KU LEUVEN

FACULTEIT WETENSCHAPPEN

Threat modeling op basis van een event-georiënteerde DSL

Jules VERBESSEM

Promotor: Prof. dr. ir. W. Joosen
Dept. Computerwetenschappen

Co-promotor: *Prof. dr. ir. K. Yskout*
Dept. Computerwetenschappen

Begeleider: *Dr. ir. L. Sion*
Dept. Computerwetenschappen
Begeleider: *Ir. J. Bellemans*
Dept. Computerwetenschappen
Assessor: *Dr. ir. T. Van hamme*
Dept. Computerwetenschappen

Proefschrift ingediend tot het
behalen van de graad van
Master of Science in Toegepaste Informatica

Academiejaar 2023-2024

© Copyright by KU Leuven

Zonder voorafgaande schriftelijke toestemming van zowel de promotor(en) als de auteur(s) is overnemen, kopiëren, gebruiken of realiseren van deze uitgave of gedeelten ervan verboden. Voor aanvragen tot of informatie i.v.m. het overnemen en/of gebruik en/of realisatie van gedeelten uit deze publicatie, wendt u tot de KU Leuven, Faculteit Wetenschappen, Celestijnenlaan 200H - bus 2100, 3001 Leuven (Heverlee), Telefoon +32 16 32 14 01.

Voorafgaande schriftelijke toestemming van de promotor(en) is eveneens vereist voor het aanwenden van de in dit afstudeerwerk beschreven (originele) methoden, producten, schakelingen en programma's voor industrieel of commercieel nut en voor de inzending van deze publicatie ter deelname aan wetenschappelijke prijzen of wedstrijden.

Inhoudsopgave

Hoofdstuk 1

Introductie

In onze huidige samenleving speelt software een onmisbare rol in vrijwel elk aspect van het dagelijks leven. Van sociale media zoals Facebook, Twitter en Instagram, tot applicaties die essentieel zijn voor basisvoorzieningen zoals drinkwater en elektriciteit. Door het feit dat een groot deel van ons leven steunt op software, brengt het ook een groot risico met zich mee. De impact van een cyberaanval kan in de huidige maatschappij rampzalig zijn. Bijvoorbeeld de *ransomware* cyberaanval op de Colonial gas pijplijn [1]. Bovendien dragen diverse aspecten bij aan het groeiend belang naar software beveiliging. Zoals het feit dat het aantal apparaten, verbonden met het internet, [2] toe neemt. Het toenemende aantal apparaten, verbonden met het internet, resulteert in meer apparaten die potentieel kwetsbaar zijn voor cyberaanvallen. Tevens wordt software complexer, resulterend in meer code [3]. Hierdoor bevinden er potentieel meer kwetsbaarheden in de software [4]. Desbetreffende kwetsbaarheden kunnen gebruikt worden in een cyberaanval. Daarnaast vertoont het aantal gebruikers van applicaties een toenemende trend [5], waardoor er mogelijks meer gegevens gestolen kunnen worden.

Een manier om software veiliger te maken, is door aan threat modeling te doen. Threat modeling [6] is een proces waarbij systematisch bedreigingen, of "threats", geëliciteerd worden. De geïdentificeerde threats worden gedocumenteerd en gemitigeerd. Het eliciteren van de bedreigingen gebeurt geregeld aan de hand van een *data flow diagram (DFD).* Een DFD is een algemeen overzicht van alle belangrijke elementen, en de stroming van hun data, zonder te diepe ingang in de implementatie details. Er bestaan diverse modellering tools om een DFD voor te stellen, zoals: SPARTA[7], OWASP Threat Dragon [8], Microsoft Threat Modeling Tool [9] en nog vele anderen.

Echter blijft het threat modeling proces niet gespaard van enkele uitdagingen. In de modellering fase van het threat modeling proces wordt enkel het eindresultaat vastgelegd, in de vorm van een DFD, en gaat de redenering om tot het eindresultaat te komen, verloren. Dit zorgt voor dubbelzinnigheid en dubbel werk. De probleemstelling wordt beschreven door volgende onderzoeksvraag:

Hoe kan het modelleren tijdens threat modeling worden ondersteund om de design beslissingen en hun bijhorende redenering te documenteren zonder het modelleren te hinderen?

Deze thesis introduceert een event-georiënteerde DSL, genaamd Threat Modeling Language, als een mogelijke oplossing voor dit probleem. Een event-georiënteerde DSL, of *domain specific language*, is een computer taal voor een bepaald domein, waarbij elk DSL-statement geïnterpreteerd moet worden als een event of actie. De reeks DSL-statements resulteren in een grafische representatie van een DFD. Daarnaast laat de taal ook toe om de redenering achter de DSL-statements direct te documenteren en te linken aan desbetreffende statements.

Hoofdstuk 2

Achtergrond

In dit hoofdstuk wordt dieper ingegaan op twee belangrijke concepten: Threat Modeling en Domain Specific Languages (DSL's). Later in de thesis wordt gezien hoe deze twee concepten samenkomen.

2.1 Threat Modeling

Zoals eerder in de inleiding vermeld, neemt de vraag naar software beveiliging toe, waarbij threat modeling een mogelijke oplossing biedt. Threat modeling [6] is een proces waarbij er systematisch bedreigingen, of "threats", geëliciteerd worden. De geïdentificeerde threats worden gedocumenteerd en gemitigeerd. Shostack [10] deelt het proces op in vier vragen:

- Wat wordt er gebouwd?
- Wat kan er fout gaan?
- Hoe kunnen de fouten worden opgelost?
- Is er goed werk geleverd?

Yskout et al. [11] splitsen het threat modeling proces ook op in vier delen. De vier deelprocessen zijn enigszins anders dan de vier vragen van Shostack. Shostack heeft immers een apart deelproces voor de evaluatie. De deelprocessen zijn:

1. Initiëren
2. Modelleren
3. Threats eliciteren
4. Threats mitigeren

In het eerste deelproces, initiëren, maken de deelnemers van de threat modeling sessies kennis met de te modelleren software. Ze krijgen alle nodige informatie over de applicatie. De verschillende deelnemers van een threat modeling sessie kunnen de volgende zijn: project eigenaar, ontwikkelaar, cybersecurity expert, IT infrastructuur specialist, software architect en nog vele anderen.

Zodra alle deelnemers bekend zijn met de applicatie start het tweede deelproces, modelleren. Het software systeem wordt gemodelleerd in een *Data Flow Diagram (DFD)*. Een DFD is een overzicht van alle elementen van de software en de communicaties ertussen. In sectie 2.1.1 wordt er in meer detail gegaan over *Data Flow* Diagrammen.

Niettemin omvat het proces meer dan het construeren van diagrammen. Na het modelleren van het systeem wordt overgegaan tot het eliciteren van bedreigingen. Threats eliciteren is de bedreigingen identificeren door het DFD te observeren. Hiervoor zijn drie strategieën: *asset*-georiënteerd, *attacker*-georiënteerd & software-georiënteerd. Voor de software-georiënteerde strategie zijn er technieken zoals STRIDE. In sectie 2.1.2 worden alle strategieën besproken en in 2.1.3 wordt er dieper in gegaan op STRIDE.

Ten slotte worden de threats gemitigeerd. Mitigeren is het implementeren van tegenmaatregelen om de impact van een aanval te beperken. Om de bedreigingen te kunnen mitigeren, is het noodzakelijk om ze te analyseren en te rangschikken. Rangschikken gebeurt aan de hand van een risicobeoordeling. In sectie 2.1.5, zal hierop terug gekomen worden. De bedreiging met de hoogste risicobeoordeling wordt als eerste gemitigeerd. Er wordt gesproken over bedreigingen *mitigeren* in plaats van *oplossen*, omdat er niet altijd een sluitende oplossing is voor een bedreiging. Ter verduidelijking, aanvallers kunnen andere manieren vinden om het systeem kwetsbaar te maken, waardoor een oplossing mogelijks niet sluitend is. Ter afsluiting moeten de gemitigeerde bedreigingen nog geëvalueerd worden. Omdat de bedreigingen, net zoals het ontwerp van het software systeem zelf, continu veranderen is het belangrijk om tijdig te itereren over het huidige threat model.

Sinds het ontstaan van threat modeling hebben softwareontwikkelaars hulpmiddelen ontwikkeld om het proces te vergemakkelijken. In een latere hoofdstuk 3 worden enkele relevante applicaties en *domain specific languages* beschreven.

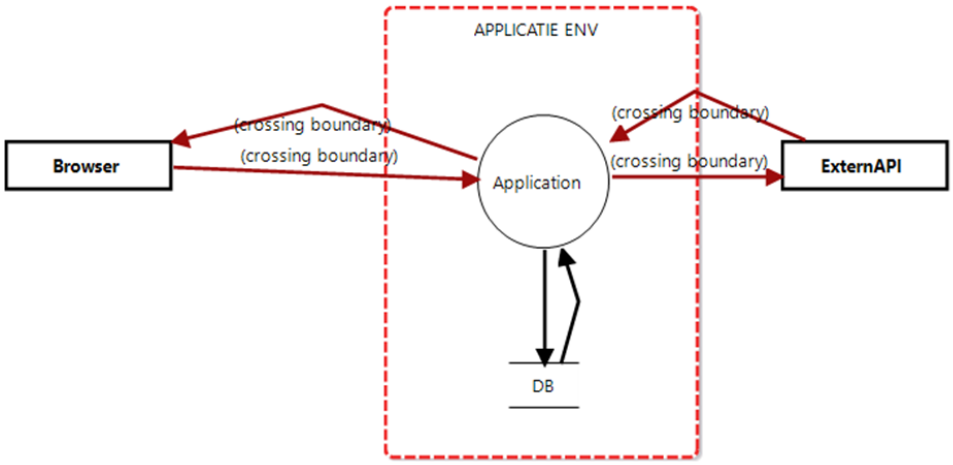
2.1.1 Data Flow Diagram

Het modelleren van de architectuur van het softwaresysteem werd initieel gedaan op een whiteboard. Tegenwoordig zijn er diverse modellering tools zoals: SPARTA[7], OWASP Threat Dragon [8], Microsoft Threat Modeling Tool [9] en nog vele anderen.

Het modelleren van het softwaresysteem wordt gedaan aan de hand van een *Data Flow Diagram (DFD)*. Het is een algemeen overzicht van alle belangrijke elementen zonder te diepe ingang in de implementatie details. Dit maakt het eenvoudiger voor niet-technische profielen om eveneens deel te nemen aan de *threat modeling* sessies.

Het diagram bevat verschillende systeem elementen. Elementen representeren mogelijks entiteiten, waaruit het softwaresysteem is opgebouwd, zoals de interne micro services: processen en *datastores*. Naast een proces of een *datastore*, kunnen elementen ook externe entiteiten zijn. De vernoemde entiteiten communiceren met elkaar. Dit wordt voorgesteld met een *data flow*. Een voorbeeld van een *data flow* is *REST* communicatie over *HTTPS* of *SQL statements* naar de databank. Ten slotte, het laatste element van een DFD, de *trust boundary*. De taak van een *trust boundary* is één of meerdere entiteiten, en eventuele *data flows* tussen de entiteiten, te overkoepelen. Alle entiteiten binnen een *trust boundary* hebben volle vertrouwen in elkaar. Het is een grens die een specifiek deel van de applicatie representeert, geassocieerd met eigen beveiligingsmaatregelen. Ter illustratie, figuur 2.1.

Het diagram is slechts een startpunt en niet de essentie van het *threat* model. Het



Figuur 2.1: DFD van een fictieve applicatie gemaakt in SPARTA.

DFD wordt gebruikt in andere sessies om de bedreigingen te eliciteren en te mitigeren, en dient als een visueel hulpmiddel om te kunnen redeneren over alle componenten van het systeem.

2.1.2 Strategie

Nadat de architectuur van het softwaresysteem gemodelleerd is, wordt er overgegaan naar de interactieve sessies gericht op het eliciteren van potentiële bedreigingen. Er zijn drie diverse strategieën om bedreigingen te eliciteren, die Shostack [10] bespreekt: *asset*-georiënteerd, *attacker*-georiënteerd & software-georiënteerd.

De *asset*-georiënteerde strategie focust zich op het zoeken van de waardevolle bezittingen van het softwaresysteem. Voorbeelden van zulke bezittingen of *assets* zijn data, informatie, computer hardware, de software zelf, elektronische communicatie, etc. Nadat de waardevolle bezittingen geïdentificeerd zijn, worden passende maatregelen geïmplementeerd om de bezittingen te beschermen.

De *attacker*-georiënteerde strategie focust zich op het identificeren van de potentiële aanvaller. Nadat de potentiële aanvallers geïdentificeerd zijn, worden passende maatregelen geïmplementeerd om de aanvallers tegen te gaan. Een *attack tree* is techniek die gebruikt wordt om *attacker*-georiënteerd te werken. Een nadeel van de *attacker*-georiënteerde strategie is de "human factor". De vraag "Wie zou zo iets doen?" kan gesteld worden, waarbij de medemens het voordeel van de twijfel krijgt, volgens Shostack [10]. Dit leidt tot een lagere risicobeoordeling van de bedreigingen.

De software-georiënteerde strategie is de beste en meest gebruikte manier van werken en focust zich op de individuele componenten van het eerder gemodelleerde softwaresysteem. Tijdens de sessie om bedreigingen te gaan eliciteren, worden de individuele componenten beschouwd en verschillende technieken gebruikt zoals: STRIDE, *attack lists*, etc. Ieder component wordt onder de loep genomen, door volgende vragen te stellen:

- Wat is het?
- Wie zou het aanvallen?

- Hoe zou de aanvaller het aanvallen?
- Hoe kan een aanval vermeden worden?

2.1.3 STRIDE

STRIDE is, volgens Shostack [10], de meest gebruikte techniek om bedreigingen te eliciteren, op basis van de individuele DFD elementen. De geïdentificeerde bedreigingen worden gecategoriseerd in één van de zes STRIDE categorieën. Het woord STRIDE is een ezelsbrug voor volgende categorieën:

- Spoofing: Wanneer een persoon of proces zich voordoet als iets of iemand anders.
- Tampering: Wanneer de aanvaller de inhoud van een object verandert.
- Repudiation: Het weten of een gebruiker of proces al dan niet een actie heeft uitgevoerd.
- Information Disclosure: Informatie doorgeven aan een aanvaller.
- Denial of Service: Resources uitputten zodat het systeem crasht en anderen gebruikers het systeem niet kunnen gebruiken.
- Elevation of Privilege: Acties uitvoeren waar de aanvaller normaal geen toegang tot heeft.

Omdat sommige bedreigingen onder meerdere STRIDE categorieën kunnen vallen, is het niet altijd eenvoudig deze eenduidig aan een categorie toe te kennen. Vandaar wordt STRIDE vooral gezien als een methodologie om bedreigingen te eliciteren, niet om ze te categoriseren.

2.1.4 Privacy Threat Modeling

Privacy is een groeiende bezorgdheid voor de eindgebruiker. Bedrijven achter de softwaresystemen vragen steeds meer persoonlijke data aan de gebruiker. Een uitbreiding van threat modeling is rekening houden met de privacy van de eindgebruiker. Shostack [10] bespreekt in zijn boek twee technieken om privacy bedreigingen te modelleren: PIA en LINDDUN. PIA staat voor Privacy Impact Assessments en is een terminologie die gebruikt wordt om te verwijzen naar systematische processen om de privacy impact van een systeem te evalueren. De andere techniek is LINDDUN. De door DistriNet uitgevonden techniek [12] werkt zoals STRIDE. Iedere component wordt beschouwd en er wordt gezocht naar een privacy bedreiging voor de eindgebruiker. LINDDUN is een ezelsbrug voor: *Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, Non-compliance.*

Een uitdaging bij privacy threat modeling is de diverse percepties van privacy van iedere eindgebruiker. Dit maakt het complex om een beleid te veralgemenen.

2.1.5 Risicobeoordeling

Nadat alle potentiële bedreigingen geëliciteerd zijn, is het belangrijk om deze te mitigeren. Echter kan het mitigeren van een bedreiging zeer kostelijk zijn. Bedreigingen worden eerst beoordeelt, zodat de belangrijkste bedreigingen eerst gemitigeerd kunnen worden. De risicobeoordeling is gebaseerd op twee belangrijke factoren: het dreigingsniveau en de waarschijnlijkheid. Als alle bedreigingen een waarde hebben gekregen, worden ze gerangschikt. Er bestaan diverse technieken om aan een risicobeoordeling te doen: risico beoordeling formule (risico * impact), DREAD, etc.

DREAD

DREAD is een methode om bedreigingen te beoordelen op basis van hun potentiële gevaar. DREAD is, zoals beschreven door Obiora Nweke en Wolthusen [13], een ezelsbrug voor:

- Damage: Wat is de financiële of reputationele schade in het geval van een aanval?
- Reproducibility: Hoe makkelijk het is om de aanval uit te voeren?
- Exploitability: Wat is de kans op de aanval?
- Affected users: Hoeveel gebruikersdata data zal onthuld of aangepast worden?
- Discoverability: Wat is de kans dat een aanvaller de bedreiging ontdekt?

Voor een risicobeoordeling met DREAD moet iedere categorie een waarde krijgen en het gemiddelde genomen van de waarden. De schaal waarin deze risicowaarden bepaald worden, kan vrij gekozen worden door de gebruiker van de techniek.

2.2 Domain Specific Language

In informatica wordt er gewerkt met talen om te communiceren met computers. Over het algemeen, zijn het *General-Purpose Languages* (GPLs), ontworpen om diverse computer problemen op te lossen. Java is een voorbeeld van een GPL. Echter hebben GPL's een probleem. Alves [14] vergelijkt GPLs met algemene dokters. Algemene dokters doen de eerste diagnose van een probleem/ziekte, maar er is nood aan een specialist voor de volledige verdiepende diagnose. GPLs zijn niet gemaakt om verdiepende problemen op te lossen. Hiervoor zijn *Domain Specific Languages* (DSLs) ontworpen. DSL's zijn computertalen gespecialiseerd in één bepaald toepassing domein. DSL's kunnen tekstueel of grafisch zijn. Een voorbeeld van een grafische DSL is de modellering taal UML. Alves [14] deelt DSL's op in twee categorieën:

- Externe DSL: DSL's met een eigen interpreteerder of compiler. Voorbeelden van externe DSL's zijn: SQL, CSS, etc.
- Interne DSL: DSL's die zijn geïmplementeerd in een GPL. Voorbeelden van interne DSL's zijn: Mockito testraamwerk voor Java, JSX een syntaxisextensie voor JavaScript, etc.

Hoofdstuk 3

Gerelateerd Werk

Dit hoofdstuk gaat dieper in op enkele concepten gerelateerd rond het domein threat modeling, zoals *Architectural decision records (ADRs)*, applicaties en DSL's. Ten eerste wordt het concept ADR uitgelegd. Vervolgens worden enkele applicaties voor threat modeling besproken. In een latere sectie wordt reeds bestaande DSL's besproken. Ten slotte wordt dit hoofdstuk afgerond met een overzicht van alle besproken applicaties en DSL's, en hun functionaliteit, aan de hand van een tabel.

3.1 Architectural Decision Records

Binnen het softwareontwikkeling proces bestaat er een techniek om architecturale beslissingen te documenteren, genaamd *architectural decision records (ADRs)*. ADR's zijn korte beschrijvingen, volgens Ahmeti et al. [15] meestal beschreven in *Markdown* documenten, van de redenering achter architecturale beslissingen. De genomen beslissingen, rond *security by design*, zijn een sub set van de beslissingen tijdens het software ontwerp. Daarom gelden de positieve effecten van ADR's op het software ontwikkeling proces, ook voor de documentatie van het threat model. ADR's zijn goed te analyseren omdat ze kort en rechtuit zijn.

3.2 Applicaties

Om het threat modeling proces te vergemakkelijken zijn er applicaties geconstrueerd. De threat modellering applicaties die besproken worden zijn: OWASP Threat Dragon, Microsoft's Threat Modeling Tool, SPARTA, ThreatSpec en Pytm. Er bestaan ook commerciële applicaties zoals IriusRisk & SD Elements by Security Compass, echter wordt er in de thesis niet dieper op ingegaan. Niettemin zijn er talloze alternatieve applicaties.

3.2.1 OWASP Threat Dragon

OWASP Threat Dragon [8] is een open-source desktop applicatie voor zowel Windows, Linux en Mac. Verder is er ook een OWASP Threat Dragon webapplicatie. In de applicatie kan er een DFD gemaakt worden. Nadien is het mogelijk om specifieke bedreigingen te koppelen aan de elementen van het DFD. Vervolgens is het mogelijk om de (privacy) bedreigingen te categoriseren in de STRIDE (of LINDDUN) categorieën. Ten slotte kan

het DFD geëxporteerd worden naar een JSON bestand, wat het makkelijk maakt om de modellen te delen.

3.2.2 Microsoft's Threat Modeling Tool

Microsoft's Threat Modeling Tool (MS's TMT) [9] is een desktop applicatie voor Windows. Het werd ontwikkeld om gebruikt te worden tijdens de *Microsoft Security Development Lifecycle (SDL)*. Zoals Threat Dragon kan er in Microsoft's Threat Modeling Tool een DFD gemaakt worden om vervolgens aan de elementen bedreigingen te koppelen. Erbovenop komt de functionaliteit om automatisch bedreigingen te genereren. Tegenover Threat Dragon is het in Microsoft's Threat Modeling Tool enkel mogelijk om de threats te categoriseren, in de door Microsoft ontwikkelde techniek, STRIDE. Vervolgens is het ook mogelijk om het DFD te exporteren, in XML formaat. Ten slotte kan er een overzicht gegenereerd worden van alle bedreigingen, in de vorm van een overzichtelijk rapport.

3.2.3 SPARTA

SPARTA [7] is een desktop applicatie, gemaakt door KU Leuven, voor Windows, Linux en Mac. SPARTA staat voor *Structured Process for Application Risk Testing and Analysis*. Het acroniem reflecteert de kern functionaliteiten van de applicatie. Ook in SPARTA is het mogelijk om een DFD te maken. Opnieuw kunnen aan de elementen van het DFD bedreigingen gekoppeld worden, door ze automatisch te genereren. De elementen worden verrijkt met numerieke waarden, die dienen als input voor de geautomatiseerde risico analyse. Vervolgens is het mogelijk om in SPARTA tegenmaatregelen voor te stellen. Deze verminderen het risico op bepaalde bedreigingen, wat gereflecteerd wordt in de automatisch risico analyse. Daarbovenop is het mogelijk om in SPARTA bedreigingen te categoriseren in zowel STRIDE en, het door KU Leuven gemaakte, LINDDUN. Ten slotte biedt SPARTA een Java bibliotheek aan om aparte applicaties of extensies voor SPARTA te maken.

3.2.4 ThreatSpec

ThreatSpec [16] is een open-source applicatie, ontworpen om het threat modeling proces te versnellen, door bedreigingen te identificeren in code, aan de hand van annotaties. Eveneens kunnen tegenmaatregelen in de code voorgesteld worden door een annotatie. De annotatie gebeurt in de commentaar van de code aan de hand van het "@" symbool. Door middel van de annotaties wordt een DFD en een rapport gegenereerd.

3.2.5 Pytm

Pytm [17] is een Python *framework* voor threat modeling. Door het systeem te specificeren in een Python bestand aan de hand van Pytm *statements*, kan via de terminal een DFD, sequentie diagram en een rapport gegenereerd worden. Het rapport is een leesbaar *Markdown* bestand met de specificaties, gespecificeerd in het Python bestand.

3.3 Bestaande DSL's

In sectie 2.2 werd het verschil tussen GPL's en DSL's reeds aangehaald. Ook voor Threat Modeling bestaan er DSL's. Tijdens de literatuurstudie rond DSL's voor threat modeling, *cybersecurity* en DFD's kwam het volgende aan bod: MAL, CoreLang, STRIDELang, PowerLang, SCL-Lang, ICSLang, VehicleLang, ThreatGet's DSL, Pimac DSL & UMLsec. Hoe dan ook kunnen er talloze alternatieven zijn. In deze sectie worden de DSL's verder besproken.

3.3.1 MAL

Meta Attack Language (MAL) is een *meta* taal. Een *meta* taal is een taal met als doel het beschrijven of manipuleren van andere talen. De syntax, semantiek en structuur van een taal worden beschreven door een *meta* taal.

MAL is een *meta* taal ontwikkeld voor het construeren van andere DSL's in het domein van aanvalsimulaties. MAL helpt bij het maken van DSL's die automatisch *attack graphs* instanties genereren vanuit een systeemarchitectuur model. *Attack graphs* illustreren hoe een aanvaller door verschillende handelingen te verrichten zich binnen een systeem kan verplaatsen. Het is kostelijk om een *attack graph* te maken omdat ze niet hergebruikt kunnen worden. Vervolgens zijn *attack graphs* heel gedetailleerd en uitsluitend gemaakt door cybersecurity experts.

Volgens Wideł W. et al [18] kan MAL beschouwd worden als een combinatie van UML klasse- en object diagrammen en probabilistische *attack trees*. Unified Modeling Language (UML) is een grafische DSL om het design van een applicatie voor te stellen op een objectgeoriënteerde manier. *Attack trees* zijn diagrammen in een boomstructuur, die visualiseren hoe een software bezitting mogelijks wordt aangevallen. *Attack trees* kunnen gebruik maken van logica. Zo zijn er twee soorten nodes. Een änd-node is een node waarbij alle subnodes voldaan moeten zijn. Er is ook een ör-node. Dit is een node waarbij maar één subnode voldaan moet zijn.

De bezittingen, of *assets* van een software worden voorgesteld als UML klassen, teven eens krijgen de *assets* associaties met elkaar. De associaties hebben rollen en multipliciteiten zoals bij UML. De stappen van de aanval zijn gelinkt aan een *asset*. Zoals bij *attack trees* kunnen de stappen van de *attack graph* *AND*-stappen of *OR*-stappen zijn. Ten slotte kunnen in MAL ook tegenmaatregelen voorgesteld en gekoppeld worden met een aanval stap. Een *attack graph* kan gezien worden als een collectie van gebundelde *attack trees*.

Cybersecurity experts gebruiken *attack graphs* om de mogelijke stappen van aanvallers in het systeem te modelleren. Het aantal stappen zijn de lengte van een pad. Aanvallers kiezen vaak het kortste pad. De *attack graph* kan uitbreid worden door numerieke waarde aan de stappen toe te voegen, zodat er een kortste-pad-algoritme kan gebruikt worden om te voorspellen welke paden de aanvallers gaan nemen.

3.3.2 MAL gebaseerde DSL's

Aan de hand van de meta taal MAL werden verschillende DSL gemaakt. Zoals de hieronder besproken DSL's: CoreLang, STRIDELang, PowerLang, SCL-Lang, icsLang en vehicleLang.

CoreLang

CoreLang [19] is een tekstuele DSL ontwikkeld met MAL, gebruikt om aanval scenario's op een softwaresysteem te analyseren met behulp van *attack graphs*. CoreLang bevat alle algemene concepten, of de *core*, van IT infrastructuur om *attack graphs* te kunnen opstellen, op een abstract niveau. Een reeks van CoreLang-stellingen produceren een *attack graph*. De stellingen of *statements* declareren de bezittingen of *assets* en hun associaties. Aan de *assets* kunnen aanval stappen gekoppeld worden.

STRIDELang

STRIDELang [20] is een tekstuele DSL die gebaseerd werd op CoreLang. Het voegt de concepten, STRIDE en DREAD, toe aan CoreLang. In STRIDELang zijn er zes bezittingen die staan voor iedere categorie van STRIDE. De aanval stappen die gekoppeld worden aan de bezittingen krijgen vervolgens een DREAD waarde.

PowerLang

PowerLang [21] is een DSL ontwikkeld met MAL, gebruikt om aanval scenario's op energie-gerelateerde IT en OT infrastructuren te analyseren met behulp van *attack graphs*. PowerLang bevat alle concepten van energie-gerelateerde IT en OT infrastructuren om *attack graphs* te kunnen opstellen.

SCL-Lang

SCL-Lang [22] is een DSL ontwikkeld met MAL, gebruikt om aanval scenario's te analyseren met behulp van *attack graphs*. SCL-Lang bevat alle concepten rond systeem configuratie voor elektronische onderstation, zoals controle- en monitoring logica, om *attack graphs* te kunnen opstellen.

icsLang

Hacks en Katsikeas [23] bespreken ICSLang, een DSL ontwikkeld met MAL gebruikt om aanval scenario's te analyseren met behulp van *attack graphs*. ICSLang bevat alle concepten rond industriële controlesystemen, ICS is een acroniem voor *Industrial Control Systems*, om *attack graphs* te kunnen opstellen.

VehicleLang

VehicleLang[24] is een DSL ontwikkeld met MAL, gebruikt om aanval scenario's te analyseren met behulp van *attack graphs*. VehicleLang bevat alle concepten rond de moderne automotive IT infrastructuur om *attack graphs* te kunnen opstellen.

3.3.3 ThreatGet's DSL

Ebrahimi et al [25] gebruiken ThreatGet's DSL om anti-patronen te specificeren. Een anti-patroon is een software ontwerp patroon waarvan bewezen is dat het een enkele nadelen heeft, zoals het *singleton* patroon. Een anti-patroon in software beveiliging kan leiden tot

kwetsbaarheid en bedreiging veroorzaken. De syntax van ThreatGet's DSL bestaan uit patronen en filters om de elementen van het DFD te beschrijven.

3.3.4 Pimac DSL

Pimac DSL [26] representeert de systeemarchitectuur op een manier waarbij de *attack surface* wordt benadrukt, door de interfaces van het systeem expliciet te modelleren. Met Pimac DSL is het mogelijk om analyses uit te voeren op: *attack surface*, *attack impact* en *attack routing*. Een systeemarchitectuur bestaat uit elementen die onderling afhankelijk zijn omdat ze met elkaar interactie hebben.

3.3.5 UMLsec

UMLsec [27] is een extensie op de DSL: UML. Informatie relevant voor de beveiliging van de applicatie wordt geannoteerd aan de elementen van het UML diagram. Annotaties gebeuren via de bestaande UML notatie, bijvoorbeeld: <<critical>>. UMLsec laat toe om tijdens het modelleren van de applicatie rekening te houden met de beveiliging van de software, en de data die de software bewerkt. Het is verstandig om vroeg in het ontwikkelingsproces rekening te houden met beveiliging. Door vroeg fouten rond beveiliging op te sporen, zorgt het voor minder kwetsbaarheden in het eind systeem en uiteindelijk minder risico op cyberaanvallen. Bijkomend wordt tijd bespaard, omdat het oplossen van de kwetsbaarheden langer duurt dan het voorkomen. Bovendien is besparen op tijd ook besparen op geld.

3.4 Overzicht

Met de verduidelijking van het gerelateerd werk in gedachten worden de functionaliteiten samengevat in onderstaande tabel. De functionaliteiten worden onderverdeeld in twee categorieën: de basis functionaliteiten rond threat modeling en de functionaliteiten rond het documenteren en bewaren van beslissingen. Merk op dat gerelateerd werk die de functionaliteit ondersteunt een volledig bolletje krijgt, diegene die het niet ondersteunt niet. Bovendien zijn er ook tools met halve bolletjes, deze ondersteunen de functionaliteit maar deels.

3.4.1 Basis functionaliteiten

De basis functionaliteiten van de threat modeling applicaties en DSL's zijn: het creëren van een DFD, threats koppelen aan DFD elementen, risico analyse, werken met STRIDE, werken met LINDDUN, rapportering en mogelijks nog diverse. Hieronder wordt de basis functionaliteiten gekoppeld aan de gerelateerde werken. Gezien de essentie van de basis functionaliteiten verdienen ze een plaats in de tabel.

Ten eerste zullen alle besproken applicaties en UMLSec een vorm van het bewerken van het DFD ondersteunen, al dan niet via *drag-n-drop* of het laten genereren op basis van specificaties.

Ten tweede ondersteunen de applicaties ook bijkomende functionaliteit om bedreigingen te linken aan de elementen van het DFD, echter kunnen enkel MS's TMT en SPARTA automatisch threats genereren.

Ten derde ondersteunen de applicaties ook allemaal een vorm van risico analyse, waarbij threats een prioriteit krijgen. Niettemin zal enkel SPARTA werken met een meer geavanceerde manier, gebaseerd op numerieke waarde.

Ten vierde zullen de technieken STRIDE en LINDDUN door zowel Threat Dragon en SPARTA ondersteund worden. Echter ondersteunt MS's TMT enkel STRIDE.

En tot slot wordt de functionaliteit om tekstuele rapporten te genereren, gebaseerd op het DFD, ondersteunt door MS's TMT, ThreatSpec en Pytm.

3.4.2 Documentatie functionaliteiten

De essentie van documenteren wordt besproken in hoofdstuk 4, hieronder is reeds een oplijsting van de vormen van documentatie die de gerelateerde werken al dan niet ondersteunen:

Ten eerste ondersteunen alle besproken applicaties en UMLsec, de mogelijkheid om meta elementen toe te voegen aan het DFD. Een meta element is een grafisch element waar een beschrijving aan toegevoegd kan worden zodat het meer informatie geeft aan één of meerdere DFD elementen. Meta elementen hebben de vorm van een *sticky note* of dergelijke.

Ten tweede ondersteunen de benoemde applicaties een vorm van tekstuele beschrijving per element van het DFD.

Ten derde is er ook de mogelijkheid om een mitigatie toe te voegen aan een bedreiging. Een mitigatie beschrijft tegenmaatregelen voor een bedreiging. De tekstuele manier van Threat Dragon om mitigaties toe te voegen aan bedreigingen is beperkt. Nochtans heeft SPARTA een uitgebreide vorm van mitigaties toevoegen aan de bedreigingen en elementen. Waarbij een mitigatie een nieuw element representeert met zijn eigen attributen.

Ten vierde ondersteunt geen enkel gerelateerd werk de mogelijkheid om de acties op het DFD te bewaren als een sequentiële lijst die de geschiedenis van het DFD representeert. De acties op het DFD zijn bijvoorbeeld het toevoegen van een proces of het wijzigen van een *flow*, etc. De acties op het DFD die de geschiedenis representeren, zouden vervolgens geanalyseerd kunnen worden. Het analyseren van de stappen die tot het eindresultaat leiden, kan waardevol inzicht geven.

Ten slotte bestaat er geen enkel gerelateerd werk dat de mogelijkheid ondersteund om de rationaal achter de acties te documenteren. De documentatie zou dan gelinkt worden met de actie. Het documenteren gebeurt *inline* met het modelleren. Waarbij iedere actie een verklaring heeft van reden achter de genomen actie. Het bewaren van de redenering achter de beslissing moet op een efficiënte manier gebeuren. De efficiënte manier impliceert dat het documenteren tegelijk gebeurt met het modelleren, zodat er na het modelleren niet gedocumenteerd moet worden, en er extra tijd verloren gaat.

Gerelateerd werk en hun functionaliteit										
	Basis functionaliteiten					Documentatie functionaliteiten				
Applicaties & DSL's	DFD creatie	Auto-genereren threats	Risico analyse	STRIDE & LINDDUN	Rapport	Meta elementen	Documentatie element	Mitigatie	Geschiedenis	Rationaal efficiënt documenteren
Threat Dragon	●	◐	◐	●		●	●	◐		
MS's TMT	●	●	◐	◐	●	●	●			
SPARTA	●	●	●	●		●	●	●		
ThreatSpec	●				●		●			
Pytm	●				●		●			
MAL										
CoreLang										
STRIDELang										
UmlSec	●					●				

Hoofdstuk 4

Probleemstelling

Zoals voorheen aangegeven is de beveiliging van softwaresystemen van groeiend belang. De rol van threat modeling is cruciaal voor de groeiende vraag naar *cybersecurity*. Echter blijft het threat modeling proces niet gespaard van enkele uitdagingen.

Zoals uitgelegd in hoofdstuk 2 wordt het threat modeling proces opgedeeld in vier fases. Één fase is de modellering fase, tijdens deze fase wordt er geredeneerd over de ontwikkeling van het *data flow diagram*, gebaseerd op de architectuur van de betreffende software. Het toevoegen van processen en het aanpassen van *data flows*, etc, zijn enkele voorbeelden van acties. Deze acties zetten de redenering om in een eindresultaat.

In het huidige threat modeling proces wordt enkel het eindresultaat vastgelegd, in de vorm van een DFD, en gaat de redenering, om tot het eindresultaat te komen, verloren. Het resultaat van de beslissingen tijdens het modelleren heeft aanzienlijke gevolgen voor de beveiliging van de software. Documenteren van de beslissingen is dan ook belangrijk voor de beveiliging van de software en een algemeen essentieel belang volgens Stettina en Heijstek [28].

In sectie 3.1 werd de techniek *architectural decision records (ADRs)* besproken. ADR's hebben echter nog een tekortkoming die aangepakt moet worden vooraleer ze in het threat modeling proces gebruikt kunnen worden. Momenteel is het noodzakelijk om na de modellering sessies tijd vrij te maken om de ADR's uit te schrijven. De ADR's opstellen tijdens de modellering sessie zorgt voor extra onnodig werk omdat het design nog kan wijzigen. Na de sessie is het design vastgelegd en wordt de redenering erachter gedocumenteerd in een ADR. Bijkomend vragen ADR's ook veel onderhoud, omdat door iedere design wijziging het noodzakelijk is om de overeenkomstige ADR manueel aan te passen. Dit verhindert het modellering proces. Ter conclusie bieden ADR's een gedeeltelijke oplossing voor en zijn niet optimaal voor het threat modeling proces.

Sectie 3.4 omvatte enkele bestaande threat modellering applicaties en DSL's en hun functionaliteiten besproken. Er was ondersteuning voor documentatie op basis van meta elementen, beschrijvingen van elementen en beschrijvingen van mitigaties. Maar er bleek geen enkel van deze werken ondersteuning te bieden om de geschiedenis van acties op het DFD te bewaren. Teven eens is er geen efficiënte manier om de redenering achter de beslissingen van de genomen acties te bewaren. Het gebrek aan ondersteuning demonstreert dat dit probleem in de praktijk nog niet is opgelost.

Ten slotte beschrijft Yskout et al. [11] het volwassenheidsniveau van het threat modeling proces als een reeks niveaus. Een uitdaging die het threat modeling proces heeft om het volwassenheidsniveau te kunnen verhogen naar het volgende niveau, is het ontwikkelen

van een techniek die de aanpassingen aan het model zou traceren en documenteren. Dit zou op een makkelijke manier beslissingen bewaren die zijn gemaakt zonder het modelleren te beperken. De techniek bespreekt een mogelijke oplossing voor de probleemstelling, echter is het gebrek aan de besproken techniek een bekrachtiging van het feit dat het gebrek aan documentatie een probleem is.

Het probleem van de huidige staat van het threat modeling proces, dat de redenering achter design beslissingen verloren gaat, veroorzaakt 2 subproblemen: dubbelzinnigheid en dubbelwerk.

Het gebrek aan documentatie van de redenering achter design keuzes kan leiden tot dubbelzinnigheid rond eerder genomen beslissingen, volgens Ahmeti et al. [15]. Na verloop van tijd wordt het moeilijker om de originele redenering achter design keuzes te herinneren. De beslissingen die tijdens het modelleren werden genomen worden nu niet gedocumenteerd. De deelnemers van de sessies moeten, in de huidige situatie, onthouden waarom ze de beslissingen hebben genomen. Het menselijk geheugen kan moeilijk nauwkeurig beslissingen bewaren en daarom vervaagt de herinnering na verloop van tijd, volgens Ahmeti et al. [15]. Daarnaast is een team van threat modelleerders niet in steen gezet. Er kunnen altijd mensen vertrekken of bijkomen. Als een ervaren modelleerder het team verlaat, neemt hij/zij ook zijn/haar kennis mee. Als die kennis rond de redenering achter de design beslissingen niet gedocumenteerd is, verdwijnt de belangrijke informatie samen met de teamgenoot. Bijgevolg wanneer er een nieuwe teamgenoot zou bijkomen bemoeilijkt het gebrek aan documentatie het *onboarding* proces. Het begrijpen van de context van de eerder genomen beslissingen, wordt belemmert zonder documentatie. Het begrip van de context is nodig om sneller actief deel te nemen aan de sessies.

Als bijkomend resultaat van het gebrek aan documentatie, kan het leiden tot dubbelwerk. Dubbelwerk is het feit dat dezelfde concepten herhaaldelijk worden onderzocht. Dit veroorzaakt tijdsverlies, wat op zijn beurt leidt tot financieel verlies.

De probleemstelling van deze thesis wordt beschreven door volgende onderzoeksvraag:

Hoe kan het modelleren tijdens threat modeling worden ondersteund om de design beslissingen en hun bijhorende redenering te documenteren zonder het modelleren te hinderen?

Hoofdstuk 5

Methodologie

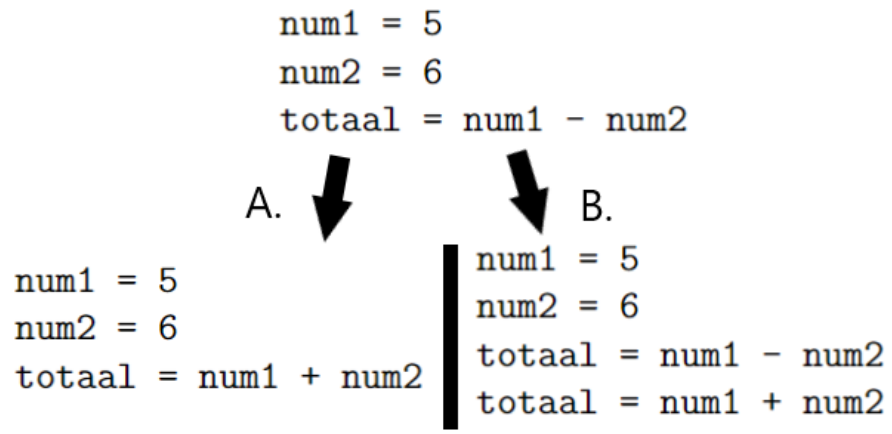
Met inzicht in het besproken probleem van het threat modeling proces, verdiept de thesis in het onderzoeken van een oplossing. Een event-georiënteerde DSL, zoals de titel van de thesis stelt, kan een mogelijke oplossing bieden. Hieronder wordt er dieper ingegaan op de definitie van een event-georiënteerde DSL en vanwaar de inspiratie komt. Vervolgens worden de doelstellingen voor een dergelijke DSL besproken, samen met richtlijnen voor de syntax. Nadien worden de operaties van de DSL uitgelegd, samen met de achterliggende acties.

5.1 Event-georiënteerde DSL

Een bekrachtiging van de probleemstelling, was het gebrek aan een techniek die het volwassenheidsniveau van het threat modeling proces naar het volgende niveau kon tillen. Deze techniek zou de aanpassingen aan het model moeten traceren, om zo beslissingen te documenteren, zonder het modelleren te beperken. Yskout et al. [11] benoemen de methode, die gebruikt maakt van de techniek, de "model-as-you-go" methode. Ze suggereerden dat de methode zou werken met een DSL. De techniek omvat een directe oplossing van de probleemstelling. Dat impliceert dat werken met een DSL op een "model-as-you-go" methode ook een directe oplossing is.

De inspiratie voor de oplossing kwam deels uit het *event sourcing* patroon. Dit is een design patroon waarbij elke wijziging aan een object van een softwaresysteem wordt vastgelegd als een *event* en bewaard blijft in een centrale *tracking service*. De *events* identificeren de acties die gebeurd zijn op de objecten, zodat alle veranderingen aan het softwaresysteem bijgehouden worden. Bovendien moeten de *events immutable* zijn. Dit impliceert dat, na de creatie van het *event*, het *event* onveranderlijk blijft. Aan de hand van de bijgehouden *events* kunnen er, volgens Fowler [29], verscheidene acties genomen worden. Één mogelijke actie is het *backtracken* van de *events* om het traject naar het eindresultaat te reconstrueren en analyseren. Deze actie beschrijft de missende functionaliteit van de besproken gerelateerde werken. De *events* komen overeen met de DSL-statements en de status van het softwaresysteem is de status van het DFD.

In het deelproces van threat modeling, het modelleren, worden applicaties gebruikt, om een DFD te creëren. Zoals eerder in de probleemstelling vermeld, worden de genomen stappen om het eindresultaat te bereiken, niet gedocumenteerd in de applicaties. De oplossing van de probleemstelling, werken met de event-georiënteerde DSL, zal de stappen wel bewaren. Deze stappen worden in de event-georiënteerde DSL gezien als



Figuur 5.1: Voorbeeld verschil tussen werken met een normale DSL en een event-georiënteerde DSL

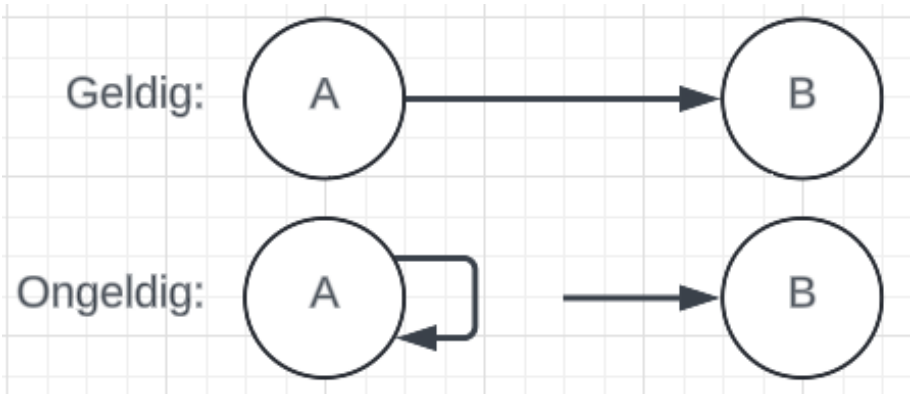
DSL-statements. Ieder statement wijzigt minstens één element van het DFD. Dit zorgt ervoor dat de opeenvolging van statements resulteert in het gewenste DFD. Het verschil tussen een normale en een event-georiënteerde DSL wordt voorgesteld aan de hand van volgend scenario waarbij een DSL gebruikt wordt om twee nummers met elkaar op te tellen. Maar de gebruiker van de DSL typt onopzettelijk een "-" teken, in plaats van een "+" teken. De manier om de fout te corrigeren illustreert het verschil tussen een normale en event-georiënteerde DSL. Bij een normale DSL zal de gebruiker zijn fout willen corrigeren door het derde statement aan te passen, zodat het programma er als volgt uit ziet, zie pijl A van figuur 5.1. In tegenstelling tot de werking van normale DSL's is de oplossing van de probleemstelling een event-georiënteerde DSL. Bij een event-georiënteerde DSL is iedere stap/*event* een aparte statement en zal de gebruiker de fout niet corrigeren door het derde statement aan te passen, maar door een nieuw statement te schrijven. Zodat het programma er als volgt uit ziet: zie pijl B van figuur 5.1. Er wordt een statement geschreven om een bepaalde actie uit te voeren. Om erna een nieuw statement te schrijven dat de actie ongedaan maakt. De events zorgen voor de documentatie van het pad naar het eindresultaat. Dit is de reden dat een event-georiënteerde DSL een oplossing biedt.

5.2 Doelstellingen & richtlijnen

Vooraleer de oplossing, een event-georiënteerde DSL, gebouwd kan worden, is het noodzakelijk om de doelstellingen op te stellen, die vervolgens geëvalueerd kunnen worden. Deze doelstellingen hebben betrekking op functionaliteit, geldigheid, documentatie van rationale en operationeelheid. Eveneens moeten er ook richtlijnen zijn voor de syntax van de DSL, zodat het gebruiks- en leervriendelijk is.

5.2.1 Functionaliteit

Ten eerste moet de DSL ontworpen zijn om alle essentiële functionaliteiten aan te bieden die nodig zijn voor het maken van een DFD. Eens het DFD opgesteld is, moeten er DSL-statements zijn die de mogelijkheid bieden om alle aspecten van het DFD te wijzigen.



Figuur 5.2: Voorbeeld verschil tussen geldig en ongeldig DFD

Stel dat de DSL niet de minimale functionaliteiten ondersteunt om een DFD te maken, en erna te bewerken, dan zou het niet mogelijk zijn om het gewenste DFD te maken.

5.2.2 Geldigheid

Ten tweede moet elke willekeurige reeks DSL-statements resulteren in een geldig DFD. Een DFD is ongeldig bij één van volgende twee situaties. De eerste situatie betreft een *flow* zonder bron of bestemming. Ter illustratie, als er twee entiteiten zijn: A en B. Bijhorend is er een *flow* tussen de twee entiteiten van A naar B, genaamd F. Stel dat entiteit A verwijderd wordt, resulteert dit in *flow* F zonder bron van waar het vertrekt, wat leidt tot in een ongeldig DFD. De tweede situaties betreft een *flow* die vertrekt van een entiteit en eindigt in dezelfde entiteit, dan leidt dit ook tot een ongeldig DFD. Anders gezegd, een *flow* met een gelijke bron en bestemming. Zie figuur 5.2 voor verduidelijking.

Door te werken met een reeks DSL-statements die resulteren in een geldige DFD heeft de eindgebruiker minder flexibiliteit. Dit minimaliseert fouten omdat eindgebruikers gedwongen zijn om te werken met de vaste en beveiligde statements die aangeboden zijn. Door het vermijden van fouten wordt er minder tijd besteed aan foutafhandeling. Minder fouten detecteren en corrigeren, verhoogt de productiviteit. Daarnaast zorgt minder flexibiliteit ook voor betere bruikbaarheid en leerbaarheid, omdat gebruikers alleen te maken krijgen met de relevante en correcte statements. Ten slotte is het een event-georiënteerde DSL, waarbij ieder statement een *events toestaan die* is. *Events* toestaan die geen geldig DFD maken zou contra intuïtief zijn.

5.2.3 Documentatie rationale

Ten derde is de essentie van de DSL om te documenteren, daarom moeten de DSL-statements de redenering achter de beslissingen documenteren. Om deze doelstelling te voldoen moet de gebruiker bij ieder DSL-statement de mogelijkheid krijgen om de rationale documentatie te linken aan het statement. Het kan zijn dat de uitvoering van een bepaalde beslissing in de DSL leidt tot een reeks DSL-statements. Dan moet de DSL de mogelijkheid aanbieden om desbetreffende statements te groeperen en de documentatie aan de groepering te linken.

5.2.4 Operationeelheid

Ten slotte mag het werken met de DSL het modelleren niet verhinderen. Ter verduidelijking: het gebruik van de DSL moet gebeuren tijdens het modelleren, zodat er nadien geen tijd verloren gaat aan het documenteren. De reden voor de operationele doelstelling is voor dezelfde redenen waarom ADR's, besproken in hoofdstuk 4, geen volledige oplossing bieden voor het probleem. Zijnde na het modelleren nog documenteren en de slechte onderhoudbaarheid.

5.2.5 Richtlijnen

Karsai et al [30] spreken in hun artikel over verscheidene richtlijnen voor een goed DSL:

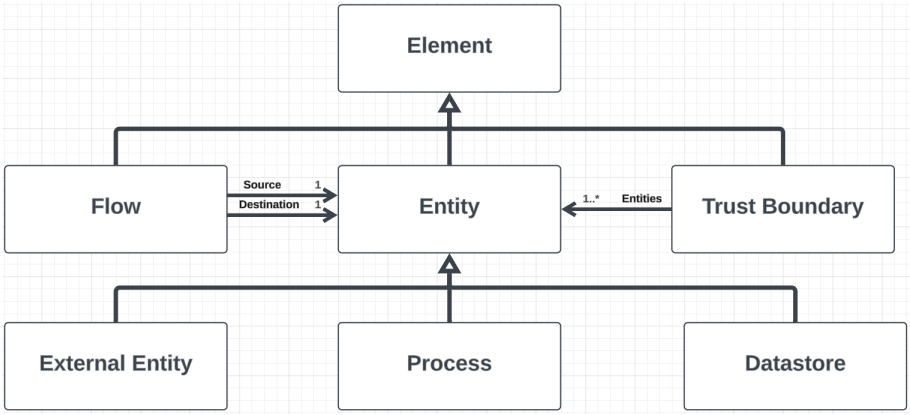
- Een taal dient consistent te zijn, wat impliceert dat de syntax uniform is.
- Onnodige generalisatie bemoeilijkt het leren van de taal.
- Limiteer het aantal taalelementen, zoals sleutelwoorden.
- De syntax moet zelf-documenterend zijn. Dit houdt in dat de syntax duidelijk en ondubbelzinnig de functie van het *statement* beschrijft.
- Identificeer gebruiksconventies in de syntax: wanneer er hoofdletters gebruikt moeten worden, de rangschikking van de elementen (b.v. attributen voor de methodes), etc.
- Pas op met *syntactic sugar*. Te veel *syntactic sugar* zorgt voor een onleesbare syntax. Een voorbeeld van *syntactic sugar* is het gebruik van de *lambda*-notatie in plaats van functie-notatie in Java.

Een vuistregel is het KISS acroniem voor *Keep It Simple, Stupid*. Bij het maken van een nieuwe DSL is het aangeraden om inspiratie te halen van bestaande DSLs/GPLs. De syntax voor de bestaande DSLs is vaak eerder onderzocht op gebruiksvriendelijkheid. Maar er moet een link zijn tussen het domein van het nieuwe DSL en het bestaande inspiratie DSL. Ten slotte is het aangeraden om commentaar toe te laten.

Bij naleving van de besproken richtlijnen bij de ontwikkeling, resulteert dit in een DSL dat zowel gebruikers- en leervriendelijker is.

5.3 Operaties

Met de doelstellingen en richtlijnen van de event-georiënteerde DSL in gedachten kan er overgegaan worden naar de operaties van de DSL. Om de operaties te kunnen verklaren worden nog kort de verschillende elementen van het DFD, en hun hiërarchische structuur, verklaard. Een DFD bestaat uit "entiteiten", dit kunnen externe entiteiten, processen of *datastores* zijn. Tussen de entiteiten zijn er *flows*. Ten slotte worden de entiteiten gegroepeerd door een *trust boundary*. In figuur 5.3 is overzicht te zien van alle elementen en hun hiërarchische structuur. De kernfunctionaliteit van de DSL is het documenteren van de rationaal achter de beslissingen. Daarom moet iedere operatie ook de mogelijkheid bieden om direct *inline* met het DSL-statement te documenteren. Hieronder worden de operaties verklaard, met de achterliggende acties die de operaties veroorzaakt, en waarom hiervoor gekozen is.



Figuur 5.3: DFD elementen hiërarchie

5.3.1 Toevoegen

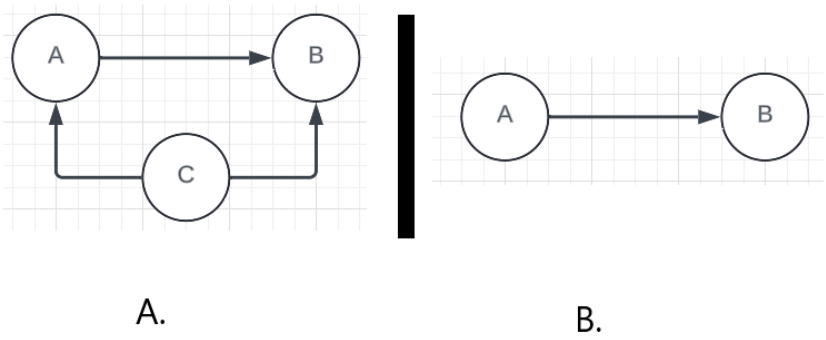
Om een volledig *data flow diagram* op te stellen moet er een mogelijkheid zijn om alle verschillende elementen toe te voegen. Daarom moet er voor ieder element een statement bestaan, die een nieuw desbetreffend element van het DFD introduceert. De operatie om een nieuw element toe te voegen aan het DFD moet het type van het element aangeven, zodat het DSL-statement resulteert en het juiste type element. De types zijn: `process` \ `external entity` \ `datastore` \ `flow` \ `boundary`. Vervolgens is het belangrijk om een goede *identifier* (ID) te kiezen voor het nieuwe element, omdat elk element binnen het DFD een unieke ID heeft. De ID wordt gebruikt om de individuele elementen van het DFD aan te spreken in de DSL-statements. Vervolgens is het mogelijk om een element een naam te geven. De output van het DSL-bestand, het DFD, presenteert de elementen op basis van hun naam. De naam moet niet uniek zijn, dus kan het zijn dat er meerdere elementen bestaan met dezelfde naam. Daarnaast is de naam optioneel. Dit laat de gebruiker toe om geen naam te specificeren. Wat nuttig kan zijn als de ID van het element genoeg benaming is voor het element. In dit geval zal de ID automatisch gebruikt worden als naam. Ten slotte moet het mogelijk zijn om een beschrijving van het element te geven. De beschrijving beschrijft kort wat het element doet en waarvoor het staat. Het is nuttig om de beschrijving optioneel te laten als het element een beschrijvende ID en/of naam heeft.

Als het te introduceren element een *data flow* is moeten de ID's van de bron en bestemming van de *flow* gekozen worden. De *flow* kan niet van een entiteit naar zichzelf verwijzen, zoals uitgelegd in de geldigheid doelstelling. Echter zal het statement ook falen als de gekozen bron of bestemming niet bestaat.

Om een nieuwe *trust boundary* te introduceren, moeten de ID's van de entiteiten gekozen worden die in de *trust boundary* zitten. Het statement zal falen als één van de gekozen entiteiten binnen in de *trust boundary* niet eerder is aangemaakt. Er kan zicht echter wel de situatie voordoen waarbij een entiteit in verschillende *trust boundaries* zit.

5.3.2 Verwijderen

Eens een DFD gemaakt is, is het mogelijk om één of meerdere elementen van het DFD te verwijderen. Om een element te verwijderen moet het desbetreffend element geselecteerd



Figuur 5.4: Voorbeeld effect verwijder operatie.

worden op basis van ID. De stelling is ongeldig wanneer er geen element bestaat met de overeenkomstige ID. Evenzeer heeft de operatie om één element te verwijderen gevolgen op andere elementen. Om te vermijden dat het verwijderen van een element het DFD in een ongeldige toestand brengt, worden de *flows* geassocieerd met het te verwijderen element mee verwijderd. Dit garandeert dat elke reeks DSL-statements resulteert in een geldig DFD, en de geldigheid doelstelling gegarandeerd blijft. Bijkomend wordt er verondersteld dat enkel entiteiten worden verwijderd, indien deze niet relevant zijn. Eveneens worden de *flows*, gekoppeld aan niet relevante entiteiten, automatisch mee minder relevant. Uiteindelijk zullen de relevante *flows* die gekoppeld zijn met het te verwijderen entiteit, gewijzigd en gekoppeld worden aan andere entiteiten, via een operatie die later nog besproken wordt. Bovendien zullen de meeste applicaties ook de *flows* geassocieerd met het te verwijderen element mee verwijderen. Ter illustratie, figuur 5.4 visualiseert het effect van het verwijderen van entiteit C.

5.3.3 Hernoemen, verander ID & beschrijving

Wanneer de functionaliteit van een element verandert moet de naam, beschrijving en mogelijks de ID van dat element mee veranderen. Er kunnen ook andere situaties ontstaan waar in de naam, ID en/of beschrijving van een element veranderd moeten worden. Om een element te hernoemen, de beschrijving of ID te veranderen moet het desbetreffend element geselecteerd worden op basis van ID. De stelling is ongeldig wanneer het element niet bestaat. Om de ID te veranderen van een element is de stelling ook ongeldig als het nieuwe ID al bestaat, zodat elk element een uniek ID heeft op het DFD.

5.3.4 Verander flow attributen

Als om een bepaalde reden de *flow* vanuit een andere entiteit moet vertrekken, dan vereist dit de wijziging van de bron van de *flow*. Als om een andere reden de *flow* moet verwijzen naar een andere entiteit, dan vereist dit de wijziging van de bestemming van de *flow*. Ten slotte kan het ook zijn dat er gewenst is om de bron en bestemming van de *flow* om te draaien. Om de attributen van de *flow* aan te passen moet desbetreffend *flow* geselecteerd worden op basis van de ID. Het statement is ongeldig als er geen *flow* overeenkomt met de opgegeven ID. Het selecteren van de bron of bestemming gebeurt ook op basis van ID,

eveneens zal het statement ook niet slagen als de nieuwe bron of bestemming van de *flow* niet bestaat. Tot slot is er de beperking dat een flow geen gelijke bron en bestemming kan hebben, wat de geldigheid doelstelling garandeert.

5.3.5 Verander trust boundary attributen

Na het creëren van een nieuwe entiteit of door andere omstandigheden kan het nodig zijn om een entiteit toe te voegen aan een *trust boundary*. Er kan ook behoefte zijn om een entiteit uit een *trust boundary* te halen. Om de attributen van de *trust boundary* aan te passen moet desbetreffend *trust boundary* en de entiteit geselecteerd worden op basis van de ID. Het statement zal ongeldig zijn als er geen *trust boundary* en geen entiteit bestaat met de opgegeven ID. Het statement zal ook niet slagen als de toe te voegen entiteit reeds in de *trust boundary* zit. Bij het verwijderen van een entiteit uit de *trust boundary* zal het statement wederom niet slagen als de te verwijderen entiteit niet in de *trust boundary* zit.

5.3.6 Entiteiten samenvoegen

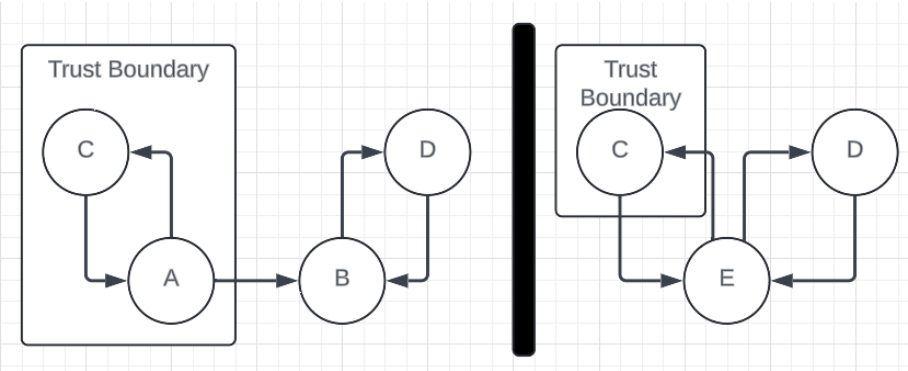
Er is een mogelijkheid dat er zich een scenario voordoet waarbij twee soort gelijke entiteiten bijna hetzelfde voorstellen of hetzelfde werk doen. Het kan opportuun zijn om desbetreffende entiteiten te *mergen* in een nieuwe entiteit. Er kan overwogen worden om een nieuwe entiteit te maken, de *flows* aan te passen en de twee oude entiteiten te elimineren. Echter zal dit ervoor zorgen dat er geen directe link is tussen de nieuwe aangemaakte entiteit en de twee verwijderden. De link is noodzakelijk om tijdens het analyseren van de *events* een traceerbaar pad te hebben naar het eindresultaat. Daarom biedt de DSL een operatie aan, die twee entiteiten samenvoegen in één stap/*event*.

De *flows* tussen de twee samenvoegende entiteiten worden automatisch verwijderd. Ook worden de *flows* gaande en aankomende aan de twee samenvoegende entiteiten gelinkt aan de nieuwe entiteit. De reden voor de automatische regeling van de *flows* is voor de doelstelling, geldigheid. Vervolgens wordt de nieuwe entiteit buiten eender welke *trust boundary* geplaatst. Om de nieuwe entiteit in een *trust boundary* te plaatsen wordt de eerder besproken operatie gebruikt om een nieuw element aan de *trust boundary* toe te voegen.

Er werd gekozen voor de mogelijkheid om eender welk type entiteit te gaan samenvoegen. Het kan dus zijn dat een externe entiteit en een *datastore* samengevoegd worden tot een nieuw proces. Deze manier van samenvoegen oogt vreemd, maar laat de eindgebruiker toe om zelf flexibel te werk te gaan.

Om de twee entiteiten te *mergen* moet de desbetreffende elementen geselecteerd worden op basis van hun ID. Het statement is ongeldig als één van de te *mergen* entiteiten niet bestaat. De nieuwe entiteit zal een ID en mogelijks een naam en/of beschrijving krijgen. Eveneens zal het statement ook niet slagen als er al een element bestaat met de ID van de nieuwe entiteit, dit zorgt ervoor dat ieder element een unieke ID heeft.

Ter illustratie van de *merge* operatie, in figuur 5.5 is het effect gevisualiseerd van het samenvoegen van entiteiten A en B naar entiteit E.



Figuur 5.5: Voorbeeld effect *merge* operatie.

5.3.7 Entiteit splitsen

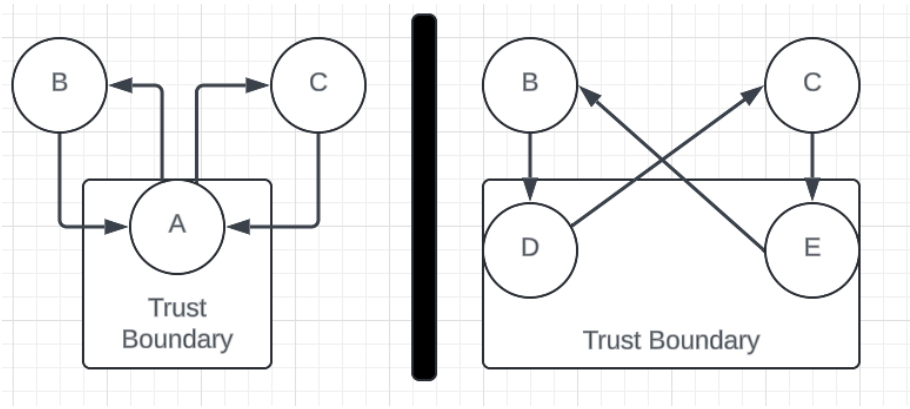
Het kan zijn dat er gewenst is om één entiteit op te splitsen in twee nieuwe. Een mogelijke manier om de entiteit op te splitsen is door twee nieuwe entiteiten aan te maken, de *flows* te hervormen en de oude entiteit te verwijderen. Echter zal dit ervoor zorgen dat er geen directe link is tussen de twee nieuwe en de verwijderde entiteit. Daarom biedt de DSL een operatie aan, die een entiteit splitst in twee nieuwe entiteiten, in één stap/*event*.

De nieuwe entiteiten krijgen automatisch dezelfde *trust boundary* als de oude entiteit. Echter moeten de *flows* van de oude entiteit manueel geïntegreerd worden in de nieuwe entiteiten, omdat dit niet automatisch kan gebeuren. Er moet dus een mechanisme komen dat de *flows* van de oude entiteit koppelt aan één van de twee nieuwen.

Er werd gekozen voor de mogelijkheid om een entiteit op te splitsen in twee entiteiten met diverse types. Het kan zijn dat een proces opgesplitst wordt in een *datastore* en een externe entiteit. Deze manier van splitsen oogt vreemd, maar laat de eindgebruiker toe om zelf flexibel te werk te gaan.

Om een entiteit op te splitsen moet de te splitsen entiteit geselecteerd worden op basis van ID. Het statement is ongeldig als de te splitsen entiteit niet bestaat. De twee nieuwe entiteiten zullen een ID en mogelijks een naam en/of beschrijving krijgen. Eveneens zal het statement ook niet slagen als één van de nieuwe entiteiten een ID heeft dat overeenkomt met de ID van een eerder aangemaakt element. Voor dezelfde reden, moeten de twee aangemaakte entiteiten verschillende ID hebben. De reden is dat ieder element een unieke ID moet hebben.

Ter illustratie van de splits operatie, in figuur 5.6 is het effect gevisualiseerd van het splitsen van entiteit A in entiteiten D en E. Hou er rekening mee dat de regeling van de *flows* manueel gebeurt, het effect van de splits operatie op de *flows* is ter illustratie.



Figuur 5.6: Voorbeeld effect splits operatie.

5.3.8 Groeperen

Documentatie van rationale is een doelstelling van de DSL. Het is mogelijk dat een reeks van DSL-statements een bepaald concept documenteert, in dat geval moeten desbetreffende statements gegroepeerd worden. De groepering moet een naam krijgen, om ze te identificeren, en een documentatie attribuut, om rationaal achter de groepering te kunnen documenteren.

De DSL ondersteunt drie soorten van groepering: *design, countermeasure & refactoring.* Een reeks van DSL-statements, gemarkeerd als een *design* groepering, stellen een reeks *events* voor die gebeurd zijn omwille van het ontwerp van de software. Een reeks DSL-statements die een *countermeasure* voorstelt, groeperen een design keuzen die genomen wordt om een kans op een bepaalde bedreiging te verminderen. Een reeks DSL-statements die een *refactoring* voorstelt, groeperen een design keuzen die genomen werd om het ontwerp van een software te wijzigen.

Hoofdstuk 6

Implementatie

Dit hoofdstuk focust op de implementatie van de event-georiënteerde DSL, rekening houdend met de doelstellingen en richtlijnen. De implementatie zal resulteren in de DSL, genaamd **Threat Modeling Language**, die een oplossing biedt voor de probleemstelling van deze thesis. De nieuw geïntroduceerde DSL krijgt bijhorende extensie `.tml`. In volgende sectie wordt de *DSL-parser* besproken, gemaakt in Eclipse Xtext, om de DSL-statements te transformeren naar een DFD. Vervolgens wordt de syntax verklaard.

6.1 Eclipse Xtext

Eclipse Xtext [31] is de technologie die gebruikt werd voor het bouwen van de event-georiënteerde DSL. Deze sectie behandelt eerst enkele concepten die essentieel zijn voor het ontwikkeling van een DSL: een *parser* en een *Integrated Development Environment (IDE)*. Vervolgens worden de noodzakelijke stappen voor het construeren van een DSL beschreven. Verder wordt de werking verklaard voor het implementeren van de syntax. En ten slotte worden de *Validator* en *Generator* klasse toegelicht.

6.1.1 Parser en IDE

Voor het ontwikkelen van een DSL zijn er twee concepten nodig: een syntax & een *parser*. De syntax beschrijft de operaties die de *parser* moet uitvoeren. De *parser* leest de operaties in de vorm van DSL-statements en interpreteert de statements zodat het resulteert in de gewenste acties. Er zijn verschillende methoden om een eigen DSL te ontwikkelen. Een mogelijke methode is om zelf een eigen *DSL-parser* te ontwikkelen. Dit proces is echter tijdrovend, omdat het neerkomt op "het wiel opnieuw uitvinden". Anderzijds bestaan er applicaties die ondersteuning bieden bij het ontwikkelen van een *parser*. Voor deze thesis is de applicatie Eclipse Xtext gebruikt. Echter zijn er ook talloze alternatieven, zoals ANTLR [32]. Er werd gekozen voor Eclipse Xtext omdat het de meest geschikte tools is voor het doeleinde van deze thesis, het maken van een DSL.

Een *parser* splitst een DSL statement op in verschillende delen, zoals sleutelwoorden, *identifiers*, *literals* & scheidingsteken. Sleutelwoorden van een DSL zijn woorden die op voorhand een bepaalde betekenis krijgen. *Identifiers* helpen om verschillende elementen te identificeren. *Literals* zijn waardes die gebruikt worden voor berekeningen of beschrijvingen van elementen. Voorbeelden van *literals* zijn nummers, booleaanse waarden of *strings*. Een scheidingsteken helpt de syntax van de DSL overzichtelijk te houden door

de verschillende statements en delen van een statement op te splitsen. Voorbeelden van scheidingsteken zijn spaties, puntkomma's, maar ook rekenkundige tekens, zoals het gelijkheidsteken. Om de statements op te delen gebruikt een *parser* een *Abstract Syntax Tree (AST)*. De AST representeert de abstracte structuur van de syntax in een boom vorm. De AST wordt gebruikt om de onderdelen van de DSL-statements te transformeren naar de acties die uitgevoerd moeten worden.

Een werkende *DSL-parser* vormt niet het einde van het ontwikkelen van de DSL. Werken met DSL gebeurt in een IDE, die de *DSL-parser* integreert. Een *Integrated Development Environment* (IDE) is een programma dat helpt bij het ontwikkelen van software. Er bestaan verschillende IDE's zoals Visual Studio Code, IntelliJ, maar ook Eclipse is een IDE. De IDE laat de ontwikkelaar toe om DSL-codes te schrijven in een bestand. Zodat nadien de *DSL-parser* het DSL-bestand interpreteert en transformeert naar de acties gevisualiseerd in de IDE. Bijkomend hebben IDE's ook features zoals *autocomplete*, syntax markering en foutafhandeling. *Autocomplete* assisteert de gebruiker van de DSL door suggesties te geven, hierdoor is het gebruiksvriendelijker en verhoogt de productiviteit. Syntax markering kleurt woorden en formatteert op basis van de syntax. De verbeterde leesbaarheid van de code draagt bij aan een efficiëntere fout detectie. Ten slotte moet een IDE helpen bij foutafhandeling. Syntax fouten die gemaakt zijn door de gebruiker van de DSL worden gemarkeerd met een rode onderstreping. Naast syntactische fouten kunnen ook er fouten optreden bij het interpreteren van het .tml-bestand, waarbij de IDE duidelijke foutmeldingen moet tonen om de oorzaak van het probleem te identificeren. Met Eclipse Xtext is het eenvoudig om een IDE te construeren die de *DSL-parser* integreert.

6.1.2 Een eigen DSL met Eclipse Xtext

Bij het ontwikkelen van een DSL dient er rekening gehouden te worden met de volgende aspecten. Ten eerste moet de ontwikkelaar de syntax definiëren in Xtext. Eveneens is het mogelijk dat de ontwikkelaar regels wil introduceren op de syntax, dit gebeurt aan de hand van *constraint checks* geschreven in de *Validator*, zie subsectie 6.1.4. Vervolgens moet de ontwikkelaar in de *Generator* de acties achter de DSL-statements beschrijven, zie subsectie 6.1.5. Na het voltooien van alle stappen, dienen de ontwikkelde concepten samen te komen. Een mogelijke aanpak is om vanuit het Xtext project een nieuwe Eclipse IDE *workspace* op te starten. In desbetreffende nieuwe *workspace* is de *DSL-parser* als plug-in voor-geïnstalleerd. Dit maakt het mogelijk om in de nieuwe *workspace* de DSL te gebruiken.

6.1.3 Syntax met Xtext

De eerste stap van het maken van een DSL is om de syntax te definiëren. De syntax van de event-georiënteerde DSL wordt gedefinieerd in het Xtext-bestand, door de operaties te declareren. Een operatie declareren gebeurt door ze een naam te geven, gevolgd door een dubbelpunt. De sleutelwoorden van de syntax worden geschreven tussen enkele aanhalingstekens. *Literals* en *identifiers* krijgen eerst een naam, gevolgd door het type, gescheiden door een gelijkheidsteken (bijvoorbeeld `naam=STRING`). Het type ID is het ingebouwde identificatie type van Eclipse Xtext en aangeraden voor de *identifiers*. Met ronde haakjes kan een deel van de operaties gegroepeerd worden. Stel dat er gewenst is om een deel van de operatie optioneel te maken, dan kan een vraagteken `(...)?` of

sterretje `(...)*` gebruikt worden. Het vraagteken staat voor nul of één keer, terwijl het sterretje staat voor nul of meerdere keren. Ter illustratie van de hernoem-operatie:
`'rename' id=ID 'into' newName=STRING documentation=STRING?;`
De Xtext-code voor de Threat Modeling Language is te vinden op GitLab[33].

Het gebruik van de in Xtext beschreven syntax vereist de generatie van de "Xtext Artifacts". Een deel van de *Xtext Artifacts* is Java code. Desbetreffende code stelt de syntax voor, gemaakt in Xtext. De code vormt een *Eclipse Modeling Framework (EMF)* model. Een EMF-model is een beschrijving van de structuur en inhoud van de syntax. Het wordt gedefinieerd met behulp van Ecore, een metamodel om de structuur van het EMF-model te beschrijven. Met de gegenereerde *Xtext Artifacts* kan de nieuwe syntax, omgezet in Java code, gebruikt worden in de *Validator* en *Generator*.

6.1.4 Validator

Met Xtext kan de syntax beschreven worden, maar om dieper gaande regels te definiëren rond het gebruik van de taal, moeten er *constraint checks* gedefinieerd worden. Voorbeelden van *constraint checks* zijn: de bron en bestemming van een *flow* kunnen niet hetzelfde zijn, een element hernoemen naar huidige naam kan niet, etc. *Constraint checks* worden gedefinieerd in de Xtend "Validator" klasse. Xtend is een *General-Purpose Language (GPL)* gebaseerd op de syntax van Java, hierdoor is het mogelijk om in Xtend de externe *libraries* van Java te gebruiken. De taal is gemaakt om alle aspecten van de DSL's, te implementeren, zoals het schrijven van *constraint checks*, het schrijven van de *Generator*, etc. Volgens Bettini [34] wordt er met Xtend leesbaar en "clean" code geschreven, tegen over Java. De documentatie van Xtend is te vinden op de website van Eclipse [35]. De *Validator* ondersteunt de declaratie van methoden die geannoteerd zijn met de `@Check` annotatie. In desbetreffende methodes moet er gecontroleerd worden of een statement aan de regels voldoet. In het geval dat de regels niet voldaan zijn, dient de "error" of de "warning" methode worden aangeroepen. In de methodes in kwestie wordt de problematische *literal* van het DSL-statement en de oorzaak van het probleem gespecificeerd. De Eclipse IDE *workspace*, met de *DSL-parser* als plug-in, zal een foutmelding of waarschuwing geven wanneer een statement niet voldoet aan de *constraint checks*. De foutmelding of waarschuwing gebeurt aan de hand van een rode onderstreep in de IDE. Desondanks controleert de *Validator* één enkele DSL-statement. Met de *Validator* is het niet mogelijk om complexere *constraint checks* te doen, die meerdere DSL-statements tegelijk controleren.

6.1.5 Generator

Ten slotte moeten de acties achter de operaties gedefinieerd worden. De mogelijke acties zijn divers. Voor de implementatie van de event-georiënteerde DSL resulteren de acties in een lijst van DFD elementen, zie eerste paragraaf 6.1.5. Vervolgens wordt de lijst van DFD elementen getransformeerd naar een DFD, zie tweede paragraaf 6.1.5. De lijst van elementen wordt getransformeerd naar een tekstuele en grafische representatie van een DFD. De grafische representatie wordt gerealiseerd door middel van Graphviz. Graphviz is een verzameling van open-source software tools, ontwikkeld om grafen te tekenen. De code voor de implementatie van de DSL is te vinden op GitLab [36].

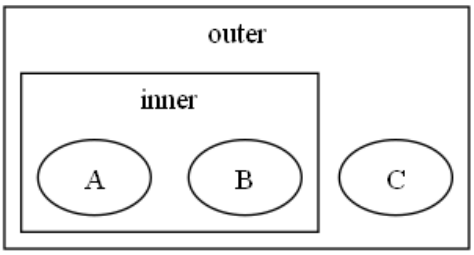
In de Xtend "Generator" klasse worden de acties achter de operaties gedeclareerd.

Deze klasse heeft een methode genaamd "doGenerate", die twee parameters mee krijgt genaamd "Resource" en "IFileSystemAccess2". Beide parameters horen bij het EMF-model. De *resource* parameter bevat een attribuut, genaamd "allContents". Het attribuut is een *iterator* met een sequentiële lijst van alle DSL-statements neergeschreven door de gebruiker van de DSL in het .tml-bestand. Ten slotte kan de *IFileSystemAccess2* parameter gebruikt worden om bestanden te genereren.

De werking achter de *Generator* klasse verloopt als volgt. Als het .tml-bestand die de code van de DSL bewaart, in de nieuwe Eclipse IDE *worskpace* wordt opgeslagen, dan zal de *doGenerate* methode aangeroepen worden van de *Generator* klasse.

Backend De acties achter de operaties van de event-georiënteerde DSL werden gedefinieerd in de zelfgeschreven *DLSHandler* klasse. De klasse heeft een methode genaamd *handleDSLFile*. De *Generator* klasse roept de methode *handleDSLFile* op wanneer de *doGenerate* methode wordt opgeroepen. De *handleDSLFile* methode krijgt het *iterator* object van de *Resource* parameter mee. In de *handleDSLFile* methode wordt een lijst aangemaakt, bedoeld voor het bewaren van de DFD elementen. Erna wordt er sequentieel geïtereerd over de DSL-statements, die de gebruiker van de DSL heeft neergeschreven in het .tml-bestand. Per DSL-statement wordt er gezocht naar het type operatie geassocieerd met het DSL-statement. Vervolgens worden de *literals* gebruikt, om het element te manipuleren die het DSL-statement aanspreekt. Het gemanipuleerde element wordt uiteindelijk opgeslagen in de lijst met de elementen van het DFD. Stel dat een DSL-statement ongeldig is. Bijvoorbeeld, de hernoem-operatie probeert een element te hernoemen op basis van een ID. Maar die ID bestaat niet. Dan geeft de *handleDSLFile* een *exception* terug.

Frontend Nadat de *handleDSLFile* methode een lijst met DFD elementen teruggeeft, zal de *doGenerate* methode andere methodes oproepen om het DFD te representeren. De eerste methode *printTextualDFD* genereert een tekstuele representatie van een DFD. In desbetreffende methode wordt er geïtereerd over de DFD elementen. Ieder DFD element wordt getransformeerd naar een *string*. Desbetreffende *strings* worden samengevoegd, om vervolgens geplaatst te worden in een tekstbestand, genaamd "demo.txt". In het tekstbestand komt een oplijsting van alle elementen en hun attributen. Nadien wordt de *printDot* methode opgeroepen. Deze methode genereert een *string* die gebruikt wordt om een DOT bestand te genereren, genaamd "dfd.dot". DOT is een DSL die Graphviz kan interpreteren om grafen te definiëren. In de *printDot* methode wordt er ook geïtereerd over alle elementen van het DFD. En wordt ieder DFD element, behalve de *trust boundaries*, getransformeerd naar een node, voorgesteld als een *string*. Een node, of een knoop, is onderdeel van een graaf. Tussen de *nodes* worden verbindingen gemaakt. Echter moet deze *string* voldoen aan de syntax van de DOT taal. De *trust boundaries* worden voorgesteld als een sub-graaf. De specificatie van de subgraaf vereist de identificatie van de *nodes* die deze omvat. Om geneste *trust boundaries* te definiëren moet de buitenste sub-graaf meer elementen hebben dan de binnenste sub-graaf. Ter illustratie, stel dat *trust boundary* "inner" elementen A en B heeft. En stel dat *trust boundary* "outer" de *trust boundary* "inner" en een extra element C heeft. Dan heeft de sub-graaf "outer" een geneste subgraaf "inner", zie figuur 6.2. Als alle elementen gerepresenteerd zijn in het DOT bestand, wordt het bestand getransformeerd naar een DFD. De methode *printPNG*, aangeroepen in de *doGenerate* methode, transformeert het DOT bestand naar een PNG. Dit gebeurt



Figuur 6.1: Voorbeeld geneste sub-grafen in Graphviz

aan de hand van het opdracht-prompt commando "-Tpng". Het commando is ontwikkeld door Graphviz. Ten slotte kan het zijn dat de *handleDSLFile* een *exception* teruggeeft, omdat er een ongeldig DSL-statement is. De methode *doGenerate* vangt de *exception* op en de boodschap van de *exception* wordt vertoond in het demo.txt bestand.

6.2 Syntax

In sectie 5.3 werden de verschillende operaties van de DSL opgesomd. Met behulp van Eclipse Xtext zijn de operaties omgevormd in de syntax van de DSL. In deze sectie wordt er dieper ingegaan in de syntax. De kern functionaliteit van de DSL, namelijk het documenteren van de rationale achter de architecturale beslissingen, zorgt ervoor dat het noodzakelijk is dat de DSL de functionaliteit aanbiedt om te documenteren en te linken aan de DSL-statement in kwestie. Deze manier van documenteren wordt in deze paper *inline* documenteren genoemd, en gebeurt aan de hand van het rationale-attribuut

6.2.1 Toevoegen

Om een volledig data *flow* diagram op te stellen moet er voor ieder type element een statement bestaan, die een nieuw desbetreffend element van het DFD introduceert. Verder heeft zoals voorheen aangegeven ieder element een ID. De gekozen ID van het element is een aaneengeschreven woord.

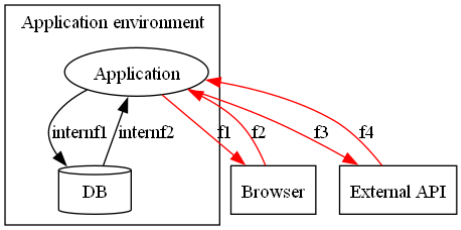
Entiteiten

Om nieuwe entiteiten te introduceren, worden praktisch dezelfde statements gebruikt. Ieder statement start met het trefwoord `add`, vervolgens de naam van het type van de nieuwe entiteit `process` \ `external entity` \ `datastore`. Om tot slot de nieuwe entiteit een ID, een naam (voorafgaand met het trefwoord `name`) en een beschrijving (voorafgaand met het trefwoord `dsc`) te geven. Ter illustratie:

```
add <<type>> <<ID>> name "<<name>>" dsc "<<description>>"
"<<documentation>>"
```

Flow

Om een nieuwe *flow* te introduceren wordt bijna hetzelfde statement gebruikt als de entiteiten, met de uitbreiding om de bron en de bestemming van de *flow* te kiezen. Ter illustratie:



Figuur 6.2: DFD geresulteerd uit de toevoeg statements.

```
add flow <<ID>> name "<<name>>" dsc "<<description>>"
<<sourceID>> -> <<destinationID>> "<<documentation>>"
```

Trust boundary

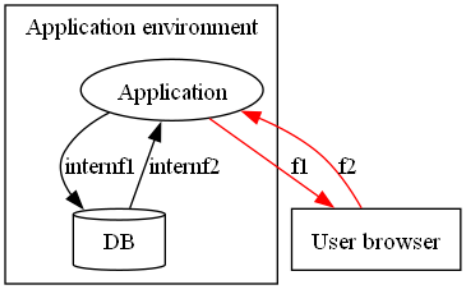
Om een nieuwe *trust boundary* te introduceren wordt net zoals bij de entiteiten en *flow*, vrijwel hetzelfde statement gebruikt, met de uitbreiding om de entiteiten te kiezen die in de *trust boundary* zitten. De opsomming van entiteit wordt gesplitst met een spatie. Ter illustratie:

```
add boundary <<ID>> name "<<name>>" dsc "<<description>>"
<<entityID>> <<entityID>> <<entityID>>  ... "<<documentation>>"
```

Voorbeeld

Figuur 6.2 illustreert een DFD gemaakt op basis van volgende statements:

```
add process Application dsc "Backend Java application"
add datastore DB dsc "NoSQL"
add flow internf1 Application -> DB
add flow internf2 DB -> Application
add external entity Browser name "User browser"
add flow f1 dsc "HTTPS" Application -> Browser
add flow f2 dsc "HTTPS" Browser -> Application
add external entity ExternAPI name "External API" dsc "An api
    provided by an external application" "When using the
   functionality of an external application the security part
   is not our responsablility."
add flow f3 dsc "HTTPS" Application -> ExternAPI
add flow f4 dsc "HTTPS" ExternAPI -> Application
add boundary ApplicatieENV name "Application environment" :
   Application DB "The application and database run in the
   same environment so there exists a high level of mutual
   trust between both parties involved."
```



Figuur 6.3: DFD geresulteerd uit de verwijder statement.

6.2.2 Verwijderen

Er moet de mogelijkheid zijn om één of meerdere elementen van het DFD te verwijderen. Een element verwijderen gebeurt aan de hand van de ID van desbetreffend element. Een entiteit, *trust boundary* of *flow* verwijderen gebeurt op basis van hetzelfde statement. Begin het statement met `remove`, gevolgd door de ID van het te verwijderen element. Ter illustratie:

```
remove <<ID>> "<<documentation>>"
```

Voorbeeld

Ter illustratie, stel dat uit figuur 6.2 de `External API` wordt verwijderd. Dit resulteert in figuur 6.3, waar zowel *flows* `f3` en `f4` mee verwijderd worden. Hiervoor is volgend statement nodig:

```
remove ExternAPI "The applications does not use the externa api
   anymore, flows using the ExternAPI will be deleted too (f3,f4)"
```

6.2.3 Verander ID

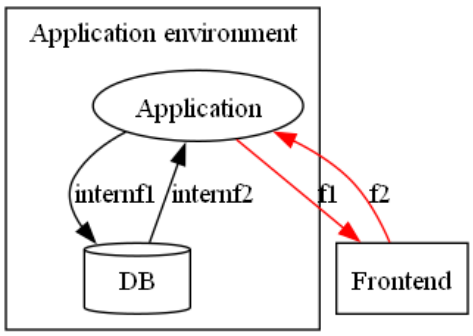
Het kan zijn dat er gewenst wordt om de ID te veranderen. Deze stelling is voor iedere type element hetzelfde. Begin het statement met het trefwoord `change`, dan de ID, vervolgens het trefwoord `ID`, om te eindigen met de nieuwe ID. De nieuwe ID moet opnieuw een aaneengeschreven woord zijn. Ter illustratie:

```
change <<oldID>> id <<newID>> "<<documentation>>"
```

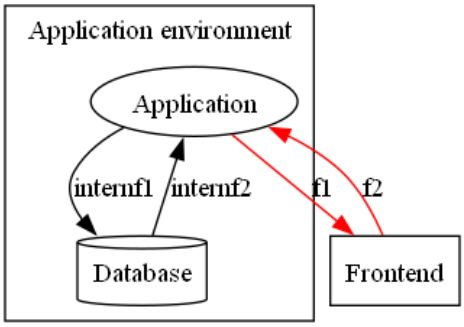
Voorbeeld

Ter illustratie, stel dat uit figuur 6.3 de ID van de *Browser* veranderd wordt naar *Frontend*, resulteert dit in figuur 6.4. Omdat de `Frontend` geen naam heeft zal de ID gebruikt worden. Volgend statement resulteert in figuur 6.4:

```
change Browser id Frontend "The id browser is to specific, with
   changing the id to frontend we get a more generale element."
```



Figuur 6.4: DFD geresulteerd na het aanpassen van de ID.



Figuur 6.5: DFD geresulteerd na het hernoemen.

6.2.4 Hernoemen

Het kan zijn dat er een element hernoemd wordt. Deze stelling is voor ieder type element hetzelfde. Begin het statement met het trefwoord `rename`, gevolgd door de ID van het te hernoemen element (`<<ID>>`), om vervolgens het trefwoord `into` te schrijven, om te eindigen met de nieuwe naam (`<<name>>`). Ter illustratie:

```
rename <<ID>> into "<<name>>" "<<documentation>>"
```

Voorbeeld

Ter illustratie, stel dat uit figuur 6.4 de *datastore* `DB` hernoemd wordt naar `Database`, resulteert dit in figuur 6.5. Volgend statement resulteert in de figuur:

```
rename DB into "Database" "The name Database is beter than the
   abbreviation DB"
```

6.2.5 Verander beschrijving

Als de functionaliteit van een element verandert, moet de beschrijving dat ook weergeven, waardoor de beschrijving mee aangepast wordt. Het statement begint met trefwoord `change`, gevolgd met de ID van het element (`<<ID>>`), met erna het trefwoord `description`, om te eindigen met de nieuwe beschrijving (`<<description>>`) tussen aanhalingstekens. Ter illustratie:

```
change <<ID>> description "<<description>>" "<<documentation>>"
```

Voorbeeld

Ter illustratie, stel dat uit figuur 6.5 *datastore Database* verandert van een SQL databank naar een NoSQL databank. De beschrijving van de *datastore* moet dit representeren, via volgend statement:

```
change DB description "NoSQL database for beter performance."
```

6.2.6 Verander flow bron

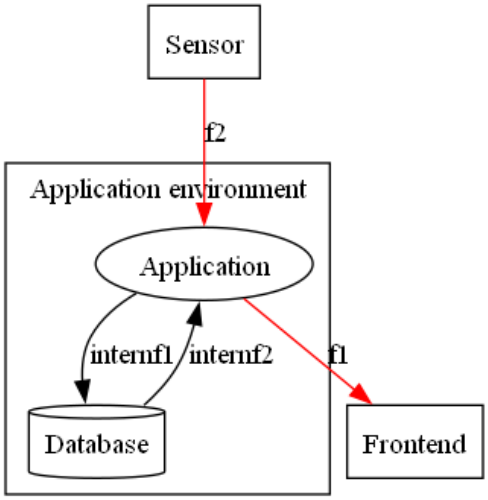
Als om een bepaalde reden de *flow* vanuit een andere entiteit moet vertrekken, dan vereist dit de wijziging van de bron van de *flow*. Het statement dat deze functionaliteit ondersteunt start met het trefwoord `change`, gevolgd met de ID van de *flow* (`<<ID>>`), vervolgens het trefwoord `source`, om af te sluiten met de ID van de entiteit die de nieuwe bron van de *flow* representeert (`<<sourceID>>`). Ter illustratie:

```
change <<ID>> source <<sourceID>> "<<documentation>>"
```

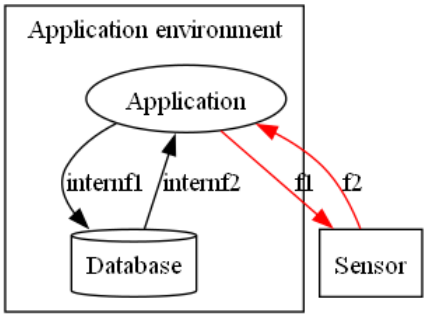
Voorbeeld

Ter illustratie, stel dat figuur 6.5 uitgebreid wordt met een nieuwe externe entiteit, genaamd `Sensor`. De `Sensor` is de enige input van de applicatie, zodat de `Frontend` enkel output krijgt van de applicatie. Vervolgens zal de *flow* `f2` van bron moeten veranderen. Volgende statements zullen figuur 6.6 tot stand brengen:

```
add external entity Sensor dsc "This is the only input of the
   application" "The new sensor needed for input"
change f2 source Sensor "Connecting the sensor with the
   application"
```



Figuur 6.6: DFD geresulteerd na het aanpassen van de bron.



Figuur 6.7: DFD geresulteerd na het aanpassen van de bestemming.

6.2.7 Verander flow bestemming

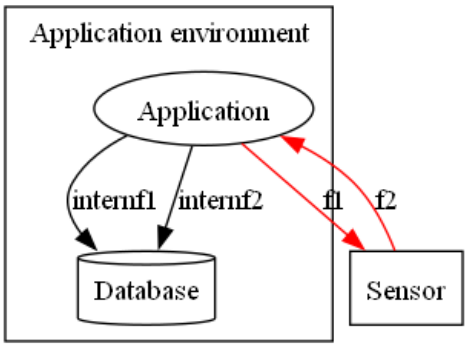
Als om een bepaalde reden de *flow* moet verwijzen naar een andere entiteit, dan vereist dit de wijziging van de bestemming van de *flow*. Het statement dat deze functionaliteit ondersteunt start met het trefwoord `change`, gevolgd met de ID van de *flow* (`<<ID>>`), met erna het trefwoord `destination`, om af te sluiten met de ID van de entiteit die de nieuwe bestemming van de *flow* representeert (`<<destinationID>>`). Ter illustratie:

```
change <<ID>> destination <<destinationID>> "<<documentation>>"
```

Voorbeeld

Ter illustratie, stel dat figuur 6.6 aangepast wordt, zodat er geen `Frontend` meer is, maar de communicatie met de applicatie via sensors gebeurt. De bestemming van *f1* moet vervolgens veranderen. Evenzeer wordt de alleenstaande entiteit, de `Frontend`, verwijderd. De aanstaande statements zullen resulteren in figuur 6.7:

```
change f1 destination Sensor "Let the application communicate
   with the sensors"
remove Frontend "No UI frontend is available, everything works
   with sensors"
```



Figuur 6.8: DFD geresulteerd na het omdraaien van de *flow*.

6.2.8 Inverteer flow

In bepaalde situaties kan de gebruiker verlangen om de bron en de bestemming van de *flow* om te draaien. Het statement dat deze functionaliteit teweegbrengt start met het trefwoord `change`, gevolgd met de ID van de *flow* (`<<ID>>`), om te eindige met de sleutelwoorden `revert flow`. Ter illustratie:

```
change <<ID>> revert flow "<<documentation>>"
```

Voorbeeld

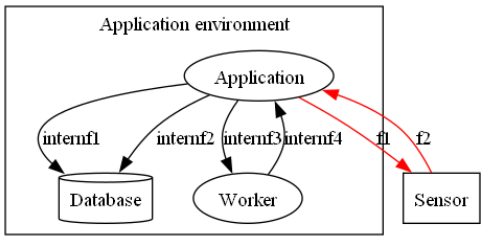
Ter illustratie, stel dat figuur 6.7 aangepast wordt, zodat er enkel maar geschreven wordt naar de `Databank`. De bron en bestemming van *flow* `internf2` moet vervolgens omdraaien. Volgend statement zal resulteren in figuur 6.8:

```
change internf2 revert flow "No more options to read data from te
    Databank. There is a flow secured and not secured flow."
```

6.2.9 Voeg entiteit toe aan boundary

Na het creëren van een nieuwe entiteit of door andere omstandigheden kan het nodig zijn om een entiteit toe te voegen aan een *trust boundary*. Het statement dat resulteert in deze functionaliteit start met het trefwoord `change`, gevolgd met de ID van de *trust boundary* (`<<ID>>`), met erna het trefwoord `add`, om af te sluiten met de ID van de toe te voegen entiteit (`<<entityID>>`). Ter illustratie:

```
change <<ID>> add <<entityID>> "<<documentation>>"
```



Figuur 6.9: DFD geresulteerd na het toevoegen van een entiteit aan de *trust boundary*.

Voorbeeld

Ter illustratie, stel de architectuur van het softwaresysteem van figuur 6.8 wordt aangepast zodat er een nieuw proces, genaamd *Worker*, wordt toegevoegd. Dit zou resulteren in betere beschikbaarheid en verlichte werkdruk op het proces, *Application*. De navolgende statements zullen de figuur 6.9 teweeg brengen:

```
add process Worker "A new worker to balance the workload"
change ApplicatieENV add Worker "The new worker is part of the
   system"
add flow internf3 Application -> Worker
add flow internf4 Worker -> Application
```

6.2.10 Verwijder entiteit van boundary

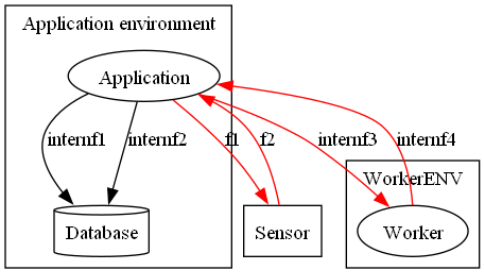
Er kan behoefte zijn om een entiteit uit een *trust boundary* te halen. Het statement hiervoor start met het trefwoord `change`, gevolgd met de ID van de *trust boundary*, met erna het trefwoord `remove`, om af te sluiten met de ID van de entiteit die verwijderd wordt uit de *trust boundary* (`<<entityID>>`). Ter illustratie:

```
change <<ID>> remove <<entityID>> "<<documentation>>"
```

Voorbeeld

Ter illustratie, stel de architectuur van het softwaresysteem van figuur 6.9 wordt aangepast, zodat het `Worker` proces in een eigen *trust boundary* zit, buiten de bestaande `ApplicatieENV` *trust boundary*. Een nieuwe *trust boundary*, genaamd `WorkerENV`, moet vervolgens eerst worden aangemaakt. Het `Worker` proces wordt toegevoegd aan de *trust boundary* `WorkerENV`. Om vervolgens het proces `Worker` te verwijderen uit de `ApplicatieENV` *trust boundary*. De hier opvolgende statements zullen de besproken architecturale aanpassingen teweegbrengen en resulteren in figuur 6.10:

```
add boundary WorkerENV: Worker "A new trust boundary only for the
    worker"
change ApplicatieENV remove Worker "Adding the worker to its
   boundary"
```



Figuur 6.10: DFD geresulteerd na het verwijderen van een entiteit van de *trust boundary*.

6.2.11 Entiteiten samenvoegen

Er is een mogelijkheid dat er zich een scenario voordoet waarbij twee soort gelijke entiteiten hetzelfde voorstellen of hetzelfde werk doen. Het *mergen* statement start met het gelijknamige trefwoord `merge`, gevolgd door de ID van het eerst te *mergen* entiteit (`<<entity1ID>>`), gevolgd door het trefwoord `and`, om erna het tweede te *mergen* entiteit te kiezen (`<<entity2ID>>`). Hierna zal het trefwoord `into` gebruikt worden. Vervolgens het type van de nieuwe entiteit te kiezen (`<<type>>`), zijnde `process` \ `external entity` \ `datastore`. Tot slot de ID, de naam en de beschrijving van de nieuwe entiteit. Een voorbeeld van het *merge* statement:

```
merge <<entity1ID>> and <<entity2ID>> into <<type>> <<ID>> name
   "<<name>>" dsc "<<description>>" "<<documentation>>"
```

Alternatieve syntax

Een alternatieve syntax is een statement dat twee entiteiten samenvoegt, in de zin van integreer de eerste entiteit in de tweede. De syntax van dit statement had er als volgt uitgezien: start met het trefwoord `merge`, vervolgt door de ID van de eerste entiteit (`<<entity1ID>>`), vervolgens het trefwoord `into`, om het statement af te ronden met de id van de tweede entiteit waarin de eerste wordt ingevoegd (`<<entity2ID>>`). Een voorbeeld van het *merge* statement is:

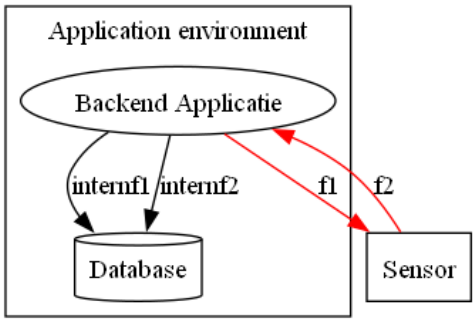
```
merge <<entity1ID>> into <<entity2ID>> "<<documentation>>"
```

Ook bij de alternatieve syntax zullen de *flows* tussen beide entiteiten automatisch worden verwijderd. Hierboven op zullen de *flows* die gekoppeld zijn aan de eerste entiteit automatisch gekoppeld worden aan de tweede entiteit. Tot slot zal de tweede entiteit in zijn *trust boundaries* blijven.

De reden dat er niet voor de desbetreffende syntax is gekozen, is omdat die minder flexibel is. Flexibiliteit van de oorspronkelijke syntax laat de gebruiker toe om makkelijker een nieuwe entiteit aan te maken van eender welk type. De volgorde van de entiteiten die samengevoegd moeten worden zorgen bij de alternatieve syntax ook voor de achterliggende acties. Wat extra complex is.

Voorbeeld

Ter illustratie, stel dat de architecturale beslissingen die genomen zijn voor het maken van figuur 6.10 niet het gewenste resultaat geven. Het kan zijn dat de gebruiker toch



Figuur 6.11: DFD geresulteerd na het merge statement.

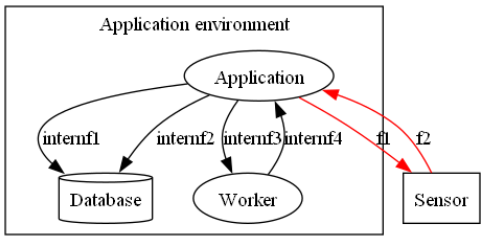
niet wil werken met de `Worker`. Één optie is om de `Worker` te verwijderen. Echter als de gebruiker ervoor kiest om de processen, `Application` en `Worker`, samen te voegen in een nieuw proces genaamd `BackendApplicatie`. Dan zal er een traceerbaar pad zijn, van de genomen acties met bijhorende redeneringen, naar het eindresultaat. Tot slot moet het nieuwe proces in de juiste *trust boundary* komen en de leegstaande *trust boundary* nog verwijderd worden. Het voorbeeld om zetten in DSL-statements resulteert in figuur 6.11:

```
merge Application and Worker into process BackendApplicatie name
   "Backend Applicatie" "This is the same as the previous
   Application process."
change ApplicatieENV add BackendApplicatie "Add the new element
   to the existing boundary."
remove WorkerENV "Remove the old not needed boundary."
```

6.2.12 Entiteit splitsen

Het kan zijn dat er gewenst is om één entiteit op te splitsen in twee nieuwe. Het split statement start met het gelijknamig trefwoord `split`, gevolgd door de ID van de op te splitsen entiteit (`<<ID>>`), vervolgt door het trefwoord `into`. Vervolgens moet het type (`<<type>>`), zijnde `process` \ `external entity` \`datastore`, de ID, de naam en de beschrijving van de eerste nieuwe entiteit gekozen worden. Het trefwoord `and` zorgt voor de overgang naar het kiezen van het type (`<<type>>`), de ID, de naam en de beschrijving van de tweede nieuwe entiteit. Tot slot eindigt het statement met het trefwoord `flows` en de groepering van *flow-fix-statements* gegroepeerd door accolades: “{}”. Een voorbeeld van het splits statement:

```
split <<ID>> into <<type>> <<entity1ID>> name "<<entity1Name>>"
   dsc "<<entity1Description>>" and <<type>> <<entity2ID>> name  "
   <<entity2Name>>" dsc "<<entity2Description>>" flows {
    flow-Fix statement
    flow-Fix-statement
    ...
} "<<documentation>>"
```



Figuur 6.12: DFD geresulteerd na het splits statement.

Flow-fix-statement

De *flow-fix-statements* verdelen de *flows* van de oude entiteit over de nieuwen. Dit moet gebeuren samen met het *split* statement, zodat de taal voldoet aan de doelstelling: geldigheid.

De *flow-fix-statements* scheiden gebeurt op basis van een nieuwe lijn of een spatie. Iedere *flow-fix-statement* start met de ID van de *flow* (`<<flowID>>`), gevolgd met de *syntax sugar* `->`, om te eindige met de ID van één van de nieuwe entiteiten (`<<entiteitID>>`). Ter illustratie: `<<flowID>> -> <<entiteitID>>`

De *flow-fix-statements* vervangt de bron/bestemming van de *flow*, wat de oude entiteit is, door de nieuwe gekozen entiteit.

Het *split* statement zal enkel slagen indien iedere *flow* van de oude entiteit gecorrigeerd en gekoppeld is aan één nieuwe entiteit. Iedere *flow* zal dus in exact één *flow-fix-statement* moeten terugkomen. Indien er meer *flow-fix-statement* dan *flows* zijn, zal het statement niet slagen.

Voorbeeld

Ter illustratie, stel dat de architecturale beslissing om de `Worker` te verwijderen uit figuur 6.11 niet het gewenste resultaat leverde. Dit kan opgelost worden door de `Worker` terug te brengen. Één optie is om opnieuw een `Worker` proces te maken. Echter door `BackendApplicatie` op te splitsen in `Application` en `Worker` kan er een traceerbaar pad gemaakt worden van de nieuwe entiteiten naar de oude. *Flows* `f1, f2, internf1 & internf2` moeten gecorrigeerd worden. Ten slotte worden *flows* `internf3` en `internf4` aangemaakt. Dit voorbeeld omvormen in volgende DSL-statements zal resulteren in figuur 6.12:

```
split BackendApplicatie into process Application name "Backend
   Application" dsc "This backend uses Java" and process Worker
   name "Backend Worker" dsc "This process runs on Python" flows {
    f1->Application
    f2->Application
    internf1->Application
    internf2->Application
} "Splits the BackandApplication up into the previous 2 processes
    because of the strong decoupeling."
add flow internf3 Application -> Worker
add flow internf4 Worker -> Application
```

6.2.13 Groeperen

Het is mogelijk dat een reeks van DSL-statements een bepaald concept documenteren. De desbetreffende statements zouden gegroepeerd moeten worden. Ieder groepering statement start met desbetreffende type (`<<type>>`): `countermeasure` \ `design` \ `refactoring`, gevolgd met de ID van de groepering (`<<ID>>`), voortgaand met de documentatie tussen aanhalingstekens (`<<documentatie>>`). Het statement eindigt met de groepering van de DSL-statements, gegroepeerd door accolades: "{}". Ter illustratie:

```
<<typen>> <<ID>> "<documentatie>>" {
    DSL-statement
    DSL-statement
    ...
}
```

Voorbeeld groeperen

Ter illustratie, enkele DSL-statements die als volgt gegroepeerd kunnen worden:

```
design initialDesign "The initial design of the software" {
    add process Application dsc "Backend Java application"
    add datastore DB dsc "NoSQL"
    add flow internf1 Application -> DB
    add flow internf2 DB -> Application
    add external entity Browser
    add flow f1 dsc "HTTPS" Application -> Browser
    add flow f2 dsc "HTTPS" Browser -> Application
    add external entity ExternAPI dsc "An api provided by an
       external application"
    add flow f3 dsc "HTTPS" Application -> ExternAPI
    add flow f4 dsc "HTTPS" ExternAPI -> Application
    add boundary ApplicatieENV: Applicatie DB
}
refactoring v1.1Design "Just some casual refactoring because of
   design changes" {
    remove ExternAPI
    rename DB into Databank
}
countermeasure addfirewall "Adding a firewall between the Browser
    and the application will provide extra security." {
    add process firewall
    add flow f5 browser -> firewall
    add flow f6 firewall-> browser
    change f1 destination firewall
    change f2 source firewall
}
```

Hoofdstuk 7

Evaluatie

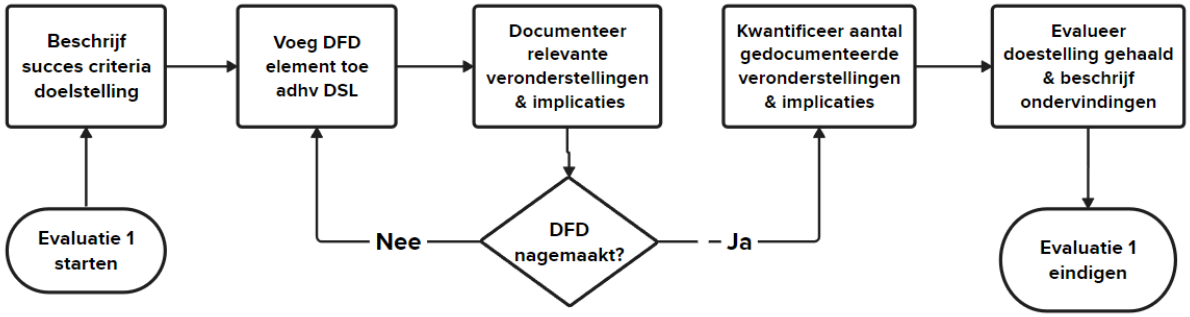
De nieuw geconstrueerde event-georiënteerde DSL Threat Modeling Language wordt gecontroleerd om te valideren of deze voldoet aan de vooropgestelde doelstellingen. De doelstellingen zijn:

- Functionaliteit
- Geldigheid
- Documentatie van rationale
- Operationeelheid

Voor de evaluatie zijn er twee afzonderlijke tests uitgevoerd. De eerste test evalueert de documentatie van rationale, en de tweede test evalueert de functionaliteit, geldigheid en operationeelheid. Beide testen zijn gebaseerd op het threat model van de open-source applicatie SecureDrop [37]. SecureDrop wordt gebruikt door media organisaties om externe documenten te gebruiken op een veilige manier, zodat de oorsprong van de documenten anoniem blijft. Het beschermt de identiteit van de bronnen van de documenten zodat de bronnen op een veilige manier gevoelige informatie kunnen delen. In sectie 7.1 wordt de werking van SecureDrop verklaard. SecureDrop heeft hun threat model [38], met bijhorend DFD, publiek staan. Dit maakt het mogelijk om gebruik te maken van de veronderstellingen, die ze hebben op de elementen van hun DFD. Het threat model is gebaseerd op deze veronderstellingen. Voorts is er een lijst van gevolgen bij mogelijke inbreuken in het systeem. De gevolgen die worden opgelijst waren het resultaat van het threat model. Echter is er geen documentatie met architecturale beslissingen rond hun design beschikbaar. Het publieke threat model en DFD wordt tijdens de evaluatie gebruikt als een *case*-evaluatie. Waarbij de *case* wordt nagebouwd en de veronderstellingen en gevolgen gebruikt als voorbeeld documentatie.

7.1 SecureDrop

De applicatie SecureDrop is gecreëerd voor het veilig gebruik van documenten, aangeleverd door onbekende bronnen. De werking van SecureDrop is als volgt. Eerst uploadt een derde partij, volgens SecureDrop genaamd een *source*, een document naar de SecureDrop applicatie. Het desbetreffende document wordt geëncrypteerd en opgeslagen. Wanneer



Figuur 7.1: *Workflow* diagram van de eerste evaluatie.

een journalist een document nodig heeft zal hij/zij het geëncrypteerde document in kwestie opzoeken en downloaden. Om het document te openen moet de journalist gebruik maken van een *airgapped viewing station*. Dit werkstation is geïsoleerd van het netwerk en mag nooit met een netwerk verbonden worden. Het document decrypten en openen in het *airgapped viewing station* zorgt ervoor dat de bronnen van het document niet worden onthuld. Bijkomend kan het mogelijks geïnfecteerde document enkel het *station* besmetten. Aangezien het *station* niet verbonden is met het netwerk, kan het mogelijks geïnfecteerde document niet heel de organisatie aanvallen. Omdat het *airgapped viewing station* niet met het internet is verbonden, kan het ook niet de documenten downloaden van de SecureDrop applicatie. Hiervoor zal de journalist eerst het geëncrypteerd document moeten downloaden op een aparte pc, dan overzetten naar een USB, om vervolgens de USB te gebruiken in het *airgapped viewing station*. Het document exporteren van het *airgapped viewing station* naar een andere pc om het reeds veilige document te gebruiken, is opnieuw aan de hand van een USB.

7.2 Documentatie rationale

De eerste test evalueert de doelstelling rond documentatie van rationale. Het proces voor desbetreffende test is als volgt, zet eerst de evaluatie op door de criteria voor het behalen van de doelstelling te beschrijven. Eveneens moet het design en threat model van SecureDrop op voorhand bestudeerd worden. Vervolgens de uitwerking van de evaluatie, door het DFD na te maken met de event-georiënteerde DSL en de veronderstellingen en gevolgen van het openbare threat model te documenteren in de beschrijvings- en rationale-attributen. Verder wordt de verhouding van de verdeling van de veronderstellingen en de gevolgen over de beschrijvings- en rationale-attributen bijgehouden in een tabel. Ten slotte worden de resultaten van de test besproken. Figuur 7.1 illustreert het evaluatieproces aan de hand van een *workflow* diagram.

7.2.1 Opzet evaluatie

Omschrijf de criteria waaraan moet worden voldaan om de doelstelling, documentatie van rationale, te behalen, voor de eerste evaluatie test op te zetten. De doelstelling documentatie van rationale zal voldaan zijn wanneer volgende twee criteria gelden. Ten eerste moet er de mogelijkheid zijn om de redenering achter iedere beslissing te documenteren

en te linken aan het betreffend DSL-statement. Ten tweede, kan de uitvoering van een bepaalde beslissing in de DSL leiden tot een reeks DSL-statements. Dan moet de DSL de mogelijkheid aanbieden om desbetreffende statements te groeperen en de documentatie aan de groepering te linken.

7.2.2 Uitwerking

Voor de uitvoering van de eerste evaluatie wordt het DFD van SecureDrop voorgesteld in de DSL en moeten de relevante veronderstellingen en gevolgen gedocumenteerd en gelinkt worden aan desbetreffende DSL-statements. Dit gebeurt op volgende manier, telkens er in de DSL een nieuw element werd geïntroduceerd, werd er gezocht in de lijst van veronderstellingen en in de lijst van gevolgen naar relevante factoren die gedocumenteerd konden worden in de beschrijvings- of rationale-attributen. Het .tml bestand dat gebruikt werd voor de eerste test is the vinden op GitLab [39] en in de bijlage A.1.1. Het DFD resulterend van het .tml-bestand is te vinden in bijlage A.1.2. De verhoudingen van de veronderstellingen en gevolgen verdeeld over de beschrijvings- of rationale-attributen worden bewaard in tabel 7.1. SecureDrop deelt de veronderstellingen op in vier categorieën:

- Veronderstellingen met betrekking tot de actoren van de applicatie.
- Veronderstellingen met betrekking tot de hardware waarop de diverse microservices van de applicatie draaien.
- Veronderstellingen met betrekking tot de organisatie die SecureDrop gebruikt.
- Algemene veronderstellingen, die niet in de andere categorieën passen.

SecureDrop deelt de gevolgen op in vijf categorieën:

- Gevolgen bij een inbreuk in de diverse microservices van de applicatie.
- Gevolgen bij een fysieke overname van de diverse microservices van de applicatie.
- Gevolgen van een lokale netwerk aanvaller.
- Gevolgen van globale aanvallers.
- Gevolgen van mogelijke acties van bezoekers van de applicatie.

	Beschrijving #	Documentatie #	Geplaatst #	Totaal	Beschrijving #	Documentatie #	Geplaatst #
Veronderstellingen							
Actoren	3	1	4	14	21%	7%	29%
Hardware	11	1	12	13	85%	8%	92%
Organisatie	0	1	1	5	0%	25%	25%
Algemeen	1	0	1	5	20%	0%	20%
Totaal veronderstellingen	15	3	18	36	42%	8%	50%
Gevolgen							
Inbreuk	20	15	35	51	39%	29%	69%
Fysieke overnamen	0	7	7	19	0%	37%	37%
Lokale netwerk aanvaller	0	0	0	3	0%	0%	0%
Globale aanvaller	0	3	3	5	0%	60%	60%
Willekeurig persoon	0	2	2	5	0%	40%	40%
Totaal Gevolgen	20	27	47	83	24%	33%	57%
Algemeen totaal	**40**	**42**	**82**	**134**	**30%**	**31%**	**61%**

Tabel 7.1: De verhouding van de veronderstellingen en gevolgen gedocumenteerd in de beschrijvings- of rationale-attributen

7.2.3 Resultaten

Tijdens het uitvoeren van de eerste evaluatie was het mogelijk om veronderstellingen en gevolgen te documenteren en te linken aan de gewenste de DSL-statements, zowel *inline* met het statement als via een groepering element. Hieruit bleek dat de doelstelling documentatie van rationale van de eerste evaluatie was voldaan.

De implementatie van de DSL liet toe om aan ieder DSL-statement *inline* documentatie te linken. Het feit dat aan elk statement documentatie gekoppeld kan worden betekent niet dat de inhoud van de documentatie er op zijn plaats is. Uit de test bleken er echter ook veronderstellingen en gevolgen te zijn die niet thuishoren bij het beschrijvings- en rationale-attribuut onafhankelijk van het element. Een voorbeeld hiervan zijn de gevolgen van een lokale netwerk aanvaller. Dit komt omdat desbetreffende veronderstellingen en gevolgen algemene informatie bevatten over de applicatie, waardoor ze niet aan een element of een reeks DSL-statements gekoppeld kunnen worden. Om die redenen kon 39% van de veronderstellingen en gevolgen niet gebruikt worden. In totaal zijn er van de 134 veronderstellingen en gevolgen, er 82 gebruikt (61%).

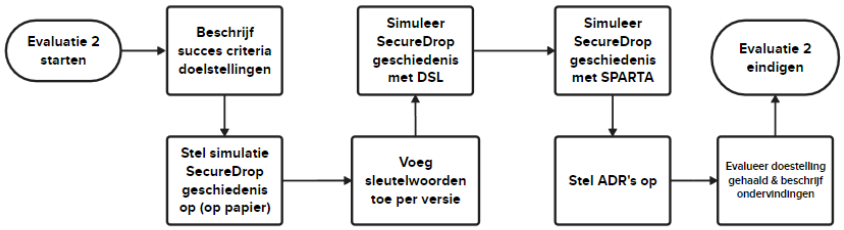
Bovendien was het complex om de locatie voor sommige veronderstellingen en gevolgen na te gaan. Echter is de distributie over de beschrijvings- en rationale-attributen bijna gelijk. In totaal werden 30% van de veronderstellingen en gevolgen geplaatst bij het beschrijvings-attribuut en 31% bij het rationale-attribuut.

Verder werden de veronderstellingen meestal geplaatst in beschrijvings-attributen, omdat de veronderstellingen eerder beschreven wat het element is. Terwijl de gevolgen bijna gelijk werden verdeeld tussen het beschrijvings-attribuut en het rationale-attribuut van de elementen. Enkel gevolgen bij een inbreuk in de diverse microservices werden zowel geplaatst in de beschrijvings- en rationale-attributen. De andere drie categorieën van de gevolgen werden enkel geplaatst bij de rationale-attributen.

Ten slotte bleek uit de eerste evaluatie test dat de beschrijvings- en rationale-attributen veel tekst hadden, en onoverzichtelijk werden. De onoverzichtelijke structuur van de tekst in de beschrijvings- en rationale-attributen hebben een negatieve impact op de leesbaarheid van zowel de tekst als het gehele .tml-bestand. In secties 8.1, 8.2, 8.3, 8.4 worden er mogelijke uitbreidingen voorgesteld voor de event-georiënteerde DSL. De uitbreidingen zouden de DSL overzichtelijker maken, echter blijft de verantwoordelijkheid bij de eindgebruiker van de DSL, zie sectie 8.5.

7.3 Functionaliteit, geldigheid en operationeelheid

De tweede test evalueert, zoals de titel van de sectie aangeeft, de functionaliteit, geldigheid en operationeelheid. Het proces voor desbetreffende test is als volgt. Ten eerste wordt de test opgezet, door de criteria te beschrijven om aan de doelstelling te voldoen, en door een simulatie van de geschiedenis van SecureDrop te construeren. Vervolgens de uitwerking, door de test twee keer uit te voeren. De eerste keer met de DSL en de tweede keer met SPARTA + ADR's. Ten slotte worden de resultaten van de test besproken. Figuur 7.2 illustreert het evaluatieproces aan de hand van een *workflow* diagram.



Figuur 7.2: *Workflow* diagram van de tweede evaluatie.

7.3.1 Opzet evaluatie

Om de tweede evaluatie test op te zetten wordt er beschreven wanneer de doelstellingen voldaan zijn. De eerste doelstelling, functionaliteit, zal voldaan zijn bij volgende twee criteria. Ten eerste moet het gewenste DFD uitgedrukt kunnen worden op basis van de DSL-statements. Ten tweede moet ieder aspect van het DFD gewijzigd kunnen worden. De tweede doelstelling, geldigheid, zal voldaan zijn als elke reeks van DSL-statements resulteert in een geldig DFD. Een DFD is ongeldig wanneer er een *flow* bestaat zonder bron of bestemming. Of wanneer er een *flow* van een entiteit naar zichzelf verwijst. De derde doelstelling, operationeelheid, zal voldaan zijn wanneer het documenteren in de taal soepel en efficiënt verlopen, zonder de gebruiker van de taal het gevoel te geven dat documenteren in de taal het modelleren verhindert en zonder na het modelleren nog te gaan moet documenteren.

Voor de tweede evaluatie test wordt opnieuw het threat model van SecureDrop gebruikt. Maar in plaats van het volledige DFD van SecureDrop na te bouwen, zoals bij de eerste evaluatie, wordt bij deze test de geschiedenis van SecureDrop gesimuleerd. De test start met het bouwen van een gelimiteerde initiële versie van SecureDrop. Vervolgens worden zeven wijzigingen aangebracht aan de initiële versie zodat alle aspecten van de DSL getest kunnen worden. Voor de doelstelling operationeelheid wordt de test twee keer uitgevoerd. De eerste uitvoering gebruikt de DSL voor het bouwen en documenteren van de versies van het DFD. En de tweede uitvoering gebruikt de applicatie SPARTA [7] om de versies van het DFD te maken en ADRS's om te documenteren. Om een betere vergelijking te kunnen maken tussen de beiden uitvoeringen van de test, is er op voorhand geredeneerd over de initiële versie en de zeven wijzigingen, zodat beide uitvoeringen van de test consistente wijzigen doorvoeren. Iedere versie van het DFD is op voorhand uitgewerkt op papier. Bovendien heeft iedere papieren versie ook een oplijsting van kernwoorden gekregen. Die kernwoorden worden gebruikt tijdens het maken documentaties van de verscheidene versies. Het is vanzelfsprekend dat het schrijven van de documentatie bij het tweede keer vlotter gaat. Dit komt omdat er al eens over de zinnen is nagedacht en onrechtstreeks onthouden is wat er in de eerste uitvoering van de test geschreven werd. Echter bevoordeelt dit de tweede uitvoering van de test met SPARTA en ADR's, en niet de DSL voorgesteld door deze thesis. Tijdens het interpreteren van de resultaten wordt er rekening gehouden met de volgorde van het uitvoeren van de test.

7.3.2 Uitwerking

Voor de uitwerking van de evaluatie zijn beide uitvoeringen van de test enig sinds anders. Bij de eerste uitvoering wordt de DSL gebruikt om de geschiedenis te simuleren, tijdens het modelleren van elke versie wordt gelijktijdig gedocumenteerd. Bij de tweede uitvoering worden eerst alle verschillende versies gemodelleerd in SPARTA. En pas na het modelleren worden de ADR's opgesteld van de verschillende versies. Zie figuur 7.2 ter referentie.

Zoals eerder vermeld simuleert deze test een mogelijke geschiedenis van het SecureDrop system. Hiervoor is er een initiële versie vanwaar er wordt gestart. Vervolgens zijn er zeven versies die de initiële versie wijzigen zodat alle aspecten van de DSL getest worden. De wijzigingen zijn: het toevoegen van het *airgapped viewing station*, toevoegen van *firewalls*, het verwijderen van bepaalde *firewalls*, toevoegen van het *monitoring* systeem, een *firewall* opsplitsen, de opgesplitste *firewalls* samenvoegen en de externe services extern maken. De zeven wijzigen resulteren uiteindelijk in het originele DFD van SecureDrop.

Initiële versie

De initiële versie is gebaseerd op het originele DFD van SecureDrop. Enkel het *monitoring* systeem en het *airgapped viewing station* zijn verwijderd. Bijkomend worden de "apt" externe services gezien als een proces die een *datastore* aanspreekt, in plaats van een externe entiteit. In bijlage A.2.1 zijn de DSL-statements, het resultaat van de DSL-statements en het DFD gemaakt in SPARTA te zien.

Toevoegen airgapped viewing station

Bij de eerste wijziging van de initiële versie wordt het *airgapped viewing station* toegevoegd, samen met de nodige processen, *flows* en *trust boundaries*. Ook het *publishing station* wordt geïntroduceerd. In bijlage A.2.2 zijn de DSL-statements, het resultaat van de DSL-statements, de ADR en het DFD gemaakt in SPARTA.

Toevoegen firewalls

Bij de tweede wijziging worden er drie nieuwe processen toegevoegd. De drie processen representeren *firewalls*. Eén *firewall* die het verkeer controleert naar de SecureDrop applicatie, één voor het administrator *workstation* en één voor het journalist *workstation*. Teven eens wordt er een nieuwe *trust boundary* geïntroduceerd. In bijlage A.2.3 zijn de DSL-statements, het resultaat van de DSL-statements, de ADR en het DFD gemaakt in SPARTA.

Verwijderen van de admin en journalist firewalls

Bij de derde wijziging worden twee *firewalls* verwijderd die bij de vorige wijziging werden geïntroduceerd. In de ADR is te lezen waarom het toevoegen van de twee *firewalls* een probleem was in eerste instantie. In bijlage A.2.4 zijn de DSL-statements, het resultaat van de DSL-statements, de ADR en het DFD gemaakt in SPARTA.

Toevoegen monitoring systeem

Bij de vierde wijziging wordt het monitoring system van het originele DFD toegevoegd. In bijlage A.2.5 zijn de DSL-statements, het resultaat van de DSL-statements, de ADR en het DFD gemaakt in SPARTA.

Opsplitsen firewall

Bij de vijfde wijziging wordt de *Dedicated Hardware Firewall* opgesplitst in twee nieuwe *firewalls*. Eén voor de SecureDrop applicatie en één voor het monitoringsysteem. In bijlage A.2.6 zijn de DSL-statements, het resultaat van de DSL-statements, de ADR en het DFD gemaakt in SPARTA.

Samenvoegen firewall

Bij de zesde wijziging wordt de nieuwe *firewall* van de vorige wijziging terug verwijderd. Visueel lijkt het of de *firewall* van het monitoringsysteem niet meer bestaat, echter is die samengevoegd met de *firewall* van de applicatie. Dit resulteert in hetzelfde DFD als het DFD na de vierde wijziging. In bijlage A.2.7 zijn de DSL-statements, het resultaat van de DSL-statements, de ADR en het DFD gemaakt in SPARTA.

Extern maken externe services

Bij de zevende, en laatste, wijziging worden de externe "apt" processen vervangen door externe entiteiten. Dit resulteert in het originele DFD van SecureDrop. In bijlage A.2.8 zijn de DSL-statements, het resultaat van de DSL-statements, de ADR en het DFD gemaakt in SPARTA.

7.3.3 Resultaten

Uit de evaluatie bleek dat de verschillende versies van het DFD voorgesteld konden worden. Hieruit bleek dat de doelstelling functionaliteit voldaan was. Tijdens de eerste uitvoering van de test werd er tot de conclusie gekomen dat er aan de DSL geen operaties ontbraken om de verschillende versies van het DFD op te stellen. Echter kan er niet uitgesloten worden dat in de toekomst andere gebruikers van de DSL verlangen naar meer complexere operaties die de taal niet aanbiedt, zoals: het mergen van meer dan twee elementen, het splitsen van een element in meer dan twee elementen, en decompositie maken van een proces, etc.

Vervolgens bleek ook uit de evaluatie dat elke reeks van DSL-statements resulteerde in een geldig DFD. Per nieuw DSL-statement dat tijdens de uitvoering van de test werd geschreven, werd er gecontroleerd of de tot dan huidige reeks DSL-statement resulteerde in een geldig DFD. Bijkomend werd bij deze test elke operatie van de DSL gebruikt, zodat alle operaties getest werden. Hieruit bleek dat de doelstelling geldigheid ook voldaan was. Bovendien kan uit de implementatie bewezen worden dat elke reeks DSL-statements resulteerde in een geldig DFD. Het bewijs gaat als volgt. Ten eerste, operaties die geen *flow* manipuleren kunnen niet resulteren in een ongeldig DFD. Ten tweede, operaties die de ID, naam en beschrijving van een *flow* manipuleren en de *flow* verwijderen zullen ook

niet resulteren in een ongeldig DFD, omdat deze de bron en bestemming van de *flow* niet wijzigen. Volgende operaties wijzigen de bron en bestemming van een *flow*:

- Toevoegen: Bij het toevoegen wordt er gecontroleerd dat de bron en bestemming niet hetzelfde zijn. Eveneens is het verplicht om zowel een bron en bestemming mee te geven.
- Verwijderen entiteit: Bij het verwijderen van een entiteit wordt de geassocieerde *flows* mee verwijdert, zodat er geen *flow* kan bestaan zonder bron of bestemming.
- Veranderen bron/bestemming: Bij het veranderen van de bron of bestemming van een *flow* wordt er gecontroleerd dat de *flow* vertrekt van een entiteit, niet naar dezelfde entiteit verwijst.
- Samenvoegen: Bij het samenvoegen van twee entiteiten worden de tussenliggende *flows* automatisch verwijdert en de *flows* geassocieerd aan de te *merge* entiteiten automatisch gekoppeld aan de nieuwe entiteit. Dit zorgt ervoor dat er nooit een *flow* bestaat zonder bron of bestemming en met gelijke bron en bestemming.
- Splitsen: Er wordt gecontroleerd of alle *flows* manueel gewijzigd zijn, aan de hand van de *flow-fix-statements*, bij een splits operatie. Dit zorgt ervoor dat er nooit een *flow* bestaat zonder bron of bestemming en met gelijke bron en bestemming.

Tijdens het uitvoeren van deze test werden de gewenste functionaliteiten van de operaties, die hierboven opgelijst worden, bevestigd. Door het feit dat de aparte operaties niet resulteren in een ongeldig DFD, zal de combinatie van DSL-statement eveneens niet resulteren in een ongeldig DFD.

Ten slotte bleken beide uitvoeringen van de testen in grote lijnen even lang te duren, wat suggereert dat het documenteren van de voorafgemaakte beslissingen tijdens het modelleren, het modelleren zelf niet verhindert. Bijkomend moest er na het modelleren niet meer worden gedocumenteerd. Hieruit bleek dat de doelstelling operationeelheid was voldaan. Echter had het werken met de DSL ook twee nadelen tegenover de tweede uitvoering van de test. Het eerste nadeel was dat het schrijven van de ADR's vlotter ging dan het documenteren in de taal. Omdat de structuur van ADR's, volgens Ahmeti et al [15], lijkt op een invulformulier met drie vragen: wat is de beslissing, wat was de context en wat zijn de gevolgen. Terwijl documenteren met de DSL meer open is. In sectie 8.3 wordt er een oplossing geïntroduceerd die hierop inspeelt. Echter geeft de bevinding, dat het documenteren in de taal minder vlot ging tegen over documenteren in de ADR's, een vertekend beeld omdat het documenteren in de ADR's tijdens de tweede uitvoering van de test gebeurde. Dit zorgt ervoor dat het makkelijker was om te documenteren, omdat de documentatie van de eerste uitvoering van de tetst onrechtstreeks werd onthouden. Het tweede nadeel is dat werken met de DSL minder gebruiksvriendelijk is dan werken met een *drag-n-drop* applicatie, zoals SPARTA. Dit komt omdat de DSL enkele zwakke punten heeft die besproken worden in sectie 8.6. Daarentegen, heeft werken met de DSL eveneens een voordeel. Omdat de DSL toelaat om direct de redenen van een bepaalde actie te documenteren, is de documentatie naar mening beter. Bij de tweede uitvoering van de test wordt er pas gedocumenteerd nadat alle versies van het SecureDrop systeem gemodelleerd waren in SPARTA. Dit zorgde ervoor dat het moeilijker was om te herinneren waarom bepaalde acties genomen waren, dus werd het moeilijker om de redenering achter

de acties te documenteren. Omdat er op voorhand van deze evaluatie al is geredeneerd over de verschillende versies van het systeem, was dit geen aanzienlijk probleem. Maar in een echt scenario is dit mogelijk ernstiger.

Hoofdstuk 8

Discussie

In dit hoofdstuk worden bevindingen van de thesis besproken. De eerste vier secties introduceren aanpassingen die de event-georiënteerde DSL potentieel overzichtelijker maakt. Vervolgens wordt er besproken dat de eindgebruiker de verantwoordelijkheid draagt voor het correcte gebruik van de DSL. Nadien wordt de gebruiksvriendelijkheid van TML bekritiseerd aan de hand van enkele nadelen. Aansluitend wordt er besproken welke factoren van het .tml-bestand geanalyseerd kunnen worden, in een zogenaamde *event analyse*. Het laatste deel van het hoofdstuk omvat een discussie rond de flexibiliteit van de geldigheid doelstelling.

8.1 Beschrijvende element attributen

Uit de eerste evaluatie test bleek dat de onoverzichtelijke structuur van de tekst in het beschrijvings-attribuut deels verantwoordelijk is voor een onoverzichtelijk .tml-bestand. Echter kan de DSL aangepast worden om te compenseren voor de onoverzichtelijke eigenschap. Door nieuwe attributen te introduceren, die de eigenschappen van de DFD elementen beschrijven, vermindert de tekst in het beschrijvings-attribuut. Minder tekst in het beschrijvings-attribuut zorgt voor een overzichtelijker .tml-bestand.

Vaak beschrijft het beschrijvings-attribuut welke type data het element bevat, gebruikt of verstuurt. Momenteel is de gebruiker vrij dit te beschrijven hoe hij het zelf wilt. De introductie van een data-attribuut voor de DFD elementen maakt het overbodig om het type data te beschrijven in het beschrijvings-attribuut. Het nieuwe attribuut zou werken op basis van data typen. De taal zou standaard data typen kunnen aanbieden, echter zou de taal ook de mogelijkheid aanbieden om de eindgebruiker zelf data types te laten declareren. De gebruiker creëert eerst de data typen, alvorens de data typen met de elementen geassocieerd kunnen worden. Voorbeelden van type data zijn: inloggegevens, persoonlijke gegevens, public/private sleutels, bankgegevens, gevoelige bedrijfsgegevens, etc.

Andere mogelijke attributen, die het beschrijvings-attribuut potentieel overzichtelijker maken, zijn hieronder opgelijst:

- `providesAuthentication`: Het komt vaak voor dat een externe entiteit authenticatie aanbiedt. Door een nieuw optioneel booleaanse attribuut te introduceren, is het overbodig om het te beschrijven in het beschrijvings-attribuut.

- `isWebApp`: Het komt vaak voor dat een proces een web applicatie voorstelt. Door een nieuw optioneel booleaanse attribuut te introduceren, is het overbodig om het te beschrijven in het beschrijvings-attribuut.
- `isSQL`: In de beschrijving van een *datastore* wordt vaak het type databank beschreven, dit kan verplaatst worden naar een nieuw optioneel booleaans attribuut.
- `protocol`: In de beschrijving van *flows* wordt vaak het gebruikte protocol beschreven, dit kan verplaatst worden naar een nieuw optioneel attribuut.
- `publicNetwork`: Gevoelige gegevens onbeschermd transporteren over een publiek netwerk kan gevaarlijk zijn. Vaak wordt in de beschrijving van *flows* gespecificeerd als de *flow* data transporteert in een publiek netwerk, dit kan verplaatst worden naar een nieuw optioneel booleaans attribuut.
- `isDirect`: Communicatie tussen entiteiten kan zowel direct als indirect zijn. Door een nieuwe optioneele booleaans attribuut te introduceren, moet de informatie met betrekking tot het type communicatie niet in het beschrijvings-attribuut geplaatst worden.

De zes voorbeelden dienen ter inspiratie voor potentieel nieuwe attributen, resulterend in verbeterde leesbaarheid van het beschrijvings-attribuut. Echter zijn er talloze alternatieven. Voor beveiligings-gerelateerde attributen stellen Sion et al. [40] een alternatieve oplossing voor. In volgende sectie 8.2 gaat hier dieper op in.

8.2 Beveiligings-gerelateerde elementen

Zoals in vorige sectie vermeld stellen Sion et al. [40] een alternatief voor, om de beveiligings-gerelateerde attributen te vervangen. Zij breiden het assortiment van DFD elementen uit met een reeks beveiligings-gerelateerde elementen, zoals bedreigingen en tegenmaatregelen. De beveiligings-gerelateerde elementen worden samen met het DFD gebruikt bij het eliciteren van bedreigen.

Het toevoegen van desbetreffende beveiligings-gerelateerde elementen aan de DSL zou mogelijks extra voordelen hebben. In de huidige status van de DSL moet informatie, met betrekking tot bedreigingen en tegenmaatregelen, in beschrijvings- of rationale-attributen geplaatst worden. Als de DSL de functionaliteit ondersteunt om beveiligings-gerelateerde elementen te introduceren, kan de informatie verplaatst worden naar desbetreffende element. Dit resulteert in minder tekst in de beschrijvings- en rationale-attributen, waardoor het volledige .tml-bestand overzichtelijker wordt. De nieuwe beveiligings-gerelateerde elementen moeten verwijzen naar de elementen die beïnvloed zijn en naar de DSL-statements die het resultaat zijn van het beveiliging-gerelateerde element in kwestie.

8.3 Verbetering rationale-attribuut

De onoverzichtelijke eigenschap van de taal kan eveneens gecompenseerd worden door het rationale-attribuut te wijzigen. Momenteel is de gebruiker vrij om zelf de structuur te kiezen in het rationale-attribuut. Echter was dat attribuut bedoeld om architecturale

beslissingen te documenteren. Daarom is het een goed idee om de structuur van ADR over te nemen. Volgens Ahmeti et al [15] hebben ADR's de volgende drie attributen: beslissing, context en consequenties. In het beslissings-attribuut wordt de inhoud van de beslissing beschreven. Het context-attribuut beschrijft de reden achter de beslissing en de invloedrijke factoren. Ten slotte beschrijft het consequentie-attribuut de gevolgen van de beslissing. Door het rationale-attribuut om te vormen naar deze structuur zou de documentatie overzichtelijker zijn.

8.4 Documentatie element

De event-georiënteerde DSL laat gebruikers toe om documentatie *inline* of aan de hand van een groepering-element toe te voegen. Dit zorgt ervoor dat de documentatie verspreid is over het onoverzichtelijke .tml-bestand. Door een documentatie-element te introduceren, kan de documentatie verzameld en ordelijk georganiseerd worden. Dit resulteert in een overzichtelijk .tml-bestand. De documentatie elementen moeten verwijzen naar de DSL-statements in kwestie.

De gebruiker van de DSL zou de documentatie elementen kunnen gebruiken op een ongewenste manier. Waarbij de gebruiker eerst alles modelleert en dan pas zal documenteren, zoals bij de tweede uitvoering van de tweede evaluatie test, zie figuur 7.2 ter referentie. Dan kan de gebruiker van de taal niet genieten van de voordelen die gepaard komen met de DSL. Echter zou de DSL toelaten om documentatie elementen op te stellen tijdens het modelleren. Maar de eindverantwoordelijkheid ligt steeds bij de eindgebruiker om het documentatie element te gebruiken op de correcte manier.

8.5 Verantwoordelijkheid eindgebruiker

De aanpassingen om het event-georiënteerde DSL overzichtelijker te maken zou de eindgebruiker helpen om tijdens het modelleren te documenteren. Belangrijk om te weten is dat de verantwoordelijkheid om DSL op een correcte manier te gebruiken nog steeds bij de eindgebruiker ligt. Ter illustratie, de eindgebruiker van de taal wordt niet gedwongen om alles in het beschrijvings-attribuut of het rationale-attribuut te zetten. Sterker nog, de gebruiker kan ervoor kiezen om niet te documenteren in de DSL.

Er werd gekozen om de beschrijvings- en rationale-attributen niet verplicht te maken. Dit zorgt voor betere flexibiliteit van de taal, maar legt de verantwoordelijkheid bij de gebruiker. In dit geval kan het zijn dat de gebruiker er voor kiest om de taal niet te gebruiken zoals het bedoeld is. In dat geval kan de gebruiker niet genieten van de voordelen die gepaard gaan met de DSL. Een mogelijke oplossing zou zijn om de beschrijvings- en rationale-attributen verplicht te maken. Maar dit forceert de gebruiker om mogelijks nutteloze tekst te schrijven en zou de productiviteit verlagen.

8.6 Gebruiksvriendelijkheid TML

Uit het resultaat van de eerste evaluatie, sectie 7.2, bleek dat de taal een onoverzichtelijke eigenschap heeft. Hiervoor werden reeds enkele oplossingen aangeboden. Bovendien zorgt de onoverzichtelijkheid ervoor dat de taal minder gebruiksvriendelijk is voor zowel het analyseren van de code als de documentatie.

Uit het resultaat van de tweede evaluatie, sectie 7.3, bleek dat Threat Modeling Language minder gebruiksvriendelijk is tegenover het gebruiken van een *drag-n-drop* applicatie, zoals SPARTA. Dit komt doordat de taal drie nadelen heeft. Ten eerste, is het werken met de DSL minder gebruiksvriendelijk aangezien de gebruiker bekend moet zijn met de syntax van de taal. Eveneens voor de ontwikkelaar van de taal is dit niet eenvoudig. Door de syntax nauwkeurig te evalueren op gebruiksvriendelijkheid, zou de syntax mogelijks herzien kunnen worden om ze consistenter en eenvoudiger te maken. Ten tweede, zorgt het gebruik van de ID's ook voor minder gebruiksvriendelijkheid. Door de elementen een beschrijvende ID te geven met een consistente structuur, is het makkelijker om de ID's te onthouden. Echter blijft het moeilijk om alle ID's van al de elementen van buiten te kennen, en moet er regelmatig gezocht worden naar ID's van specifieke elementen. Dit zou opgelost kunnen worden door een hulpmiddel dat de gebruiker helpt bij het onthouden van de ID's. Enkele voorbeelden zijn: een *autocomplete* voor de ID's, een functie in de IDE met de oplijsting van de ID's, de mogelijkheid om de elementen van de grafische representatie van het DFD klikbaar te maken zodat de ID's gevisualiseerd kunnen worden. Ten derde, heeft de DSL-statements omvormen naar een DFD met Graphviz ook een nadeel. Om geneste *trust boundaries* te tonen, moeten de *trust boundaries* omgevormd worden naar geneste sub-grafen. Echter interpreteert Graphviz een sub-graaf als een geneste sub-graaf, wanneer de buitenste sub-graaf dezelfde elementen heeft als de geneste sub-graaf, plus een extra element. Als dit niet het geval is interpreteert Graphviz het DFD anders en leidt dit tot non-deterministische resultaten. Dat maakt het moeilijk om bij fouten, die gemaakt zijn door het gebruik van de DSL, de oplossing te vinden. Dit zou opgelost kunnen worden door te werken met een andere technologie dan Graphviz.

8.7 Event analyse

Nadat een applicatie in de DSL wordt gemodelleerd, en de redenatie achter design beslissingen in de DSL gedocumenteerd, kan er vereist zijn de DSL code en de documentatie te analyseren. Desbetreffende analyse wordt in deze thesis "event analyse" genoemd. De *event analyse* is mogelijk door het feit dat het .tml-bestand de geschiedenis van het DFD representeert, als een sequentiële lijst van alle acties op het DFD. Aan de hand van het bestand kan de geschiedenis van het DFD geanalyseerd worden. Een mogelijke analyse is het pad van de initiële versie naar de huidige versie van het DFD, traceren en analyseren. Er kunnen ook diverse factoren van de geschiedenis van het DFD geanalyseerd worden, zoals:

- Welke operaties er gebeurt zijn op een element.
- Welke documentatie is er gelinkt aan een element. Dit is de documentatie van alle DSL-statements die het element manipuleren.

- Het meest gewijzigde element. Dit is het element dat voorkomt in de meeste DSL-statements.
- Het aantal DSL-statements per operatie.
- Het aantal groeperingen per groepering type. Dit kan gebruikt worden bij een vergelijking tussen de verhouding van *desing* en *countermeasure* groeperingen.
- Welke statements elkaar neutraliseerde. Bijvoorbeeld een statement dat een element introduceert en een statement dat hetzelfde element verwijdert. Hetzelfde met de splits en samenvoeg operaties, etc.
- Het aantal statement dat gedocumenteerd wordt, zowel *inline* als via een groepering element, in percentage.
- Voor de uitbreiding, voorgesteld in sectie 8.3, welke elementen gebruiken hetzelfde data type.

Dit zijn een reeks factoren die geanalyseerd kunnen worden omdat de DSL toelaat om de geschiedenis van heet DFD te bewaren. Hiernaast zijn er nog talloze alternatieven factoren die geanalyseerd kunnen worden.

8.8 Flexibiliteit geldigheid doelstelling

De event-georiënteerde DSL interpreteert een DFD als ongeldig, bij volgende twee criteria. Ten eerste mag er geen *flow* bestaan zonder bron en/of bestemming. Ten tweede moet een *flow* startende van één entiteit eindigen in een andere entiteit. Ondanks potentieel ongebruikelijke kenmerken van een DFD, interpreteert de *DSL-parser* deze als geldig, zolang de DFD voldoet aan de geldigheid doelstelling. Volgende kenmerken van DFD's zijn ongebruikelijk, maar eveneens geldig:

- Losse entiteit: Dit zijn entiteiten die niet verbonden zijn met andere entiteiten aan de hand van een *flow*.
- Lege *trust boundaries*.
- *Flows* tussen externe entiteiten en *datastores*: Het is logisch dat een proces geplaatst wordt tussen de connecties van een externe entiteit en een *datastore*.

Er werd gekozen voor gebruiksvriendelijkheid en flexibiliteit in plaats van het afkeuren van de ongebruikelijke kenmerken. Ter illustratie, in het geval dat losse entiteiten niet mogen bestaan: dan moet bij het aanmaken van een nieuwe entiteit direct een *flow* naar een andere entiteit gedefinieerd worden. Dit resulteert in een complexere syntax.

Hoofdstuk 9

Conclusie

Deze thesis zocht een antwoord op volgende onderzoeksvraag:
Hoe kan het modelleren tijdens threat modeling worden ondersteund om de design beslissingen en hun bijhorende redenering te documenteren zonder het modelleren te hinderen?
Een event-georiënteerde DSL, genaamd **Threat Modeling Language**, wordt als een mogelijke oplossing aangeboden, geïmplementeerd aan de hand van technologieën zoals Eclipse Xtext en Graphviz. Met de DSL kan een DFD geconstrueerd worden, en kunnen ontwerp beslissingen en hun bijhorende redenering gedocumenteerd en gekoppeld worden aan desbetreffende DSL-statements. Uit het toepassen van de DSL op een evaluatie-case bleek dat de doelstellingen, functionaliteit, geldigheid, documentatie van rationale en operationeelheid, voldaan waren. Echter bleek de DSL een onoverzichtelijke en minder gebruiksvriendelijke eigenschap te hebben. Hiervoor werden er in hoofdstuk 8 enkele oplossingen aangeboden: nieuwe beschrijvende element attributen, nieuwe beveiliging-gerelateerde elementen, een verbetering van de structuur van het rationale-attribuut en een nieuw documentatie element. Maar de eindverantwoordelijkheid ligt steeds bij de eindgebruiker om taal te gebruiken zoals het bedoeld is. Deze thesis stelt de event-georiënteerde DSL voor als een *proof-of-concept*, waarbij het mogelijk is om de geschiedenis van het DFD te analyseren in een *event analyse*. Met de uitbreidingen besproken in hoofdstuk 8 kan de taal verder ontwikkeld worden naar een volwassen DSL.

Dankwoord

Ik zou graag mijn begeleiders, Dr. ir. L. Sion en Ir. J. Bellemans, bedanken voor hun begeleiding en adviezen gedurende dit traject. Daarnaast ben ik mijn vrienden en familie dankbaar voor hun steun en aanmoediging.

Bijlage A

A.1 Evaluatie test 1

A.1.1 .tml bestand

```
design initialDesign "The initial desing of the Secure Drop
   application"{
        add process firewall name "SecureDrop Dedicated Hardware
           Firewall"
        dsc "A pfSense-based which is  is a free and open source
           firewall and router that also features unified threat
           management, load balancing, multi WAN, and more.It is
           not compromised by malware."

        add process aptApp name "APT" dsc "Advanced package tool,
            or APT, is a free-software user interface that works
           with core libraries to handle the installation and
           removal of software."
        add process ntpApp name "NPT" dsc "The Network Time
           Protocol (NTP) is a networking protocol for clock
           synchronization between computer systems."
        add process dnsApp name "DNS" dsc "The Domain Name System
            (DNS) is a hierarchical and distributed name service
           that provides a naming system for computers."
        add process ossecClient name "OSSEC Client" dsc "OSSEC (
           Open Source HIDS SECurity) is a free, open-source host-
           based intrusion detection system (HIDS). It performs
           log analysis, integrity checking, Windows registry
           monitoring, rootkit detection, time-based alerting, and
            active response." "
                - An attacker with access to the ossec user can:
                   Add, view, modify, and delete the log files,
                   and in doing so send inaccurate information to
                   the Monitor Server and the admin.
        "
        add process secureDropApp name "SecureDrop Application"
           dsc "The application runs on the Ubuntu OS and is not
           compromised by malware.
```

```
                - The server sees the plaintext codename, used as
                   the login identifier, of every source.
                - The server sees the plaintext submissions of
                   every source.
                - server sees the plaintext communication between
                   journalists and their sources.
                - The server stores and (optional) TLS private
                   key and certificate (if HTTPS is enabled on the
                   source interface)
                - The server stores hashes of codenames, created
                   with scrypt and randomly-generated salts.
                - The server stores journalist password hashes,
                   created with script and randomly-generated
                   salts, as well as TOTP seeds.
                - The server stores only encrypted submissions
                   and communication on disk.
                - The server stores a GPG key for each source,
                   with the sources codename as the passphrase.
                - The server may store plaintext submissions in
                   memory for at most 24 hours.
                - The server stores sanitized Tor logs, created
                   using the SafeLogging option, for the Source
                   Interface, the Journalist Interface, and SSH.
                - The server stores both access and error logs
                   for the Journalist Interface.
                - The server stores connection history and audit
                   logs for the admin.
                - The server can connect to the Monitor Server
                   using an SSH key and a passphrase.
" "
        - If the Application Server is compromised, the system
           user the attacker has control over defines what kind of
            information the attacker will be able to view and what
            kind of actions the attacker can perform.
        - If the Application Server is seized, the attacker will
           be able to view any and all unencrypted files on the
           server. An attacker will be able to modify any and all
           files on the server. This includes all files in use by
           the SecureDrop Application. If the server is seized
           while it is powered on, the attacker can also analyze
           any plaintext information that resides in RAM. The
           attacker can also tamper with the hardware.
"
        add datastore DB dsc "A password-protected database"
        add flow f1app secureDropApp -> DB
        add flow f2app DB -> secureDropApp
        add flow f3app firewall -> secureDropApp "The server sees
            the plaintext codename, used as the login identifier,
           of every source."
        add flow f4app aptApp -> firewall
```

```
        add flow f5app ntpApp -> firewall
        add flow f6app dnsApp -> firewall
        add boundary app name "SecureDrop Application Server":
           aptApp ntpApp dnsApp ossecClient secureDropApp DB
        add boundary secDrop name "SecureDrop Area": aptApp
           ntpApp dnsApp ossecClient secureDropApp DB firewall
}

design monitoring "The monitoring server keeps track of the
   Application Server and sends out alerts if theres a problem.
    It runs the processes on the Ubuntu OS.
                - If the Monitor Server is compromised, the
                   system user the attacker has control over
                   defines what kind of information the attacker
                   will be able to view and what kind of actions
                   the attacker can perform.
                - If the Monitor Server is seized, the attacker
                   will be able to view any and all unencrypted
                   files on the server. This includes all files in
                    use by OSSEC. If the server is seized while it
                    is powered on, the attacker can also analyze
                   any plaintext information that resides in RAM.
                   The attacker can also tamper with the hardware.
                - If the Monitor Server is no longer online or
                   tampered with, this will have an effect on the
                   quantity and accuracy of notifications sent to
                   admins or journalists.
"{
        add process aptMon name "APT" dsc "Advanced package tool,
            or APT, is a free-software user interface that works
           with core libraries to handle the installation and
           removal of software."
        add process ntpMon name "NTP" dsc "The Network Time
           Protocol (NTP) is a networking protocol for clock
           synchronization between computer systems."
        add process dnsMon name "DNS" dsc "The Domain Name System
            (DNS) is a hierarchical and distributed name service
           that provides a naming system for computers."
        add process ossecServer name "OSSEC Server" dsc "Sends
           alerts, some via email." "The server stores the OSSEC
           Alert Public Key the OSSEC alerts are encrypted:
                - The server stores the plaintext alerts on disk,
                    data may also reside in RAM.
                - The server stores the OSSEC Alert Public Key
                   the OSSEC alerts are encrypted to.
                - The server stores plaintext credentials for the
                    SMTP relay used to send OSSEC alerts.
                - The server stores the email address the
                   encrypted OSSEC alerts are sent to.
```

```
                - The server stores connection history and audit
                   logs for the admin.
                - The server stores OSSEC and Procmail logs on
                   disk.
                - The server can connect to the Application
                   Server using an SSH key and a passphrase.
                - An attacker with access to the ossec user can:
                        - View all ossec logs and alerts on disk.
                        - Modify the ossec configuration.
                        - Send (or suppress) emails to
                           administrators and journalists.
"
        add process postfix name "Postfix"
        add flow f1mon ossecClient -> ossecServer
        add flow f2mon ossecServer -> postfix
        add flow f3mon postfix -> firewall
        add flow f4mon aptMon -> firewall
        add flow f5mon ntpMon -> firewall
        add flow f6mon dnsMon -> firewall
        add boundary mon name "SecureDrop Monitoring Server":
           aptMon ntpMon dnsMon ossecServer postfix
        change secDrop add aptMon
        change secDrop add ntpMon
        change secDrop add dnsMon
        change secDrop add ossecServer
        change secDrop add postfix
}

design docSource "The document have to be supplied to the
   application by the source.
                - Use of Tor Browser will leave traces that can
                   be discovered through a forensic analysis of
                   the sources property following either a
                   compromise or physical seizure. Unless the
                   compromise or seizure happens while the source
                   is submitting documents to SecureDrop, the
                   traces will not include information about sites
                    visited or actions performed in the browser.
                - Use of Tails with a persistent volume will
                   leave traces on the device the operating system
                    was installed on. Unless the compromise or
                   seizure happens while the source is submitting
                   documents to SecureDrop, or using the
                   persistent volume, the traces will not include
                   information about sites visited or actions
                   performed in the browser or on the system.
                - An attacker with access to the sources
                   codename can:
                        - Show that the source has visited the
                           SecureDrop site, but not necessarily
```

```
                                submitted anything.
                            - Upload new documents or submit messages
                                .
                            - Communicate with the journalist as that
                                source.
                            - See any replies from journalists that
                                the source has not yet deleted.
                    - A global adversary may be able to link a source
                        to a specific SecureDrop server.
                    - A global adversary may be able to link a source
                        to a specific journalist.
                    - A global adversary may be able to forge an SSL
                        certificate and use it to spoof an
                        organizations HTTPS Landing Page, thereby
                        tricking the source into visiting a fake
                        SecureDrop site.
                    - A random person can attempt to get sensitive
                        information from a SecureDrop users browser
                        session, such as the sources codename.
"{
        add process sourceTorBrowser name "Tor Browser" dsc "Runs
            the Tor browser on the Tails OS and is not infected by
            Malware. It obtains an authentic copy of Tails and Tor
            Browser." "Remain anonymous, even against a forensic
           attacker"
        add flow sourceappf dsc "Uses the Tor protocol"
           sourceTorBrowser -> firewall "The server sees the
           plaintext submissions of every source."
        add boundary sourceArea name "Source Area":
           sourceTorBrowser
}

design admin "The admin workstation runs the Tails OS and is not
   infected by malware. The workstation is used to monitor and
   work on the SecureDrop Application remotely.
                - To access the Journalist Interface, the
                    Application Server, or the Monitor Server, the
                    attacker needs to obtain the admins login
                    credentials and the admins two-factor
                    authentication device. Unless the attacker has
                    physical access to the servers, the attacker
                    will also need to obtain the onion service
                    values for the Interface and the servers. This
                    information is stored in a password-protected
                    database in a persistent volume on the
                    admins Tails device. The volume is protected
                    by a passphrase. If the admins two-factor
                    authentication device is a mobile phone, this
                    will also be protected by a passphrase.
```

```
- An attacker with access to the persistent
   volume on the admins Tails device can:
        - Add, modify, and delete files on the
           volume.
        - Access the onion service values used by
           the Interfaces and the servers.
        - Access SSH keys and passphrases for the
           Application Server and the Monitor
           Server.
        - Access the GPG key and passphrase for
           the encrypted OSSEC email alerts.
        - Access the credentials for the account
           the encrypt alerts are sent to.
        - Access the admins personal GPG
           public key, if stored there.
- An attacker with admin access to the Journalist
    Interface can:
        - Add, modify, and delete journalist
           users.
        - Change the codenames associated with
           sources within the Interface.
        - Download, but not decrypt, submissions.
        - Communicate with sources.
        - Delete one or more submissions.
        - Delete one or more sources, which
           destroys all communication with that
           source and prevents the source from
           ever logging back in with that codename
           .
- An attacker with admin access to the
   Application Server can:
        - Add, modify, and delete software,
           configurations, and other files
        - See all HTTP requests made by the
           source, the admin, and the journalist.
        - See the plaintext codename of a source
           as they are logging in.
        - See the plaintext communication between
           a source and a journalist as it
           happens.
        - See the stored list of hashed codenames
           .
        - Access the GPG public key used to
           encrypt communications between a
           journalist and a source.
        - Download stored, encrypted submissions
           and replies from the journalists.
        - Decrypt replies from the journalists if
           the sources codename, and thus the
           passphrase, is known.
```

```
                        - Analyze any plaintext information that
                          resides in RAM, which may include
                          plaintext of submissions made within
                          the past 24 hours.
                        - Review logs stored on the system.
                        - Access the Monitor Server.
                - An attacker with admin access to the Monitor
                  Server can:
                        - Add, modify, and delete software,
                          configurations, and other files.
                        - Change the SMTP relay, email address,
                          and GPG key used for OSSEC alerts.
                        - Analyze any plaintext information that
                          resides in RAM.
                        - Review logs stored on the system.
                        - Trigger arbitrary commands to be
                          executed by the OSSEC agent user, which
                          , assuming the attacker is able to
                          escalate privileges, may affect the
                          Application Server.
                - Physical Seizure of the Admins Property Can
                  Achieveaccess any stored, decrypted
                  documents taken off the Secure Viewing Station.
                - A physical seizure of, and access to, the
                  admins Tails persistent volume, password
                  database, and two-factor authentication device
                  will allow the attacker to access both servers
                  and the Journalist Interface.
"{
        add process ssh dsc "Secure Shell Protocol (SSH) is a
           cryptographic network protocol for operating network
           services securely over an unsecured network. The RSA
           (4096-bit GPG and SSH keys) are valid. It obtain
           authentic copies of Tails. The two-factor
           authentication device used with the workstation are not
            compromised by malware."
        add process tor
        add flow adminf1 dsc "Is clearnet" ssh -> tor
        add flow adminf2 dsc "Uses the Tor protocol" tor ->
           firewall
        add boundary adminarea name "Admin Workstation": ssh tor
}

design journalist "The journalist uses the SecureDrop system via
   his/hers workstation.
                - To access the Journalist Interface, the
                  attacker needs to obtain the journalists
                  login credentials and the journalists two-
                  factor authentication device or seed. Unless
                  the attacker has physical access to the server,
```

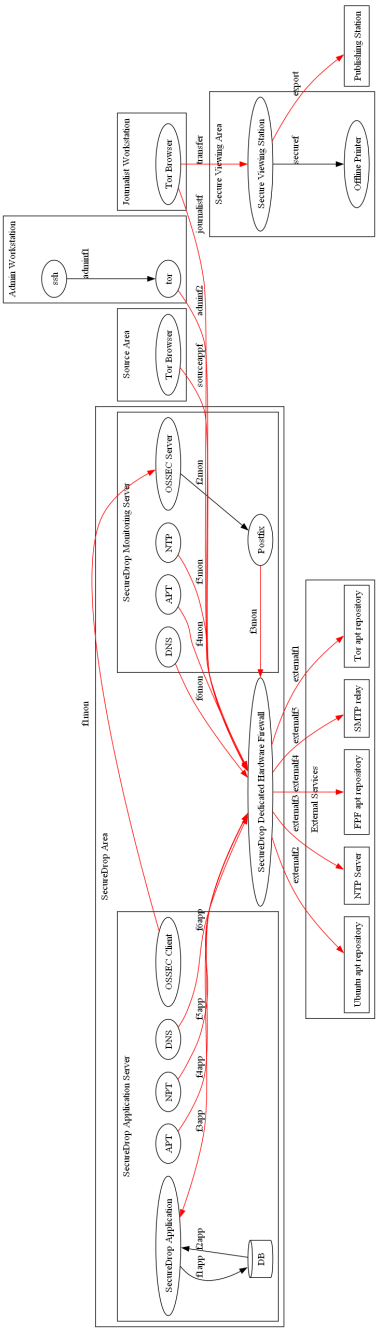
```
                    the attacker will also need to obtain the
                   onion service value for the Interface. This
                   information is stored in a password-protected
                   database in a persistent volume on the
                   journalists Tails device. The volume is
                   protected by a passphrase. If the
                   journalists two-factor authentication device
                    is a mobile phone, this will also be protected
                    by a passphrase.
                - An attacker with access to the persistent
                   volume on the journalists Tails device can:
                        - Add, modify, and delete files on the
                           volume.
                        - Access the onion service values used by
                            the Journalist Interface.
                        - Access SSH keys and passphrases for the
                            Application Server and the Monitor
                           Server
                - An attacker with journalist access to the
                   Journalist Interface can:
                        - Change the codenames associated with
                           sources within the interface.
                        - Download, but not decrypt, submissions.
                        - Delete one or more submissions.
                        - Communicate with sources.
                        - If the journalist has admin privileges
                           on SecureDrop, they can create new
                           journalist accounts
                - Physical Seizure of the Journalists
                   Property Can Achieveaccess any stored,
                   decrypted documents taken off the Secure
                   Viewing Station.
                -A physical seizure of, and access to, the
                   journalists Tails persistent volume,
                   password database, and two-factor
                   authentication device will allow the attacker
                   to access the Journalist Interface.
"{
        add process journalistTorBrowser name "Tor Browser" dsc "
           Runs the Tor browser on the Tails OS and is not
           infected by malware. It obtain authentic copies of
           Tails. The two-factor authentication device used with
           the workstation are not compromised by malware."
        add flow journalistf dsc "Uses the Tor protocol"
           journalistTorBrowser -> firewall
        add boundary journalistarea name "Journalist Workstation"
           : journalistTorBrowser
}
```

```
design secureView "A physically-secured and air-gapped laptop
   running the Tails operating system from a USB stick, that
   journalists use to decrypt and view submitted documents. A
   random person can submit malicious documents, e.g. malware that
    will attempt to compromise the Secure Viewing Station.
"{
        add process secureViewStation name "Secure Viewing
           Station" dsc "The computer and the Tails device are not
            compromised by malware."
        add process printer name "Offline Printer"
        add flow securef dsc "The printer is connected by cabel
           and must not have not have WiFi or Bluetooth."
           secureViewStation -> printer
        add flow transfer dsc "The transfer of the encrypted file
            happens with a physical USB" journalistTorBrowser ->
           secureViewStation

        add boundary securearea name "Secure Viewing Area":
           secureViewStation printer

        add external entity publish name "Publishing Station" dsc
            "The PC used to use the decrypted documents and
           publish articles using the documents"
        add flow export dsc "The export of the decrypted file
           happens with a physical USB" secureViewStation ->
           publish
}

design externals "The external package uses to operate the
   SecureDrop application." {
        add external entity torApt name "Tor apt repository" dsc
           "Collection of software packages to easily install and
           update Tor through the Advanced Package Tool (APT)
           system."
        add external entity ubuntuApt name "Ubuntu apt repository
           " dsc "Collection of software packages to easily
           install and update Ubuntu through the Advanced Package
           Tool (APT) system."
        add external entity ntpServer name "NTP Server" dsc "A
           network device that uses the Network Time Protocol (NTP
           ) to synchronize the clocks of computers over a network
           ."
        add external entity fpfApt name "FPF apt repository" dsc
           "Collection of software packages  provided by the
           Freedom of the Press Foundation (FPF) to easily install
            and update tools related to privacy through the
           Advanced Package Tool (APT) system."
        add external entity smtprelay name "SMTP relay" dsc "A
           server that transfers email messages from one mail
           server to another."
```

```
        add flow externalf1 firewall -> torApt
        add flow externalf2 firewall -> ubuntuApt
        add flow externalf3 firewall -> ntpServer
        add flow externalf4 firewall -> fpfApt
        add flow externalf5 firewall -> smtprelay
        add boundary externalServ name "External Services":
           torApt ubuntuApt ntpServer fpfApt smtprelay
}
```

A.1.2 Resulterend DFD

Zie figuur A.1.



Figuur A.1: DFD gegenereerd door de DSL-statements van de eerste evaluatie test.

A.2 Evaluatie test 2

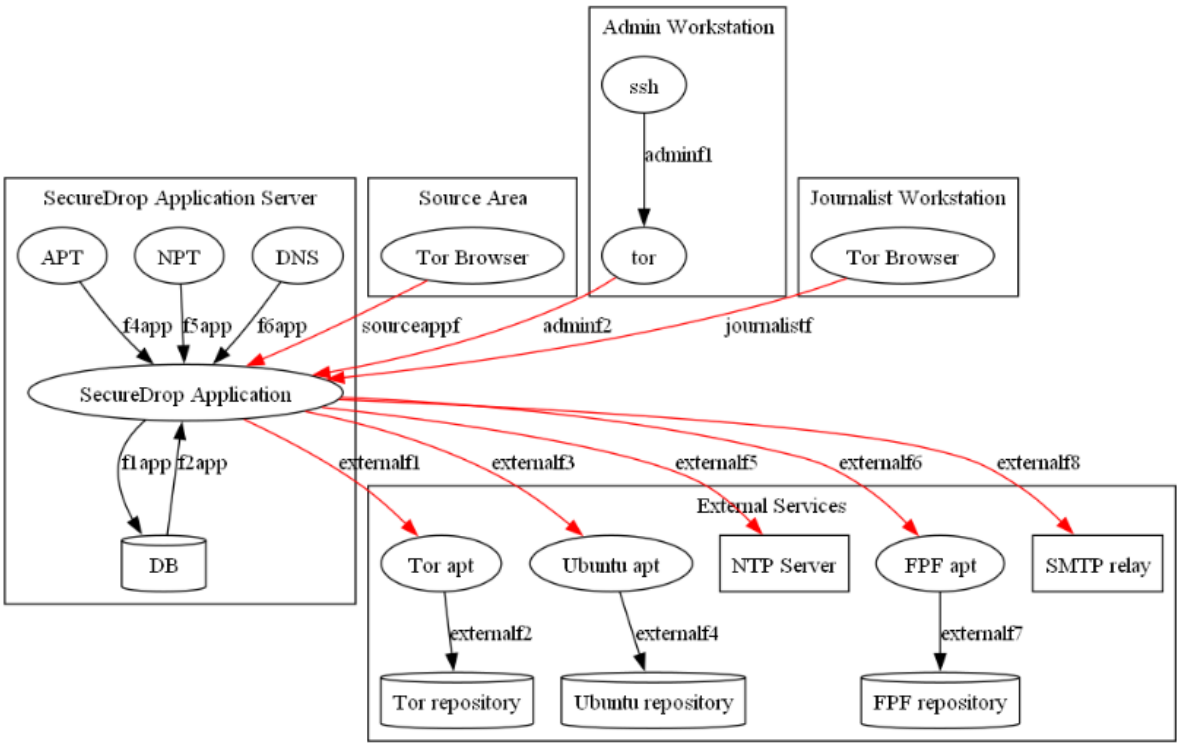
A.2.1 Initiële versie

DSL

```
design SecureDropApplicationDesign "
        SecureDrop is an open-source whistleblower submission
           system that media organizations can use to securely
           accept documents from and communicate with anonymous
           sources.

        Sources and journalists connect to SecureDrop using the
           Tor network. The SecureDrop software is running on
           premises on dedicated infrastructure (two physical
           servers and a firewall).
" {
        add process aptApp name "APT" dsc "Advanced package tool,
            or APT, is a free-software user interface that works
           with core libraries to handle the installation and
           removal of software."
        add process ntpApp name "NPT" dsc "The Network Time
           Protocol (NTP) is a networking protocol for clock
           synchronization between computer systems."
        add process dnsApp name "DNS" dsc "The Domain Name System
            (DNS) is a hierarchical and distributed name service
           that provides a naming system for computers."
        add process secureDropApp name "SecureDrop Application"
           dsc "The application runs on the Ubuntu OS"
        add datastore DB dsc "A password-protected database"
        add flow f1app secureDropApp -> DB
        add flow f2app DB -> secureDropApp
        add flow f4app aptApp -> secureDropApp
        add flow f5app ntpApp -> secureDropApp
        add flow f6app dnsApp -> secureDropApp
        add boundary app name "SecureDrop Application Server":
           aptApp ntpApp dnsApp secureDropApp DB

        add process sourceTorBrowser name "Tor Browser" dsc "Runs
            the Tor browser on the Tails OS and is not infected by
            Malware." "The document have to be supplied to the
           application by the source."
        add flow sourceappf dsc "Uses the Tor protocol"
           sourceTorBrowser -> secureDropApp
        add boundary sourceArea name "Source Area":
           sourceTorBrowser

        add process ssh dsc "Secure Shell Protocol (SSH) is a
           cryptographic network protocol for operating network
           services securely over an unsecured network. The RSA
```

```
   (4096-bit GPG and SSH keys) are valid"
add process tor
add flow adminf1 dsc "Is clearnet" ssh -> tor
add flow adminf2 dsc "Uses the Tor protocol" tor ->
   secureDropApp
add boundary adminarea name "Admin Workstation": ssh tor
   "The admin workstation runs the Tails OS and is not
   infected by malware. The workstation is used to monitor
    and work on the SecureDrop Application remotely."

add process journalistTorBrowser name "Tor Browser" dsc "
   Runs the Tor browser on the Tails OS and is not
   infected by malware."
add flow journalistf dsc "Uses the Tor protocol"
   journalistTorBrowser -> secureDropApp
add boundary journalistarea name "Journalist Workstation"
   : journalistTorBrowser "The journalist uses the
   SecureDrop system via his/hers workstation."

add process torApt name "Tor apt"
add datastore torRepo name "Tor repository" dsc "
   Collection of software packages to easily install and
   update Tor through the Advanced Package Tool (APT)
   system."
add process ubuntuApt name "Ubuntu apt"
add datastore ubuntuRepo name "Ubuntu repository" dsc "
   Collection of software packages to easily install and
   update Ubuntu through the Advanced Package Tool (APT)
   system."
add external entity ntpServer name "NTP Server" dsc "A
   network device that uses the Network Time Protocol (NTP
   ) to synchronize the clocks of computers over a network
   ."
add process fpfApt name "FPF apt"
add datastore fpfRepo name "FPF repository" dsc "
   Collection of software packages  provided by the
   Freedom of the Press Foundation (FPF) to easily install
    and update tools related to privacy through the
   Advanced Package Tool (APT) system."
add external entity smtprelay name "SMTP relay" dsc "A
   server that transfers email messages from one mail
   server to another."
add flow externalf1 secureDropApp -> torApt
add flow externalf2 torApt -> torRepo
add flow externalf3 secureDropApp -> ubuntuApt
add flow externalf4 ubuntuApt -> ubuntuRepo
add flow externalf5 secureDropApp -> ntpServer
add flow externalf6 secureDropApp -> fpfApt
add flow externalf7 fpfApt -> fpfRepo
add flow externalf8 secureDropApp -> smtprelay
```

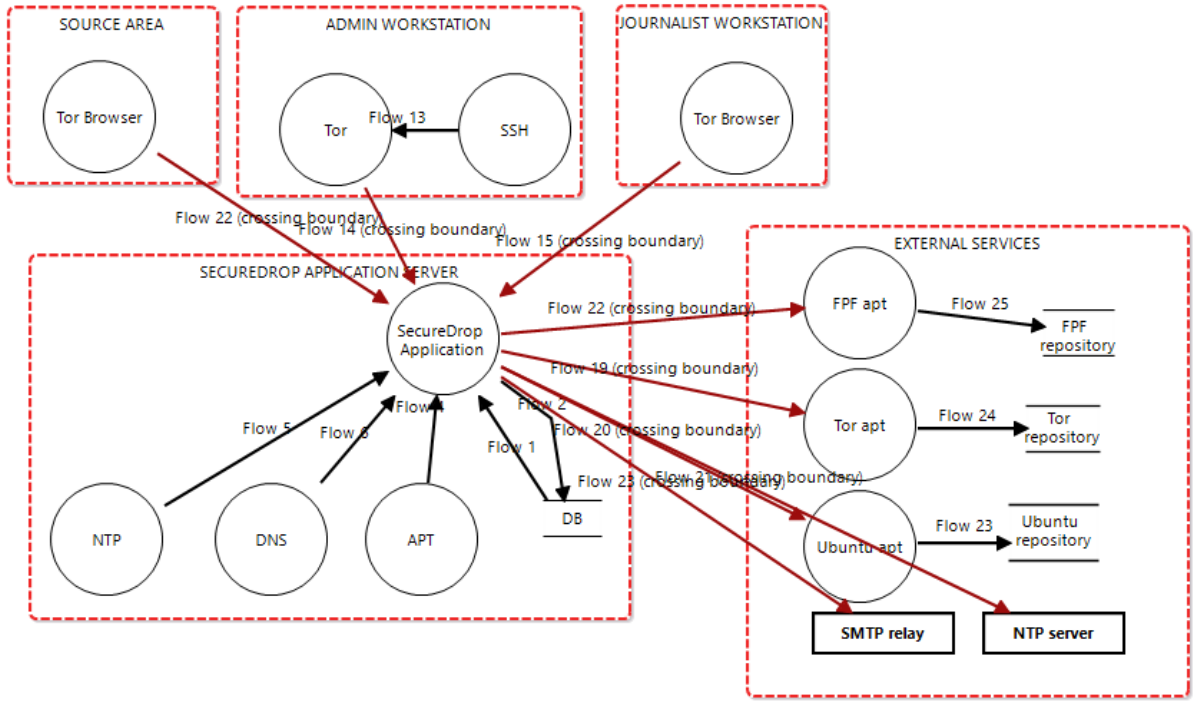
```
        add boundary externalServ name "External Services":
           torRepo ubuntuRepo fpfRepo torApt ubuntuApt ntpServer
           fpfApt smtprelay "The external package uses to operate
           the SecureDrop application."
}
```



Figuur A.2: DFD gegenereerd door de DSL-statements van de initiële versie.

Output DSL

Zie figuur A.2 voor de output van de DSL-statements voor de initiële versie.



Figuur A.3: DFD gemaakt in SPARTA van de initiële versie.

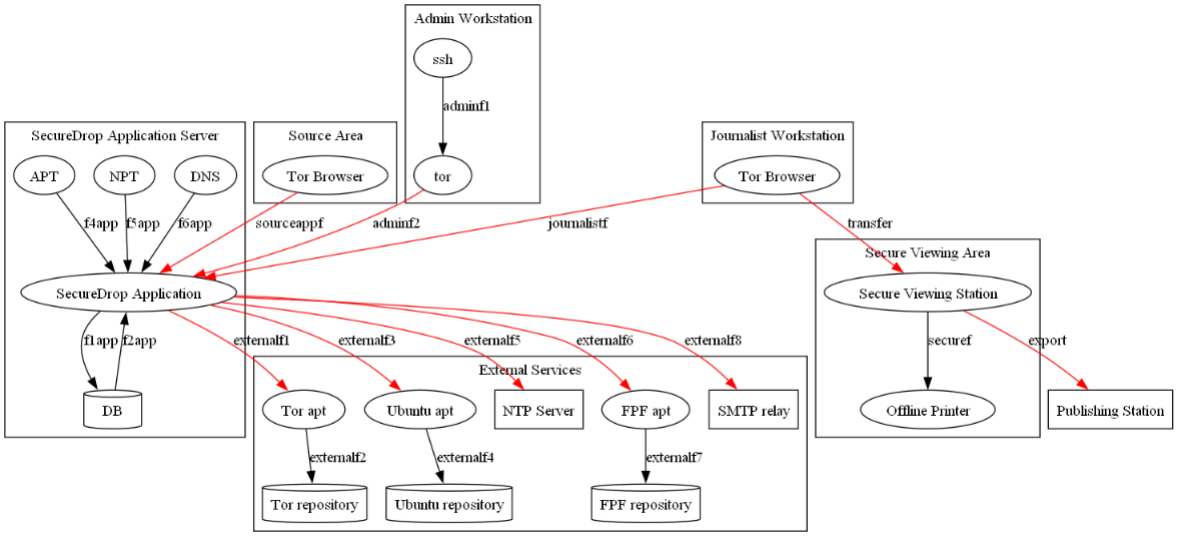
SPARTA DFD

Zie figuur A.3 voor het DFD van de initiële versie, nagemaakt in SPARTA.

A.2.2 Toevoegen airgapped viewing station

DSL

```
design AirgappedViewingStation "
        When the journalist downloads the encrypted document he/
           she can only decrypt it in the airgapped viewing
           station. This is a pc that is connected to nothing. So
           when the document gets decrypted the malware in the
           meta data can only attack the the viewing station. When
            this happens nothing other can get infected. The
           viewing station can easily be wiped and reinstalled.

        Decypting the document in a pc that is connected to the
           network will possibly affect in the infection of the
           whole network. To prevent this we opt for this solution
           .

        The result of this is that an extra pc is needed only for
            this reason. The journalist has to go to the viewing
           station each time they want to open a new document
           which affect prodouctivity.
" {
        add process secureViewStation name "Secure Viewing
           Station"
        add process printer name "Offline Printer"
        add flow securef dsc "The printer is connected by cabel
           and must not have not have WiFi or Bluetooth."
           secureViewStation -> printer
        add flow transfer dsc "The transfer of the encrypted file
            happens with a physical USB" journalistTorBrowser ->
           secureViewStation
        add boundary securearea name "Secure Viewing Area":
           secureViewStation printer "A physically-secured and air
           -gapped laptop running the Tails operating system from
           a USB stick, that journalists use to decrypt and view
           submitted documents."

        add external entity publish name "Publishing Station" dsc
            "The PC used to use the decrypted documents and
           publish articles using the documents"
        add flow export dsc "The export of the decrypted file
           happens with a physical USB" secureViewStation ->
           publish
}
```



Figuur A.4: DFD gegenereerd door de DSL-statements van de versie met de eerste wijziging.

Output DSL

Zie figuur A.4 voor de output van de DSL-statement van de eerste wijziging.

ADR 1: Airgapped Viewing station

Decision

To mitigate the risk of network infection, journalists will only decrypt encrypted documents on an airgapped viewing station. This station is isolated from any network connections. By decrypting documents in this controlled environment, any potential malware contained within the document's metadata can only impact the viewing station itself. Furthermore, the viewing station can be easily wiped and reinstalled if necessary.

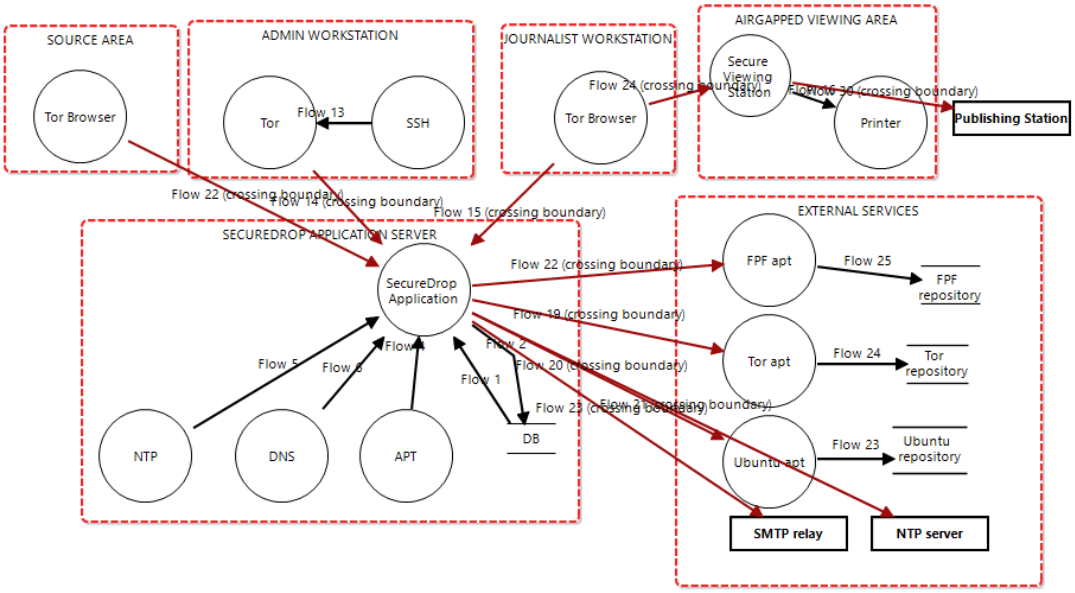
Context

Decrypting documents on a network-connected PC poses a significant risk of infecting the entire network. To be safe against this potential threat, we have chosen to implement this solution.

Consequences

As a result of this decision, an additional PC dedicated only to this purpose is required. Journalists will need to physically access the viewing station each time they wish to open a new document, potentially impacting productivity.

Figuur A.5: ADR airgapped viewing station, geschreven in Markdown



Figuur A.6: DFD gemaakt in SPARTA van de van de versie met de eerste wijziging.

ADR

Zie figuur A.5 voor de ADR van de eerste wijziging.

SPARTA DFD

Zie figuur A.6 voor het DFD van de eerste wijziging gemaakt in SPARTA.

A.2.3 Toevoegen firewalls

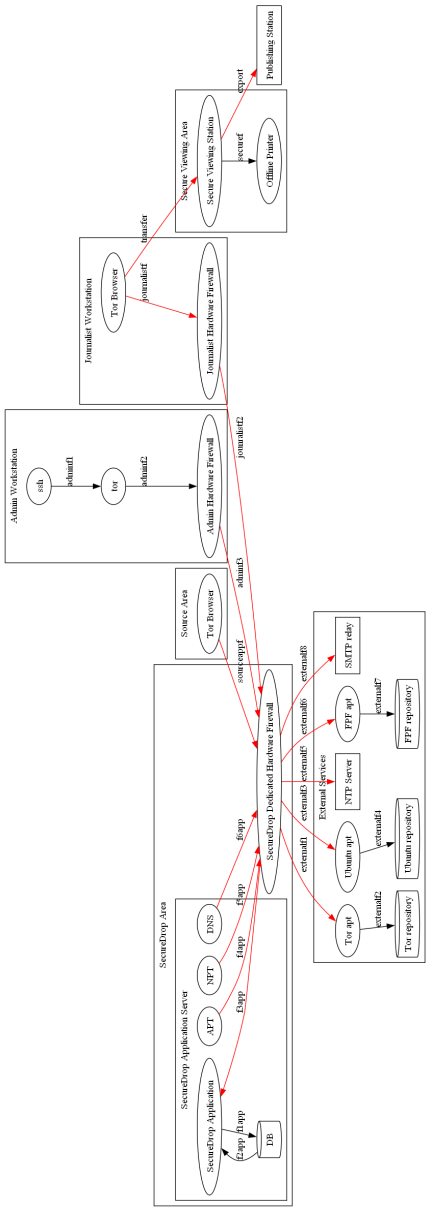
DSL

```
countermeasure Firewalls "
        We opt to add a firewall in the SecureDrop area so that
           we can control all incoming traffic.
        We opt to add a firewall in the admin area so that we can
            control all the outgoing traffic to the SecureDrop
           area.
        We opt to add a firewall in the journalist area so that
           we can control all the incomming traffic.

        Addig the firewall at those places result in better
           controlablity of the traffic wich makes the application
            secure to that adversaries can not act malicious.

        This decision highly impact the performance beceause each
            traffic package has to be scanned. Adding those
           expensive devices drives the cost.
" {
        add process firewall name "SecureDrop Dedicated Hardware
           Firewall"
        dsc "A pfSense-based which is  is a free and open source
           firewall and router that also features unified threat
           management, load balancing, multi WAN, and more."
        add flow f3app firewall -> secureDropApp
        change f4app destination firewall
        change f5app destination firewall
        change f6app destination firewall
        add boundary secDrop name "SecureDrop Area": aptApp
           ntpApp dnsApp secureDropApp DB firewall

        change sourceappf destination firewall

        add process firewallAdmin name "Admin Hardware Firewall"
        change adminf2 destination firewallAdmin
        add flow adminf3 firewallAdmin -> firewall
        change adminarea add firewallAdmin

        add process firewallJournalist name "Journalist Hardware
           Firewall"
        change journalistf destination firewallJournalist
        add flow jounralistf2 firewallJournalist -> firewall
        change journalistarea add firewallJournalist

        change externalf1 source firewall
        change externalf3 source firewall
        change externalf5 source firewall
        change externalf6 source firewall
```

```
        change externalf8 source firewall
}
```

Output DSL

Zie figuur A.7 voor de output van de DSL-statement van de tweede wijziging.



Figuur A.7: DFD gegenereerd door de DSL-statements van de versie met de tweede wijziging.

ADR 2: Firewalls

Decision

We have decided to implement firewalls in the following areas to enhance security and control traffic:

- SecureDrop area: A firewall will be added to manage all incoming traffic.
- Admin area: A firewall will be added to control all outgoing traffic to the SecureDrop area.
- Journalist area: A firewall will be added to regulate all incoming traffic.

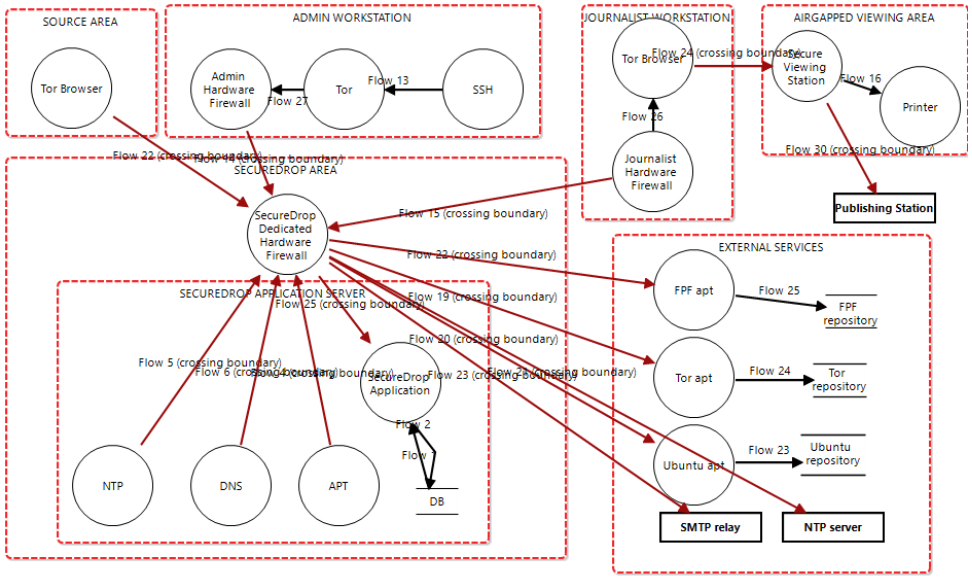
Context

Adding firewalls at these locations will provide better control over network traffic, enhancing security and preventing malicious adversaries.

Consequences

This decision will impact performance, as each traffic packet must be scanned, potentially slowing down the network. Additionally, the cost will increase due to the need for these expensive devices.

Figuur A.8: ADR Firewalls, geschreven in Markdown



Figuur A.9: DFD gemaakt in SPARTA van de van de versie met de tweede wijziging.

ADR

Zie figuur A.8 voor de ADR van de tweede wijziging.

SPARTA DFD

Zie figuur A.9 voor het DFD van de tweede wijziging gemaakt in SPARTA.

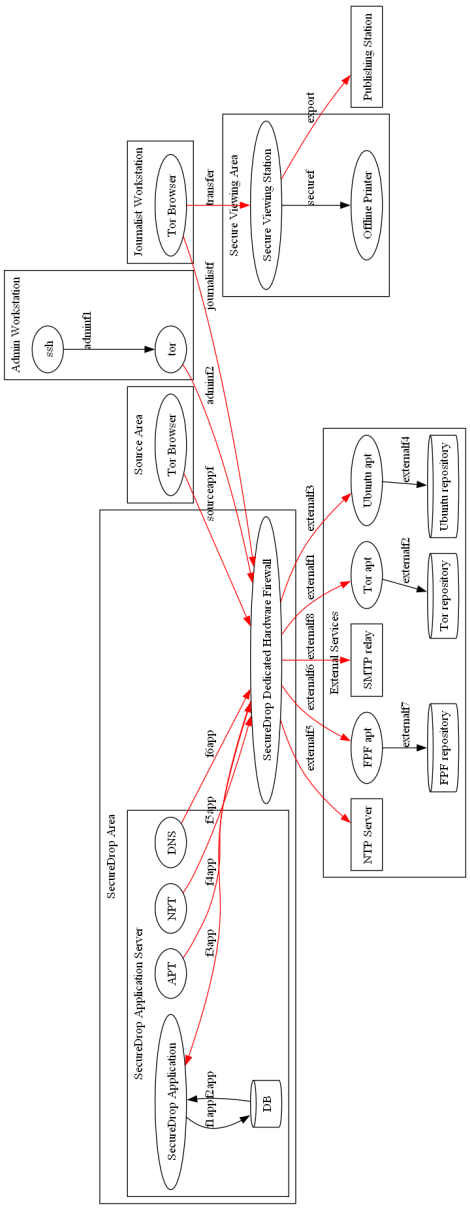
A.2.4 Admin en journalist firewalls verwijderen

DSL

```
refactoring noadminjournalistFirewall "
        We opt to delete the firewall in the admin area so that
           we use to control all the outgoing traffic to the
           SecureDrop area.
        We opt to delete the firewall in the journalist area so
           that we use to control all the incoming traffic.

        Deleting the firewall at those places result in no
           controllability of the traffic which makes the
           application less secure to that adversaries can
           possibly act malicious. But this will severely drive
           down cost. Those firewall where an overkill. Due to the
            fact that we struggle with rising cost this was an
           needed change. Additionally, the throughput of the
           system will be significantly higher.

        This decision highly impact the performance because each
           traffic package does not have to be scanned anymore.
           Removing those expensive devices drives the cost.
" {
        change adminf2 destination firewall
        remove firewallAdmin

        change journalistf destination firewall
        remove firewallJournalist
}
```

Output DSL

Zie figuur A.10 voor de output van de DSL-statement van de derde wijziging.



Figuur A.10: DFD gegenereerd door de DSL-statements van de versie met de derde wijziging.

ADR 3: No Firewalls in admin/journalist area

Decision

We have decided to remove the firewalls in the following areas:

- Admin area, which was used to control all outgoing traffic to the SecureDrop area.
- Journalist area, which was used to control all incoming traffic.

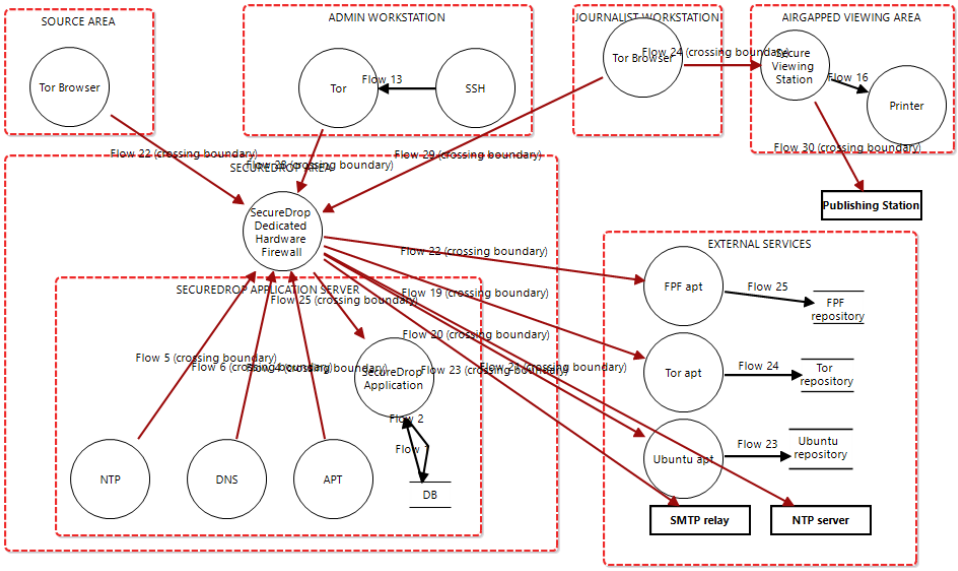
Context

While removing these firewalls reduces the controllability of traffic and potentially makes the application less secure, it will significantly reduce costs. Because we are facing rising costs, this change was necessary. Additionally, system throughput will be significantly improved.

Consequences

This decision will significantly improve system performance since traffic packets will no longer need to be scanned, and it will reduce costs by eliminating the need for expensive firewall devices. However, it also means that there will be less control over network traffic, potentially increasing security risks.

Figuur A.11: ADR No Firewalls in admin/journalist area, geschreven in Markdown.



Figuur A.12: DFD gemaakt in SPARTA van de van de versie met de derde wijziging.

ADR

Zie figuur A.11 voor de ADR van de derde wijziging.

SPARTA DFD

Zie figuur A.12 voor het DFD van de derde wijziging gemaakt in SPARTA.

A.2.5 Toevoegen monitoring systeem

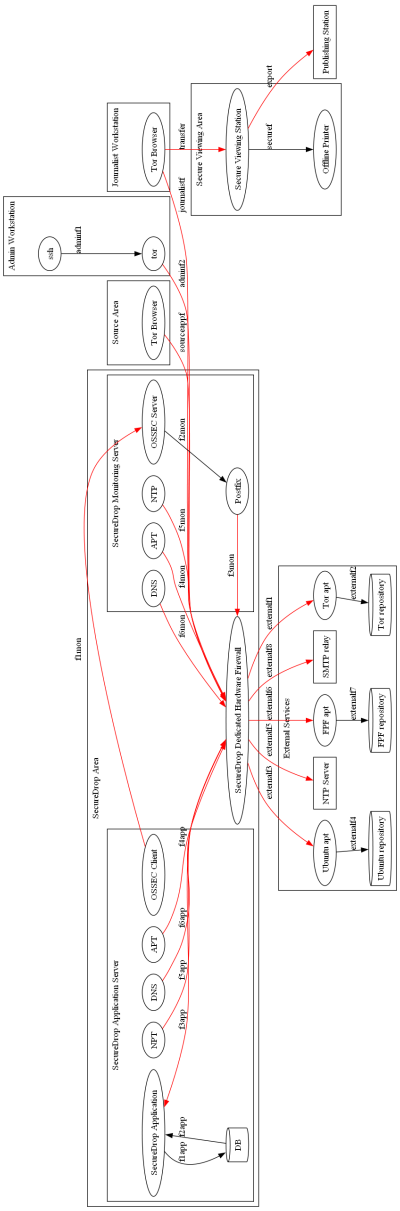
DSL

```
design Monitoring "
        We opt to add a monitoring server to the SecureDrop
           system. Here we run the OSSEC system on so we can
           monitoring every thing running on the application. The
           admin will get full control of the monitoring system
           and can be notified when something is wrong.

        There was need to more controlablity of the application.
           There was no central point to manage the things that
           ware happening with the application.

        This decision highly impact the ability to responed to
           vurnabilities. Fast response time means less down time
           and less affected systems.
" {
        add process aptMon name "APT" dsc "Advanced package tool,
            or APT, is a free-software user interface that works
           with core libraries to handle the installation and
           removal of software."
        add process ntpMon name "NTP" dsc "The Network Time
           Protocol (NTP) is a networking protocol for clock
           synchronization between computer systems."
        add process dnsMon name "DNS" dsc "The Domain Name System
            (DNS) is a hierarchical and distributed name service
           that provides a naming system for computers."
        add process ossecClient name "OSSEC Client" dsc "OSSEC (
           Open Source HIDS SECurity) is a free, open-source host-
           based intrusion detection system (HIDS). It performs
           log analysis, integrity checking, Windows registry
           monitoring, rootkit detection, time-based alerting, and
            active response."
        change app add ossecClient
        add process ossecServer name "OSSEC Server" dsc "Sends
           alerts, some via email." "The server stores the OSSEC
           Alert Public Key the OSSEC alerts are encrypted."
        add process postfix name "Postfix"
        add flow f1mon ossecClient -> ossecServer
        add flow f2mon ossecServer -> postfix
        add flow f3mon postfix -> firewall
        add flow f4mon aptMon -> firewall
        add flow f5mon ntpMon -> firewall
        add flow f6mon dnsMon -> firewall
        add boundary mon name "SecureDrop Monitoring Server":
           aptMon ntpMon dnsMon ossecServer postfix
        change secDrop add ossecClient
        change secDrop add aptMon
```

```
        change secDrop add ntpMon
        change secDrop add dnsMon
        change secDrop add ossecServer
        change secDrop add postfix
}
```

Output DSL

Zie figuur A.13 voor de output van de DSL-statement van de vierde wijziging.



Figuur A.13: DFD gegenereerd door de DSL-statements van de versie met de vierde wijziging

ADR 4 Monitoring

Decision

We have decided to add a monitoring server to the SecureDrop system. This server will run the OSSEC system, allowing monitoring of all application activities. The admin will have full control over the monitoring system and receive notifications if any issues arise.

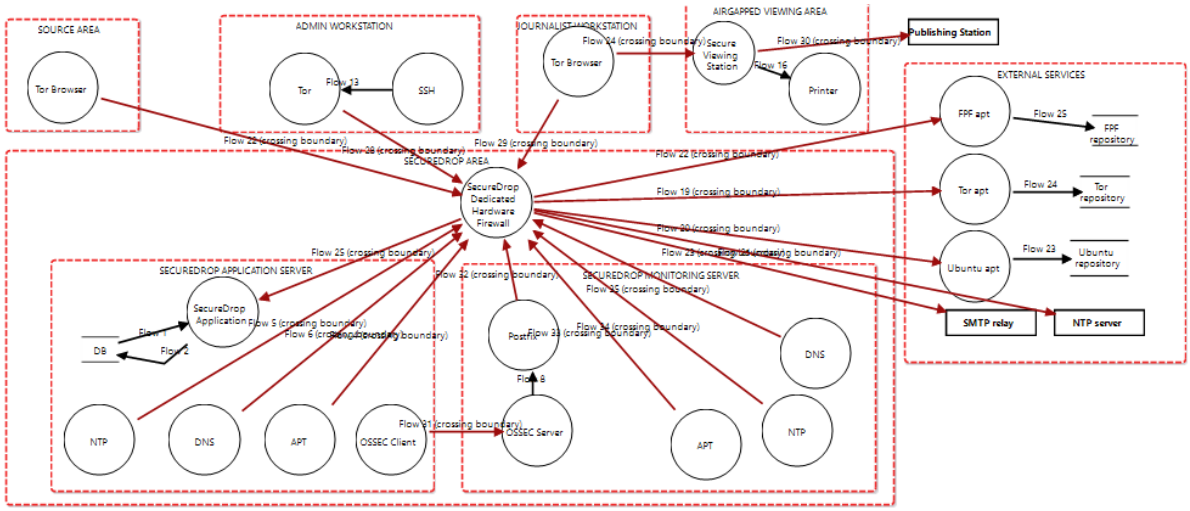
Context

There was a need for greater controllability of the application, as there was no central point to manage and monitor activities within the system.

Consequences

This decision will significantly enhance the ability to respond to vulnerabilities. Faster response times will result in less downtime and fewer affected systems, thereby improving overall security and stability.

Figuur A.14: ADR Monitoring, geschreven in Markdown.



Figuur A.15: ADR gemaakt in SPARTA van de van de versie met de vierde wijziging.

ADR

Zie figuur A.14 voor de ADR van de vierde wijziging.

SPARTA DFD

Zie figuur A.15 voor het DFD van de vierde wijziging gemaakt in SPARTA.

A.2.6 Opsplitsen firewall

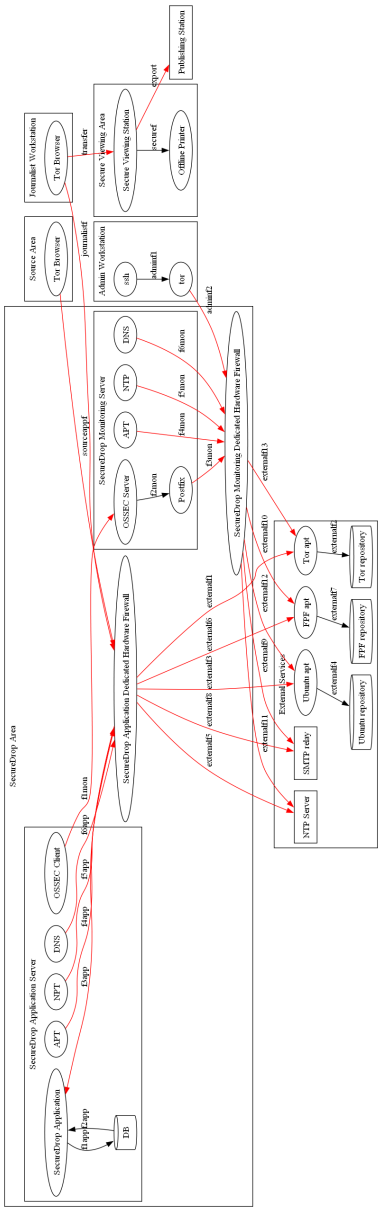
DSL

```
countermeasure SeperateFirewall "
        We opt to add an extra firewall purly for the monitoring
           system.

        There was need to an extra firewall for seperation of
           concern because the main firewalls was supposedly
           overworked.

        This decision impact the ability to track traffic coming
           in the monitoring and the SecureDrop systems. Adding an
            extra firewall will be costly but will not have an
           significant impact on the budget.
" {
        split firewall into process appFirewall name "SecureDrop
           Application Dedicated Hardware Firewall" dsc "A pfSense
           -based which is a free and open source firewall and
           router that also features unified threat management,
           load balancing, multi WAN, and more."
        and process monFirewall name "SecureDrop Monitoring
           Dedicated Hardware Firewall" dsc "A pfSense-based which
            is a free and open source firewall and router that
           also features unified threat management, load balancing
           , multi WAN, and more."
        flows {
                f3app -> appFirewall
                f4app -> appFirewall
                f5app -> appFirewall
                f6app -> appFirewall
                f3mon -> monFirewall
                f4mon -> monFirewall
                f5mon -> monFirewall
                f6mon -> monFirewall
                sourceappf -> appFirewall
                adminf2 -> monFirewall
                journalistf -> appFirewall
                externalf1 -> appFirewall
                externalf3 -> appFirewall
                externalf5 -> appFirewall
                externalf6 -> appFirewall
                externalf8 -> appFirewall
        }
        add flow externalf9 monFirewall -> smtprelay
        add flow externalf10 monFirewall -> fpfApt
        add flow externalf11 monFirewall -> ntpServer
        add flow externalf12 monFirewall -> ubuntuApt
        add flow externalf13 monFirewall -> torApt
```

```
}
```

Output DSL

Zie figuur A.16 voor de output van de DSL-statement van de vijfde wijziging.



Figuur A.16: DFD gegenereerd door de DSL-statements van de versie met de vijfde wijziging.

ADR 5: Seperate Firewalls

Decision

We have decided to add an extra firewall specifically for the monitoring system.

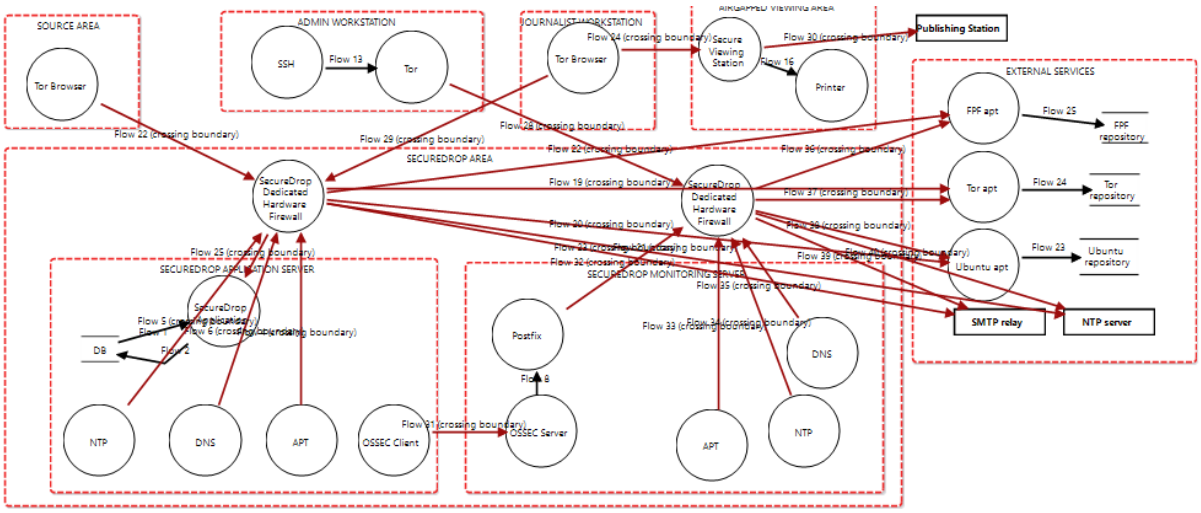
Context

An additional firewall is needed to separate concerns because the main firewalls were reportedly overworked.

Consequences

This decision will enhance the ability to track traffic coming into the monitoring and SecureDrop systems. While adding an extra firewall will incur additional costs, it will not significantly impact the budget.

Figuur A.17: ADR Seperate Firewalls, geschreven in Markdown



Figuur A.18: DFD gemaakt in SPARTA van de van de versie met de vijfde wijziging.

ADR

Zie figuur A.17 voor de ADR van de vijfde wijziging.

SPARTA DFD

Zie figuur A.18 voor het DFD van de vijfde wijziging gemaakt in SPARTA.

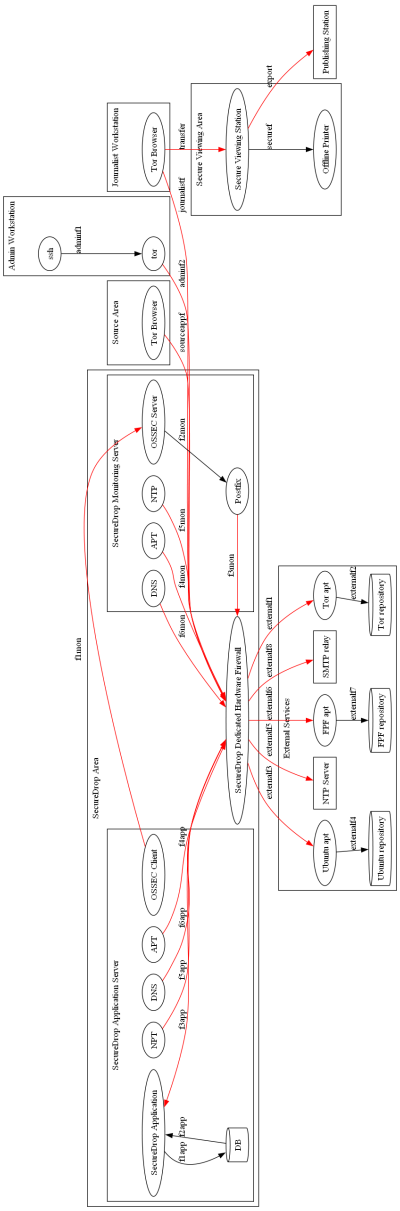
A.2.7 Samenvoegen firewalls

DSL

```
refactoring mergeFirewalls "
        We opt to remove the extra firewall purly for the
           monitoring system.

        There was no need to an extra firewall for seperation of
           concern because the main firewalls was not overworked.
           A junior developer has done the research whether adding
            those firewalls will be beneficial, later his work was
            reviewed, and we concluded that it was not necessary
           to add those firewalls.

        This decision impact the ability to track traffic coming
           in the monitoring and the SecureDrop systems. Removing
           an extra firewall will drive down cost but will not
           have an significant impact on the budget.
"{
        merge appFirewall and monFirewall into process firewall
           name "SecureDrop Dedicated Hardware Firewall"
        dsc "A pfSense-based which is  is a free and open source
           firewall and router that also features unified threat
           management, load balancing, multi WAN, and more."
        change secDrop add firewall

}
```

Output DSL

Zie figuur A.19 voor de output van de DSL-statement van de zesde wijziging.



Figuur A.19: DFD gegenereerd door de DSL-statements van de versie met de zesde wijziging

ADR 6: Delete Seperate Firewalls

Decision

We have decided to remove the extra firewall dedicated to the monitoring system.

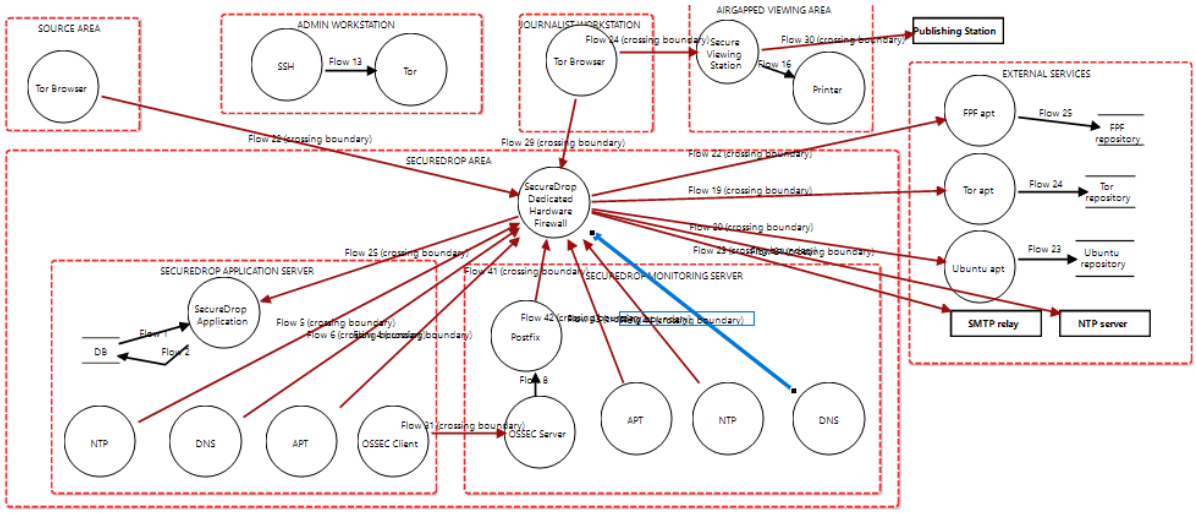
Context

Upon review, it was determined that the main firewalls are not overworked, and there is no need for an additional firewall for separation of concerns. A junior developer conducted research on the necessity of these firewalls and considert them necessary, but aftwer a senior developer conducted his own research and concluded that the extra firewall is unnecessary.

Consequences

This decision will reduce the ability to track traffic specifically entering the monitoring and SecureDrop systems. However, removing the extra firewall will decrease costs, although this reduction will not significantly impact the overall budget.

Figuur A.20: ADR Delete Seperate Firewalls, geschreven in Markdown.



Figuur A.21: ADR gemaakt in SPARTA van de van de versie met de zesde wijziging.

ADR

Zie figuur A.20 voor de ADR van de zesde wijziging.

SPARTA DFD

Zie figuur A.21 voor het DFD van de zesde wijziging gemaakt in SPARTA.

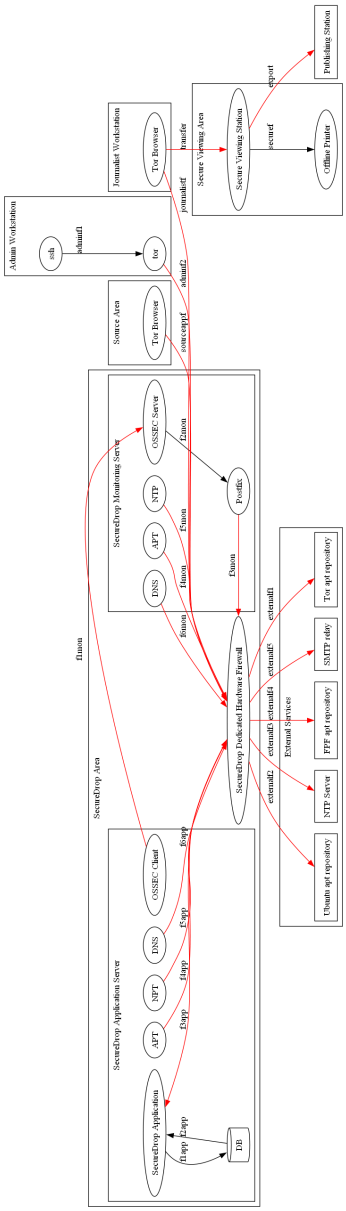
A.2.8 Extern maken externe services

DSL

```
refactoring externalServices "
        We opt to make all the services in the extern services
           trust boundary a external entity.

        There was no need to make the external services processes
            and datastores because we do not operate them. They
           are services provided by thirty parties and we do not
           control them. We only use them to operate the
           SecureDrop system.

        This decision will not impact any design and is purly for
            aesthetic reasons.
"{
        merge fpfApt and fpfRepo into external entity fpfExtern
           name "FPF apt repository" dsc "Collection of software
           packages  provided by the Freedom of the Press
           Foundation (FPF) to easily install and update tools
           related to privacy through the Advanced Package Tool (
           APT) system."
        merge ubuntuApt and ubuntuRepo into external entity
           ubuntuExtern name "Ubuntu apt repository" dsc "
           Collection of software packages to easily install and
           update Ubuntu through the Advanced Package Tool (APT)
           system."
        merge torApt and torRepo into external entity torExtern
           name "Tor apt repository" dsc "Collection of software
           packages to easily install and update Tor through the
           Advanced Package Tool (APT) system."
}
```

Output DSL

Zie figuur A.22 voor de output van de DSL-statement van de laatste wijziging.



Figuur A.22: DFD gegenereerd door de DSL-statements van de versie met de laatste wijziging.

ADR 7: Make external services external

Decision

We have decided to classify all services within the external services trust boundary as external entities.

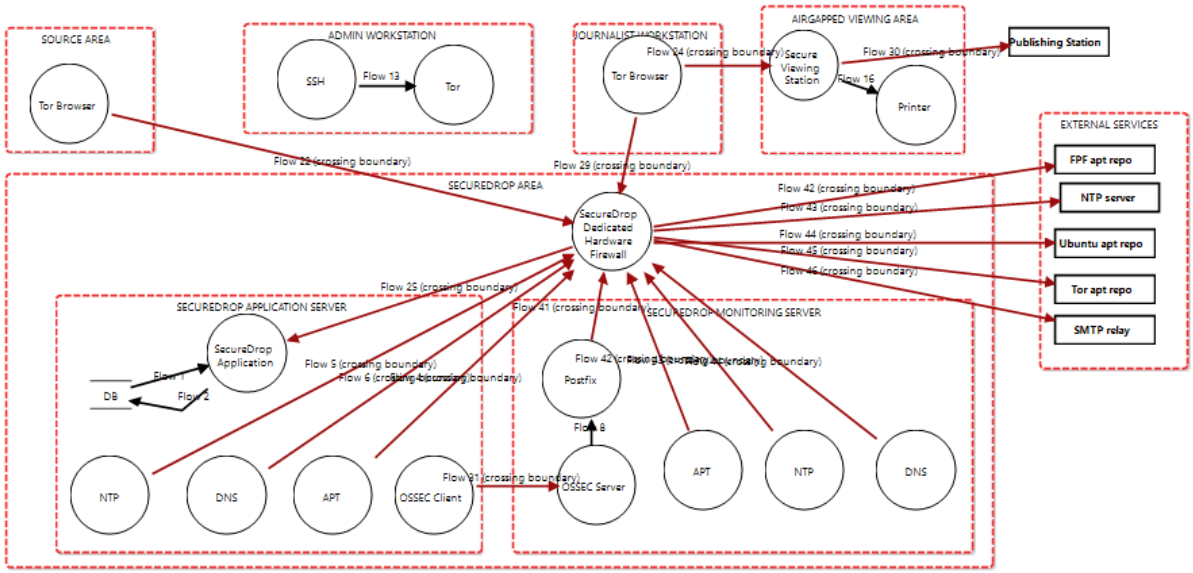
Context

It is unnecessary to manage external services' processes and datastores since we do not operate them. These services are provided by third parties, and we do not have control over them. They are simply utilized to operate the SecureDrop system.

Consequences

This decision will not affect the design of the system and is primarily for aesthetic purposes, clarifying the distinction between internally managed and third-party services.

Figuur A.23: ADR Delete Seperate Firewalls, geschreven in Markdown.



Figuur A.24: ADR gemaakt in SPARTA van de van de versie met de laatste wijziging.

ADR

Zie figuur A.23 voor de ADR van de laatste wijziging.

SPARTA DFD

Zie figuur A.24 voor het DFD van de laatste wijziging gemaakt in SPARTA.

Bibliografie

[1] J. Beerman, D. Berent, Z. Falter, and S. Bhunia, "A review of colonial pipeline ransomware attack," in *2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing Workshops (CCGridW)*. IEEE, 2023, pp. 8–15.

[2] P. Taylor. Nnumber of internet users worldwide from 2005 to 2023. [Online]. Available: https://www.statista.com/statistics/512650/worldwide-connected-devices-amount/

[3] I. Debian, Public Interest. Historical statistics source lines of code. [Online]. Available: https://sources.debian.org/stats/#hist_sloc

[4] N. N. V. Database. Historical statistics vulnerabilities in java. [Online]. Available: https://nvd.nist.gov/vuln/search/statistics?form_type=Basic&results_type=statistics&query=java&search_type=all&isCpeNameSearch=false

[5] A. Petrosyan. Number of connected devices worldwide in 2014 and 2028, by device. [Online]. Available: https://www.statista.com/statistics/273018/number-of-internet-users-worldwide/

[6] Z. Braiterman, A. Shostack, J. Marcil, S. de Vries, I. Michlin, W. Kim, R. Hurlbut, B. SE Schoenfield, F. Scott, M. Coles, C. Romeo, A. Miller, I. Tarandach, A. Douglen, and M. French. Threat modeling manifesto. 02/10/2023. [Online]. Available: https://www.threatmodelingmanifesto.org/

[7] L. Sion, D. Van Landuyt, K. Yskout, and W. Joosen, "Sparta: Security & privacy architecture through risk-driven threat assessment," in *2018 IEEE International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2018, pp. 89–92.

[8] OWASP, "Threat dragon," https://www.threatdragon.com/.

[9] Microsoft, "Microsoft threat modeling tool," https://www.microsoft.com/en-us/download/details.aspx?id=49168.

[10] A. Shostack, *Threat Modeling: Designing for Security*. Wiley, 2014.

[11] K. Yskout, T. Heyman, D. Van Landuyt, L. Sion, K. Wuyts, and W. Joosen, "Threat modeling: from infancy to maturity," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, 2020, pp. 9–12.

[12] K. Wuyts, L. Sion, and W. Joosen, "Linddun go: A lightweight approach to privacy threat modeling," in *2020 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 2020, pp. 302–309.

[13] L. Obiora Nweke and S. Wolthusen, "A review of asset-centric threat modelling approaches," 2020.

[14] I. Alves. (2023) Making your life easier with domain-specific languages. 28/10/2023. [Online]. Available: https://medium.com/wearewaes/making-your-life-easier-with-domain-specific-languages-dsl-1838d351d35

[15] B. Ahmeti, M. Linder, and R. Wohlrab, "Exploring the adoption and effectiveness of architecture decision records in agile software development: An action research study," 2018.

[16] S. Fraser, A. Smotrakov, and M. Rautio, "Threatspec," https://threatspec.org/.

[17] z. Tarandach, "Pytm," https://github.com/izar/pytm.

[18] W. Wideł, S. Hacks, M. Ekstedt, P. Johnson, and R. Lagerström, "The meta attack language-a formal description," *Computers & Security*, vol. 130, p. 103284, 2023.

[19] S. Katsikeas, S. Hacks, P. Johnson, M. Ekstedt, R. Lagerström, J. Jacobsson, M. Wällstedt, and P. Eliasson, "An attack simulation language for the it domain," in *International Workshop on Graphical Models for Security*. Springer, 2020, pp. 67–86.

[20] L. Cerovic, "Stridelang: Creation of a domain-specific threat modeling language using stride, dread and mal," 2022.

[21] S. Hacks, S. Katsikeas, E. Ling, R. Lagerström, and M. Ekstedt, "powerlang: a probabilistic attack simulation language for the power domain," *Energy Informatics*, vol. 3, pp. 1–17, 2020.

[22] E. Rencelj Ling and M. Ekstedt, "Generating threat models and attack graphs based on the iec 61850 system configuration description language," in *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, 2021, pp. 98–103.

[23] S. Hacks and S. Katsikeas, "Towards an ecosystem of domain specific languages for threat modeling," in *International Conference on Advanced Information Systems Engineering*. Springer, 2021, pp. 3–18.

[24] S. Katsikeas, P. Johnsson, S. Hacks, and R. Lagerström, "Vehiclelang: A probabilistic modeling and simulation language for modern vehicle it infrastructures," *Computers & Security*, vol. 117, p. 102705, 2022.

[25] M. Ebrahimi, C. Striessnig, J. C. Triginer, and C. Schmittner, "Identification and verification of attack-tree threat models in connected vehicles," *arXiv preprint arXiv:2212.14435*, 2022.

[26] T. N. Sun, B. Drouot, F. R. Golra, J. Champeau, S. Guerin, L. Le Roux, R. Mazo, C. Teodorov, L. Van Aertryck, and B. L'Hostis, "A domain-specific modeling framework for attack surface modeling," in *ICISSP 2020: 6th International Conference on Information Systems Security and Privacy*, vol. 1, no. 978-989-758-399-5, 2020, pp. 341–348.

[27] J. Jürjens, "Umlsec: Extending uml for secure systems development," in *International Conference on The Unified Modeling Language*. Springer, 2002, pp. 412–425.

[28] C. J. Stettina and W. Heijstek, "Necessary and neglected? an empirical study of internal documentation in agile software development teams," in *Proceedings of the 29th ACM international conference on Design of communication*, 2011, pp. 159–166.

[29] M. Fowler. (2005) Event sourcing. 22/10/2023. [Online]. Available: https://martinfowler.com/eaaDev/EventSourcing.html

[30] G. Karsai, H. Krahn, C. Pinkernell, B. Rumpe, M. Schindler, and S. Völkel, "Design guidelines for domain specific languages," *arXiv preprint arXiv:1409.2378*, 2014.

[31] E. Foundation. Xtext documentatie. [Online]. Available: https://eclipse.dev/Xtext/

[32] T. Parr. Antlr (another tool for language recognition). [Online]. Available: https://www.antlr.org/

[33] J. Verbessem. Threatmodellanguage.xtext. [Online]. Available: https://gitlab.kuleuven.be/distrinet/education/thesissen-2023-2024/master-thesis-jules-verbessem/-/blob/main/Development/DSLXtext/xtext.threatmodel/src/xtext/ThreatModelLanguage.xtext?ref_type=heads

[34] L. Bettini, *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.

[35] E. Foundation. Xtend documentatie. [Online]. Available: https://eclipse.dev/Xtext/xtend/

[36] J. Verbessem. Threat modeling language development code. [Online]. Available: https://gitlab.kuleuven.be/distrinet/education/thesissen-2023-2024/master-thesis-jules-verbessem/-/tree/main/Development/DSLXtext?ref_type=heads

[37] A. Swartz, K. Poulsen, and J. Dolan. Securedrop. [Online]. Available: https://securedrop.org/

[38] ——. Securedrop threat model. [Online]. Available: https://docs.securedrop.org/en/stable/threat_model/threat_model.html

[39] J. Verbessem. evaluatietest1.tml. [Online]. Available: https://gitlab.kuleuven.be/distrinet/education/thesissen-2023-2024/master-thesis-jules-verbessem/-/blob/main/Evaluatie/evaluatieTest1.tml

[40] L. Sion, K. Yskout, D. Van Landuyt, and W. Joosen, "Solution-aware data flow diagrams for security threat modeling," in *Proceedings of the 33rd Annual ACM Symposium on Applied Computing*, 2018, pp. 1425–1432.

Computerwetenschappen
Celestijnenlaan 200A - bus 2402
3000 LEUVEN, BELGIË
tel. +32 16 32 77 00
fax + 32 16 00 00 00
www.kuleuven.be