Auditoria i desenvolupament segur

José María Alonso Cebrián
Vicente Díaz Sáez
Antonio Guzmán Sacristán
Pedro Laguna Durán
Alejandro Martín Bailón

PID_00191646

UOC
Universitat Oberta de Catalunya

Índex

1. Introducció

Aquest mòdul descriu algunes de les mesures que s'han d'adoptar per a protegir la infraestructura davant possibles atacs. El problema és que no hi ha una solució perfecta. Hi ha diferents tipus de vulnerabilitat que no disposen de solució, i cada dia apareixen nous tipus d'atac per als quals no s'ha pensat una defensa. Per a acabar-ho d'arreglar, la protecció és un procés difícil i no estàndard que depèn de molts factors.

No obstant això, hi ha una sèrie de procediments i d'aspectes que s'han de considerar durant la implantació i el manteniment que ajuden a fer molt més difícil qualsevol atac. Els atacs són imprevisibles, pel que fa a quan passaran i com passaran: l'important és estar preparat per a quan passin. Aquesta preparació pot evitar la gran majoria de temptatives d'atacs, incloent-hi els automàtics, que no discriminen els objectius, com els cucs que infecten el màxim nombre possible de víctimes. Una protecció adequada pot arribar a evitar fins i tot tipus d'atacs desconeguts avui dia.

Un altre aspecte important que s'ha de considerar és la implantació de polítiques i procediments. Tot això implica seguir una sèrie de polítiques, crear un entorn adequat per a la base de dades, evitar qualsevol funcionalitat innecessària, etc. Això se sol traduir en un acord entre seguretat i funcionalitat.

Finalment, estudiarem eines que ajuden a determinar l'estat de seguretat de la infraestructura administrada i a instaurar les mesures necessàries per a evitar problemes, i també on podem aconseguir recursos per a conèixer les millors pràctiques en aquest sentit. A més, explicarem accions per a assegurar alts nivells de seguretat que fan experts mitjançant auditories.

2. Auditories

Una vegada s'han implantat totes les mesures de seguretat, tota la funcionalitat que calia està preparada, el personal està format adequadament i els plans de contingència i les polítiques per a tots els aspectes possibles estan preparats, és hora de veure quina mesura falla.

Per més que un administrador estigui completament convençut de la seguretat del seu sistema, no en pot estar mai segur del tot. El millor que podem fer és que un auditor extern expert en el tema provi la seguretat i ens ajudi a detectar les possibles debilitats del sistema abans que ho faci un atacant potencial. Per això és molt recomanable que es facin auditories de manera regular.

Les **auditories de caixa negra** posen l'atacant en una situació en què no té cap informació de l'objectiu i ha de partir de zero per a esbrinar tot el que pugui i intentar aconseguir el control d'aquest objectiu. És la situació que trobaria un atacant que no tingués coneixement intern de l'objectiu.

En les **auditories de caixa blanca**, l'auditor disposa de totes les credencials necessàries per a accedir al sistema, i també d'ajuda dels administradors. En aquesta situació, l'auditor utilitza els seus privilegis per a estudiar el sistema a fons des de dins, comprovar que disposa de totes les mesures de seguretat adequades, i també les polítiques recomanades.

2.1. OWASP

OWASP[1] és una associació sense ànim de lucre que vetlla per la seguretat en el Web. Entre les seves comeses hi ha la formació i la conscienciació dels usuaris i els programadors. També generen codi per a fer aplicacions més segures o per a verificar les aplicacions web.

(1)Open Web Application Security Project.

Altres aportacions

Destaquen les de les xerrades periòdiques que organitzen arreu del món per incitar a millorar la seguretat de les aplicacions i proposar noves solucions per a problemes coneguts.

Per la rellevància i la fiabilitat que té destaca el *top ten* de fallades de seguretat informàtica que van publicar el 2007. En aquesta classificació hi tenim les tècniques més comunes i els errors més usuals a l'hora d'auditar la seguretat d'una aplicació web. Es basen en les dades reals dels experts en seguretat que col·laboren amb l'OWASP:

1) *Cross site scripting* (XSS): tècnica que consisteix a introduir codi JavaScript dins de l'aplicació que visita un usuari.

2) Injeccions de codi (SQL Injection, XPath Injection i LDAP Injection): modifica o extreu informació des d'un magatzem de dades.

3) *Remote file inclusion*: executa un fitxer extern al servidor com si hi fons a dins.

4) Referència directa a fitxers (*local file inclusion*, LFI): un usuari pot manipular l'URL per accedir a un recurs que, inicialment, el desenvolupador no havia pensat permetre'n l'accés.

5)*Cross site request forgery* (CSRF): manipular el navegador de l'usuari mitjançant XSS perquè faci accions no volgudes entre dominis.

6) Fuites d'informació i errors no controlats (*path disclosure*): qualsevol error no controlat pot ser el motiu que faci que un atacant conegui dades internes sobre l'aplicació i el seu entorn.

7) Autenticació feble: usar un sistema d'encriptació feble per a assegurar les comunicacions pot donar lloc a fer que els atacants aconsegueixin generar un testimoni (*token*) d'autenticació vàlid per a qualsevol usuari.

8) Emmagatzematge insegur de credencials: emmagatzemar les contrasenyes en text pla o amb algorismes reversibles és un risc greu de seguretat perquè un atacant podria extreure les contrasenyes mitjançant SQL Injection.

9) Comunicacions insegures: enviar informació sense una encriptació prou potent pot donar lloc a fer que un atacant intercepti aquestes comunicacions i les desxifri.

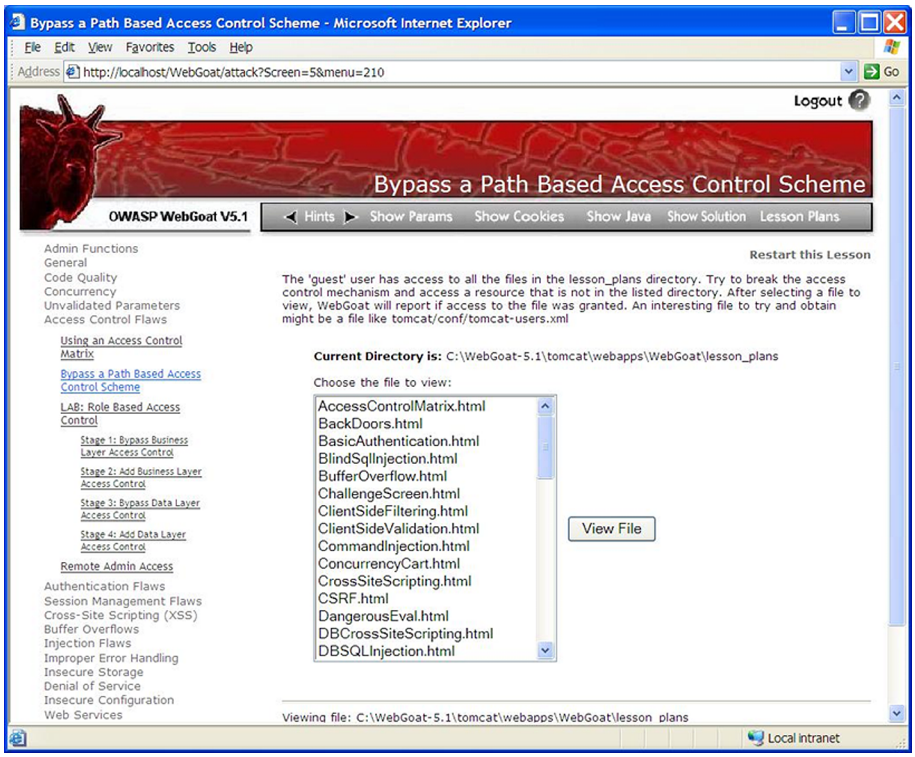
10) Fallades en restringir l'accés a URL: l'última de les fallades de seguretat que exposem en aquesta llista és ocultar recursos en zones sense control d'accés i confiar que no hi haurà cap usuari malintencionat que intentarà localitzar aquestes zones.

Com es veu, la llista cobreix un gran ventall d'opcions pel que fa a fallades de seguretat. El compendi és tan complet que algunes empreses i grups de desenvolupament l'han adoptat com a decàleg per a comprovar la seguretat de les seves aplicacions, com per exemple el sistema de gestió de continguts Plone.

Una altra de les activitats de l'OWASP és la generació d'eines que ajudin al desenvolupament segur d'aplicacions i a l'auditoria de seguretat dels servidors web. Entre els projectes que promouen hi ha els següents:

1) WebGoat: és un projecte educatiu. Consisteix en una aplicació web JSP amb diferents tipus de vulnerabilitats. Això serveix com a entrenament per a les persones que estan interessades a aprendre seguretat web.

Figura 1. WebGoat en funcionament

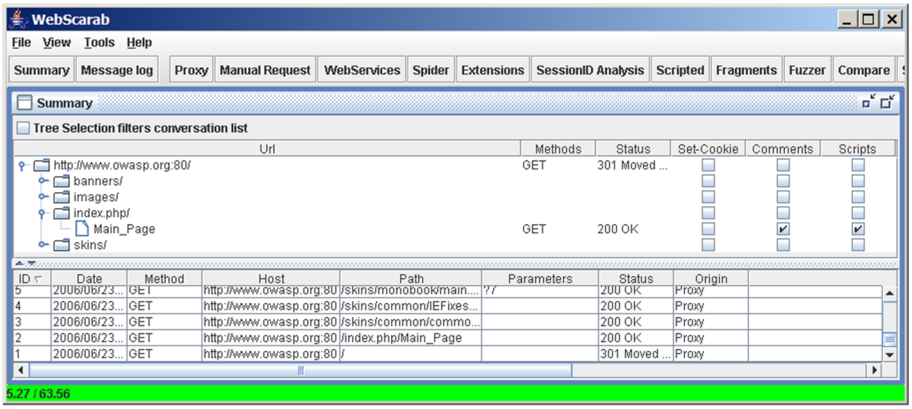


Les fallades que conté l'aplicació inclouen algun dels punts següents:

- *Cross site scripting*.
- Manipulació de camps ocults.
- Debilitats en les galetes (*cookies*) de sessió.
- SQL Injection.
- Blind SQL Injection.
- Informació sensible en comentaris.
- Serveis web insegurs.

2) **WebScarab** : segurament la millor manera d'aconseguir modificar el comportament d'una aplicació web és enviar dades que no s'espera rebre. Per a fer aquesta acció no hi ha res que vagi tan bé com un servidor intermediari (*proxy*) que ens permeti modificar en temps real les peticions que fem. WebScarab és un d'aquests servidors.

Figura 2. Pantalla principal de WebScarab



WebScarab té una sèrie d'opcions que el fan útil per a qualsevol analista de seguretat. Algunes d'aquestes característiques són les següents:

- Extreu comentaris HTML i de *scripts* de les pàgines que es visiten.
- Converteix camps ocults en camps de text normals, per editar-los d'una manera més còmoda.
- Fa accions de cerca d'URL dins de les pàgines en què s'ha navegat, per detectar nous objectius.
- Cerca possibles fallades d'XSS en les pàgines visitades.
- Permet l'*script* per a modificar automàticament algun aspecte d'un lloc web cada vegada que es visiti.

Com es veu és un producte molt complet i complex, que permet a l'auditor deixar de banda certs aspectes i automatitzar-los.

3) AntiSamy : les fallades d'XSS van demostrar que eren devastadores quan un petit cuc[2] XSS va tombar la pàgina web de MySpace.com. D'això neix aquest projecte, una biblioteca que permet sanejar l'entrada del nostre codi web per evitar la inclusió de codi XSS. Per defecte ve amb quatre tipus de regles que són més o menys restrictives segons l'escenari que vulguem recrear:

[(2)] Aquest cuc es deia *Samy*, com el seu creador.

- **antisamy-slashdot.xml**: es basa en les regles del popular lloc de notícies. Solament es poden introduir les etiquetes `<a>`, `<b>`, `<blockquote>`, `<i>` i `<u>`. A més, no es permet cap tipus de CSS.
- **antisamy-ebay.xml**: encara que eBay no ofereix una llista pública de les etiquetes HTML que permet, els desenvolupadors han intentat imitar les funcionalitats que permet aquest popular lloc de compravenda.

- **antisamy-myspace.xml**: són les regles que van donar motiu al projecte. Es permet un gran nombre d'etiquetes i CSS, però cap tipus de JavaScript.

- **antisamy-nothinggoes.xml**: és el filtre més restrictiu. No permet cap tipus d'etiqueta HTML, fitxer CSS o JavaScript. És útil per a usar-lo com a base per als nostres propis filtres.

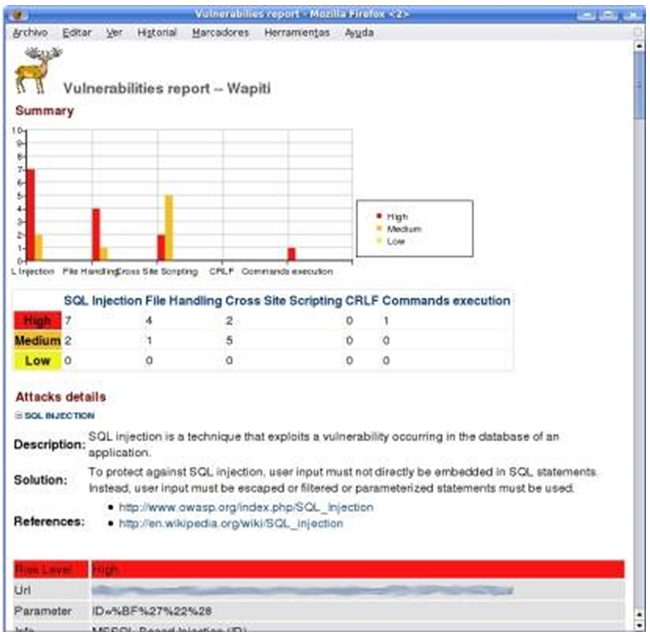
Hi ha versions d'aquesta biblioteca per a Java, .NET i fins i tot Python. En canvi, no s'ha desenvolupat per a PHP, per al qual recomanen l'ús de la biblioteca HTMLPurifier.

4) Wapiti : és un analitzador de seguretat de llocs web. Programat a Python, permet recórrer un lloc web a la recerca d'errors com aquests:

- Extracció d'informació d'errors.
- LDAP Injection.
- XSS.
- SQL Injection.
- Execució d'ordres.

A més d'això, permet generar un informe complet amb gràfics i dades estadístiques sobre les vulnerabilitats oposades.

Figura 3. Informe de Wapiti



2.2. Escàner de vulnerabilitats de caixa negra

Els anomenats *escàners de vulnerabilitats de caixa negra* són els que fan una anàlisi a la recerca de vulnerabilitats sense revisar el codi font de l'aplicació, això és, solament comprova la seguretat dels elements disponibles per a un usuari extern.

El 70% de les aplicacions web són vulnerables a algun tipus d'atac que permet el robatori d'informació sensible o confidencial. És possiblement un dels entorns en què s'hauria de prioritzar més la protecció ja que les pàgines web són accessibles cada dia de tot l'any a tothora. Són la via més utilitzada pels pirates (*hackers*) a l'hora de fer una intrusió, i localitzen els punts febles de les aplicacions en els formularis, els sistemes d'inici de sessions, els continguts dinàmics, les comunicacions contra processadors dorsals (*back-ends*), etc.

Acunetix i W3af són dues conegudes aplicacions utilitzades per a les auditories d'aplicacions web. Aquest tipus d'eines automatitzades s'han d'usar com a complement d'una auditoria de seguretat web, però sense ignorar auditories *pen-testing*[3].

(3)Auditories no automatitzades.

2.2.1. Acunetix

Acunetix és una aplicació comercial enfocada a la seguretat web a escala interna. Ofereix el seu producte com a complement a les auditories internes de seguretat que es facin sobre el codi. Fa escàners automatitzats amb possibilitat d'execució tant en interfície gràfica com en línia d'ordres. Solament és disponible per a plataformes Windows.

Malgrat que és un programa comercial ofereix per a baixar una versió gratuïta que es limita a la recerca de fallades d'XSS en el domini que especifiquem. A més, permet fer un escàner complet sobre tres pàgines vulnerables, programades amb:

- PHP,
- ASP,
- i ASP.Net.

Les característiques principals d'Acunetix són aquestes:

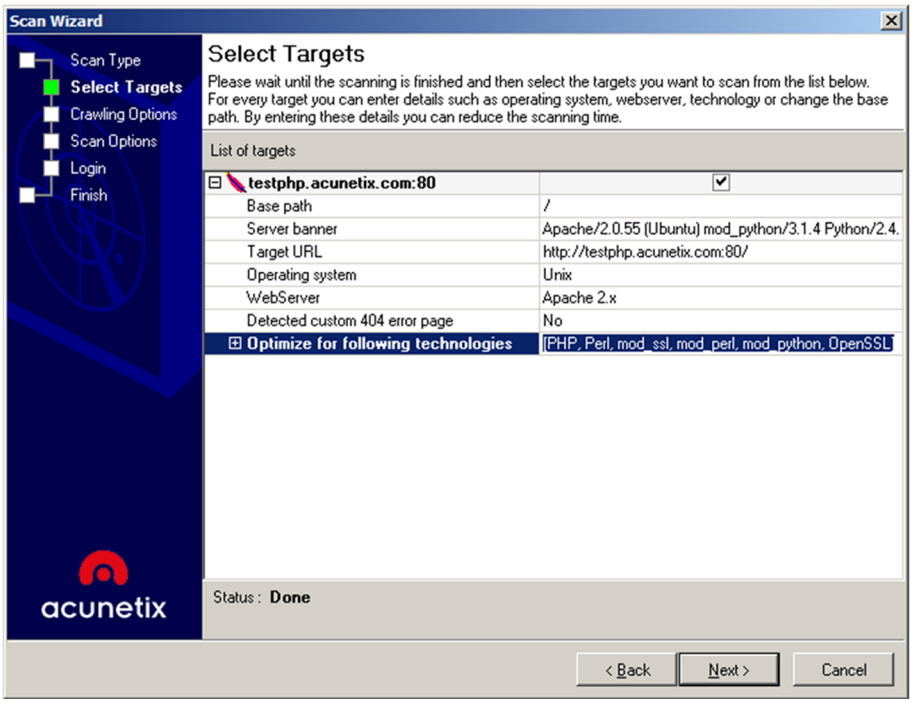
- Té un analitzador JavaScript, que permet auditar Ajax i aplicacions amb Web 2.0.
- Utilitza les tècniques més avançades d'SQL Injection i *cross site scripting*.
- Utilitza tecnologia AcuSensor.
- Analitza llocs web incloent-hi contingut flaix, SOAP i Ajax.

- Fa un escàner de ports contra el servidor web i busca vulnerabilitats en els serveis d'aquests ports.
- Detecta el llenguatge de programació de l'aplicació.
- Proporciona extensos informes de l'estat de seguretat.

Per a analitzar les funcionalitats del programa farem un escàner sobre un dels dominis vulnerables, en què se'ns permeten provar totes les característiques del programa. En aquest cas ho farem sobre el domini que executa l'aplicació PHP.

El primer pas és establir el domini a escàner. Amb aquesta dada, Acunetix fa una primera anàlisi del domini, i intenta detectar les característiques intrínseques del servidor: tecnologia, servidor HTTP, capçaleres, etc. Aquestes dades les usa per a configurar automàticament les regles més òptimes per a l'anàlisi.

Figura 4. Auxiliar d'Acunetix



Des d'aquest auxiliar podrem especificar alguns paràmetres per a millorar l'eficiència de la nostra anàlisi i també per a fer-la menys intrusiva. Algunes de les opcions que podem configurar són aquestes:

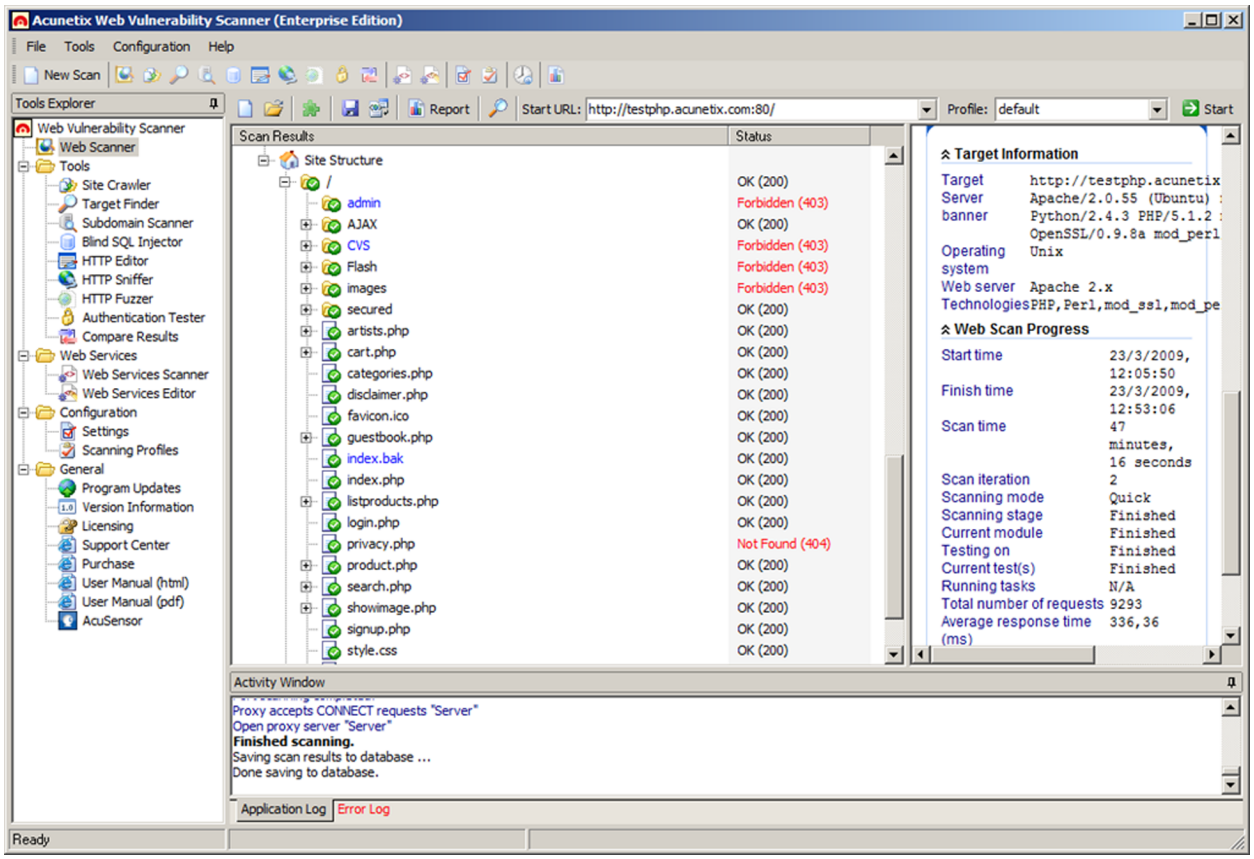
- Enviar dades mitjançant els formularis que hi ha a la pàgina web.
- Intentar extreure la llista de directoris.
- Ignorar majúscules en els noms dels fitxers.
- Processar els fitxers robots.txt i sitemap.xml.
- Manipular les capçaleres HTTP.
- Habilitar l'escàner de ports.
- Establir usuari i contrasenya per a l'autenticació HTTP.
- Habilitar l'anàlisi del codi JavaScript mitjançant l'execució d'aquest codi.

Com es veu, obté molts detalls que s'aconsegueixen per mitjà de moltes peticions i sol·licituds al servidor web, de manera que llançar aquesta eina contra el nostre servidor s'hauria de limitar a hores de baix trànsit per a no interferir-ne el funcionament normal.

Després de completar l'auxiliar se'ns presentarà una finestra d'anàlisi en què anirem comprovant en temps real el que analitza i descobreix el programa.

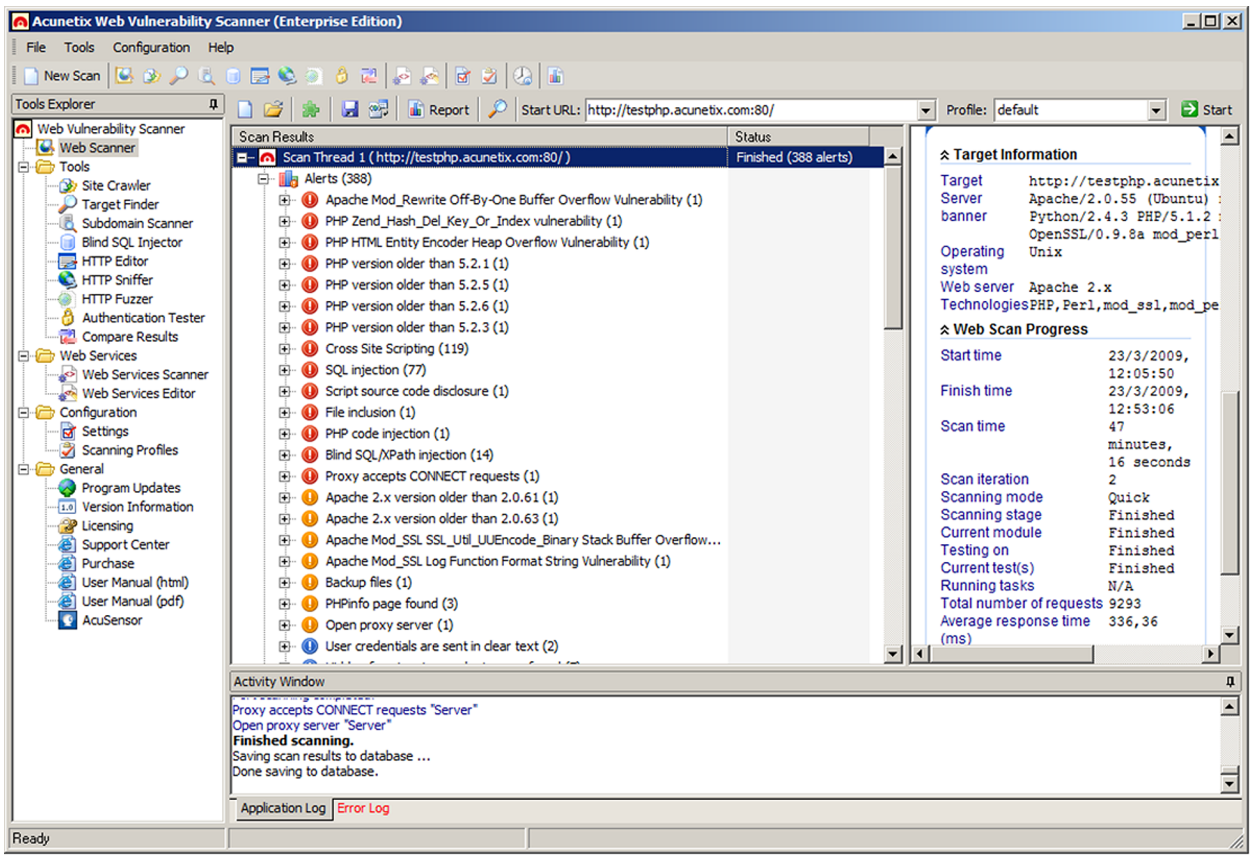
Una de les característiques més interessants d'Acunetix és la possibilitat de veure un arbre dels fitxers que hi ha en el servidor i que podrem recórrer com si fos un explorador de fitxers del nostre sistema operatiu.

Figura 5. Arbre de fitxers d'un lloc mitjançant Acunetix



Sobre cadascuna de les pàgines detectades el programa fa una sèrie de comprovacions per a determinar si l'aplicació serà vulnerable a qualsevol de les tècniques esmentades abans. Aquestes anàlisis donaran lloc a un informe detallat del que ha passat.

Figura 6. Informe final d'Acunetix



Aquesta anàlisi, malgrat que és completa i exhaustiva, és molt intrusiva. Un atacant real no llançaria mai una eina d'aquest tipus per a localitzar les vulnerabilitats del nostre lloc web, de manera que la podem usar com a cas real per a auditar les nostres contramesures enfront d'atacs reals.

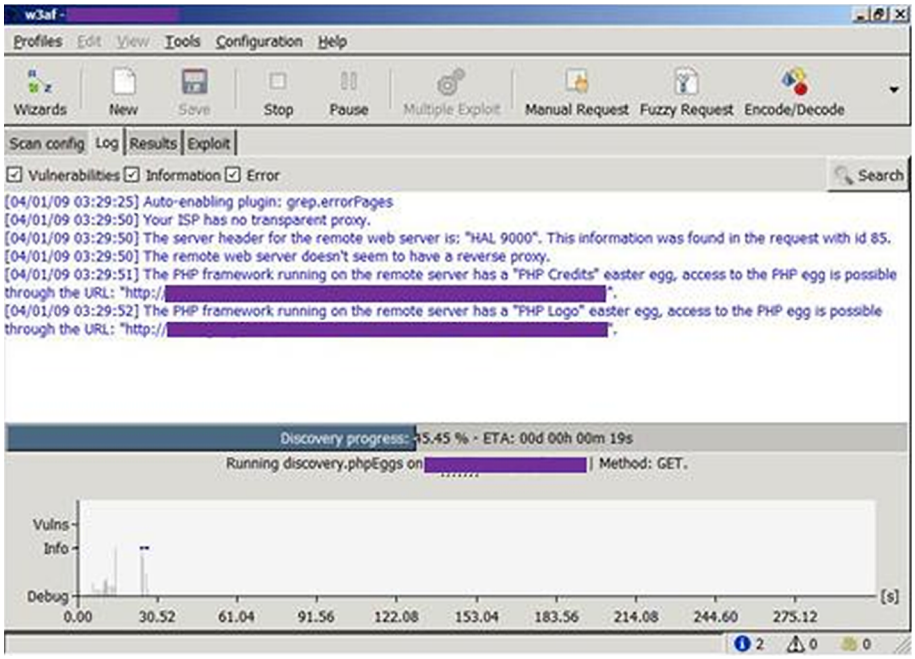
2.2.2. W3af

W3af és una eina d'auditoria web de font pública (*open source*) disponible tant en Windows com en Linux (nota al marge). Disposa d'una interfície gràfica d'usuari i en línia d'ordres. L'objectiu de W3af és crear un entorn de treball (*framework*) per a trobar i executar vulnerabilitats en aplicacions web.

La característica principal de W3af és que el seu sistema d'auditoria està basat completament en connectors (*plug-ins*)[4] escrits en Python, de manera que aconsegueix crear un entorn de treball fàcil d'escalar i una comunitat d'usuaris que contribueixen amb la programació de nous connectors davant les noves fallades de seguretat web que poden anar apareixent.

[4]Tots aquests connectors estan documentats en el web oficial de W3af.

Figura 7. Visor de registres de W3af durant una auditoria web

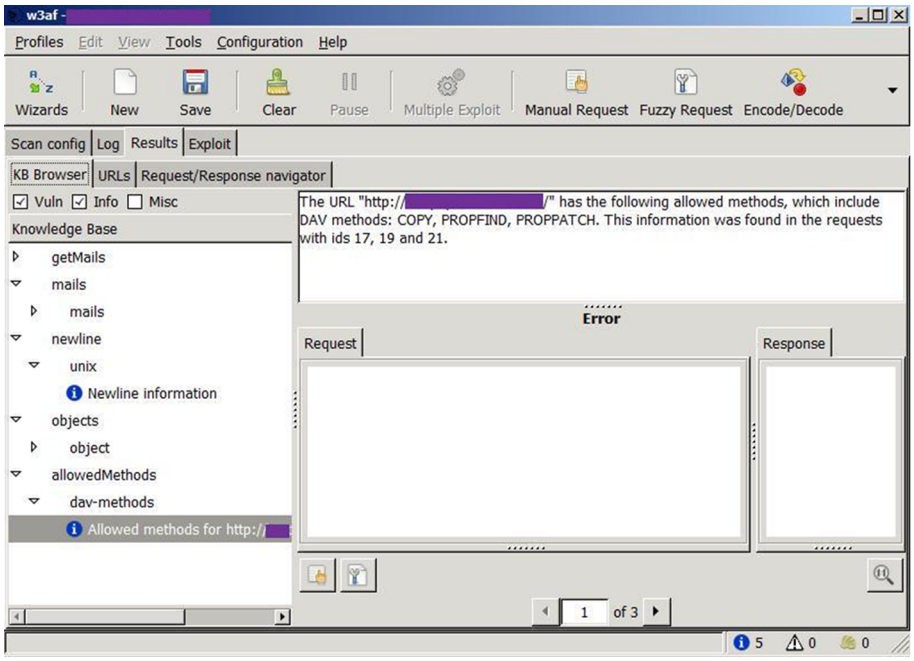


Entre les vulnerabilitats que detecten i exploten els connectors disponibles hi ha les següents:

- CSRF.
- XPath Injection.
- WebDAV.
- Desbordaments de memòria intermèdia (*buffer overflows*).
- Extensions de *FrontPage*.
- SQL Injection.
- XSS.
- LDAP Injection.
- *Remote file inclusion*.

Vegem com, malgrat que és una eina lliure, és tan completa com qualsevol altra.

Figura 8. Informe generat per W3af



Malgrat haver comentat solament dos dels escàners de vulnerabilitats web més coneguts, n'hi ha molts d'altres, cadascun amb característiques pròpies. No obstant això, l'ús indiscriminat d'aquestes eines pot donar lloc a problemes relacionats amb l'automatització de les tasques. Sempre val més que una persona executi aquestes proves perquè les seves accions són menys previsibles i més exactes i estan més ben adaptades a la realitat d'una intrusió.

3. Fortificació: serveis, permisos i contrasenyes

Tant una aplicació web com una base de dades són peces de programari d'una complexitat elevada que proporcionen una sèrie de serveis. Els **serveis** són:

- **Públics** i hi pot accedir directament un altre programari.

- **Privats** per a fer determinades tasques internes, o que interactuïn solament amb el sistema operatiu que els allotja.

En tot cas, en instal·lar qualsevol programari se solen activar una sèrie de serveis per defecte que s'encarreguen de proporcionar la major part de funcionalitat necessària. Tanmateix, això pot comportar un problema per a la seguretat. Un dels principis que s'han d'aplicar és el d'economia, això és, utilitzar el nombre mínim de recursos per a oferir el servei estrictament necessari. Per tant, això es tradueix a utilitzar solament el mínim de serveis que proporcionin la funcionalitat volguda. Això es deu al fet que cada servei addicional és susceptible de patir algun tipus de vulnerabilitat: com més serveis s'ofereixen, més creix la finestra d'atac. No obstant això, no sempre és senzill determinar quins serveis són els necessaris.

A més de l'habilitació i la deshabilitació de serveis, s'han de protegir degudament els que es decideixen deixar actius. Alguns serveis han estat cèlebres pel fet de presentar vulnerabilitats que han permès l'accés a la base de dades, i d'altres incorporen contrasenyes per defecte (o simplement no incorporen cap contrasenya), de manera que s'ha de fer un treball adequat de configuració abans de publicar el servei en entorns de producció.

A continuació mostrem algunes de les recomanacions de seguretat referents a les **contrasenyes dels serveis**:

- Contrasenyes per defecte: s'han de canviar totes aquestes contrasenyes o s'han de deshabilitar els serveis que les usen.

- Serveis sense contrasenya: s'ha d'evitar que hi hagi aquest tipus de serveis o s'han de restringir adequadament mitjançant eines addicionals.

- Serveis poc usats: és important discriminar quins serveis s'usen realment a fi de deshabilitar els que no són necessaris.

Una vegada s'ha trobat el conjunt de serveis que proporciona la funcionalitat necessària per a l'entorn de treball, cal concentrar-se en els permisos que tenen aquests serveis. La relació de la base de dades i el sistema operatiu es basa en un

usuari que executa els serveis mitjançant cert nombre de programes. L'usuari que executa aquests programes disposa d'una sèrie de privilegis segons el grup a què pertany i els privilegis addicionals de què disposa.

En cas que un atacant aconsegueixi el control del que s'encarrega de l'execució d'un servei, no aconseguirà mai fer cap acció que no pugui fer l'usuari amb el qual s'executa aquest servei. Per això, s'han d'intentar mantenir aquests permisos al mínim per evitar que, en cas que un atacant tingui èxit, aconsegueixi el control total de l'equip. Hi ha un parell d'excepcions:

- En alguns casos s'ha d'executar el servei amb l'usuari amb màxims privilegis en el sistema (sobretot en entorns Windows).

- Encara que un atacant arribi a controlar un servei corrent amb un usuari que no tingui tots els privilegis, sempre hi ha la possibilitat que aconsegueixi una escalada de privilegis, si bé això afegeix dificultat a l'atac.

Una vegada definits els permisos amb què interactua la base de dades o l'aplicació web amb el sistema operatiu mitjançant els serveis, cal centrar-se en els permisos que tindran els diferents usuaris dins de la base de dades o aplicació. Els principis bàsics que hem vist en l'apartat anterior també es poden aplicar en aquest, com expliquem a continuació.

Els diferents usuaris tenen diferents nivells de permisos segons els rols assignats. És una cosa semblant als usuaris d'un sistema operatiu, en què els permisos els donen els grups a què pertanyen. A part del rol assignat, normalment es poden atorgar i revocar permisos individuals als usuaris, de manera que els privilegis d'un usuari no han pas de coincidir exactament amb els dels grups a què pertany.

Per tant, configurar els **permisos** per als usuaris és semblant a la manera com es fa el mateix procés en un sistema operatiu: intentar restringir a cada usuari perquè solament tingui el mínim conjunt necessari de permisos per a fer totes les tasques habituals, però res més.

El procés consisteix en dos passos:

- **Autenticació**: l'usuari s'identifica en el sistema mitjançant el mecanisme corresponent (habitualment, el parell usuari-contrasenya).

- **Autorització**: segons l'usuari, s'atorguen una sèrie de permisos associats al seu perfil sobre els objectes que hi ha en l'aplicació o base de dades i les accions que hi pot fer.

És recomanable no reaprofitar usuaris mai, encara que executin serveis que necessitin el mateix grup de privilegis, ja que és important disposar d'usuaris individuals, per exemple, per a seguir el rastre d'un possible problema o intrusió mitjançant el registre de les accions en els registres (*logs*) del sistema. També és una bona pràctica emprar un sistema de nomenclatura d'usuaris estàndard que ajudi a facilitar la tasca d'administració, donant noms descriptius per als usuaris de serveis i utilitzar un mateix patró per als usuaris "particulars".

Un dels errors més típics és usar un usuari amb un gran conjunt de privilegis per a la interacció entre la base de dades i una aplicació web, a vegades l'usuari administrador. Normalment, les aplicacions necessiten funcionalitat avançada i un nivell alt de permisos, de manera que es pot donar el cas de proporcionar un usuari administrador per a simplificar la creació d'un rol adequat. Aquest tipus de decisions són les que poden comprometre tota la base de dades en cas d'una intrusió, fins i tot el sistema operatiu en cas que aquest usuari pugui executar procediments que hi interactuïn.

Les escalades de privilegis fan referència directa a aquests rols i usuaris administradors. En cas que un atacant (o usuari legítim) disposi d'un usuari amb un conjunt de permisos limitats i vulgui fer certes accions que no li són permeses, pot intentar escalar privilegis en el sistema. Si ho aconsegueix, accedeix a un altre conjunt superior de permisos o, a tot estirar, a un usuari amb permisos d'administració, i aconsegueix així un control potencial de tot el sistema.

És important revisar la configuració per defecte de la base de dades. A vegades hi ha una sèrie de rols predefinits amb una sèrie de permisos que no han pas de ser els que volem usar per als nostres usuaris.

Una vegada definits els rols i assignats els usuaris per a cadascun d'aquests rols, és important definir una política[5]. Per exemple, es pot aplicar una política pel que fa a actualitzacions del sistema o pel que fa a emergències. Una política no s'ha pas de restringir solament a les accions que ha de fer de manera automàtica la base de dades, sinó que val més entendre-les des d'un context més ampli. De fet, poden ser un conjunt d'accions que ha de fer el personal d'administració en cas d'una emergència i estar escrites en un foli. No obstant això, és important definir-les, perquè impliquen pensar sobre un problema i estableixen un protocol d'actuació. En situacions en què és difícil pensar amb claredat, sempre ajuda una guia feta a consciència i amb tranquil·litat.

(5)Una política són una sèrie d'accions i restriccions que es defineixen per a la base de dades i que es poden aplicar en diferents nivells.

En una política es poden afegir restriccions i controls que afectin en bona mesura els usuaris; un dels exemples més habituals és en relació amb les contrasenyes. Es tracta de la porta d'entrada al sistema de manera legítima, i per tant una política adequada pot evitar molts problemes. En general, el fet de no disposar d'una política de contrasenyes adequada comporta dependre dels usuaris pel que fa a la complexitat d'aquestes contrasenyes, la qual cosa propicia que hi hagi atacs de força bruta que puguin tenir èxit i entrar de manera legítima.

Finalment, cal fer referència a l'emmagatzematge d'aquestes contrasenyes. Varia molt segons la base de dades que s'utilitzi, de la versió, i del mètode d'autenticació dins de la mateixa base de dades. Com s'ha vist en les arquitectures, algunes bases de dades permeten sistemes d'autenticació mixtos que interactuen amb el sistema operatiu, i d'altres són completament natives, cosa que fa que la manera d'emmagatzematge depengui del sistema operatiu mateix.

En algunes de les primeres versions de MySQL, la contrasenya s'emmagatzemava completament en clar, de manera que si algú aconseguia l'accés a la taula en què s'emmagatzemen els usuaris, hi podia consultar tranquil·lament el contingut. En altres casos, s'utilitzen sistemes d'encriptació més o menys segurs. En cas d'un sistema d'autenticació mixt, l'emmagatzematge de la contrasenya sol dependre del sistema operatiu. Per exemple, en versions antigues d'SQL Server s'emmagatzema en una entrada de registre. En les primeres versions s'emmagatzemava completament en clar, però després es va optar per un mètode d'encriptació, encara que era fàcilment reversible, de manera que tampoc no era gaire segur.

És important conèixer bé el sistema d'autenticació per a evitar deixar punts febles en el procés, i també establir totes les mesures i les polítiques necessàries, i donar als usuaris solament els permisos estrictament necessaris.

4. Desenvolupament segur

Com hem vist, el principal punt d'atac per a un programari que comuniqui amb la base de dades són *tots* els paràmetres d'entrada. Per això s'han de filtrar sempre tots.

Ara coneixerem les tècniques bàsiques que hi ha per a evitar els problemes associats amb aquest aspecte:

1) Llistes negres (o filtra el que saps que és dolent)

Aquesta tècnica es basa a filtrar totes les entrades que es corresponen amb un patró conegut d'atac o vulnerabilitat. D'aquesta manera, se sol tenir una llista de patrons que pot consistir en literals o en expressions regulars reconegudes com a malicioses, i quan es detecta en qualsevol entrada es fa el filtratge.

Hi ha com a mínim dos problemes associats amb aquesta tècnica, cosa que la fa poc recomanable:

- El primer és que una vulnerabilitat pot ser explotada de diverses maneres, i per tant, tret que la llista negra sigui molt exhaustiva, pot molt ben ser que es pugui fer un atac reeixit malgrat aquesta llista.

- El segon està relacionat amb la constant evolució dels atacs i de les diverses tècniques que sorgeixen per a explotar-los, de manera que les llistes poden quedar obsoletes de seguida. Això és un problema greu per la gran quantitat de dificultats que implica haver de gestionar llistes d'aquest tipus, especialment si són en el codi font de l'aplicació.

2) Llistes blanques (o deixar passar el que no és perillós)

Aquest cas és l'oposat a l'anterior. Es deixen passar les entrades que coincideixen amb una llista blanca, sia una llista de literals, d'expressions regulars o de requisits que ha de complir l'entrada, sia una entrada que tingui un nombre de caràcters màxim o que estigui composta solament de caràcters numèrics.

Quan es pot aplicar aquesta tècnica, es possiblement la més efectiva perquè els desenvolupadors solament permeten el tractament d'entrades que se sap que no tenen cap problema. No obstant això, el problema està en el fet que com que aquesta tècnica és tan restrictiva hi ha moltes vegades en què no es pot aplicar, com ara entrades de text lliure o noms en què cal acceptar cometes simples, per exemple, que són caràcters usats sovint per a injeccions SQL.

D'aquesta manera, malgrat que és una tècnica efectiva, normalment no es pot aplicar sempre o és difícil de configurar per la complexitat de trobar llistes blanques adequades per a tota la casuística d'entrada.

3) Neteja de paràmetres

Aquesta aproximació considera que no es pot fer una tipificació completa de l'entrada per a aplicar una llista blanca i que, per tant, no es pot confiar en l'entrada. No obstant això, es fa un tractament a aquests paràmetres per a evitar que siguin perillosos, malgrat que no s'hi pot confiar, i per a evitar-ne el filtratge.

En aquest cas, es tracta d'evitar caràcters o literals considerats perillosos, com per exemple les cometes simples per a evitar casos d'injecció d'SQL, o literals com ara "`<script>`" per a evitar atacs de *cross site scripting*. Es pot aplicar el mateix amb literals que s'utilitzen en sentències SQL, com "`unión`" o "`select`". L'eliminació d'aquests literals també es pot canviar per substitució.

Per exemple, l'entrada següent:

```
<scri<script>pt>
```

quedaria d'aquesta manera:

```
<script>
```

en cas d'un tractament d'eliminació no iteratiu, de manera que aquest tipus de tècniques també tenen problemes si no es fan adequadament. A part de la iterativitat, s'ha de tenir en compte que les expressions regulars han de ser flexibles per a evitar tractaments inadequats en casos com entrades amb majúscules, minúscules i caràcters en blanc intercalats. És una cosa que pot semblar trivial, però no ho és, i moltes vegades hi ha filtratges aparentment bons que fallen davant algun cas concret, de manera que resulten inútils a la pràctica.

4) Tractament d'excepcions adequat

Hi ha molts errors provocats per un tractament inadequat de paràmetres d'entrada que són inesperats per la lògica de l'aplicació. Durant molt temps, aquest tipus de problemes en el tractament incorrecte de cadenes llargues de caràcters ha provocat els errors de desbordament de memòria intermèdia en programes compilats, encara que actualment ja no és una cosa tan freqüent.

Tanmateix, això posa en relleu la importància de considerar totes les opcions, i en cas de produir-se una condició inesperada, tenir un tractament adequat d'excepcions que eviti que l'aplicació es corrompi o que es mostri un missatge d'error que proporcioni pistes a un atacant o es pugui aprofitar per a obtenir dades internes de la base de dades.

Finalment, respecte a tot el tema del filtratge de paràmetres d'entrada, s'ha de considerar que hi ha tècniques d'ofuscació per a evitar ser detectats per aquest tipus de solucions. S'ha de tenir en compte que hi ha diferents possibilitats per a evitar reconèixer entrades malicioses, com ara l'ús de diferents tipus de codificacions que reconeix el servidor d'aplicacions o la base de dades, com UTF-8 o *URL encoding*, però que no cal que es tinguin en compte a l'hora de fer el filtratge.

D'altra banda, es poden utilitzar les funcions de la base de dades mateixa per a convertir els caràcters a partir del codi ASCII en literals que entén i interpreta la base de dades.

Hi ha diverses maneres d'intentar enganyar els filtres d'entrada. A continuació en mostrem uns quants exemples, cadascun dels quals explota una tècnica diferent per a evitar ser filtrats:

- Evitar caràcters bloquejats: encara que l'aplicació bloquegi certs caràcters, sempre es pot intentar treballar sense aquests caràcters. Per exemple, el filtratge de les cometes simples no afecta en el cas de camps numèrics. En cas de filtratge de punts i comes per a la separació de sentències quan s'intenta fer una injecció de diverses d'aquestes sentències no és un problema, ja que la base de dades interpreta correctament totes les sentències si són correctes sintàcticament, encara que no estiguin separades per punt i coma. Finalment, en el cas dels símbols de comentaris, pot ser que no siguin necessaris si s'usa una sentència equivalent que no trenqui l'estructura sintàctica de la sentència original.

- En el cas de les codificacions, alguns filtres no les tenen en compte, de manera que en lloc d'intentar la injecció amb el literal SELECT, es pot fer amb la cadena *URL encoding* %53%45%4c%45%43%54.

- Inserir comentaris dins de les sentències injectades. En fer-ho, un analitzador (*parser*) incorrecte no detectarà les paraules clau.

```
Select/*xx*/password/*xx*/from/*xx*/users;
```

En alguns casos, la base de dades permet inserir els comentaris fins i tot dins de les mateixes paraules clau, de manera que encara fa més difícil crear un filtratge adequat.

Ús de funcions de la base de dades per a ofuscar sentències. Algunes funcions que creen sentències a partir de codificacions, com ara codi ASCII, dificulten encara més la tasca de filtrar adequadament l'entrada. Per exemple, es podria usar `chr(97) ||chr(100) || chr(109) || chr(105) || chr(110)` en lloc del literal "`admin`" en una sentència, cosa que la base de dades interpretaria perfectament.

Ús d'execució dinàmica. Algunes bases de dades permeten l'execució de sentències passades com a paràmetre directament, interpretant la sentència paràmetre de diverses maneres, com a trossos de text que després reassembla o converteix en codificació hexadecimal. Vegem-ne un exemple:

```
exec ('sel' + 'ect * f' + 'rom users');
```

Finalment, s'ha de tenir en compte que tots aquests filtres s'han de fer sempre en la part del servidor, i mai en la del client mateix, que és qui pot fer l'intent d'intrusió. Encara que filtrar els paràmetres d'entrada en la mateixa aplicació del client (o en el navegador, si és una aplicació web) sembli una bona idea pel que fa a eficiència (el filtratge es fa en el client mateix i es pot estalviar un processament innecessari en el servidor), aquesta solució és completament inefectiva. Qualsevol filtratge en el client es pot evitar de diverses maneres, i com a exemple podem fer servir la més evident: el trànsit de sortida de la seva màquina, després de qualsevol filtratge que pugui fer qualsevol aplicació, el pot interceptar l'usuari mateix i el pot modificar com vulgui, de manera que fa completament inútil el filtratge que ha dut a terme la nostra aplicació.

L'única manera de poder fer aquest procés és mitjançant l'ús de canals segurs de comunicació que encriptessin adequadament el trànsit de sortida i establissin una relació de confiança entre client i servidor. Així i tot, s'ha de tenir en compte que per més segura que sembli una solució d'aquest estil sempre és susceptible de ser piratejada (per exemple, es pot fer enginyeria inversa en l'aplicació i evitar qualsevol filtratge), de manera que el filtratge en el servidor sempre és necessari.

Per acabar, comentarem alguns dels problemes més importants que té el desenvolupament segur. Hi ha una gran quantitat de programari que es desenvolupa per a petites aplicacions de manera ràpida i poc curosa, i sembra així la llavor de la desgràcia. Les presses, els pressupostos ajustats que no tenen en compte la seguretat, l'ús d'entorns de treball de desenvolupament aparentment miraculosos però que no tenen en compte aspectes bàsics per a evitar vulnerabilitats típiques, la convivència de programari heretat que ningú no sap com està fet ni tampoc s'atreveixen a tocar-lo… Tots aquests escenaris són molt freqüents actualment, i comporten possiblement el pitjor problema pel que fa a desenvolupament insegur.

4.1. Auditoria de codi font

Aquesta tècnica consisteix, en la versió més bàsica, a fer cerques dins del mateix codi per a localitzar patrons de fragments de codi que siguin potencialment vulnerables a problemes coneguts. El recomanable és que l'auditoria no la faci la mateixa persona que ha escrit el codi, sinó un equip diferent, per a assegurar la imparcialitat.

Podem pensar que fer una auditoria d'aquest estil amb un codi de mida gran pot ser una odissea, però no obstant això hi ha eines per a ajudar en aquesta tasca. En aquest punt, s'ha de destacar que auditar codi no és solament fer una cerca de text, encara que algunes eines es limiten a això, sinó que és una mica més complex. S'ha de seguir l'execució del programa i la contextualització del codi en certs punts potencialment vulnerables, com ara el filtratge de paràmetres d'entrada i la interacció amb la base de dades. La solució més adoptada consisteix en una barreja entre eines automàtiques i validació manual, com acostuma a passar en aquests casos.

S'ha de destacar que una programació ordenada i modular ajuda en gran manera a netejar el codi i a estructurar-lo; per tant, fa molt més senzilla l'auditoria. Per això, el fet de seguir unes bones pràctiques de programació repercuteix en la seguretat del sistema. Un codi que només l'entén el programador que l'ha escrit és difícil d'auditar per tercers.

Vegem un exemple de què es podria buscar en el codi font per intentar evitar una injecció de codi SQL. Es tracta de buscar els punts potencialment vulnerables a aquest problema i assegurar que estan degudament protegits, com per exemple a partir d'un filtratge adequat dels paràmetres d'entrada. Una de les construccions que poden donar lloc a aquest problema són les sentències creades dinàmicament juntament amb paràmetres d'entrada de l'usuari. Per exemple:

```
StringBuilder consulta = newStringBuilder ("Select id, nombre, descripcion, precio
From libros Where" + condicion);
```

```
consulta.Append ("id=");
consulta.Append (Request.QueryString("ID").toString());
```

En aquest cas, la construcció seria vulnerable a una injecció de codi SQL en cas que no es faci un filtratge apropiat del paràmetre ID en cap punt de l'aplicació. Una manera senzilla de buscar parts de codi que tinguin aquest problema potencial és buscar cadenes que formen part de la construcció de sentències SQL,

com "Select", "Insert" o "Delete". També s'ha de tenir en compte el cas en què aquests literals s'assignessin a constants, de manera que aquesta mateixa cerca s'hauria d'aplicar als literals esmentats.

Més que no pas la recerca de problemes concrets, moltes vegades es tracta de buscar fragments de codi potencialment perillosos per la funció que tenen. Bàsicament, es tracta d'intentar buscar la interacció amb el sistema de fitxers, amb el sistema operatiu i amb la base de dades. Ens centrarem en la base de dades.

La recerca dels punts d'interacció varia segons la tecnologia usada. Per exemple, en Java s'utilitzen normalment les següents API per a executar consultes contra la base de dades, de manera que seria interessant buscar els punts en què s'utilitzen per a estudiar que no hi hagi problemes:

```
java.sql.Connection.createStatement
java.sql.Statement.execute
java.sql.Statement.executeQuery
```

Hi ha una alternativa que representa una API més robusta a problemes de seguretat, com les injeccions:

```
java.sql.Connection.prepareStatement
java.sql.PreparedStatement.setString
java.sql.PreparedStatement.execute
java.sql.PreparedStatement.executeQuery
```

En l'exemple següent, es veu que si s'usen degudament s'evita una injecció SQL, mentre que amb la primera API el codi seria vulnerable:

```
String usuario = "admin' or 1=1-";
String pass = "contraseña";
```

```
Statement s = con.prepareStatement("Select * from usuarios where usuario =? and password = ?");
s.setString(1,usuario);
s.setString(2.pass);
s.executeQuery();
```

En quedaria el codi següent, que seria el que s'executaria contra la base de dades:

```
Select * from usuarios where usuario = 'admin'' or 1=1-- and password = 'contraseña';
```

En aquest cas, no s'aconseguiria l'efecte volgut per l'atacant, de manera que l'API és molt millor pel que fa a seguretat. No obstant això, és molt recomanable fer un bon filtratge previ de paràmetres, tal com hem explicat abans.

Continuant el repàs de tecnologies, en ASP.NET les API següents són les més comunes per a interaccionar amb la base de dades:

```
System.Data.SqlClient.SqlCommand
System.Data.SqlClient.SqlDataAdapter
System.Data.Oledb.OleDbCommand
System.Data.Odbc.OdbcCommand
System.Data.SqlServerCe.SqlCeCommand
```

Igual que en el cas anterior, s'ha de treballar amb els mètodes recomanats en aquestes API per a evitar injeccions. Per a aconseguir un efecte igual a l'exemple anterior en Java, s'han d'utilitzar els mètodes per a afegir paràmetres a sentències disponibles en totes les API anteriors. Per exemple:

```
OdbcCommand c = new OdbcCommand ("Select * from users where usuario = @user and password
= @pass", connection);
c.Parameters.Add (new OdbcParameter('@user',OdbcType.Text).Value=user);
c.Parameters.Add (new OdbcParameter('@pass',OdbcType.Text).Value=pass);
c.ExecuteNonQuery();
```

Finalment, per acabar el repàs d'alguns dels llenguatges més comuns, vegem les API més comunes per a la comunicació amb la base de dades de PHP:

```
mysql_query
mssql_query
pg_query
```

Els mètodes següents són els més recomanables per a evitar les injeccions, i permeten la inserció de paràmetres de manera controlada, com hem vist en els exemples anteriors:

```
mysqli->prepare
stmt->prepare
stmt->bind_param
stmt->execute
odbc_prepare
```

Per exemple, igual que en els casos anteriors, un codi que seria equivalent als casos no vulnerables a injecció:

```
$sql = $db_connection->prepare ("Select * from users where user = ? and password = ?");
$sql->bind_param("ss",$user,$pass);
$sql->execute();
```

En PHP hi ha una directiva relacionada amb la seguretat pel que fa a filtratge de paràmetres que es defineix a l'entorn del sistema. Es tracta de `magic_quotes_gpc`, que, si està activada, qualsevol cometa simple, cometes

dobles, barra invertida (\) i caràcter nul "s'escapen" utilitzant una barra invertida. Hi ha una directiva semblant anomenada `magic_quotes_sybase`, en què el comportament és el mateix amb la diferència que la "fuita" es fa mitjançant una cometa simple. Encara que aquesta funcionalitat no evita les injeccions del tot (per exemple, amb els camps de tipus numèric), comporten una dificultat afegida per a l'atacant per a explotar el sistema. S'ha de considerar que pot ser que el comportament afegit per aquesta directiva no sigui el volgut per a l'aplicació que l'usa. A més, el fet de confiar en una directiva de l'entorn en lloc d'assegurar el filtratge en l'aplicació mateixa no és el més recomanable. S'ha d'intentar fer el filtratge en tots els mòduls que configuren el sistema, de manera que com més mesures millor, però no s'ha de confiar mai en un filtratge per a no posar prou atenció en un altre.

La funcionalitat de `magic_quotes` ha estat una de les més ampliades i utilitzades en PHP 5, de manera que actualment n'hi ha en milions de desenvolupaments, i per aquest motiu en parlem en aquest curs. No obstant això, en la nova versió de PHP ja no hi és.

4.2. Eines de filtratge: Web Application Firewalls

Hi ha eines dissenyades per a eliminar alguns dels problemes que hem explicat més amunt, com el filtratge de paràmetres d'entrada. L'ús d'aquestes eines *no* substitueix crear un codi segur en cap cas. No obstant això, poden servir per a afegir una capa extra de seguretat en cas que no puguem fer un desenvolupament propi o utilitzem programari de tercers.

Els **Web Application Firewalls** són les aplicacions que s'instal·len a escala de servidor per a controlar les peticions malintencionades que podrien donar lloc a un possible atac. Això es fa mitjançant unes regles que es configuren i que permeten detectar possibles atacs fins i tot abans que es produeixin. A més, aquests sistemes actuen en conjunció amb els servidors web, de manera que permeten el bloqueig de la petició maliciosa i eviten que l'atac s'arribi a produir.

Els Web Application Firewall (WAF) són un tipus de sistemes de detecció d'intrusos[6] a escala d'aplicació. Hi ha altres tipus de sistemes de detecció d'intrusos, com per exemple a escala de xarxa, que analitzen els paquets a la recerca de patrons coneguts. No obstant això, encara que puguin ser més efectius contra atacs generals, es mostren menys eficients contra atacs més selectius com ara els atacs web. A més, com que són específics per a una aplicació, actuen més bé i són més eficients.

(6) Les sigles en anglès són IDS.

Actualment hi ha moltes solucions per a protegir els nostres servidors, però aquí en veurem dues:

- Mod_Security, un mòdul per a Apache que ens permet aturar peticions malicioses i falsificar algunes de les dades que generi el nostre servidor per a enganyar un possible atacant.

- Request Filtering, inclòs a Internet Information Services 7, ens ofereix una manera gràfica de crear regles de filtratge.

4.2.1. Mod_Security

Mod_Security és un mòdul per a Apache. Apache permet l'expansió de les seves funcionalitats mitjançant mòduls, que solen estar programats en C o C++ i es poden trobar fàcilment a Internet.

Per a configurar el servidor hem de baixar el mòdul adequat per al nostre sistema operatiu o directament compilar el codi font. En tot cas hem d'obtenir un fitxer `mod_security2.so` que hem de col·locar en una carpeta `mod_security2` dins de la carpeta de mòduls de l'Apache.

Amb aquesta acció encara no tindrem el mòdul en funcionament; hem de fer una petita configuració, que passa per establir una sèrie de línies dins del fitxer `httpd.conf` de la carpeta `conf`.

Les línies que hem d'afegir són aquestes:

```
LoadModule unique_id_module modules/mod_unique_id.so
LoadModule security2_module modules/mod_security2/mod_security2.so
Include conf/rules/*.conf
```

L'última línia, la que fa referència als fitxers amb extensió `conf` dins de la carpeta `rules`, és la que s'encarrega de carregar totes les regles de configuració que baixen al costat de `mod_security`. Aquestes regles contenen la configuració bàsica del mòdul i també diverses regles específiques per a diferents tipus d'atacs.

La configuració bàsica passa per editar el fitxer `modsecurity_crs_10_config.conf`, que conté l'estructura base per a començar a funcionar amb aquest mòdul. Hem de descomentar la línia següent perquè el mòdul comenci a identificar les peticions que detectem com a atacs possibles:

```
SecDefaultAction "phase:2,log,deny,status:403,t:lowercase,t:replaceNulls, t:compressWhitespace"
```

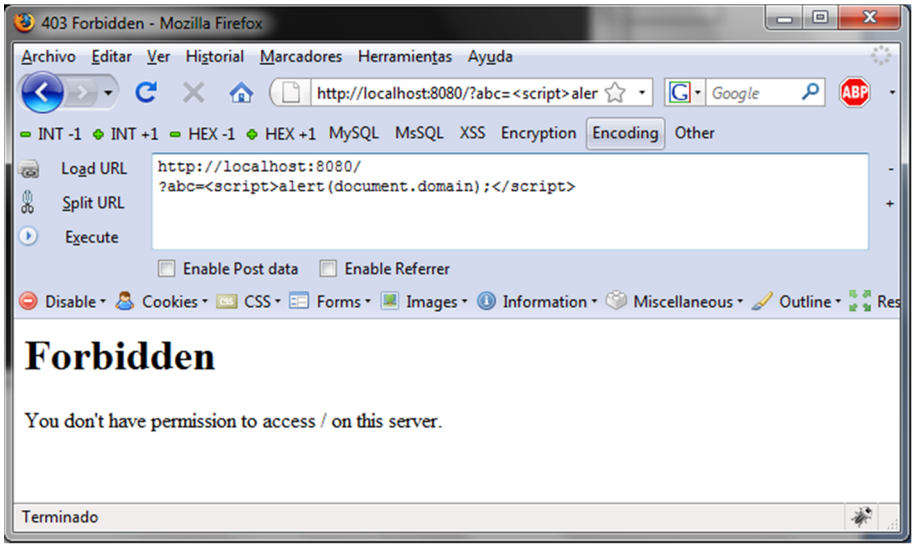
Aquesta línia el que fa bàsicament és activar el `mod_security` en funció `log` i denega (*deny*) totes les peticions que compleixin una regla de les que hi ha activades. A més, retorna un codi 403 de recurs prohibit.

Si hem seguit els passos correctament i hem reiniciat el servidor Apache (això s'ha de fer perquè el fitxer de configuració es torni a llegir i es carregui a la memòria), podrem intentar produir un atac controlat. Encara que ara mateix el nostre servidor no disposa de cap aplicació web, podem generar una petició que conté un atac XSS.

```
http://localhost/?abc=<script>alert(document.domain);</script>
```

Aquesta simple petició web ens retornarà un error 403 i generarà una entrada en el fitxer de `log`, per defecte el fitxer `log/mod_security2.log`.

Figura 9. Pàgina d'error produïda per mod_security



En la figura anterior veiem una petició amb XSS bloquejada. La regla que ha saltat en l'exemple anterior per bloquejar la petició que conté la cadena d'XSS és al fitxer `modsecurity_crs_40_generic_attacks.conf`. La regla es compon d'expressions regulars que permeten localitzar un possible atac d'XSS.

```
SecRule REQUEST_FILENAME|ARGS|ARGS_NAMES "(?:\b(?:(?:type\b\W*?\b(?:text\b\W*?\b(?:j(?:ava)?
|ecma|vb)|application\b\W*?\bx-(?:java|vb))script|c(?:opyparentfolder|reatetextrange)|get
(?:special|parent)folder|iframe\b.{0,100}?\bsrc)\b|on(?:(?:mo(?:use(?:o(?:ver|ut)|down
|move|up)|ve)|key(?:press|down|up)|c(?:hange|lick)|s(?:elec|ubmi)t|(?:un)?load|dragdrop
|resize|focus|blur)\b\W*?=|abort\b)|(?:l(?:owsrc\b\W*?\b(?:(?:java|vb)script|shell|http)
|ivescript)|(?:href|url)\b\W*?\b(?:(?:java|vb)script|shell)|background-image|mocha):|s
(?:(?:tyle\b\W*=.*\bexpression\b\W*|ettimeout\b\W*?)\(|rc\b\W*?\b(?:(?:java|vb)script|shell
|http):)|a(?:ctivexobject\b|lert\b\W*?\(|sfunction:))|<(?:(?:body\b.*?\b(?:backgroun|onloa)d
|input\b.*?\btype\b\W*?\bimage)\b| ?(?:(?:script|meta)\b|iframe)|!\[cdata\[)|(?:\.(?:
(?:execscrip|addimpor)t|(?:fromcharcod|cooki)e|innerhtml)|\@import)\b)" \ "phase:2,capture,
t:none,t:htmlEntityDecode,t:compressWhiteSpace,t:lowercase,ctl:auditLogParts=+E,log,auditlog,
msg:'Cross-site Scripting (XSS) Attack',id:'950004',tag:'WEB_ATTACK/XSS',logdata:'%{TX.0}',
```

```
severity:'2'"
```

Igual que aquesta regla per a detectar atacs XSS n'hi ha moltes d'altres per a detectar, d'una manera genèrica, atacs d'SQL Injection, LDAP Injection, *local file inclusion*, etc.

4.2.2. Request Filtering

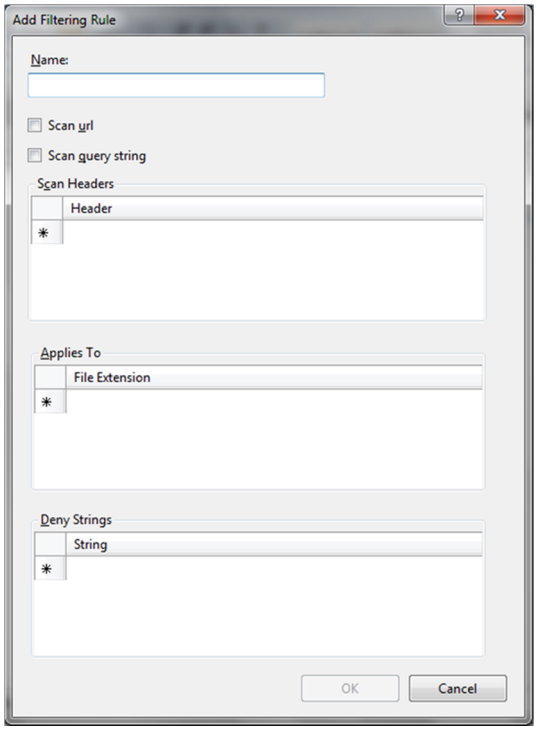
El servidor web de Microsoft, Internet Information Services, inclou en la versió 7 un mòdul especialitzat en el bloqueig de peticions web que puguin ser malicioses. Aquest mòdul es pot activar juntament amb la instal·lació del servidor web i no requereix una configuració inicial.

Quan el tenim instal·lat i hem obert la consola de configuració, ens trobem davant d'una interfície gràfica que ens permetrà establir restriccions en diferents àmbits de les peticions web. Aquests peticions són les següents:

1) Extensions de fitxers: a vegades no volem que els usuaris puguin accedir a tipus de fitxers concrets, com per exemple els que tenen extensió `.bak` o `.old`. Aquestes extensions són típiques de quan fem accions d'actualització de la nostra aplicació web i solen comportar un greu risc de seguretat. Amb aquest filtre podem bloquejar extensions que sabem que la nostra aplicació no farà servir mai. Si hem instal·lat el suport per a ASP.Net se'ns afegiran automàticament com a denegats diferents tipus d'extensions que s'usen durant el desenvolupament d'una aplicació web. Això es fa per a evitar un error d'un programador que pugi un fitxer no volgut al nostre servidor.

2) Regles personalitzades: potser és la zona més semblant a `mod_security`. Des d'aquí i mitjançant algunes opcions podem generar regles de bloqueig. Permet un control més gran sobre les regles.

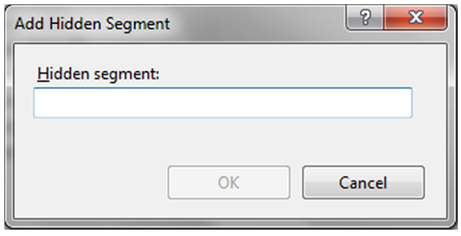
Figura 10. Diàleg de generació de regles



3) Zones ocultes: en tota aplicació web hi ha unes zones privades o unes zones en què els programadors han emmagatzemat fitxers de configuració. Aquests directoris no haurien ser accessibles per a ningú des del seu navegador. Des d'aquesta zona establim els directoris que volem bloquejar. Per defecte, i si hem instal·lat suport per a ASP.Net, tindrem els elements següents:

- `web.config`
- `bin`
- `App_code`
- `App_GlobalResources`
- `App_LocalResources`
- `App_WebReferences`
- `App_Data`
- `App_Browsers`

Figura 11. Bloqueig de segments

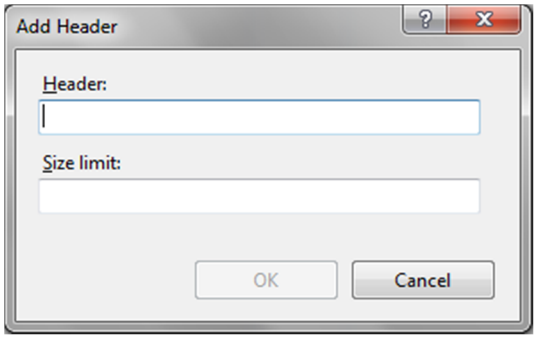


4) URL: l'apartat URL és potser la zona més general. Des d'aquí podem prohibir l'accés a URL complets. Això és útil, per exemple, per a bloquejar un fitxer concret dins d'un directori.

5) Verbs HTTP: molts atacs web es produeixen quan l'atacant fa servir verbs HTTP incorrectes. Això pot provocar que l'aplicació no sàpiga com ha de respondre o que respongui d'una manera no volguda. Des d'aquest apartat podem bloquejar o permetre l'accés de certs verbs com `GET`, `POSTo HEAD` .

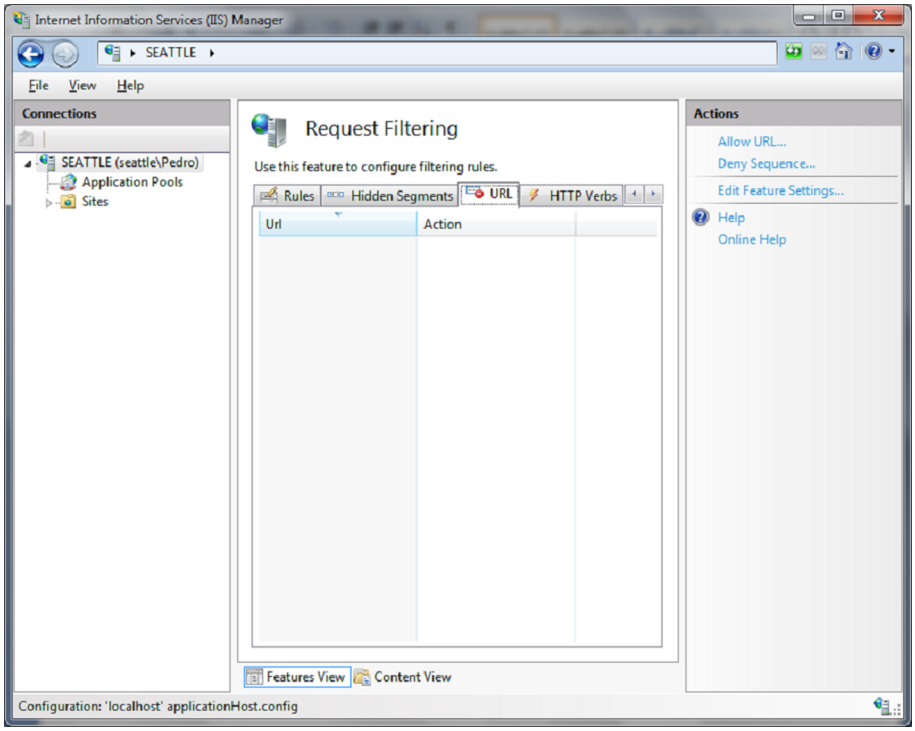
6) Capçaleres: els valors enviats a les capçaleres HTTP a vegades són interpretats per les aplicacions per a generar estadístiques o recopilar dades necessàries dels clients. A vegades els programadors es descuiden de sanejar les entrades rebudes mitjançant capçaleres perquè consideren que són impossibles de modificar. Des d'aquest apartat podrem bloquejar capçaleres específiques.

Figura 12. Bloqueig de capçaleres



7) Paràmetres: el nombre més gran d'atacs és produït per l'ús de paràmetres incorrectes o mal formats. Des d'aquesta última opció podrem establir filtres enfront de cadenes que puguin causar un risc per al nostre servidor.

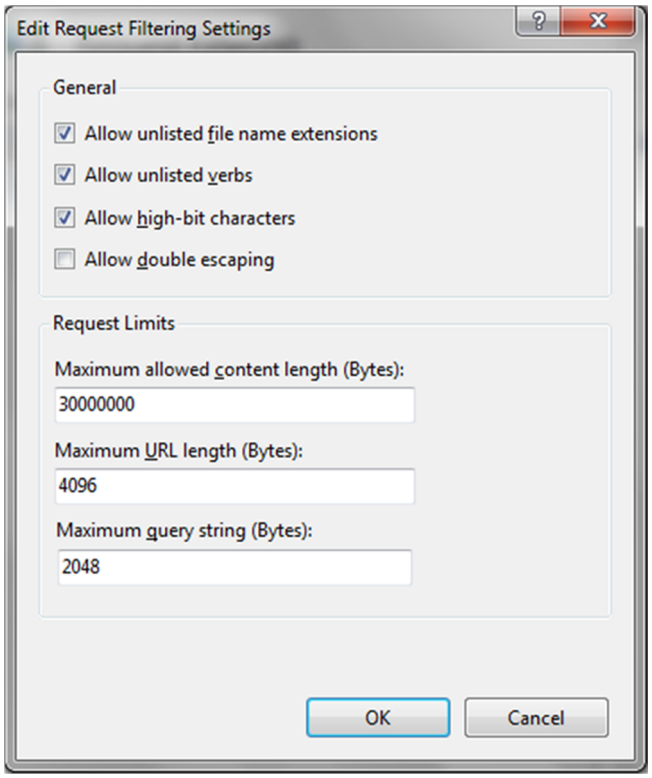
Figura 13. Pantalla general del mòdul Request Filtering



A més de totes aquestes opcions específiques per a elements concrets de les peticions HTTP, podem trobar un últim apartat de configuració. La configuració general del mòdul ens permet establir una sèrie de límits pel que fa a la mida de les peticions i a algunes característiques de seguretat permeses:

- Permet extensions de fitxers que no són a la llista.
- Permet verbs que no són a la llista.
- Permet caràcters de codi ASCII ampliat (com la *ñ*).
- Permet doble escapada de caràcters.
- Mida màxima de la petició.
- Mida màxima de l'URL.
- Mida màxima dels paràmetres.

Figura 14. Configuració general del mòdul



Aquestes opcions són molt útils. Les mides especificades per defecte com a màximes són prou grans perquè no hi hagi cap pàgina que deixi de funcionar. Aquestes mides les hem d'ajustar als límits reals del nostre servidor després de fer una anàlisi de l'ús de l'aplicació.

L'objectiu és deixar que els nostres usuaris interactuïn amb el nostre servidor durant un temps prudencial, i deixar que facin totes les tasques possibles en l'aplicació. Després d'aquest temps hem d'analitzar el registre del servidor i inferir quins són els limitis reals de la nostra aplicació. Per exemple, si hem detectat que no hi ha cap URL que excedeix de 68 caràcters i que els paràmetres

enviats no sobrepassen els 54, podrem establir uns límits de potser 80 caràcters per a l'URL i 70 per als paràmetres, deixant un marge de seguretat per a no bloquejar futures peticions.

Això és molt útil enfront d'atacs d'SQL Injection, en què els atacants poden arribar a generar URL de més de 500 caràcters, per les moltes condicions que han d'establir per extreure a cada moment el contingut volgut de la base de dades.

Bibliografia

Hope, H.; Walther, W. (2009). *Web Security Testing CookbookSystematic Techniques to Find Problems Fast*. O'Reilly Media.

Mcdonald, J. (2006). *Art of Software Security Assessment*. Pearson Professional Education.

Ristic, I. (2005). *Apache Security*. Ed. O'Reilly.

Schaefer, K.; Cochran, J.; Forsyth, S.; Baugh, R.; Everest, M.; Glendenning, D. (2008). *Professional IIS 7*. Ed. Wrox.

Takanen, A.; Demott, J. D.; Miller; C. (2007). *Fuzzing for Sotfware Security Testing and Quality Assurance*. Ed. Artech House.

Webs d'interès

http://www.petefinnigan.com/orasec.htm

http://www.codeproject.com/KB/database/SqlInjectionAttacks.aspx

http://www.techsupportalert.com/search/p1528.pdf

http://www.microsoft.com/sql/prodinfo/previousversions/securingsqlserver.mspx

http://www.databasesecurity.com/oracle-forensics.htm

http://iase.disa.mil/stigs/checklist/

http://www.codeproject.com/KB/database/SqlInjectionAttacks.aspx

http://msdn.microsoft.com/

http://www.ngssoftware.com/research/papers/cursor-snarfing.pdf

http://www.sommarskog.se/dynamic_sql.html