LDAP Injection & Blind LDAP Injection

Estos documentos han sido escritos y publicados por:

Chema Alonso, MVP de Windows Security y escribe diariamente en su blog de "Un Informático en el lado del mal".

Chema trabaja en Informática 64 y escriben en los blogs Un Informático en el lado del mal y vista-tecnica

Recopilación: Cristian Borghello, Director de www.segu-info.com.ar

V1.0 – 071006

Indice

LDAP Injection & Blind LDAP Injection (Parte I de III)

****
Artículo publicado en:
http://elladodelmal.blogspot.com/2007/10/ldap-injection-blind-ldap-injection.html
****

Las técnicas de inyección de código se han utilizado para saltarse las restricciones de seguridad de las aplicaciones y sistemas, y, para logar este objetivo, se han especializado para afectar las distintas tecnologías. Una de las "últimas" en estudiarse han sido las técnicas de Inyección LDAP. De éstas la primera referencia que encontramos es la que nos ofrece Sacha Faust, de la empresa SPI Dynamics en su documento "LDAP Injection". El resto de artículos que se han publicado hacen siempre referencia a este mismo documento, convirtiéndolo casi en la única fuente documentada y pública de este tipo de técnicas. La mayoría de las publicaciones que hay sobre este tema referencian una y otra vez los mismos ejemplos publicados por Sacha Faust y las que no lo hacen aportan poco en el estudio de este problema.

LDAP

LDAP (lLightweight Directory Access Protocol) es un protocolo de acceso "ligero" a un directorio de servicios sobre TCP/IP. Inicialmente nació como una forma de consultar las bases de datos jerarquizadas de información en los árboles X.500 pero con el tiempo cobró entidad propia y hoy en día el directorio de información es un árbol LDAP. Con sus propias estructuras y documentos que formalizan esta tecnología. La versión actual está documentada por la RCF 4511, de Junio de 2006. El protocolo de acceso LDAP se utiliza para consultar y modificar los objetos almacenados.

Filtros LDAP

Es importante comprender el funcionamiento de los filtros LDAP, para ello, podemos consultar la RFC 4515 de Junio de 2006. En ella se adjunta la definición completa del leguaje de filtros, pero podemos resumirlo en la siguiente estructura:

Filter = (filtercomp)
Filtercomp = and / or / not / item
And = & filterlist
Or = | filterlist
Not = ! filter
*Filterlist = 1*filter*
Item = simple / present / substring
Simple = attr filtertype assertionvalue
Filtertype = "=" /"~="/ ">=" / "<="
*Present = attr = **
*Substring = attr "=" [initial] * [final]*
Initial = assertionvalue
Final = assertionvalue

Como se puede apreciar un filtro siempre va entre paréntesis. Además se dispone de un conjunto muy reducido de operadores lógicos AND "&", OR "|" y NOT "!", los operadores relacionales <=, >=, = y ~= y el comodín * que se utiliza para sustituir por a "uno o varios caracteres". Así pues, ejemplos de filtros válidos serían:

- *(&(!(objectClass=Impresoras))(uid=s*))*: En este ejemplo estaríamos buscando todos los objetos que cumplan que no son de la clase Impresoras y cuyo atributo

uid tiene un valor que comienza por "s"

- *(&(objectClass=user)(uid=*))*: Este filtro nos devolvería la lista de todos los objetos de tipo user, tengan el valor que tengan en el atributo uid.

No serían filtros válidos aquellos que no utilicen notación prefija del operador o no utilicen un anidamiento correcto de paréntesis, como por ejemplo:

- *((objectClass=Impresoras)|(nombre=Epson*))*: En este ejemplo el operador lógico OR "|" no va correctamente situado.

- *((&(objectClass=Impresora))((nombre=Epson*Color))*: En este caso los paréntesis no están bien anidados.

LDAP Injection en aplicaciones Web

Las técnicas de inyección de comandos LDAP en aplicaciones web se basan en los mismos principios que las técnicas de SQL Injection. En una aplicación web el programador, en un determinado punto recoge datos enviados por el usuario que van a ser utilizados para generar una consulta LDAP. En un entorno vulnerable el programador no realiza el filtrado de los parámetros y el atacante aprovecha este fallo de seguridad para poder inyectar código y cambiar el resultado que se obtiene con el filtro.

El ejemplo de vulnerabilidad explicada por Sacha Faust en su documento muestra un entorno en el que se extrae más información mediante la unión de un filtro. Para ello supone un entorno en el que el programador va a generar un filtro simple del tipo *(atributo=valor)*. Entendemos filtro simple como aquel que no lleva un operador. Para ello construye una consulta mediante la concatenación del valor recibido convertido a cadena de caracteres que después ejecuta.

string: filter = "(uid=" + CStr(userName) + ")"

En este entorno, el atacante podría realizar una inyección de código LDAP para acceder a más objetos saltándose las restricciones del programa original. Siguiendo con el ejemplo de Sacha Faust, éste propone que se podría realizar una inyección de la siguiente forma: *(some attribute=user input)(|(cn=*))*. Para construir esta inyección, utilizando el código vulnerable habría que introducir como parámetro de entrada la siguiente cadena: ***sfaust)(|(cn=*)***

"*sfaust*" es la entrada de datos esperada por el programa para construir el filtro. Posteriormente, el atacante cierra el paréntesis que introduce el programador y abre un nuevo filtro con el operador lógico OR en el que se pide la devolución de todos los objetos sea cual sea su valor en cn.

Esta inyección, que Sacha Faust prueba sobre un árbol LDAP SunOne Directory Server 5.0 devuelve la suma de los resultados de los dos filtros: *(uid=sfaust)* y *(|(cn=*))*. Sin embargo, si nos ceñimos a la definición del lenguaje de creación de filtros marcada por la RFC vemos que esta consulta no es correcta pues los filtros no están siguiendo las normas de anidamiento y notación prefija. Sin embargo, sí podemos tomar la consulta como dos filtros independientes bien formados, aunque en segundo caso no sería necesario añadir el operador lógico OR. Probadas estas consultas sobre árboles ADAM y OpenLDAP vemos que NO devuelven más datos y por lo tanto este sistema de inyecciones NO puede ser utilizado con ellos.

Primeras conclusiones

Tras realizar estas pruebas podemos extraer las siguientes conclusiones:

1.- Para realizar una inyección de código LDAP en una aplicación que trabaje contra ADAM u OpenLDAP es necesario que el filtro original, es decir, el del programador tenga un operador OR o AND. A partir de este punto se pueden realizar inyecciones de código que permitan extraer información o realizar ataques Blind, es decir, a ciegas.

2.- En ese mismo entorno es necesario que la consulta generada tras la inyección esté correctamente anidada en un único par de paréntesis general o bien que el componente permita la ejecución con información que no se va a utilizar a la derecha del filtro.

"AND" LDAP Injection

En este entorno nos encontraríamos con que el programador ha creado una consulta LDAP con un operador AND y uno o los dos parámetros son solicitados al usuario y no está filtrado correctamente en servidor. En el caso de inyecciones en consultas LDAP que lleven el operador AND estamos obligados a utilizar el primer atributo algo válido, pero se pueden utilizar las inyecciones para mediatizar los resultados y por ejemplo, realizar escaladas de privilegios o acceso a otros objetos del árbol LDAP.

En este caso nos encontraríamos con una consulta del siguiente tipo:

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

Ejemplos: Supongamos un entorno en el que se muestra la lista de todos los documentos a los que un usuario del nivel poco privilegiado tiene acceso mediante una consulta que incluye el directorio de documentos en un parámetro inyectable. Es decir, la consulta original es:

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

Un atacante podría construir una inyección del siguiente modo para poder acceder a los documentos de nivel de seguridad alto.

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

Para conseguir este resultado se habrá inyectado en el nombre del directorio la siguiente cadena: ***almacen)(nivel_seguridad=alto))(|(directorio=almacen***

Como se puede ver, con esa inyección se han formado dos filtros correctamente formados pero los árboles ADAM y OpenLDAP devolverán sólo los resultados del primer filtro con lo que conseguiremos acceder a documentos, en este ejemplo, fuera de nuestra visibilidad incial.

"OR" LDAP Injection

En este entorno nos encontraríamos con que el programador ha creado una consulta LDAP con un operador OR y uno o los dos parámetros son solicitados al usuario:

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

Supongamos en el ejemplo del árbol LDAP que tenemos una consulta inyectable del siguiente tipo: *(|(cn=D*)(ou=Groups))* Es decir que devuelve todos los objetos cuyo valor en "cn" comience por "D" o cuyo valor en "ou" sea "Groups". Al ejecutarla obtenemos:

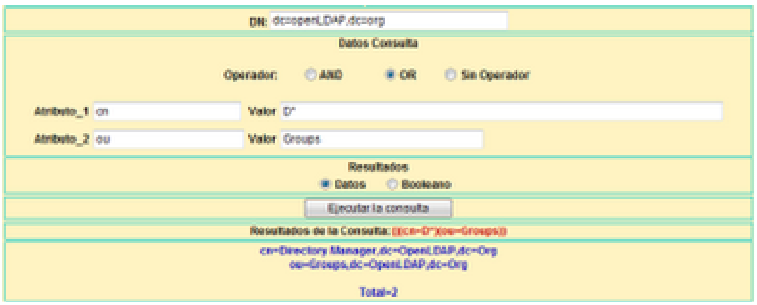

Imagen: Consulta OR sin inyección

Si esta consulta sufriera una inyección de código en el primer parámetro, podríamos realizar una consulta que nos devolviera la lista de usuarios almacenados. Para ello realizamos la inyección en el primer valor de la siguiente cadena: ***void)(uid=*))(|(uid=****

Imagen: Lista de usuarios obtenida tras la inyección

Al formarse la consulta LDAP esta quedará construida de la siguiente forma: ***(|(cn=void)(uid=*))(|(uid=*)(ou=Groups))***, permitiendo obtener, como se puede ver en la captura anterior la lista de todos los usuarios del árbol LDAP.

LDAP Injection & Blind LDAP Injection (Parte I de III)

****

Artículo publicado en:
http://elladodelmal.blogspot.com/2007/10/ldap-injection-blind-ldap-injection_06.html

****

Blind LDAP Injection

Una de las evoluciones de las inyecciones de comandos son las acciones a ciegas; los ataques "Blind". Es común encontrarse ya documentados los ataques Blind SQL Injection, pero de los ataques Blind LDAP Injection sin embargo no se ha hablado nada aún. ¿Es posible realizar un ataque a ciegas para extraer información de un árbol LDAP? La respuesta, obviamente, es Sí.

El entorno para realizar un ataque a ciegas suele ser una aplicación web que cumple dos características:

- La aplicación no muestra los resultados obtenidos de la consulta realiza al árbol LDAP.

- El uso de los resultados produce cambios en la respuesta http que el cliente recibe.

Para realizar un ataque a ciegas necesitamos poder crear la lógica para extraer información cambiando los resultados obtenidos con el filtro LDAP y observando los cambios en las respuestas http.

Blind LDAP Injection en un entorno "AND"

En esta situación tenemos una consulta generada en el lado del servidor del siguiente tipo:

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

Para realizar la inyección deberemos tener en cuenta que en un entrono AND tenemos un parámetro fijo que nos va a mediatizar. El entorno ideal es que el atributo1 sea lo más general posible, por ejemplo objectClass, o cn, con lo que podremos seleccionar casi cualquier objeto del árbol. Después deberemos aplicarle un valor que sea True, es decir, que siempre seleccione objetos para tenerlo fijado a valor true y utilizar como segundo atributo uno conocido que también nos garantice que sea True.

Supongamos la siguiente consulta LDAP:

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

que se lanza para saber si en el almacén de una empresa hay impresoras Epson de tal manera que el programador, si recibe algún objeto, simplemente pinta en la página web un icono de una impresora Epson.

Está claro que lo más que conseguiremos tras la inyección es ver o no el icono de la impresora. Bien, pues a ello. Fijemos la respuesta positiva tal y como hemos dicho, realizando una inyección del siguiente tipo:

*(&(objectClass=***)(objectClass=*))(&(objectClass=void**)(impresoraTipo=Epso*

n))*

Lo que está puesto en negrita sería la inyección que introduciría el atacante. Esta inyección debe devolver siempre algún objeto luego, en el ejemplo planteado, en la respuesta http veríamos un icono de la impresora. A partir de ahí podríamos extraer los tipos de objectClass que tenemos en nuestro árbol LDAP.

Reconocimiento de clases de objetos de un árbol LDAP

Sabiendo que el filtro *(objectClass=*)* siempre devuelve algún objeto deberíamos centrarnos en el segundo filtro realizando un recorrido de la siguiente forma:

(&(objectClass=)(objectClass=users))(&(objectClass=foo)(impresoraTipo=Epson*
))*
(&(objectClass=)(objectClass=personas))(&(objectClass=foo)(impresoraTipo=Eps
on*))*
(&(objectClass=)(objectClass=usuarios))(&(objectClass=foo)(impresoraTipo=Epso
n*))*
(&(objectClass=)(objectClass=logins))(&(objectClass=foo)(impresoraTipo=Epson*
))*
(&(objectClass=)(objectClass=…))(&(objectClass=foo)(impresoraTipo=Epson*))*

De tal forma que todas aquellas inyecciones que devolvieran el icono de la impresora en la página web serían respuestas afirmativas que nos indicarían la existencia de ese tipo de objetos.

Otra forma más eficiente sería realizando un barrido en el espectro del alfabeto de posibles valores utilizando una búsqueda con comodines, para ello realizaríamos un árbol que comenzaría por todas las letras del abecedario, de la siguiente forma:

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

De tal manera que seguiríamos desplegando el árbol de aquellas que hubieran dado una respuesta positiva.

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

Y así sucesivamente con lo que, desplegando siempre las positivas podríamos llegar a saber el nombre de todos los objetcClass de un sistema.

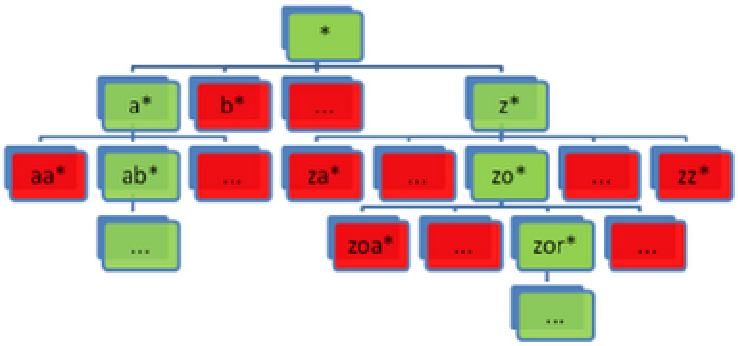

Imagen: Árbol de despliegue.Se profundizará en todas las respuestas positivas (color verde)

Este mecanismo no solo funciona para objectClass sino que sería válido para cualquier atributo que un objeto compartiera con el atributo fijo. Una vez sacados los objectClass, podríamos proceder a realizar el mismo recorrido para extraer la información de ellos, por ejemplo, si queremos extraer los nombres de todos los usuarios de un sistema, bastaría con realizar el barrido con la siguiente inyección:

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

Este método obliga a hacer un recorrido en amplitud por un árbol cuya posibilidad de expandirse en cada nodo siempre es igual al alfabeto completo. Se puede reducir este alfabeto realizando un recorrido para averiguar que caracteres no están siendo utilizados en ninguna posición en ningún objeto utilizando un recorrido de la siguiente forma:

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

De esta forma se puede excluir del alfabeto a aquellos caracteres que no hayan devuelto un resultado positivo tras este primer recorrido. Si desearamos buscar un único valor, podríamos realizar una búsqueda binaria de cada uno de los caracteres utilizando los operadores realionales <= o >= para reducir el número de peticiones.

Blind LDAP Injection en un entorno "OR"

En esta situación tenemos una consulta generada en el lado del servidor del siguiente tipo:

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

En este entorno la lógica que debemos utilizar es al contrario. En lugar de fijar el valor del primer filtro a un valor siempre cierto deberemos fijarlo a un valor siempre falso y a partir de ahí, extraer la información, es decir, realizaríamos una inyección de la siguiente forma:

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

Con esta inyección obtendremos el valor negativo de referencia. En el ejemplo planteado de las impresoras disponibles deberemos obtener un resultado sin el icono de la impresora. Luego podremos inferir la información cuando sea verdadero el segundo filtro.

(|(objectClass=void)(objectClass=users))(&(objectClass=void)(impresoraTipo=Epson))*
(|(objectClass=void)(objectClass=personas))(&(objectClass=void)(impresoraTipo=Epson))*
(|(objectClass=void)(objectClass=usuarios))(&(objectClass=void)(impresoraTipo=Epson))*
(|(objectClass=void)(objectClass=logins))(&(objectClass=void)(impresoraTipo=Epson))*
(|(objectClass=void)(objectClass=…))(&(objectClass=void)(impresoraTipo=Epson))*

De igual forma que en el ejemplo con el operador AND podríamos extraer toda la información del árbol LDAP utilizando los comodines.

Conclusiones

LDAP es una tecnología en expansión cuya implantación en los sistemas de directorio y meta directorio la hacen cada día más popular. Los entornos Intranet realizan uso intensivo de esta tecnología para ofrecer sistemas de Single Sing-On e incluso es común encontrar entornos LDAP en los servicios de Internet.

Esta popularidad hace que sea necesario incluir las pruebas anteriores dentro de los procesos de auditoría de seguridad de las aplicaciones web. Las pruebas de inyecciones de auditoría que se estaban realizando hoy en día, siguiendo la documentación existente, son de poca ayuda pues en los entornos más comunes como ADAM u OpenLDAP no funcionan.

La solución para proteger las aplicaciones frente a este tipo de inyecciones es, como en otros entornos de explotación, filtrar los parámetros enviados desde el cliente. En este caso filtrar operadores, paréntesis y comodines para que nunca un atacante sea capaz de inyectar lógica dentro de nuestra aplicación.

LDAP Injection & Blind LDAP Injection (Parte III de III)

****

Artículo publicado en:
http://elladodelmal.blogspot.com/2007/10/ldap-injection-blind-ldap-injection_7929.html

****

En este parte del artículo simplemente se han probado las inyecciones descritas por Sacha Faust en su documento "LDAP Injection" sobre entornos ADAM y OpenLDAP.

LDAP Injection con ADAM de Microsoft

Para realizar todas las pruebas de las cadenas a inyectar hemos utilizado la herramienta LDAP Browser que permite conectarse a distintos árboles LDAP y una cliente web creado con el componente IPWorksASP.LDAP de la empresa /n Software, para realizar las pruebas de ejecución de filtros. En la imagen 1 se muestra la estructura que hemos creado de ejemplo en ADAM.

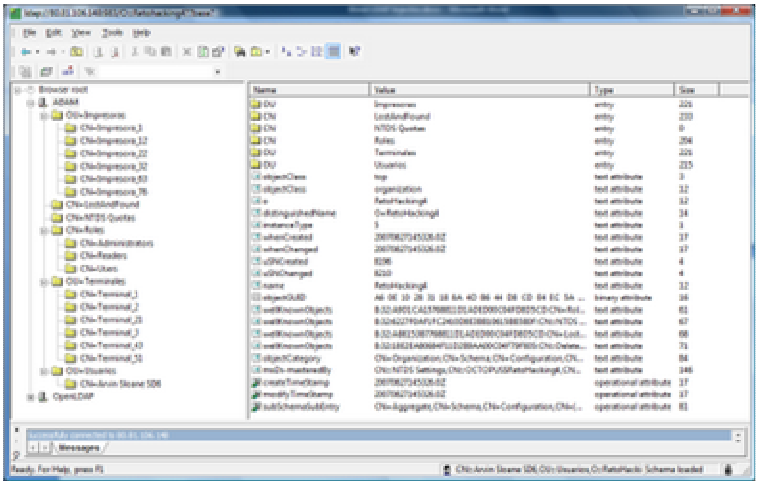

Imagen: Estructura del árbol LDAP creado en ADAM

Supongamos ahora que la aplicación web utiliza una consulta con el siguiente filtro: (cn=Impresora_1). Al lanzar esta consulta se obtiene 1 objeto, como se puede ver en la imagen siguiente:

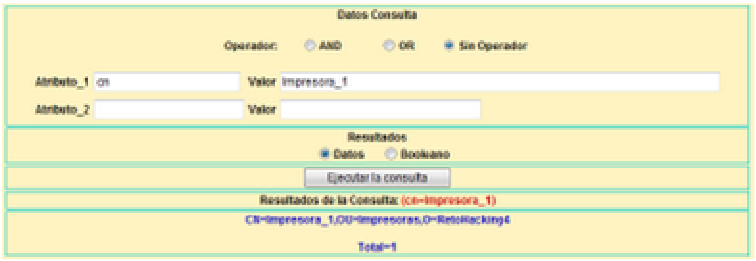

Imagen: Se obtiene un objeto de respuesta con el filtro (cn=Impresora_1)

Lo deseable por un atacante que realice una inyección sería poder acceder a toda la información mediante la inclusión una consulta que nos devolviera todas las impresoras, en un supuesto ataque. En el ejemplo, vemos cual debería ser el

resultado a obtener con una consulta siguiendo la RFC 4415 sobre ADAM:
(|(cn=Impresora_1)(cn=Impresora*))

Imagen: Todas las impresoras

Sin embargo, como se puede apreciar, para construir ese filtro, necesitaríamos inyectar un operador y un paréntesis al principio. En el ejemplo, la inyección no "parece" ser muy útil pues se está realizando en ambas condicionantes sobre cn, pero ese filtro sólo está creado para ilustrar la necesidad de inyectar código antes del filtro y en el medio del filtro. Si probamos la inyección propuesta por Sacha Faust en ADAM: ***(cn=Impresora_1)(|(cn=Impresora*))***

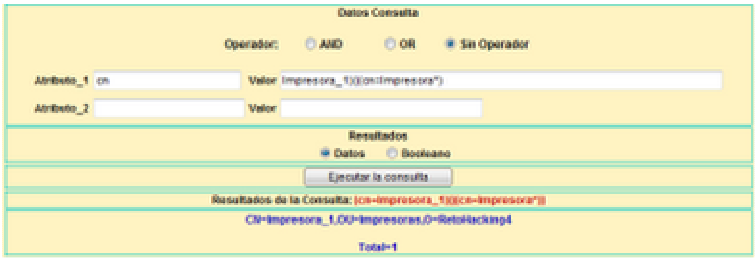

Imagen: Inyección sin resultados

Como se puede apreciar en la captura, en la última prueba, la inyección no produce ningún error, pero a diferencia de las pruebas que realiza Sacha Faust con SunOne Directory Server 5.0, el servidor ADAM no devuelve más datos. Es decir, sólo devuelve los datos del primer filtro completo y el resto de la cadena es ignorado.

LDAP Injection con OpenLDAP

Para realizar las pruebas de inyección en OpenLDAP hemos utilizado el árbol que ofrece de pruebas el propio proyecto OpenLPAD.org cuya estructura es la siguiente:

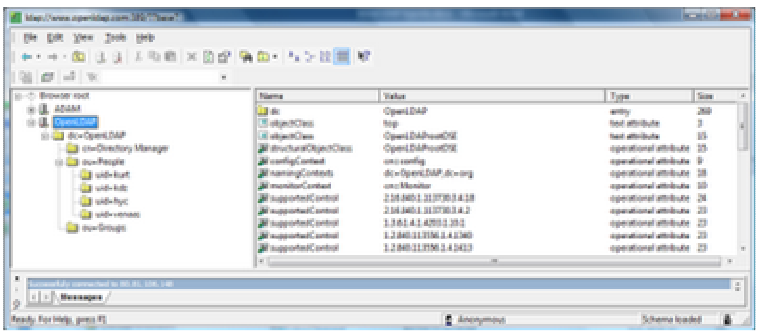

Imagen: Estructura del árbol LDAP en OpenLDAP

Sobre esta estructura ejecutamos una consulta para buscar a un usuario obteniendo un único objeto como resultado: ***(uid=kurt)***

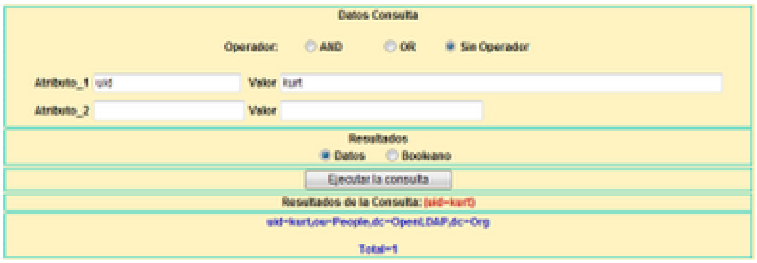

Imagen: Al ejecutar el filtro (uid=kurt) obtenemos un único resultado

Si quisiéramos ampliar el número de resultados, siguiendo la RFC 4415 deberíamos ejecutar una consulta en la que inyectáramos un operador OR y un filtro que incluyera a todos los resultados, como se puede ver en la siguiente imagen: ***(|(uid=kurt)(uid=*))***

Imagen: Todos los usuarios

Si realizamos la inyección tal y como propone Sacha Faust en su documento sobre LDAP obtendríamos los siguientes resultados: ***(uid=kurt)(|(uid=*))***

Imagen: Inyección OpenLDAP. Se ignora el segundo filtro.

Como puede apreciarse en la captura, el servidor OpenLDAP ha ignorado el segundo filtro y solo ha devuelto un usuario, de igual forma que realizaba ADAM.

Capturando credenciales LDAP con CAIN. Step by Step. Por Juan Luís Rambla

****

Artículo publicado en:
http://elladodelmal.blogspot.com/2007/10/capturando-credenciales-ldap-con-cain.html

****

A través de las siguientes líneas vamos a describir como se produce el ataque de robo de credenciales de autentificación LDAP, mediante el uso de la técnica complementaria de Man in the middle. Para la realización de esta operación presentamos un escenario de LDAP basado en Directorio Activo, para lo cual contamos con un dominio llamado Informatica64.hol, al que realizaremos una conexión mediante una aplicación estándar de consulta LDAP como es LDAP Browser en su versión 2.6.

Como herramienta para la realización del ataque de hombre en medio, utilizaremos Caín & Abel 4.9.6, que cuenta con la posibilidad de realizar Sniffing de sesiones LDAP y que nos servirá para recuperar la contraseña de autentificación contra el servicio de directorio existentes para el dominio de Informatica64. Para ello durante el proceso de "Bind" del cliente contra el servidor LDAP, se interceptará todo el proceso de autentificación del usuario, con el consiguiente robo de credenciales. En esta ocasión el objetivo es el robo de credenciales del usuario "Administrator".

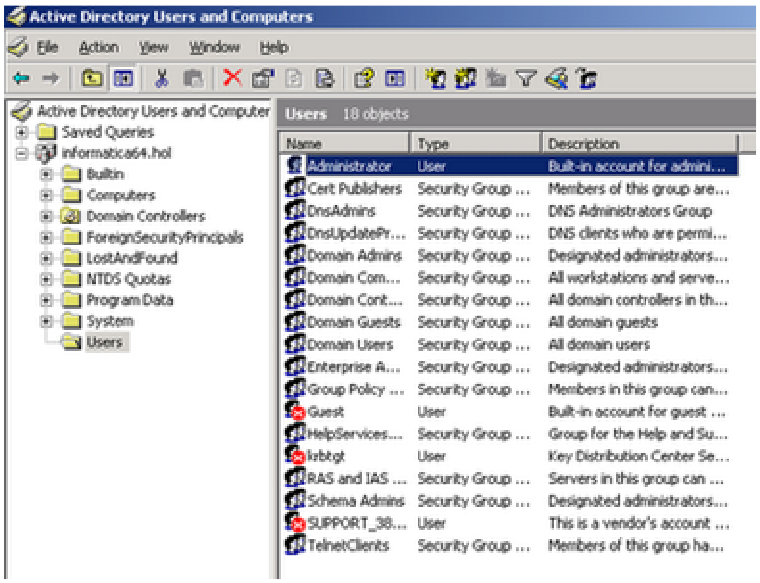

Fig. 1.- Directorio Activo.

Con objeto de realizar el proceso de autentificación LDAP, procedemos a configurar la herramienta LDAP Browser, con las conexión al Controlador de dominio y las credenciales de autentificación del usuario correspondiente en formato de Nombre Distinguido (DN).

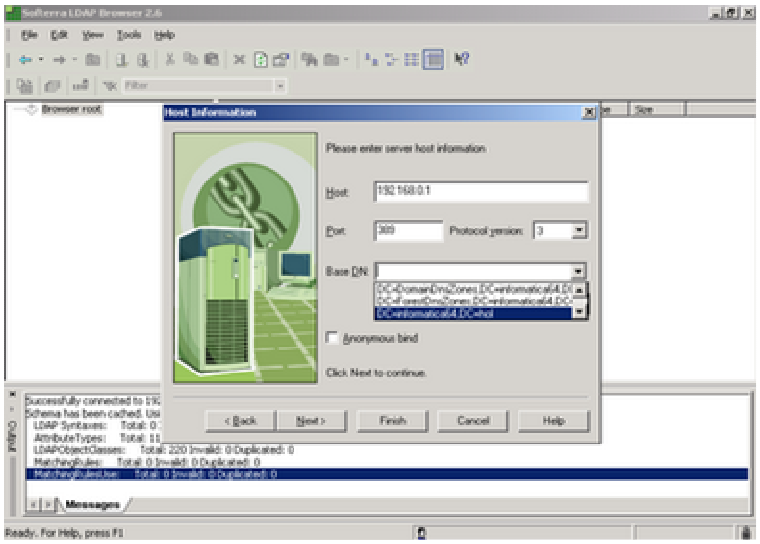

Fig. 2.- Configuración conexión cliente LDAP.

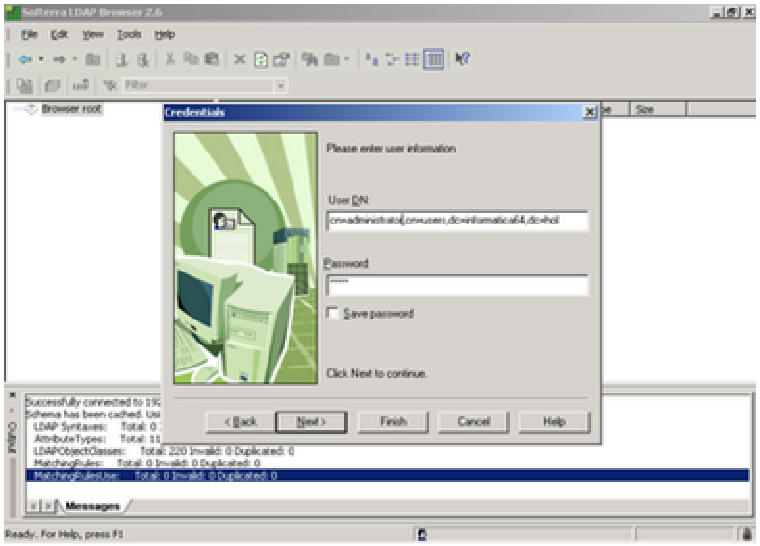

Fig. 3.- Configuración de credenciales.

Una vez realizada la configuración del cliente y proporcionado las credenciales se procede a la conexión con el servidor LDAP y que tal y como se refleja se obtiene como resultado el servicio de directorio, así como sus objetos, clases, atributos y propiedades.

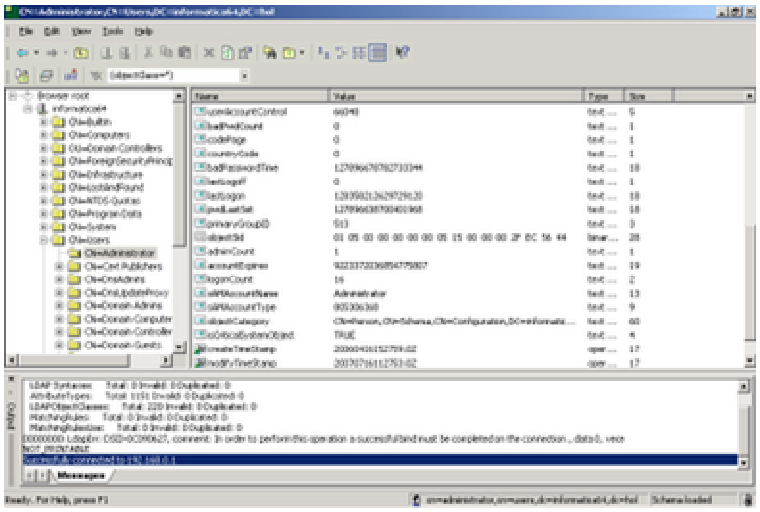

Fig. 4.- Conexión LDAP.

Como hemos comprobado que la conexión se ha realizado correctamente, vamos a proceder a la interceptación y el sniffing de esta sesión LDAP. Como para la realización de cualquier otra técnica de ataque de hombre en medio, se procede a la realización del envenenamiento de la dirección física, de las dos máquinas que intervienen en el proceso (cliente y servidor LDAP) y la posterior captura de las credenciales de autentificación.

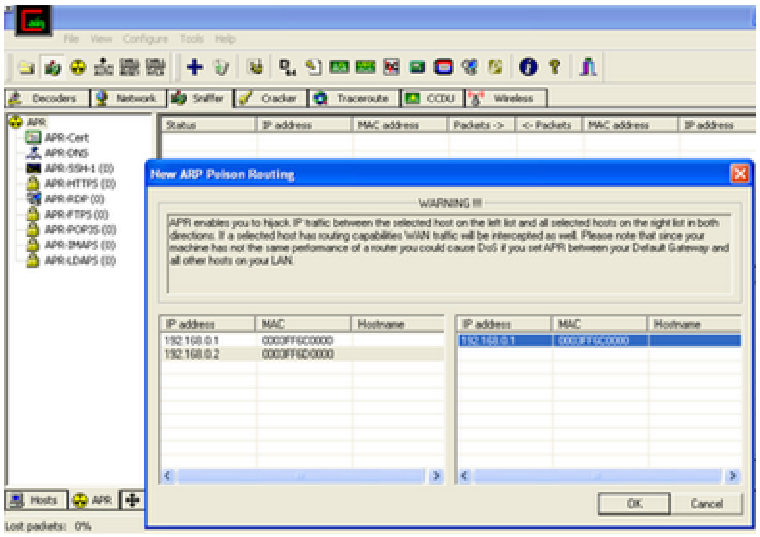

Fig. 5.- Configuración APR.

Una vez puesta en marcha la sesión de Snifing, esperamos a que se intercepte la autentificación LDAP del cliente, para a obtención de las credenciales esperadas. En este caso la autentificación interceptada, se produce al usuario 'Administrator', que ha utilizado como contraseña la palabra 'admin'.

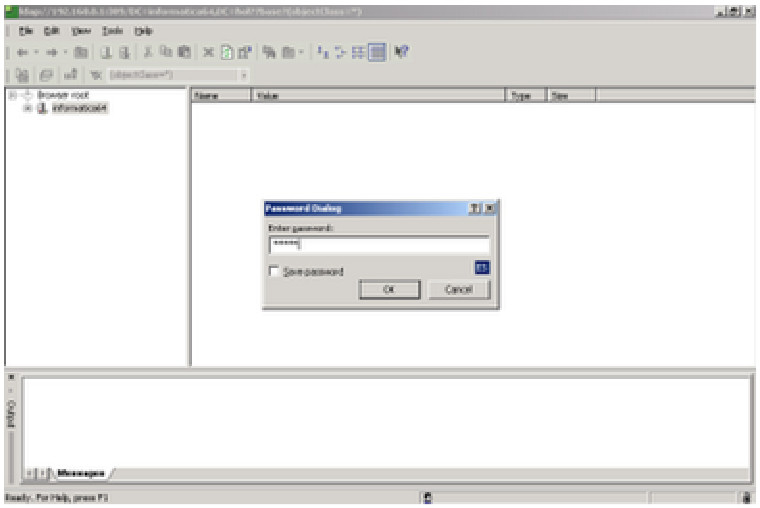

Fig. 6.- Autentificación LDAP.

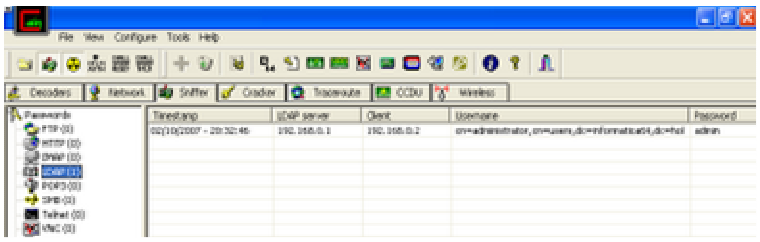

Fig. 7.- Robo de credenciales LDAP.