Radboud Universiteit Nijmegen

Masterscriptie

Cross-site scripting voorkomen

Auteur:
R. Ben Moussa

Begeleider:
Prof. dr. M. van Eekelen

Scriptienummer:
611

25 augustus 2011

Radboud Universiteit Nijmegen

Masterscriptie

Cross-site scripting voorkomen

Auteur:	R. Ben Moussa
Email:	r.benmoussa@student.ru.nl
Studentnummer:	s0544205
Begeleider:	Prof. dr. M. van Eekelen
Co-referent:	dr. S. Smetsers
Scriptienummer:	611

25 augustus 2011

Inhoudsopgave

Samenvatting

In deze scriptie is onderzoek gedaan naar het voorkomen van veelgemaakte fouten in software. Als eerste zijn alle punten uit de CWE/SANS top 25 van meest gevaarlijke programmeerfouten geanalyseerd. Per fout in deze top 25 is er een korte toelichting gegeven. Tevens is er gekeken hoe goed het punt te voorkomen is en of de fout statisch of dynamisch analyseerbaar is. Naar aanleiding van dit vooronderzoek is gebleken dat Cross-site scripting één van de meest gevaarlijke en meest voorkomende fouten is.

Om de omvang van het onderzoek te beperken is er gericht op het voorkomen van Cross-site scripting in Java webapplicaties. Een webapplicatie is gevoelig voor Cross-site scripting wanneer invoerparameters direct in de uitvoer worden gezet. Uitvoer wordt in de gegevenscontext van een webapplicatie gezet. Voorbeelden van verschillende contexten zijn HTML-body, JavaScript en CSS. Wanneer deze uitvoer niet voor de juiste gegevenscontext wordt geëncodeerd kan het als 'code' worden geïnterpreteerd, wat aanvallers de mogelijkheid geeft om kwaadaardige code te injecteren. Om dit te voorkomen dient elk stukje uitvoer voor de *juiste* context te worden geëncodeerd. De *juiste* context is de gegevenscontext waarin uitvoer geplaatst gaat worden. Omdat de presentatielaag bepaalt in welke context de uitvoer terechtkomt, moet in deze laag ook de encodering plaatsvinden.

Voor dit onderzoek is er ingegaan op Velocity, een presentatielaag voor Java webapplicaties. Om van een webapplicatie te controleren of alle uitvoer is geëncodeerd is de Velocity Anti-XSS tool ontwikkeld. Deze tool checkt Velocity-code door middel van statische analyse en geeft aan op welke plekken er uitvoer wordt gedaan zonder deze te encoderen.

Cross-site scripting kan uitgesloten worden wanneer de webapplicatie aan de volgende twee eisen voldoet:

1. Elke vorm van uitvoer is geëncodeerd voor de juiste context.

2. Het encoderen gebeurt middels een veilig bewezen routine of bibliotheek.

De manier waarop de Anti-XSS tool werkt is breder toepasbaar. Er dient een uniforme methode te worden gebruikt voor het encoderen. Als de plekken (of aanroepen) in de presentatielaag waar uitvoer wordt gedaan bekend zijn, kan hierop gecontroleerd worden.

Summary

This thesis conducts a research to preventing the most significant programming errors that can lead to serious software vulnerabilities. First, all errors of the top 25 were analysed. For each error an explanation is given and to what extent it can be prevented, static analyzed and dynamic analyzed. This preliminary studie revealed that Cross-site scripting occurs frequently and is often easy to exploit.

To limit the scope of this project the further research will focus on preventing Cross-site scripting in Java web applications. A web application is vulnerable for Cross-site scripting when data retrieved from user input is directly printed as output. These input parameters are printed in the data context of the web application. Examples of different contexts are the HTML body, JavaScript and CSS. When output is not encoded for the *right* context, it can be intrepreted as code by the browser, which allows attackers to inject malicious code. To prevent code injection each output should be encoded properly for the context it's printed in. The context where the output will be printed is defined by the presentation layer of the web application. Therefore, to determine the right encoding, it needs to be applied in the presentation layer.

This research aims on Velocity as a presentation layer for Java based web applications. To check if all output is encoded, the Velocity Anti-XSS tool is developed. The Anti-XSS tool performs static analysis of the code and points out spots of output which aren't encoded.

Cross-site scripting can be prevented when the web applications meets two requirements:

1. Any output is encoded for the right context
2. Encoding is handled by a routine or library which is proven secure

The working of the Anti-XSS tool is broadly applicable. The tool merely requires a uniform method for encoding. When all types of output (or calls) in the presentation layer are known, a tool can verify if all output is encoded.

Hoofdstuk 1

Inleiding

De meeste zwakheden in software worden veroorzaakt door een klein aantal programmeerfouten. Door het identificeren van onveilige programmeergewoonten en het ontwikkelen van veilige alternatieven kunnen deze fouten worden gereduceerd of geëlimineerd tijdens de ontwikkelfase.

1.1 De CWE/SANS top 25

CWE/SANS heeft een top 25 samengesteld van de meest gevaarlijke programmeerfouten die tot serieuze zwakheden in software kunnen leiden. Deze fouten staan in deze lijst omdat ze regelmatig voor komen, relatief gemakkelijk te vinden zijn en ook meestal makkelijk te misbruiken zijn. Deze fouten zijn gevaarlijk omdat ze aanvallers de mogelijkheid geven om controle over de software over te nemen, gegevens te stelen of te voorkomen dat de software überhaupt werkt.

De CWE verschaft een meetbare, eenduidige verzameling van softwarezwakheden. Dit zorgt ervoor dat er effectief gekeken kan worden naar oplossingen om zwakheden te voorkomen.

1.2 Kwaliteit van webapplicaties

De programmeerfouten uit de CWE/SANS top 25 gelden voor software in het algemeen, in dit onderzoek zal worden ingegaan op de kwaliteit van webapplicaties. Er wordt gekeken in hoeverre kwaliteit gewaarborgd kan worden met behulp van de CWE/SANS top 25. De top 25 geeft een weergave van fouten die vaak voorkomen. Gezocht zal worden naar een manier om dergelijke fouten aantoonbaar te kunnen uitsluiten.

1.3 Onderzoek

Het doel van dit onderzoek is om van een stuk software te kunnen bepalen hoe het gesteld is met de CWE/SANS top 25[8]. Het onderzoek zal zich richten op de programmeertaal Java. De programmeerfouten uit de top 25 zullen vanuit een Java-perspectief benaderd worden. De uitkomsten

kunnen voor meerdere soorten Java-toepassingen gebruikt worden. Maar omdat er toch verschillen zijn tussen de mogelijke toepassingen, zal dit onderzoek zich richten op webapplicaties.

In eerste instantie zal gekeken worden in hoeverre op dit moment de punten uit de top 25 zijn aan te tonen. Er zal aangeven worden met wat voor soort tools dit kan, of welke andere methodes hiervoor zijn. Uiteindelijk is ervoor gekozen om Cross-site scripting uit te sluiten (Hoofdstuk 2). Voor Cross-site scripting zal een methode onderzocht worden om deze fouten aantoonbaar uit te kunnen sluiten. Als dit deel is afgerond zullen de resultaten getoetst worden op een praktijkvoorbeeld in hoofdstuk 5.

De onderzoeksvraag is: *"In hoeverre is het mogelijk Cross-site scripting uit te sluiten binnen een webapplicatie?"*. Er zal een methodiek bedacht worden waarmee Cross-site scripting kan worden uitgesloten in een webapplicatie.

In hoofdstuk 2 zal de CWE/SANS top 25 geanalyseerd worden. Hier zal per fout gekeken worden hoe goed het te voorkomen is en of de fout statisch of dynamisch analyseerbaar is. Cross-site scripting is één van de meest gemaakte fouten en hier zal het onderzoek zich verder op richten. Vanaf hoofdstuk 3 wordt er dieper ingegaan op Cross-site scripting en hoe deze fouten ontstaan. Ook zullen de verschillende types Cross-site scripting worden behandeld en wat er gedaan kan worden om deze te voorkomen. Om Cross-site scripting te kunnen voorkomen moet de context begrepen worden. Uitleg over webapplicaties en Velocity is gegeven in hoofdstuk 4. Uiteindelijk is er een Anti-XSS tool ontwikkeld voor Velocity. De werking van de Anti-XSS tool en de resultaten op een praktijkvoorbeeld staan in hoofdstuk 5. Uiteindelijk wordt er afgesloten met de conclusie van het onderzoek in hoofdstuk 6.

Hoofdstuk 2

Analyse van programmeerfouten in Java webapplicaties

De meeste zwakheden in software worden veroorzaakt door een klein aantal programmeerfouten. Door het identificeren van onveilige programmeergewoonten en het ontwikkelen van veilige alternatieven kunnen deze fouten worden gereduceerd of geëlimineerd tijdens de ontwikkelfase. De CWE/SANS heeft een top 25[8] opgesteld van meest significante programmeerfouten. Dit zijn fouten die regelmatig voorkomen, meestal makkelijk te vinden zijn en makkelijk te misbruiken zijn.

In dit hoofdstuk wordt er voor elk fout uit de top 25 een beknopte toelichting gegeven over hoe de kwaliteit hiervan gewaarborgd kan worden. Uiteindelijk wordt er een fout uitgekozen waar nader onderzoek naar gedaan kan worden.

2.1 Tools om programmeerfouten te voorkomen

In tabel 2.1 staat een aantal tools welke een bijdrage kunnen leveren aan kwaliteitswaarborging van webapplicaties. Deze zijn allen open source en de broncode is beschikbaar. De verschillende tools en technieken hebben verschillende voor- en nadelen welke uitgebreid zijn uiteengezet door [12].

Tool	**Soort analyse**	**Werking**
FindBugs	statische code analyse	analyseert Java byte-code op bug patronen
PMD	statische code analyse	analyseert Java sourcecode op potentiële bugs
WebScarab	dynamische analyse	framework dat applicaties analyseert die gebruik maken van HTTP(S) bevat fuzzer functionaliteit

Tabel 2.1: Tools

2.1.1 FindBugs

FindBugs [6] scant byte-code van Java op bug-patronen. Het is mogelijk om zelf patronen te schrijven waar FindBugs op kan checken. Deze patronen worden detectors genoemd. Er is wel kennis van Java byte-code nodig om zelf detectors te schrijven. Hierdoor kan FindBugs in het begin een stijle leercurve hebben.

2.1.2 PMD

PMD[2] scant sourcecode van Java. Het is mogelijk om zelf regels te schrijven waarop gecheckt kan worden. Deze regels worden 'rules' genoemd. Het schrijven van eigen regels voor PMD is een stuk gemakkelijker dan voor FindBugs.

2.1.3 WebScarab

WebScarab [18] is een tool dat de werking van HTTP(S)-gebaseerde applicaties analyseert. Het is in principe een framework en de verschillende soorten functionaliteit zijn plug-ins. WebScarab is makkelijk uit te breiden door zelf plug-ins te schrijven.

2.2 Analyse van de CWE/SANS top 25

In dit hoofdstuk zal elk punt uit de top 25[8] uitgewerkt worden. Eerst zal er kort besproken worden wat het punt precies inhoudt. Vervolgens zal er beschreven worden wat er nodig is om het punt te waarborgen.

Hoe goed is het te voorkomen beschrijft in welke mate een fout is te voorkomen. Wanneer een fout gemakkelijk te voorkomen is krijgt het een '+'. Dit houdt in dat er bestaande methodes zijn om dergelijke fouten te kunnen voorkomen. Aantonen dat deze fout niet voorkomt staat hier buiten. Bij '-/+' is het vrij makkelijk te voorkomen, maar dient er aandacht aan besteed te worden dat deze fout daadwerkelijk voorkomen wordt. Bijvoorbeeld code standaarden en/of het gebruik van een standaard framework om deze fout te voorkomen. Wanneer er een '-' wordt toegekend is het bijna onmogelijk deze fout met zekerheid te kunnen voorkomen.

Statische Analyse beschrijft in hoeverre een fout kan worden gevonden middels statische analyse tools. Bij een '+' zijn er tools beschikbaar die ondersteuning bieden deze fout te detecteren. Een '-/+' wordt toegekend wanneer het mogelijk is om dergelijke fouten statisch te kunnen vinden, maar hier is nog geen standaard oplossing voor, of tool dit hiervoor ondersteuning biedt. Wanneer het praktisch onmogelijk is deze fout middels statische analyse uit te sluiten wordt er een '-' toegekend.

Dynamische Analyse Wanneer er standaardoplossingen beschikbaar zijn wordt er een '+' toegekend. Als het mogelijk is om een dergelijke fout te vinden middels dynamische analyse, maar er

is standaard geen ondersteuning voor, wordt er een '-/+' toegekend. Een '-' wordt toegekend als deze fout onmogelijk dynamisch getest kan worden.

2.2.1 CWE-602 Client-Side Enforcement of Server-Side Security

Client-side handhaven van server-side security. Clients vertrouwen om security checks te doen in naam van je server. Aanvallers kunnen de client reverse-engineren en hun eigen aangepaste client bouwen waarbij de security checks zijn weggelaten. De consequenties zijn afhankelijk van hetgeen de security checks zouden moeten beschermen. De meest voorkomende doelen zijn authenticatie, authorisatie en invoervalidatie. Wanneer veiligheid is geïmplementeerd in de servers, moet er niet alleen op de clients worden vertrouwd om dat te handhaven.

Te voorkomen: +
Dit is in principe gemakkelijk te voorkomen. Elke vorm van controle dient op de server uitgevoerd te worden. Er kunnen wel controles worden gedaan door de client om verkeer tussen client en server te beperken. Maar elke check die aan de clientkant gebeurt dient ten minste ook aan de serverkant plaats te vinden.

Statisch analyseerbaar: -
Voor een ontwikkelaar is het gemakkelijk te voorkomen. Echter is het lastig automatisch vast te leggen van welke veiligheidseisen de server afhankelijk is, en of deze dan gewaarborgd worden in servercode.

Dynamisch analyseerbaar: +
Er kunnen dymanische test cases opgesteld worden die elke vorm van beveiliging aan de clientkant probeert te omzeilen door direct te communiceren met de server.

2.2.2 CWE-250 Execution with Unnecessary Privileges

Uitvoeren met onnodige privileges. Software kan extra privileges nodig hebben om bepaalde operaties te doen, maar deze privileges langer laten dragen dan nodig kan erg veel risico's met zich meebrengen. Want wanneer software met extra privileges draait, zal het toegang hebben tot bronnen waar de gebruiker niet direct bij kan.

Te voorkomen: +
Er zijn verschillende technieken voor handen om dit soort zwakheden te beperken. Om de omgeving te beschermen kan er gebruik gemaakt worden van virtualisatie, sandboxes of jails.

nr	ID	Te voorkomen	Statisch analyseerbaar	Dynamisch analyseerbaar
Onveilige interactie tussen componenten				
1	CWE-20	-/+	-/+	-/+
2	CWE-116	-/+	-/+	-/+
3	CWE-89	-/+	-/+	-/+
4	CWE-79	-/+	-	-/+
5	CWE-78	-/+	-	-/+
6	CWE-319	++	-	-/+
7	CWE-352	-/+	-	-/+
8	CWE-362	-	-	- -
9	CWE-209	-/+	-	-
Riskant resource management				
10	CWE-119	-/+	+	+
11	CWE-642	++	-/+	-
12	CWE-73	+	-/+	-
13	CWE-426	++	-/+	-/+
14	CWE-94	+	-	-
15	CWE-494	+	-	-
16	CWE-404	-/+	+	+
17	CWE-665	+	+	-/+
18	CWE-682	+	-/+	++
Lekke toegangscontrole				
19	CWE-285	++	-	+
20	CWE-327	++	-	-
21	CWE-259	++	-/+	-
22	CWE-732	-/+	-	+
23	CWE-330	+	+	-
24	CWE-250	+	-	-
25	CWE-602	+	-	+

Tabel 2.2: CWE Top25

Statisch analyseerbaar: –
Het is erg lastig om dit soort zwakheden statisch uit te sluiten. Het gaat hier vooral om hoe het proces wordt uitgevoerd en in welke omgeving. Met bijvoorbeeld JAAS kunnen in code wel privileges verdeeld worden. Dit staat echter los van de privileges die het proces binnen zijn omgeving heeft.

Dynamisch analyseerbaar: -
Er zijn vulnerability scanners die dit soort zwakheden checken.

2.2.3 CWE-330 Use of Insufficiently Random Values

Gebruik van waarden die niet willekeurig genoeg zijn. Stel je voor hoe snel een casino failliet zou gaan wanneer gokkers kunnen voorspellen welk nummertje er wordt gedraaid, of welke kaart er getrokken wordt. Bepaalde veiligheidsmaatregelen zijn afhankelijk van sterke willekeurigheid. Als deze willekeurigheid niet wordt verschaft, is de veiligheid van deze maatregelen een stuk lager. Als de aanvaller weet welk algoritme wordt gebruikt, kan deze het volgende willekeurige getal veel makkelijker raden. Wanneer dit vaak genoeg wordt geprobeerd zal er een succesvolle aanval plaats kunnen vinden.

Te voorkomen: +
Er zijn verschillende hardwarematige en softwarematige oplossingen om willekeurige waarden te berekenen. Zorg ervoor dat voor securitydoeleinden alleen cryptografisch veilige generatoren worden gebruikt. De OWASP ESAPI heeft functionaliteit om veilig willekeurige waarden te bepalen.

Statisch analyseerbaar: -/+
Er kan bijvoorbeeld gecontroleerd worden of willekeurige nummers veilig genoeg gegenereerd worden, dus of er functions voor gebruikt worden die voor cryptografische doeleinden zijn bedoeld.

Dynamisch analyseerbaar: -/+
Om entropieproblemen op te sporen kunnen FIPS-2-testen uitgevoerd worden. FIPS staat voor Federal Information Processing Standard, en FIPS-2 is een standaard dat wordt gebruikt om cryptografische modules te accrediteren.

2.2.4 CWE-732 Incorrect Permission Assignment for Critical Resource

Incorrecte toewijzing van toestemming voor kritische bronnen. Het is onbeleefd om iets mee te nemen zonder eerst toestemming te vragen. Maar onbeleefde gebruikers zijn bereid om te kijken hoe ver ze kunnen gaan. Wanneer je kritische programma's, gegevensvoorraden of configuratiebestanden hebt met permissie die jouw bronnen leesbaar maken

---

Te voorkomen: -/+
Dit soort zwakheden zijn niet lastig te voorkomen. Er zijn verschillende technische hulpmiddelen en frameworks beschikbaar die hiervoor kunnen zorgen. Echter is het wel erg lastig uit te sluiten dat een dergelijke fout in de software zit.

Statisch analyseerbaar: -
Wanneer er kritische bronnen worden gebruikt zoals bijvoorbeeld een configuratiebestand, kan er gecontroleerd worden of dit bestand geen onveilige permissies heeft. Het is lastig dit probleem uit te sluiten middels statische analyse.

Dynamisch analyseerbaar: -/+
Er kan toezicht gehouden worden op hoe de software werkt met het besturingssysteem en het netwerk. Echter het hiermee uitsluiten blijft lastig.

2.2.5 CWE-259 Hard-Coded Password

Een hard-coded account en/of wachtwoord in de authenticate module van de software is erg gemakkelijk voor een ervaren reverse-engineer. Het kan de kosten voor testen en ondersteuning flink verlagen, maar dat doet het ook met de veiligheid van het product voor de klant. Als het wachtwoord hetzelfde is voor alle software, dan is elke klant kwetsbaar wanneer het wachtwoord bekend wordt. Omdat het hard-coded is, is het ook erg lastig om dit probleem dan op te lossen voor systeembeheerders.

Te voorkomen: ++
In principe is dit erg makkelijk te voorkomen door duidelijke richtlijnen voor softwareontwikkelaars hiervoor op te stellen.

Statisch analyseerbaar: -
Er zijn tools die wel pogingen doen om hard-coded wachtwoorden te vinden. Echter zou dit probleem niet middels statische analyse uitgesloten kunnen worden.

Dynamisch analyseerbaar: -

Met behulp van monitoringtools kan handmatig gezocht worden naar wachtwoorden. Echter een methode om dit probleem uit te sluiten middels dynamische analyse is er niet.

2.2.6 CWE-327 Use of a Broken or Risky Cryptographic Algorithm

Gebruik van onveilig geachte of riskante cryptografische algoritmes. Cryptografie kun je gebruiken om te voorkomen dat aanvallers gevoelige gegevens kunnen lezen of manipuleren. Het is echter noodzakelijk geen verouderde algoritmen hiervoor te gebruiken. Dit zijn algoritmen waarvan vroeger gedacht werd dat er miljoenen jaren rekenkracht nodig waren om ze te kraken, maar die tegenwoordig binnen een paar dagen of uren gekraakt kunnen worden. Tevens zegt dit punt dat je nooit je eigen cryptografische algoritmen moet schrijven.

Dit risico kun je ten eerste voorkomen door nooit zelf cryptografische algoritmen te schrijven. Wanneer er een cryptografisch algoritme wordt gebruikt, dient een tool te zijn hebben die kan aantonen dat alleen 'veilige' algoritmen worden gerbuikt. Veilige algoritmen zijn op dit moment niet te kraken binnen aanzienbare tijd. Op dit moment is er niet een tool die out of the box hierop checkt. Wel kunnen bestaande tools uitgebreid worden zodat deze hierop checken.

Te voorkomen: ++
Dit probleem is erg makkelijk te voorkomen door duidelijke richtlijnen voor softwareontwikkelaars hiervoor op te stellen.

Statisch analyseerbaar: -
Er kan gecontroleerd worden op veelgebruikte bibliotheken die verouderd zijn geworden. Echter is het lastig vast te stellen wanneer een bibliotheek verouderd is. Verder blijft het probleem dat hiermee dit probleem niet aantoonbaar uitgesloten kan worden.

Dynamisch analyseerbaar: -
Dit probleem is niet aantoonbaar uit te sluiten middels dynamische analyse.

2.2.7 CWE-285 Improper Access Control (Authorization)

Onjuiste toegangscontrole (authorisatie). Stel je organiseert een feestje voor een paar goede vrienden en hun introducées. Je nodigt iedereen uit in je woonkamer, maar terwijl je bijpraat met één van je vrienden, is één van de introducées je koelkast aan het plunderen, kijkt in je medicijnkastje en bedenkt wat er zoal in je nachtkastje zou zitten. Software heeft dezelfde authorisatieproblemen die nog ergere gevolgen kunnen hebben. Als je niet verzekert dat gebruikers alleen maar kunnen doen wat ze mogen doen, zullen aanvallers de onjuiste authorisatie misbruiken en functionaliteit

gebruiken die alleen was bedoeld voor bepaalde gebruikers.

Te voorkomen: ++
Door duidelijke richtlijnen op te stellen en te inventariseren wie waartoe toegang heeft kunnen dit soort problemen voorkomen worden.

Statisch analyseerbaar: -
Het is erg lastig dit soort problemen uit te sluiten middels statische analyse.

Dynamisch analyseerbaar: +
Middels dynamische analyse zou er ten minste gecheckt kunnen worden of een gebruiker meer kan dan in principe mogelijk zou moeten zijn. Echter moet hiervoor wel van te voren duidelijk worden vastgesteld wie waartoe toegang heeft.

2.2.8 CWE-682 Incorrect Calculation

Incorrecte berekening. Computers kunnen berekeningen uitvoeren waarvan het resultaat wiskundig gezien nergens op slaat. Bijvoorbeeld, wanneer twee grote integers worden vermenigvuldigd kan het resultaat kleiner zijn vanwege een integer overflow. Of berekening kunnen onmogelijk uit te voeren zijn, zoals delen door nul. Wanneer aanvallers controle hebben over de invoer die wordt gebruikt voor numerieke berekeningen, kunnen de genoemde zwakheden veiligheidsconsequenties hebben. Bijvoorbeeld een negatieve prijs voor een product. Of een crash omdat het programma probeert te delen door nul.

Te voorkomen: +
Met consequent strenge invoervalidatie kan dit probleem voor een groot deel voorkomen worden.

Statisch analyseerbaar: -/+
Er kan gecheckt worden of elke invoer van een berekening op een bepaalde manier gevalideerd is. Zo zou een analyse uit kunnen wijzen of een integer eventueel te groot kan zijn. 100% uitsluiten is erg lastig.

Dynamisch analyseerbaar: ++
Met behulp van fuzzers zouden allerlei mogelijke berekeningen uitgevoerd kunnen worden. Het moet dan van belang zijn dat de software blijft draaien en altijd het juiste resultaat geeft.

2.2.9 CWE-665 Improper Initialization

Onjuiste initialisatie. Een goede initialisatie zorgt ervoor dat de software draait zonder dat het uitvalt tijdens een belangrijke operatie. Als je gegevens en variabelen niet op de juiste manier initialiseert, kan het mogelijk zijn dat een aanvaller deze initialisatie voor je doet. Wanneer zulke variabelen worden gebruikt in veiligheidskritieke operaties, zoals authenticatie, dan zouden deze aangepast kunnen worden om bepaalde veiligheidsmaatregelen te omzeilen.

Te voorkomen: +
Dit probleem kan deels weggenomen worden door een programmeertaal te gebruiken die de kans op dit soort fouten verkleint. In Java zal er een compile-time error optreden wanneer een variabele niet expliciet is geïnitialiseerd.

Statisch analyseerbaar: +
Middels statische analyse kan er gecontroleerd worden of variabelen op de juiste manier geïnitialiseerd worden.

Dynamisch analyseerbaar: -/+
Dit soort fouten kunnen voorkomen in code-paden die niet goed getest zijn, zoals zeldzame foutcondities. Middels dynamische analyse kunnen ten minste alle mogelijke paden gedekt worden middels testen.

2.2.10 CWE-404 Improper Resource Shutdown or Release

Onjuist afsluiten of vrijlaten van bronnen. Wanneer bronnen niet meer nodig zijn, moeten ze correct worden weggegooid. Anders kan de omgeving vol raken of zelfs besmet. Dit is bijvoorbeeld van toepassing op geheugen, bestanden, cookies, en datastructuren. Een aanvaller kan voordeel halen uit onjuist afgesloten bronnen. Zo kan de aanvaller controle krijgen over bronnen waarvan je dacht dat ze afgesloten waren. Dit kan tot hogere bronconsumptie leiden omdat dan geen bronnen meer worden vrijgelaten aan het systeem. Dus wanneer de vuilnis niet wordt gecontroleerd voordat het wordt weggegooid, kunnen aanvallers het doorzoeken op waardevolle informatie.

Te voorkomen: -/+
Er zijn programmeertalen die de kans op dit soort fouten aanzienlijk verkleinen. Bijvoorbeeld Java dat garbage collection uitvoert om niet meer gebruikte (gede-alloceerde) objecten te verwijderen.

Statisch analyseerbaar: +
Er kan gecheckt worden of objecten na gebruik weer verwijderd of netjes gesloten worden. Er zijn tools beschikbaar die deze functionaliteit bevatten.

Dynamisch analyseerbaar: +
Met fuzzers en stresstests kan de software aangeroepen worden met een groot aantal threads of processen. Vervolgens kan van onverwacht gedrag gedetecteerd worden. Tijdens dergelijke tests mag de software wat trager reageren, maar niet instabiel worden, crashen of verkeerde resultaten leveren.

2.2.11 CWE-494 Download of Code Without Integrity Check

Het downloaden van code zonder een integriteitscontrole. Wanneer code wordt gedownload en uitgevoerd, vertrouw je erop dat de bron niet kwaadaardig is. Ook al vertrouw je de bron, aanvallers kunnen verschillende trucs uithalen om de code aan te passen voordat het de bestemming bereikt. De bron hacken, DNS spoofen of DNS-cache vergiftigen of misschien zelfs de code aanpassen terwijl het onderweg is door het netwerk.

Te voorkomen: +
Door code altijd over een versleutelde verbinding te sturen en cryptografische handtekeningen te gebruiken om de integriteit van de code te controleren zorg je ervoor dat de triviale zwakheden zijn uitgesloten. Verder zijn er standaard technieken die gebruikt kunnen worden om de code te ondertekenen zoals Authenticode.

Statisch analyseerbaar: –
In principe is dit erg lastig te analyseren omdat de zwakke plek zit in het proces van het overzetten van de code. Echter kan er wel voor gezorgd worden dat bijvoorbeeld code die is gedownload wordt uitgevoerd met speciale rechten, zodat het geen schade kan brengen aan de software waarbinnen het draait.

Dynamisch analyseerbaar: -
Dit is ook lastig omdat de zwakke plek in de code van de software zelf zit. Echter kan hiermee wel getest worden in hoeverre code dat is gedownload schade aan kan richten binnen een stuk software.

2.2.12 CWE-94 Failure to Control Generation of Code (aka 'Code Injection')

Code-generatie niet kunnen bedwingen. Om softwareontwikkeling te vereenvoudigen kan er code gegenereerd worden. Er kan bijvoorbeeld functionaliteit gegenereerd worden die frequent nodig is. Codegeneratie is erg aantrekkelijk voor aanvallers. Het vormt een directe dreiging wanneer uw code direct is aan te roepen door een derde partij. Bijvoorbeeld wanneer de externe invoer invloed

heeft op welke code wordt uitgevoerd, of erger wanneer deze invoer direct doorgegeven wordt aan de code. Dit impliceert volledige controle van de aanvaller over uw code.

Te voorkomen: +
Belangrijk om dit te voorkomen is bewustzijn bij de ontwikkelaar.

Statisch analyseerbaar: -
Dit komt vooral voor bij geïnterpreteerde talen. Het is lastig te analyseren of dynamische gegenereerde code voorkomt. Wel kan er gecontroleerd worden op bepaalde aanroepen die codegeneratie mogelijk maken.

Dynamisch analyseerbaar: -
Fuzzers en tools om gevaarlijke code te injecteren kunnen gebruikt worden om dit dynamisch te analyseren.

2.2.13 CWE-426 Untrusted Search Path

Onbetrouwbaar zoekpad. Software maakt vaak gebruik van bronnen, zoals code bibliotheken of configuratiebestanden. Deze bronnen kunnen gevonden worden door middel van het "zoekpad". De software is echter afhankelijk van de ontwikkelaar of zijn omgeving voor het juiste zoekpad. De ontwikkelaar; in de code staat expliciet waar de resource te vinden is (bijvoorbeeld het volledige pad). De omgeving; in de code staat een relatief pad naar de resource. Met behulp van omgevingsvariabelen kan de resource gevonden worden. Wanneer het zoekpad onder beheer van een aanvaller valt, kan de aanvaller het zoekpad aanpassen zodat het verwijst naar bronnen die hij kiest. Dit heeft als gevolg dat de software verkeerde bronnen gebruikt. Dit risico is ook aanwezig wanneer een element uit het zoekpad onder beheer is van de aanvaller. Dus dat de aanvaller bijvoorbeeld schrijfrechten heeft op een map in het zoekpad of op de huidige werkmap.

Dit risico kun je voorkomen door het zoekpad hard te zetten op een verzameling paden waarvan bekend is dat ze veilig zijn. Hier mogen dus geen paden in staan waar de software of andere gebruikers schrijfrechten op hebben. Verder moet het zoekpad niet te wijzigen zijn door derden. Als er een ander programma wordt aangeroepen, dan zal het volledige pad naar dat programma gegeven moeten worden.

Om aan te tonen dat deze fout niet voor komt, heb je iets nodig dat ervoor zorgt dat er in de code alleen volledige paden worden gebruikt. Dit mogen ook alleen paden zijn naar betrouwbare directories. Indien er toch relatieve paden worden gebruikt, moet er aangetoond zijn dat het zoekpad te vertrouwen is.

Te voorkomen: ++
Door ten alle tijden volledige paden te gebruiken voorkom je het probleem dat het zoekpad beïnvloed

kan worden.

Statisch analyseerbaar: -/+
Door middel van statische analyse kan aangetoond worden dat alleen functies gebruikt worden die volledige paden vereisen.

Dynamisch analyseerbaar: -/+
Fuzzers kunnen gebruikt worden om dit dynamisch te analyseren.

2.2.14 CWE-73 External Control of File Name or Path

Extern beheer van het pad of de naam van een bestand. Gegevens worden vaak uitgewisseld via bestanden. Meestal is het de bedoeling niet alle bestanden op het systeem beschikbaar te stellen. Daarom wordt er vaak een map met beperkte rechten gebruikt. Wanneer invoer van een buitenstaander wordt gebruikt om een pad naar het bestand op te stellen, zou het resultaat kunnen verwijzen naar een plek buiten de beperkte map. Een aanvaller kan bijvoorbeeld één of meerdere keren ".." gebruiken om het besturingssysteem uit de beperkte map te laten navigeren.

Te voorkomen: +
Door nooit direct gebruikersinvoer te gebruiken om een pad op te stellen kan dit voorkomen worden. Als de gebruikersinvoer wel invloed moet hebben op welk bestand bijvoorbeeld geopend moet worden kunnen white lists gebruikt worden.

Statisch analyseerbaar: -/+
Door middel van statische analyse zou aangetoond kunnen worden dat gebruikersinvoer nooit direct wordt gebruikt in paden of bestandsnamen.

Dynamisch analyseerbaar: -
Fuzzers kunnen gebruikt worden om dit dynamisch te analyseren.

2.2.15 CWE-642 External Control of Critical State Data

Extern beheer van kritische informatie over de staat. Er zijn verschillende manieren om informatie over de staat van gebruikers op te slaan. Bijvoorbeeld in configuratiebestanden, cookies, profielen, verborgen formuliervelden of registersleutels. Dit zijn allemaal plaatsen welke aangepast zouden kunnen worden door de gebruiker. Wanneer gegevens uit dit soort plaatsen wordt gebruikt ter controle van rechten en toegang kan er een lek zijn.

Er zijn frameworks die het beheer van de staat op een juiste manier kunnen handhaven zodat deze zwakheid makkelijker voorkomen wordt. De ESAPI[9] heeft hiervoor bijvoorbeeld de Session Management feature.

Te voorkomen: ++
Technisch kan deze zwakheid makkelijk voorkomen worden. Vooral bewustwording van ontwikkelaars is belangrijk. Het is belangrijk om te weten welke potentiële locaties toegankelijk zijn voor aanvallers.

Statisch analyseerbaar: -/+
Er kan aangetoond worden of er informatie extern wordt bijgehouden. Echter bepalen of deze informatie kritische gegevens over de staat bevat is lastig. Deze zwakheid is niet met alleen statische analyse uit te sluiten.

Dynamisch analyseerbaar: -
Dynamische tools met grote testsuites met verschillende invoer en fuzzers kunnen gebruikt worden. Deze zouden veelvoorkomende fouten moeten kunnen opmerken.

2.2.16 CWE-119 Failure to Constrain Operations within the Bounds of a Memory Buffer (Buffer Overflow)

Buffer overflows komen voor wanneer wordt getracht data in een stukje geheugen te stoppen dat groter is dan het stukje gereserveerde geheugen. Of zoals een natuurkundige wet zegt: “Wanneer je meer dingen in een container stopt dan het kan bevatten, maak je een zooitje”. Programmeertalen die eigen geheugenbeheer toepassen zijn in principe niet gevoelig voor Buffer overflows. Maar bijvoorbeeld een interface naar native code kan wel mogelijke buffer overflows opleveren.

Wanneer er buiten de grens van de buffer wordt gelezen, kan de aanvaller toegang krijgen tot gevoelige informatie. Daarnaast kan de aanvaller een buffer overflow lek misbruiken om een functie pointer te verwijzen naar zijn eigen kwaadaardige code.

Te voorkomen: -/+
Dit probleem komt in bepaalde programmeertalen voor. Dit probleem is in het algemeen gemakkelijk te voorkomen wanneer een taal wordt gebruikt waarbij de ontwikkelaar niet verantwoordelijk is voor het geheugenbeheer. Maar dat de taal, of de omgeving waarin de toepassing wordt uitgevoerd, verantwoordelijk is voor het geheugenbeheer.

Statisch analyseerbaar: +
Mogelijke buffer overflows kunnen vaak gedecteerd worden middels statische analyse. Echter kan statische analyse geen rekening houden met invloeden uit de omgeving.

Dynamisch analyseerbaar: +
Deze zwakheid kan worden gedetecteerd middels dynamische analyse. Tools die verschillende invoer proberen zoals fuzzers zijn hier uitermate geschikt voor. De software zou gedurende zo'n test misschien trager worden, maar het zou niet verkeerde resultaten mogen teruggeven of crashen.

2.2.17 CWE-209 Error Message Information Leak

Informatielekkage in foutmeldingen. Het is belangrijk dat foutmeldingen niet kunnen leiden tot de onthulling van "geheime informatie". Anders kan een aanvaller deze informatie gebruiken om de software te misbruiken. Voorbeelden van "geheime informatie"zijn serverinstellingen, credentials voor authenticatie of het besturingssysteem waar de webapplicatie op draait.

Er moet gewaarborgd worden dat foutmeldingen alleen bruikbare informatie bevat voor degene die het leest, en verder voor niemand. Verder moeten foutmeldingen geen informatie geven over de methodes die zijn gebruikt om de fout te achterhalen. Zulke gedetailleerde informatie kan een aanvaller helpen om een andere aanval te bedenken die de validatie passeert.

Te voorkomen: -/+
In principe is deze fout gemakkelijk te voorkomen. Het gaat hier niet om de technische moeilijkheid, maar het bewustzijn van de ontwikkelaar.

Statisch analyseerbaar: -
Veel voorkomende patronen zoals bijvoorbeeld stack traces of paden naar bestanden kunnen automatisch gevonden worden. Maar het schenden van business rules of privacy is niet zomaar mogelijk.

Dynamisch analyseerbaar: -
Hiervoor geldt hetzelfde als voor statische analyse.

2.2.18 CWE-362 Race Condition

Race condities komen voor wanneer er geprobeerd wordt om twee of meer operaties op hetzelfde moment uit te voeren. Om operaties correct uit te voeren, moeten de instructies in de juiste volgorde gedaan worden. Wanneer er twee of meer operaties tegelijk uitgevoerd worden kan er een conditie ontstaan waarbij de normale functionaliteit verstoord wordt. Een aanvaller zou dit kunnen gebruiken om waardevolle informatie te krijgen of om oneigenlijke acties uit te voeren.

Om race condities te waarborgen is er iets nodig dat garandeert dat instructies in de correcte

volgorde uitgevoerd worden. Binnen Java zijn er mogelijkheden om race condities uit te sluiten. Deze mogelijkheden hebben echter een nadelige impact op de performance. Meestal is performance juist van belang in systemen waar race condities voor kunnen komen. Daarom worden deze mogelijkheden om race condities uit te sluiten niet altijd toegepast.

Te voorkomen: -/+
Mogelijke race condities zijn lastig volledig uit te sluiten. Het is belangrijk dat er wordt bepaald op welke plek er mogelijkh een race conditie voorkomt. Deze code moet dan gebruik maken van thread safe-functies. Verder dient het gebruik van gedeelde bronnen geminimaliseerd worden om het risico te verlagen.

Statisch analyseerbaar: -
Alleen veel voorkomende patronen van race condities zijn te detecteren met behulp van statische analyse.

Dynamisch analyseerbaar: -
Dynamisch kan er worden laten zien dat er race condities op kunnen treden. Echter is het onmogelijk om het hiermee uit te sluiten. Overigens zijn race condities met een erg nauw tijdvenster erg lastig te detecteren.

2.2.19 CWE-352 Cross-Site Request Forgery

Cross-Site Request Forgery, afgekort CSRF, maakt gebruik van het vertrouwen dat een webapplicatie heeft in de browser van de gebruiker. Een CSRF-aanval zorgt ervoor dat de browser van een slachtoffer een request stuurt naar een kwetsbare webapplicatie. Deze webapplicatie handelt dan de request af alsof deze van het slachtoffer afkomt. Wanneer het slachtoffer is geauthenticeerd bij de webapplicatie, krijgt de aanvaller toegang als deze gebruiker.

Om CSRF te voorkomen moet gegarandeerd zijn dat elke request met de juiste bedoeling wordt gedaan. Dit kan afgedwongen worden door 'challenge tokens' te gebruiken. Deze 'challenge tokens' zijn willekeurige waarden die in alle formulieren en gevoelige URL's wordt ingevoegd. Gevoelige URL's zijn links die verbonden zijn aan gevoelige operaties op de server. De server kan dan bij gevoelige requests verifiëren of de token aanwezig en correct is.

Om CSRF te voorkomen is OWASP bezig met CSRFGuard [10]. Deze is beschikbaar voor Java en .NET. Voor Java is het een JEE-filter dat onder andere challenge tokens implementeert. Verder is OWASP ook bezig met CSRFTester[11], een tool dat checkt op mogelijke CSRF-lekken. Deze tool genereert op HTML gebaseerde CSRF-aanvallen. Beide tools kunnen het risico op CSRF sterk verlagen. Echter blijft het lastig om aan te tonen dat CSRF helemaal is uitgesloten van een webapplicatie.

Te voorkomen: -/+
Met een goede voorlichting en standaard aanpak is dit probleem best te voorkomen.

Statisch analyseerbaar: -
CSRF is op het moment erg lastig te detecteren met automatiche technieken. Elke webapplicatie heeft een eigen specifieke securityinstellingen met betrekking op requests. Het hangt dus van de instellingen af hoe er met een request wordt omgegaan. Er zijn requests die beïnvloedbaar zijn door een buitenstaander (denk aan een google search query). Aan de andere kant zijn er de vertrouwelijke requests die in naam van een gebruiker worden gedaan, welke onder geen enkele voorwaarde beïnvloedbaar mogen zijn door derden.

Dynamisch analyseerbaar: -/+
Er zijn tools die helpen in het vinden van CSRF-lekken. Echter sluiten deze tools het niet uit.

2.2.20 CWE-319 Cleartext Transmission of Sensitive Information

Gevoelige informatie versturen in klare tekst. Wanneer gegevens over een netwerkt worden verstuurd passeert het verschillende doorvoerpunten (nodes) op zijn pad naar bestemming. Aanvallers kunnen zonder erg veel moeite zich toegang verschaffen tot deze gegevens. Het enige dat een aanvaller hoeft te doen is, één van de nodes onder zijn beheer hebben, of een node in hetzelfde netwerk als het doorvoerpunt in beheer hebben. Wanneer een applicatie gevoelige informatie verstuurt over het netwerk, zoals persoonsgegevens of authenticatiecredentials, zal dit door een aanvaller gelezen kunnen worden. Het is dus van belang dat deze gegevens versleuteld verstuurd worden.

Dit is gewaarborgd wanneer elk stukje gevoelige informatie versleuteld verstuurd wordt. De meest veilige optie is om simpelweg alle communicatie over een beveiligde verbinding te versturen. Zodoende is dit hele punt dan ook gewaarborgd.

SSL is een standaard protocool om versleutelde verbindingen op te zetten. Verder zijn er tools waarmee alle communicatie afgeluisterd kan worden. Hiermee kan het gecontroleerd worden of er informatie onversleuteld verstuurd wordt.

Te voorkomen: ++
Het versturen van gevoelige informatie in klare tekst is gemakkelijk te voorkomen. Door alle communicatie via een versleutelde verbinding te laten plaatsvinden is dit punt gewaarborgd.

Statisch analyseerbaar: -
Wanneer informatie ook onversleuteld verstuurd wordt is het erg lastig uit te sluiten of alle gevoelige informatie wel versleuteld wordt. Er moet dan bepaald worden welke informatie gevoelig is. Vervolgens moet ook aangetoond worden dat deze informatie alleen via een versleutelde verbinding

verstuurd wordt.

Dynamisch analyseerbaar: -/+
Er zijn tools die verkeer afluisteren. Deze kunnen gebruikt worden om te checken of er onversleutelde informatie wordt verstuurd. Echter bepalen of deze informatie ook gevoelig is, is moeilijk te waarborgen.

2.2.21 CWE-78 Failure to Preserve OS Command Structure (aka 'OS Command Injection')

Commando-injectie. Wanneer er een aanroep naar een ander programma op het besturingssysteem wordt gedaan moet er rekening gehouden worden met de argumenten die aan het commando worden meegegeven. Net als bij SQL-injectie geldt hier dat wanneer de aanvaller het commando of de argumenten kan beïnvloeden, hij nare dingen kan uithalen. Dit risico kan voorkomen worden door white lists toe te passen waarin de geldige commando's en/of tekens staan.

Er is iets nodig dat aantoont dat invoer van buitenaf geen invloed kan hebben op commando's die worden aangeroepen of de argumenten die hieraan worden meegegeven. Zoals eerder genoemd kan dit risico voorkomen worden door white lists toe te passen. Als dit punt gewaarborgd dient te worden, zal je moeten aantonen dat deze white lists veilig zijn, en overal worden toegepast.

Te voorkomen: -/+
Commando-injectie is in principe makkelijk te voorkomen. Je kunt van tevoren bedenken wat er fout kan gaan, en er zijn standaard oplossingen om dit te voorkomen.

Statisch analyseerbaar: -
Het is ansich niet makkelijk statisch te analyseren. Maar er zou bijvoorbeeld wel gecheckt kunnen worden of white lists correct worden toegepast. Daarnaast zou dan wel aangetoond moeten worden dat het correct toepassen van white lists commando-injectie uitsluit.

Dynamisch analyseerbaar: -/+
Er zijn tools die pogen verschillende commando's te injecteren. Hier kunnen gaten mee gevonden worden. Echter zal dit probleem niet met dergelijke tools uitgesloten kunnen worden.

2.2.22 CWE-79 Failure to Preserve Web Page Structure (aka 'Cross-site scripting')

Cross-site scripting (XSS) is het injecteren van code in een webapplicatie die door andere gebruikers wordt bekeken. De code die wordt geïnjecteerd is meestal HTML of JavaScript.

Cross-site scripting kan voorkomen worden door te begrijpen in welke context de uitvoer gebruikt wordt, en deze uitvoer dan op de passende manier te encoderen. De encodering kan namelijk verschillen afhankelijk of het deel uitmaakt van de HTML-body, JavaScript, CSS, etc.

FindBugs checkt alleen op overduidelijke Cross-site scripting-lekken. Dit is lang niet genoeg om te kunnen waarborgen dat Cross-site scripting niet voorkomt. Verder is er de ESAPI (Enterprise Security API) van OWASP welke gebruikt kan worden om alle uitvoer op de juiste manier te encoderen.

Te voorkomen: -/+
Cross-site scripting is vrij makkelijk te voorkomen. Je kunt van tevoren bedenken wat er fout kan gaan, en er zijn standaard oplossingen om dit te voorkomen.

Statisch analyseerbaar: -
Het is lastig te analyseren. Er zijn nu al wel tools die voorbeelden van Cross-site scripting-lekken kunnen vinden. Deze kunnen alleen niet alle mogelijke Cross-site scripting uit te sluiten.

Dynamisch analyseerbaar: -/+
Met tools kunnen verschillende parameters uitgeprobeerd worden en gecheckt worden op incomplete validatie. Dit zal Cross-site scripting echter niet compleet uitsluiten.

2.2.23 CWE-89 Failure to Preserve SQL Query Structure (aka 'SQL Injection')

SQL-injectie. Een webapplicatie maakt meestal gebruik van een database. Communicatie met de database gebeurt middels SQL. Bijvoorbeeld om te authenticeren voert de gebruiker zijn naam en wachtwoord in. De applicatie kijkt of deze gebruiker toegang verdient middels een SQL-query die op de database wordt uitgevoerd. De applicatie creëert een SQL-query op basis van de invoer van de gebruiker. Wanneer een aanvaller de structuur deze SQL-query kan beïnvloeden kan hij erg nare dingen uithalen.

Er is iets nodig dat voorkomt dat een aanvaller de structuur van de SQL-queries kan beïnvloeden. Dit kan gedaan worden door ervoor te zorgen dat elk stukje dynamische data dat in een query wordt gezet valide is. In het voorbeeld van authenticatie zijn de ingevoerde gebruikersnaam en wachtwoord het stukje dynamische data van de SQL-query. Met behulp van statische analyse kan aangetoond worden dat dynamische data alleen wordt meegegeven middels getypeerde parameters.

Te voorkomen: +
Om dit risico te voorkomen kun je "prepared statements" gebruiken. Hierbij is het echter nog wel van belang dat deze prepared statements correct gebruikt worden. Om dit af te dwingen moet elk stukje dynamische data als getypeerde parameter worden meegegeven. Tegenwoordig wordt er vaak een ORM-laag[1] gebruikt. Hierbij geldt hetzelfde als voor prepared statements. Dit kan helpen bescherming te bieden tegen SQL-injecties, indien het op de juiste manier toegepast wordt. Dus de stukjes dynamische data dienen als getypeerde parameter meegegeven te worden.

Statisch analyseerbaar: -/+
In combinatie met het gebruik van een framework is het makkelijk statisch te analyseren. Er kan dan gecheckt worden of het framework correct wordt toegepast. FindBugs checkt op correct gebruik van "prepared statements" en het uitvoeren van dynamisch gegenereerde strings op databases.

Dynamisch analyseerbaar: -/+
Met tools kunnen verschillende parameters uitgeprobeerd worden en gecheckt worden op incomplete validatie en SQL-injectie.

2.2.24 CWE-116 Improper Encoding or Escaping of Output

Onjuiste encodering of escaping van uitvoer. Computers hebben de eigenschap om precies te doen wat je zegt, en niet perse wat je bedoelt. Onvoldoende encoderen van uitvoer is vaak het vergeten broertje van slechte invoervalidatie. Alleen is dit de wortel van alle injectiegebaseerde aanvallen. Aanvallers kunnen commando's aanpassen die naar andere componenten worden gestuurd. Dit kan mogelijk leiden tot een complete overname van de applicatie. Dit verandert "doe wat ik bedoel" in "doe wat de aanvaller zegt". Wanneer de applicatie uitvoer genereert in de vorm van gestructureerde berichten zoals een query is het van belang dat de structuur en de daadwerkelijke gegevens gescheiden blijven. Een gebruiker mag alleen invloed hebben op de gegevens die verstuurd worden, maar niet op de structuur (commando's) van het bericht. In webapplicaties is dit makkelijk te vergeten omdat de commando's en de gegevens samengevoegd in één stroom worden verstuurd, waarbij enkele karakters de grens aangeven tussen commando's en gegevens. Zoals bij HTML of XML.

Te voorkomen: -/+
Technisch gezien is het niet lastig deze fout te voorkomen. Echter is het niet altijd duidelijk en triviaal wat de gevolgen kunnen zijn van een bepaalde manier van of niet encoderen.

[1] ORM, Object-relational mapping is een techniek om data uit te wisselen tussen relationele databases en object georiënteerde programmeertalen.

Statisch analyseerbaar: -/+
Er kan statische gecheckt worden of uitvoer geëncodeerd wordt. Echter of er correcte encodering plaatsvindt blijft lastig.

Dynamisch analyseerbaar: -/+
Er kan dynamisch getest worden of uitvoer op de juiste manier geëncodeerd wordt door allerlei mogelijke vormen uit te proberen.

2.2.25 CWE-20 Improper Input Validation

Onjuiste invoervalidatie is de grootste oorzaak van zwakheden in software. Er moet altijd gewaarborgd worden dat de invoer voldoet aan de verwachtingen. Wanneer dit niet het geval is kan dit leiden tot zwakheden waar aanvallers de invoer kunnen aanpassen in iets onverwachts wat nare gevolgden kan hebben. Veel van de huidige zwakheden kunnen worden voorkomen, of tenminste sterk gereduceerd worden, wanneer er juiste invoervalidatie plaatsvindt. 'Onjuist invoervalidatie' is de oorzaak van veel andere fouten uit de CWE top 25.

Te voorkomen: -/+
Het is geen hogere wiskunde om invoervalidatie op de juiste manier te doen. Echter is voor een ontwikkelaar niet altijd triviaal om te zien waar en hoe er gevalideerd moet worden. Er zijn genoeg hulpmiddelen beschikbaar die technisch ondersteuning bieden en validatie kunnen doen. Echter blijft het de taak van de ontwikkelaar om de juiste validatie bij de invoer te kiezen.

Statisch analyseerbaar: -/+
Er kan statische gecheckt worden of invoer gevalideerd wordt. Echter of er een correcte validatie plaatsvindt is erg lastig.

Dynamisch analyseerbaar: +
Er kunnen dynamische casussen opgesteld worden voor elke manier van invoer. Verder kun je specificeren aan welke eisen invoer moet voldoen, vervolgens kan er dan dynamisch gecheckt worden of er altijd aan deze eisen wordt voldaan.

2.3 Selectie van de te onderzoeken programmeerfout

Voor veel punten is er niet zomaar een uniforme oplossing om deze te waarborgen. Voor een groot deel van de punten zijn wel hulpmiddelen zoals frameworks of bibliotheken beschikbaar. Dit is echter niet genoeg om uit te kunnen sluiten dat zulke fouten voorkomen in de code.

Cross-site scripting is één van de meest gevaarlijke zwakheden die webapplicaties kunnen bevatten. In theorie is deze fout vrij makkelijk te voorkomen. Echter worden er in de praktijk toch de meeste fouten mee gemaakt. In de rest van dit onderzoek zal uitgebreider worden ingegaan op Cross-site scripting.

Hoofdstuk 3

Cross-site scripting

In hoofdstuk 2 is laten zien dat Cross-site scripting één van de meest gemaakte programmeerfouten is.

Een webapplicatie is gevoelig voor Cross-site scripting wanneer invoerparameters direct in de uitvoer worden gezet. Deze invoerparameters kunnen stukjes code bevatten die de browser van het slachtoffer misleiden zodat deze bepaalde informatie verstuurt of andere acties uitvoert. Dit maakt het voor een aanvaller mogelijk om het 'same-origin'-beleid [5] van browsers te omzeilen.

Het 'same-origin'-beleid houdt in dat scripts die geladen worden van een bepaald domein, geen toegang hebben tot gegevens die behoren tot een ander domein. Bijvoorbeeld een script dat geladen wordt van `www.A.com` heeft geen toegang tot gegevens die in de browser worden bijgehouden voor `www.B.com`. Cookies zijn een voorbeeld van belangrijke gevoelige gegevens die voor websites worden bijgehouden in een browser. Wanneer een aanvaller cookies in handen krijgt van een slachtoffer, kan deze meestal acties uitvoeren in naam van het slachtoffer.

In Cross-site scripting-aanvallen wordt het 'same-origin'-beleid omzeild omdat wanneer het slachtoffer een gegenereerde pagina ontvangt, de kwaadaardig geïnjecteerde code wordt uitgevoerd in de context van het domein waarop de gevoelige webapplicatie draait. Dit heeft als gevolg dat deze (kwaadaardig geïnjecteerde) code toegang heeft tot alle gegevens die in de browser worden bijgehouden onder het gevoelige domein, zoals bijvoorbeeld de cookies.

In figuur 3.1 is een stukje JavaScript-code te zien. Wanneer dit stukje in een webapplicatie geplaatst kan worden, en deze code als JavaScript wordt uitgevoerd door de browser, dan is de betreffende webapplicatie gevoelig voor Cross-site scripting-aanvallen. Dit is een simpel voorbeeld dat middels JavaScript een popup weergeeft, maar het toont effectief een lek aan. Wanneer een dergelijk lek eenmaal is gevonden zijn de Cross-site scripting-mogelijkheden eindeloos.

```
1 <script>alert('XSS mogelijk!');</script>
```

Figuur 3.1: Cross-site scripting

3.1 Cross-site scripting-aanval in de praktijk

Rond 5 april 2010 is een server van Apache gecompromitteerd [14]. Hackers hebben root-rechten weten te bemachtigen van de server waar de issue tracker JIRA [1] op draait. De aanval bestaat uit verschillende stappen waarbij de aanvaller bij elke stap een stukje meer rechten krijgt. De eerste stap was het misbruiken van een Cross-site scripting-lek in JIRA.

De aanvaller had een issue aangemaakt in JIRA met de volgende inhoud: *ive got this error while browsing some projects in jira http://tinyurl.com/XXXXXXXXX*. Deze tinyURL verwees naar een speciaal vervaardigde link naar de JIRA-instantie die een Cross-site scripting-lek misbruikt. Wanneer een ingelogde gebruiker op deze tinyURL klikte werden de gebruikers' cookies naar de aanvaller gestuurd. Verschillende beheerders hebben op deze link geklikt, waardoor de aanvaller hun sessie kon overnemen, inclusief hun JIRA-rechten. Doordat de aanvaller nu een stuk meer rechten heeft, is deze de volgende stappen gaan doen en heeft uiteindelijk root-rechten over de server verkregen.

Cross-site scripting was mogelijk omdat er een deel van de invoer van de gebruiker direct werd geprint (uitvoer). Dus zonder deze eerst te encoderen voor de juiste context. Hieronder staat de regel code met die het Cross-site scripting-lek bevat:

```
<input type="text" name="commentAuthor" value=
"<%= TextUtils.noNull(request.getParameter("commentAuthor")) %>" />
```

Na de aanval is er zeer snel een patch voor dit lek gekomen:

```
<input type="text" name="commentAuthor" value=
"<%= TextUtils.htmlEncode(TextUtils.noNull(request.getParameter("commentAuthor"))) %>" />
```

Te zien is dat de uitvoer nu wordt geëncodeerd voor de context waarin het geplaatst wordt middels `TextUtils.htmlEncode()` . Wanneer de Anti-XSS tool zou zijn toegepast, zou elke vorm van uitvoer worden gecheckt of deze is geëncodeerd. Er zou dan een waarschuwing over gegeven worden, waardoor dit lek makkelijk gedicht had kunnen worden voordat de code in gebruik werd genomen. De presentatielaag van JIRA is JSP[2] en de Anti-XSS tool werkt momenteel alleen op Velocity. Desalniettemin gaat het om precies hetzelfde principe, en zou de Anti-XSS tool op dezelfde patronen kunnen checken.

3.2 Types Cross-site scripting

In het algemeen wordt er onderscheid gemaakt tussen twee verschillende types Cross-site scripting-aanvallen; persistente en weerspiegelde (reflected) Cross-site scripting. Deze twee verschillen op de manier waarop een script geïnjecteerd wordt in een webapplicatie.

[1] http://www.atlassian.com/software/jira/
[2] http://java.sun.com/products/jsp/

3.2.1 Persistentie

Bij persistente Cross-site scripting zorgt de aanvaller ervoor dat de kwaadaardige code opgeslagen wordt in een gegevensbron die door de webapplicatie wordt gebruikt.

Bijvoorbeeld een database van pagina's. Pas wanneer een gebruiker de pagina opvraagt waarin de kwaadaardige code getoond wordt, zal de daadwerkelijke aanval uitgevoerd worden. Dit is bijvoorbeeld mogelijk op een forum waar verschillende gebruikers berichten kunnen plaatsen. De aanvaller plaatst een bericht dat kwaadaardige JavaScript-code bevat zoals te zien in Figuur 3.2. Dit bericht wordt opgeslagen in de database van pagina's. Pas wanneer een gebruiker het geplaatste bericht bekijkt zal deze de kwaadaardige code ontvangen als deel van het bericht. De browser van de gebruiker zal dan deze code uitvoeren, welke de cookie van de gebruiker naar de aanvaller stuurt.

```
1 Nu is uw cookie gestolen!
2 <img src="foto.jpg">
3 <script>
4   document.images[0].src =
5   "http://attackerssite.com/foto.jpg?gestolencookie="+document.cookie;
6 </script>
```

Figuur 3.2: Persistente Cross-site scripting

3.2.2 Weerspiegeling

Bij de weerspiegelde aanval wordt de kwaardaardige code niet opgeslagen, maar direct 'weerspiegeld' of 'teruggekaatst' naar de gebruiker.

Bijvoorbeeld een zoekformulier dat de ingevulde zoektermen direct in de resultaten toont. Deze zwakheid kan door een aanvaller misbruikt worden door iemand een URL te sturen die verwijst naar het zoekformulier. In deze URL zit dan een stukje kwaadaardige code verwerkt. Figuur 3.3 toont een dergelijke URL. Wanneer de gebruiker op de URL klikt, zal het zoekformulier verzonden worden met de kwaadaardige code als zoekterm. De kwaadaardige code wordt direct teruggestuurd naar de gebruiker. Deze code wordt vervolgens uitgevoerd in zijn browser als deel van de pagina met zoekresultaten.

```
http://example.com/search.php?query=<script>alert('XSS mogelijk!');</script>
```

Figuur 3.3: Weerspiegelde Cross-site scripting

3.3 Cross-site scripting voorkomen

Om Cross-site scripting-lekken uit te sluiten moet het onmogelijk zijn om de uitvoer van een webapplicatie te kunnen beïnvloeden. Door altijd op elke uitvoer correcte encodering toe te passen kunnen alle types Cross-site scripting voorkomen worden.

3.3.1 Begrijp de context van de uitvoer

Injectie breekt uit een gegevenscontext en schakelt over naar code-context. Een voorbeeld van gegevenscontext is hieronder te zien in.

```
<div> gegevenscontext </div>.
```

Als een stukje kwaadaardigde code in de gegevenscontext geplaatst wordt, kan deze uitbreken. Bijvoorbeeld:

```
 <div> gegevens <script>alert('XSS mogelijk!');</script> context </div>.
```

Hier wordt er een subcontext geïntroduceerd dat scripting toestaat. Om te voorkomen dat er uit de HTML-context gebroken kan worden dienen de volgende karakters geëncodeerd te worden als HTML:

```
&, <, >, ", ', /
```

De HTML-representatie van deze karakters ziet er als volgt uit:

```
 & wordt &amp;
 < wordt &lt;
 > wordt &gt;
 " wordt &quot;
 ' wordt &#x27;
 / wordt &#x2F;
```

Er zijn verschillende variaties op de net besproken aanval. In het volgende voorbeeld wordt de invoer niet direct in HTML gezet, maar in het 'value'-attribuut van een HTML-element. Hieronder is een formulier te zien dat opnieuw wordt opgebouwd wanneer er een fout optreedt.

```
Er is een fout opgetreden tijdens het opslaan.
Probeer het nog eens. <br />
<form action="/verwerkformulier" method="post">
  Naam:
  <input type="text" name="naam" value="$request.getParameter('naam')" />
  <input type="submit">
</form>
```

Hier kan uit de context gebroken worden door de volgende regel in te voeren als naam

```
"><script>alert('XSS')</script><"
```

De "> zorgt ervoor dat de huidige context (het HTML-attribuut) wordt afgesloten. Vervolgens wordt er een nieuwe context gestart middels de <script>-tag. Hier kan echter niet dezelfde encodering toegepast worden als voorgesteld bij het eerste voorbeeld. Stel dat er bij naam V&D wordt ingevuld. Als er een fout optreedt, zal er vervolgens komen te staan: V&D. Verder zijn het aanhalingstekens die de huidige context kunnen afsluiten. Dus in dit voorbeeld dienen de aanhalingstekens van de invoer geëscaped te worden voordat ze op de uitvoer worden gezet. De uitvoer zou er dan als volgt uitzien:

```
\"><script>alert(\'XSS\')</script><\"
```

Doordat de aanhalingstekens nu geëscaped zijn kunnen ze niet de huidige context afsluiten.

Er zijn nu twee voorbeelden gegeven hoe onvoldoende encodering misbruikt kan worden om uit de context te breken. Verder is laten zien dat elke context een eigen manier van encoderen nodig heeft.

3.3.2 Gebruik veilige encodering

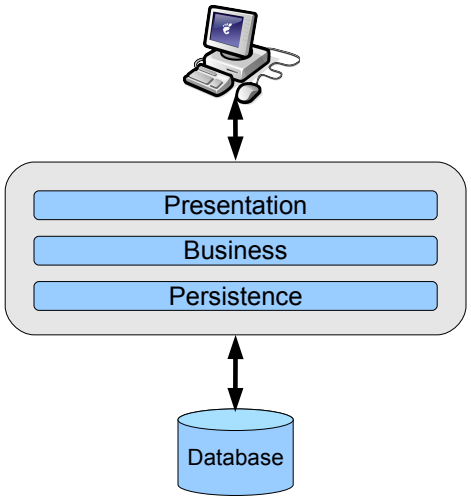
Alle uitvoergegevens dienen specifiek te worden geëncodeerd voor de context waarin ze worden geplaatst. In principe is het niet erg lastig om deze encoderingen te programmeren. Echter zijn er wel een hoop valkuilen.

Neem bijvoorbeeld het invoegen van gegevens in een JavaScript-context. Het lijkt misschien logisch om aanhalingstekens te escapen middels `\"`. Maar dit zijn gevaarlijke waarden en kunnen verkeerd geïnterpreteerd worden door de geneste parsers van de browser. Verder kun je vergeten het escape-karakter zelf (de slash) te escapen. Als de slashes niet escaped worden kunnen aanvallers dit gebruiken om hun aanhalingsteken te neutraliseren. In het net genoemde voorbeeld wordt `"` escaped naar `\"`. Als de slashes niet escaped worden kan een aanvaller `\"` invoeren. Deze wordt dan escaped naar `\\"`. In plaats van dat het aanhalingsteken geëscaped is, is de slash escaped en wordt het aanhalingsteken gewoon geprint.

Daarom is het van belang om voor encodering een bibliotheek te gebruiken welke focust op veiligheid en deze regels implementeert. Een goed voorbeeld is de encoder van de Enterprise Security API (ESAPI) van OWASP. Hier wordt aangeven voor welke context gegevens geëncodeerd moeten worden. En de ESAPI zorgt voor de juiste encodering. Deze bibliotheek is gebouwd en wordt gecontroleerd door experts op het gebied van webapplicatie-security en is voor iedereen in te kijken (open source). Een dergelijke oplossing wordt aanbevolen boven het zelf bouwen van security-mechanismen.

3.3.3 Webapplicatie architectuur

In figuur 3.4 is geschetst hoe een webapplicatie in het algemeen eruit ziet. Technisch gezien hoeft niet elke webapplicatie op een dergelijke manier te zijn opgebouwd, maar de functionaliteit van



Figuur 3.4: Abstracte representatie van een webapplicatie

deze drie lagen zit wel in elke webapplicatie.

Presentation De presentatielaag is de interface tussen de logica van de applicatie en de gebruiker. Deze laag zorgt voor de opmaak van de uitvoer. Hier wordt dus ook bepaald in welke context de gegevens komen te staan. Alle gegevens die nodig zijn voor de opmaak worden uit de businesslaag verkregen.

Business In de businesslaag vindt de logica van de webapplicatie plaats. Bijvoorbeeld: in de gegevensbron zijn naam en achternaam als twee losse entiteiten opgeslagen. Maar de presentatielaag vereist één entiteit. De Businesslaag kan dan ervoor zorgen dat de twee velden worden opgehaald, aan elkaar worden geplakt, en dan doorgegeven worden aan de presentatielaag.

Persistence De persistentielaag zorgt voor communicatie met de gegevensbronnen. Hierdoor kan de businesslaag op een uniforme manier gegevens ophalen en toevoegen, ongeacht in welke en wat voor soort bron deze gegevens zijn opgeslagen.

Data source Gegevensbron; in dit onderdeel worden alle gegevens opgeslagen. Meestal is dit een database. Maar dat zouden ook bestanden of een externe feed kunnen zijn.

Figuur 3.5 is een voorbeeld van een webapplicatie welke gegevens uit een onbetrouwbare bron haalt. Te zien is dat het ophalen van deze gegevens wordt gedaan door de 'Persistentielaag'. Zoals in paragraaf 3.3.3 is uitgelegd zal de presentatielaag bepalen in welke context deze gegevens

uiteindelijk terecht komen. Om Cross-site scripting hier uit te kunnen sluiten dient de encodering dus in de presentatielaag toegepast te worden.

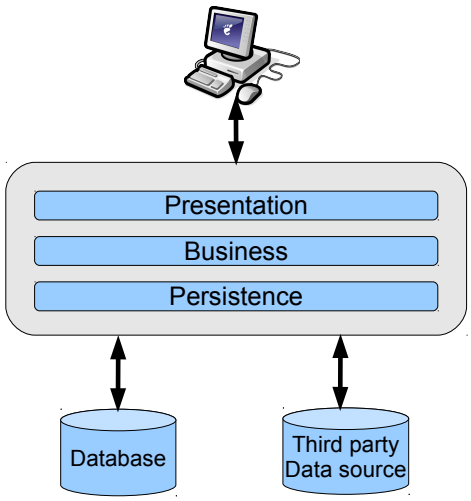
3.3.4 Cross-site scripting-risico's verlagen

Er zijn nu verschillende maatregelen aangedragen waarmee Cross-site scripting voorkomen kan worden. Echter zijn er ook maatregelen die de impact van Cross-site scripting-aanvallen aanzienlijk verkleinen.

Cookie-diefstal voorkomen met HTTPOnly

Veelvoorkomende Cross-site scripting-aanvallen stelen cookies van het slachtoffer met behulp van JavaScript. Zoals in hoofdstuk 3 is laten zien, injecteert de aanvaller een stukje clientside-script om cookies uit te lezen. De kans op een dergelijke aanval kan voorkomen worden door de HTTPOnly-vlag aan te zetten in de header van de HTTP-response. Dit zegt tegen de browser dat de cookies niet beschikbaar zijn voor clientside-scripts. Wanneer de HTTPOnly-vlag aan staat zullen de cookies dus wel meegestuurd worden als er een webapplicatie wordt bezocht in het juiste domein, maar voor JavaScript zal de cookie niet toegankelijk zijn. Stel dat er een Cross-site scripting-lek aanwezig is in de webapplicatie, dan kunnen de cookies van slachtoffers in ieder geval niet gestolen worden middels JavaScript.

Zoals al is aangegeven is dit een prima middel om de kans op cookie-diefstal te verkleinen. Een belangrijke voorwaarde is dat de browser HTTPOnly moet ondersteunen. Wanneer de browser deze



Figuur 3.5: Webapplicatie haalt gegevens vanaf een derde partij.

optie niet heeft geïmplementeerd wordt de HTTPOnly-vlag genegeerd en zullen clientside-scripts toegang hebben tot cookies.

3.4 Conclusie

Nu is duidelijk wat Cross-site scripting inhoudt en hoe het voorkomen kan worden. In het volgende hoofdstuk zal een methode worden onderzocht om mogelijke Cross-site scripting-lekken op te sporen in een webapplicatie. Omdat dit onderzoek zich voornamelijk richt op Java webapplicaties is er gekozen om de presentatielaag Velocity als voorbeeld te nemen.

Hoofdstuk 4

Webapplicaties in dotCMS met Velocity

Velocity is een template engine gebasseerd op Java. Velocity kan gegevens uit Java-objecten presenteren in onder andere platte tekst, HTML, XML, PDF, SQL en PostScript. In webapplicaties kan het MVC-model

krachtig afgedwongen worden doordat Velocity-templates geen "Business-logica" bevatten. Dit stelt ontwikkelaars in staat de focus te leggen op het schrijven van functionele code, terwijl ontwerpers zich beperken tot aantrekkelijke uitvoer.

4.1 De werking van Velocity

In deze paragraaf wordt beschreven wat een template engine is en hoe Velocity werkt.

Een template engine gebruikt data en één of meerdere templates om vervolgens één of meerdere resultaatdocumenten te genereren. De template engine zorgt ervoor dat data en presentatie, welke gescheiden aangeleverd worden, samengesmolten worden tot een document. Er is meestal ook wat simpele logica mogelijk; zoals het conditioneel tonen van een stuk tekst (if/else), en door een lijst van waarden itereren. Deze mogelijkheden zijn bedoeld om voor de presentatie te gebruiken. Iteratie kan bijvoorbeeld gebruikt worden om data op te sommen dat in een lijststructuur is opgeslagen.

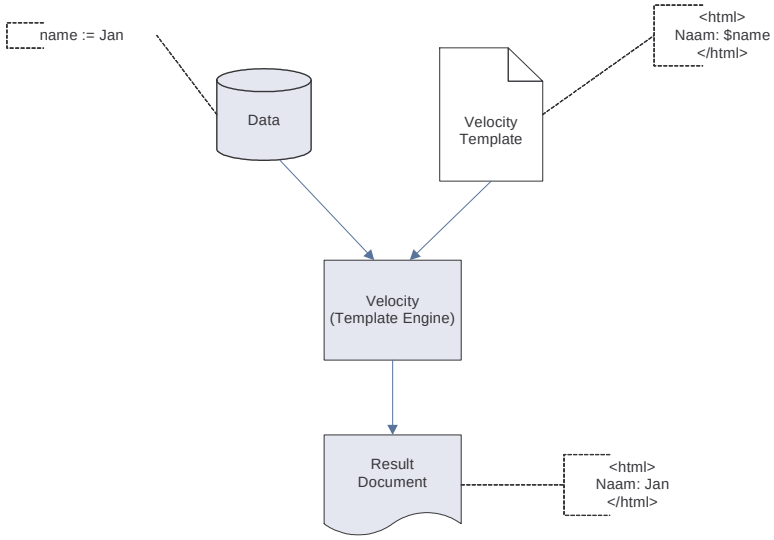
De Velocity Template Engine is open source en verkrijgbaar onder de Apache Software-licentie. Deze licentie staat toe om de broncode aan te passen en de aangepaste code te distribueren. Verder mag het gebruikt worden in zowel open source en vrije software als gesloten software. In Velocity-templates kunnen *references* (opmaakdeclaraties) worden opgenomen. Deze references worden van een *Context*-object gehaald, en de daarbij behorende waarde wordt dan in het document gezet.

Om een document te genereren met Velocity moeten de volgende stappen worden doorlopen:

1. De data beschikbaar stellen aan Velocity.
2. Aangegeven welke template(s) gebruikt dient te worden.

3. Aangeven dat de data en de template gemerged moeten worden.

Zie Fig. 4.1 voor een illustratie van deze stappen.



Figuur 4.1: Data en een template worden samengesmolten tot een HTML document

Voordat de werking van de checker wordt uitgelegd, zal er concreter worden uitgelegd hoe een document geproduceerd kan worden middels Velocity.

1. Instantieer een `VelocityEngine`.
2. Instantieer een `VelocityContext`.
3. Maak een `Writer` aan. Een `Writer` is een abstracte klasse om 'character streams' weg te schrijven, bijvoorbeeld naar een tekstbestand.
4. Zet de data in de `VelocityContext`. De data dient aangeleverd te worden als sleutel/waarde-paar. De sleutel is van het type `String` en de waarde van het type `Object` dus de waarde kan elk datatype in Java representeren. In Fig. 4.1 is 'name' de sleutel en 'Jan' de waarde.
5. Vervolgens roep `VelocityEngine.mergeTemplate()` aan; en geef aan deze methode de template, de context, en de writer waarnaar het resultaat document geschreven moet worden mee.

De 'references' zijn de plekken in de template waar uitvoer wordt gedaan. Tijdens het samenvoegen (`mergeTemplate`) zoekt Velocity naar references in de template. Een reference is van de vorm `$sleutel` of `${sleutel}`. Wanneer de reference op de context staat, zal deze vervangen worden door de waarde ervan. In het gegeven voorbeeld zal `$name` vervangen worden door 'Jan'.

4.2 Velocity in dotCMS

DotCMS is een Web Content Management Systeem. Het biedt de mogelijkheid om meerdere websites te draaien en de inhoud daarvan te beheren. DotCMS komt in twee verschillende versies, namelijk dotCMS Enterprise en dotCMS Open Source. Voor het onderzoek is dotCMS Open Source gebruikt. Deze versie is gratis beschikbaar en te downloaden[1] en is gebasseerd op de volgende open source-technologieën:

- Liferay Enterprise Portal - Portal
- Apache Struts - MVC Framework
- DWR - Java AJAX framework
- Spring - Application Framework
- Hibernate ORM - Object Relation Mapping Engine
- Velocity Templating Language - Templating Engine
- Velocity Tools - Expose tools and widgets to Velocity Templates
- Lucene Search Engine - Java search engine
- TinyMCE - Cross browser WYSIWYG editor

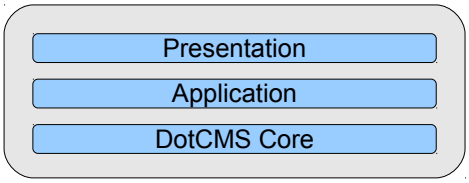
De twee Velocity-projecten, 'Velocity Templating Language' en 'Velocity Tools', zijn van belang voor dit onderzoek.

In figuur 4.2 is beknopt weergegeven hoe de architectuur van dotCMS in elkaar zit. Voor een gedetailleerd overzicht zie Appendix D.2 Architectuur van dotCMS. De Velocity-templates bevinden zich in de presentatielaag van dotCMS. De presentatielaag wordt uitgebreid behandeld in paragraaf 3.3.3 Webapplicatie architectuur.

Velocity als template engine bevindt zich in de 'Application-laag'. In deze laag worden de `VelocityEngine` en `VelocityContext` geïnstantieerd. Vervolgens wordt de benodigde data opgevraagd uit de 'dotCMS Core'-laag en in de `VelocityContext` gezet. Als Velocity een resultaat heeft gemaakt wordt dit document verstuurd naar de gebruiker. Het resultaat is bijvoorbeeld het opgemaakte HTML-document waar de gebruiker om vroeg.

4.3 Templates en Containers

In dotCMS kan Velocity-code in twee verschillende entiteiten opgeslagen worden; in een template of in een container. In dotCMS zijn zowel templates als containers in principe Velocity-templates. Het verschil zit in de manier waarop deze door dotCMS worden opgeslagen en gebruikt. Een template in dotCMS zal vanaf nu een dotCMS-template genoemd worden. Zodoende kan deze niet verward worden met een Velocity-template.



Figuur 4.2: Architectuur dotCMS

Een dotCMS-template bevat de layout van een pagina. Het is te vergelijken met een HTML-pagina, met het verschil dat het Velocity-code kan bevatten. Verder kan een dotCMS-template containers bevatten. Containers die in templates zijn geplaatst definiëren het gebied waar gebruikers content kunnen toevoegen en hoe deze content wordt gepresenteerd.

Containers hebben twee functies. Allereerst worden ze gebruikt als 'include'-bestanden. Zo kunnen containers worden gebruikt om stukken HTML-code in te plaatsen die gebruikt worden in verschillende templates. Een goed voorbeeld van dergelijke containers zijn een globale header, footer, een menubalk of andere websitebrede content. Daarnaast kan een container bepalen waar de content geplaatst wordt op de template en hoe deze content opgemaakt moet worden. In Appendix D.1 Templates en Containers wordt een voorbeeld van een dotCMS-template gegeven. Tevens zijn daaronder de containers neergezet die in deze dotCMS-template worden gebruikt.

Hoofdstuk 5

Voorkom Cross-site scripting in Velocity

Het aantal mogelijke Cross-site scripting-lekken kan verlaagd worden door elke vorm van uitvoer op de juiste manier te encoderen. Door adequate encodering van uitvoer wordt het succesvol injecteren van scripts voorkomen. Dit houdt in dat uitvoer wordt geëncodeerd aan de hand van de context waarin het geplaatst gaat worden. Voorbeelden van verschillende contexten zijn HTML body, JavaScript en CSS. Elke context heeft een specifieke manier van encoderen nodig. Velocity encodeert standaard niets.

5.1 Velocity Anti-XSS tool

Het is nu duidelijk hoe Cross-site scripting voorkomen kan worden en hoe Velocity werkt. In dit hoofdstuk zal uitgelegd worden hoe ervoor gezorgd kan worden dat mogelijke Cross-site scripting-lekken worden voorkomen in Velocity-templates. Er wordt nu gekeken naar een oplossing om Velocity-templates te controleren.

Zelf iets bouwen dat templates checkt

Er kan zelf een tool gemaakt worden dat Velocity-templates controleert. De grammatica van een Velocity-template is vastgelegd in JavaCC. Hierdoor kan met behulp van JavaCC gemakkelijk een parser gegenereerd worden voor Velocity-templates.

Een uitbreiding op Velocity maken die templates checkt

De broncode van het Velocity-project is open source verkrijgbaar en mag aangepast worden. Deze code bevat functionaliteit welke templates inleest en vervolgens bepaalde dingen doet. Zoals een *reference* vervangen voor de bijbehorende waarde. In principe moet de Anti-XSS tool een template

inlezen en controleren of elke reference geëncodeerd is. De functionaliteit voor het inlezen van een template en vinden van references zit al in de Velocity Template Engine.

5.2 Aanpassingen Velocity

Er is gekozen om een aanpassing op Velocity te maken. Op het moment van mergen (zie paragraaf 4.1 De werking van Velocity) wordt dan gecheckt of er encoding op de variabelen is toegepast. De check vindt plaats op het moment voordat een Reference wordt vervangen door zijn waarde.

Hier wordt toegelicht welke aanpassingen er zijn gedaan aan de bestaande Velocity-code. In principe zal de Velocity Engine alleen worden gebruikt om te bepalen welke plekken in de template niet geëncodeerd zijn. Maar uiteindelijk willen we de checker wel meer laten doen. Deze moet minstens een lijst produceren met niet geëncodeerde plekken in een template. Omdat dit buiten de scope valt van het doel van de functionaliteit van Velocity, is geprobeerd de code zoveel mogelijk buiten de Velocity Engine te schrijven. Hierdoor zal de tool gebruikt kunnen worden met een minimale code aanpassing aan een bestaande versie van Velocity. Om dit te bewerkstelligen is er gebruik gemaakt van een EventHandler. Na het aanmaken van de `VelocityContext`, wordt hier een `NotEscapedEventHandler` aan toegevoegd. Verder is er op het moment dat er een Reference gevonden wordt, dus net voordat deze vervangen gaat worden door zijn waarde, een check gebouwd. Wanneer een Reference niet geëncodeerd is, zal er een `NotEscaped`-event afgevuurd worden. Dit zorgt ervoor dat de `NotEscapedEventHandler` getriggered wordt. Deze krijgt de naam van de template, het regelnummer en het kolomnummer door. Het bijhouden van de data en genereren van een overzicht kan nu netjes buiten de Velocity Engine plaatsvinden.

5.2.1 Checken op encodering

De tool zoekt in Velocity-templates naar 'References'. Vervolgens wordt er gecheckt of deze Reference geëncodeerd is. De tool gaat er vanuit dat er een bibliotheek wordt gebruikt die de encodering verzorgt. Deze bibliotheek of klasse dient in de `VelocityContext` gezet te worden voor erop gecheckt kan worden.

In een configuratiebestand kan aangegeven worden welke class de encodering verzorgt. Hieronder een voorbeeld voor als de encodering wordt gedaan door een standaard bibliotheek van Velocity, de EscapeTool. De instelling in het configuratiebestand komt er als volgt uit te zien:

```
escape.tool.class=org.apache.velocity.tools.generic.EscapeTool
```

Voor elke Reference zal de tool dan checken of EscapeTool hierop wordt toegepast. Na het uitvoeren wordt een lijstje gemaakt van References die niet geëncodeerd zijn. De Reference wordt aangegeven door middel van de naam van het bestand of template, het regelnummer en het kolomnummer (positie op de regel).

5.2.2 Velocity-templates ophalen uit dotCMS

Om de templates uit te kunnen lezen uit dotCMS is er een plug-in voor dotCMS geschreven. Deze plug-in haalt alle Templates en Containers (zie 4.3) van een bepaalde webapplicatie op en biedt ze aan in XML formaat. Vervolgens heb ik een uitbreiding op de tool gemaakt zodat deze direct de XML-uitvoer van de dotCMS-plug-in kan inlezen. Zodoende kan de tool op elk moment de meest recente Velocity-code van een bestaande webapplicatie checken.

5.3 Testen

Om te bepalen welke stukjes code precies gecheckt moeten worden is er gekeken naar de Velocity-template Language Referentie [4]. Voor elke mogelijke taalconstructie in deze specificatie is een voorbeeld-template en een test gemaakt. Zoals in paragraaf 4.1 is uitgelegd kan een reference verschillende vormen hebben. De template in Listing 5.1 toont alle geldige vormen van een reference die uitvoer doet. Deze voorbeeld-template doet uitvoer van variabelen op zes plekken. De tool zou

Listing 5.1: references.vm

```
1 Variables:
2 Shorthand notation: $var
3 Silent Shorthand notation: $!var
4 Formal notation: ${var}
5 Silent Formal notation: $!{var}
6
7 Properties:
8 Regular Notation: $customer.name
9 Formal Notation: ${purchase.total}
```

dus zes plekken moeten checken of deze geëncodeerd zijn. Hieronder zijn de resultaten te zien die de tool geeft na het checken van references.vm.

```
$var [template=references.vm, line=2, column=21] not encoded!
$!var [template=references.vm, line=3, column=28] not encoded!
${var} [template=references.vm, line=4, column=18] not encoded!
$!{var} [template=references.vm, line=5, column=25] not encoded!
$customer.name [template=references.vm, line=8, column=19] not encoded!
${purchase.total} [template=references.vm, line=9, column=18] not encoded!

Total places of output: 6
encoded: 0
non-encoded: 6
```

Alle zes plekken zijn gecheckt en geen enkele plek is geëncodeerd. Cross-site scripting zou dus zeker niet uitgesloten kunnen worden in deze template. In de volgende template (Listing 5.2) wordt de kans op Cross-site scripting verlaagd door elke vorm van uitvoer te encoderen. Hieronder zijn

Listing 5.2: references-escaped.vm

```
1 Variables:
2 Shorthand notation: $esc.html($var)
3 Silent Shorthand notation: $esc.html($!var)
4 Formal notation: $esc.html(${var})
5 Silent Formal notation: $esc.html($!{var})
6
7 Properties:
8 Regular Notation: $esc.html($customer.name)
9 Formal Notation: $esc.html(${purchase.total})
```

de resultaten te zien van references-escaped.vm.

```
Total places of output: 6
encoded: 6
non-encoded: 0
```

De tool heeft alle zes references gecheckt en ze allemaal geëncodeerd bevonden. De overige testen en resultaten zijn opgenomen in Appendix B Velocity-template tests

5.4 Anti-XSS tool in de praktijk

De tool is uitgevoerd op Velocity-templates van een bestaande webapplicatie binnen het open source framework dotCMS. Deze webapplicatie bevat in totaal 28 templates. In paragraaf 5.4.1 wordt een voorbeeld-template gegeven en uitgelegd.

5.4.1 Voorbeeld-template

De template in Listing 5.3 is gekopieerd van de bestaande webapplicatie. Er zijn alleen een aantal HTML-tags weggehaald in dit voorbeeld om het wat overzichtelijk te houden. Deze tags zijn alleen weggehaald in deze illustratie, maar zitten wel in de daadwerkelijke test. Omdat in de weggelaten delen geen uitvoer wordt gedaan heeft dit geen invloed op de resultaten van de Anti-XSS tool.

Listing 5.3: vacature.vm

```
1 ## Define error message
2 #set($msgGeenVac = "<h2>Vacature niet beschikbaar</h2>
```

```
3   <i>De vacature kan niet worden weergegeven</i>")
4
5  #set($vacancyId = $request.getParameter('vacid'))
6
7  #if (!$UtilMethods.isSet($vacancyId))
8    $msgGeenVac
9  #else
10   #getContentDetail("$vacancyId")
11
12   <h1>$!{title}</h1>
13   Datum publicatie: $!{publicatieDatum}
14   Omschrijving: $!{content}
15   <a href="/vacatures/sollicitatie.dot?vacature=$!{title}">
16     Solliciteer op deze vacature
17   </a>
18 #end
```

5.4.2 Resultaten

In totaal zijn er 66 plekken waar uitvoer wordt gedaan.

Variabele	Type	Opmerking
Template-Case_detail		
$!{title}	TP	
$!{content}	TP	
$!{geleverdeTechnologie}	TP	
$!{geleverdeInspanning}	TP	
$!{aantalGebruikers}	TP	
$!{periode}	TP	
Template-Cases_XML		
$cnt	TP	
$!case.opHomepage	TP	
$case.intro	TP	
$cnt	TP	
$case.url	TP	
Template-Contact_routebeschrijving		
$HTMLPAGE_TITLE	TP	
$!gps_points	TP	
$!{centertext}	TP	
$!address	TP	
Detailpagina		

$!content.title	TP	
$!content.content	TP	
$!content.body	TP	
Template-Homepage		
$HTMLPAGE_META	TP	
Template-Zoekresultaten		
$searchResult.identifier	TP	
$searchResult.title	TP	
$searchResult.title	TP	
$searchResult.identifier	TP	
$searchResult.title	TP	
$searchResult.identifier	TP	
$searchResult.title	TP	
$searchResult.identifier	TP	
$searchResult.title	TP	
Container-Cases		
$!title	TP	
$!title	TP	
$!title	TP	
$!logoImageURI	TP	
$!contentIntro	TP	
$!{ContentInode}	TP	
Container-googleMaps		
$!{latitude}	**FP**	${latitude} wordt toegewezen, geen uitvoer
$!{longitude}	**FP**	${longitude} wordt toegewezen, geen uitvoer
Container-Header		
$HTMLPAGE_TITLE	TP	
$HTMLPAGE_META	TP	
${term}	TP	
Container-Rightside		
$!{title}	TP	
$!{content}	TP	
$url	TP	
Container-Rightside_dyn		
$UtilMethods.encodeURL($!{content.Title})	**FP**	Geen encodering d.m.v. EscapeTool
$!{content.inode}	TP	
$!{content.title}	TP	
$!{content.Intro}	TP	
$detailLink	TP	
Container-Smoelenboek		
$!{photoImageURI}	TP	

$!{name}	TP	
$!{name}	TP	
$!{function}	TP	
$!{content}	TP	
Container-Sollicitatie_formulier		
$!request.getParameter('vacature')	TP	Cross-site scripting-lek
Container-Vacature_Detail_dyn		
$msgGeenVac	TP	
$!{title}	TP	
$!{publicatieDatum}	TP	
$!{content}	TP	
$!{title}	TP	
$msgGeenVac	TP	
Container-Vacatures_dyn		
$msgGeenVac	TP	
$!{oVac.title}	TP	
$!{oVac.intro}	TP	
$!{oVac.inode}	TP	
$msgGeenVac	TP	
Container-Webpage_Content		
$!{title}	TP	
$!{body}	TP	
Variabele	**Type**	**Opmerking**

True Positives: 63
False Positives: 3

Bijna elke waarschuwing is terecht. Er zijn twee waarschuwingen (False Positives) waar een variabele wordt toegewezen (assignment), maar deze worden niet geprint, dus zijn geen uitvoer. Dit is punt waarop de tool verbeterd kan worden. Verder wordt `$UtilMethods.encodeURL( $!{content.Title})` ook opgemerkt als niet geëncodeerd. In feite is deze wel geëncodeerd, maar niet door middel van de EscapTool zoals de tool verwacht. Dit is dus geen potentieel Cross-site scripting-lek. Om deze waarschuwing weg te halen zal dat stukje uitvoer veranderd moeten worden naar: `\$esc.url( $!{content.Title})`.

De code `$!request.getParameter( 'vacature')` is gevoelig voor weerspiegelde Cross-site scripting. Hier wordt namelijk gebruikersinvoer direct als uitvoer gebruikt. De rest van de gevonden plekken zijn eventueel gevoelig voor persistente Cross-site scripting.

Hoofdstuk 6

Conclusie

In deze scriptie is onderzoek gedaan naar het voorkomen van veelgemaakte fouten in software. Als eerst zijn alle punten uit de CWE/SANS top 25 van meest gevaarlijke programmeerfouten geanalyseerd. Per fout in deze top 25 is er een korte toelichting gegeven. Tevens is er gekeken hoe goed het punt te voorkomen is en of de fout statisch of dynamisch analyseerbaar is. Naar aanleiding van dit vooronderzoek is gebleken dat Cross-site scripting één van de meest gevaarlijke en meest voorkomende fouten is.

Om de omvang van het onderzoek te beperken is er gericht op het voorkomen van Cross-site scripting in Java wepapplicaties. Een webapplicatie is gevoelig voor Cross-site scripting wanneer invoerparameters direct in de uitvoer worden gezet. Uitvoer wordt in de gegevenscontext van een webapplicatie gezet. Voorbeelden van verschillende contexten zijn HTML-body, JavaScript en CSS. Wanneer deze uitvoer niet voor de juiste gegevenscontext wordt geëncodeerd kan het als 'code' worden geïnterpreteerd, wat aanvallers de mogelijkheid geeft om kwaadaardige code te injecteren. Om dit te voorkomen dient elk stukje uitvoer voor de *juiste* context te worden geëncodeerd. De *juiste* context is de gegevenscontext waarin uitvoer geplaatst gaat worden.

Om van een webapplicatie te controleren of alle uitvoer is geëncodeerd is de Velocity Anti-XSS tool ontwikkeld. Deze tool checkt de code door middel van statische analyse en geeft aan op welke plekken er uitvoer wordt gedaan zonder deze te encoderen. Cross-site scripting kan uitgesloten worden wanneer de webapplicatie aan de volgende twee eisen voldoet:

1. Elke vorm van uitvoer is geëncodeerd voor de juiste context.

2. Het encoderen gebeurt middels een veilig bewezen routine of bibliotheek.

Het kan gewenst zijn dat een bepaalde uitvoer juist niet geëncodeerd mag worden. Bijvoorbeeld wanneer deze uitvoer HTML bevat dat door een *vertrouwde* gebruiker is ingevoerd. Er dient dan expliciet aangegeven te worden dat deze uitvoer niet geëncodeerd hoeft te worden, en *vertrouwd* wordt. Hiermee wordt wel het risico op Cross-site scripting verhoogt. Verder wordt het lastiger om Cross-site scripting aantoonbaar uit te sluiten. Om Cross-site scripting uit te kunnen sluiten zal

dan per specifiek *vertrouwde* uitvoer bewezen moeten worden dat deze veilig is, zodoende dat deze geen Cross-site scripting kan bevatten.

Het is vrijwel onmogelijk om Cross-site scripting aantoonbaar uit te sluiten voor webapplicaties in het algemeen. Er zijn verschillende platforms en programmeertalen waarin webapplicaties ontwikkeld worden. Deze eisen elk een specifieke manier of bibliotheek voor encodering. Er kan wel vastgelegd worden welke plekken (calls, methodes) uitvoer doen. Maar alleen in de presentatielaag is bekend in welke context deze uitvoer gezet wordt. Daarom zal elke presentatielaag op een specifieke manier gecontroleerd moeten worden.

6.1 Bredere toepasbaarheid

Voor dit onderzoek is de Velocity Anti-XSS tool geschreven om Velocity-templates te controleren. Deze Anti-XSS tool is beperkt toepasbaar; alleen wanneer Velocity als presentatielaag wordt gebruikt. De methode die is gebruikt om deze tool de maken is echter breder toepasbaar.

Er wordt vereist dat een bibliotheek gebruikt wordt dat de encodering verzorgd. Voor alle talen op het Java-platform kan de ESAPI van OWASP gebruikt worden als bibliotheek voor encodering. Vervolgens moet de code zo aangepast worden dat elke vorm van uitvoer wordt geëncodeerd door deze bibliotheek. De inspanning die nodig is om deze methode op andere talen toe te passen is relatief laag. Het aanpassen van de Velocity Anti-XSS tool is het belangrijkste dat er moet gebeuren. Echter bestaande code aanpassen zodat het voldoet aan de eisen om Cross-site scripting uit te sluiten, dus elke vorm van uitvoer encoderen, kan erg veel inspanning vereisen. Dit is sterk afhankelijk van de omvang en complexiteit van de bestaande code. Maar voor een project waar nog aan begonnen moet worden is deze methode uitermate geschikt, en kost praktisch geen extra inspanning tijdens de ontwikkeling.

Bibliografie

[1] Dotcms. http://www.dotcms.org/.

[2] Pmd. http://pmd.sourceforge.net/.

[3] E. Fong and V. Okun. Web Application Scanners: Definitions and Functions. In *HAWAII INTERNATIONAL CONFERENCE ON SYSTEM SCIENCES*, volume 40, page 4707. IEEE, 2007.

[4] Apache Software Foundation. Velocity template language reference. http://velocity.apache.org/engine/devel/vtl-reference-guide.html.

[5] Mozilla Foundation. Javascript security: Same origin. http://www.mozilla.org/projects/security/components/same-origin.html, 2006.

[6] D. Hovemeyer and W. Pugh. Finding bugs is easy. *ACM SIGPLAN Notices*, 39(12):92–106, 2004.

[7] B. Livshits, M. Martin, and M.S. Lam. SecuriFly: Runtime Protection and Recovery from Web Application Vulnerabilities. Technical report, Technical report, Stanford University, Sept. 2006, 2006.

[8] B. Martin, M. Brown, A. Paller, and S. Christey. cwe/sans top 25 most dangerous programming errors. http://cwe.mitre.org/top25/archive/2009/2009_cwe_sans_top25.html, [Accessed: January 10, 2011], 2009.

[9] OWASP. Enterprise security api. http://www.owasp.org/index.php/ESAPI.

[10] OWASP. Csrf guard project. http://www.owasp.org/index.php/Category:OWASP_CSRFGuard_Project, [Accessed: June 1, 2010], 2010.

[11] OWASP. Csrftester project. http://www.owasp.org/index.php/Category:OWASP_CSRFTester_Project, [Accessed: June 1, 2010], 2010.

[12] N. Rutar, CB Almazan, and JS Foster. A comparison of bug finding tools for Java. In *Software Reliability Engineering, 2004. ISSRE 2004. 15th International Symposium on*, pages 245–256, 2004.

[13] Inc. Sun Microsystems. Secure coding guidelines for the java programming language, version 3.0. [Online]. Available: http://www.oracle.com/technetwork/java/seccodeguide-139067.html. [Accessed: Aug. , 2011].

[14] Apache Infrastructure Team. issues.apache.org compromised. `https://blogs.apache.org/infra/entry/apache_org_04_09_2010`, 2010.

[15] P. Vogt, F. Nentwich, N. Jovanovic, E. Kirda, C. Kruegel, and G. Vigna. Cross-site scripting prevention with dynamic data tainting and static analysis. In *Proceeding of the Network and Distributed System Security Symposium (NDSS07)*. Citeseer, 2007.

[16] Michael S. Ware and Christopher J. Fox. Securing java code: heuristics and an evaluation of static analysis tools. *SAW '08: Proceedings of the 2008 workshop on Static analysis*, pages 12–21, 2008.

[17] G. Wassermann and Z. Su. Static detection of cross-site scripting vulnerabilities. In *Proceedings of the 30th international conference on Software engineering*, pages 171–180. ACM, 2008.

[18] WebScarab. `http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project`.

Bijlage A

Cross-site scripting-lekken in Blackboard

In Blackboard versie 9 wordt uitvoer niet of niet juist geëncodeerd. Zo is het voor een student mogelijk om JavaScript te injecteren in het forum. Hiermee is het mogelijk om cookies te stelen waardoor iemands sessie overgenomen kan worden.

Hier wordt een extern JavaScript-bestand ingeladen in de huidige pagina. Hier kan van alles in staan, maar voor Blackboard zijn we geïnteresseerd in het verkrijgen van cookies van een andere gebruiker zodat diens sessie overgenomen kan worden.

Wanneer deze JavaScript-code uitgevoerd wordt, zal de browser een plaatje opvragen op de URL http://website.com/?cookie=. Achter cookie= wordt de cookie van de huidige gebruiker gezet. Op website.com draait een script die de cookie opslaat. Nu kan de student zijn eigen cookie vervangen door die van de andere gebruiker. De student zal nu Blackboard kunnen gebruiken als die andere gebruiker. Als die andere gebruiker een docent is, heeft de student dezelfde rechten als docent verkregen. Waardoor de student dus bijvoorbeeld cijfers kan veranderen of tentamens kan inzien.

A.1 Mogelijke oplossingen

De oorzaak van het probleem is dat het voor gebruikers mogelijk is om HTML-tags en JavaScript in te voeren op verschillende pagina's. In het voorbeeld heb ik laten hoe je hiermee iemands sessie kan overnemen door de cookies te stelen. Echter zijn er nog vele andere kwaadaardige dingen mogelijk als je scripts kunt invoeren (welke later worden uitgevoerd door de browser van het slachtoffer).

Een oplossing is om Blackboard zodanig te veranderen dat het niet mogelijk is om HTML-tags in te voeren. Of als ze wel ingevoerd kunnen worden, dan mogen ze niet door de browser van het slachtoffer uitgevoerd kunnen worden als script, maar moet gewoon de ruwe tekst getoond worden. Een relatief eenvoudige oplossing is om white listing toe te passen. Hiermee kun je aangeven welke HTML-tags toegestaan zijn. Alles wat dan niet in de white list voorkomt zal op de juiste manier geëncodeerd worden. Op deze manier kun je ervoor dat er wel bepaalde HTML ingevoerd kan

worden, maar geen potentieel kwaadaardige HTML-tags of JavaScript.

Bijlage B

Velocity-template tests

B.1 if-else

Listing B.1: ifelse.vm

```
#set ($escaped = 1)

escaped = $escaped

#if ($escaped)
    The condition is $esc.html($waar)
#else
    The condition is $onwaar
#end
```

Hieronder de resultaten van ifelse.vm B.1.

```
$encoded [template=ifelse.vm, line=3, column=11] not encoded!
$onwaar [template=ifelse.vm, line=8, column=22] not encoded!

Total places of output: 3
encoded: 1
non-encoded: 2
```

B.2 foreach

Hieronder de resultaten van foreach.vm B.2.

Listing B.2: foreach.vm

```
1
2 list: $list
3
4 #foreach( $name in $list )
5     $name is great!
6     $esc.html($name) is great AND properly encoded!
7 #end
```

```
$list [template=foreach.vm, line=2, column=7] not encoded!
$name [template=foreach.vm, line=5, column=5] not encoded!

Total places of output: 3
encoded: 1
non-encoded: 2
```

B.3 include

Listing B.3: include.vm

```
1
2 ## To test the inclusion of a template
3 The contents of the file are not rendered through the template engine
4
5 #include("include2.vm")
```

Hieronder de resultaten van include.vm B.3.

```
Total places of output: 0
encoded: 0
non-encoded: 0
```

B.4 parse

Hieronder de resultaten van parse.vm B.4.

```
$html [template=include2.vm, line=3, column=7] not encoded!
```

Listing B.4: parse.vm

```
1
2 ## To test the inclusion of a template
3
4
5 #parse("include2.vm")
```

```
Total places of output: 1
encoded: 0
non-encoded: 1
```

B.5 macro

Listing B.5: macro.vm

```
1
2 #macro( tablerows $color $somelist )
3   #foreach( $something in $somelist )
4       <tr><td bgcolor=$color>$something</td></tr>
5   #end
6 #end
7
8
9 #set( $greatlakes = ["Superior","Michigan","Huron","Erie","Ontario"] )
10 #set( $color = "blue" )
11 <table>
12     #tablerows( $color $greatlakes )
13 </table>
```

Hieronder de resultaten van macro.vm B.5.

```
$color [template=macro.vm, line=4, column=23] not encoded!
$something [template=macro.vm, line=4, column=30] not encoded!

Total places of output: 2
encoded: 0
```

```
non-encoded: 2
```

Bijlage C

Resultaten Anti-XSS tool

De tool is uitgevoerd op 28 templates van een bestaande webapplicatie. Er is gecontroleerd of de Velocity EscapeTool is toegepast op elke variabele uitvoer. Elke melding die de Anti-XSS gaf wordt getoond met daaropvolgend een aantal regels Velocity-code waar de melding over gaat. Hierna zal aangegeven worden hoe deze plek geëncodeerd zou moeten worden. Voor encodering zal de Velocity EscapeTool aangeraden worden.

C.1 HTML-body

`$!{title} [template=Template-Case_detail, line=12]`

```
12 <h1>$!{title}</h1>
```

De variabele $!{title} wordt in HTML-context gebruikt. Deze dient dan dus als HTML geëncodeerd te worden. Het zou dan moeten worden:

```
12 <h1>$esc.html($!{title})</h1>
```

C.2 HTML-URL

De eerste twee meldingen zijn False Positives. Op regel 10 wordt er namelijk geen uitvoer gedaan, maar wordt de waarde van een reference toegewezen aan een variabele (een assignment). Regel 12 en 13 moeten geëncodeerd worden voor de context HTML-body , middels $esc.html(). $detailLink op regel 14 wordt geplaatst binnen een 'a' tag. Om deze op de juiste manier te encoderen dient er $esc.url() op toegepast te worden.

```
$UtilMethods.encodeURL($!{content.Title})
    [template=Container-Rightside_dyn, line=10, column=59]
$!{content.inode} [template=Container-Rightside_dyn, line=10, column=105]
$!{content.title} [template=Container-Rightside_dyn, line=12, column=31]
$!{content.Intro} [template=Container-Rightside_dyn, line=13, column=8]
$detailLink [template=Container-Rightside_dyn, line=14, column=14]
```

```
## Rightside of the F8 website
#pullContent('+structureInode:21356 +languageId:1* +deleted:false +live
    :true' '100' 'text1')

#if ($list.size()>0)
  #foreach($content in $list)
    #if ($content.isSet($url))
      #set($detailLink = $url)
    #else
      #set($detailLink = "./leesmeer.dot?lees=$UtilMethods.encodeURL($
          !{content.Title})&cid=$!{content.inode}")
    #end
    <div class="h4block"> <h4>$!{content.title}</h4> </div>
    <b>$!{content.Intro}</b><br />
    <a href="$detailLink">Lees meer</a>
  #end
#end
```

C.3 HTML-attribuut

```
$!request.getParameter('vacature')
    [template=Container-Sollicitatie_formulier, line=99, column=149]
```

In dit voorbeeld dient de uitvoer geëncodeerd te zijn voor een HTML-attribuut.

```
<input id="solliciteerNaar" value="$!request.getParameter('vacature')"
    />
```

moet worden:

```
<input id="solliciteerNaar" value="$esc.htmlAttribute($!request.
    getParameter('vacature'))" />
```

Dit voorbeeld is extra interessant omdat je hier kunt zien dat er direct invoer wordt neergezet. `$request.getParameter('vacature')` haalt de waarde van vacature uit de querystring en print dat direct (zonder controle of encodering) in de pagina. Hier was weerspiegelde (reflected) Cross-site scripting mogelijk. Zie paragraaf 3.2.2.

C.4 XML

```
$cnt [template=Template-Cases_XML, line=9, column=23]
$!case.opHomepage [template=Template-Cases_XML, line=9, column=29]
$case.intro [template=Template-Cases_XML, line=9, column=55]
```

Hieronder deel van een Velocity Template dat een XML-bestand genereert.

```
1  <?xml version="1.0" encoding="iso-8859-1"?>
2  <content>
3  #pullContent('+structureInode:22062 +languageId:1* +deleted:false +live
       :true' '100' 'date1')
4  #set($cnt=1)
5  ## - Content
6  <section name="casecontent">
7  #foreach($case in $list)
8    #if($!case.opHomepage == 1)
9      <item name="case_0$cnt">$!case.opHomepage<![CDATA[$case.intro]]></
           item>
10     #set($cnt = $cnt + 1)
11   #end
12 #end
13 </section>
14 ...
```

Het is van belang dat de variabele gegevens in deze Velocity-template als XML worden geëncodeerd middels de $esc.xml() functie. Deze drie meldingen hebben betrekking op regel 9. Regel 9 moet worden:

```
9  <item name="case_0$esc.xml($cnt)">$esc.xml($!case.opHomepage)<![CDATA[
       $esc.xml($case.intro)]]></item>
```

Bijlage D

DotCMS

D.1 Templates en Containers

In Listing D.1 is een template van een homepagina te zien. Op regel 12 is het eerste stukje Velocity-code. Dit is een stukje commentaar. Op regel 13 wordt de container 'Hoofdmenu' (D.2) ingevoegd. Hieronder wordt elke container getoond die in de template van homepagina wordt ingevoegd.

Listing D.1: Homepagina

```
1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3
       .org/TR/xhtml1/DTD/xhtml1-strict.dtd">
2  <html xmlns="http://www.w3.org/1999/xhtml" lang="en" xml:lang="en">
3  <head>
4    <meta http-equiv="Content-Type" content="text/html; charset=iso
         -8859-1" />
5    <title>First8</title>
6    <link href="/global/css/style.css" rel="stylesheet" type="text/css"
         />
7  </head>
8  <body>
9  <div id="container">
10
11   <div id="topmenu">
12     ## Hoofdmenu
13     #parse( $container5676 )
14   </div>
15
16   <div class="colmask">
17     <div class="colmid">
18       <div class="colleft">
```

```
19          <div class="col1wrap">
20            <div class="col1" id="content">
21            <!-- Column 1 start -->
22              ## Webpage Content
23              #parse( $container21261 )
24            <!-- Column 1 end -->
25            </div>
26          </div>
27          <div class="col2" id="column_left">
28
29            <!-- Column 2 start -->
30            <div id="leftmenu">
31              ## Submenu
32              #parse( $container5692 )
33            </div>
34            ## Left Bottom
35            ##parse( $container21469 )
36            <!-- Column 2 end -->
37          </div>
38
39          <div class="col3" id="column_right">
40            <!-- Column 3 start -->
41            ## Rightside
42            #parse( $container21362 )
43            <!-- Column 3 end -->
44          </div>
45
46        </div>
47      </div>
48    </div>
49    ## Footer
50    #parse( $container33131 )
51  </div>
52
53  <div style="clear:both;"></div>
54
55  </body>
56  </html>
```

Listing D.2: Hoofdmenu

```
#navigation(1,1)
```

Aanroep naar de macro 'navigation'.

Listing D.3: Webpage Content

```
<h1>$!{title}</h1>
<p>$!{body}</p>
```

HTML met twee variabelen.

Listing D.4: Submenu

```
#navigation(2,1)
```

Aanroep naar de macro 'navigation'.

Listing D.5: Left Bottom

```
<div id="sitemap_menu">
<a href="/global/docs/AlgemeneVoorwaardenConclusionBV.pdf">Algemene
    voorwaarden</a>
<a href="/home/sitemap.dot">Sitemap</a>
<a href="/home/privacy.dot">Privacy</a>
<a href="/home/disclaimer.dot">Disclaimer</a>
</div>
```

Statische HTML (bevat geen variabelen).

Listing D.6: Right Side

```
<div class="h4block"> <h4>$!{title}</h4> </div>
<div class="content_right">
$!{content}
#if ($UtilMethods.isSet($url))
  <br /><br />
  <a href="$url" target="_blank">Lees meer</a>
#end
</div>
```

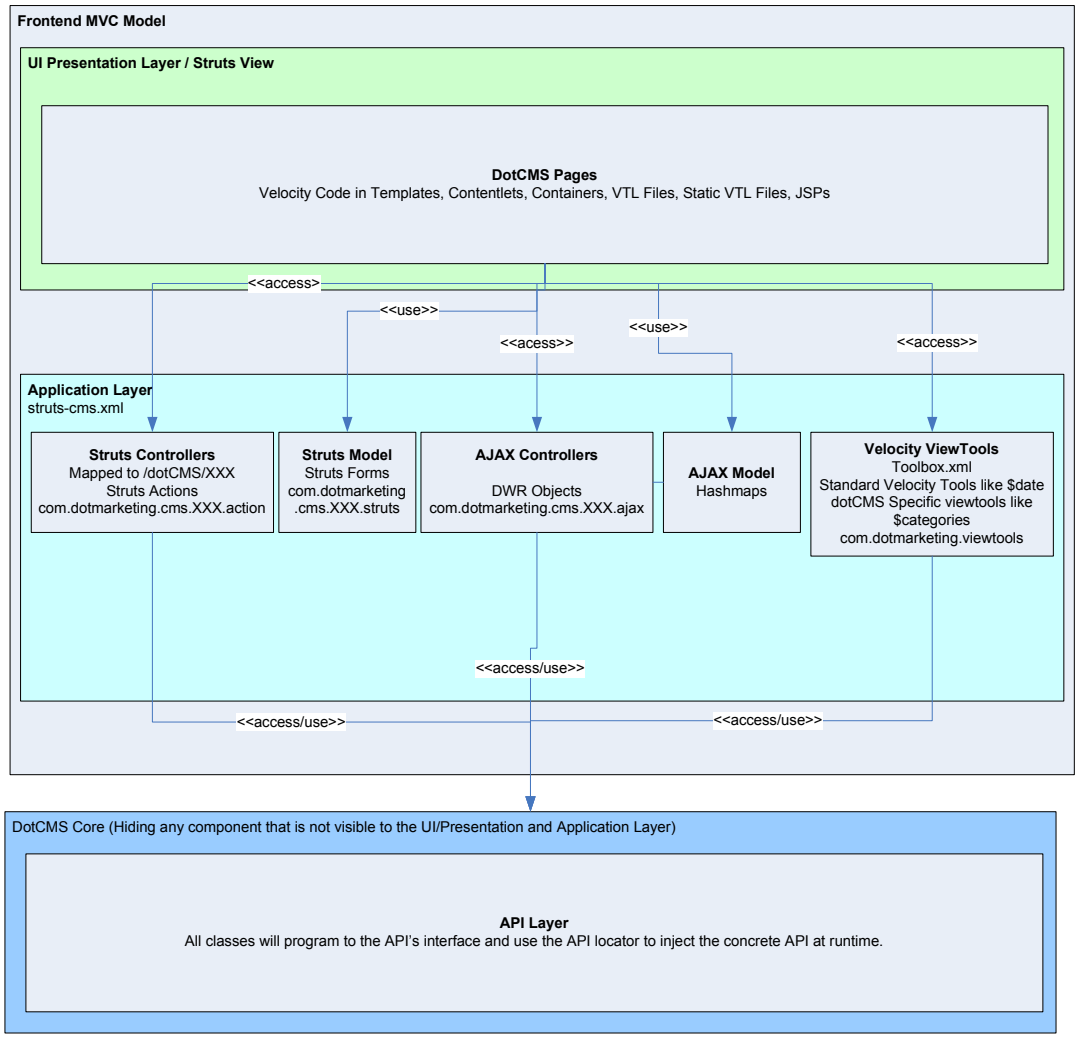
HTML met variabelen.

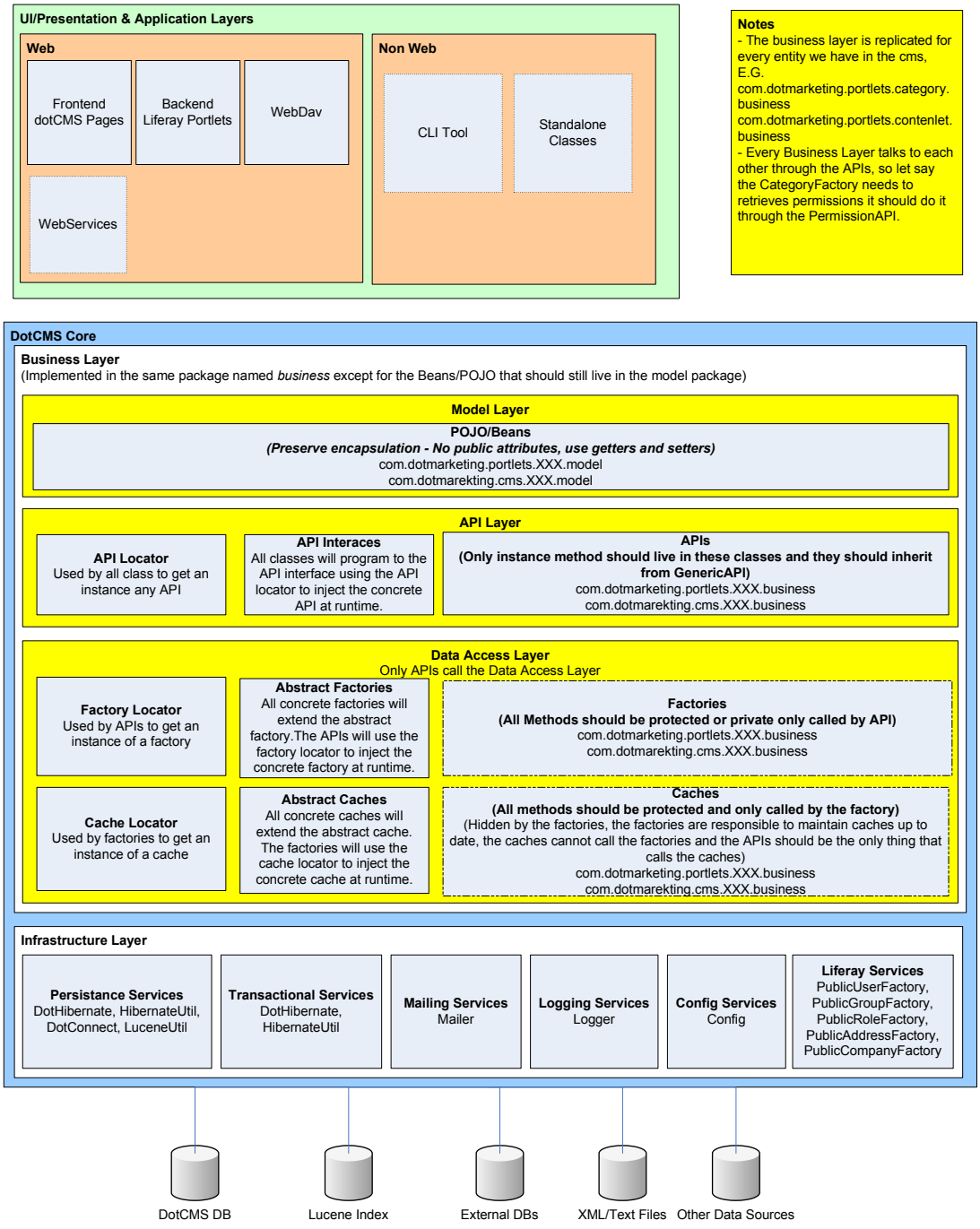
Listing D.7: Footer

```
<div id="footermenu">
  <ul>
    <li>
      <a href="/global/docs/AlgemeneVoorwaarden.pdf">Algemene
          voorwaarden</a>
    </li>
    <li>
      <a href="/home/sitemap.dot">Sitemap</a>
    </li>
    <li>
      <a href="/home/privacy.dot">Privacy</a>
    </li>
    <li>
      <a href="/home/disclaimer.dot">Disclaimer</a>
    </li>
    <li>
      <a href="http://www.conclusion.nl/">
        <img src="/global/images/logo_compact2.png" />
      </a>
    </li>
    <li>
  </ul>
</div>
```

Statische HTML (bevat geen variabelen).

D.2 Architectuur van dotCMS




UI/Presentation & Application Layers
Web
Frontend dotCMS Pages
Backend Liferay Portlets
WebDav
WebServices
Non Web
CLI Tool
Standalone Classes
Notes
- The business layer is replicated for every entity we have in the cms, E.G. com.dotmarketing.portlets.category.business com.dotmarketing.portlets.contenlet.business
- Every Business Layer talks to each other through the APIs, so let say the CategoryFactory needs to retrieves permissions it should do it through the PermissionAPI.
DotCMS Core
Business Layer
(Implemented in the same package named business except for the Beans/POJO that should still live in the model package)
Model Layer
POJO/Beans
(Preserve encapsulation - No public attributes, use getters and setters)
com.dotmarketing.portlets.XXX.model
com.dotmarekting.cms.XXX.model
API Layer
API Locator
Used by all class to get an instance any API
API Interaces
All classes will program to the API interface using the API locator to inject the concrete API at runtime.
APIs
(Only instance method should live in these classes and they should inherit from GenericAPI)
com.dotmarketing.portlets.XXX.business
com.dotmarekting.cms.XXX.business
Data Access Layer
Only APIs call the Data Access Layer
Factory Locator
Used by APIs to get an instance of a factory
Abstract Factories
All concrete factories will extend the abstract factory.The APIs will use the factory locator to inject the concrete factory at runtime.
Factories
(All Methods should be protected or private only called by API)
com.dotmarketing.portlets.XXX.business
com.dotmarekting.cms.XXX.business
Cache Locator
Used by factories to get an instance of a cache
Abstract Caches
All concrete caches will extend the abstract cache. The factories will use the cache locator to inject the concrete cache at runtime.
Caches
(All methods should be protected and only called by the factory)
(Hidden by the factories, the factories are responsible to maintain caches up to date, the caches cannot call the factories and the APIs should be the only thing that calls the caches)
com.dotmarketing.portlets.XXX.business
com.dotmarekting.cms.XXX.business
Infrastructure Layer
Persistance Services
DotHibernate, HibernateUtil, DotConnect, LuceneUtil
Transactional Services
DotHibernate, HibernateUtil
Mailing Services
Mailer
Logging Services
Logger
Config Services
Config
Liferay Services
PublicUserFactory, PublicGroupFactory, PublicRoleFactory, PublicAddressFactory, PublicCompanyFactory
DotCMS DB
Lucene Index
External DBs
XML/Text Files
Other Data Sources