Bijlage 2: Controlepunten voor de auditor ten aanzien van broncode

Hierna volgt een lijst van controlepunten die KPMG hanteert bij de assessment van beveiligingsmaatregelen in de broncode. Hierbij wordt tijdens de assessment de nadruk gelegd op de adequaatheid van de parametervalidatie en foutafhandeling.

Architectuureisen
* Stel kwaliteitseisen aan de programmeeropzet, -omgeving en -resultaten.
* Voer opdrachten uit aan de serverkant waar mogelijk.
* Vertrouw niet op de omgevingsinstellingen.
* Vertrouw niet op de client.
* Neem geen gevoelige gegevens op in code.
* Gebruik geen geïnterpreteerde code als gecompileerde code ook mogelijk is.

Verwerkingseisen
* Controleer alle gebruikersinvoer.
* Controleer alle parameters.
* Geef bij controles eerst aan wat *wel* mag, dan wat *niet* mag.

Foutafhandeling
* Zorg voor opvang van uitzonderingssituaties.
* Zorg voor een goede foutafvanging en foutafhandeling.

Parameterbeheersing
* Geef geen gevoelige informatie door via de URL of via cookies.
* Geef geen ongevalideerde opdrachten aan een systeem-shell.
* Vermijd het gebruik van functies met systeemrechten.
* Neem geen systeemcommando's op in scripts of includes.
* Gebruik geen verborgen variabelen (ze worden toch ontdekt).
* Gebruik geen statische buffers.

Implementatie-eisen
* Log en controleer gebruikers die fouten veroorzaken.
* Draai een script nooit met systeemrechten.
* Maak scripts niet langer of complexer dan strikt noodzakelijk.
* Bewaar scripts in één directory.