Integració del reporting de Zed Attack Proxy amb JasperReports

Nom Estudiant: Bruno Albalat Montenegro

Programa: Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions (MISTIC)

Nom Consultor: Agustí Solanas i Jordi Duch

Centre: Universitat Oberta de Catalunya

Data Lliurament: 15/06/2015

FITXA DEL TREBALL FINAL

Títol del treball:	Integració del reporting de Zed Attack Proxy amb JasperReports
Nom de l'autor:	Bruno Albalat Montenegro
Nom del consultor:	Agustí Solanas i Jordi Duch
Data de lliurament (mm/aaaa):	06/2015
Àrea del Treball Final:	Seguretat d'aplicacions web
Titulació:	Màster Universitari en Seguretat de les Tecnologies de la Informació i de les Comunicacions (MISTIC)

Resum del Treball (màxim 250 paraules):

Aquest treball tracta de millorar el reporting de la eina Zed Attack Proxy integrant-la a base de crear una expansió en forma de add-on amb la eina de creació de reports JasperReports.

Aquesta expansió permet la creació de reports personalitzables que son una alternativa al reporting actual, i intenta solucionar el problema que tenia ZAP en aquesta àrea. Es poden crear d'aquesta manera una multitud de reports en els formats PDF, HTML, DOCx, PPTx, ODT i XLS; quan la alternativa actual integrada al sistema proporciona només una sortida HTML i una XML, les dues no modificables.

S'ha donat èmfasi a la capacitat de tenir uns reports fàcilment modificables per a donar a ZAP unes capacitats que altres eines de reporting no tenen. Es proveeix per tant amb una sèrie de exemples molt fàcilment modificables que permeten que personal no tècnic pugui editar-los sense un grau elevat de coneixements fent servir la eina JasperSoft Studio.

Abstract (in English, 250 words or less):

This project attemps to improve on Zed Attack Proxy's reporting by creating an expansion in form of an add-on, which integrates ZAP's scanning with the report creating tool JasperReports.

This add-on allows us to create customizable reports that allow an alternative to the current ZAP reporting, and tries to solve ZAP's issues when handling reports. We are able to create now a wide variety of reports that output PDF, HTML, DOCx, PPTx, ODT i XLS formats, instead of the simple HTML and XML formats that are used by default on ZAP.

The ability of having easily-modifiable reports has been one of our priorities, attempting to give ZAP a push over some of the other reporting tools in this area. Several ready-to-use examples are included with the tool so that non-technical users are able to edit them very easily without a broad knowledge by using the visual tool JasperSoft Studio.

Paraules clau (entre 4 i 8):

ZAP, JasperReports, Reporting

Índex

Llista de figures

1. Introducció

1.1 Context i justificació del Treball

Aquest treball sorgeix al treballar amb varies eines de anàlisi de vulnerabilitats d'aplicacions web. En fer una comparativa era evident que existia una diferència important entre algunes eines comercials i altres alternatives de codi obert, i aquesta diferència no es reflexava tant en el resultats del anàlisi de vulnerabilitats, donat que no s'apreciaven diferències significatives entre els dos; les diferències es podien veure en la manera en que els resultats eren representats.

Això s'acusava especialment en la eina ZAP, acrònim de Zed Attack Proxy, que és una aplicació d'anàlisi de vulnerabilitats que te un creixement important en quan a funcionalitat i adaptació, però que clarament te una limitació important en quant al seu report de resultats.

Ens plantegem per tant crear un report més atractiu, però no només això. Donat que no som dissenyadors, i encara que ho fóssim, no podem oferir un disseny de report que sigui al gust de tots els usuaris. Intentem no limitar-nos a oferir una alternativa de report, volem oferir algo més que el que s'està oferint a qualsevol altre aplicació, pel que el nostre objectiu no és només millorar el report, sinó que fer-ho completament customitzable.

El reporting de ZAP actualment s'ofereix en formats HTML i XML, i el format HTML no te un disseny cuidat sinó que és més aviat funcional. Volem també ser capaços de oferir altres formats de report. Estem parlant doncs de flexibilitzar el reporting de ZAP per a que sigui customitzable tant a nivell de disseny com, si és possible, de resultat. Això vol dir que no només es podria planificar com és el report sinó que a més si el format és editable es podria treballar amb ell.

Finalment, cal dir que és una gran motivació el poder col·laborar amb un projecte open source amb codi propi, donat que sempre estic treballant amb eines d'aquest tipus i és bo poder donar-lis algo de valor.

1.2 Objectius del Treball

Els objectius del treball s'extreuen directament del context. El que volem és millorar el reporting de ZAP, i per fer això el que volem és :

- Oferir una alternativa senzilla, que pugui ser feta servir pels usuaris tipus de ZAP.
- Que sigui una alternativa que sigui editable tant a nivell de disseny com de resultats.
- Oferir a més una alternativa que expandeixi les possibilitats de tipus de resultats per a que es pugui treballar amb diferents formats segons les necessitats dels usuaris.

Creiem que si aquests tres punts es compleixen estarem donant un gran valor afegit a la eina, i omplirem una necessitat que s'evidenciava amb el us de ZAP.

1.3 Enfocament i mètode seguit

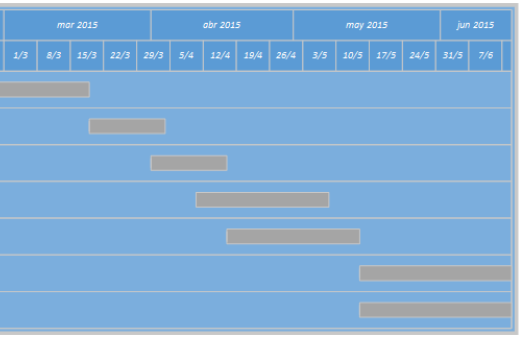
Com veurem més endavant en la explicació de la investigació de les diferents aproximacions que hem contemplat, existia la possibilitat de realitzar una aplicació independent; la realització de una aplicació web i finalment la de la integració amb el producte en si mateix, donat que és open source.

Totes les alternatives tenien el seu valor, la aplicació independent oferien una llibertat per a poder interpretar i fer el què veiéssim necessari, i ens oferien la familiaritat de desenvolupar una aplicació que no depengués de res més. La aplicació web ens oferia a part un entorn de caire més orientat a ser permanent pel que podríem treballar o jugar amb la creació de estadístiques entre reports, historials, i altres.

Hem considerat que la integració directament a ZAP, tot i ser més complicada per a treballar-hi i que ens pugui limitar més, és la més interessant donat que el usuari que fa servir ZAP pot no estar interessat en desplegar una aplicació web o executar una aplicació independent per a recolzar el reporting. En canvi, si es tracta de una opció més dintre de ZAP pot arribar a ser una funcionalitat popular amb futur i molta utilitat.

1.4 Planificació del Treball

ID	Task Name	Start	Finish	Duration
1	Investigació Projecte	25/02/2015	18/03/2015	16d
2	Investigació ZAP	19/03/2015	03/04/2015	12d
3	Investigació eina reporting	01/04/2015	16/04/2015	12d
4	Creació Add-on	10/04/2015	08/05/2015	20d
5	Creació Report	17/04/2015	14/05/2015	20d
6	Establiment Millores	15/05/2015	15/06/2015	22d
7	Treball en memòria	15/05/2015	15/06/2015	22d



Per a desenvolupar el projecte ens plantegem objectius bàsics que representen les fites que hem d'anar complint per a poder acabar a temps amb garanties d'èxit el projecte. Al principi només teníem un objectiu en el qual treballem en definir amb detalls el que volem realitzar al projecte, donat que el millor que podem fer és tenir clar quin és nostre objectiu abans de poder treballar per a evitar realitzar canvis que afectin a la planificació.

Un cop ja tenim decidit el producte a millorar i quines son les millores que volem fer comencem la investigació sobre com podem fer-ho investigant

ZAP i com podem realitzar aquesta expansió. Quan ja comencem a tenir una idea, i de manera paral·lela, comencem a investigar com podem treballar amb reports que siguin editables de manera senzilla.

Tenim com a objectiu començar a treballar a principis d'abril amb el codi per a generar el report i amb una mica de marge, començar al mateix temps a treballar amb el report. Quan ja tenim els dos, com a mínim una versió inicial, treballarem en la documentació i en múltiples iteracions de millora tant del codi com del report.

1.5 Breu sumari de productes obtinguts

Els productes obtinguts son:

- Una extensió de ZAP amb la capacitat d'executar dissenys de JasperReports que facin servir les dades de alertes de ZAP.
- Una sèrie de reports de exemple per a ajudar a treballar al usuari amb aquest format.
- Una sortida xml d'informació de les alertes amb més detalls útils per a poder treballar amb ella, bé sigui amb la extensió o per aplicar-la a qualsevol altre lloc. Aquesta es generarà cada cop que es faci servir la extensió.

1.6 Breu descripció dels altres capítols de la memòria

La resta de capítols es divideixen seguint la trajectòria que ha seguit aquest projecte. Comencem amb un capítol breu dedicat al plantejament, que repassa de nou les motivacions i el perquè de aquest projecte en concret. Un cop ja ho hem avaluat es segueix amb un capítol tractant la investigació de les necessitats, que ens informa sobre com es va arribar a decidir quines eren les necessitats més important que tenia ZAP com a millora.

Quan ja hem plantejat bé el projecte passem a un anàlisi de propostes, que ens defineixen les investigacions fetes per els tres tipus de aplicació que podíem desenvolupar: Web, independent o extensió de ZAP mateix.

Un cop ja definides aquestes passem a descriure com es desenvolupa la extensió de ZAP, indicant els passos necessaris per preparar el entorn de desenvolupament.

Descrivim més endavant la investigació que s'ha realitzat de les diferents eines de reporting, i en descrivim les més importants i prometedores que hem trobat.

Quan ja tenim el entorn preparat i la eina de reporting decidida descrivim com crear un add-on i com obtenir les dades que necessitarem per el report. Quan ja les tenim passem a dissenyar un report senzill, per a seguidament tenir un capítol sobre com generem el report de ZAP amb la extensió.

Ja tenint un report, passem a crear-ne un de més avançat, avaluant les necessitats que tenim i explicant com hem afegit més dades a la sortida XML per a poder facilitar la creació de reports amb dades útils per als reports sense complicar el treball amb JasperReports.

Finalment quan ja tenim el report avançat creat expliquem com hem afegit la generació de diferents formats de sortida per a que els diferents usuaris puguin escollir quina és la que millor els convé.

2. Resta de capítols

2.1 Plantejament

Tot i que s'ha tractat abans en el punt 1.1, de manera resumida, és interessant repassar el plantejament de la necessitat d'aquest treball. Per motius professionals durant el transcurs de les diverses responsabilitats laborals m'he vist confrontat amb diverses tasques involucrades amb la seguretat d'aplicacions, en especial en aplicacions web, donat que és el principal producte de la organització a la que pertanyo.

En el moment en que aquestes tasques involucren comunicació amb altres treballadors, especialment amb aquells que es dediquen més a la gestió que als aspectes tècnics del desenvolupament, trobem que sempre hi ha una barrera important que sobrepassar. És un esforç que s'ha de realitzar per les dues parts. Per part de la direcció s'ha de realitzar un esforç per comprendre la part tècnica per a poder prendre decisions informades, i per part del treballador s'ha de aconseguir poder comunicar clara i eficientment els punts més rellevants que volem que arribin a la direcció per ajudar a que les decisions puguin ser preses.

No només es tracta de comunicació amb la direcció, per desgràcia, donat que la experiència ens diu que la seguretat en aplicacions és un dels temes més desconeguts per molts veterans del desenvolupament, bé per manca de interès o per falta de formació. Això fa que comunicar-se a nivell horitzontal amb l'equip també sigui relativament difícil.

Podem entendre que si no ens podem comunicar eficientment es pot perdre gravetat sobre les recomanacions o resultats trobats a sistemes que haguem analitzat.

Si provem diverses eines que realitzin tests de penetració en trobarem poques que siguin tan diverses com ZAP. No només no consumeix una quantitat desorbitada de recursos com altres eines fan, sinó que ens permet treballar amb opcions realment interessants. La seva facilitat de integració amb un entorn de desenvolupament continu la fa especialment interessant per a que sigui un dels pilars de test en el desenvolupament de una aplicació web que estiguem desenvolupant. A més te funcionalitats interessants que permeten treballar amb entorns complexes. Es disposa de un cercador per a pàgines AJAX, es treballa amb facilitat amb https i es permet configurar amb facilitat la utilització de un token de protecció CSRF, entre altres. Totes aquestes funcionalitats haurien de donar a ZAP una empenta a l'hora de ser una eina a destacar entre altres, però en la meva experiència proposar ZAP sempre suposa tractar el mateix problema: el reporting.

ZAP disposa de dos propostes de reporting: Una visió estàndard en HTML i una en XML, les dues es podrien considerar poc atractives i bastant limitades. Si bé la proposta HTML permet llegir les alertes que s'han produït, no és una experiència que es pugui comparar a altres eines de

caire més comercial. No es proporciona un resum executiu, ni es disposa de estadístiques en format gràfic que ens ajudin a comprendre ràpidament l'estat. No només això, sinó que no tenim cap manera de endreçar ni de gestionar nosaltres mateixos la informació.

Eines comercials com a Webinspect ens proporcionen resums que son rellevants a les alertes que es donen. També es permet cert grau de customització que al final és el que cercarem quan vulguem donar resultats que siguin esclaridors per a altres treballadors i per a la direcció.

Amb aquest projecte ens hem proposat salvar la diferència que existia entre les eines més comercials i ZAP, i la intentem superar. La idea és doble: Proposar una solució que faci més atractiu i llegible el resultat del report de ZAP, per suposat, però no és aquest el objectiu principal del projecte.

El que hem volgut aconseguir realment en el projecte és donar les eines necessàries per a qualsevol desenvolupador, gestor i, en general, qualsevol persona involucrada amb el projecte, per a obtenir les dades que consideri necessàries o importants a l'hora d'extreure un report de ZAP i que les pugui modificar i editar de manera senzilla, donat que les dades rellevant en un report variaran en funció del projecte i de les persones involucrades. A més, hem volgut que el format dels resultats del reporting s'adaptin a les necessitats de la persona que els estigui fent servir, pel que cal disposar de múltiples sortides.

La solució que es proposa finalment al projecte ha estat integrar ZAP amb una eina de reporting molt estesa i de la qual ja hi ha molta gent familiaritzada com és Jasper Reports. Aquesta eina ens permet gestionar reports de manera visual, pel que no es requereixen elevats coneixements tècnics per treballar amb la modificació. Es donarà una plantilla d'exemple des de la qual es podrà veure com s'extreuen els diferents camps de informació per a poder realitzar sense problemes la customització del report.
Finalment s'ha tingut en consideració diverses maneres de realitzar aquesta integració, i s'ha decidit que la més efectiva, si be no la més senzilla, és realitzar la integració de manera que el nou reporting formi part de la eina ZAP mateixa. Aprofitant que es tracta d'una eina open source i que es disposa d'un sistema de creació de extensions s'ha investigat com realitzar la integració amb ella, de manera que a nivell de usabilitat sigui molt senzill fer servir aquest nou generador de reports.

2.2 Investigació de les necessitats

Abans de arribar al plantejament de les diferents alternatives que s'han considerat a l'hora de desenvolupar el projecte s'ha realitzat una investigació per a determinar quins eren els punts més importants a l'hora de millorar el reporting de ZAP.

Després de comparar ZAP amb altres eines i comprovar també les limitacions d'aquestes ens hem volgut centrar en dos components ens concret: La habilitat de manipulació de dades i la flexibilitat per a treballar amb visualitzacions complexes com ara documents de tipus pdf.

Al analitzar com es podia gestionar la capacitat de manipular les dades ens plantejàvem dues solucions; o bé podíem oferir una solució en la que es manipularien les dades abans de crear un report, per poder seleccionar com seria el resultat, o bé s'oferiria un format de sortida del report en que es podria editar al document mateix el resultat. Si bé aquesta darrera proposició és probablement més flexible, la primera és probablement preferible donat que la feina s'ha de realitzar només un cop vers a fer-ho cada cop que es s'obté el resultat de un escaneig. A més contemplem la opció de combinar les dues opcions, per exemple oferint un format de sortida editable a partir de una plantilla.

2.3 Anàlisi de propostes

Per a la resolució del problema que plantegem amb la sortida del report de ZAP hem considerat una sèrie de propostes que podrien ser vàlides. Aquestes propostes difereixen en termes de usabilitat, facilitat de implementació, entorn necessari per a la seva execució i flexibilitat en el seu canvi.

2.3.1 Aplicació independent

Com a primera idea es va plantejar realitzar una aplicació independent en Java. Aquesta aplicació hauria de proveir de una interfície que fes senzill el generar un report a partir de una sortida de un anàlisi de ZAP. Aquesta aplicació es podia plantejar de varies maneres: O bé es treballava amb una sortida de un anàlisi de ZAP, en HTML o, més fàcilment, en XML, i treballàvem amb això per a produir un report, o bé es buscava una manera de fer-ho més dinàmic llençant el anàlisi des de l'aplicació Java mateixa. Això es podria fer donat que ZAP ens proveeix de una interfície REST que ens permetria realitzar aquest tipus de accions.

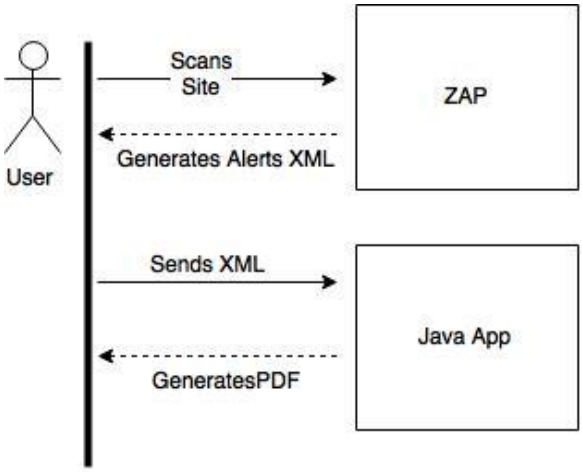
Finalment, ens plantegem la possibilitat de connectar directament amb la base de dades, de manera que podem obviar el connectar-nos amb ZAP de qualsevol manera i podem simplement consultar els resultats de les darreres accions a la base de dades.

2.3.1.1 Investigació

Per a poder tenir una idea més clara de què era més realista a l'hora de plantejar-nos com podríem crear una aplicació bàsica que generés un PDF com a representat d'un format relativament complex. A l'hora de fer aquesta aplicació es va decidir, per motius de senzillesa, agafar com a entrada la sortida XML de ZAP que es genera al realitzar un report.

Un cop ja tenim la sortida el plantejament és simplement descobrir com podríem tenir una sortida en format complex a partir d'aquesta entrada. Després de cercar varies alternatives veiem que les dues més viables son fer servir *iText*, una llibreria que ens permet crear de manera programàtica objectes PDF i la utilització de una transformació XSLT i la conseqüent generació de un PDF a partir d'un generador com *Apache FOP*. Una tercera opció, que considerarem a totes les alternatives, seria fer servir un generador de reports independent més elaborat i treballar amb el disseny.

2.3.1.2 Diagrama d'usabilitat



Il·lustració 1: Diagrama d'usabilitat d'aplicació independent

2.3.1.3 Conclusió

Tot i ser una opció que podria ser senzilla de implementar, trobem que aquesta opció no aporta un valor important al que es fa amb ZAP. L'esquema de treball és incòmode, donat que hem de generar un report amb ZAP i després fer servir la nostre aplicació independentment. No creiem que es tracti d'una operació que sigui útil de cara a un usuari final, si bé podria ser interessant com a exercici.

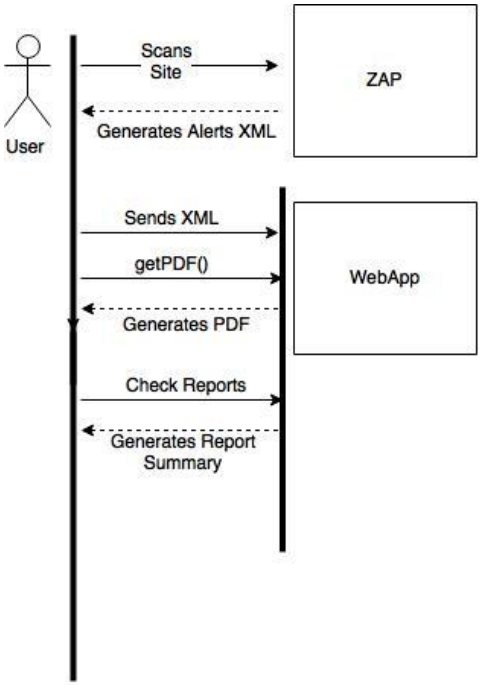
2.3.2 Aplicació web

Una opció interessant seria la de desenvolupar una aplicació web. Aquesta aplicació agafaria les dades del report de ZAP i treballaria amb elles en un portal que l'usuari tindria accessible. A partir d'aquí hi haurien varies opcions, com per exemple donar una opció de que el usuari es descarregués el format transformat en altres formats, de manera similar a com ho hauríem fet amb Java. Tindríem doncs les mateixes opcions de realitzar transformacions en codi o cercar mètodes externs de generació de reports. El que podria ser interessant és fer aquesta aplicació web com a una base en la que podem anar penjant i comparant els diferents resultats dels reports, a part de generar els resultats.

2.3.2.1 Investigació

La investigació és similar a la que realitzem per a la aplicació independent. Hem de afegir a la investigació que realitzàvem abans les tecnologies que ens farien falta per a crear una interfície i poder fer una aplicació visual. Les opcions més lògiques donada la seva rapidesa d'adaptació seria fer servir Javascript i alguna de les múltiples llibreries que existeixen per a realitzar gràfics, com ara HighCharts, que ens proveeix de una visualització molt atractiva.

2.3.2.2 Diagrama d'usabilitat



Il·lustració 2: Diagrama d'usabilitat d'aplicació web

2.3.2.3 Conclusió

Aquesta opció és més interessant en el sentit que és més completa que la anterior, donat que ens dona la capacitat addicional de tenir visualitzacions o resums web. Planteja varius problemes similars a la anterior, i fins i tot més grans, quan ens plantegem que realment tornem a necessitar una aplicació addicional, cosa que pot fer que sigui incòmode o poc pràctic de fer anar. A més haurem de treballar en tenir un servidor d'aplicacions on córrer la aplicació, que pot ser un problema donat que és molt possible que molts dels usuaris de ZAP no vulguin complicar-se o muntar aquest tipus de infraestructura.

2.3.3 Extensió de ZAP

Finalment investiguem la possibilitat de integrar directament la aplicació amb ZAP. La idea va venir després de veure alguns vídeos de funcionament de ZAP en el que els desenvolupadors principals parlaven

de que la idea darrera de ZAP és fer de l'aplicació una plataforma que es pugui estendre fàcilment per a satisfer les necessitats de tots els usuaris.

Donat que es tracta d'una plataforma de codi lliure i que el que es pretén és millorar-la i col·laborar, es va plantejar doncs que investigant com estendre la aplicació en si mateixa suposaria una millor integració amb el concepte d'ella mateixa, i facilitaria el seu us. No obstant això s'esperaven certes limitacions a l'hora de fer això. Al no controlar el ecosistema podem preveure que estarem bastant més limitats a l'hora de desenvolupar, donat que no sempre podrem fer servir les llibreries que ens facin falta o podrem controlar la interfície com ens sigui més còmode, ja que ens hem de adaptar al ecosistema.

2.3.3.1 Investigació

Per a comprovar com hauríem de procedir a l'hora de integrar la aplicació amb ZAP comprovem primer les pàgines web del projecte. Allà trobem que existeixen un grup allotjat a Google que ens proporciona informació general sobre el que podem fer per a realitzar la integració: https://code.google.com/p/zaproxy/

Un cop visitem aquest lloc web ja intuïm que la integració no serà del tot senzilla donat que no veiem la informació a la wiki molt esclaridora, però tenim una millor idea del que s'ha de realitzar. La idea general que ens queda és que sembla que el projecte està fortament lligat amb Apache Ant, que està desenvolupat en Java i que les extensions tenen un repositori separat, en el qual haurem de treballar. Per a poder desenvolupar una extensió haurem de seguir els exemples de les extensions ja creades doncs, donat que no tenim un clar esquelet del que s'ha de realitzar. El que si sembla és que haurem de crear una tasca Ant per a la compilació del nostre codi i un cop aconseguit el format de la extensió haurem de carregar-la dintre del programari principal de ZAP.

Una ullada a la wiki sobre la informació que se'ns dona al apartat d'extensió de ZAP no ens dona massa idea del que ens trobarem:

Extending ZAP

There are various ways you can extend ZAP, as documented below.

Note that this is just an overview, there are some basic working examples in the ZAP Extensions project which will be added to in the future.

If you have any questions then please ask them on the zaproxy-develop Google Group.

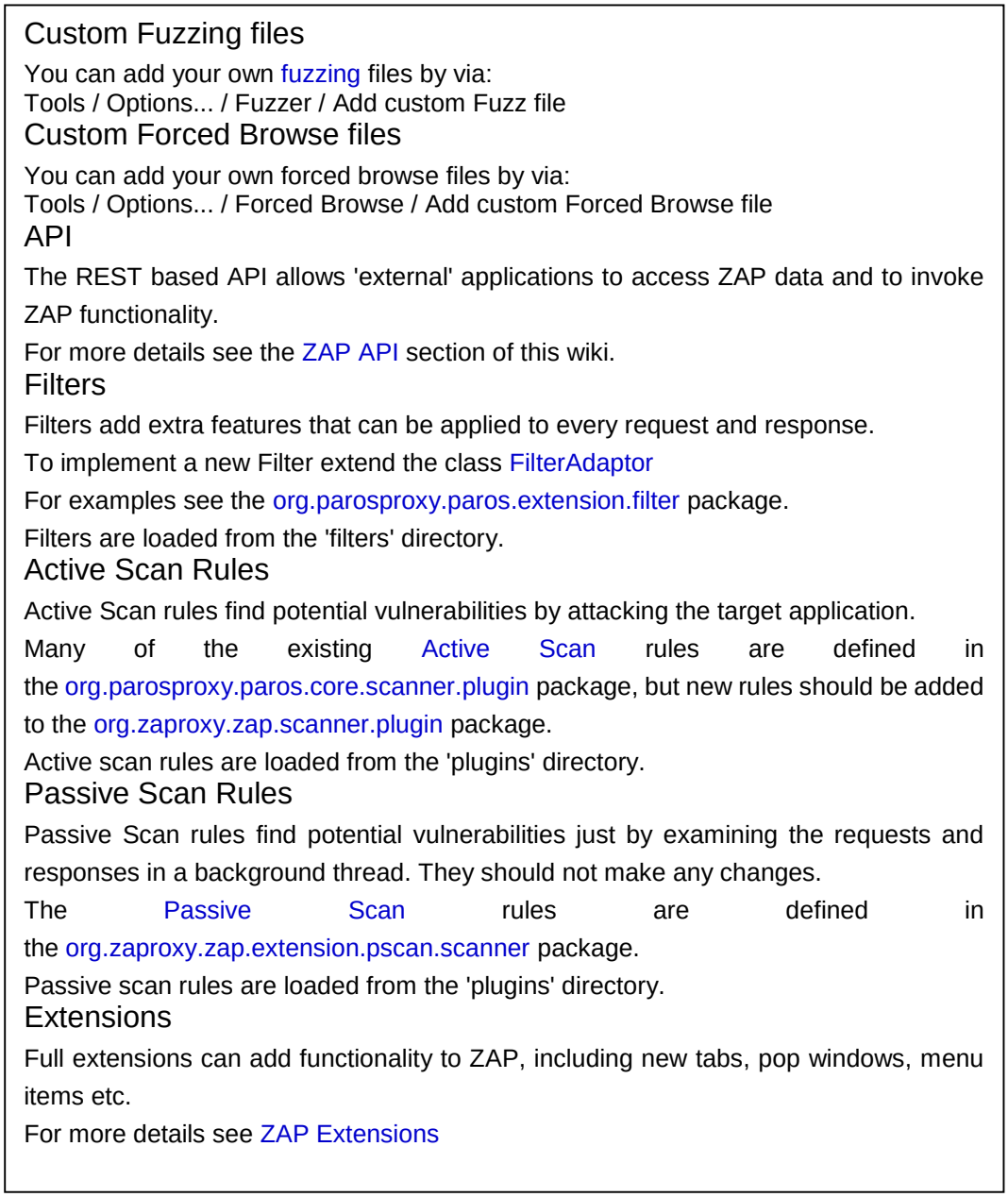
Auto tag regexs

You can get ZAP to automatically tag requests and responses via:
Tools / Options... / Passive Scan / "Add / Edit scan definition"

Invoking other applications

You can invoke other applications from ZAP passing across the context information.

For more details see the Options Application screen in the User Guide.



Il·lustració 3: Wiki de ZAP sobre la expansió

Veiem bastanta informació sobre el tipus de escaneig, i ens fem a la idea de que la funcionalitat bàsica de la extensió de ZAP ha estat normalment utilitzada per a oferir una millor cobertura en els tipus de escaneig i regles.

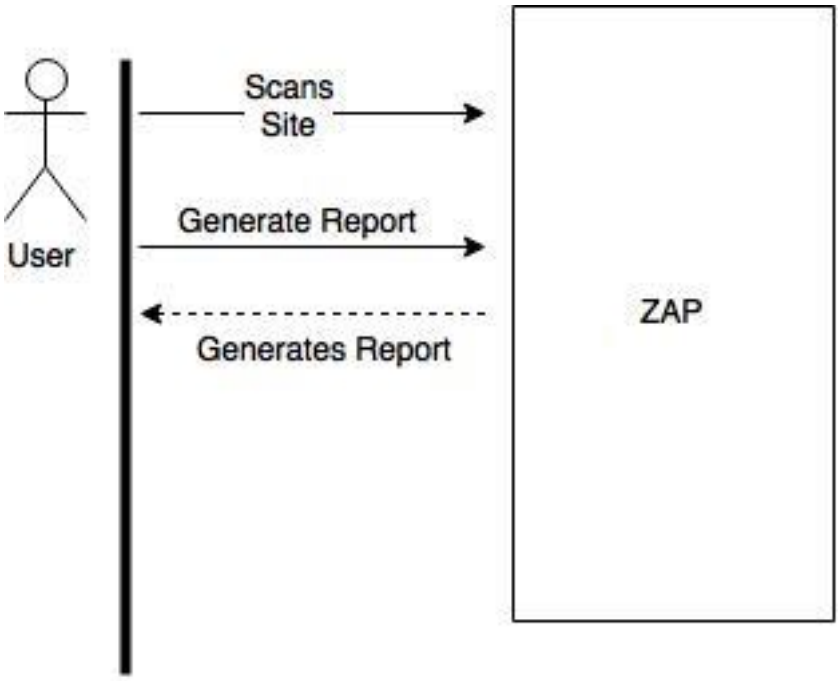
Al enllaç que trobem de extensions completes de ZAP tampoc trobem massa informació, donat que se'ns parla bàsicament de la gestió de dependències, que a més veiem que és una pàgina que avisa que està desactualitzada i que el procés ha canviat, i de la desinstal·lació de les extensions, que no ens concerneix de moment:

ZAP Extensions

There are lots of examples of extensions under the org.zaproxy.zap.extension package and in the zap-extensions project.
Extensions are loaded from the 'plugins' directory.
More information about implementing extensions:

- Dependencies
- Unloading

Il·lustració 4: Wiki de ZAP sobre la expansió (II)



Il·lustració 5: Diagrada d'usabilitat d'extensió de ZAP

2.3.3.1 Conclusió

No és fàcil concloure què en pensem de la investigació de la realització de un add-on de ZAP per a estendre'l. Decidim realitzar això donat que a nivell de usabilitat és clar que serà la opció que més fàcil serà que es faci servir, donat que qualsevol de les altres és improbable que sobrepassi el context d'un projecte. En qualsevol cas, podem veure que no serà un trajecte fàcil en el sentit de que no disposem de gaire informació respecte a com podrem crear un entorn de desenvolupament i treballar-hi, i un cop aconseguit ens haurem de fixar en altres extensions per a veure quines son les funcionalitats bàsiques de la extensió de ZAP.

2.4 Desenvolupament d'una extensió de ZAP

Explicarem ara com s'ha realitzat tot el desenvolupament de la extensió en si mateixa, començant per la preparació del entorn.

2.4.1 Preparació del entorn de treball

Haurem de definir i configurar un entorn de treball amb els que treballar. Aquest entorn haurà de incloure tant el entorn que preparem per treballar amb ZAP com el que necessitarem per desenvolupar les extensions. Intentarem que aquest entorn ens sigui familiar, i per tant apostarem per desenvolupar amb eines amb les quals ja estem familiaritzats.

2.4.1.1 Preparació del entorn per ZAP

Un cop hem decidit desenvolupar una extensió el primer que hem de fer és aconseguir compilar el projecte per a poder-hi treballar. Per a fer això investiguem on aconseguir el codi i veiem que ZAP actualment es desenvolupa fent servir un sistema de versió de control subversion, tot i que s'està en un procés de migració a git. Per els propòsits del projecte ens basarem en el sistema subversion donat que no està en proves ni sent migrat, i serà el que ens proporcionarà una experiència més estable. Un cop descobrim a la wiki quina és la adreça on hem de fer el checkout de subversion veiem que tenim suficients permisos per llegir-lo:

```
svn checkout http://zaproxy.googlecode.com/svn/trunk/
```

Només farem checkout del "trunk" donat que volem treballar amb la darrera versió de ZAP, no amb les branques o les ja publicades.

Quan ja tenim el codi passem a intentar tenir un entorn de desenvolupament funcional. Ja des de un primer moment veiem que ZAP està usualment desenvolupat fent servir el programari de la fundació Eclipse, donat que trobem un enllaç a un espai de treball per a aquest entorn, ja preparat per a evitar que s'hagi de configurar res més. Decidim no fer servir aquest espai de treball donat que es prefereix, per no perdre temps més endavant, fer servir el programari de la empresa Intellij, Idea, donat que es tracta de desenvolupament en Java i ja estem acostumats a desenvolupar fent servir aquest programari, pel que a l'hora de treballar amb codi serem més eficients. Es podria treballar perfectament amb Eclipse, per això, i els resultats serien els mateixos.

Per a poder començar decidim importar el projecte des dels objectius de Apache Ant, donat que veiem que és una de les maneres en que el programari ens permet realitzar la importació. Al cap d'una estona veiem que potser no ha estat la millor manera de realitzar-ho, donat que sembla que els objectius de Ant no han estat ben analitzats per el programari i no sembla que tinguem ben importat el codi.

Com veiem que potser serà millor no complicar la importació intentant fer-la més automàtica del que podria ser decidim importar directament des de

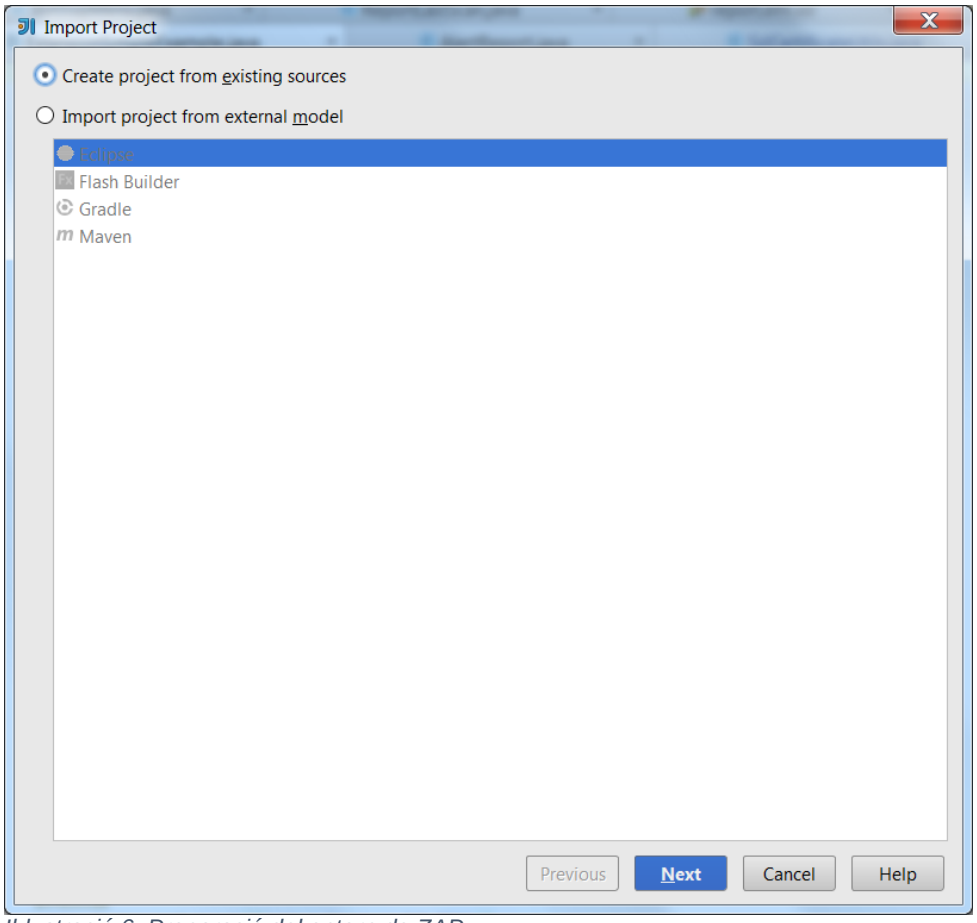
el codi font. Al finalitzar descobrim que el codi s'ha importat correctament, però sembla que tenim alguns conflictes. Al intentar compilar veiem que sembla que hi ha problemes amb alguns dels fitxers, i al intentar compilar i investigar una mica trobem que hi pot haver alguns problemes amb els genèrics de Java que fan servir el constructor diamant, pel que podem concloure que el nivell de llenguatge mínim que hem de fer servir és Java 1.7. Un cop realitzat aquest canvi veiem que s'han arreglat la majoria dels problemes, però trobem un problema que fa més mala pinta a l'hora de resoldre-ho, donat que sembla que uns dels mètodes de HSQLDB, la base de dades principal de ZAP, no està sent trobat.

Al cap de un bon temps en el que realitzem una sèrie de proves com intentar importar el projecte de diverses maneres, configurar les llibreries manualment o depurar els problemes a nivell de classloader, veiem finalment que un dels projectes que està inclòs a dintre del repositori de ZAP està donant problemes, però no el podem treure. Comprovem que en aquest projecte existeix un altre versió de la llibreria de hsqldb dintre d'aquest i està sent carregada pel nostre entorn, donant conflictes a la importació i donant per tant problemes a l'hora de intentar compilar.

Ens encarreguem de treure aquesta llibreria de l'entorn i podem comprovar que ja podem compilar el projecte principal de ZAP amb normalitat.

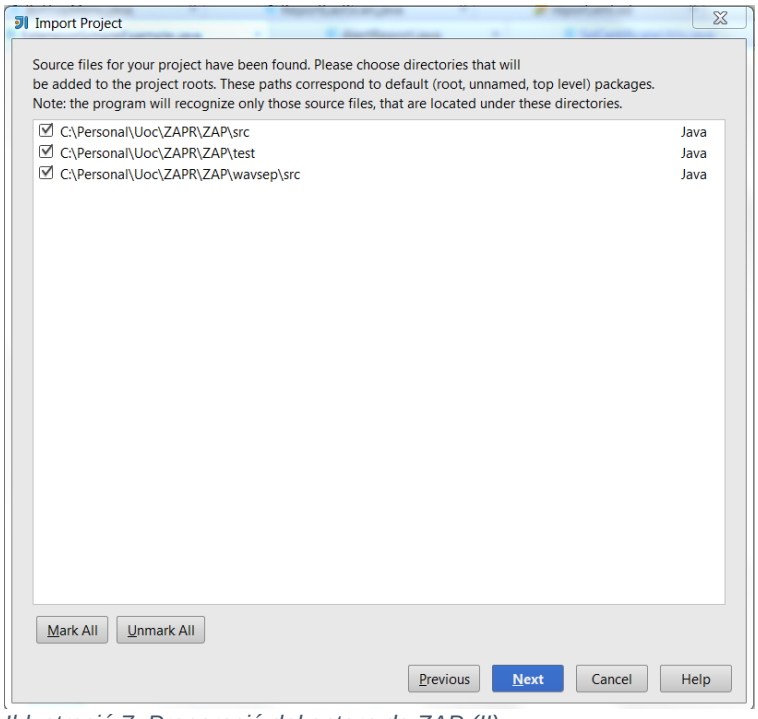
Els passos que hem realitzat son:

Creem el projecte a partir del codi font.



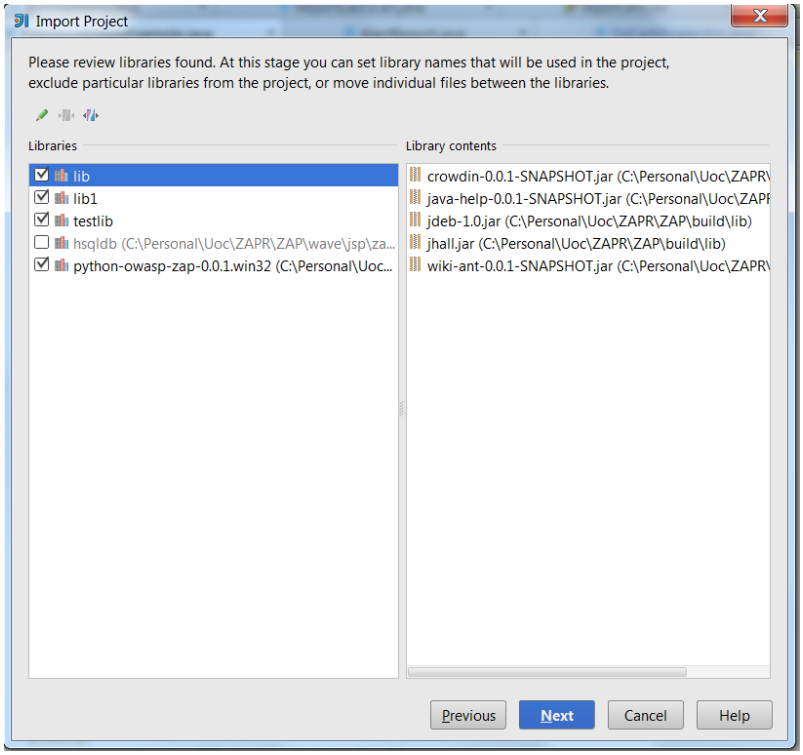
Il·lustració 6: Preparació del entorn de ZAP

Importem tots els projectes de ZAP.



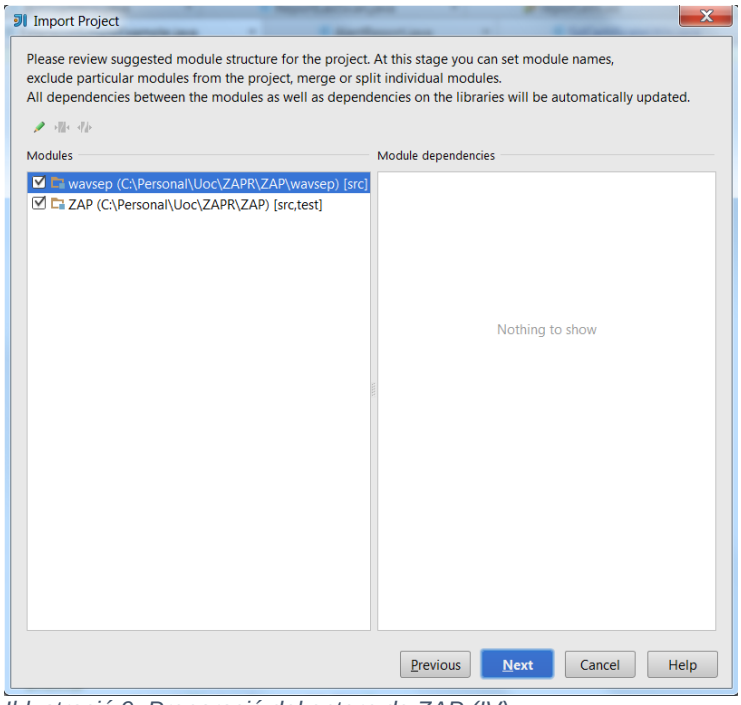
Il·lustració 7: Preparació del entorn de ZAP (II)

Al definir quines llibreries importem, excloim la llibreria hsqldb dintre del modul "*wave*".



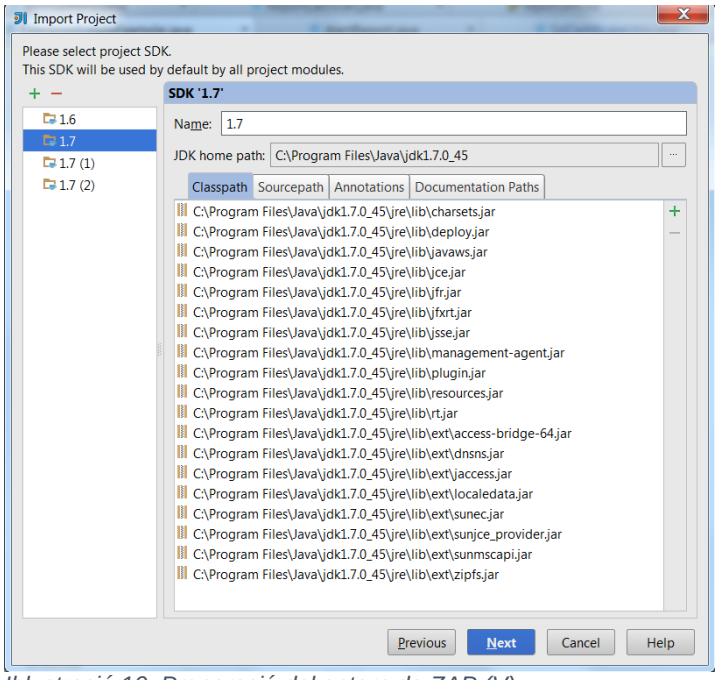
Il·lustració 8: Preparació del entorn de ZAP (III)

Importem tots els moduls.



Il·lustració 9: Preparació del entorn de ZAP (IV)

Ens assegurem que el SDK de Java és com a mínim el 1.7. També funcionarà amb 1.8.



Il·lustració 10: Preparació del entorn de ZAP (V)

Finalment ens assegurem que el nivell del llenguatge també està com a mínim a Java 1.7, donat que sinó tindrem problemes amb els genèrics.

2.4.1.2 Preparació del entorn per les extensions de ZAP

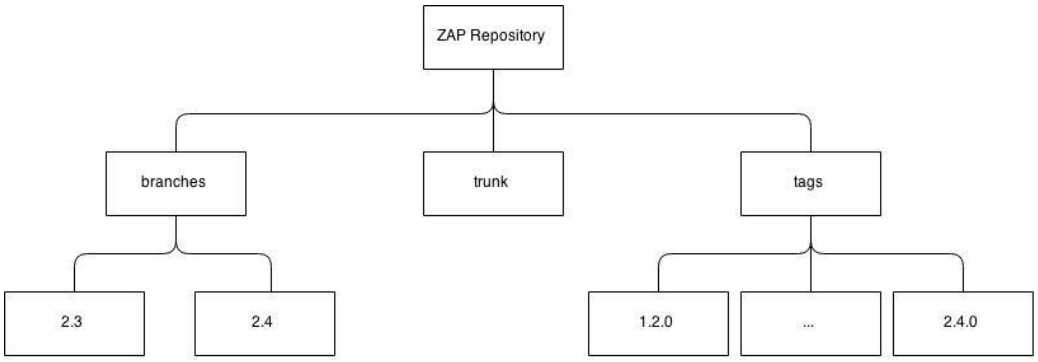
Hem preparat ja el entorn de ZAP donat que entenem que és el necessari per a desenvolupar extensions, però en el entorn no trobem on es troben les extensions amb les que haurem de treballar per a poder crear la nostra.

Investigant una mica descobrim que existeix un altre repositori Subversion amb el que haurem de treballar per a desenvolupar les extensions. Aquest repositori és el que conté totes les extensions de ZAP, i està localitzat a http://zap-extensions.googlecode.com/svn/trunk. Procedim a baixar-nos el codi doncs d'aquest repositori.

```
svn checkout http://zap-extensions.googlecode.com/svn/trunk/
```

Veiem que trobem exemples de diferents extensions. Destaquem una que trobem especialment interessant, doncs s'anomena "Example Top Menu", i sembla que ens ajudarà a crear un menú de nivell superior.

Examinant el repositori Subversion de les extensions de ZAP ens trobem que hi està estructurat de manera diferent al de ZAP en si mateix. El repositori de ZAP estava organitzat de la següent manera:



Il·lustració 11: Estructura del repositori de ZAP

Trobem que aquesta estructura és probablement estàndard: Dintre del repositori existeixen les branques de desenvolupament on es realitzen els canvis a les versions en les que encara es treballa, bé per arreglar errors o per introduir noves funcionalitats. Té sentit que es treballi encara en la versió 2.3 de ZAP donat que tot just s'ha publicat la 2.4 i això portarà a que hi hagi encara molts usuaris que no puguin o vulguin canviar encara a la 2.4 si comporta canvis importants.

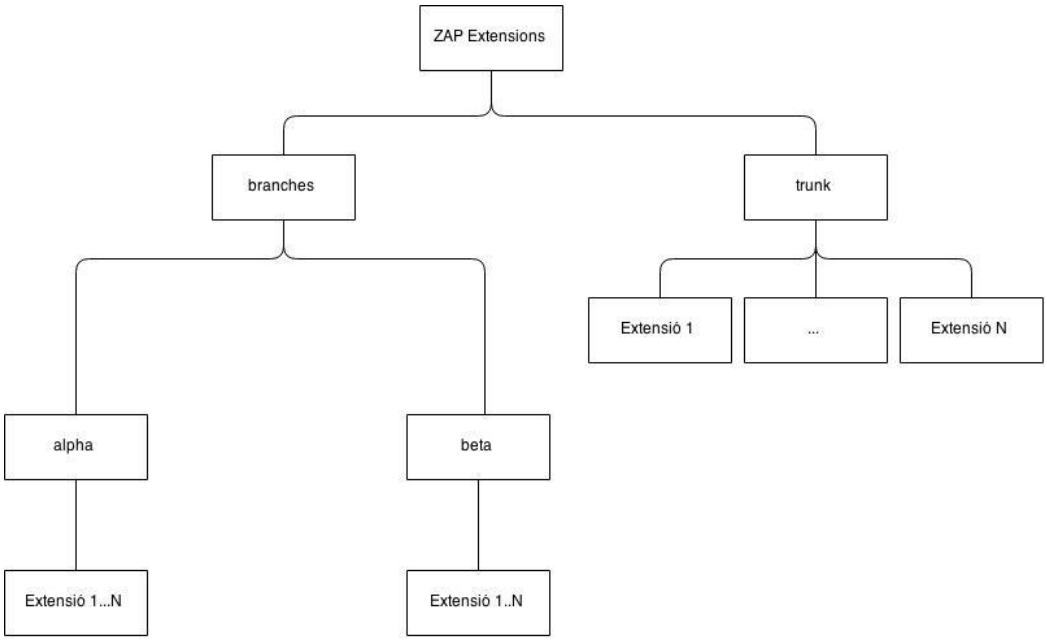
Per un altra banda veiem la branca principal o trunk, que és on es realitza el desenvolupament principal i on s'introdueixen les noves funcionalitats que no s'inclouran en versions antigues. Finalment, veiem la carpeta "tags", on podem veure l'estat del codi per a les publicacions que s'han anat fent de ZAP.

Quan fem una ullada al repositori de les extensions de ZAP entenem que la organització varia donat la natura d'aquestes. Les extensions de ZAP son externes al nucli, i no pertanyen al cicle de vida d'aquest. No només això, sinó que cada una tindrà el seu propi cicle. És per això que totes elles s'organitzen de manera independent.

La organització del repositori no es limita a això, sinó que veiem que trobem les extensions organitzades en tres branques principals: alpha, beta i trunk. Totes elles tenen una organització idèntica, pel que veiem que es tracta de tres branques on es desenvolupen les extensions. Com no trobem informació a la wiki sobre les branques i la seva funcionalitat preguntem als desenvolupadors, que ens informen que les branques indiquen el nivell de revisió que han tingut aquestes extensions.

Una extensió a alpha tindrà com a requisit que compili i funcioni, bàsicament. Una en la branca beta serà una extensió que hagi estat revisada per el equip desenvolupador de ZAP, i una en el trunk tindrà a més una sèrie de normatives de codi, usabilitat i funcionalitat revisades per a poder-hi pertànyer. Totes les noves extensions aniran a la branca alpha, pel que la prenem com a base per a compilar una extensió d'exemple.

Resumint, així és com està organitzat el repositori de les extensions de ZAP:



Il·lustració 12: Estructura del repositori de les extensions de ZAP

Un cop ho entenem, veiem que no bastarà amb fer servir el "trunk", com en el projecte principal de ZAP, pel que haurem de fer servir tot el repositori:

```
svn checkout http://zap-extensions.googlecode.com/svn
```

Un cop ja hem realitzat aquestes passes podem passar a treballar de nou amb el programari Intellij Idea per a poder passar a realitzar una compilació de exemple.

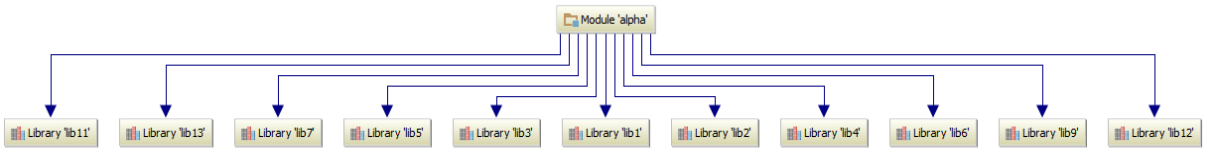
De manera similar a anteriorment importem el projecte a Idea. Com que necessitem treballar amb la branca alpha importarem no només la branca principal, sinó el projecte complet per a poder-hi treballar.

Un cop ho realitzem analitzem com funciona el sistema de compilació de extensions. Similarment a com funciona el nucli de ZAP, veiem que existeix un sistema de compilació basat en Apache Ant. Observem també una dependència amb el projecte del nucli de ZAP, i que no podem compilar cap de les extensions donat que no troba la ruta on posar-les per que es carreguin amb el programari principal.

Això ens fa veure que a més no podrem debugar les extensions probablement si no tenim inclòs també ZAP en el mateix projecte, per el que acabem amb una configuració en la que posem les tres branques de les extensions i el trunk de ZAP en un mateix projecte. No sembla que tinguem conflictes amb les llibreries del nucli de ZAP, per sort, però veiem que no podem compilar de manera normal sense tenir alguns problemes de llibreries. Per sort els objectius d'Ant ja sembla que s'encarreguen d'arreglar alguns dels problemes que podem tenir de compilació.

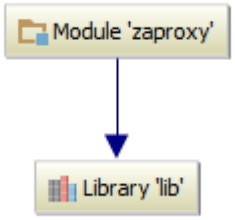
Observem que tractem amb importació de llibreries per extensió. Per fernos a una idea, aquesta és la diferència entre la importació de llibreries del projecte central de ZAP i la branca alpha de les extensions:

Mòdul alpha:



Il·lustració 13: Estructura del mòdul alpha

Modul central:



Il·lustració 14: Estructura del mòdul central

Com veiem es tracta d'un plantejament diferent, més modular.

2.4.2.1 Modificació dels fitxers d'Ant

El primer que fem un cop hem preparat el entorn és intentar compilar la branca "alpha", en la qual treballarem per a poder publicar la nostre extensió. Podem veure que, igual que abans, no trobem satisfactòriament la ruta cap on hem de copiar les extensions compilades.

Analitzem el fitxer Ant, donat que ara ho tenim tot al mateix projecte, i veiem que el problema es dona en la organització al nostre sistema de fitxers, donat que aquest es planeja de manera diferent per defecte. Després de fer els canvis adients a les rutes ens trobem amb aquests canvis al fitxer *build.xml:*

Il·lustració 15: Canvi de ruta al fitxer ANT

```
<project name="ZAP-Extensions (Alpha)" default="build-all"
basedir=".">
    <description>Build ZAP extensions</description>

    <property name="src" location="../src" />
    <property name="src.version" value="1.7" />
    <property name="build" location="build" />
    <property name="build.lib.dir" location="lib" />
    <property name="temp" location="temp" />
    <property name="dist" location="zap-exts" />
    <property name="dist.lib.dir" location="../lib" />
    <property name="status" value="alpha" />
    <property name="versions.file" location="${dist}/ZapVersions.xml"
/>
    <property name="wiki.dir" location="../../zap-extensions-wiki" />
    <!-- This assumes you also have the zaproxy project -->
    <property name="zap.plugin.dir"
location="../../../../zaproxy/src/plugin" />
    <!-- This assumes you also have the zaproxy-2.4 project (used for
the weekly release -->
    <property name="zap-2.4.plugin.dir" location="../../../../zaproxy-
2.4/src/plugin" />
```

Com veiem ja ens diu en aquets fitxer que assumeix que tenim el projecte zaproxy (el projecte del nucli del codi de ZAP) disponible. Un cop fet això, després de varies proves i errors, sembla que ja estem llestos per a compilar les extensions de ZAP, pel que ja podrem fer proves amb les que intentarem millorar el reporting.

2.4.3 Investigació de eines de reporting

Un cop ja hem realitzat la primera compilació i tenim el entorn preparat cal decidir exactament amb quina eina de reporting treballarem per a poder millorar el reporting de ZAP. El que busquem és una eina que no només sigui fàcilment editable i que es pugui modificar sense tocar codi o tenir coneixements tècnics, sinó que a més tingui, si és possible, un editor visual i sigui utilitzada per un públic ampli de manera que puguin reutilitzar-se els coneixements que s'hagin adquirit prèviament.

Considerem diverses eines existents al mercat, i analitzem els pros i els contres de la seva implementació.

2.4.3.1 BIRT

BIRT és una de les eines més interessants de les que hem avaluat. Es tracta de un projecte desenvolupat per Eclipse que és tant una eina de reporting com de Business Intelligence. Ha estat essent desenvolupada des del 2004, pel que és una eina madura, i és open source.

Els components principals que engloben a BIRT son el "BIRT Report Designer", que és la eina visual que funciona com a editor gràfic de

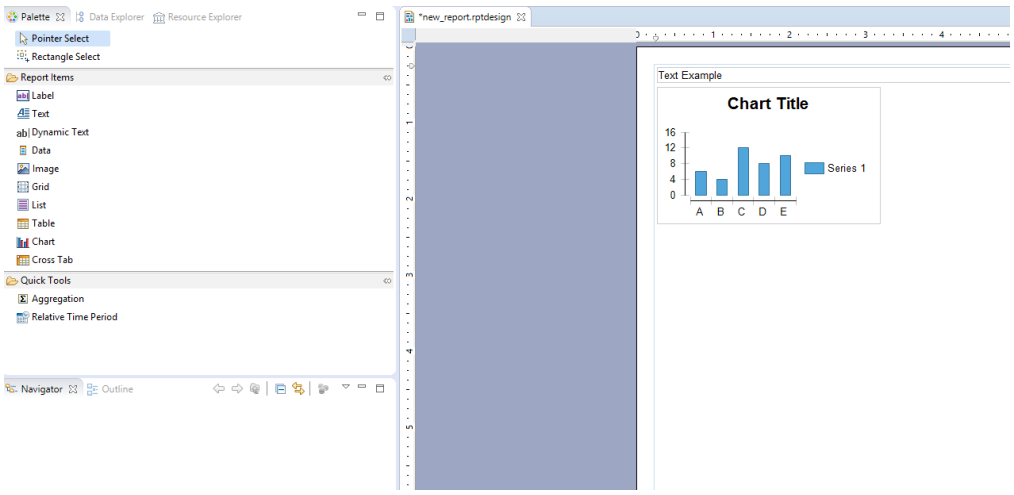
reports, que els usuaris faran servir per a poder gestionar els reports, i el "BIRT Runtime", que és el executable Java que permet passar els dissenys dels reports en XML, gestiones els orígens de dades i generar la sortida en el format especificat per el usuari.

A part d'aquests components BIRT també disposa de components que es poden executar de manera independent com el "BIRT Chart Engine", que permet la creació de taules de manera independent; el "BIRT Chart Designer", que és una eina basada en OSGI que ens permet dissenyar les taules en aplicacions que també suportin OSGI; el "BIRT Viewer", un plugin de Eclipse i aplicació java que ens permetrà veure el resultat de tenir reports amb les dades ja poblades i, finalment, el "Eclipse Data Tools project", que és el projecte per el qual es gestionen les dades que arriben al report.

BIRT és especialment interessant per la facilitat amb la qual es poden editar els reports de manera visual. Aquesta característica fa fàcil la creació de reports simples i complicats de igual manera, i permet també la utilització de codi personalitzat per als reports si el editor no és suficient.

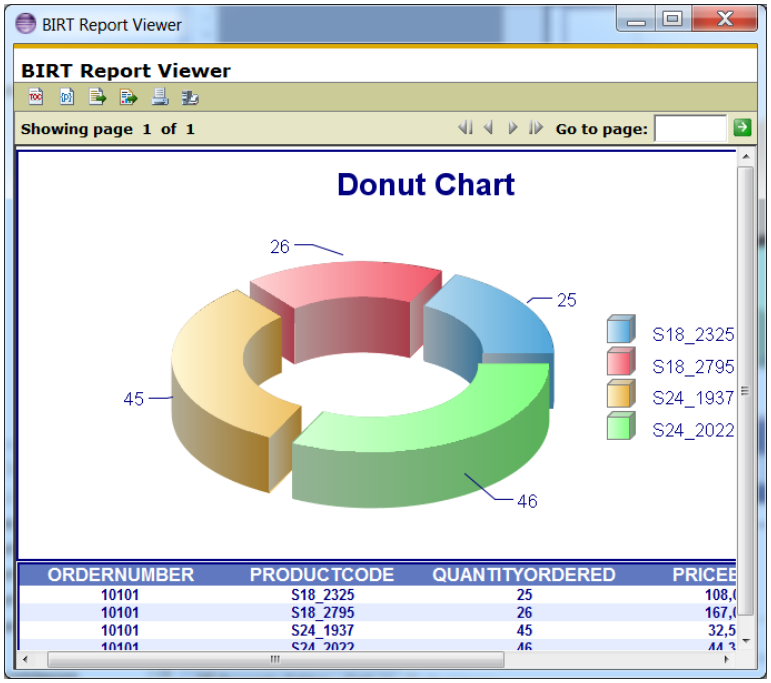
La generació de reports en BIRT està basada en una matriu, al contrari de moltes de les alternatives, que es defineixen a nivell de píxel. Es pot dir que BIRT fa servir una distribució molt semblant a la de les pàgines web.

Podem veure la interfície de BIRT:



Il·lustració 16: Interfície de BIRT

Un exemple de report:

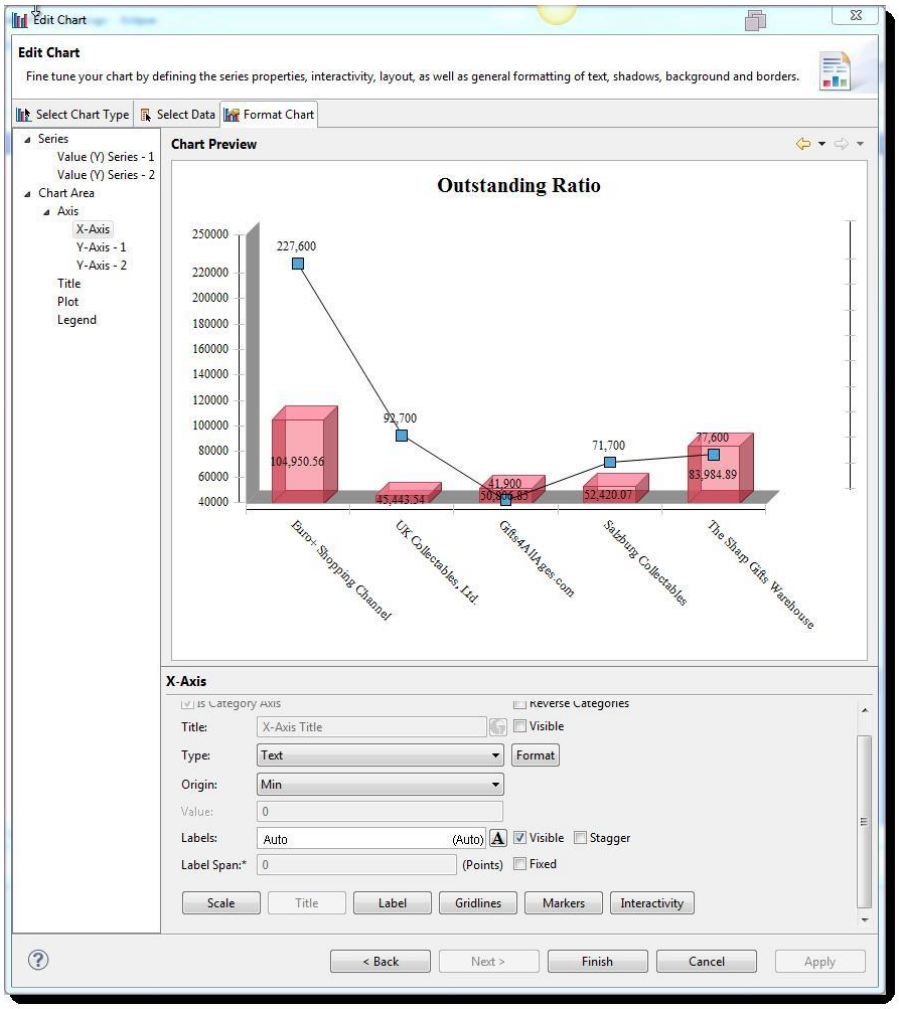


Il·lustració 17: Report de BIRT

Els reports de BIRT es composen enterament de fitxers en XML, a diferència d'altres alternatives, on s'han de compilar. Per un altre banda, la seva aproximació al disseny similar a la del disseny de pàgines web fa que els resultats de els reports amb BIRT siguin de manera general agraïts de veure en pàgines web, i ho fan una gran alternativa si el seu darrer objectiu és la visualització web en si.

Un altre punt positiu de BIRT és la seva facilitat a l'hora de generar gràfics de manera completament visual i de, donada una entrada de dades, previsualitzar-los per a poder contemplat quins seran els resultats. A més, els gràfics generats amb BIRT ens semblen especialment atractius visualment, el que li dona un punt positiu. També veiem que BIRT està molt ben documentat, contràriament a algunes de les alternatives

S'ha d'afegir que, en qualsevol cas, la interfície, tot i ser poderosa, pot demostrar ser bastant complicada per a una primera aproximació.



Il·lustració 18: Report de BIRT (II)

Podem dir que BIRT és una bona opció, però la seva base de usuaris no és tan gran, probablement donat que és una eina més aviat per reporting i no tant per Business Intelligence en general. Els últims anys no s'han fet gaire canvis a BIRT, el que dona una sensació de que el projecte està bastant abandonat, i la seva interfície pot ser una mica massa al principi. Finalment, si s'han de imprimir els reports, son millors les alternatives donada la precisió amb la qual se'ns deixa controlar el posicionament del elements.

2.4.3.2 Pentaho

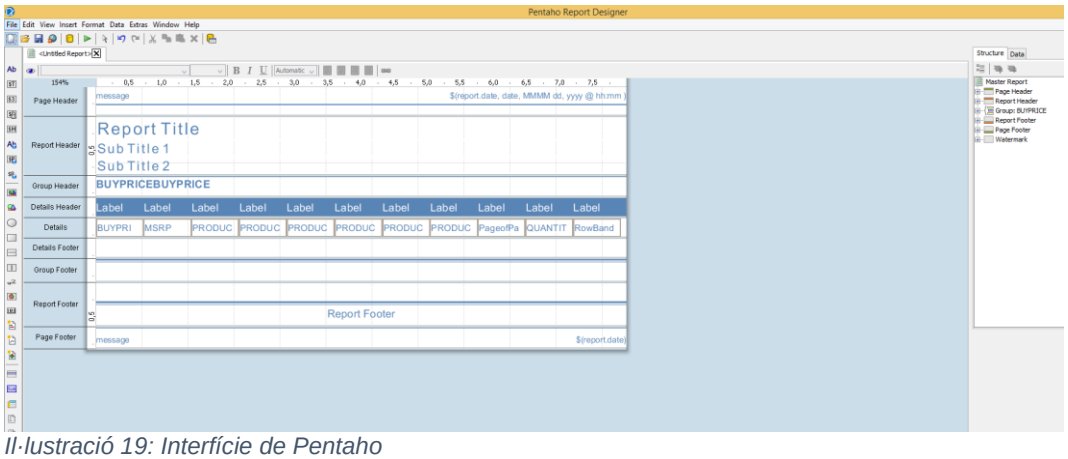
Pentaho és un altre eina open source, ara adquirida per Hitachi, que proporciona la capacitat per a realitzar reports. A diferencia de BIRT, aquesta eina es basa més aviat en Business Intelligence, i els reports son només una part del que es treballa amb ella.

Un dels components principals de Pentaho amb els quals treballaríem son el "Pentaho Report Designer", eina que de manera similar al que hem vist abans amb BIRT o el que veurem amb Jasper, ens permet crear reports de manera visual amb una interfície gràfica intuïtiva que internament

genera un fitxer XML. Aquest component no és un add-on de Eclipse com a les altres eines sinó que és una aplicació independent. Treballaríem també amb el component “Pentaho Reporting Engine Classic”, que és una col·lecció de classes en Java que ens permetran passar aquest report que hem creat en XML fent servir “Pentaho Report Designer” o editant un XML directament, compilar les dades de les fonts que configurem i generar una sortida en diversos formats. Finalment també podríem treballar amb “Kettle”, que és la eina de integració de dades de Pentaho, que ens permet extreure, transformar i carregar dades de diverses fonts en els nostres reports.

Pentaho també te com a components el “Pentaho Reporting SDK” i el “Pentaho BI Server”, la primera és el motor amb llibreries de suport i documentació, i el segon és una aplicació que permet que s’executin els reports contra un servidor que es pot gestionar de manera remota i proveeix dades analítiques.

En termes generals podem dir que probablement la alternativa de Pentaho sigui la més fàcil de fer servir, si bé no és la més potent. Es nota que es tracta de un conjunt de eines de les quals el reporting només n’és un component.



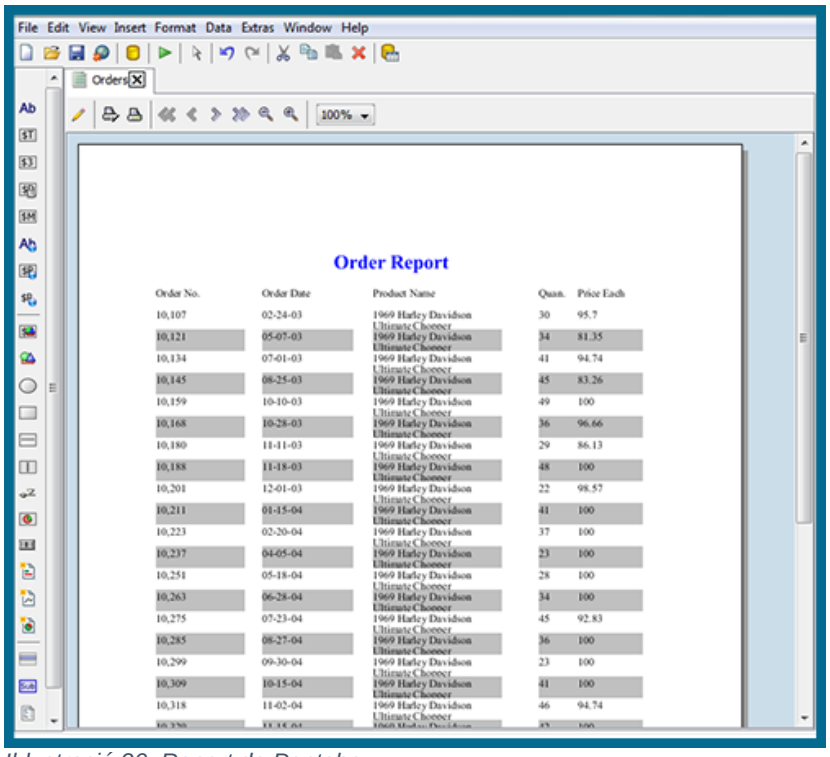
Il·lustració 19: Interfície de Pentaho

Per a la generació de gràfics Pentaho proposa un ajudant de creació que és molt útil quan no sabem com acabar de plantejar la feina, tot i que amb això i tot és difícil crear-los i no és prou potent. A més, es disposa de un disseny basat en posicionament de píxel, pel que es pot ser molt precís a l’hora de definir com volem que siguin els resultats.

La eina de transformació i processat de dades és potent i s’agraeix. A més, existeix una base gran de usuaris que han treballat amb productes de Pentaho, sobretot si s’ha tocat una mica la Business Intelligence.

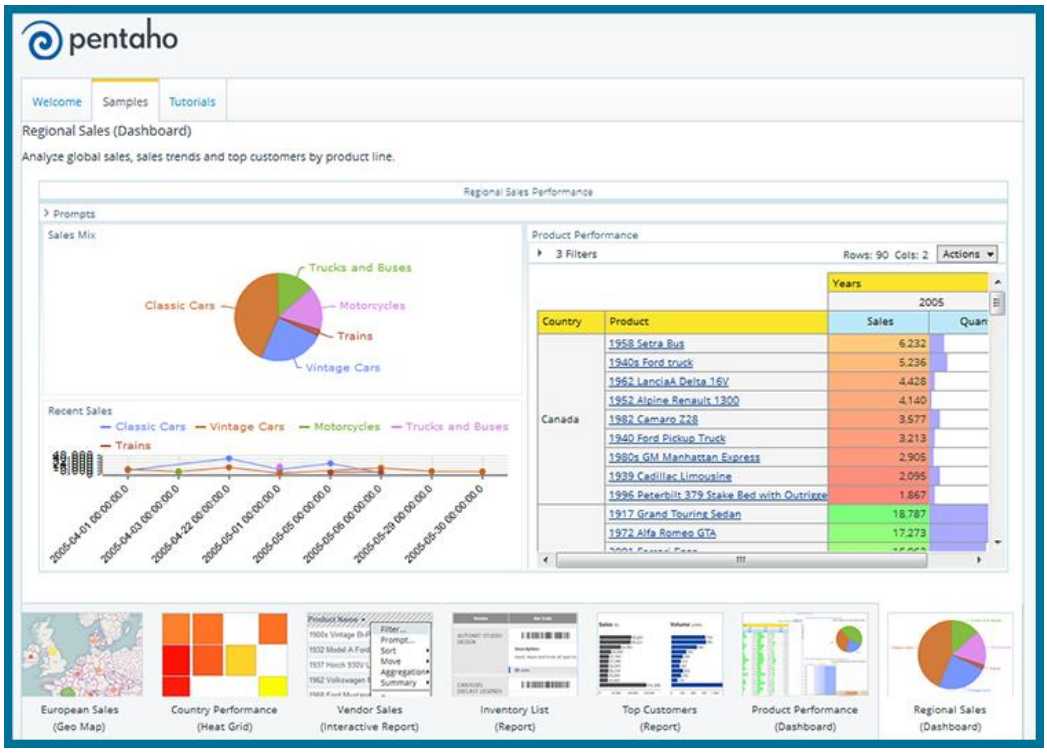
El dissenyador de reports és molt atractiu visualment i es nota que està cuidada la experiència del usuari, donat que és intuïtiu de fer anar. A més,

gestionar detalls individuals com tractar diferents colors amb condicions està molt més resolt que a les alternatives.



Il·lustració 20: Report de Pentaho

De manera similar a BIRT, es treballa només amb XML, el que facilita que es pugui modificar manualment sense la necessitat de el dissenyador (tot i no ser el objectiu en el nostre cas), però a més veiem que existeix un llenguatge similar al fet servir per Excel per a interpretar ordres directament en XML, el que ajuda als que no estiguin familiaritzats amb codi a treballar-hi si és necessari.



Il·lustració 21: Report de Pentaho (II)

Com a punts negatius, veiem que és difícil generar els gràfics, i que és una eina que es fa servir més aviat com a conjunt que independent, donat que Pentaho està molt associat al concepte de Business Intelligence. No suporta a més bastantes funcionalitats avançades que altres alternatives si ofereixen, i es queda potser en un reporting bàsic, que no és el que busquem donat que volem donar flexibilitat als usuaris a l'hora de poder fer el que vulguin amb el report.

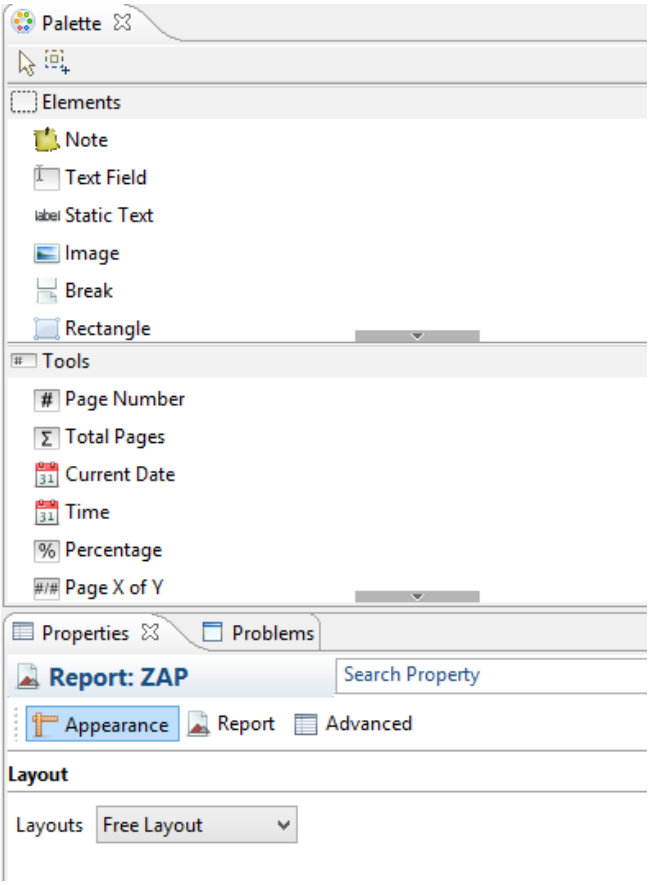
2.4.3.3 JasperReports

JasperReports és un producte de TIBCO, i forma part de un conjunt de programes orientats al reporting i portats per aquesta mateixa companyia. Es tracta d'un altre alternativa open-source, i també ens ofereix una experiència visual a l'hora de dissenyar els reports que després seran omplerts amb dades externes.

JasperReports és la eina més popular i àmpliament coneguda open source de reporting que es fa servir actualment, i es fa servir en un nombre molt gran de productes. Si bé no te una aplicació tan amplia com el conjunt de Pentaho, si que ofereix integració de dades i Business Intelligence a un nivell molt ampli. Es composa de diversos components, com:

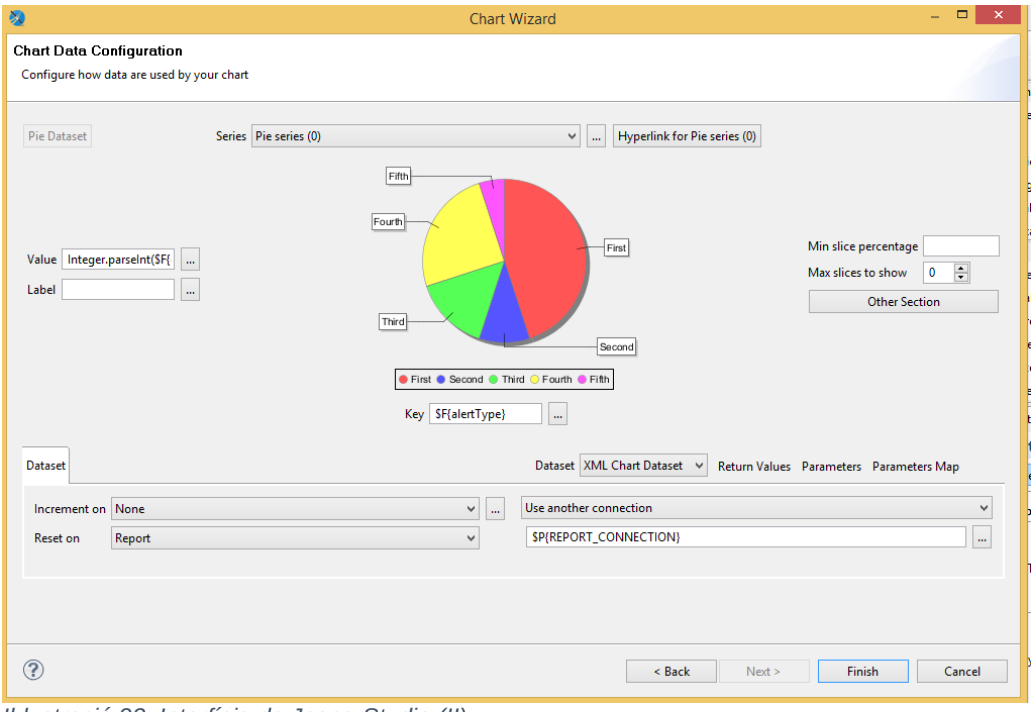
- JasperReports Library: Aquest és el conjunt de llibreries Java que permeten crear o executar programàticament els dissenys dels reports, a més de capturar les dades de varies entrades i de generar els resultats en diversos formats.

- Jaspersoft Studio: Aplicació basada en Eclipse que permet un disseny visual dels reports permetent que una persona que no sigui versada programàticament o no tingui coneixements de XML pugui crear un report. Està basada en Eclipse i substitueix la antiga eina, iReport, que te com a fi de vida el Desembre d'aquest any.



Il·lustració 22: Interfície de JasperStudio

Existeixen també altres eines que no farem servir, com el Jaspersoft Server, que permet executar els reports contra un servidor i es pot gestionar remotament. També disposa de una eina ETL per a la integració de dades, similar a Pentaho, i OLAP, per el anàlisi de dades.

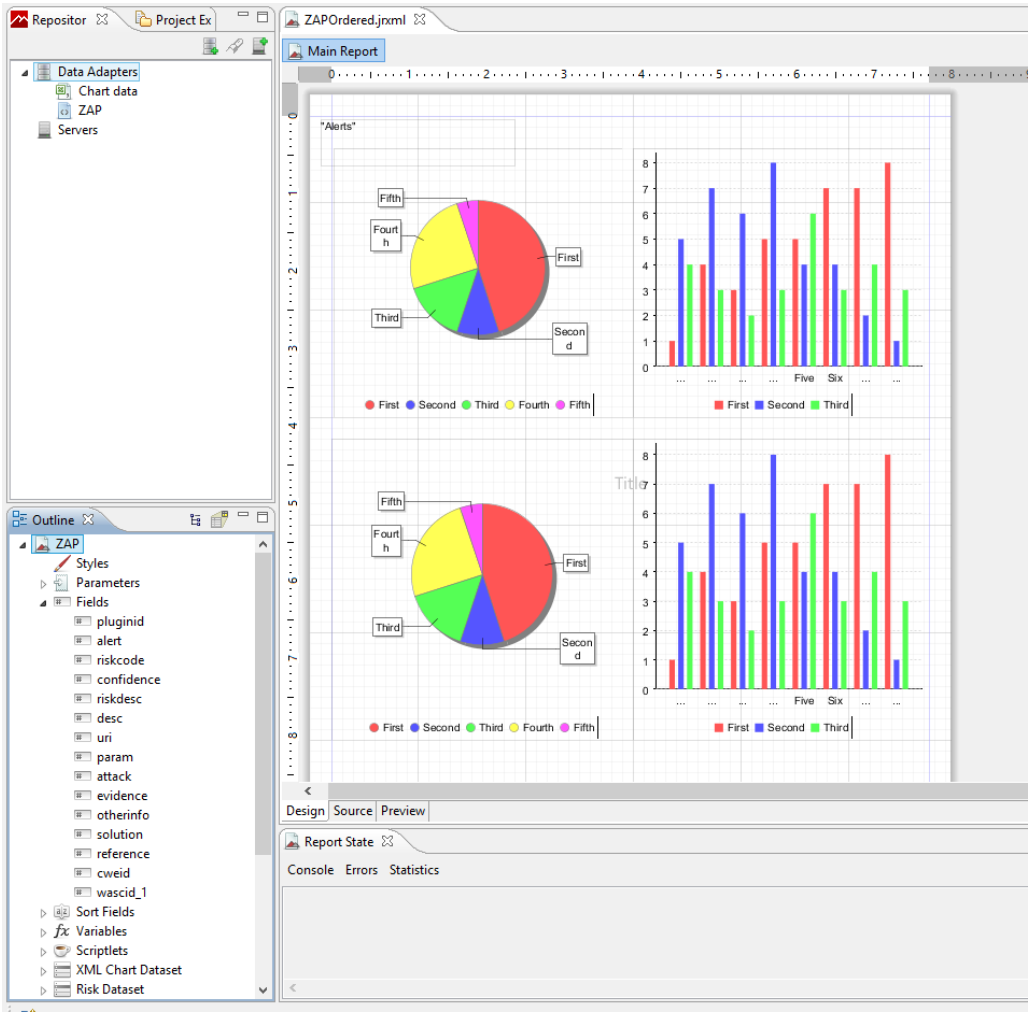


Il·lustració 23: Interfície de JasperStudio (II)

Per el nostre projecte ens interessa sobretot la eina Jaspersoft Studio, que ens permetrà editar els reports de manera visual. Fent-li una ullada veiem que es tracta d'una eina relativament intuïtiva, sobretot si ja estem familiaritzats amb les vistes de Eclipse, i que ens permet entrar en molt detall per a cada un dels components que produïm o estem generant, de manera similar al que feia BIRT, i millorant amb molt la experiència que teníem en aquest sentit amb Pentaho. Per un altre banda, també se'ns permet controlar el posicionament dels elements a nivell de píxel, pel que no tindrem els problemes de control que podríem tenir amb BIRT per a aquest tipus de operacions.

Podríem entendre en alguns sentits que JasperReports és un terme mitjà en quant a complexitat entre Pentaho i BIRT, permetent-nos un gran grau de personalització però no sent tant sobre acollidor al iniciar-nos. El mateix passa amb la generació de gràfics. No és tant potent com BIRT però ho és molt més que Pentaho. Si que podem dir que probablement en BIRT els gràfics siguin més agradables visualment.

Es tracta d'un concepte una mica diferent donat que els reports es compilen abans de fer-se servir, de manera diferent a les dues propostes anteriors. Es poden utilitzar com a dades d'entrada objectes de Java (POJOs), un tret que també el diferencia dels altres dos.



Il·lustració 24: Interfície de JasperStudio (III)

En termes generals JasperReports ens dona molta flexibilitat a l'hora de crear els reports, i s'integra bé amb fonts externes. Els gràfics es generen de forma eficient, i és molt senzill d'instal·lar per a treballar-hi, cosa que pot ajudar si tenim un usuari que no està acostumat a treballar amb Eclipse, per exemple, al no haver-se de instal·lar com a add-on d'aquest. Com a punt més interessant, per això, trobem que és bàsicament una eina en la qual encara es desenvolupa de manera activa, que sembla que tingui futur i que, a més, està sent feta servir per una enorme base de usuaris, pel que el coneixement de la seva utilització ja no seria un pas d'aprenentatge per a un percentatge dels usuaris de el nostre projecte.

Tot i això a vegades trobem que és difícil fer servir el dissenyador, sobretot per a tasques complexes, i és difícil manipular dades al treballar amb gràfics o si ho necessitem per a qualsevol altre fi.

Quan ho comparem amb les altres eines trobem que aquesta és la que més s'adiu al que estem buscant per múltiples motius. Primer, i principalment, perquè la base de usuaris és la més àmplia. Segon, perquè és un projecte viu i amb vistes de créixer, i perquè ens permet un control molt elevat sobre el nostre report. Creiem que aquests dos punts superen

els arguments en contra com son la difícil adaptació al treball amb els reports en alguns casos i el fet de que s'hagi de compilar el report abans de fer-ho servir.

Il·lustració 25: Report de JasperStudio

2.4.4 Creació de un add-on d'exemple

Un cop ja hem decidit quin motor de reporting farem servir per a la creació del add-on comencem a mirar com farem aquest reporting. Com hem fet una ullada a com funciona JasperReports (i tota la resta de eines de reporting) veiem que necessitem un "*datasource*" o origen de dades amb les dades adients per a poder començar a treballar, així que ens plantegem com crear aquest datasource.

La idea inicial seria connectar directament JasperReports amb la base de dades que es fa servir per a la gestió de la sessió, però al cap de poc ens adonem que probablement, tot i ser la manera més senzilla, no és la més eficient ni la que té més futur.

En primera instància trobem que no resulta tan senzill connectar-se a la base de dades de ZAP. Això passa perquè cada sessió és una nova instància de la base de dades i, per tant, canvia de ruta. A més a més, està el fet de que estem desenvolupant un add-on que correrà dintre de ZAP, pel que la base de dades ja estarà en ús per part de ZAP mateix. Si bé això no acostuma a representar un problema per a les bases de dades habituals, ZAP fa servir HSQLDB, que per defecte només ens permetrà una connexió simultània donat que és una base de dades local en fitxer.

Si tot això no és prou, també està el fet de que ZAP està migrant per a canviar el sistema de base de dades (si bé no tenen data sobre quan es realitzarà la migració), i la idea és poder passar a una base de dades basada en MySQL que permeti múltiples connexions i una millor gestió i organització de les sessions des del programa mateix.

Un cop vist això i després de investigar quin tipus d'entrada seria adient per a JasperReports veiem que una entrada XML seria probablement el que millor ens convindria per a poder treballar de manera còmoda. Podríem també passar-li objectes Java directament però volem tenir la possibilitat de deslligar si existeix la possibilitat la entrada que nosaltres proveïm amb la generació del report, donat que pot facilitar l'hora de fer tests o fins i tot pot ajudar a algú que pensi en agafar la nostre sortida per a realitzar una proposta més elaborada de report i necessiti diverses sortides de sessions.

Comencem doncs a investigar com aconseguir una sortida XML amb la informació que trobem serà necessària per a la realització del report, però quina és aquesta informació? Què volem transmetre exactament? Com a resposta general podem dir que el que volem és aconseguir informació de les alertes que s'estan produint al lloc que estiguem analitzant. El nostre objectiu és per tant investigar com accedir a les alertes que es produeixen com a resultat de ZAP i donar-les en un format que sigui adient per a treballar amb JasperReports.

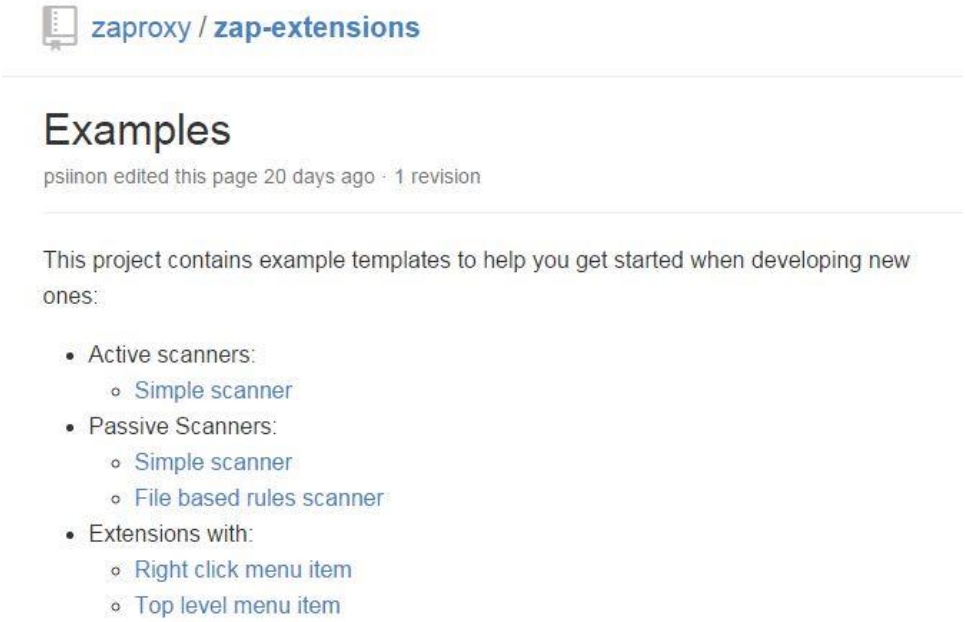
Abans de tot això caldrà, principalment, entendre com podem fer que passi alguna cosa dintre de un add-on que creem nosaltres. De fet, primer hem d'aconseguir crear un simple add-on que no faci res més que compilar i donar algun tipus de resposta i després ja podem ocupar-nos d'aconseguir la informació que veiem adient.

2.4.4.1 Estructura del add-on

Primer de tot hem de aconseguir entendre dues coses: Com afegim un add-on nou a les extensions de ZAP i quina és la estructura en la que es basen per treballar?

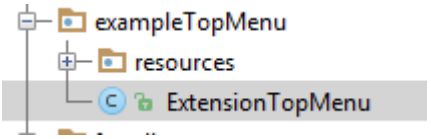
Volem que el nostre add-on sigui consistent amb el ecosistema de les altres extensions que existeixen a ZAP, pel que observem les extensions d'exemple que existeixen dintre de ZAP per a crear-ne una de nova.

A la wiki de ZAP podem veure que existeixen varius tipus de extensions, basats en el que es vol aconseguir. Els primers tipus de extensions que trobem son aquelles que volen aconseguir estendre les normes de escaneig passiu i actiu, tot i que no és el nostre objectiu. Un tercer grup d'extensions que si ens interessa, doncs, és el grup de extensions que permet més llibertat a l'hora d'estendre la funcionalitat de ZAP, doncs és el que ens permetrà afegir nous elements al menú i donar-los la funcionalitat que desitgem:



Il·lustració 26: Tipus de extensions de ZAP

Observem que dintre del repositori també existeixen aquests exemples:



Il·lustració 27: Classe d'exemple

Fem una ullada doncs al codi per veure què en podem trobar:

```
/*
 * Zed Attack Proxy (ZAP) and its related class files.
 *
 * ZAP is an HTTP/HTTPS proxy for assessing web application
security.
 *
 * Licensed under the Apache License, Version 2.0 (the
"License");
 * you may not use this file except in compliance with the
License.
 * You may obtain a copy of the License at
 *
 *   http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing,
software
 * distributed under the License is distributed on an "AS IS"
```

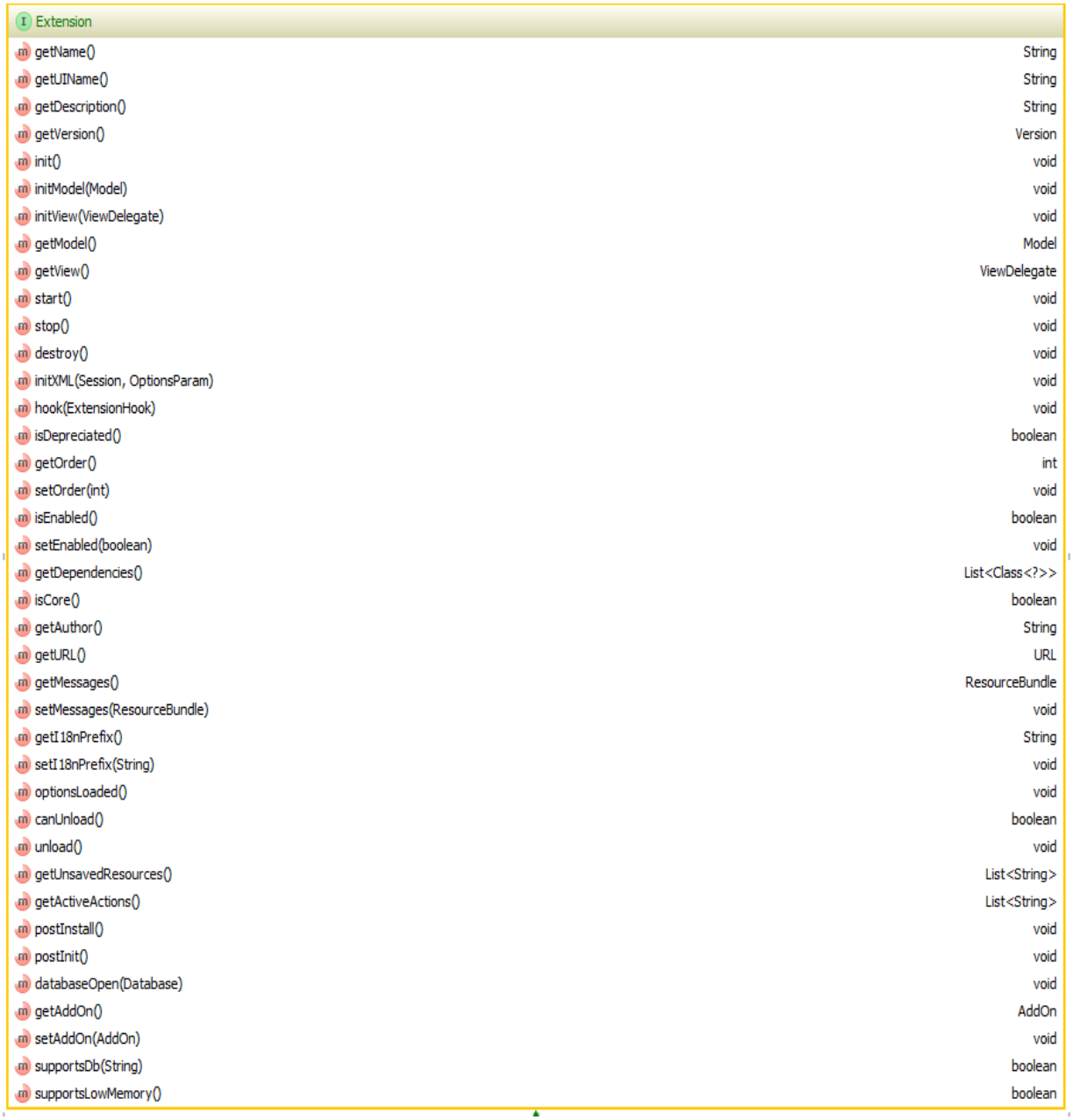
```
BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
or implied.
 * See the License for the specific language governing
permissions and
 * limitations under the License.
 */
package org.zaproxy.zap.extension.exampleTopMenu;

import java.net.MalformedURLException;
import java.net.URL;
import java.util.ResourceBundle;

import javax.swing.JMenuItem;

import org.parosproxy.paros.Constant;
import org.parosproxy.paros.extension.ExtensionAdaptor;
import org.parosproxy.paros.extension.ExtensionHook;
import org.parosproxy.paros.view.View;

/*
 * An example ZAP extension which adds a top level menu item.
 *
 * This class is defines the extension.
 */
public class ExtensionTopMenu extends ExtensionAdaptor {

    private JMenuItem menuExample = null;
    private ResourceBundle messages = null;

   /**
     *
     */
    public ExtensionTopMenu() {
        super();
      initialize();
    }

    /**
     * @param name
     */
    public ExtensionTopMenu(String name) {
        super(name);
    }

   /**
    * This method initializes this
    *
    */
   private void initialize() {
        this.setName("ExtensionTopMenu");
        // Load extension specific language files - these are
held in the extension jar
        messages = ResourceBundle.getBundle(
              this.getClass().getPackage().getName() +
".resources.Messages", Constant.getLocale());
   }

   @Override
   public void hook(ExtensionHook extensionHook) {
       super.hook(extensionHook);
```

```
        if (getView() != null) {
            // Register our top menu item, as long as we're not
running as a daemon
            // Use one of the other methods to add to a different
menu list

extensionHook.getHookMenu().addToolsMenuItem(getMenuExample());
        }

    }

    private JMenuItem getMenuExample() {
        if (menuExample == null) {
            menuExample = new JMenuItem();

menuExample.setText(getMessageString("ext.topmenu.tools.example"
));

            menuExample.addActionListener(new
java.awt.event.ActionListener() {
                @Override
                public void
actionPerformed(java.awt.event.ActionEvent e) {
                  // This is where you do what you want to do.
                  // In this case we'll just show a popup
message.

View.getSingleton().showMessageDialog(getMessageString("ext.topm
enu.msg.example"));
                }
            });
        }
        return menuExample;
    }

    public String getMessageString (String key) {
        return messages.getString(key);
    }
    @Override
    public String getAuthor() {
        return Constant.ZAP_TEAM;
    }

    @Override
    public String getDescription() {
        return messages.getString("ext.topmenu.desc");
    }

    @Override
    public URL getURL() {
        try {
            return new URL(Constant.ZAP_EXTENSIONS_PAGE);
        } catch (MalformedURLException e) {
            return null;
        }
    }
}
```

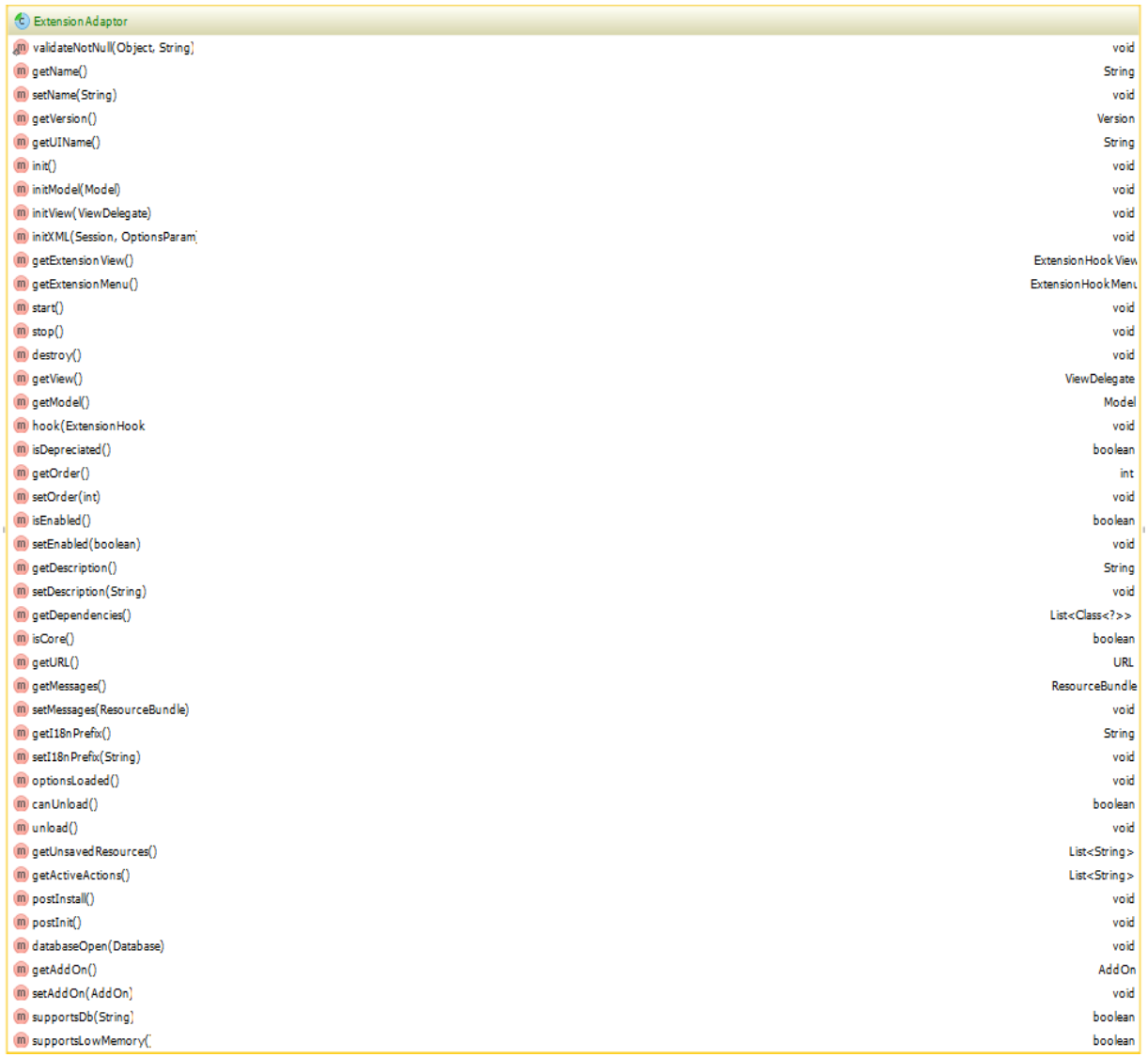
Il·lustració 28: Codi de classe d'extensió d'exemple

Observem primerament que estenem la classe *ExtensionAdaptor*. Aquesta classe implementa la interfície *Extension*, i les dues son part del codi que ZAP adopta del codi de Paros Proxy, un projecte de OWASP en el que ZAP es va basar que no es desenvolupa des del 2006. Les extensions de ZAP, fins i tot les internes, es desenvolupen a partir de la implementació i extensió d'aquesta interfície i les seves implementacions.

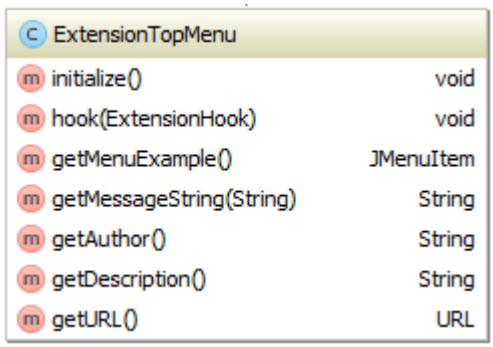
Podem fer una ullada als mètodes de les classes amb les que treballarem, per tenir una idea de els mètodes disponibles a cada extensió i la estructura que segueixen:



Il·lustració 29: Mètodes de la classe Extension

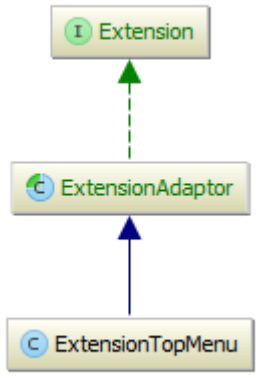


Il·lustració 30: Mètodes de ExtensionAdapter



Il·lustració 31: Mètodes de ExtensionTopMenu

Les relacions entre les tres classes son senzilles:



Il·lustració 32: Diagrames de les extensions

Un cop posada en context, podem analitzar el funcionament de la extensió per a poder crear-ne una personalitzada:

```
public ExtensionTopMenu() {
      super();
      initialize();
 }
```

Primer veiem que al constructor s'invoca al mètode de inicialització de *ExtensionAdaptor*, i més tard s'inicialitzen els paràmetres propis de la extensió.

```
/**
 * This method initializes this
 *
 */
private void initialize() {
       this.setName("ExtensionTopMenu");
       // Load extension specific language files - these are
held in the extension jar
       messages = ResourceBundle.getBundle(
              this.getClass().getPackage().getName() +
".resources.Messages", Constant.getLocale());
}
```

En la inicialització del nostre exemple d'extensió podem veure que es realitza una inicialització del nom de la extensió i seguidament es carreguen els missatges específics per a cada localització per a poder fer-los servir. Això ho analitzarem més tard.

```
@Override
public void hook(ExtensionHook extensionHook) {
    super.hook(extensionHook);

    if (getView() != null) {
       // Register our top menu item, as long as we're not
running as a daemon
```

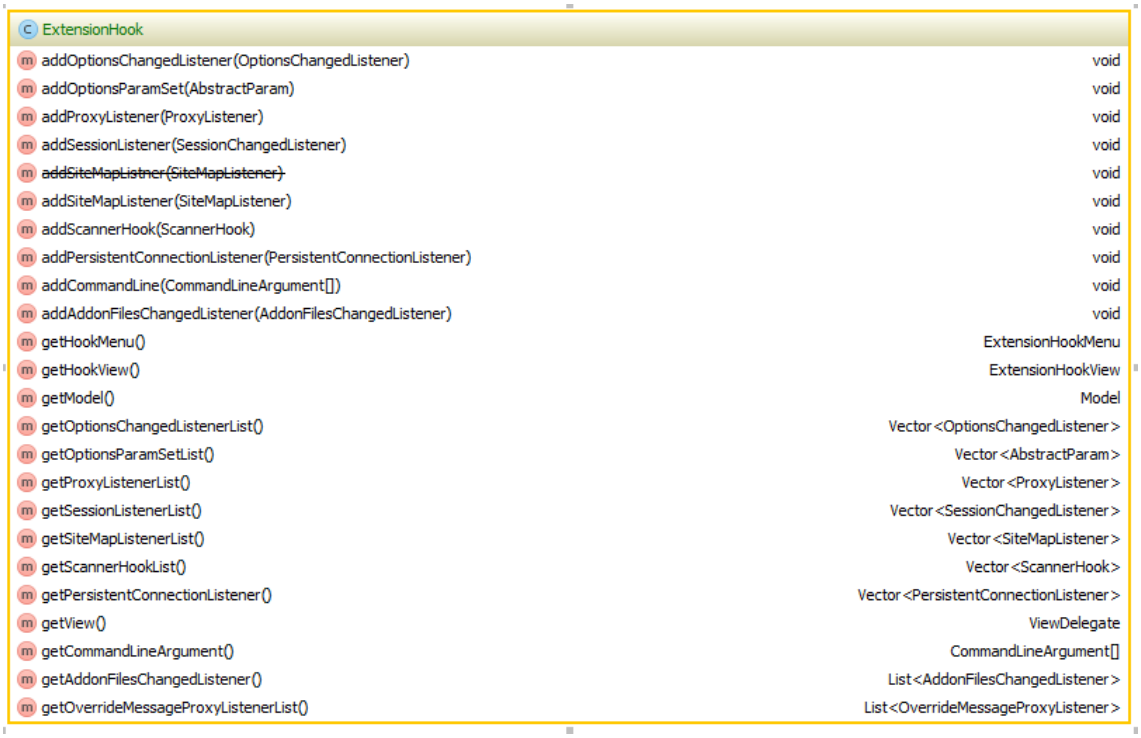
```
        // Use one of the other methods to add to a different
menu list

extensionHook.getHookMenu().addToolsMenuItem(getMenuExample());
    }

}
```

Arribem a parts més interessants o complicades. Veiem que sobreescrivim el mètode "*hook*", on cridem a *ExtensionHook*, que es tracta de un dispositiu de ZAP que ens permet lligar les nostres classes amb ZAP fent servir una sèrie de interfícies estàndard.

A la wiki de ZAP se'ns proporciona una llista de interfícies de les que disposarem:

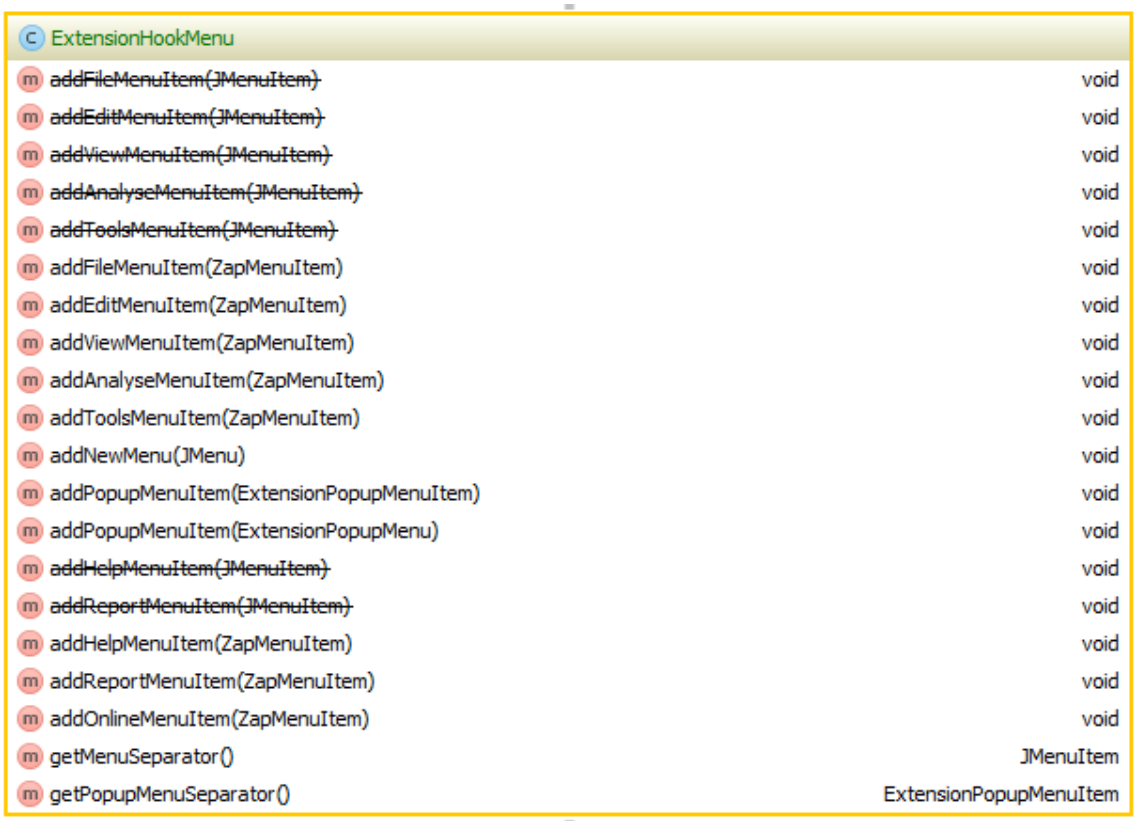
Interface	Summary
ProxyListener	Allows you to view and change all requests and responses
OptionsChangedListener	Called when options are changed
SessionChangedListener	Called when significant changes are made to the session
AbstractParam	Allows you to hook into the mechanism for persisting parameters
SiteMapListener	Called when nodes are selected in the Sites tree
PersistentConnectionListener	Called when persistent connection handshakes occur
AddonFilesChangedListener	Called when Add-on files are added or removed

Il·lustració 33: Interficies de ZAP



Il·lustració 34: Mètodes de ExtensionHook

Aquí serà on podrem afegir per tant els elements de la interfície que vulguem afegir. En aquest cas estarem afegint un element de menú a la classe *ExtensionHookMenu*, que és el lligam que tindrem amb els elements del menú:



Il·lustració 35: Mètodes de ExtensionHookMenu

Un cop fet això mirarem com creem exactament el element de menú per a poder-lo afegir:

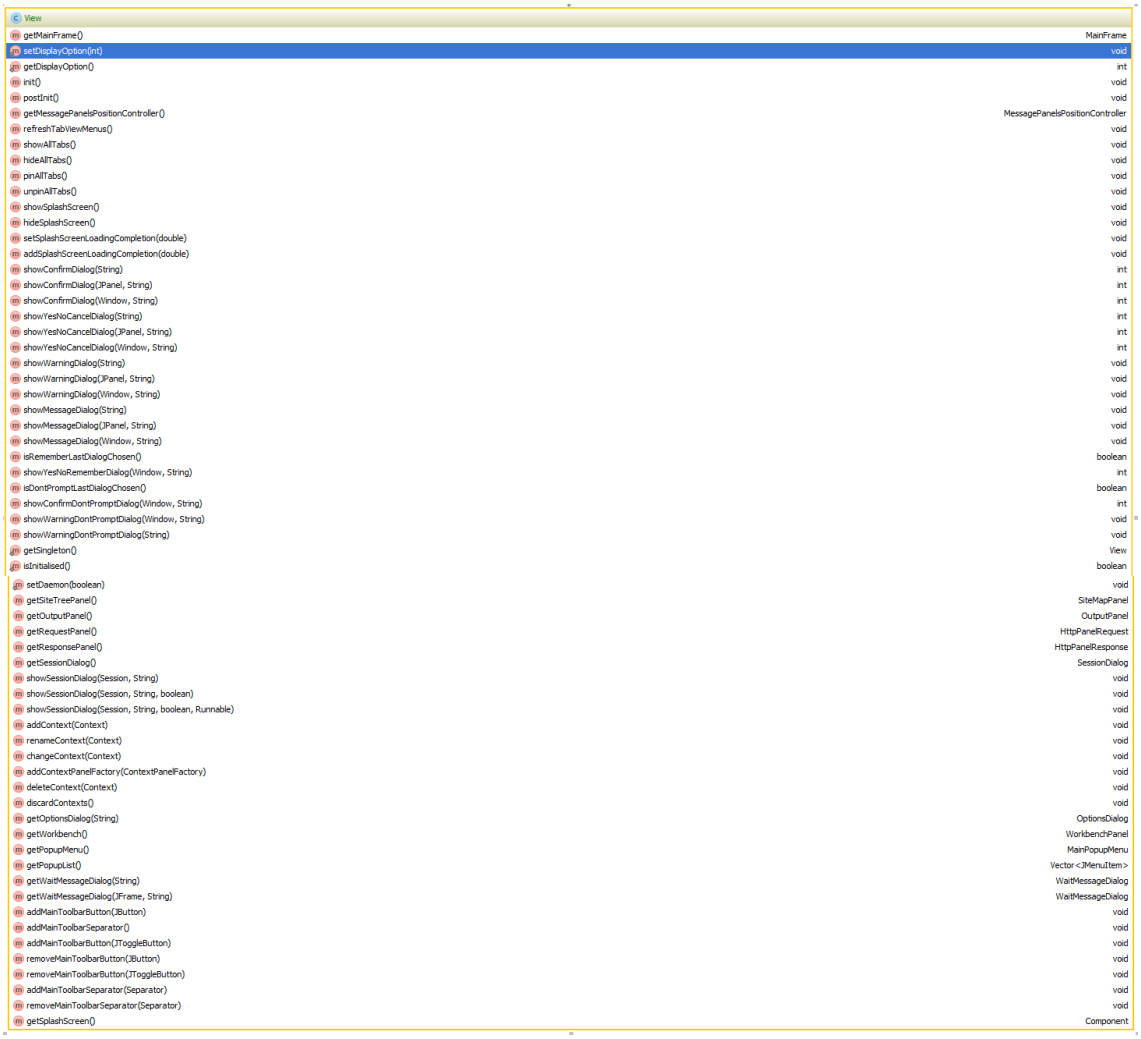
```
private JMenuItem getMenuExample() {
        if (menuExample == null) {
            menuExample = new JMenuItem();

menuExample.setText(getMessageString("ext.topmenu.tools.example"
));

            menuExample.addActionListener(new
java.awt.event.ActionListener() {
                @Override
                public void
actionPerformed(java.awt.event.ActionEvent e) {
                    // This is where you do what you want to do.
                    // In this case we'll just show a popup
message.

View.getSingleton().showMessageDialog(getMessageString("ext.topm
enu.msg.example"));
                }
            });
```

```
        }
        return menuExample;
    }
```

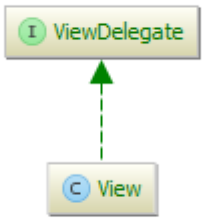
El element de menú que afegim és un element estàndard de menú de Java, un *JMenuItem*. En aquest afegirem un escoltador de accions (*ActionListener*) que ens permetrà saber quan s'ha produït una acció relacionada amb el element de menú que acabem de afegir i actuar-hi en conseqüència. En el cas del nostre exemple s'afegeix una vista en la que es proporcionarà un missatge d'exemple.

El nostre element de menú tindrà un text que afegim, com el missatge de diàleg de la vista que es farà servir com a acció, en base als missatges localitzats. Donem per tant una clau que servirà per agafar el contingut del missatge més endavant depenent de la localització del sistema.

La classe que fem servir per la vista també és heretada de Paros Proxy, la classe *View*, i ens serveix per gestionar diferents tipus d'accions relacionades amb les vistes.



Il·lustració 36: Mètodes de View



Il·lustració 37: Diagrama de View

Un cop ja hem definit la acció que es realitzarà ja tenim les parts més rellevants del codi situades. La resta de codi ens ajuda a obtenir els missatges localitzats, el autor de la extensió i la URL de les extensions de ZAP:

```
public String getMessageString (String key) {
    return messages.getString(key);
}
@Override
public String getAuthor() {
    return Constant.ZAP_TEAM;
}

@Override
public String getDescription() {
    return messages.getString("ext.topmenu.desc");
}

@Override
public URL getURL() {
    try {
        return new URL(Constant.ZAP_EXTENSIONS_PAGE);
    } catch (MalformedURLException e) {
        return null;
    }
}
```

2.4.4.2 Missatges i localització

Les extensions a ZAP que tenen diferents texts a mostrar que poden ser traduïts es gestionen mitjançant un fitxer de propietats de recursos de missatges. De manera general, el que es necessita és un fitxer amb propietats on es trobin els codis de missatges que fem servir al codi. Un cop fet servir, en temps de execució es busca aquest codi en el fitxer de missatges corresponent al idioma del sistema. En cas de no trobar-se es va a buscar el missatge al fitxer de missatges per defecte.

D'aquesta manera podem gestionar diferents idiomes i facilitar la localització de ZAP, però no n'estem obligats. El que si que haurem de tenir és el nostre fitxer de *Message.properties* per a poder obtenir els missatges per defecte.
Continguts del fitxer *Message.properties*:

```
# An example ZAP extension which adds a top level menu item.
#
# This file defines the default (English) variants of all of the
internationalised messages

ext.topmenu.desc=Example extension which provides a top level
menu
ext.topmenu.tools.example=topmenu: an example menu
ext.topmenu.msg.example=topmenu: An example message
```

2.4.4.3 Personalització de el nostre add-on.

Per a poder treballar amb aconseguir informació de les alertes agafem com a base el add-on de exemple que hem estat estudiant. Treballar amb ell és qüestió doncs de crear una nova carpeta dintre de la branca *alpha*, on estarà allotjat. Un cop creem aquesta carpeta, copiem els continguts del fitxer Java amb el codi de la aplicació i dels recursos dels missatges per a poder-hi treballar, el hi posem el nom que creiem adient i estem pràcticament preparats per a treballar-hi.

Per a poder-hi treballar del tot haurem de afegir el add-on a la llista de objectius de Ant de la branca *alpha*.

```
<target name="deploy-jasper" description="deploy the simple
jasper extension">
    <build-deploy-addon name="jasperReports" />
</target>
```

Aquest objectiu fa servir la macro *build-deploy-addon* que s'encarrega de compilar i posicionar la extensió que hem realitzat. Podem veure els diferents continguts de la macro.

Primer veiem la macro per copiar els continguts i per compilar:

```
<macrodef name="deploy-addon" description="deploy the specified
extension">
    <attribute name="name"/>
    <sequential>
        <copy todir="${zap.plugin.dir}">
            <fileset dir="${dist}">
            <include name="@{name}-*.zap"/>
            </fileset>
        </copy>
    </sequential>
</macrodef>

<macrodef name="build-deploy-addon" description="build and
deploy the specified addon">
    <attribute name="name"/>
```

```
    <sequential>
        <antcall target="clean" />
        <antcall target="compile" />

        <build-addon name="@{name}" />
        <deploy-addon name="@{name}" />
    </sequential>
</macrodef>
```

Seguidament podem veure com es realitza la compilació:

```
<macrodef name="build-addon" description="build the specified
addon">
    <attribute name="name"/>
    <element name="extra-actions" implicit="true" optional="true"
/>
    <sequential>
        <local name="zapaddon.version" />
        <xmlproperty
file="${src}/org/zaproxy/zap/extension/@{name}/ZapAddOn.xml"/>
        <local name="file" />
        <property name="file" value="@{name}-${status}-
${zapaddon.version}.zap" />

        <generatejavahelpsearchindexes
jhalljar="${dist.lib.dir}/jhall.jar"
            helpcontentsdirname="contents"
helpsetfilename="helpset*.hs">
            <dirset
dir="${src}/org/zaproxy/zap/extension/@{name}/resources/">
                <include name="help" />
                <include name="help_*_*" />
            </dirset>
        </generatejavahelpsearchindexes>

        <local name="addon.libs.zip" />
        <property name="addon.libs.zip" value="${temp}/libs-
@{name}.zip" />

        <delete file="${addon.libs.zip}" failonerror="true" />
        <zip destfile="${addon.libs.zip}" whenempty="create">
            <zipgroupfileset
dir="${src}/org/zaproxy/zap/extension/@{name}/lib/"
includes="*.jar" erroronmissingdir="false" />
        </zip>

        <jar jarfile="${dist}/${file}" update="true"
compress="true">
            <zipfileset dir="${build}" prefix="">
                <include
name="org/zaproxy/zap/extension/@{name}/**"/>
            </zipfileset>
            <zipfileset dir="${src}" prefix="">
                <include
name="org/zaproxy/zap/extension/@{name}/Messages*"/>
            </zipfileset>
            <zipfileset dir="${src}" prefix="">
                <include
name="org/zaproxy/zap/extension/@{name}/resources/**"/>
            </zipfileset>
            <zipfileset src="${addon.libs.zip}">
```

```
            <exclude name="META-INF/*.DSA" />
            <exclude name="META-INF/*.SF" />
        </zipfileset>
        <zipfileset dir="${src}"
includes="org/zaproxy/zap/extension/@{name}/ZapAddOn.xml"
fullpath="ZapAddOn.xml"/>
      </jar>
      <delete file="${addon.libs.zip}" />

      <!-- Remove search indexes previously generated, no longer
needed. -->
      <delete
dir="${src}/org/zaproxy/zap/extension/@{name}/resources/help/Jav
aHelpSearch" />
      <delete
dir="${src}/org/zaproxy/zap/extension/@{name}/resources/help_*_*
/JavaHelpSearch" />

      <!-- Include add-on files -->
      <jar jarfile="${dist}/${file}" update="true"
compress="true">
        <zipfileset
dir="${src}/org/zaproxy/zap/extension/@{name}/files/" prefix=""
erroronmissingdir="false" />
      </jar>

      <!-- allow callers to do extra actions before generating
the hash and determine the size of the file -->
      <extra-actions />

      <local name="length" />
      <length file="${dist}/${file}" property="length" />

      <local name="sha1hash" />
      <checksum file="${dist}/${file}"  algorithm="SHA-1"
property="sha1hash"/>

      <local name="hash" />
      <property name="hash"  value="SHA1:${sha1hash}"/>

      <local name="yyyymmdd" />
      <tstamp>
        <format property="yyyymmdd" pattern="yyyy-MM-dd"/>
      </tstamp>

      <appendzapaddonfile
from="${src}/org/zaproxy/zap/extension/@{name}/ZapAddOn.xml"
to="${versions.file}"
        addonid="@{name}" filename="${file}" status="${status}"
size="${length}" hash="${hash}" date="${yyyymmdd}"
        url="http://sourceforge.net/projects/zaproxy/files/add-
ons/${file}/download" />

    </sequential>
</macrodef>
```

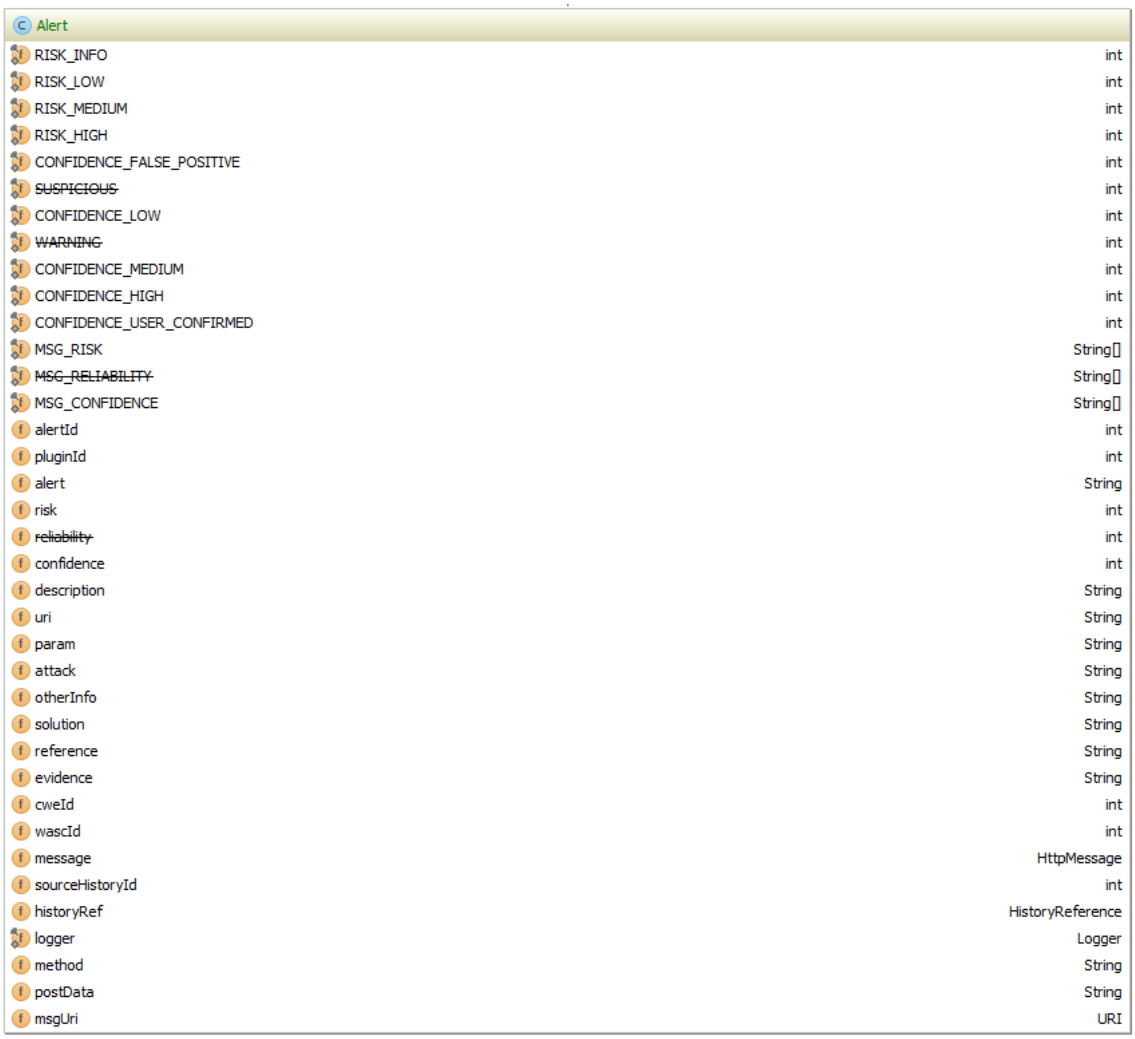
Un cop l'hem afegit ja només ens queda afegir-lo a la llista de objectius generals per quan es faci una compilació general també la tinguem:

```
<target name="build-all" depends="clean,compile"
description="build all of the extensions">
    <delete file="${versions.file}"/>

    <!-- 'Standard' addons (keep in alphabetical order ;) -->
    <build-addon name="accessControl" />
    <build-addon name="amf" />
    <build-addon name="ascanrulesAlpha" />
    <build-addon name="birtreports" />
    <build-addon name="browserView" />
    <build-addon name="callgraph" />
    <build-addon name="cmss" />
    <build-addon name="highlighter" />
    <build-addon name="httpsInfo" />
    <build-addon name="importLogFiles" />
    <build-addon name="pscanrulesAlpha" >
          <!-- Add the extra classes required -->
          <jar jarfile="${dist}/${file}" update="true"
compress="true">
             <zipfileset dir="${build}" prefix="">
                <include
name="com/veggiespam/imagelocationscanner/**"/>
             </zipfileset>
          </jar>
       </build-addon>
    <build-addon name="saml" />
    <build-addon name="sequence" />
    <build-addon name="simpleExample" />
    <build-addon name="jasperReports" />
    <build-addon name="sniTerminator" />
    <build-addon name="soap" />
    <build-addon name="sse" />
    <build-addon name="wappalyzer" />

</target>
```

I amb això ja tenim finalitzada la creació de un add-on d'exemple amb el que podem treballar per a aconseguir les dades que volem.

2.4.5 Obtenció de les dades de alertes

La representació de les vulnerabilitats a ZAP es realitza en forma de alertes. La classe *Alert* manifesta els diferents atributs de que disposen aquestes alertes, que serà la informació que voldrem representar a la nostra extensió:



Il·lustració 38: Mètodes de Alert

Veiem que estan suportats diferents camps que ens proporcionen informació:

pluginId: Identifica el scanner que s'ha fet servir per a trobar l'alerta, i és útil per a diferents aplicacions que fan servir la API de ZAP.

name: Un nom de resum per al usuari.

risk: Una indicació de com d'important és el risc:

Alert.RISK_INFO: Vulnerabilitat que dona només informació, no es considera un risc pròpiament.

Alert.RISK_LOW: Vulnerabilitat de risc baix.

Alert.RISK_MEDIUM: Vulnerabilitat de risc mitjà.

Alert.RISK_HIGH: Vulnerabilitat de risc alt.

reliability: Indicació de les probabilitats de que es tracti d'un problema real:

Alert.FALSE_POSITIVE: Falsos positius, no haurien de ser considerats de manera general donat que es considera que no son fiables.

Alert.SUSPICIOUS: Nivell mitjà de confiança, més baix que el nivell alt.

Alert.WARNING: Nivell alt de confiança.

description: Una descripció detallada de la vulnerabilitat.

uri: La adreça URI afectada.
param: Nom del paràmetre rellevant, si escau.
attack: La cadena de caràcters feta servir com a atac. Aquesta propietat no es fa servir en atacs passius.
otherinfo: Informació addicional que pot resultar útil però que no pertany a cap dels altres paràmetres.
solution: Informació sobre com evitar o prevenir aquesta vulnerabilitat, si existeix.
reference: Llista de llocs web que poden donar més informació sobre la vulnerabilitat que es tracta, separades per retorns de línia.
evidence: Cadena de caràcters que es donen o bé a la petició o a la resposta que es poden fer servir com a evidències de la vulnerabilitat.
cweId: El CWE id
wascId: el WASC Threat Classification id.

Coneixem per tant la informació que volem extreure per a poder-la fer servir als reports, pel que hem de descobrir com aconseguir treure-la basat en el context de una extensió.

Aquesta part juntament amb el disseny del report en si (especialment el treure la informació) és on s'ha patit relativament més en el projecte. No es tracta de funcionalitats complicades o difícils, però si estan mal documentades, en general. Hem hagut de inferir els significats de moltes classes i conceptes, i estudiar el codi de diverses extensions per veure com es realitzen diverses accions i treballar a partir d'aquestes en poder extreure la informació que considerem més rellevant.

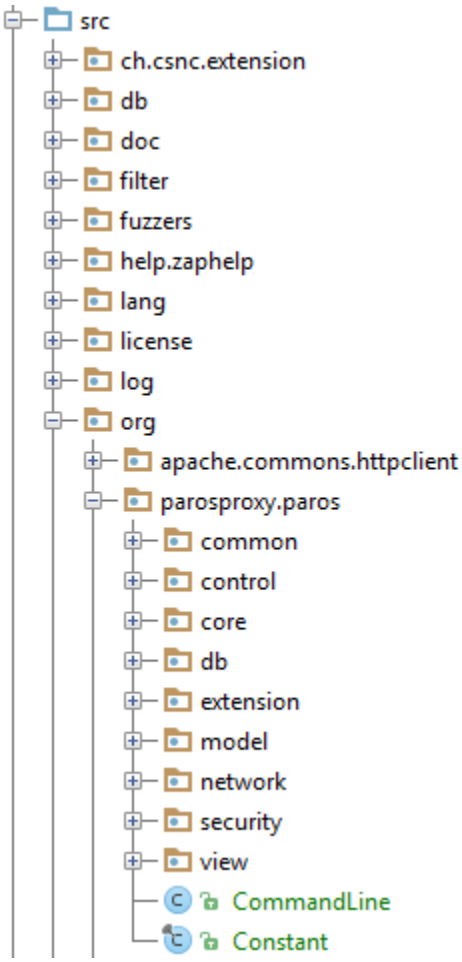
Sabem que ZAP està basat en Paros, i que molta de la funcionalitat bàsica o principal es troba entre les classes que formen part d'aquest antic projecte. Entenem que el que volem fer, en primera instància, és aconseguir la informació de sessió per a poder veure si allà trobem la representació de les alertes, i poder-les passar a un format que ens interessi per el *DataSource* que li passarem a Jasper.

Busquem sense èxit una referència a com fer això a la Wiki o a altres llocs que ens puguin donar informació de ZAP, però no trobem informació rellevant de desenvolupament que ens pugui ajudar en aquest cas concret. Fem una ullada, però, al exemple que hem analitzat abans i veiem aquet extracte de codi:

```
View.getSingleton().showMessageDialog(getMessageString("ext.topm
enu.msg.example"));
```

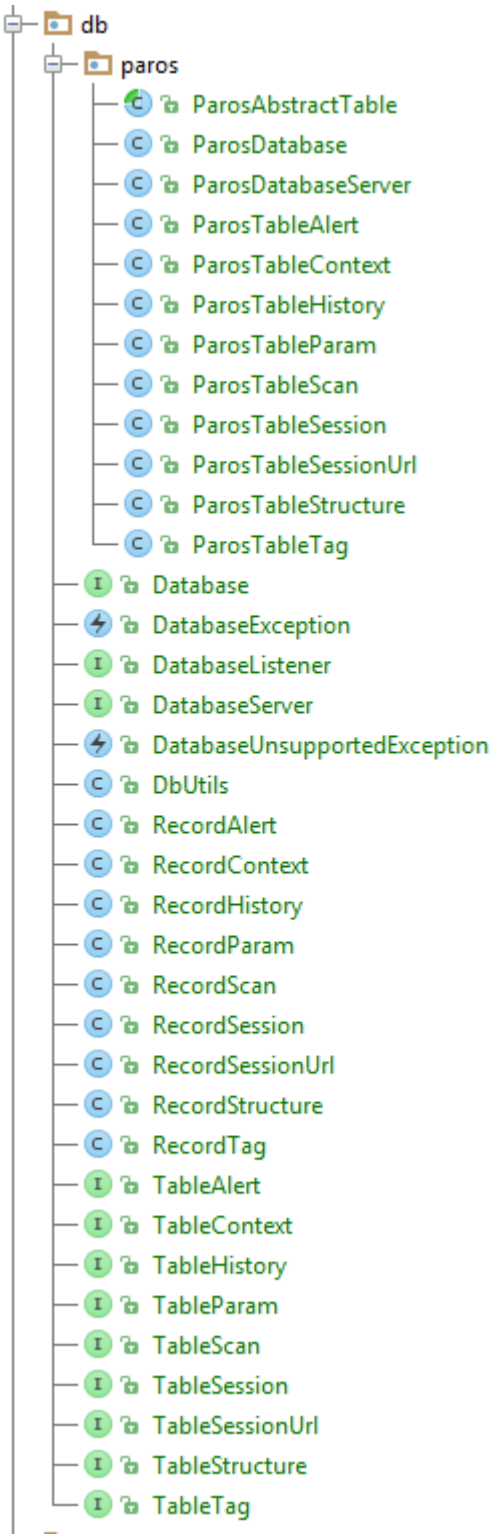
Pensem doncs que potser el que hem de fer és aconseguir instanciar la capa rellevant que ens proporcioni accés a la sessió.

Fem una ullada doncs a la estructura del codi de Paros que està a ZAP:

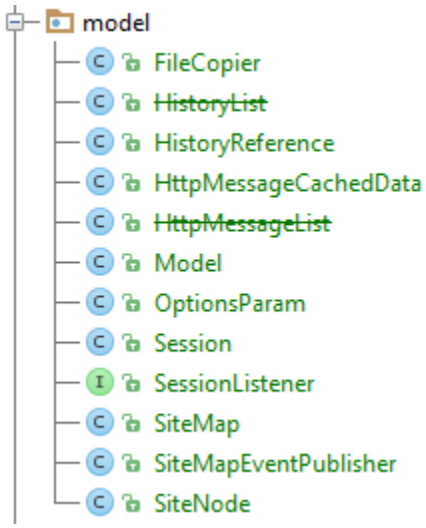


Il·lustració 39: Estructura de Paros

A nivell intuïtiu veiem al mateix nivell que la capa de Vista, representada per *View*, les capes de Base de dades *db* i de Model: *Model.*



Il·lustració 40: Estructura de Paros (II)



Il·lustració 41: Estructura de Paros (III)

Per sort veiem que a *Model* tenim el objecte de sessió que podria ser el que busquem. L'instanciarem doncs de manera similar a com fèiem amb la vista:

```
Session session = Model.getSingleton().getSession();
```

És lògic que el objecte *Model* sigui el que contingui les dades donat que segueix la nomenclatura típica del disseny de software. Observarem quins mètodes podem veure en la sessió en si:

Method and Description
addContext(**Context** c)
addExcludeFromProxyRegex(**String** ignoredRegex)
addExcludeFromScanRegexs(**String** ignoredRegex)
addExcludeFromSpiderRegex(**String** ignoredRegex)
addGlobalExcludeURLRegexs(**String** ignoredRegex) TODO The GlobalExcludeURL functionality is currently alpha and subject to change.
addOnContextsChangedListener(**Session.OnContextsChangedListener** l)
clearContextDataForType(int contextId, int type)
close()
discard()
forceGlobalExcludeURLRefresh() TODO The GlobalExcludeURL functionality is currently alpha and subject to change.
getContext(int index)
getContext(**String** name)
getContextDataStrings(int contextId, int type)
getContexts()
getContextsForNode(**SiteNode** sn)
getContextsForUrl(**String** url)
getExcludeFromProxyRegexs()
getExcludeFromScanRegexs()
getExcludeFromSpiderRegexs()
getFileName()
getFormParamParser(**String** url) Returns the form parameter parser associated with the first context found that includes the URL, or the default parser if it is not in a context
getFormParams(org.apache.commons.httpclient.URI uri, **String** formData) Returns the FORM parameters for the given URL based on the parser associated with the first context found that includes the URL, or the default parser if it is not in a context
getGlobalExcludeURLRegexs()

TODO The GlobalExcludeURL functionality is currently alpha and subject to change.

getNewContext()

getNodesInContextFromSiteTree(**Context** context)

Gets the nodes from the site tree which are "In Scope" in a given context.

getNodesInScopeFromSiteTree()

Gets the nodes from the site tree which are "In Scope".

getParams(**HttpMessage** msg, **HtmlParameter.Type** type)

Returns the specified parameters for the given message based on the parser associated with the first context found that includes the URL for the message, or the default parser if it is not in a context

getSessionDesc()

getSessionFolder()

getSessionId()

getSessionName()

getSessionUrls(int type)

getSiteTree()

getTreePath(org.apache.commons.httpclient.URI uri)

getUrlParamParser(**String** url)

Returns the url parameter parser associated with the first context found that includes the URL, or the default parser if it is not in a context

getUrlParams(org.apache.commons.httpclient.URI uri)

Returns the URL parameters for the given URL based on the parser associated with the first context found that includes the URL, or the default parser if it is not in a context

isExcludedFromScope(**SiteNode** sn)

isIncludedInScope(**SiteNode** sn)

isInScope(**HistoryReference** href)

isInScope(**SiteNode** sn)

isInScope(**String** url)

isNewState()

Tells whether this session is in a new state or not.

open(**File** file, **SessionListener** callback)
open(**String** fileName)
parse()
removeOnContextsChangedListener(**Session.OnContextsChanged Listener** l)
save(**String** fileName) Synchronous call to save a session.
save(**String** fileName, **SessionListener** callback) Asynchronous call to save a session.
saveAllContexts()
saveContext(**Context** c)
setContextData(int contextId, int type, **List**<**String**> dataList)
setContextData(int contextId, int type, **String** data)
setExcludeFromProxyRegexs(**List**<**String**> ignoredRegexs)
setExcludeFromScanRegexs(**List**<**String**> ignoredRegexs)
setExcludeFromSpiderRegexs(**List**<**String**> ignoredRegexs)
setGlobalExcludeURLRegexs(**List**<**String**> ignoredRegexs) TODO The GlobalExcludeURL functionality is currently alpha and subject to change.
setSessionDesc(**String** sessionDesc)
setSessionId(long sessionId)
setSessionName(**String** name)
setSessionUrl(int type, **String** url)
setSessionUrls(int type, **List**<**String**> urls)
snapshot(**String** fileName) Synchronous call to snapshot a session.
snapshot(**String** fileName, **SessionListener** callback) Asynchronous call to snapshot a session.

Il·lustració 42: Mètodes de la Session

A la llista de mètodes que trobem que ens poden fer servir per a trobar les alertes destaca un en quant ho explorem:

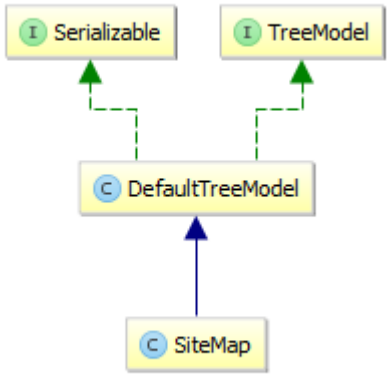
getSiteTree()

Aquest mètode ens permetrà recuperar el llistat de URLs capturats o fets servir durant un anàlisi. Un cop tenim ja la llista de urls només cal que cerquem les alertes que corresponen a cada una de elles
Fem una ullada al *SiteMap* per saber quins paràmetres podem fer servir:

Method and Description
addPath(**HistoryReference** ref) Add the HistoryReference into the SiteMap.
addPath(**HistoryReference** ref, **HttpMessage** msg) Add the HistoryReference with the corresponding HttpMessage into the SiteMap.
createTree(**Model** model)
findNode(**HttpMessage** msg)
findNode(**HttpMessage** msg, boolean matchStructural)
findNode(org.apache.commons.httpclient.URI uri)
findNode(org.apache.commons.httpclient.URI uri, **String** method, **String** postData)
getSiteNode(int href)
pollPath(**HttpMessage** msg) Return the a HttpMessage of the same type under the tree path.
removeHistoryReference(int historyId)

Il·lustració 43: Mètodes de SiteMap

No trobem res que ens serveixi per agafar el objecte que necessitem per treballar amb les alertes, pel que també haurem de consultar els mètodes de la classe que estén, *DefaultTreeModel:*



Il·lustració 44: Diagrama de SiteMap

Method and Description
addTreeModelListener(**TreeModelListener** l) Adds a listener for the TreeModelEvent posted after the tree changes.
getChild(**Object** parent, int index) Returns the child of parent at index index in the parent's child array.
getChildCount(**Object** parent) Returns the number of children of parent.
getIndexOfChild(**Object** parent, **Object** child) Returns the index of child in parent.
getRoot() Returns the root of the tree.
isLeaf(**Object** node) Returns true if node is a leaf.
removeTreeModelListener(**TreeModelListener** l) Removes a listener previously added with addTreeModelListener.
valueForPathChanged(**TreePath** path, **Object** newValue) Messaged when the user has altered the value for the item identified by path to newValue.

Il·lustració 45: Mètodes de DefaultTreeModel

Aquí ja trobem una manera de agafar la arrel del arbre amb el mètode *getRoot:*

getRoot()

Returns the root of the tree.

Podem treballar des d'aquí, tot i que es retorna un tipus *Object* general. A la classe mateixa de *SiteMap* podem trobar un exemple de utilització, que ens dona una idea de que el podríem convertir en un objecte *SiteNode* que sembla ser la representació del node del *Site*:

```
SiteNode resultNode = null;
URI uri = msg.getRequestHeader().getURI();

SiteNode parent = (SiteNode) getRoot();
String folder;
```

Un cop ja obtenim el *SiteNode* que representa la URL que s'ha analitzat es poden obtenir, entre altres coses, les alertes que s'han trobat en ella:

Method and Description
addAlert(**Alert** alert)
addCustomIcon(**String** resourceName, boolean clearIfManual)
deleteAlert(**Alert** alert)
deleteAlerts(**List**<**Alert**> alerts)
deleteAllAlerts() Deletes all alerts of this node and all child nodes recursively.
getAlerts()
getHierarchicNodeName()
getHistoryReference()
getNodeName()
getPastHistoryReference()
hasAlert(**Alert** alert)
hasHistoryType(int type)
hasJustHistoryType(int type)
isExcludedFromScope()

isIncludedInScope()
isParentOf(**String** nodeName)
removeCustomIcon(**String** resourceName)
setCustomIcons(**ArrayList**<**String**> i, **ArrayList**<**Boolean**> c)
setExcludedFromScope(boolean isExcludedFromScope, boolean applyToChildNodes)
setHistoryReference(**HistoryReference** historyReference) Set current node reference.
setIncludedInScope(boolean isIncludedInScope, boolean applyToChildNodes)
setParent(**MutableTreeNode** newParent)
toString()
updateAlert(**Alert** alert)

Il·lustració 46: Mètodes de SiteNode

Ja podem per tant treballar amb les alertes per a crear un report que les inclogui.

2.4.5.1 Presentació de les alertes en format XML

Un cop ja tenim les dades de les alertes les volem convertir en un format que puguem fer servir amb JasperReports. De moment i de manera inicial el que intentarem és crear un sumari de alertes en XML que tingui un format similar al del report que proporciona ZAP en XML per a intentar ser consistents amb la aplicació.

Quan fem proves amb el debugador de Idea veiem que quan agafem el objecte *SiteNode* que representa la arrel del arbre dels llocs descobrim que els fills de l'arrel representen les URL que hem visitat. Per tant si realitzem una consulta com aquesta:

```
SiteNode root = (SiteNode) siteMap.getRoot();
int siteNumber = root.getChildCount();
for (int i = 0; i < siteNumber; i++) {
   SiteNode site = (SiteNode) root.getChildAt(i);
```

Podem obtenir per separat la llista de tots els llocs que hem visitat. Un cop tenim el *SiteNode* aconseguim la llista de alertes relacionades amb ell:

```
List<Alert> alertList = site.getAlerts();
```

Ja tenim la llista de alertes. Com volem aconseguir la llista en format XML cerquem una manera de convertir-les sense haver de fer-ho de manera excessivament manual, i veiem que dintre dels mètodes de la classe *Alert* trobem:

```
public String toPluginXML(String urls)
```

Aquest mètode ens pot servir per a convertir les alertes en format XML. Cerquem dintre del codi per a què es fa servir el paràmetre *urls* que hem de proveir per a realitzar la transformació:

```
public String toPluginXML(String urls) {
   StringBuilder sb = new StringBuilder(150); // ZAP: Changed
the type to StringBuilder.
   sb.append("<alertitem>\r\n");
   sb.append("
<pluginid>").append(pluginId).append("</pluginid>\r\n");
   sb.append("  <alert>").append(alert).append("</alert>\r\n");
   sb.append("
<riskcode>").append(risk).append("</riskcode>\r\n");
   sb.append("
<confidence>").append(confidence).append("</confidence>\r\n");
   sb.append("  <riskdesc>").append(replaceEntity(MSG_RISK[risk]
+ " (" + MSG_CONFIDENCE[confidence] +
")")).append("</riskdesc>\r\n");
       sb.append("
<desc>").append(paragraph(replaceEntity(description))).append("<
/desc>\r\n");

       sb.append(urls);

       sb.append("
<solution>").append(paragraph(replaceEntity(solution))).append("
</solution>\r\n");
       // ZAP: Added otherInfo to the report
       if (otherInfo != null && otherInfo.length() > 0) {
           sb.append("
<otherinfo>").append(paragraph(replaceEntity(otherInfo))).append
("</otherinfo>\r\n");
       }
   sb.append("  <reference>"
).append(paragraph(replaceEntity(reference))).append("</referenc
e>\r\n");
   if (cweId > 0) {
      sb.append("  <cweid>"
).append(cweId).append("</cweid>\r\n");
   }
   if (wascId > 0) {
      sb.append("  <wascid>"
).append(wascId).append("</wascid>\r\n");
   }

   sb.append("</alertitem>\r\n");
   return sb.toString();
}
```

Podem entendre que es tracta simplement de un paràmetre per a posar una sèrie de URLs o caràcters després de la descripció, però en qualsevol cas no necessitem omplir-lo per a obtenir la sortida en XML.

Fem un petit codi que ens proporcionarà doncs el valor XML de cada alerta:

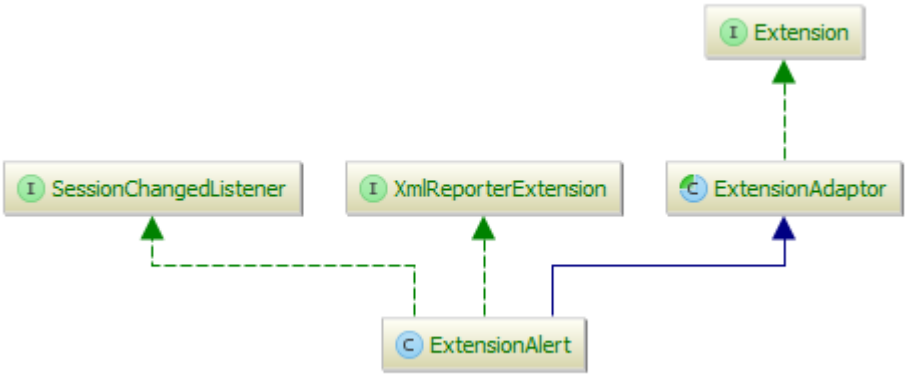
```
public StringBuilder getAlertsXML(SiteNode site){
   StringBuilder siteXML = new StringBuilder();
   List<Alert> alertList = site.getAlerts();
   for(Alert alert: alertList){
      siteXML.append(alert.toPluginXML(""));
   }
   return siteXML;
}
```

D'aquesta manera tenim el objectiu aconseguit, però no és la única manera que tenim de fer-ho. De fet, com hem comentat anteriorment, el que volem és que el contingut de la sortida sigui el més similar possible a la sortida que trobem amb la extensió XML.

Busquem doncs com es realitza la sortida en XML habitualment i trobem que existeix una interfície en forma de extensió que al ser implementada ens dona un fragment XML ben format a partir de un site:

```
package org.zaproxy.zap.extension;

import org.parosproxy.paros.model.SiteNode;

/**
 * This interface should be implemented by extensions that wish
to write data to
 * XML report. getXML() method should return well-formed XML
fragment (without the
 * <?xml> declaration) . This method is called by ReportLastScan
class
 * @author alla
 */
public interface XmlReporterExtension {

    String getXml(SiteNode site);

}
```

Aquesta és la interfície que està sent implementada per la *ExtensionAlert* que ens permet tenir la sortida en XML.



Il·lustració 47: Diagrama de ExtensionAlert

Cerquem exemples sobre com instanciar les extensions principals des de la nostre. També preferirem aquest mètode donat que desacobla la nostre extensió de la resta del codi. Diverses converses als fòrums revelen que els desenvolupadors principals de ZAP prefereixen que es facin servir les extensions per a poder accedir a les dades de ZAP, donat que dona facilitat a l'hora de fer canvis sense trencar res excessivament important.

```
public StringBuilder getExtensionsXML(SiteNode site) {
   StringBuilder extensionXml = new StringBuilder();
   ExtensionLoader loader =
Control.getSingleton().getExtensionLoader();
   int extensionCount = loader.getExtensionCount();
   for(int i=0; i<extensionCount; i++) {
      Extension extension = loader.getExtension(i);
      if(extension instanceof XmlReporterExtension) {

extensionXml.append(((XmlReporterExtension)extension).getXml(sit
e));
      }
   }
   return extensionXml;
}
```

La idea general és que, de manera similar a com ho hem fet amb altres capes, hem de instanciar la capa de *Control*, que ens permetrà carregar el carregador de extensions i anar-les iterant fins a trobar aquella que instancia la *XMLReporterExtension*, permetent-nos carregar-la i passar-li els *sites* que hem extret abans per a aconseguir la informació de les alertes en XML.

Un cop ja hem obtingut la informació de les alertes en XML, les tenim en aquest format:

```
<alertitem>
  <pluginid>10020</pluginid>
  <alert>X-Frame-Options Header Not Set</alert>
  <riskcode>2</riskcode>
  <confidence>2</confidence>
  <riskdesc>Medium (Medium)</riskdesc>
  <desc>X-Frame-Options header is not included in the HTTP response to
protect against &apos;ClickJacking&apos; attacks.</desc>
  <uri>http://192.168.222.129/codeexec/example2.php?order=age</uri>
  <param></param>
  <attack></attack>
  <otherinfo></otherinfo>
  <solution>Most modern Web browsers support the X-Frame-Options HTTP
header. Ensure it&apos;s set on all web pages returned by your site
(if you expect the page to be framed only by pages on your server
(e.g. it&apos;s part of a FRAMESET) then you&apos;ll want to use
SAMEORIGIN, otherwise if you never expect the page to be framed, you
should use DENY.  ALLOW-FROM allows specific websites to frame the web
page in supported web browsers).</solution>

<reference>http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/comb
ating-clickjacking-with-x-frame-options.aspx</reference>
</alertitem>
<alertitem>
  <pluginid>10020</pluginid>
  <alert>X-Frame-Options Header Not Set</alert>
  <riskcode>2</riskcode>
  <confidence>2</confidence>
  <riskdesc>Medium (Medium)</riskdesc>
  <desc>X-Frame-Options header is not included in the HTTP response to
protect against &apos;ClickJacking&apos; attacks.</desc>

<uri>http://192.168.222.129/commandexec/example3.php?ip=127.0.0.1</uri
>
  <param></param>
  <attack></attack>
  <otherinfo></otherinfo>
  <solution>Most modern Web browsers support the X-Frame-Options HTTP
header. Ensure it&apos;s set on all web pages returned by your site
(if you expect the page to be framed only by pages on your server
(e.g. it&apos;s part of a FRAMESET) then you&apos;ll want to use
SAMEORIGIN, otherwise if you never expect the page to be framed, you
should use DENY.  ALLOW-FROM allows specific websites to frame the web
page in supported web browsers).</solution>

<reference>http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/comb
ating-clickjacking-with-x-frame-options.aspx</reference>
</alertitem>
```

No estem discriminant entre la informació dels diferents llocs webs que s'han inicialitzat, ni s'han posat les capçaleres que s'ajunten amb la sortida estàndard de XML de ZAP, pel que ens cal afegir aquesta informació:

```
private String getAlertsXML(){
   StringBuilder sb = new StringBuilder();
   sb.append("<?xml version=\"1.0\"?>");
   sb.append("<OWASPZAPReport
version=\"").append(Constant.PROGRAM_VERSION).append("\"
generated=\"").append(ReportGenerator.getCurrentDateTimeString()
```

```
).append("\">\r\n");
   sb.append(siteXML());
   sb.append("</OWASPZAPReport>");
   return sb.toString().replaceAll("<p>|</p>","");

}

private StringBuilder siteXML() {
   StringBuilder report = new StringBuilder();
   SiteMap siteMap =
Model.getSingleton().getSession().getSiteTree();
   SiteNode root = (SiteNode) siteMap.getRoot();
   int siteNumber = root.getChildCount();
   for (int i = 0; i < siteNumber; i++) {
      SiteNode site = (SiteNode) root.getChildAt(i);
      setStats(site);
      String siteName = ScanPanel.cleanSiteName(site, true);
      String[] hostAndPort = siteName.split(":");
      boolean isSSL = (site.getNodeName().startsWith("https"));
      String siteStart = "<site name=\"" +
XMLStringUtil.escapeControlChrs(site.getNodeName()) + "\"" +
            " host=\"" +
XMLStringUtil.escapeControlChrs(hostAndPort[0])+ "\""+
            " port=\"" +
XMLStringUtil.escapeControlChrs(hostAndPort[1])+ "\""+
            " ssl=\"" + String.valueOf(isSSL) + "\"" +
            ">";
      StringBuilder extensionsXML = getExtensionsXML(site);
      String siteEnd = "</site>";
      report.append(siteStart);
      report.append(stats);
      report.append(extensionsXML);
      report.append(siteEnd);
   }
   return report;
}
```

Afegint aquesta informació ja estem simulant la sortida XML de ZAP, pel que podem passar a treballar amb un disseny amb el que treballar amb JasperReports.

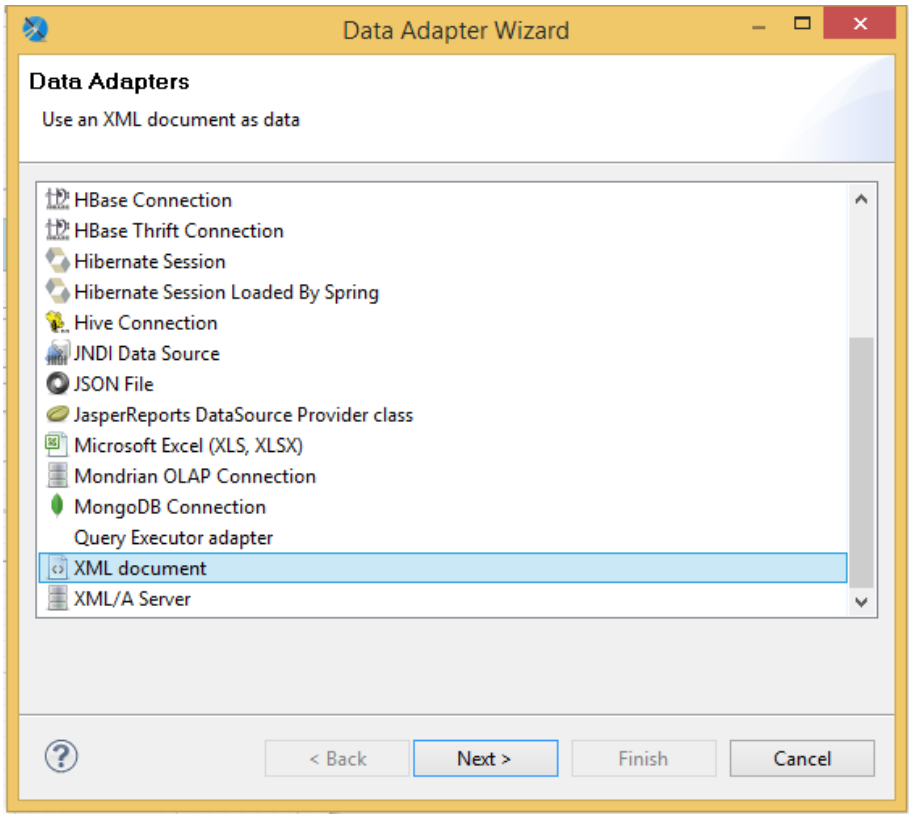
2.4.6 Disseny d'un report d'alertes d'exemple

Tenint ja un XML per a fer servir com a *datasource* d'entrada per a poder treballar amb JasperReports ens trobem amb que podem començar a dissenyar el report que tindrem com a resultat.

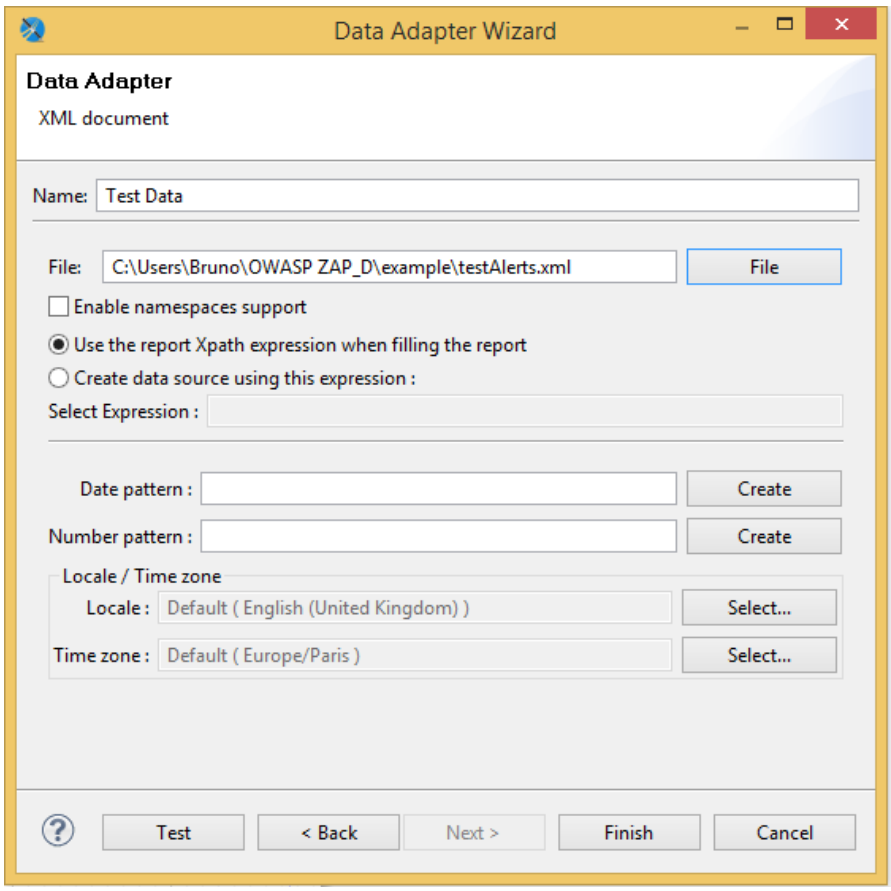
Per a fer això treballarem amb el Tibco Jaspersoft Studio tal i com hem vist abans al comentar les diferents possibilitats de integració amb eines de reporting.

El primer que hem de realitzar és descobrir com poder fer servir el XML de reporting com a *datasource* per a poder extreure les dades que es faran servir generalment a l'hora de fer reports.

Veiem que podem afegir un adaptador de dades des de la interfície, on podem escollir crear les dades a partir de un fitxer XML:



Il·lustració 48: Configuració de un Data Adapter



Il·lustració 49: Configuració de un Data Adapter (II)

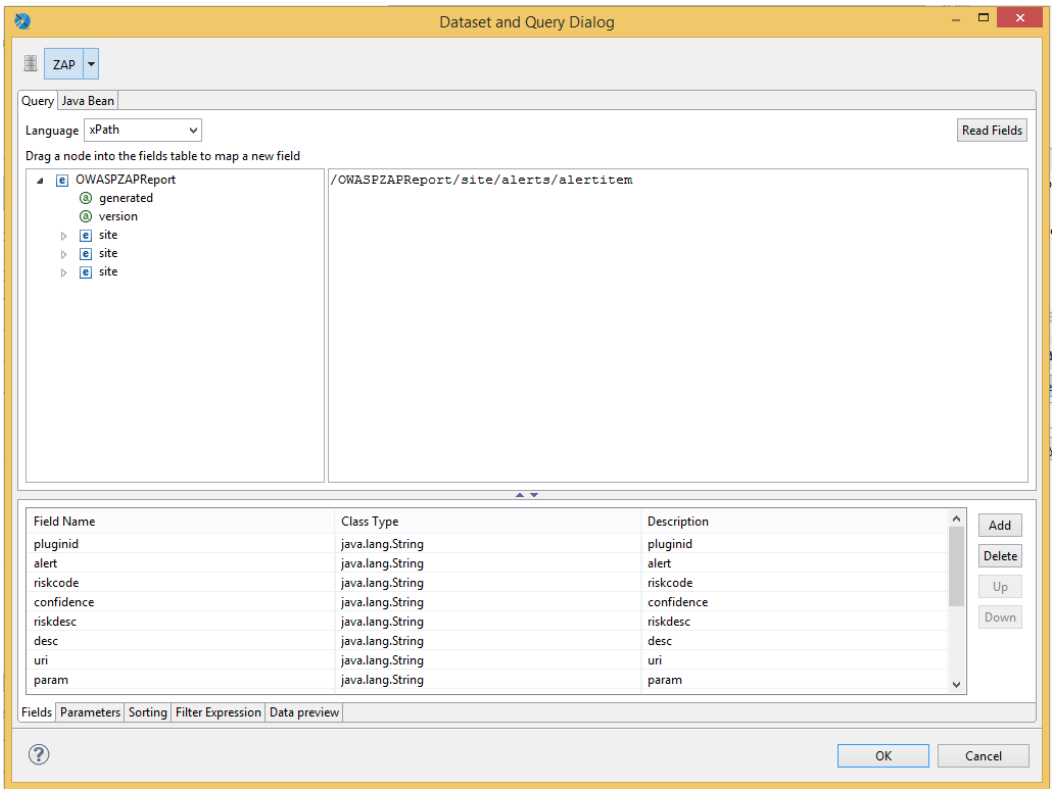
El adaptador de dades és el que defineix la informació sobre on es localitzen les dades, així com la lògica de la preparació dels paràmetres que es faran servir per les consultes posteriors al iterar sobre les dades. No es tracta, doncs, de les dades que es faran servir en si mateixes al report, sinó de definició sobre com es faran servir les dades.

Hem definit per tant el tipus de connexió com a una connexió XML i veiem que podrem fer servir expressions XPath per a omplir les dades del report un cop l'omplim. Gràcies a aquest adaptador de dades es configurarà també la connexió i es definiran els paràmetres addicionals que calguin per el report.

Les connexions son obertes i passades directament a JasperReports en la creació del report en si mateix.

Un cop ja tenim el adaptador de dades configurat necessitem definir els camps amb els quals treballarem a l'hora de fer el report en si mateix. Podem definir per tant una ruta XPath que ens ajudarà a dir quins elements del XML que estem fent servir ens interessa fer servir com a paràmetres.

En el nostre cas, per fer un report d'exemple, el que volem és la informació sobre les alertes en si mateixes. Abans hem vist la estructura de les alertes individualment, en les que el node de cada alerta individual estava definit per el indicador <*aleritem*>. El que voldrem serà doncs definir la ruta XPath fins a aquests elements i així aconseguir els elements que hi siguin dintre:



Il·lustració 50: Configuració de un Dataset

Podem llegir els camps de */OWASPZAPReport/site/alerts/alertitem* afegint-los de manera manual amb el botó "*add*" o fent servir el botó *"Read Fields"* que ens permet llegir tots els camps que es troben a la ruta que indiquem.

Podem veure doncs tots els camps amb els que treballarem:

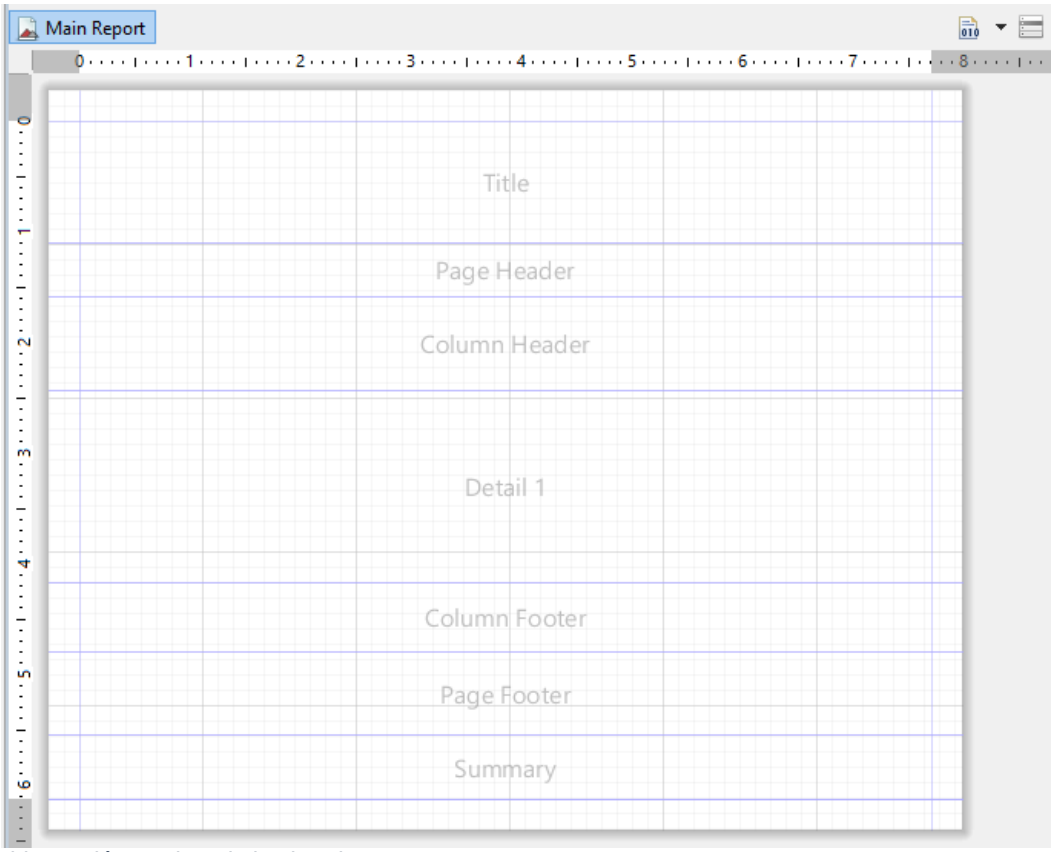
Field Name	Class Type	Description
pluginid	java.lang.String	pluginid
alert	java.lang.String	alert
riskcode	java.lang.String	riskcode
confidence	java.lang.String	confidence
riskdesc	java.lang.String	riskdesc
desc	java.lang.String	desc
uri	java.lang.String	uri
param	java.lang.String	param
attack	java.lang.String	attack
evidence	java.lang.String	evidence
otherinfo	java.lang.String	otherinfo
solution	java.lang.String	solution
reference	java.lang.String	reference
cweid	java.lang.String	cweid
wascid_1	java.lang.String	wascid

Il·lustració 51: Configuració dels camps

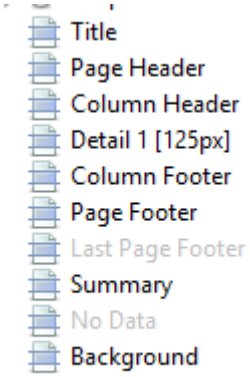
Un cop s'han obtingut els camps podem seleccionar els que ens interessen i col·locar-los segons els nostres interessos. De moment en podem fer un molt simple de exemple, on només col·loquem les dades de manera endreçada.

Malauradament no hem trobat documentació de qualitat gratuïta de JasperReports, donat que el seu model de negoci passa per oferir paquets amb el que anomenen “documentació avançada” de pagament. Podem per això treballar amb la introducció de la documentació que se’ns dona a la seva plana web, així com en base als exemples que trobem i també fer-ho en base a la prova i error.

El primer que veiem al editor visual dels reports és una sèrie de bandes que ens ajudaran, com podrem veure, a endreçar la informació de manera eficient:



Il·lustració 52: Vista de les bandes



Il·lustració 53: Vista de les bandes (II)

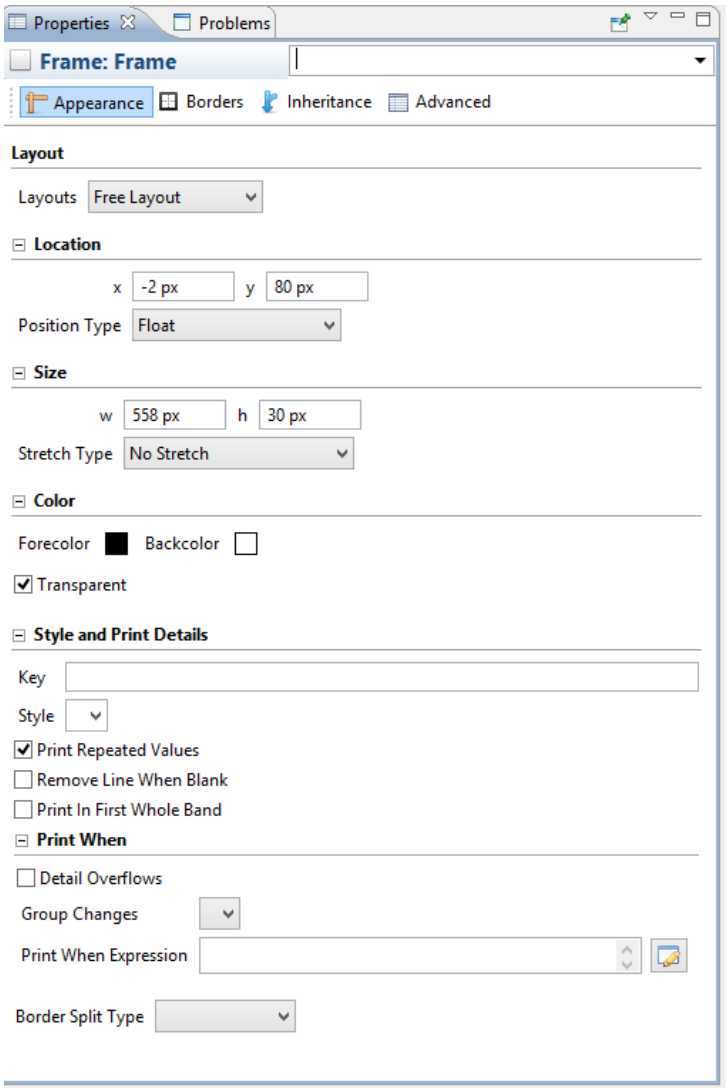
Basant-nos en la informació que tenim al lloc web de JasperSoft tenim aquestes descripcions de les bandes:

Nom	Descripció
Title	Primera banda visible, creada un sol cop i pot ser impresa a una pàgina diferent.
Page Header	Permet definir una capçalera per a les pàgines. Apareix a totes les pàgines a la posició definida a la fase de disseny.
Column Header	Banda impresa al començament de cada columna de detalls.
Group Header	Els reports poden contenir uns elements anomenats “group bands” que permet la agrupació de detalls en grups. Una capçalera de grups sempre té com a acompanyant un peu de grups, i determinen el comportament a nivell gràfic, sent possible donar-li diferents propietats que afectin a com es renderitza el report.
Group Footer	Element de compleció de grup juntament amb el “Group Header”.
Column Footer	Banda impresa al final de cada columna de detalls.
Page Footer	Banda que apareix a cada pàgina que té un “Page Header”, al final de la pàgina.
Last Page Footer	Banda que permet definir el darrer “Page Footer” de manera diferent als altres, si ens és necessari.
Summary	Banda que permet inserir detalls com a càlculs finals totals i altre informació que pot ser interessant tenir al final.
Background	Permet inserir marques d’aigua i altres gràfics de fons.

Il·lustració 54: Descripció de les bandes

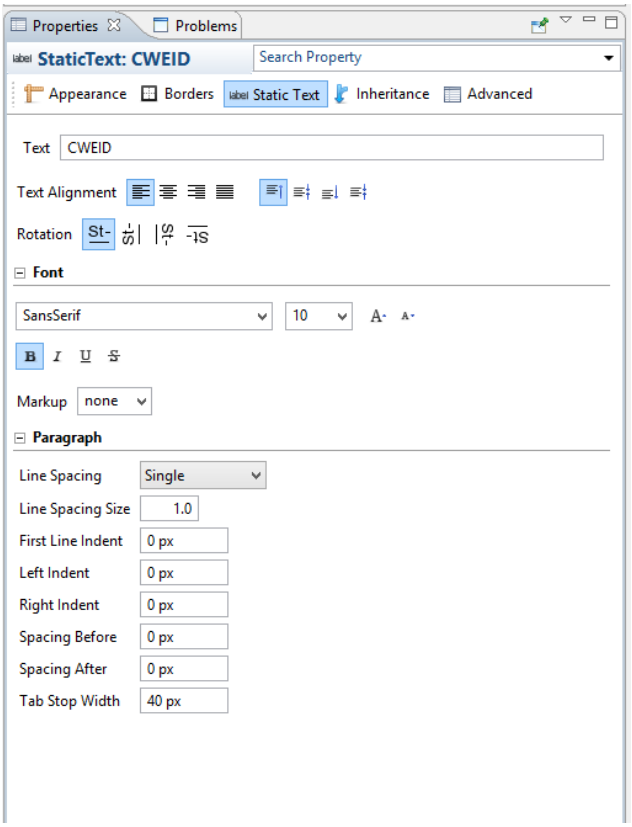
Per a poder accedir a les variables des de la capa de disseny ens podem referir a elles com a *$F(nomdevariable)*. Provarem doncs a afegir la informació a les bandes. A la descripció que hem trobat inicialment no hem trobat informació sobre la banda de detalls o "*Detail*", però aquesta és la que farem servir donat que iterarà sobre els elements que existeixin per els camps que li proporcionem.

No n'hi ha prou amb situar els caps a la banda, sinó que s'han de agrupar coherentment. Quan volem que dos camps estiguin alineats de manera horitzontal farem servir un element conegut com a *"Frame"* que ens permetrà col·locar-los sempre alineats. No només això, sinó que quan posicionem els elements veiem que si un element te una quantitat elevada de text el seu resultat sobreescriu els elements posteriors. Per a evitar això cal que fem que els *Frames* que fem servir tinguin una posició flotant en el report. Aquesta és una propietat que podem trobar en "Position Type".
Aquí podem veure les propietats de un *Frame:*



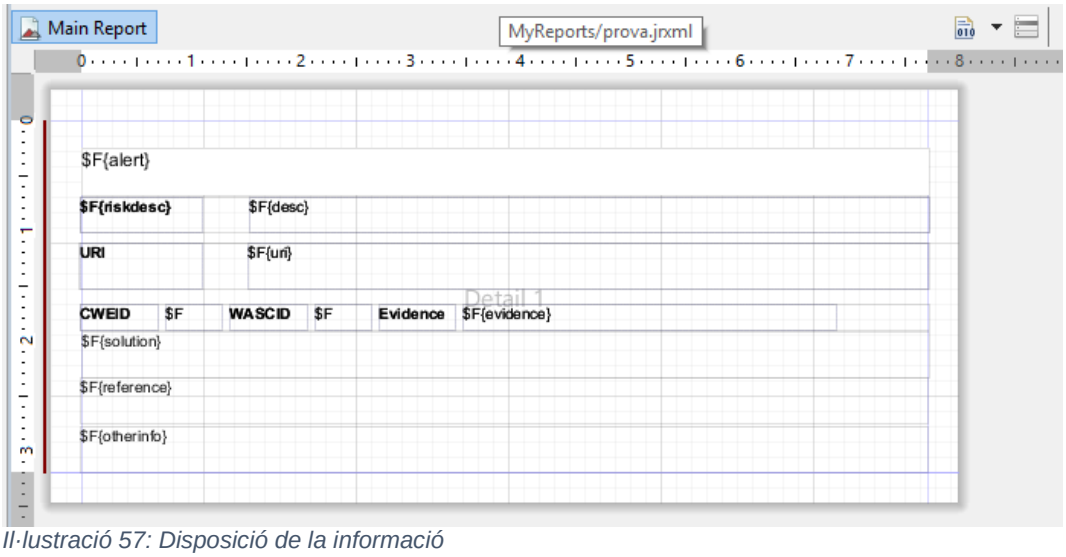
Il·lustració 55: Propietats de un Frame

Quan vulguem editar el format de un text haurem de editar les seves propietats en un panell similar. Per exemple, per fer aquest primer report podem posar text per defecte en negreta per a emfatitzar noms d'alguns dels camps:



Il·lustració 56: Configuració del text estàtic

Canviem doncs les propietats que veiem necessàries i endrecen la informació. Aquesta és la vista des de el editor visual:



Il·lustració 57: Disposició de la informació

I ensenyem una part en forma d'exemple del codi jrxml:

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Created with Jaspersoft Studio version 6.0.4.final using
JasperReports Library version 6.0.4  -->
<!-- 2015-06-01T20:11:37 -->
<jasperReport
xmlns="http://jasperreports.sourceforge.net/jasperreports"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://jasperreports.sourceforge.net/jasperr
eports
http://jasperreports.sourceforge.net/xsd/jasperreport.xsd"
name="prova" pageWidth="595" pageHeight="842" columnWidth="555"
leftMargin="20" rightMargin="20" topMargin="20"
bottomMargin="20" isTitleNewPage="true" uuid="7e7a7130-9529-
447d-afeb-8ac17ca19881">
<property name="com.jaspersoft.studio.data.defaultdataadapter"
value="ZAP"/>
<queryString language="xPath">
      <![CDATA[/OWASPZAPReport/site/alerts/alertitem]]>
</queryString>
<field name="pluginid" class="java.lang.String">
      <fieldDescription><![CDATA[pluginid]]></fieldDescription>
</field>
<field name="alert" class="java.lang.String">
      <fieldDescription><![CDATA[alert]]></fieldDescription>
</field>
<field name="riskcode" class="java.lang.String">
      <fieldDescription><![CDATA[riskcode]]></fieldDescription>
</field>
<field name="confidence" class="java.lang.String">
      <fieldDescription><![CDATA[confidence]]></fieldDescription>
</field>

<background>
      <band splitType="Stretch"/>
</background>
<detail>
      <band height="230" splitType="Stretch">
            <frame>
                  <reportElement positionType="Float" x="-1"
y="50" width="557" height="23" uuid="4240131b-1b61-45f7-a9c0-
aa09814aba1b"/>
                  <textField isStretchWithOverflow="true">
                        <reportElement key=""
positionType="Float" x="0" y="0" width="80" height="23"
uuid="af1b17b4-32fe-454a-88db-4d91cd3c019c"/>
                        <textElement>
                              <font isBold="true"/>
                        </textElement>

<textFieldExpression><![CDATA[$F{riskdesc}]]></textFieldExpressi
on>
                  </textField>
                  <textField isStretchWithOverflow="true">
                        <reportElement key=""
positionType="Float" x="111" y="0" width="446" height="23"
uuid="0f3c9d65-b826-4295-818b-bca6dc3acdfe"/>

<textFieldExpression><![CDATA[$F{desc}]]></textFieldExpression>
                  </textField>
```

```
            </frame>
            <textField isStretchWithOverflow="true">
                  <reportElement positionType="Float" x="0"
y="138" width="556" height="30" uuid="a514057e-6e6c-405b-abf0-
7a506470309e"/>

...

<textFieldExpression><![CDATA[$F{solution}]]></textFieldExpressi
on>
            </textField>
            <frame>
                  <reportElement positionType="Float" x="-2"
y="80" width="558" height="30" uuid="c5d2d918-8c49-43b0-abb7-
ffcbfa91811f"/>
                  <textField isStretchWithOverflow="true">
                        <reportElement key=""
positionType="Float" x="111" y="0" width="446" height="30"
uuid="f59f3790-960d-432e-ac78-0768abd519a6"/>

<textFieldExpression><![CDATA[$F{uri}]]></textFieldExpression>
                  </textField>
                  <staticText>
                        <reportElement key=""
positionType="Float" x="1" y="0" width="80" height="30"
uuid="cc7c1517-779c-4e41-b92a-2930cb0c13a6"/>
                        <textElement>
                              <font isBold="true"/>
                        </textElement>
                        <text><![CDATA[URI]]></text>
                  </staticText>
            </frame>
            <textField isStretchWithOverflow="true">
                  <reportElement positionType="Float" x="-1"
y="168" width="556" height="30" uuid="af7aa625-0661-49dd-9bb5-
08243e799b7f"/>

<textFieldExpression><![CDATA[$F{reference}]]></textFieldExpress
ion>
            </textField>
            <textField>
                  <reportElement positionType="Float"
mode="Opaque" x="0" y="18" width="556" height="31"
backcolor="#FFFFFF" uuid="e9f01743-55e8-43f9-821e-
71438d123e99"/>
                  <textElement>
                        <font size="12"/>
                  </textElement>

</detail>
</jasperReport>
```

A vegades ens pot ajudar editar simplement el jrxml manualment si trobem que és més senzill que trobar les dades de manera manual.

Veiem doncs com queda el resultat de compilar i omplir les dades amb aquest report:

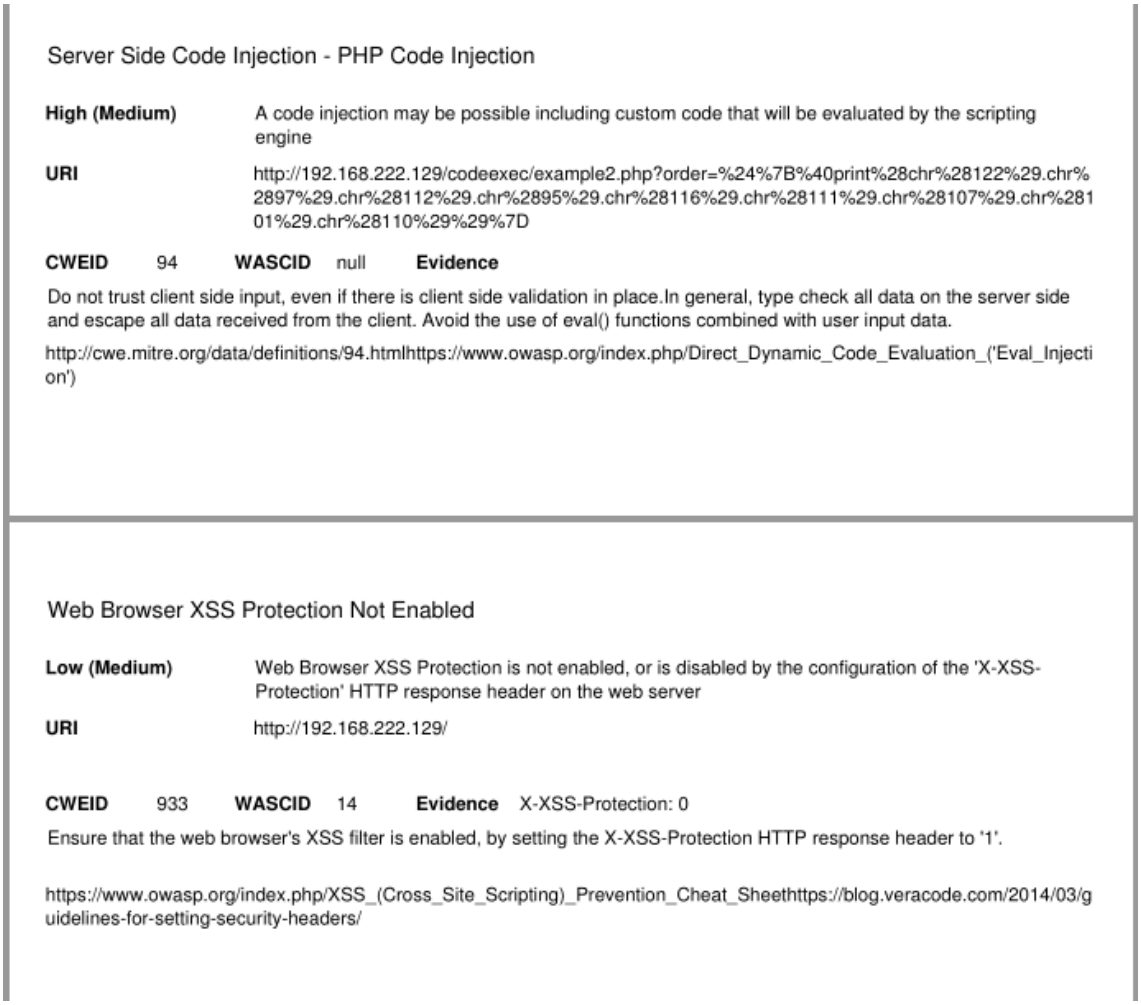
Cross Site Scripting (Reflected)

High (Medium) Cross-site Scripting (XSS) is an attack technique that involves echoing attacker-supplied code into a user's browser instance. A browser instance can be a standard web browser client, or a browser object embedded in a software product such as the browser within WinAmp, an RSS reader, or an email client. The code itself is usually written in HTML/JavaScript, but may also extend to VBScript, ActiveX, Java, Flash, or any other browser-supported technology.When an attacker gets a user's browser to execute his/her code, the code will run within the security context (or zone) of the hosting web site. With this level of privilege, the code has the ability to read, modify and transmit any sensitive data accessible by the browser. A Cross-site Scripted user could have his/her account hijacked (cookie theft), their browser redirected to another location, or possibly shown fraudulent content delivered by the web site they are visiting. Cross-site Scripting attacks essentially compromise the trust relationship between a user and the web site. Applications utilizing browser object instances which load content from the file system may execute code under the local machine zone allowing for system compromise.There are three types of Cross-site Scripting attacks: non-persistent, persistent and DOM-based.Non-persistent attacks and DOM-based attacks require a user to either visit a specially crafted link laced with malicious code, or visit a malicious web page containing a web form, which when posted to the vulnerable site, will mount the attack. Using a malicious form will oftentimes take place when the vulnerable resource only accepts HTTP POST requests. In such a case, the form can be submitted automatically, without the victim's knowledge (e.g. by using JavaScript). Upon clicking on the malicious link or submitting the malicious form, the XSS payload will get echoed back and will get interpreted by the user's browser and execute. Another technique to send almost arbitrary requests (GET and POST) is by using an embedded client, such as Adobe Flash.Persistent attacks occur when the malicious code is submitted to a web site where it's stored for a period of time. Examples of an attacker's favorite targets often include message board posts, web mail messages, and web chat software. The unsuspecting user is not required to interact with any additional site/link (e.g. an attacker site or a malicious link sent via email), just simply view the web page containing the code.

URI http://192.168.222.129/codeexec/example1.php?name=%3C%2Fdiv%3E%3Cscript%3Ealert%281%29%3B%3C%2Fscript%3E%3Cdiv%3E

CWEID 79 **WASCID** 8 **Evidence** <script>alert(1);</script>

Phase: Architecture and DesignUse a vetted library or framework that does not allow this weakness to occur or provides constructs that make this weakness easier to avoid.Examples of libraries and frameworks that make it easier to generate properly encoded output include Microsoft's Anti-XSS library, the OWASP ESAPI Encoding module, and Apache Wicket.Phases: Implementation; Architecture and DesignUnderstand the context in which your data will be used and the encoding that will be expected. This is especially important when transmitting data between different components, or when generating outputs that can contain multiple encodings at the same time, such as web pages or multi-part mail messages. Study all expected communication protocols and data representations to determine the required encoding strategies.For any data that will be output to another web page, especially any data that was received from external inputs, use the appropriate encoding on all non-alphanumeric characters.Consult the XSS Prevention Cheat Sheet for more details on the types of encoding and escaping that are needed.Phase: Architecture and DesignFor any security checks that are performed on the client side, ensure that these checks are duplicated on the server side, in order to avoid CWE-602. Attackers can bypass the client-side checks by modifying values after the checks have been performed, or by changing the client to remove the client-side checks entirely. Then, these modified values would be submitted to the server.If available, use structured mechanisms that automatically enforce the separation between data and code. These mechanisms may be able to provide the relevant quoting, encoding, and validation automatically, instead of relying on the developer to provide this capability at every point where output is generated.Phase: ImplementationFor every web page that is generated, use and specify a character encoding such as ISO-8859-1 or UTF-8. When an encoding is not specified, the web browser may choose a different encoding by guessing which encoding is actually being used by the web page. This can cause the web browser to treat certain sequences as special, opening up the client to subtle XSS attacks. See CWE-116 for more mitigations related to encoding/escaping.To help mitigate XSS attacks against the user's session cookie, set the session cookie to be HttpOnly. In browsers that support the HttpOnly feature (such as more recent versions of Internet Explorer and Firefox), this attribute can prevent the user's session cookie from being accessible to malicious client-side scripts that use document.cookie. This is not a complete solution, since HttpOnly is not supported by all browsers. More importantly, XMLHTTPRequest and other powerful browser technologies provide read access to HTTP headers, including the Set-Cookie header in which the HttpOnly flag is set.Assume all input is malicious. Use an "accept known good" input validation strategy, i.e., use a whitelist of acceptable inputs that strictly conform to specifications. Reject any input that does not strictly conform to specifications, or transform it into something that does. Do not rely exclusively on looking for malicious or malformed inputs (i.e., do not rely on a blacklist). However, blacklists can be useful for detecting potential attacks or determining which inputs are so malformed that they



Il·lustració 58: Report senzill

Tenim de moment un report senzill però veiem que la informació és aquella que volíem. El major problema que hem trobat fins ara és la manca de documentació clara a l'hora de crear aquests reports. Aquest senzill en si ha portat una quantitat molt important de hores i proves donat que és bastant complicat fer exactament el que volem, però podem veure el potencial que te aquesta eina.

2.4.7 Generació del report de Jasper des de l'extensió de ZAP

Un cop ja tenim el report inicial de Jasper realitzat i tenim codi per generar les alertes és necessari poder unir els dos components.

Per a poder fer-ho haurem de fer servir la API Java que proveeix JasperReports. Per sort es tracta de una operació senzilla donat que no hem de crear el report fent servir la API, sinó que només cal que fem servir el report i el omplim amb les dades de alertes que hem generat anteriorment, per poder generar doncs un format que ens interessi de sortida.

El primer que hem de fer després d'aconseguir les dades de les alertes és donar la opció de seleccionar un fitxer *.jasper,* un fitxer de disseny de

JasperReports compilat, que es pugui fer servir com a base de disseny. Si no es dona aquest fitxer es farà servir un fitxer que proporcionarem nosaltres amb la extensió, de manera que sempre hi hagi una manera de generar el report.

Veiem com realitzem la selecció del fitxer:

```
private File selectJasperFile(){
   //user chooses where the jasper file is, xml and reports will
be saved there
   File selectedFile = null;
   JFileChooser chooser = new
JFileChooser(Model.getSingleton().getOptionsParam().getUserDirec
tory());
   chooser.setFileFilter(new FileFilter() {

      @Override
      public boolean accept(File file) {
         if (file.isFile()
            &&
file.getName().toLowerCase().endsWith("."+Constant.messages.getS
tring(PREFIX + ".filter.ext"))) {
            return true;
      }
         return false;
      }

      @Override
      public String getDescription() {
         return Constant.messages.getString(PREFIX +
".fileSelect.msg");
      }
   });
   chooser.setCurrentDirectory(new
java.io.File(Constant.getZapHome() +
Constant.messages.getString(PREFIX + ".folder.name")+ "/"));

chooser.setDialogTitle(Constant.messages.getString(PREFIX +
".dialog.select"));
   FileNameExtensionFilter filter = new FileNameExtensionFilter(
         Constant.messages.getString(PREFIX + ".filter.name"),
Constant.messages.getString(PREFIX + ".filter.ext"));
   chooser.setFileFilter(filter);
   chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
   chooser.setAcceptAllFileFilterUsed(false);
   int rc =
chooser.showSaveDialog(View.getSingleton().getMainFrame());
   if(rc == JFileChooser.APPROVE_OPTION){
      selectedFile = chooser.getSelectedFile();
   }
   return selectedFile;
}
```

Només deixem que s'escullin fitxers de format *jasper*, per assegurar-nos que tenen el format que ens interessa.

Un cop ja tenim seleccionat el fitxer i per tant la carpeta on es desaran les dades treballem per omplir el disseny *jasper* amb el datasource que hem

generat abans. En aquesta ocasió fem una ullada primer al codi i després l'expliquem:

```
private void prepareReport() {
   JasperPrint file = null;
   String jasperFilePath= null;
   folderPath = null;
   try {
      String xml = getAlertsXML();
      File jasperFile = selectJasperFile();

      if(jasperFile==null) {
         folderPath =
Constant.getZapHome()+Constant.messages.getString(PREFIX +
".folder.name")+"/";
         jasperFilePath =
folderPath+Constant.messages.getString(PREFIX + ".file.name");
      }
      else{
         folderPath = jasperFile.getParent();
         jasperFilePath = jasperFile.getAbsolutePath();
      }
      String xmlDest =folderPath+
Constant.messages.getString(PREFIX + ".xmlfile");
      FileUtils.writeStringToFile(new File(xmlDest), xml);
Map<String, Object> params = new HashMap<>();
      Document document =
JRXmlUtils.parse(JRLoader.getLocationInputStream(xmlDest));

params.put(JRXPathQueryExecuterFactory.PARAMETER_XML_DATA_DOCUME
NT, document);
      params.put(JRXPathQueryExecuterFactory.XML_DATE_PATTERN,
"yyyy-MM-dd");
      params.put(JRXPathQueryExecuterFactory.XML_NUMBER_PATTERN,
"#,##0.##");
      params.put(JRXPathQueryExecuterFactory.XML_LOCALE,
Locale.ENGLISH);
      params.put(JRParameter.REPORT_LOCALE, Locale.US);

      file = JasperFillManager.fillReport(jasperFilePath,
params);
      String resultFile = folderPath+file.getName()+".pdf";
      JasperExportManager.exportReportToPdfFile(file,
resultFile);
   } catch (JRException e) {
      e.printStackTrace();
   } catch (IOException e) {
      e.printStackTrace();
   }
}
```

Veiem que, com diem, el primer que fem és generar les alertes en XML i després seleccionar el fitxer *jasper* corresponent. Un cop ho fem podem ja escriure el xml a fitxer fent servir el mateix directori que aquell que pertany al fitxer *jasper*, i a partir d'aquí ja passem a treballar amb codi que fa servir les llibreries de jasper pròpiament dites.

Primer de tot fem servir la classe *JRXmlUtils* de jasper per a parsejar el fitxer que tenim de alertes en un document. Per a poder llegir el fitxer

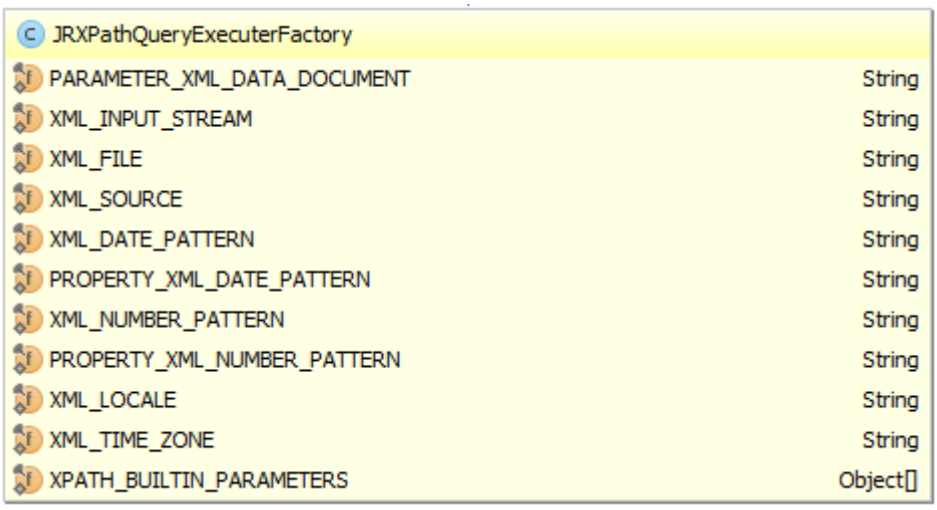
treballem amb un altre de les classes de JasperReports, *JRLoader*. En realitat podríem realitzar aquest pas sense fer servir les classes de Jasper, però creiem que ja que estem treballant amb ell és natural fer servir les seves classes:

```
JRXmlUtils.parse(JRLoader.getLocationInputStream(xmlDest));
```

Quan ja tenim el document parsejat en un format que serà adient per JasperReports ja podem passar a definir els paràmetres necessaris per a poder omplir el report amb les dades adients. Per a fer això haurem de crear un mapa de paràmetres (en aquest cas un *HashMap*) amb la llista de elements necessària per a la correcta composició. Es tractarà no només de el document XML, sinó també del format de dates, números i de la localització.

```
Map<String, Object> params = new HashMap<>();
params.put(JRXPathQueryExecuterFactory.PARAMETER_XML_DATA_DOCUMENT, document);
      params.put(JRXPathQueryExecuterFactory.XML_DATE_PATTERN, "yyyy-MM-dd");
      params.put(JRXPathQueryExecuterFactory.XML_NUMBER_PATTERN, "#,##0.##");
      params.put(JRXPathQueryExecuterFactory.XML_LOCALE, Locale.ENGLISH);
      params.put(JRParameter.REPORT_LOCALE, Locale.US);

      file = JasperFillManager.fillReport(jasperFilePath, params);
```

Podem observar que per poder definir el nom dels paràmetres ens ajudem de *JRXPathQueryExecuterFactory*, que tindrà totes les constants que necessitem com a clau del mapa de hash.



Il·lustració 59: Paràmetres de JRXPathQueryExecuterFactory

Un cop ja tenim els paràmetres preparats els podem enviar per què s'ompli de dades el report. D'aquest procés s'encarrega la classe *JasperFillManager*, que agafa reports compilats i els junta amb data

sources per a poder imprimir ja els documents en algun dels formats que vulguem.

Aquesta classe proposa una varietat de mètodes per omplir el report, donat que també podem enviar una connexió SQL, per exemple, a més d'un fitxer o input stream. També pot produir diferents resultats, un objecte, un fitxer o un output stream. No posem aquí tots els seu mètodes donat que son bàsicament permutacions del mateix.

Quan ja hem omplert el report ens queda exportar-ho a un format que ens convingui. En el nostre cas escollirem fer-ho en pdf, donat que és un format agradable de llegir i de imprimir que li falta a ZAP.

```
String resultFile = folderPath+file.getName()+".pdf";
      JasperExportManager.exportReportToPdfFile(file,
resultFile);
```

Podem realitzar una exportació senzilla fent servir el *JasperExportManager*, que ens proveeix la interfície per a exportar en els formats més senzills o populars com son HTML, XML i PDF. En el nostre cas només caldrà proveir el *JasperPrint* i la ruta de destinació per a obtenir el resultat que volem.

Method and Description
exportReportToHtmlFile(**JasperPrint** jasperPrint, java.lang.String destFileName)
exportReportToHtmlFile(java.lang.String sourceFileName)
exportReportToHtmlFile(java.lang.String sourceFileName, java.lang.String destFileName)
exportReportToPdf(**JasperPrint** jasperPrint)
exportReportToPdfFile(**JasperPrint** jasperPrint, java.lang.String destFileName)
exportReportToPdfFile(java.lang.String sourceFileName)
exportReportToPdfFile(java.lang.String sourceFileName, java.lang.String destFileName)
exportReportToPdfStream(java.io.InputStream inputStream, java.io.OutputStream outputStream)
exportReportToPdfStream(**JasperPrint** jasperPrint, java.io.OutputStream outputStream) Exports the generated report object received as first parameter into PDF format and writes the results to the output stream specified by the second parameter.
exportReportToXml(**JasperPrint** jasperPrint)

exportReportToXmlFile(**JasperPrint** jasperPrint, java.lang.String destFileName, boolean isEmbeddingImages)

exportReportToXmlFile(java.lang.String sourceFileName, boolean isEmbeddingImages)

exportReportToXmlFile(java.lang.String sourceFileName, java.lang.String destFileName, boolean isEmbeddingImages)

exportReportToXmlStream(java.io.InputStream inputStream, java.io.OutputStream outputStream)

exportReportToXmlStream(**JasperPrint** jasperPrint, java.io.OutputStream outputStream)

exportToHtmlFile(**JasperPrint** jasperPrint, java.lang.String destFileName)

Exports the generated report object received as parameter into HTML format, placing the result into the second file parameter.

exportToHtmlFile(java.lang.String sourceFileName)

Exports the generated report file specified by the parameter into HTML format.

exportToHtmlFile(java.lang.String sourceFileName, java.lang.String destFileName)

Exports the generated report file specified by the first parameter into HTML format, placing the result into the second file parameter.

exportToPdf(**JasperPrint** jasperPrint)

Exports the generated report object received as parameter into PDF format and returns the binary content as a byte array.

exportToPdfFile(**JasperPrint** jasperPrint, java.lang.String destFileName)

Exports the generated report file specified by the first parameter into PDF format, the result being placed in the second file parameter.

exportToPdfFile(java.lang.String sourceFileName)

Exports the generated report file specified by the parameter into PDF format.

exportToPdfFile(java.lang.String sourceFileName, java.lang.String destFileName)

Exports the generated report file specified by the first parameter into PDF format, the result being placed in the second file parameter.

exportToPdfStream(java.io.InputStream inputStream, java.io.OutputStream outputStream)

Exports the generated report read from the supplied input stream into PDF format and writes the results to the output stream specified by the second parameter.

exportToPdfStream(**JasperPrint** jasperPrint, java.io.OutputStream outputStream)

Exports the generated report object received as first parameter into PDF format and writes the results to the output stream specified by the second parameter.

exportToXml(**JasperPrint** jasperPrint)

Exports the generated report object supplied as parameter into XML format and returs the result as String.

exportToXmlFile(**JasperPrint** jasperPrint, java.lang.String destFileName, boolean isEmbeddingImages)

Exports the generated report object received as parameter into XML format, placing the result into the second file parameter.

exportToXmlFile(java.lang.String sourceFileName, boolean isEmbeddingImages)

Exports the generated report file specified by the parameter into XML format.

exportToXmlFile(java.lang.String sourceFileName, java.lang.String destFileName, boolean isEmbeddingImages)

Exports the generated report file specified by the first parameter into XML format, placing the result into the second file parameter.

exportToXmlStream(java.io.InputStream inputStream, java.io.OutputStream outputStream)

Exports the generated report object read from the supplied input stream into XML format, and writes the result to the output stream specified by the second parameter.

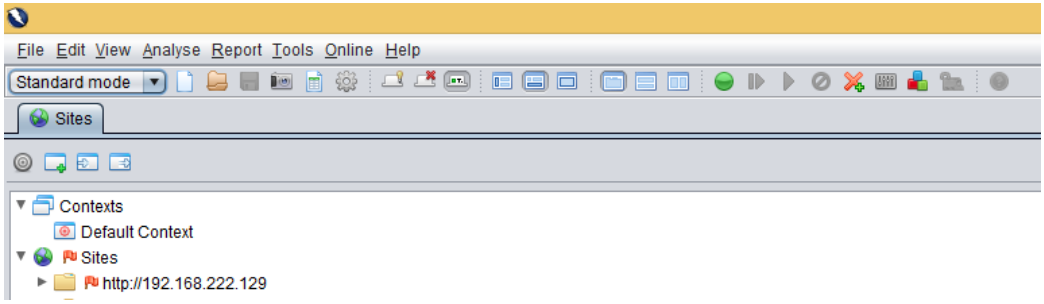
exportToXmlStream(**JasperPrint** jasperPrint, java.io.OutputStream outputStream)

Exports the generated report object supplied as the first parameter into XML format, and writes the result to the output stream specified by the second parameter.

getInstance(**JasperReportsContext** jasperReportsContext)

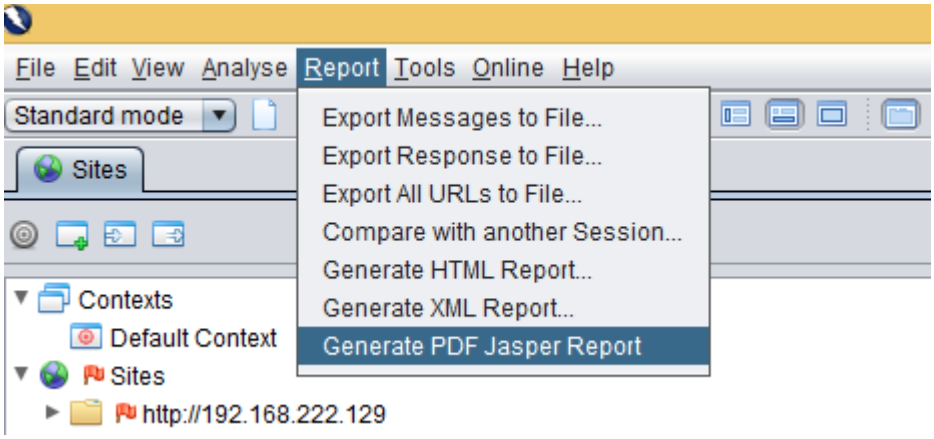
Il·lustració 60: Mètodes de JasperExportManager

Ho podem veure en acció, donada una sessió nova:



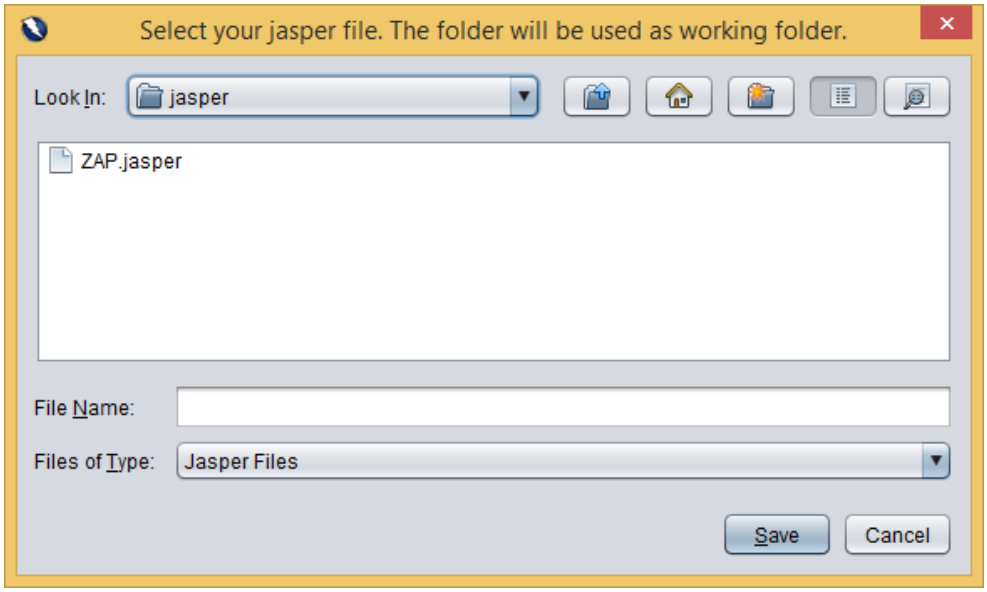
Il·lustració 61: Sessió de ZAP

Tenim un nou element al menú:



Il·lustració 62: Nou element al menú

Podem escollir el fitxer *.jasper* que hem generat abans:



Il·lustració 63: Selecció del fitxer Jasper

El pdf es genera sense problemes:

SQL Injection

High (Medium) SQL injection may be possible.

URI http://192.168.222.129/sqli/example6.php?id=4-2

CWEID 89 WASCID 19 Evidence

Do not trust client side input, even if there is client side validation in place. In general, type check all data on the server side.If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'If the application uses ASP, use ADO Command Objects with strong type checking and parameterized queries.If database Stored Procedures can be used, use them.Do *not* concatenate strings into queries in the stored procedure, or use 'exec', 'exec immediate', or equivalent functionality!Do not create dynamic SQL queries using simple string concatenation.Escape all data received from the client.Apply a 'whitelist' of allowed characters, or a 'blacklist' of disallowed characters in user input.Apply the privilege of least privilege by using the least privileged database user possible.In particular, avoid using the 'sa' or 'db-owner' database users. This does not eliminate SQL injection, but minimizes its impact.Grant the minimum database access that is necessary for the application.
https://www.owasp.org/index.php/Top_10_2010-A1https://www.owasp.org/index.php/SQL_Injection_Prevention_Cheat_Sheet

Server Side Code Injection - PHP Code Injection

High (Medium) A code injection may be possible including custom code that will be evaluated by the scripting engine

URI http://192.168.222.129/codeexec/example1.php?name=%22%3Bprint%28chr%28122%29.chr%2897%29.chr%28112%29.chr%2895%29.chr%28116%29.chr%28111%29.chr%28107%29.chr%28101%29.chr%28110%29%29%3B%24var%3D%22

CWEID 94 WASCID null Evidence

Do not trust client side input, even if there is client side validation in place.In general, type check all data on the server side and escape all data received from the client. Avoid the use of eval() functions combined with user input data.

http://cwe.mitre.org/data/definitions/94.htmlhttps://www.owasp.org/index.php/Direct_Dynamic_Code_Evaluation_('Eval_Injection')

Il·lustració 64: PDF generat amb menú

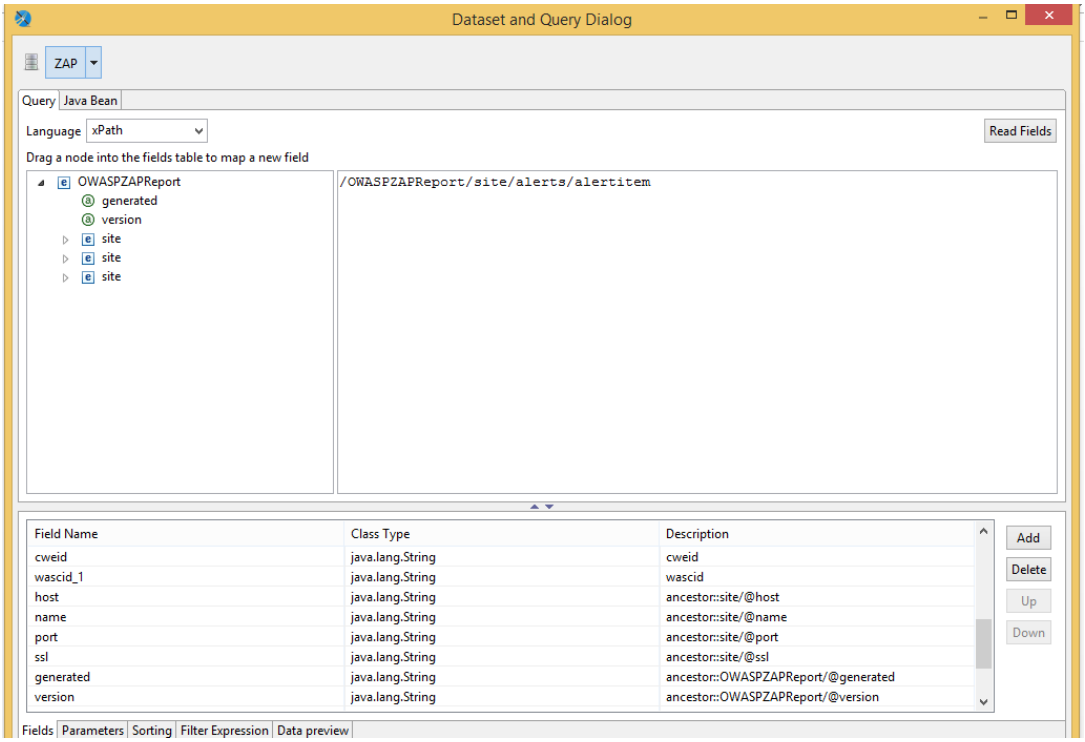
2.4.8 Creació d'un report avançat

Un cop ja tenim la creació del report aconseguida entenem que podem crear un report de exemple que sigui més atractiu de cara a fer-ho servir en situacions reals. No només ens servirà per a atraure més gent que pugui fer servir la nostre extensió, sinó que d'aquesta manera podrem descobrir possibles problemes que es poden trobar els usuaris a l'hora de fer servir Jasper amb el report que estem proveint.

Treballarem doncs amb funcions una mica més avançades de Jasper, com per exemple una portada per el report o alguns elements de resum. Seria interessant, per exemple, afegir gràfics i figures al resum, donant-li un aspecte més visual al report.

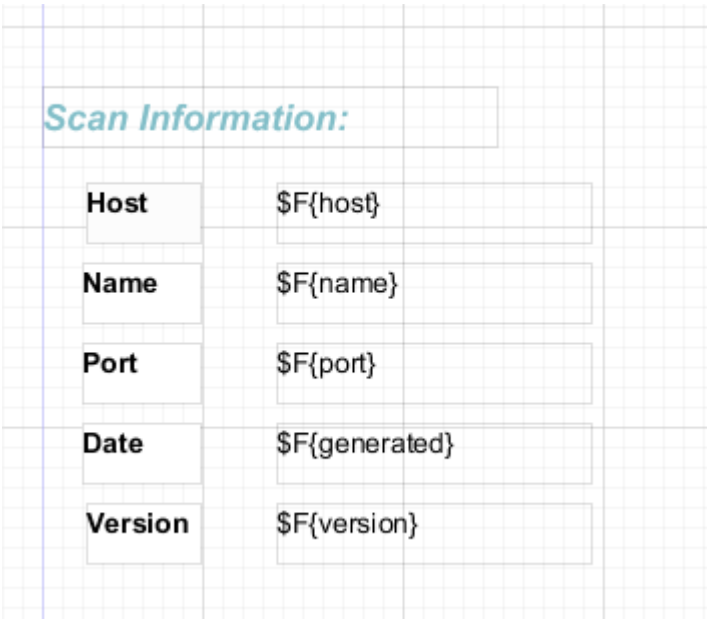
Per a la creació de la portada treballem amb una banda *Title*, que està pensada per a estar situada al inici del report com a presentació. Posar la informació bàsica del report és tan senzill com afegir-hi camps de text. Podem agafar, per exemple, les variables que defineixen el scan com a el nom del escaneig, el lloc on s'ha dirigit, el port, la data o la versió de ZAP, per donar una mica de context.

Afegir els camps és tan senzill com obtenir-los del *dataset* principal:



Il·lustració 65: Addició de camps amb el dataset principal

Un cop ja els tenim els podem col·locar de manera senzilla al report:



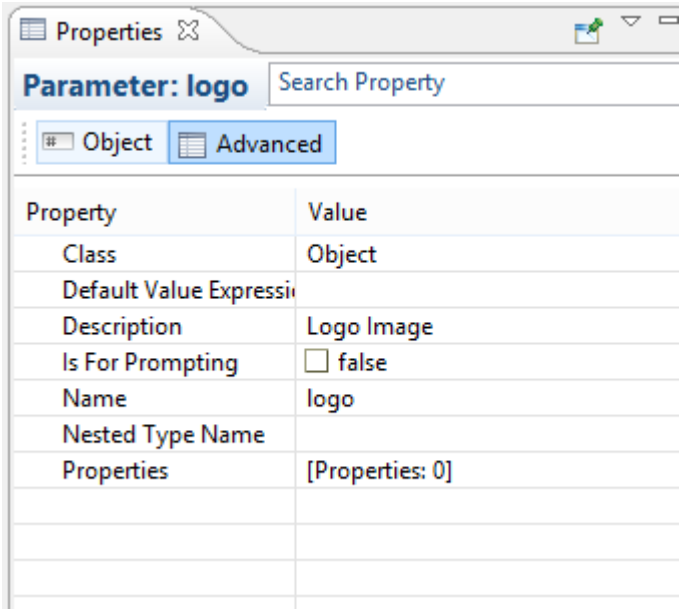
Il·lustració 66: Informació del escaneig al report

Col·loquem també un títol per al report:

Il·lustració 67: Títol del report

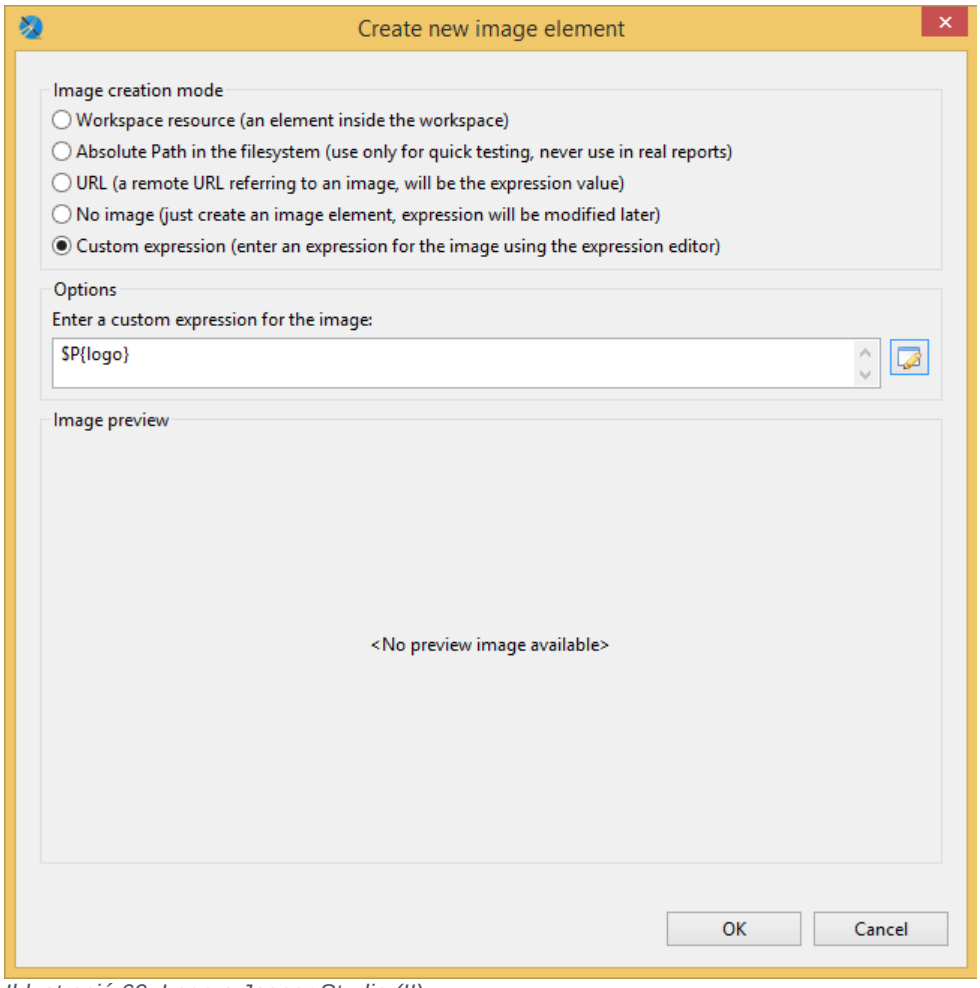
El problema vindrà quan vulguem col·locar un logotip, per exemple, donat que és molt possible que es faci servir en els reports que generin els usuaris i no és tant senzill. La imatge que passem per a logo no té sentit que sigui fixa, és a dir, que provingui d'una ruta absoluta, sinó que com és una imatge que serà usada per varios reports decidim que podria ser passada com a paràmetre al report.

Definim per tant un paràmetre amb nom "logo" pel qual passarem una imatge que puguem passar com a logotip en el report que volem fer servir. Ens hem de assegurar de que ho passem com a un objecte, no com una cadena de caràcters:



Il·lustració 68: Logo a Jasper Studio

Un cop el tenim definit creem un element de imatge amb aquest paràmetre com a valor de expressió:



Il·lustració 69: Logo a Jasper Studio (II)

Ja podem col·locar la imatge a la portada del report, aquesta serà avaluada quan ho creem si es passa un paràmetre amb el valor de “logo”:

Il·lustració 70: Logo a Jasper Studio (III)

Cal que passem el paràmetre de del codi Java amb els altres que li passem al omplir el report de dades:

```
String imagePath = folderPath+Constant.messages.getString(PREFIX
+ ".logofile");
BufferedImage image = ImageIO.read(new File(imagePath));
params.put("logo", image );
```

Com veiem simplement definim d'on llegim la imatge i la passem com a objecte. És important notar que hem de evitar passar el *InputStream*, i passar directament el contingut de la imatge. Per convenció estem afegint una imatge a la nostre carpeta de treball amb el nom "logo.png".

Il·lustració 71: Visualització de la portada

Quan ja tenim una introducció feta pel report comencem a treballar en un resum de les vulnerabilitats. Pensem que seria interessant obtenir, per exemple, gràfics o taules amb els continguts de resum, però ens trobem amb moltes limitacions donat el format de entrada. Aquestes limitacions venen per el fet de que necessitem poder fer una consulta a les dades de manera que el format que se'ns retorni sigui vàlid per l'entrada d'un gràfic o taula, és a dir, ens seria interessant, per exemple, poder comptar el

nombre de riscos d'un tipus o de certa severitat, i esperaríem obtenir un format així:

Risc	Nombre de ocurrències.
Alt	15
Mitjà	8
Baix	3
Informacional	32

Il·lustració 72: Taula de ocurrències esperada

No podem fer-ho, però, donat que quan realitzem una consulta amb el programari JasperSoft Studio, veiem que aquest només permet fer-ho amb un subconjunt limitat de XPath 1.0. Això representa un problema important a l'hora de realitzar reports amb un caire més interessant de manera senzilla, que és precisament el objectiu del projecte. Aquest problema no el tindríem si estiguéssim treballant amb un *DataSource* SQL, donat que podríem fer consultes de manera molt més fàcil que ens donessin uns resultats més elaborats.

Això no vol dir que no puguem realitzar les consultes de cap manera, si que podem, només que no de manera senzilla, pel que la nostre idea és facilitar la feina oferint certes dades extres amb el report XML que produïm quan estem adquirint les dades de les alertes.

2.4.8.1 Obtenció de dades estadístiques.

Si observem altres reports més elaborats veiem que les dades que probablement es necessitin més sovint seran la addició de riscos per tipus i per severitat. Per a aconseguir això realitzem una petita addició al XML que estem donant com a resultat.

Primer hem de definir estructures de dades que ens permetin crear fàcilment un XML a partir de una llista de elements. Creem les classes *RiskStats* i *AlertStats* que ens permetran desar les dades de manera endreçada i còmode.

Estadístiques de severitat:

```java
package org.zaproxy.zap.extension.jasperReports;

import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

/**
 * Created by Bruno on 10/05/2015.
 */
@XmlRootElement(name = "riskType")
@XmlType(propOrder = {"severity", "count"})
public class RiskStats {
    private int count;
    private String severity;

    public RiskStats(){
        this.severity="";
        this.count=0;
    }
    public RiskStats(String severity, int count) {
        this.severity = severity;
        this.count = count;
    }

    public RiskStats(String severity) {
        this.severity = severity;
        this.count = 0;
    }

    public void setSeverity(String severity) {
        this.severity = severity;
    }

    public int getCount() {
        return count;
    }

    public void setCount(int count) {
        this.count = count;
    }

    public String getSeverity() {
        return severity;
    }

    public void incCount(){
        this.count++;
    }

}
```

Estadístiques de alertes:

```java
package org.zaproxy.zap.extension.jasperReports;

import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

/**
 * Created by Bruno on 10/05/2015.
 */
@XmlRootElement(name = "alertStats")
```

```
@XmlType(propOrder = {"alertType", "occurences"})
public class AlertStats {
    public AlertStats(String alertType, int occurences){
        this.alertType = alertType;
        this.occurences = occurences;
    }

    public AlertStats(String alertType){
        this.alertType = alertType;
        this.occurences = 0;
    }

    public AlertStats(){
        this.alertType = "";
        this.occurences = 0;
    }

    public String getAlertType() {
        return alertType;
    }

    public void setAlertType(String alertType) {
        this.alertType = alertType;
    }

    public int getOccurences() {
        return occurences;
    }

    public void setOccurences(int occurences) {
        this.occurences = occurences;
    }

    public void incOcurences(){
        this.occurences++;
    }

    String alertType = "";
    int occurences = 0;
}
```

Un cop ja tenim la estructura feta el que fem és, simplement, quan tenim el *Site*, el recorrem per a establir aquestes estadístiques. En el cas de la severitat del risc es tracta d'un procés molt senzill, donat que només hem de observar el codi de risc, i podem anar incrementant el comptador de cada tipus un cop ho trobem. Com es tracta de una sèrie limitada i coneguda de possibilitats, no ens és difícil fer-ho amb un switch:

```
switch (a.getRisk()) {
      case 3:
         riskStatHigh.incCount();
         break;
      case 2:
         riskStatMedium.incCount();
         break;
      case 1:
         riskStatLow.incCount();
         break;
      case 0:
         riskStatInfo.incCount();
```

```
            break;
    }
```

En el cas dels tipus de alertes no és tan senzill donat que tenim un nombre indeterminat d'aquestes. El que podem fer és endreçar primer les alertes per tipus i anar detectant quan hi ha un canvi, moment en el que haurem de afegir la alerta a la llista.

```
Collections.sort(alerts, (Alert a1, Alert a2) ->
a1.getAlert().compareTo(a2.getAlert()));
site.deleteAllAlerts();
AlertStats alertStat = new AlertStats();
alertStat.setAlertType(alerts.get(0).getAlert());
alertStat.setOccurences(0);
int count = 0;
for (Alert a : alerts) {
   site.addAlert(a);
   if (a.getAlert().equals(alertStat.getAlertType()))
alertStat.incOcurences();
   else {
      alertStatsList.add(alertStat);
      alertStat = new AlertStats(a.getAlert(), 1);
   }
   ...
   count++;
   if(count == alerts.size())alertStatsList.add(alertStat);
}
```

Hem de estar atents a quan es tracta de la ultima alerta per a poder afegir-la a la llista també en acabar.

El codi quedaria per tant així:

```
public void setStats(SiteNode site){
   riskList = new ArrayList<>();
   alertStatsList = new ArrayList<>();
   List<Alert> alerts = site.getAlerts();
   if(alerts != null && alerts.size()>0) {

      RiskStats riskStatHigh = new RiskStats("High", 0);
      RiskStats riskStatMedium = new RiskStats("Medium", 0);
      RiskStats riskStatLow = new RiskStats("Low", 0);
      RiskStats riskStatInfo = new RiskStats("Info", 0);

      Collections.sort(alerts, (Alert a1, Alert a2) ->
a1.getAlert().compareTo(a2.getAlert()));
      site.deleteAllAlerts();
      AlertStats alertStat = new AlertStats();
      alertStat.setAlertType(alerts.get(0).getAlert());
      alertStat.setOccurences(0);
      int count = 0;
      for (Alert a : alerts) {
         site.addAlert(a);
         if (a.getAlert().equals(alertStat.getAlertType()))
alertStat.incOcurences();
         else {
            alertStatsList.add(alertStat);
```

```
            alertStat = new AlertStats(a.getAlert(), 1);
         }
         switch (a.getRisk()) {
            case 3:
               riskStatHigh.incCount();
               break;
            case 2:
               riskStatMedium.incCount();
               break;
            case 1:
               riskStatLow.incCount();
               break;
            case 0:
               riskStatInfo.incCount();
               break;
         }
         count++;
         if(count ==
alerts.size())alertStatsList.add(alertStat);
      }
      riskList.add(riskStatHigh);
      riskList.add(riskStatMedium);
      riskList.add(riskStatLow);
      riskList.add(riskStatInfo);

   }
}
```

Un cop ja estan creades ja podem transformar-les en XML:

```
public String xmlAlertStats(){
   java.io.StringWriter sw = new StringWriter();
   try {
      //Marshal Alert Stats
      JAXBContext contextAlerts =
JAXBContext.newInstance(AlertStats.class);
      Marshaller mAlerts = contextAlerts.createMarshaller();
      mAlerts.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
      mAlerts.setProperty(Marshaller.JAXB_FRAGMENT, true);

      //Marshal Risk Stats
      JAXBContext contextRisks =
JAXBContext.newInstance(RiskStats.class);
      Marshaller mRisks = contextRisks.createMarshaller();
      mRisks.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
Boolean.TRUE);
      mRisks.setProperty(Marshaller.JAXB_FRAGMENT, true);

      sw.append("<stats>");
      sw.append(System.getProperty("line.separator"));
      for(AlertStats alertStats:alertStatsList) {
         mAlerts.marshal(alertStats, sw);
         sw.append(System.getProperty("line.separator"));
           }
      for(RiskStats riskStats:riskList){
         mRisks.marshal(riskStats,sw);
         sw.append(System.getProperty("line.separator"));
      }
      sw.append("</stats>");
      sw.append(System.getProperty("line.separator"));
```

```
    } catch (JAXBException e) {
       e.printStackTrace();
    }
    return sw.toString();
}
```

I el resultat és aquest:

```
<stats>
		<alertStats>
			<alertType>Absence of Anti-CSRF Tokens</alertType>
			<occurences>3</occurences>
		</alertStats>
		<alertStats>
			<alertType>Cross Site Scripting (Reflected)</alertType>
			<occurences>9</occurences>
		</alertStats>
		<alertStats>
			<alertType>Information Disclosure - Sensitive Informations in
URL</alertType>
			<occurences>3</occurences>
		</alertStats>
		<alertStats>
			<alertType>Path Traversal</alertType>
			<occurences>2</occurences>
		</alertStats>
		<alertStats>
			<alertType>Private IP Disclosure</alertType>
			<occurences>2</occurences>
		</alertStats>
		<alertStats>
			<alertType>Remote File Inclusion</alertType>
			<occurences>2</occurences>
		</alertStats>
		<alertStats>
			<alertType>Remote OS Command Injection</alertType>
			<occurences>2</occurences>
		</alertStats>
		<alertStats>
			<alertType>SQL Injection</alertType>
			<occurences>4</occurences>
		</alertStats>
		<alertStats>
			<alertType>Server Side Code Injection - PHP Code
Injection</alertType>
			<occurences>2</occurences>
		</alertStats>
		<alertStats>
			<alertType>Web Browser XSS Protection Not
Enabled</alertType>
			<occurences>47</occurences>
		</alertStats>
		<alertStats>
			<alertType>X-Content-Type-Options Header
Missing</alertType>
			<occurences>47</occurences>
		</alertStats>
		<alertStats>
			<alertType>X-Frame-Options Header Not Set</alertType>
			<occurences>47</occurences>
```

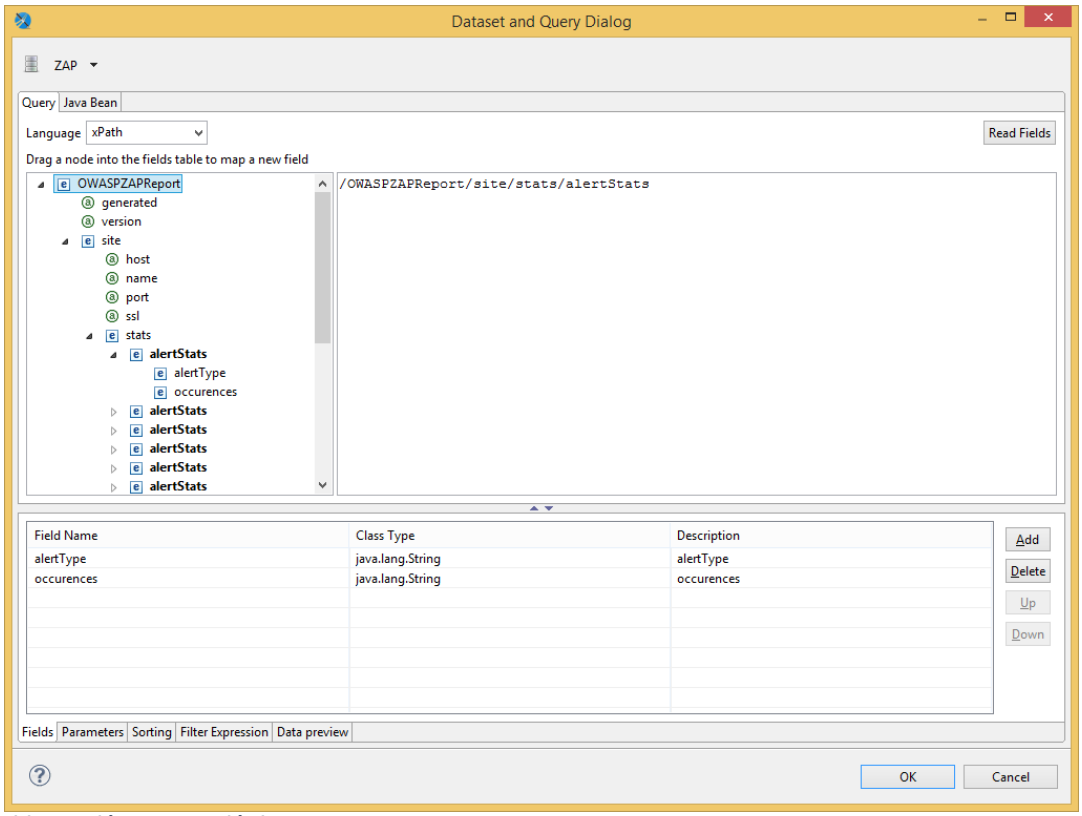
```
        </alertStats>
        <riskType>
                <severity>High</severity>
                <count>21</count>
        </riskType>
        <riskType>
                <severity>Medium</severity>
                <count>47</count>
        </riskType>
        <riskType>
                <severity>Low</severity>
                <count>99</count>
        </riskType>
        <riskType>
                <severity>Info</severity>
                <count>3</count>
        </riskType>
</stats>
```

2.4.8.2 Creació de un resum de dades al report

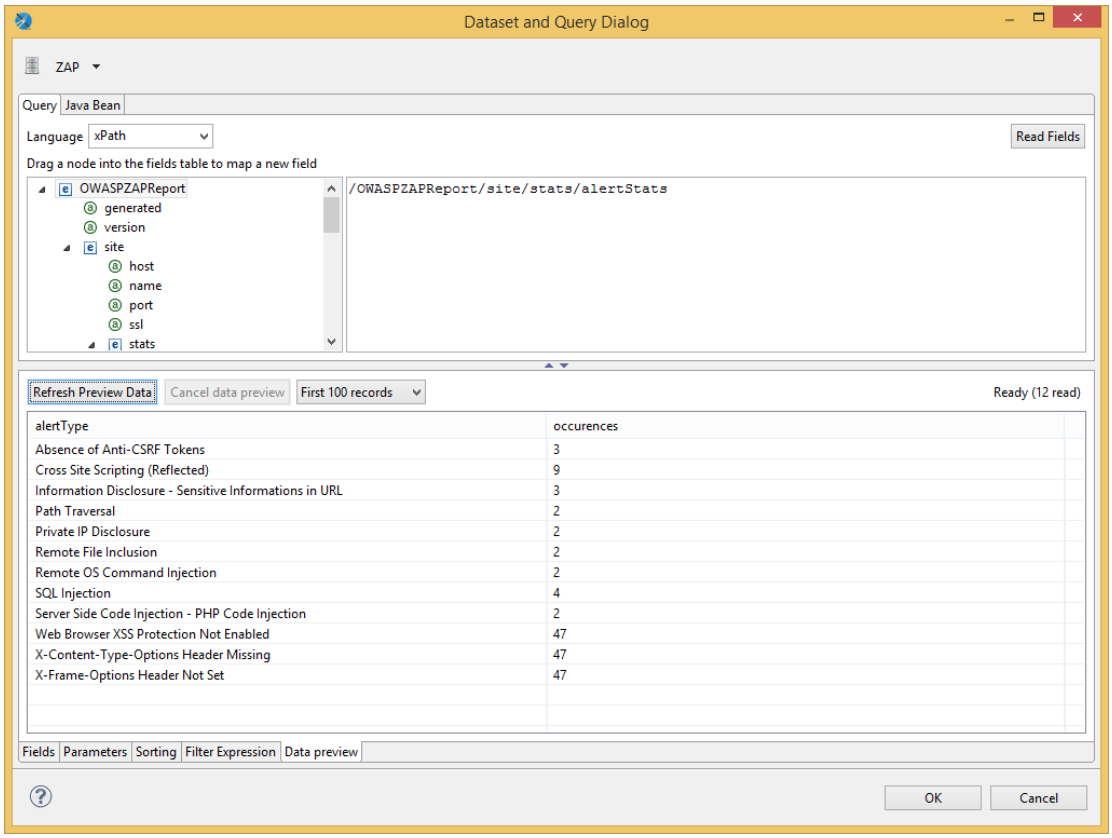
Un cop ja tenim les dades de estadístiques que volem al XML ja podem passar a crear figures més elaborades com gràfics o taules amb la informació que tenim de manera senzilla. Crearem per tant una sèrie de exemples per a que sigui molt fàcil basar-s'hi en ells per realitzar altres propostes més endavant i els entregarem amb la extensió.

Després de un temps elevat de proves descobrim que per a poder accedir de manera eficient a la informació que necessitem per les figures necessitem que aquesta estigui ja preparada per elles. La manera que tenim de fer-ho és definint series de dades a JasperReports que es fan servir en aquest tipus de estructures: *DataSets.*

Per a definir-ne un ho fem de manera similar a com definim els camps. Veiem a continuació la definició dels *datasets* de tipus i severitat dels riscos:

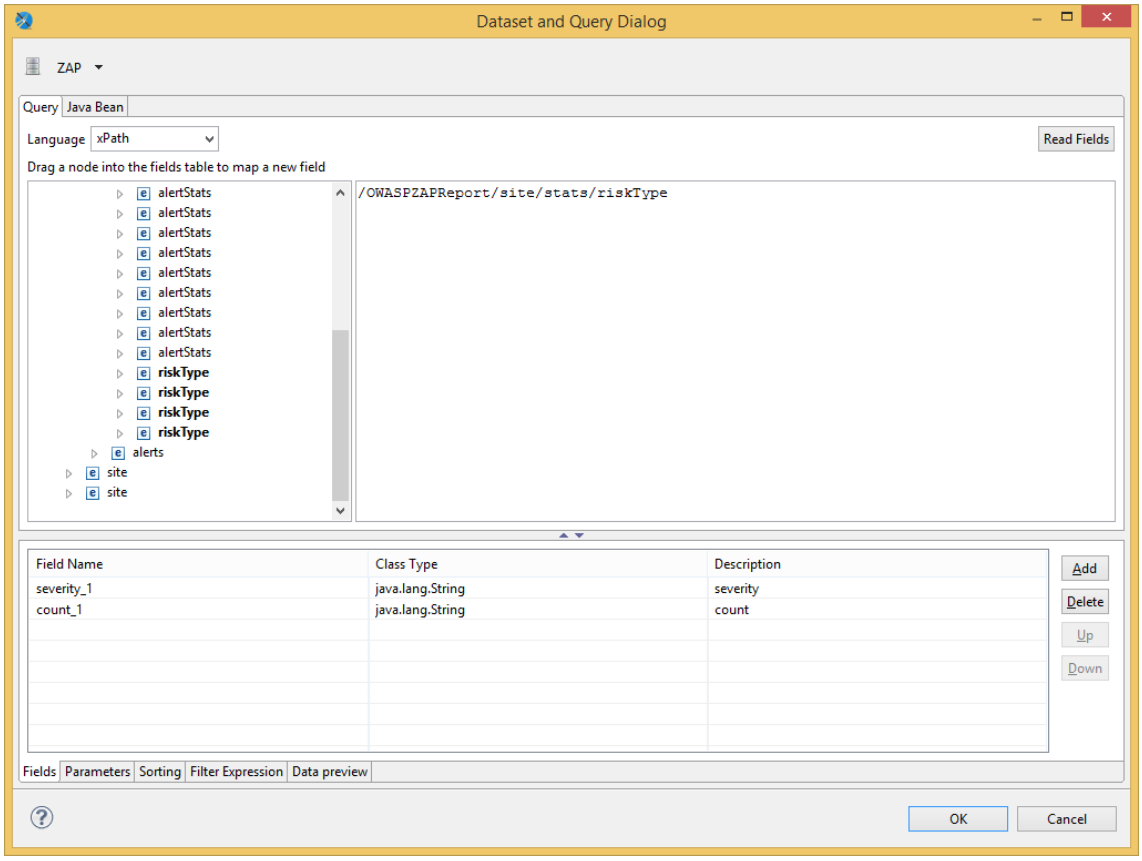


Il·lustració 73: Creació de un DataSet

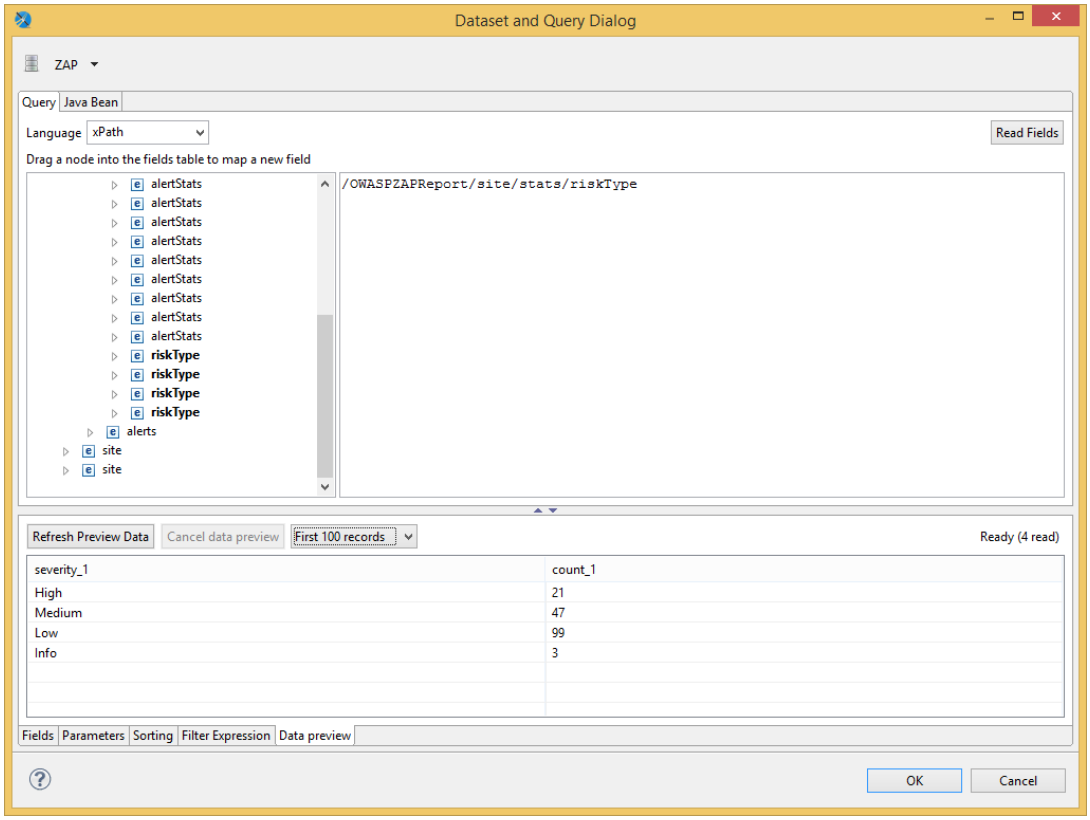


Il·lustració 74: Creació de un DataSet (II)

Veiem aquí la definició i les dades de tipus de risc.



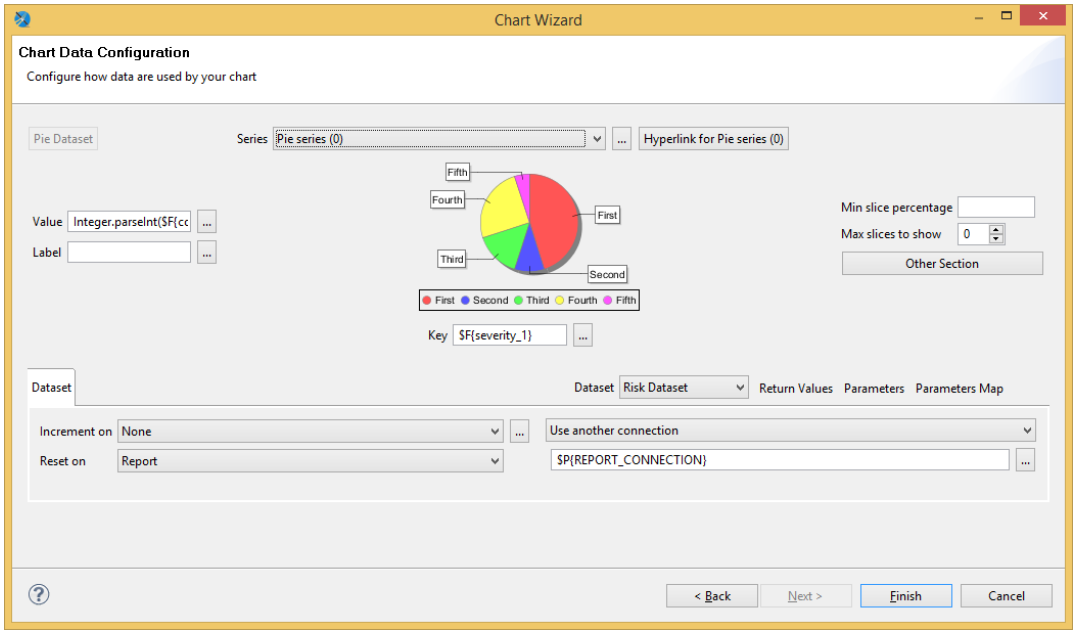
Il·lustració 75: Creació de un DataSet (III)



Il·lustració 76: Creació de un DataSet (IV)

Comprovem també la definició i dades de la severitat dels riscos. Les dades s'agafen de el XML que fem servir de exemple.

Un cop ja tenim els datasets definits podem passar a crear les estructures de dades. Per crear els gràfics farem servir el ajudant de JasperSoft Studio:

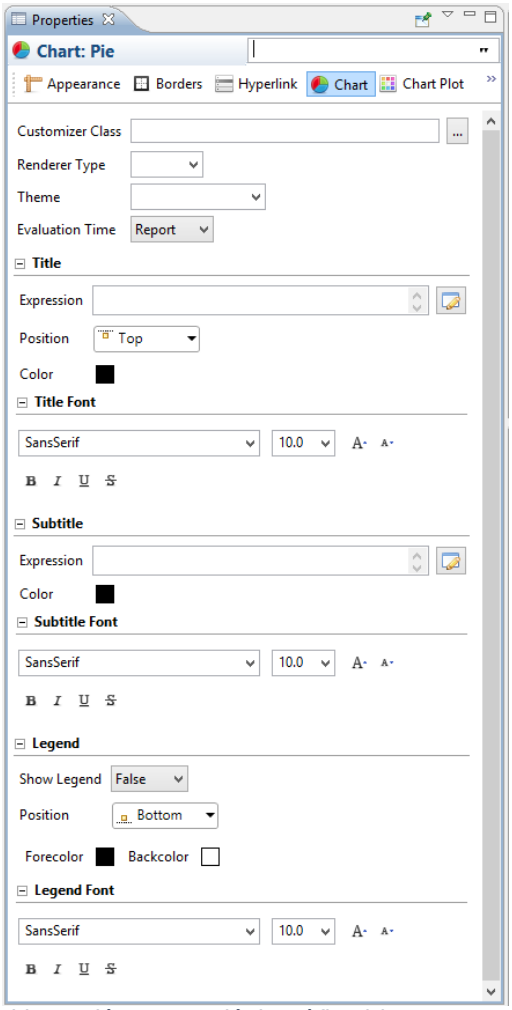


Il·lustració 77: Creació de gràfics

Agafem com a *dataset* el que hem preparat de riscos, de manera que podem posar com a clau del gràfic la severitat. Els valors seran la addició de aparicions que tenim precalculada, que passarem a un valor numèric per què es pugui fer servir:

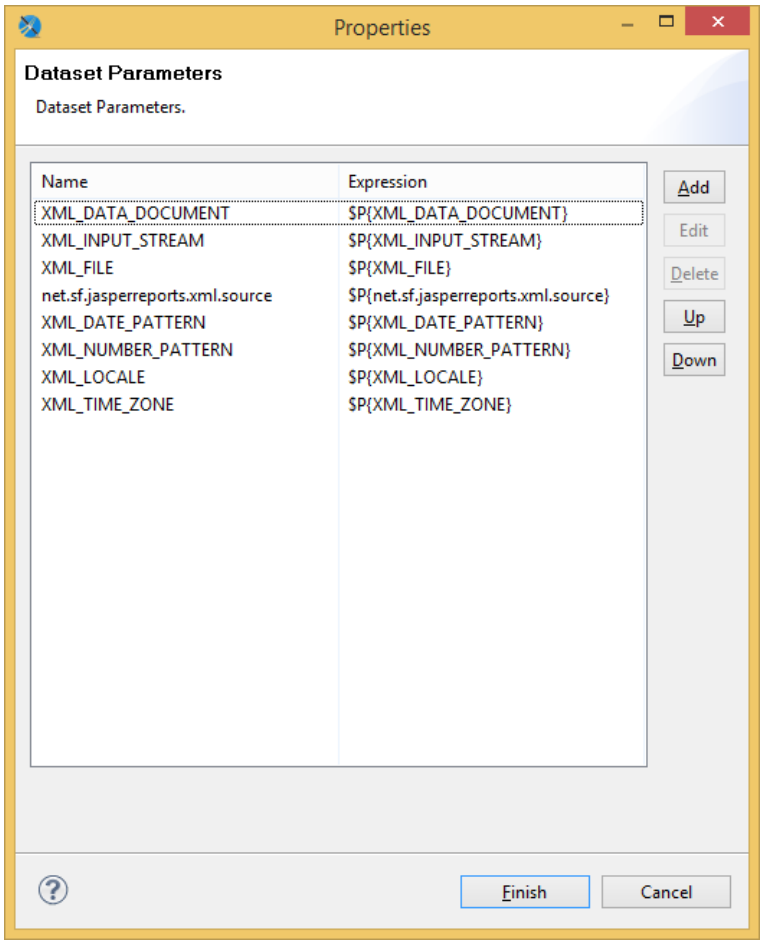
Integer.parseInt($F{count_1})

Quan ja ho tenim establert podem definir les opcions del gràfic:



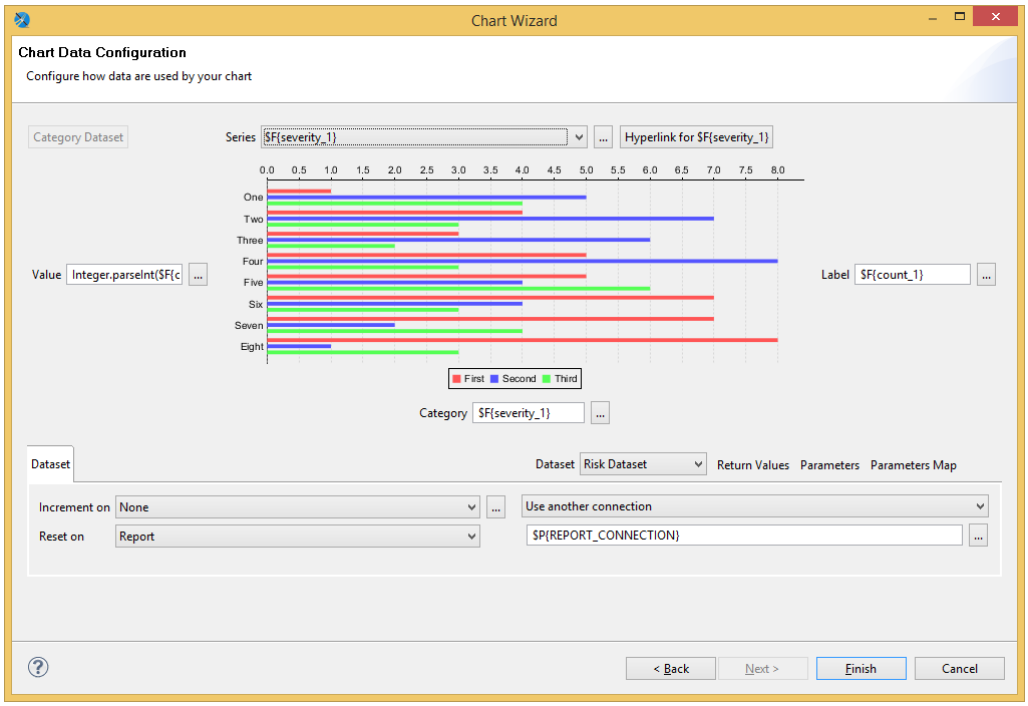
Il·lustració 78: Creació de gràfics (II)

Un cop fet es pot previsualitzar el gràfic, però veiem que no sembla que hi hagin dades per popular-lo un cop es renderitza. Investigant veiem que sembla que la connexió no es transmet bé del dataset principal al que hem definit. Copiem les propietats de lectura del dataset:

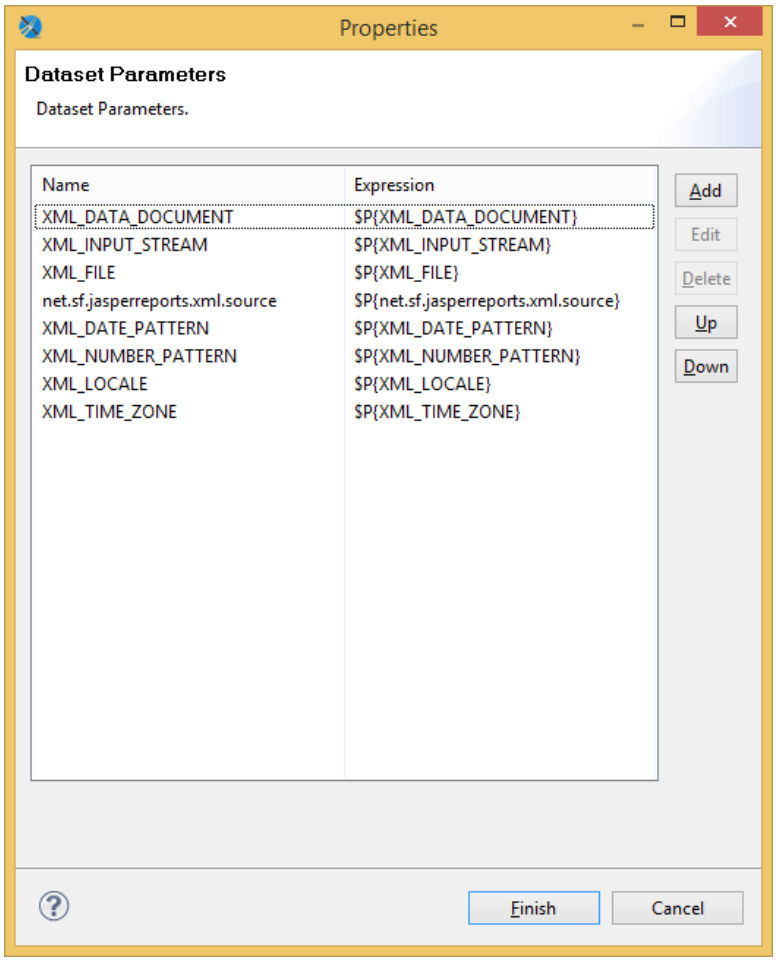


Il·lustració 79: Creació de gràfics (III)

Això arregla el problema, i podem repetir-ho per a altres tipus de gràfics:

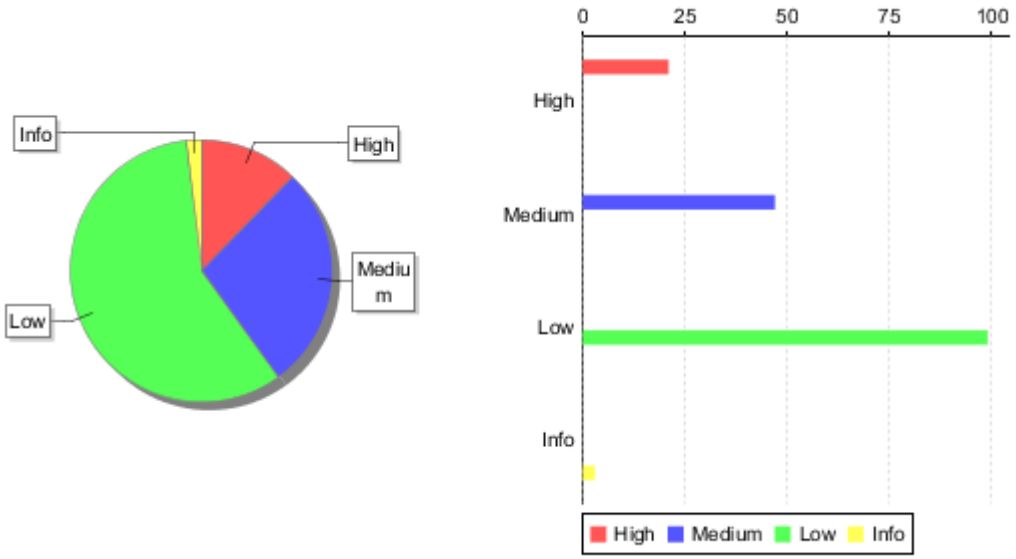


Il·lustració 80: Creació de gràfics (IV)



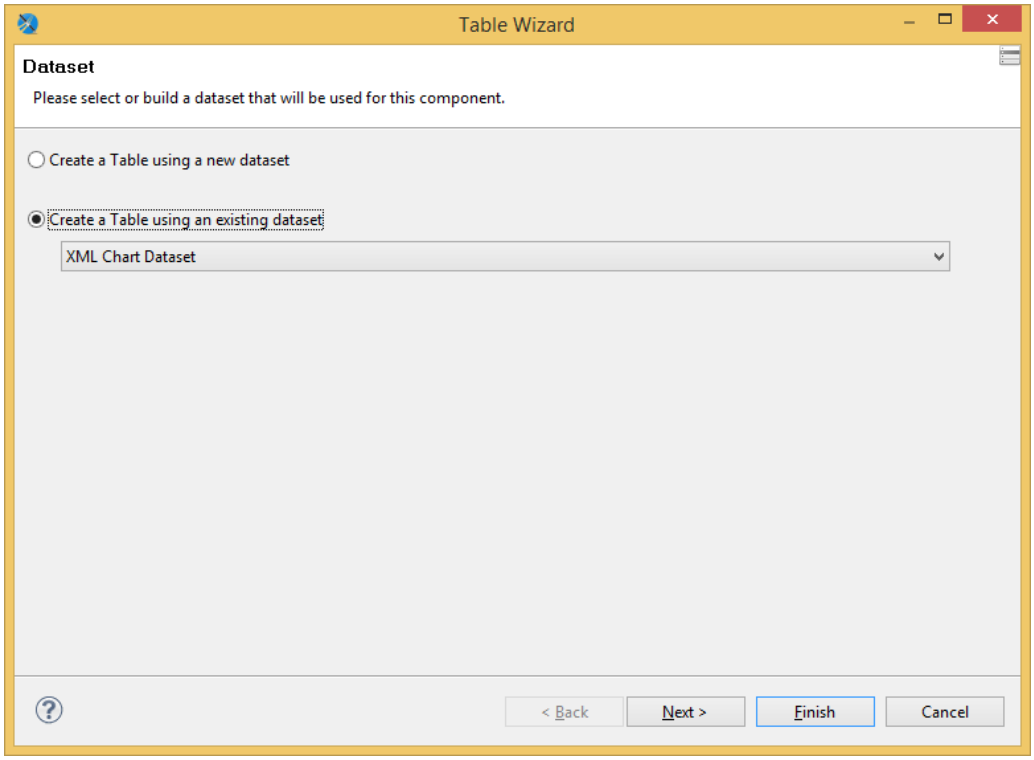
Il·lustració 81: Creació de gràfics (V)

Veiem el resultat, agafant les dades de severitat dels riscos:

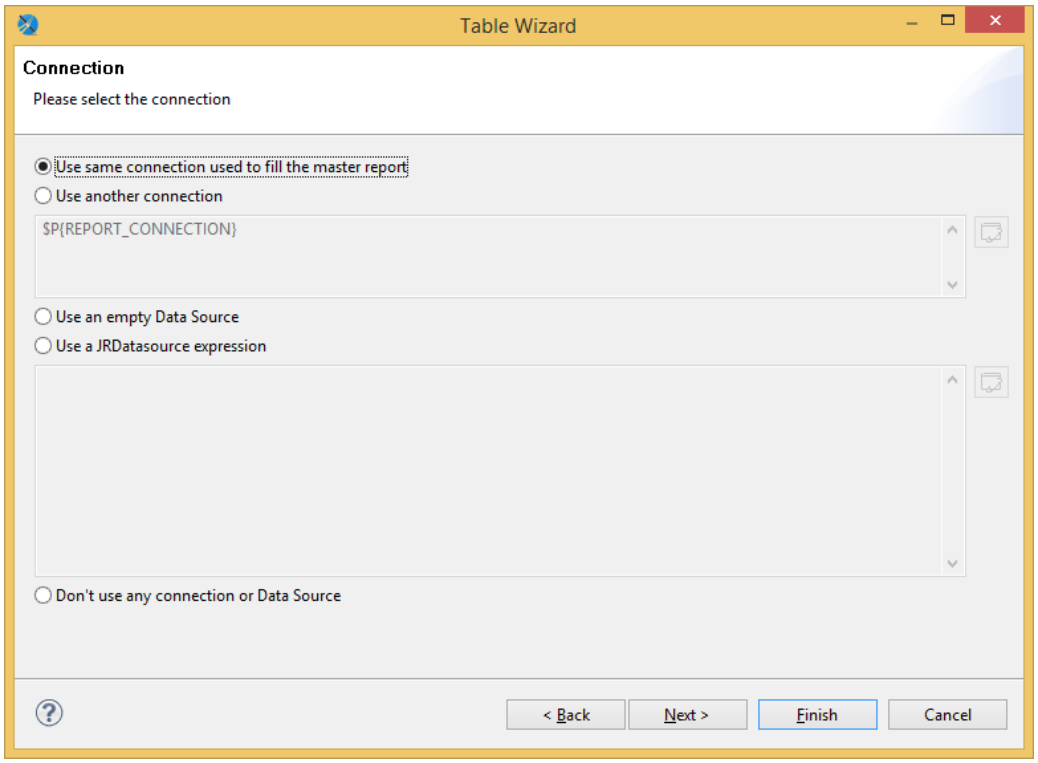


Il·lustració 82: Gràfics de riscos

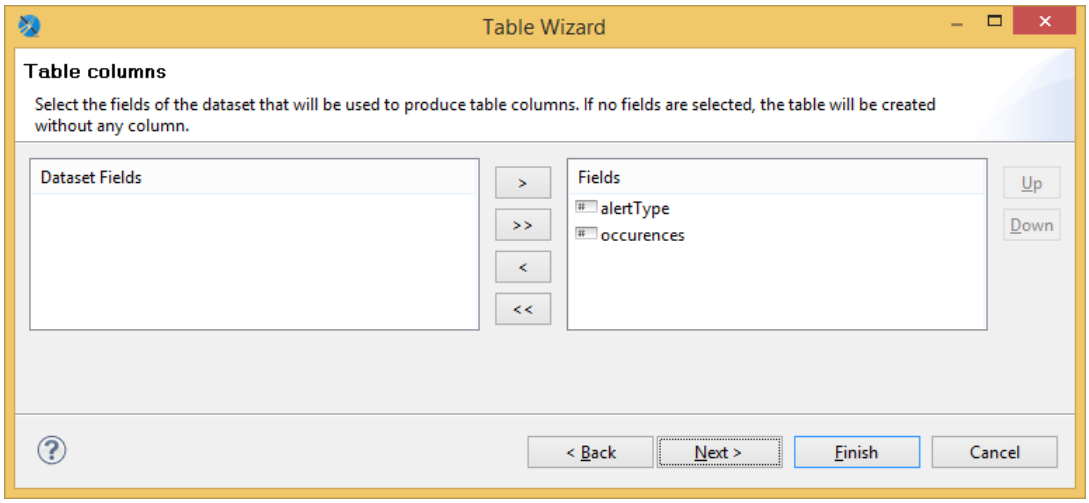
Podem realitzar un procés similar per a crear taules:



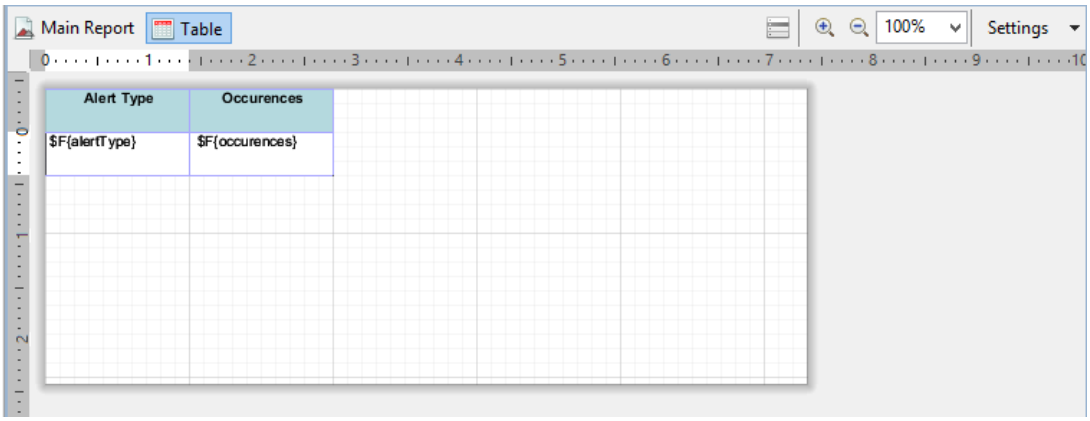
Il·lustració 83: Creació de taules



Il·lustració 84: Creació de taules (II)

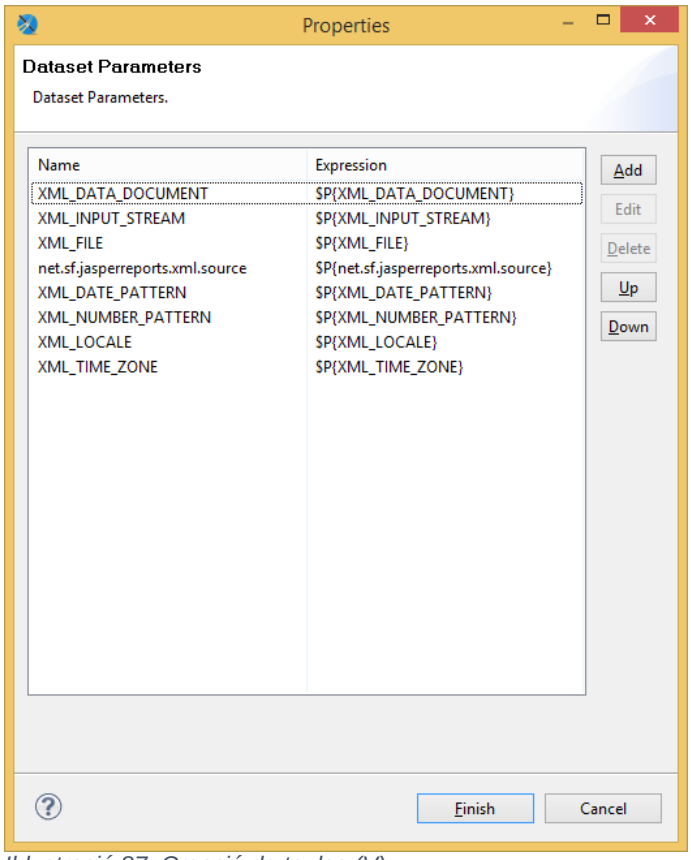


Il·lustració 85: Creació de taules (III)



Il·lustració 86: Creació de taules (IV)

Un cop ja ho tenim veiem que les dades no estan poblades, de manera similar a amb els gràfics. Configurem també els paràmetres de connexió:



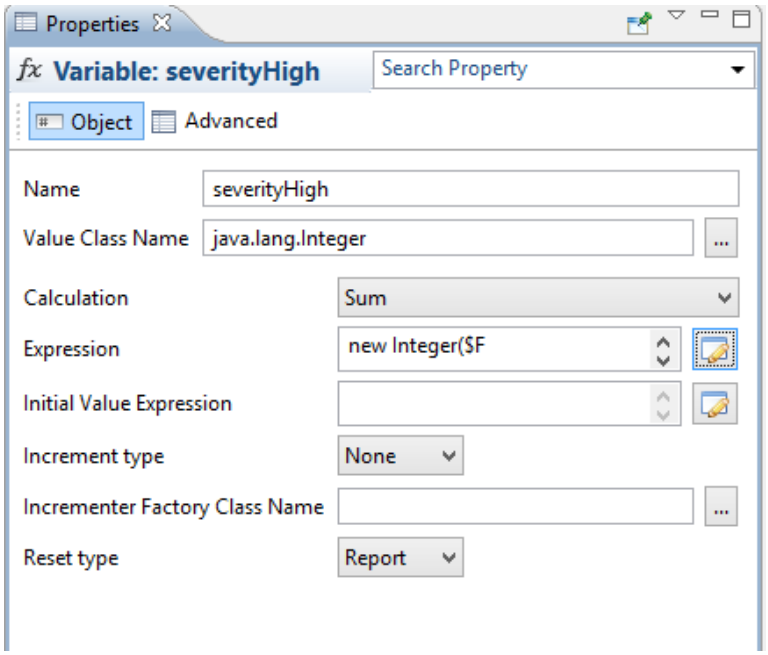
Il·lustració 87: Creació de taules (V)

Quan els configurem podem obtenir ja una taula poblada:

Alert Type	Occurences
Absence of Anti-CSRF Tokens	3
Cross Site Scripting (Reflected)	9
Information Disclosure - Sensitive	3
Path Traversal	2
Private IP Disclosure	2
Remote File Inclusion	2

Il·lustració 88: Taula de tipus de alertes

Hem de repetir que les estadístiques que hem afegit seran la única manera senzilla d'aconseguir treballar amb gràfics i taules, però també es poden aconseguir adicions al sumari fent servir variables. Per exemple, per aconseguir els números de la severitat dels riscos podem fer una variable que conti les aparicions de cada un:

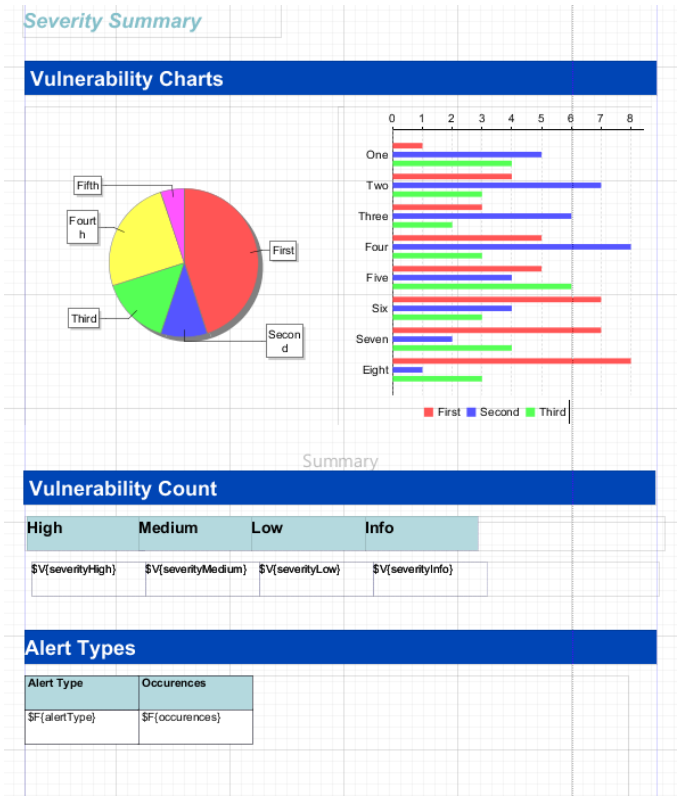


Il·lustració 89: Creació de variables

```
new Integer($F{riskdesc}.startsWith("High")? 1:0
```

D'aquesta manera, fent-ho per a tots els tipus de severitat, també podríem preparar les dades d'aquesta manera. Això seria molt més complicat amb els tipus de risc donat que no sabem quantes variacions hi haurien.

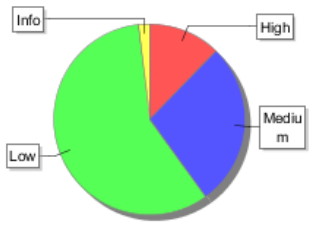
Finalment fem una proposta de disseny de resum de report al final amb aquesta informació:

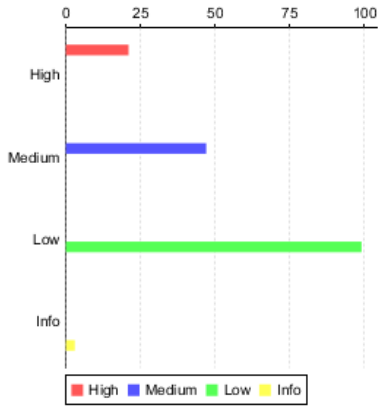


Il·lustració 90: Resum de vulnerabilitats

I ho podem veure renderitzat en dues pàgines:

Vulnerability Charts





Vulnerability Count

High	Medium	Low	Info
21	47	99	3

Alert Types

Alert Type	Occurences
Absence of Anti-CSRF Tokens	3
Cross Site Scripting (Reflected)	9
Information Disclosure - Sensitive	3
Path Traversal	2
Private IP Disclosure	2
Remote File Inclusion	2

Alert Type	Occurences
Remote OS Command Injection	2
SQL Injection	4
Server Side Code Injection - PHP Code	2
Web Browser XSS Protection Not	47
X-Content-Type-Options Header	47
X-Frame-Options Header Not Set	47

Il·lustració 91: Resum de vulnerabilitats (II)

2.4.9 Proposta de múltiples formats de sortida

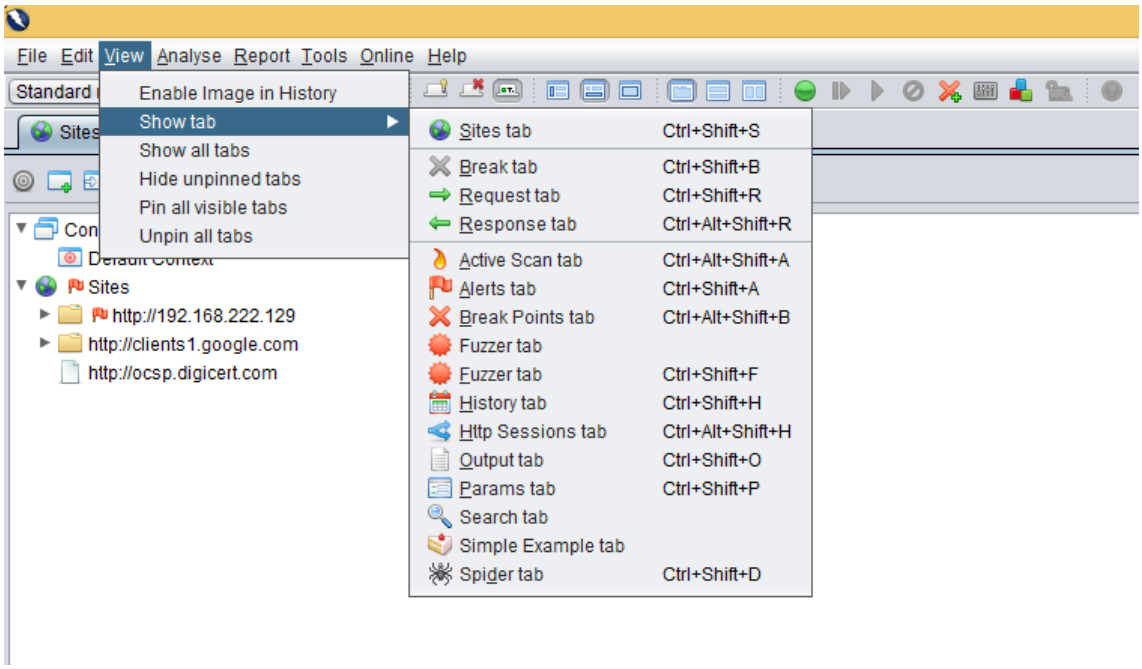
Un cop ja tenim els elements de disseny hem de pensar en afegir valor donant varietat als formats de sortida que estem oferint amb la extensió. Hem vist abans com oferíem sortida en format PDF, pel que ja era un avançament del que ofereix ZAP per defecte, que son els formats HTML i XML, en el sentit de que és còmode de enviar o de imprimir.

```
String resultFile = folderPath+file.getName()+".pdf";
      JasperExportManager.exportReportToPdfFile(file,
resultFile);
```

Això ho estàvem realitzant fent servir la classe *JasperExportManager* que ens proveeix amb la capacitat de exportar un report a diversos formats, com son PDF, HTML i XML (veure apartat 2.4.7). Podrem fer servir aquesta mateixa classe per a crear aquests formats. Per exemple, per generar HTML:

```
private void generateHTML(JasperPrint jasperPrintFile){
   String resultFile =
folderPath+jasperPrintFile.getName()+".html";
   try {

JasperExportManager.exportReportToHtmlFile(jasperPrintFile,
resultFile);
      openFile(new File(resultFile));
   } catch (JRException e) {
      e.printStackTrace();
   }
}
```

Com tenim varius formats que enviar ara haurem de crear un menú que serveixi per a poder escollir les diferents alternatives. En els exemples de extensions no trobem informació que ens ajudi, així com a la wiki o al grup de desenvolupadors, però si que trobem que un dels menús del nucli de ZAP en que s'ensenyen diferents pestanyes tenim un cas en que es fa algo similar:



Il·lustració 92: Menu amb múltiples opcions

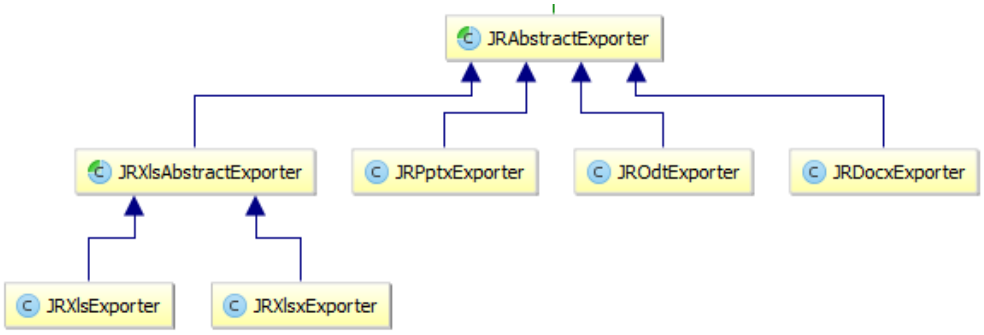
Busquem el codi que s'ha fet servir per a fer aquest menú per ZAP i veiem que es fa servir un *JMenu*, que és un element de menú bàsic de Java, i s'afegeixen elements a aquest menú. Provem a fer-ho servir i veiem que està suportat afegir un submenú d'aquesta manera al menú de *Report* de ZAP, pel que podem afegir un dels elements per a cada un dels formats que volem suportar. Afegirem també, donat que ens és senzill fer-ho, combinacions de tecles per generar més directament els reports en diferents formats. Escollim els formats PDF, HTML, DOCx, PPTx, ODT i XLS com els formats més rellevants a afegir:

```
private JMenu getJasperMenuTypes(){

    JMenu submenu = new JMenu(Constant.messages.getString(PREFIX
+ ".topmenu.tools.menu"));
    submenu.setMnemonic(KeyEvent.VK_S);

    JMenuItem menuItem = new JMenuItem("Generate PDF");
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
          KeyEvent.VK_1, ActionEvent.ALT_MASK));
    menuItem.addActionListener(new
java.awt.event.ActionListener() {
       @Override
       public void actionPerformed(java.awt.event.ActionEvent ae)
{
          JasperPrint file = prepareReport();
          generatePDF(file);
       }
    });
    submenu.add(menuItem);

    menuItem = new JMenuItem("Generate HTML");
    menuItem.setAccelerator(KeyStroke.getKeyStroke(
          KeyEvent.VK_2, ActionEvent.ALT_MASK));
    menuItem.addActionListener(new
```

```
java.awt.event.ActionListener() {
      @Override
      public void actionPerformed(java.awt.event.ActionEvent ae)
{
         JasperPrint file = prepareReport();
         generateHTML(file);
      }
   });
   submenu.add(menuItem);

   menuItem = new JMenuItem("Generate Docx");
   menuItem.setAccelerator(KeyStroke.getKeyStroke(
         KeyEvent.VK_3, ActionEvent.ALT_MASK));
   menuItem.addActionListener(new
java.awt.event.ActionListener() {
      @Override
      public void actionPerformed(java.awt.event.ActionEvent ae)
{
         JasperPrint file = prepareReport();
         generateDOCX(file);
      }
   });
   submenu.add(menuItem);

   menuItem = new JMenuItem("Generate PPTx");
   menuItem.setAccelerator(KeyStroke.getKeyStroke(
         KeyEvent.VK_4, ActionEvent.ALT_MASK));
   menuItem.addActionListener(new
java.awt.event.ActionListener() {
      @Override
      public void actionPerformed(java.awt.event.ActionEvent ae)
{
         JasperPrint file = prepareReport();
         generatePPTX(file);
      }
   });
   submenu.add(menuItem);

   menuItem = new JMenuItem("Generate ODT");
   menuItem.setAccelerator(KeyStroke.getKeyStroke(
         KeyEvent.VK_5, ActionEvent.ALT_MASK));
   menuItem.addActionListener(new
java.awt.event.ActionListener() {
      @Override
      public void actionPerformed(java.awt.event.ActionEvent ae)
{
         JasperPrint file = prepareReport();
         generateODT(file);
      }
   });
   submenu.add(menuItem);

   menuItem = new JMenuItem("Generate XLS");
   menuItem.setAccelerator(KeyStroke.getKeyStroke(
         KeyEvent.VK_6, ActionEvent.ALT_MASK));
   menuItem.addActionListener(new
java.awt.event.ActionListener() {
      @Override
      public void actionPerformed(java.awt.event.ActionEvent ae)
{
         JasperPrint file = prepareReport();
```

```
            generateXLS(file);
        }
    });
    submenu.add(menuItem);
    return submenu;
}
```

Malauradament no podem generar els formats DOCx, PPTx, ODT i XLS de la mateixa manera que els formats PDF i HTML, pel que hem de cercar alternatives per a poder generar-los. Investigant trobem una classe abstracta *JRAbstractExporter* que te com a implementacions diversos exportadors que ens serviran per a aquest propòsit:



Il·lustració 93: Diagrama de JRAbstractExporter

Una ullada als fòrums de StackOverflow ens dona una idea sobre la utilització d'ells, que és similar per cada format:

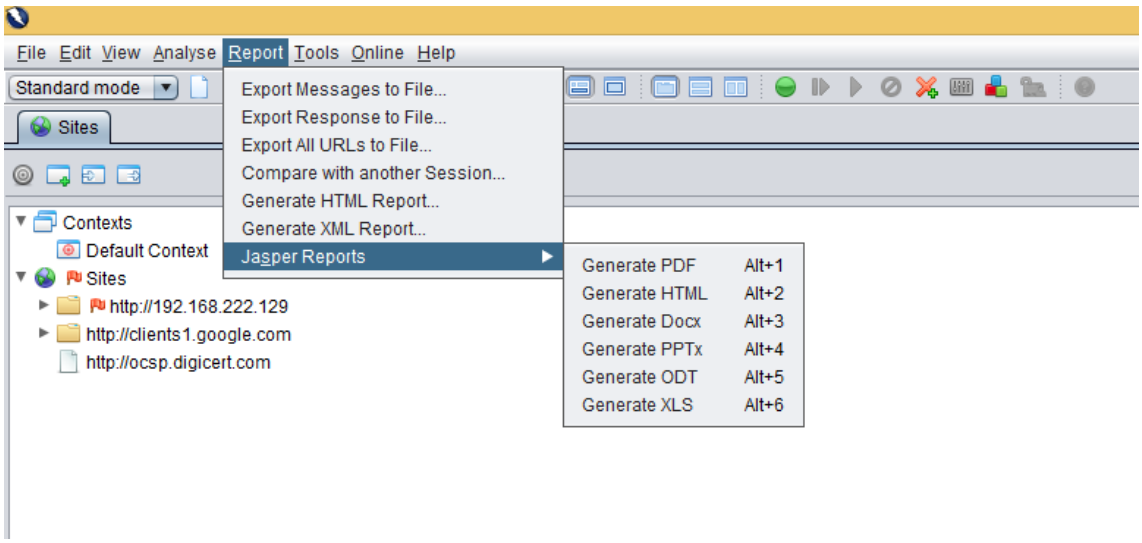
```
private void generateHTML(JasperPrint jasperPrintFile){
    String resultFile =
folderPath+jasperPrintFile.getName()+".html";
    try {

JasperExportManager.exportReportToHtmlFile(jasperPrintFile,
resultFile);
        openFile(new File(resultFile));
    } catch (JRException e) {
        e.printStackTrace();
    }
}

private void generatePDF(JasperPrint jasperPrintFile){
    String resultFile =
folderPath+jasperPrintFile.getName()+".pdf";
    try {
        JasperExportManager.exportReportToPdfFile(jasperPrintFile,
resultFile);
        openFile(new File(resultFile));
    } catch (JRException e) {
        e.printStackTrace();
    }

}

private void generateDOCX(JasperPrint jasperPrintFile){
```

```
   String resultFile =
folderPath+jasperPrintFile.getName()+".docx";
   try {
      JRDocxExporter exporter = new JRDocxExporter();
      exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrintFile);

exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME,
resultFile);
      exporter.exportReport();
      openFile(new File(resultFile));
   } catch (JRException e) {
      e.printStackTrace();
   }
}

private void generatePPTX(JasperPrint jasperPrintFile){
   String resultFile =
folderPath+jasperPrintFile.getName()+".pptx";
   try {
      JRPptxExporter exporter = new JRPptxExporter();
      exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrintFile);

exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME,
resultFile);
      exporter.exportReport();
      openFile(new File(resultFile));
   } catch (JRException e) {
      e.printStackTrace();
   }
}

private void generateODT(JasperPrint jasperPrintFile){
   String resultFile =
folderPath+jasperPrintFile.getName()+".odt";
   try {
      JROdtExporter exporter = new JROdtExporter();
      exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrintFile);

exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME,
resultFile);
      exporter.exportReport();
      openFile(new File(resultFile));
   } catch (JRException e) {
      e.printStackTrace();
   }
}

private void generateXLS(JasperPrint jasperPrintFile){
   String resultFile =
folderPath+jasperPrintFile.getName()+".xls";
   try {
      JRXlsExporter exporter = new JRXlsExporter();
      exporter.setParameter(JRExporterParameter.JASPER_PRINT,
jasperPrintFile);

exporter.setParameter(JRExporterParameter.OUTPUT_FILE_NAME,
resultFile);
      exporter.exportReport();
```

```
            openFile(new File(resultFile));
        } catch (JRException e) {
            e.printStackTrace();
        }
    }
```

Com veiem la principal diferència és que hem de passar els paràmetres de manera separada a la exportació en si, però es tracta de processos fonamentalment similars.

Quan ja tenim els mètodes preparats podem veure com ja existeixen al menú:



Il·lustració 94: Menu amb múltiples opcions (II)

Podrem generar doncs els diferents tipus de elements des del menú de reporting.

2.4.10 Generació de reports d'exemple

Un cop ja tenim tots els elements preparats podem oferir diverses variacions de reports per a que els usuaris es puguin inspirar i crear les variacions com els hi convingui millor.

Oferim els reports:

- *ZAPComplete.jasper*: Report que conté una gran quantitat dels elements de les alertes, incloent els elements que ocupen més volum com la descripció i la solució.
- *ZAP*.jasper: Report que conté només alguns dels elements de les alertes, donant només la descripció i no la solució en les alertes descobertes.
- *ZAPMin.jasper:* Report que conté una quantitat mínima de detalls evitant elements que comporten un volum elevat.
- *ZAPTable.jasper:* Report que conté els mateixos elements que ZAPComplete però que a més conté un exemple de creació de taula.

Oferim en la extensió tant el report compilat com el no compilat per a facilitar la feina amb ells.

2.4.11 Reports ens diferents formats

Un cop ja tenim un resultat podem observar que existeixen petites diferències entre els diferents formats en els que decidim exportar, bàsicament en quant a disposició, mida i altres elements de visualització general. És interessant notar que probablement es vulguin fer proves en tots els formats en els que voldrem exportar quan estem dissenyant el report, per a evitar sorpreses quan al resultat final.

Adjuntem unes captures per que es vegi el mateix report en diferents formats, per veure que tot i haver-hi diferències, són bastant poques, sent el resultat molt ben aconseguit.

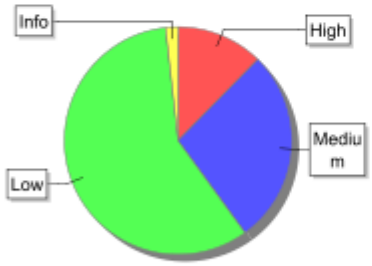
Informational (Medium)

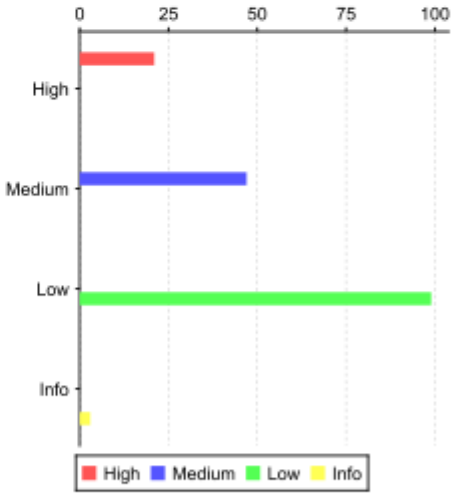
Information Disclosure - Sensitive Informations in URL

Description	The request appeared to contain sensitive information leaked in the URL. This can violate PCI and most organizational compliance policies. You can configure the list of strings for this check to add or remove values specific to your environment				
URI	http://192.168.222.129/ldap/example2.php?name=hacker&password=hacker				
CWEID	null	WASCID	13	Evidence	hacker
Param	password			Confidence	2

Severity Summary

Vulnerability Charts





Vulnerability Count

High	Medium	Low	Info
21	47	99	3

Il·lustració 95: Report final en format PDF

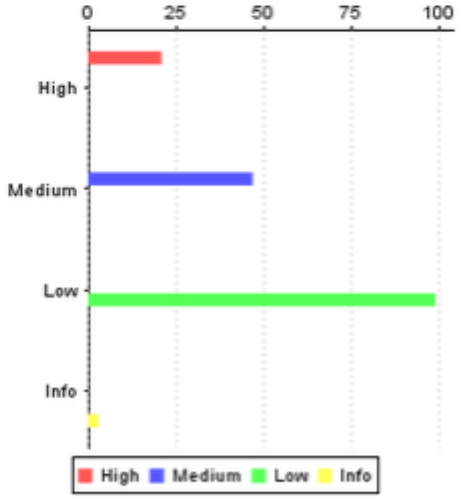
Informational (Medium)

Information Disclosure - Sensitive Informations in URL

Description	The request appeared to contain sensitive information leaked in the URL. This can violate PCI and most organizational compliance policies. You can configure the list of strings for this check to add or remove values specific to your environment				
URI	http://192.168.222.129/ldap/example2.php?name=hacker&password=hacker				
CWEID	null	**WASCID**	13	**Evidence**	hacker
Param	password			**Confidence**	2

Severity Summary

Vulnerability Charts



Vulnerability Count

High	Medium	Low	Info
21	47	99	3

Il·lustració 96: Report final en format HTML

Informational (Medium)

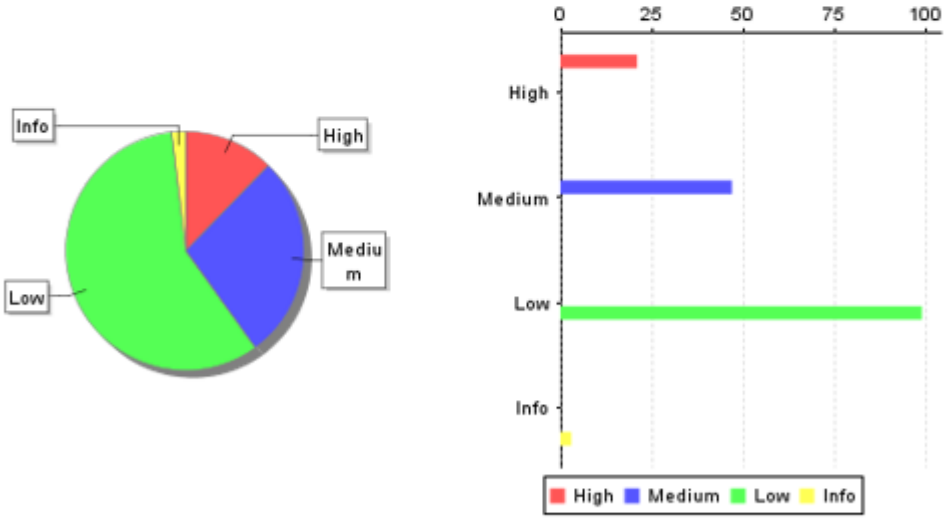
Information Disclosure - Sensitive Informations in URL

Description	The request appeared to contain sensitive information leaked in the URL. This can violate PCI and most organizational compliance policies. You can configure the list of strings for this check to add or remove values specific to your environment
URI	http://192.168.222.129/ldap/example2.php?name=hacker&password=hacker

CWEID	null	WASCID	13	Evidence	hacker
Param	password			Confidence	2

Severity Summary

Vulnerability Charts



Vulnerability Count

High	Medium	Low	Info
21	47	99	3

Il·lustració 97: Report final en format DOCx

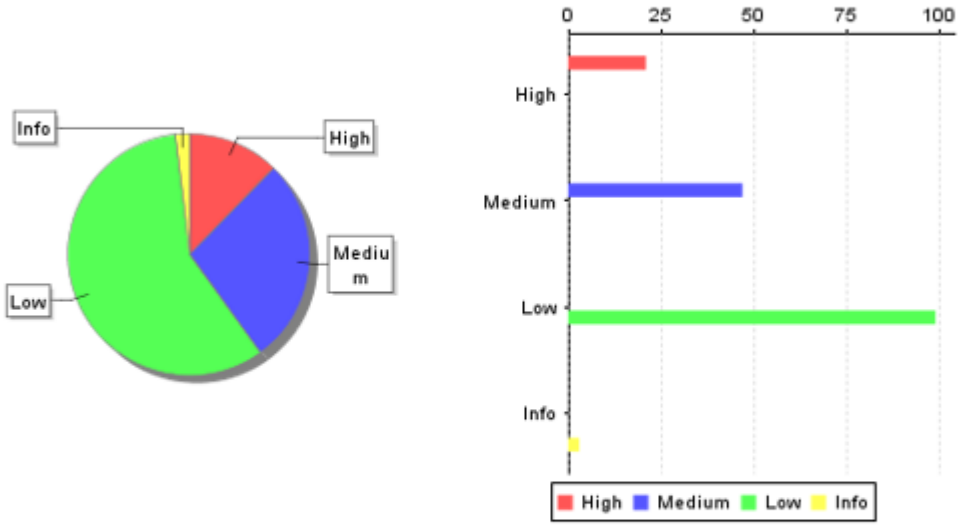
Informational (Medium)

Information Disclosure - Sensitive Informations in URL

Description	The request appeared to contain sensitive information leaked in the URL. This can violate PCI and most organizational compliance policies. You can configure the list of strings for this check to add or remove values specific to your environment				
URI	http://192.168.222.129/ldap/example2.php?name=hacker&password=hacker				
CWEID	null	**WASCID**	13	**Evidence**	hacker
Param	password			**Confidence**	2

Severity Summary

Vulnerability Charts



Vulnerability Count

High	Medium	Low	Info
21	47	99	3

Il·lustració 98: Report final en format PPTx

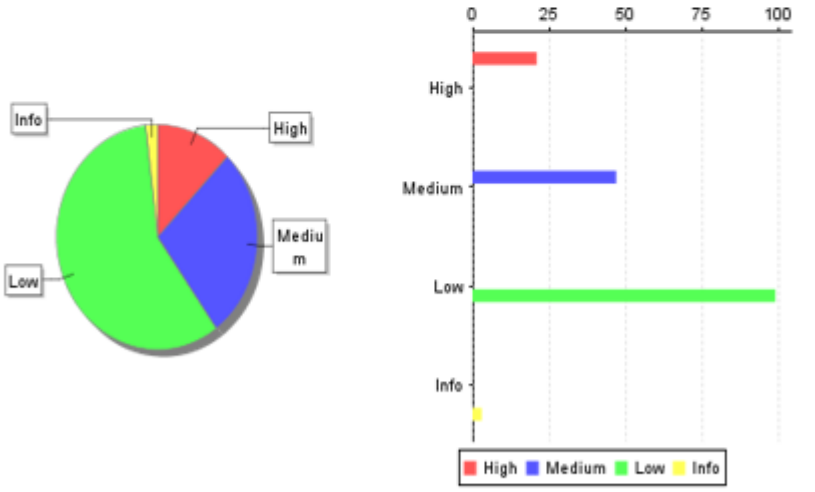
Informational (Medium)

Information Disclosure - Sensitive Informations in URL

Descriptio n	The request appeared to contain sensitive information leaked in the URL. This can violate PCI and most organizational compliance policies. You can configure the list of strings for this check to add or remove values specific to your environment				
URI	http://192.168.222.129/ldap/example2.php?name=hacker&password=hacker				
CWEID	null	**WASC ID**	13	**Eviden**	hacker
Param	password			**Confidenc**	2

Severity Summary

Vulnerability Charts



Vulnerability Count

High	Medium	Low	Info
21	47	99	3

Il·lustració 99: Report final en format XLS

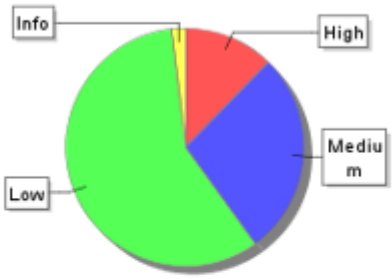
Informational (Medium)

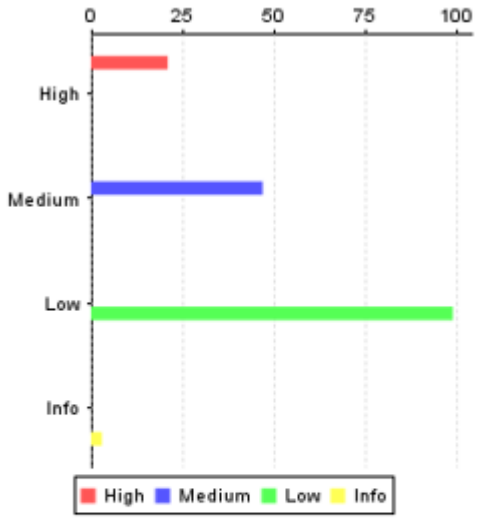
Information Disclosure - Sensitive Informations in URL

Description	The request appeared to contain sensitive information leaked in the URL.This can violate PCI and most organizational compliance policies. You can configure the list of strings for this check to add or remove values specific to your environment				
URI	http://192.168.222.129/ldap/example2.php?name=hacker&password=hacker				
CWEID	null	**WASCID**	13	**Evidence**	hacker
Param	password			**Confidence**	2

Severity Summary

Vulnerability Charts





Vulnerability Count

High	Medium	Low	Info
21	47	99	3

Il·lustració 100: Report final en format ODT

3. Conclusions

Aquest projecte ha estat molt entretingut de realitzar, si bé ha estat llarg a vegades perquè hem anat trobant problemes constants a l'hora de desenvolupar, tant en el codi de la extensió com en el del disseny dels diferents reports de JasperReports.

No és el primer cop que treballo amb eines open source, de fet hi estic bastant acostumat. És veritat que he pecat de ingenu al pensar que no tindria masses problemes fent una extensió i treballar amb la integració amb JasperReports, donat que tot i que havia vist que hi havia poca documentació pensava que podria fer-ho de manera natural o senzilla. Ha estat un veritable problema sobretot perquè al fer la investigació vaig veure que tindria un problema amb ZAP, però no me'l esperava amb JasperReports, pensava que em seria molt més senzill del que m'ha estat crear un disseny de report i configurar-ho i agafar les dades que em calien, mostrar-les com jo volia i fer propostes. La veritat és que crec que si no tingués experiència com a desenvolupador aquest projecte no l'hauria pogut acabar a temps, com a mínim no del tot.

És una llàstima perquè considero que les dues eines son molt potents. ZAP és una eina que realment m'agrada, i espero col·laborar-hi més, sobretot ara que tinc una bona idea de com fer-ho. JasperReports està feta servir a molts llocs i està molt popularitzada com a eina de reporting, però és molt dura per a iniciar-se, i realment molta de la informació que es troba és de fa molts anys pel que en molts casos no és vigent o no aplica a les transformacions que ha anat veient amb el temps.

Es pot dir que hem assolit els objectius, si bé han anat variant durant el inici del projecte, que va ser quan el vam anar definint. Quan em vaig plantejar el projecte inicialment vaig pensar en una aplicació web, que tingués una interfície que anés presentant diversos resultats de anàlisis de ZAP, es poguessin posar de costat un amb l'altre, etc. Continuo pensant que seria interessant, però no crec que tal i com està feta la arquitectura de ZAP actual sigui molt pràctic. Si hem de anar pujant cada report individualment a una aplicació que està apart crec que no convidaríem molt al us de la aplicació.

En canvi tal i com la hem fet ara crec que es pot fer servir per un nombre elevat de usuaris, ja no només per modificar-la, que per sort ho hem fet més fàcil posant exemples sobre com agafar totes les dades, de manera que qui hi vulgui treballar, si no fa algo molt complicat només cal que toqui el disseny dels exemples que proposem, sinó que els reports que donem per defecte son bastant atractius en si mateixos, i crec que son una bona alternativa als existents. Per suposat que això és completament subjectiu, però precisament per això volíem un report que fos configurable.

M'ha agradat tenir el recolzament del Simon i altres desenvolupadors de ZAP que m'han anat resolent dubtes quan els hi he anat escrivint. Crec que és una molt bona experiència per participar en codi obert i em motiva

per a seguir participant en aquest i més projectes. Personalment era un dels meus objectius que tenia des de feia temps i que no havia pogut materialitzar mai.

La planificació la he seguit com he pogut. M'ha donat la impressió que he treballat moltes més hores de les que havia planejat en el disseny dels reports, però és difícil quantificar donat que he tingut èpoques en les que li he pogut dedicar moltes més hores que en altres, en base al volum de feina que tenia a l'empresa, no sempre podia permetrem unes hores a la tarda, i hi ha hagut setmanes que si que he pogut dedicar-hi molt de temps al projecte, pel que ha compensat.

Crec que en quant a planificació el que si ha sortit bé és que he acabat les parts principals quan ho tenia planejat, de manera que he pogut anar treballant en la documentació mentre anava fent millores al codi i als reports. Això m'ha permès plasmar el procés de treball; corria el risc, si se'm allargava el desenvolupament, de no poder explicar tot el que s'havia realitzat.

Faré servir part d'aquest projecte, sobretot el que explica el inici de les configuracions i el entorn de ZAP, per millorar la Wiki de ZAP, especialment ara que estan fent una reestructuració i revisant el contingut, donat que no he estat l'únic que els hi ha comentat que ha tingut problemes molt seriós trobant informació, i ara mateix no és gaire apte per gent que no te certa experiència desenvolupant.

És veritat que s'han sacrificat elements que havíem considerat al principi, donat que no enteníem be la arquitectura de ZAP, però crec que aquests canvis son precisament en els que es podrà treballar més endavant. ZAP està realitzant un canvi de arquitectura per a fer servir una base de dades no orientada a fitxer, sembla que finalment es farà servir MySQL, i això ens donarà la possibilitat de tenir accessos simultanis a la base de dades i accés al històric. No només això, sinó que a més estem parlant ja de *ZAP as a Service*, pel que podria ser molt interessant poder combinar la idea que teníem inicialment de un reporting persistent amb aquest servei, donada la nova arquitectura.

Es podrà també ampliar la informació que obtenim, i serà molt més senzill crear els reports en la extensió que ja tenim si treballem amb un datasource amb una connexió SQL, i no amb un datasource XML que està limitat a les dades que jo li estic donant perquè son les que hem trobat que es proporcionen habitualment al usuari.

Un altre punt que en el que es podria treballar al futur és oferir uns reports que siguin més configurables des del menú mateix de reports. Bé sigui a nivell de colors, camps configurables o qualsevol altre proposició, seria un pas més endavant si es pogués treballar tant amb JasperReports com amb una configuració més bàsica directament des de la interfície.

Finalment, agraeixo molt la oportunitat de haver pogut participar en aquest projecte, sobretot per que no només no se m'ha imposat cap limitació a l'hora de voler integrar-me amb ZAP, que no era la proposta inicial, sinó que se'm ha animat i recolzat des del costat de la tutoria. També vull agrair que sempre se'm hagin contestat totes les preguntes i dubtes de manera ràpida, fent funcionar molt be la dinàmica de treball.

4. Glossari

ZAP: Zed attack proxy.
Idea: Programari de desenvolupament Intellij Idea.
Jasper: JasperReports, llibreria de creació de reports.
Datasource: Origen de dades, utilitzat per omplir de dades el report.
Ant: Apache Ant.
BIRT: Business Intelligence and Reporting Tools.
OWASP: Open Web Application Security Project.
Paros. Paros Proxy, projecte en el que ZAP està basat.

5. Bibliografia

La bibliografia consta bàsicament dels llocs web i wikis dels projectes de ZAP i JasperReports, i han estat visitades de manera constant, pel que no posem data, donat que sempre hem anat recorrent a aquestes fonts.

https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project
https://code.google.com/p/zaproxy/
https://github.com/zaproxy/zaproxy
https://github.com/zaproxy/zaproxy/wiki
https://github.com/zaproxy/zap-extensions
https://code.google.com/p/zaproxy/wiki/ZapExtensions
https://angularjs.org/
http://www.codecademy.com/es/learn/learn-angularjs
http://tutorials.jenkov.com/java-itext/index.html
http://itextpdf.com/
http://stackoverflow.com/questions/6625849/pdf-generation-with-java-what-to-use-itext-apache-pdfbox-or-fop
http://blog.jaumesola.com/how-to-create-pdf-files-from-xml
https://xmlgraphics.apache.org/fop/
http://www.findbestopensource.com/tagged/charting-library
http://www.highcharts.com/
http://olex.openlogic.com/packages/eclipseplugin-eclipse_birt
http://www.capterra.com/reporting-software/
http://opensource.com/business/14/6/three-open-source-business-tools
http://java-source.net/open-source/charting-and-reporting
http://stackoverflow.com/questions/8170325/open-source-java-reporting-framework
http://www.innoventsolutions.com/open-source-reporting-review-birt-jasper-pentaho.html
http://www.eclipse.org/birt/
http://www.eclipse.org/birt/documentation/
http://hortonworks.com/wp-content/uploads/2013/09/Actuate_OS_BIRT.pdf
http://community.pentaho.com/projects/reporting/
http://wiki.pentaho.com/display/Reporting/Pentaho+Reporting+-+User+Guide+for+Report+Designer
https://github.com/pentaho/pentaho-reporting
http://www.edureka.co/blog/pentaho-vs-jaspersoft-vs-birt/
http://www.edureka.co/blog/pentaho-tutorial-for-beginners/
http://community.jaspersoft.com/project/jasperreports-library
http://community.jaspersoft.com/wiki/community-wiki
http://community.jaspersoft.com/documentation?version=15786
http://www.tutorialspoint.com/jasper_reports/
http://community.jaspersoft.com/wiki/jaspersoft-samples-reference
http://www.developer.com/java/getting-started-with-jasperreport.html
http://www.tutorialspoint.com/jasper_reports/jasper_report_data_sources.htm
http://java.dzone.com/articles/java-reporting-part-2
http://www.developer.com/java/getting-started-with-jasperreport.html

https://groups.google.com/forum/#!forum/zaproxy-develop
http://stackoverflow.com

6. Recursos

Els recursos, incloent el codi, fitxers de treball, extensió compilada i diferents sortides d'exemple, es poden trobar a:

https://drive.google.com/open?id=0B1X3udtQlwaWfklSQkM2RXdqNHMzZWRwY21WRElkZnduMHNCSnRFNGZraU1vQVhkVXhGZE0&authuser=0