

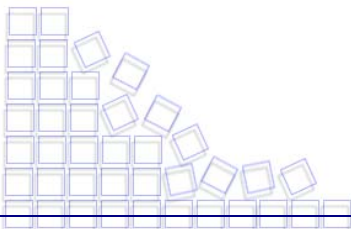
A vertical, blue-tinted photograph of a computer keyboard, showing several keys and a mouse cord. The image is positioned on the left side of the page.

LDAP Injection

Are your web applications vulnerable?

Table of Contents

Web Application and LDAP Injection	1
Overview	1
Background	1
Environment.....	1
LDAP Query Introduction	1
Attacking LDAP Search Queries.....	3
Understanding the Query Construction	3
Generating Attacks.....	8
Prevention.....	11
Incoming Data Validation.....	11
Outgoing Data Validation	12
LDAP Configuration	12
About SPI Labs.....	13
About SPI Dynamics.....	13
About the WebInspect Product Line.....	14
About the Author.....	14
Contact Information.....	14
Appendix A: LDAP References	15
Appendix B: Further Reading	16
Appendix C: Example Source Code	17
Appendix D: LDAP Search Filter Syntax.....	19



Web Application and LDAP Injection

Overview

Lightweight Directory Access Protocol (LDAP) is a widely used protocol for accessing information directories. LDAP injection is the technique of exploiting web applications that use client-supplied data in LDAP statements without first stripping potentially harmful characters from the request. The objective of this paper is to inform developers, system administrators and security professionals about various techniques that could be used to attack their applications. It also describes preventive measures for protecting applications from these intrusions.

Background

Readers should have a basic understanding of LDAP technology and web application parameter injection. For background information, see the SPI Labs white paper [“SQL Injection: Are Your Applications Vulnerable?”](#)

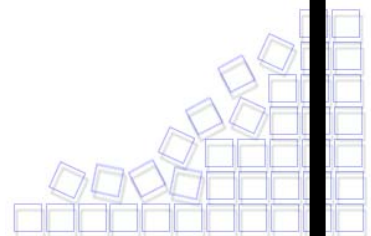
Environment

The example used in this white paper was written using Active Server Page (ASP) under Microsoft Information Server (IIS) and making use of the [LDAP control](#) written by [nSoftware](#). The back-end LDAP server used is SunOne Directory Server 5.0.

LDAP Query Introduction

Before discussing how to attack web applications using LDAP, let’s review some of the basics of how LDAP search queries are constructed and what to expect in return. Let’s look at how the example application used in this paper construct its queries and how it deals with the data returned.

The example application (see [Appendix C: Example Source Code](#)) simply takes a query argument called “user” and searches the LDAP directory for the user’s cn (common name), mail, and telephone number attributes. Once the data is returned, the application displays the information to the user, as illustrated in Figure 1.



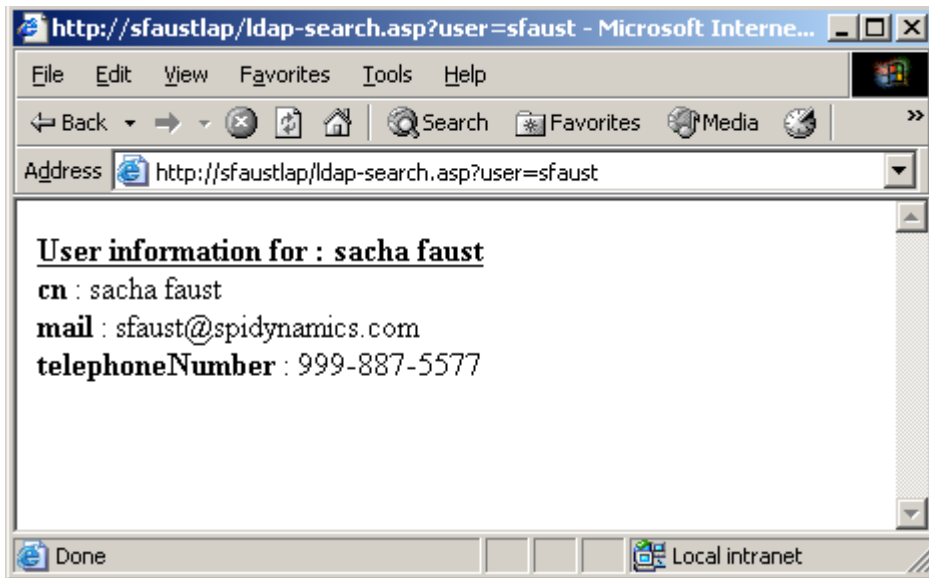


Figure 1: Normal application processing.

To examine how the application constructs the query, let's use the debug switch in the query string. Figure 2 displays the underlying LDAP query construction based on the user information.

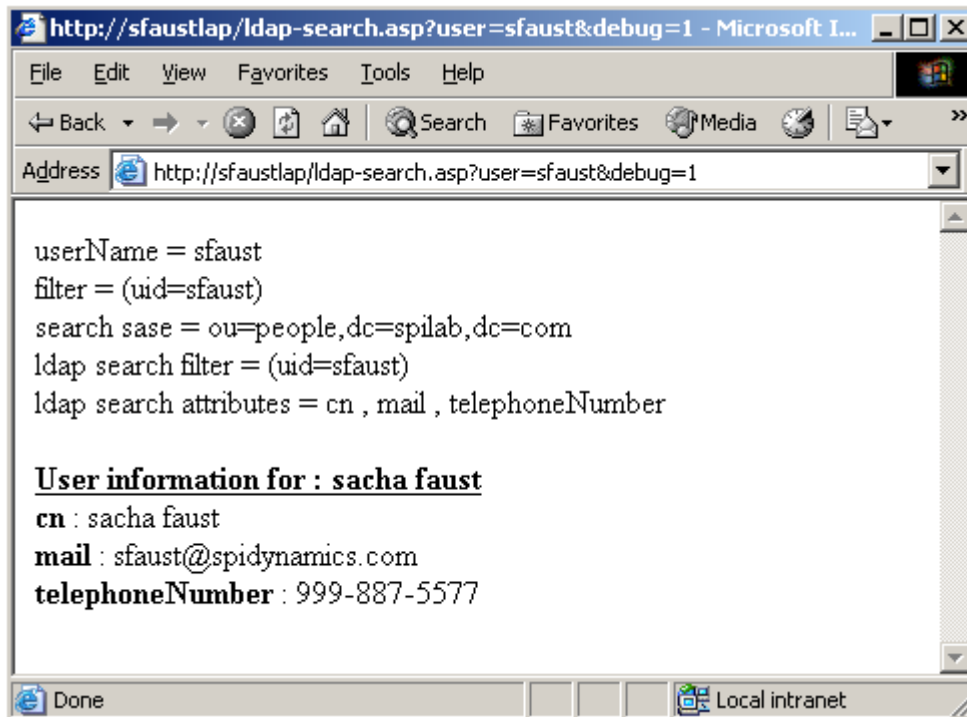


Figure 2: Normal search operation with debug information.

According to the displayed debug information, the application searches the ou=people,dc=spilab,dc=com tree of the directory for the attributes cn, mail and telephoneNumber. The filter part is a bit more complicated and provides the mechanism for LDAP injections. The filter used in the application is uid = user_supplied_value, where the value is sfaust. For more information about the syntax of LDAP search filters, refer [Appendix D: LDAP Search Filter Syntax](#) or the Microsoft MSDN topic available at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netdir/adsisearch_filter_syntax.asp

Attacking LDAP Search Queries

The most widely use of LDAP in web applications is to enable users to easily search for specific data on the Internet. For example, a college or university might electronically publish white pages that allow users to find information about students and teachers. As illustrated in Figure 3, the example LDAP-enabled Web application ([Appendix C](#)) displays specific information about a user by accepting the user name in a query argument.

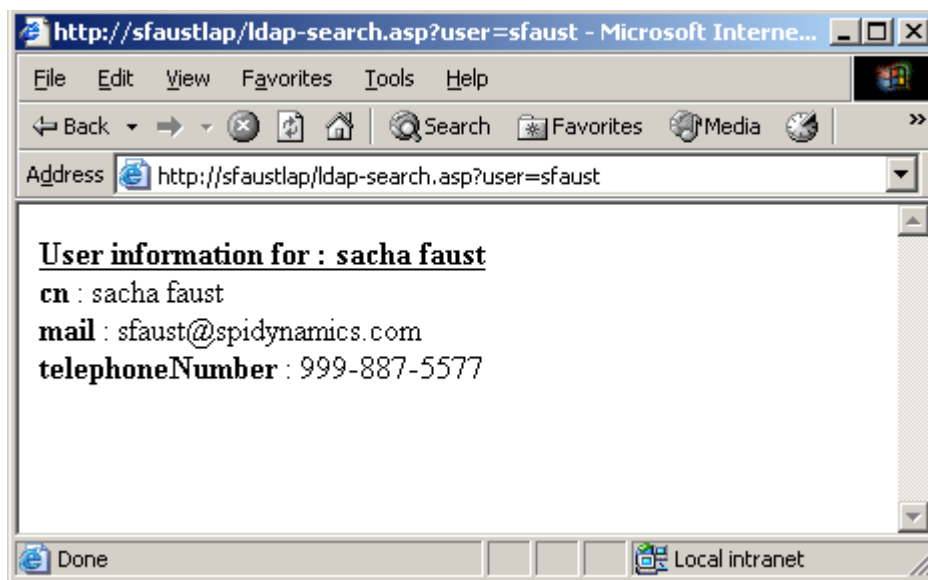
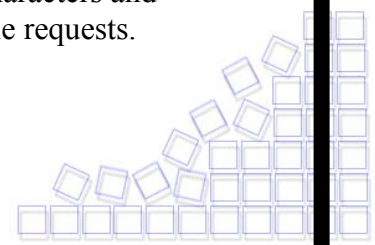


Figure 3: Normal search operation.

Understanding the Query Construction

Now that we have a basic understand of what the application does, let's examine how the application is constructing the LDAP query to get the information. The first step should be to determine if the application is attempting any type of validation on the data sent by the user. To test for this, we can send a few requests with unusual characters and see how the application reacts to them. Figures 4 and 5 illustrate these simple requests.



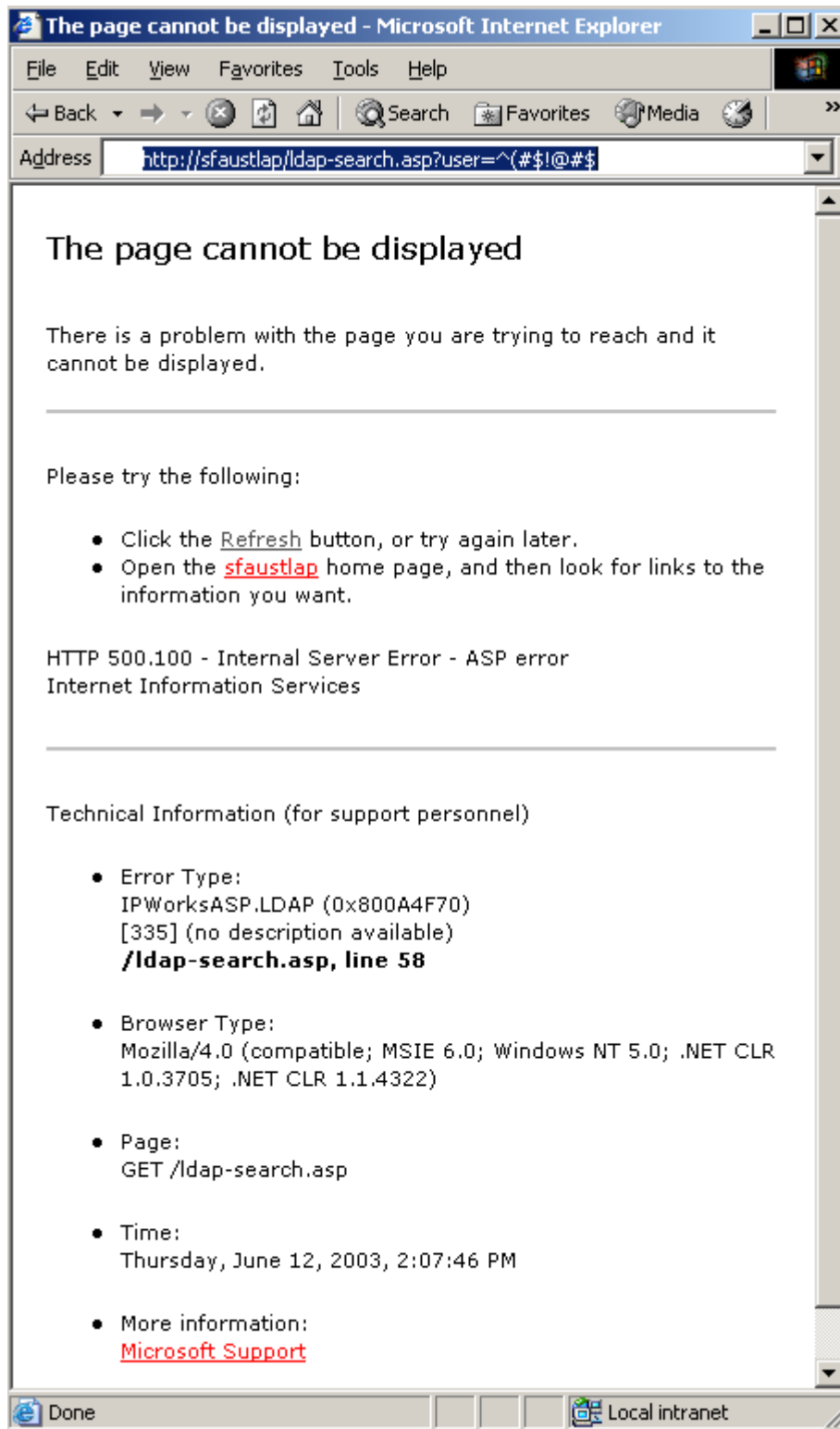


Figure 4: First test for user input validation.

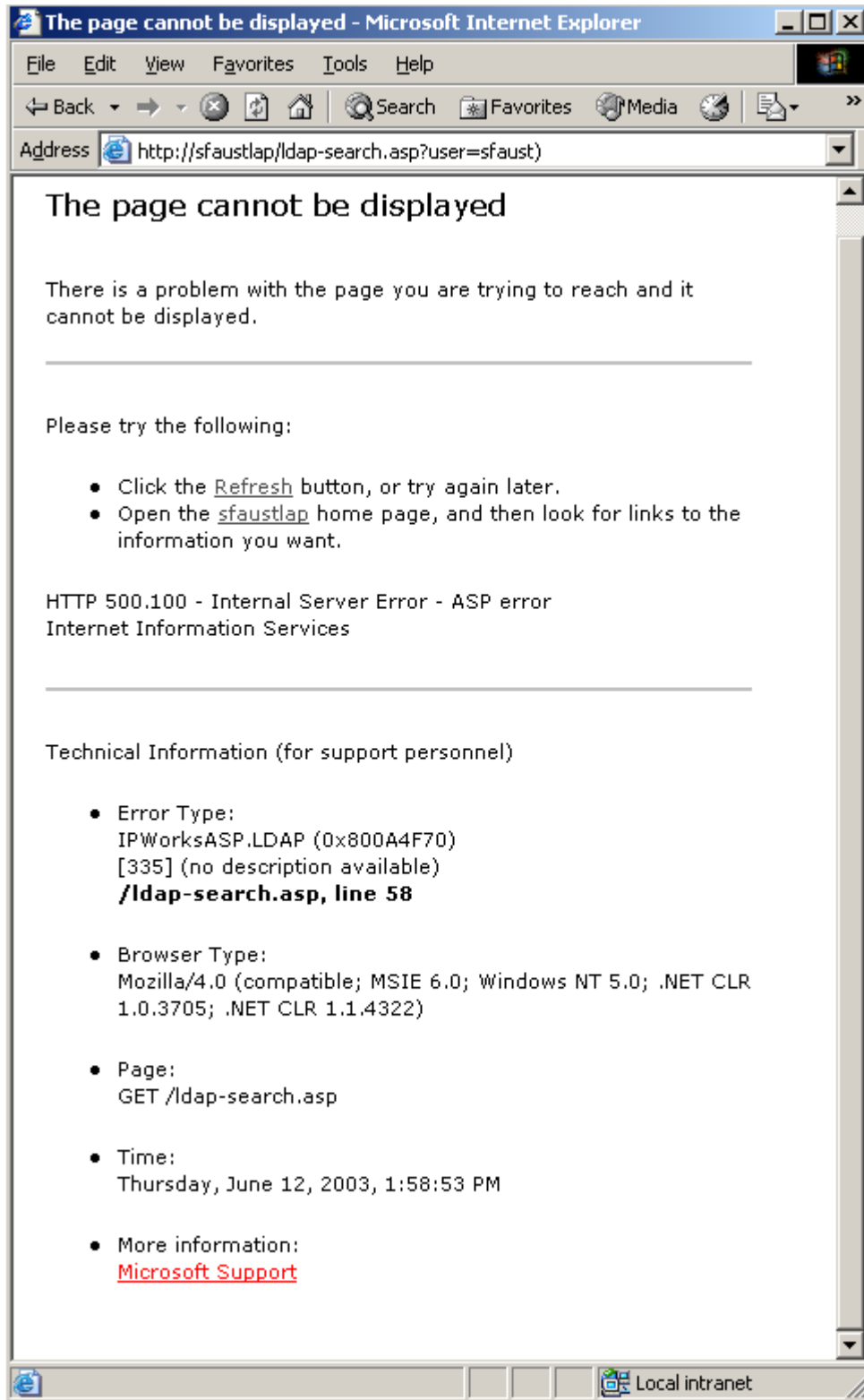
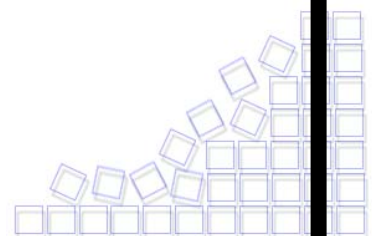


Figure 5: Second test for user input validation.



In the first validation test, the probe sends data that even the least sophisticated data validation routine would reject (as seen in Figure 4). In the second test, the probe sends a request that might look like a valid character in an LDAP query string (as seen in Figure 5). The returned data illustrated in Figures 4 and 5 indicate that the application is not validating the data passed in the query and is storing the value directly into the LDAP object. Since the data is injected in the query, the application returns an error because of the invalid LDAP query created. This test application also sends a closing parenthesis, but it could use any valid filter character such as the following: | (&.

Having identified the type of validation performed by the target application, the attacker can reverse-engineer the structure of the LDAP query to determine how the user-supplied data is used to perform the search. The LDAP search filters are always enclosed between parentheses (see [Appendix D: LDAP Search Filter Syntax](#) for more information). To locate the data in the filter string, try to generate valid LDAP filters by adding a few valid characters to the beginning and end of the argument value, and then examine the response.

If the application reports an error, the probe is generating an invalid request. If the application returns without errors or with new data, the probe has created a valid query. This process can be time-consuming depending on the size of the query and the number of arguments sent. Figures 6 and 7 present a few examples.

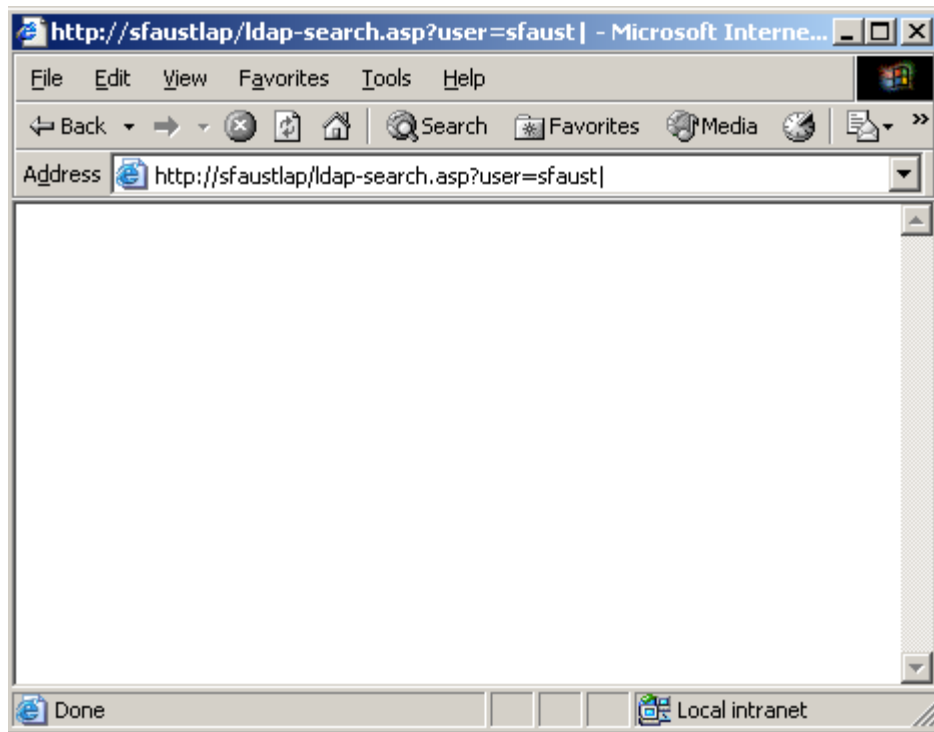


Figure 6: Sending “|” returns no errors and no data.

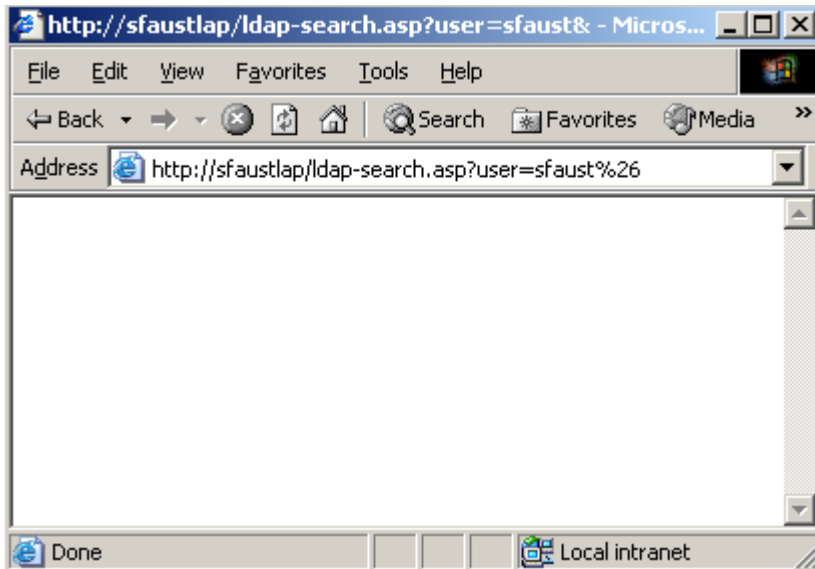
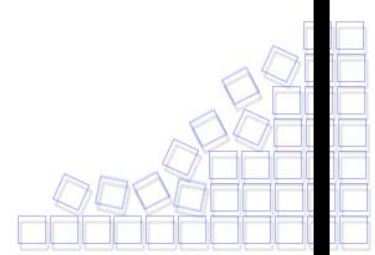


Figure 7: Sending ampersand (%26) returns no errors and no data.

As illustrated in Figures 6 and 7, we can see how the query is structured by sending logical operators such as OR and AND. The actual query substitutes the symbols “|” for OR and “&” (URL-encoded as %26, to prevent the target application from interpreting the operator as query name/value separator) for AND. Because the target application does not return an error, the injected values must have created a valid query. With that information in mind and by looking at the LDAP query syntax, the attacker can conclude that the application generates a query in the following format: (some attribute=user input). The injected value in Figure 6 would generate (some attribute=user input|). In Figure 7, the injected value would generate (some attribute=user input&). Both are valid queries that return no values.

To verify these assumptions, try to get the cn value of the user sfaust. This requires injecting data to generate a query that looks like (some attribute=user input)((cn=*)), which instructs the server to return any cn values. Since the assumption is that the query looks like (some attribute = user input), to get (some attribute=user input)((cn=*)), we will need to inject sfaust)((cn=*). Figure 8 illustrates the actual string and the results of the query.



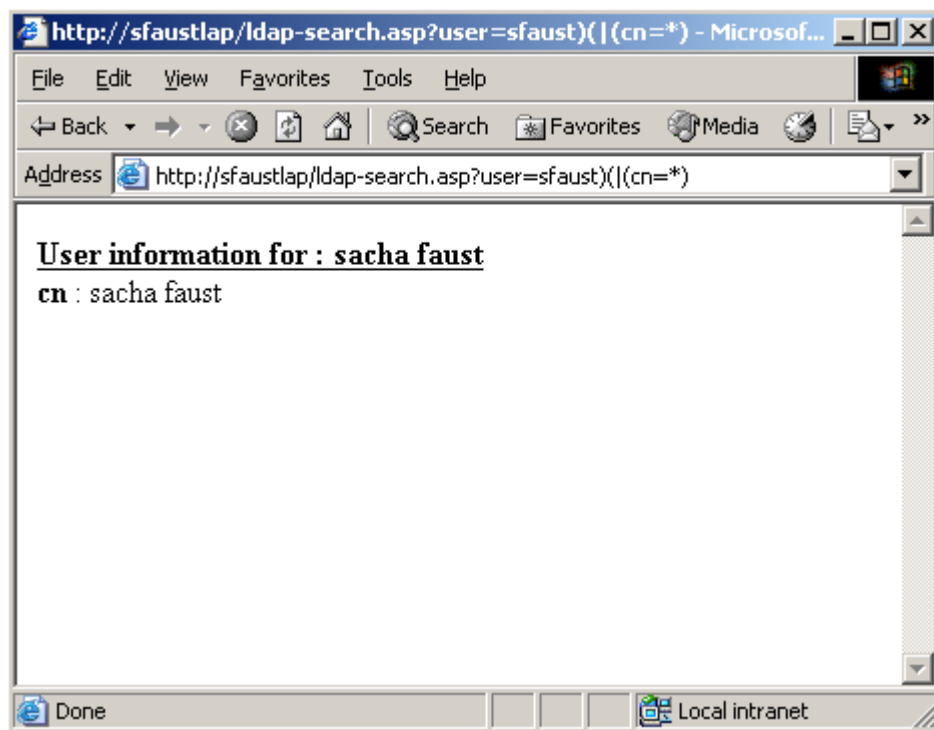
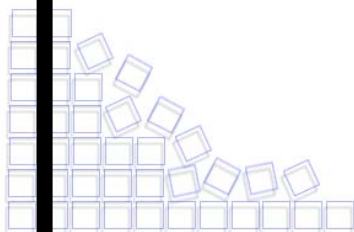


Figure 8: Getting the cn value of the user.

In this example, the injection works, confirming the assumptions about the structure of the query.

Generating Attacks

Having determined the structure of the query, we can generate additional attacks to access more information. First, we need to discover what attributes are available by querying the LDAP server to obtain an objectclass listing. Then, simply refer to <http://docs.sun.com/source/816-6699-10/objclass.html> to see the attributes included in each objectclass. If the objectclass listed is not on the above site, you can usually find the information by searching the Internet with your favorite search engine. Figure 9 shows the list of available objectclasses for our user.



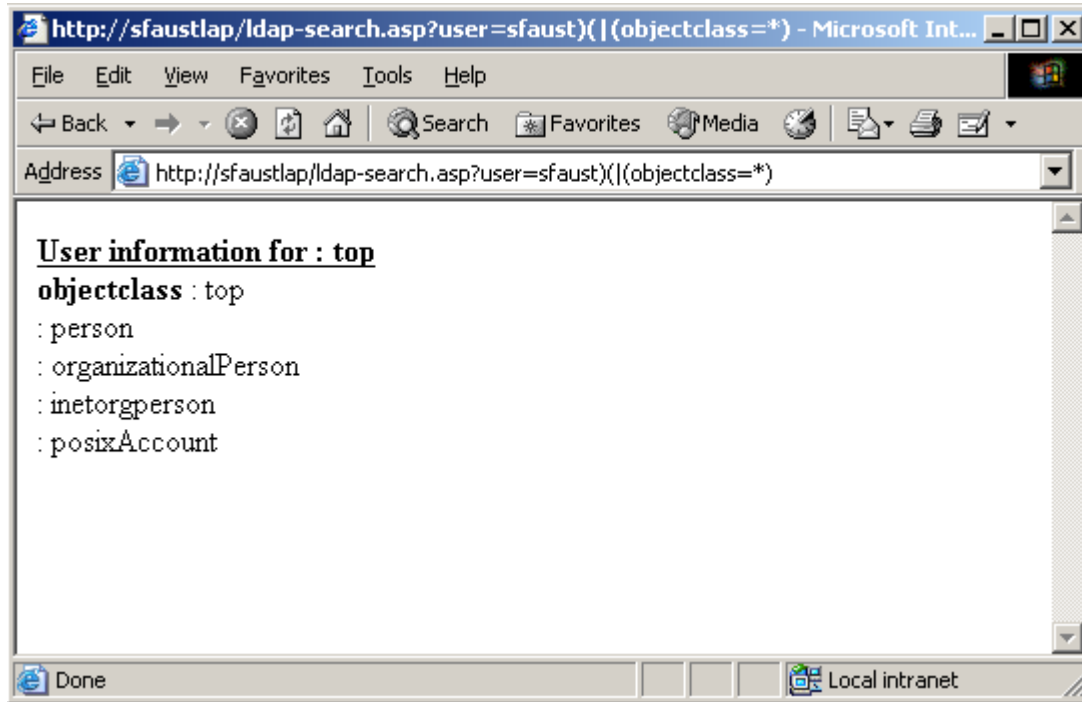
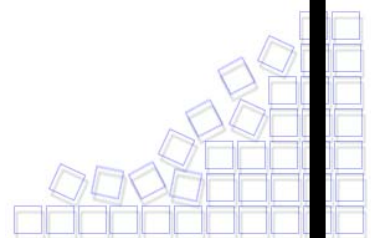


Figure 9: Getting a list of available objectclasses.

Now that we have a list of objectclasses, we can pick one and see if we have rights to view the data. This example uses the *posixAccount* objectclass, but any could contain interesting information. By looking at the class definition in [RFC 2307](#), we see that the following attributes are required and should be available.

- cn
- uid
- uidNumber
- gidNumber
- homeDirectory

In theory, we should be allowed to see any of these attributes. Figure 10 illustrates the attempt to get the home directory of the user *sfaust*.



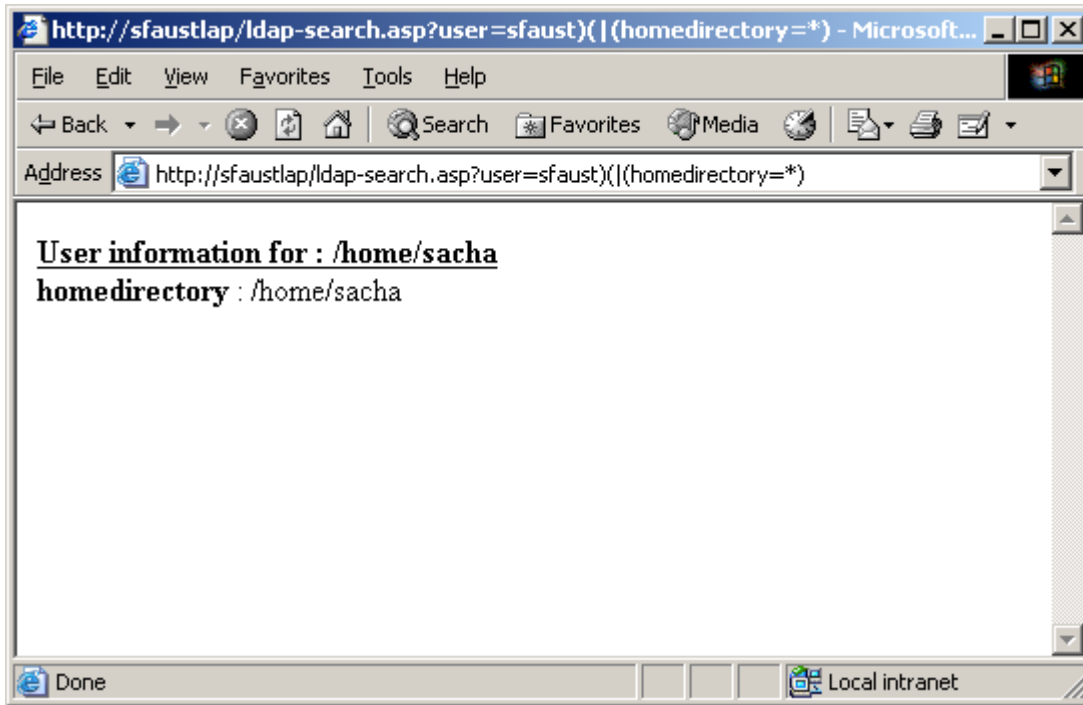
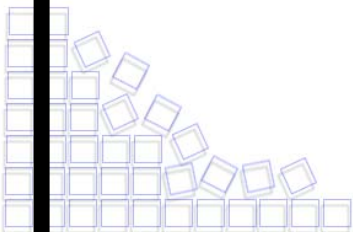


Figure 10: Getting the home directory for user sfaust.

Bingo! We now know that we can view the attributes of the *posixAccount* objectclass. We can apply the same techniques to all the objectclasses and obtain the data.

To obtain a listing of all the users on the system (and view their settings), simply use a wildcard character as the user value. Figure 11 illustrates this request and the subsequent response.



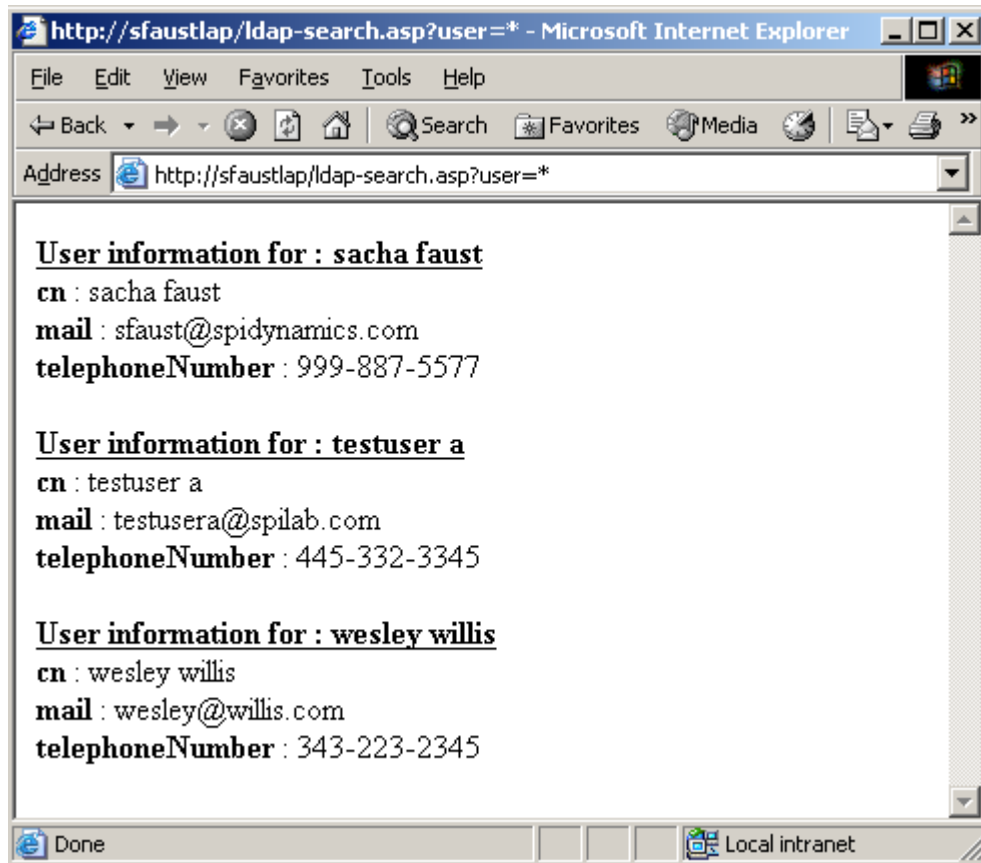


Figure 11: Getting a listing of all users.

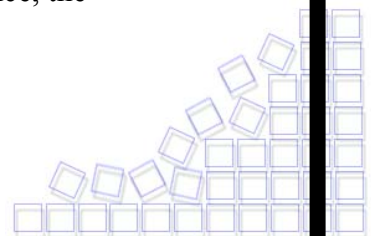
Prevention

Protecting LDAP-enabled web applications demands the effort of developers as well as the LDAP administrators. Though effective at reducing the risk of such an attack, the approaches discussed in the next section are not complete solutions. It is best to remember that web application security, by its own definition, must be a continually evolving process. As hackers change their methodologies, so must those who want to implement a secure Web application.

Incoming Data Validation

All client-supplied data needs to be cleaned of any characters or strings that could possibly be used maliciously. This should be done for all applications, not just those that use LDAP queries. Stripping quotes or putting backslashes in front of them is nowhere near enough. The best way to filter data is with a default-deny regular expression that includes only the type of characters that you want. For instance, the following regular expression will return only letters and numbers:

```
s/[^0-9a-zA-Z]/g
```



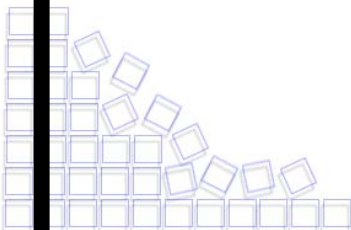
Make your filter as specific as possible. Whenever possible use only numbers. After that, numbers and letters only. If you need to include symbols or punctuation of any kind, make absolutely sure to convert them to HTML substitutes (such as “ " ” or “ > ”). For instance, if the user is submitting an email address, allow only the “at” sign, underscore, period, and hyphen in addition to numbers and letters, and only after those characters have been converted to their HTML substitutes.

Outgoing Data Validation

All data returned to the user should be validated and the amount of data returned by the queries should be restricted as an added layer of security.

LDAP Configuration

Implementing tight access control on the data in the LDAP directory is imperative, especially when configuring the permissions on user objects, and even more importantly if the directory is used for single sign-on solution. You must fully understand how each objectclass is used and decide if the user should be allowed to modify it. Allowing users to modify their uidNumber attribute, for example, may let the user change access levels when accessing systems. The access level used by the Web application to connect to the LDAP server should be restricted to the absolute minimum required. That way, even if an attacker manages to find a way to break the application, the damage would be limited. In addition, the LDAP server should not be directly accessible on the Internet, thereby eliminating direct attacks to the server itself.



About SPI Labs

SPI Labs is the dedicated application security research and testing team of SPI Dynamics. Composed of some of the industry's top security experts, SPI Labs is focused specifically on researching security vulnerabilities at the web application layer. The SPI Labs mission is to provide objective research to the security community and all organizations concerned with their security practices.

SPI Dynamics utilizes direct research from SPI Labs to provide daily updates to WebInspect, the leading Web application security assessment software. "Zero-day" vulnerability commitment gives customers immediate access to the most up-to-date technology available to identify potential vulnerabilities within their Web applications and servers.

SPI Labs engineers comply with the standards proposed by the Internet Engineering Task Force (IETF) for responsible security vulnerability disclosure. SPI Labs policies and procedures for disclosure are outlined on the SPI Dynamics web site at: <http://www.spidynamics.com/spilabs.html>.

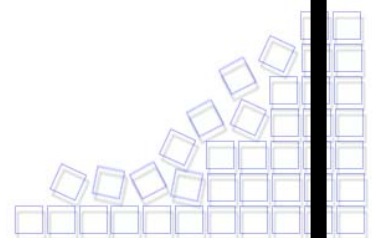
SPI Labs is directed by Caleb Sima, SPI Dynamics co-founder and chief technology officer.

About SPI Dynamics

SPI Dynamics, the expert in web application security assessment, provides software and services to help enterprises protect against the loss of confidential data through the web application layer. The company's flagship product line, WebInspect, assesses the security of an organization's applications and web services, the most vulnerable yet least secure IT infrastructure component. Since its inception, SPI Dynamics has focused exclusively on web application security. SPI Labs, the internal research group of SPI Dynamics, is recognized as the industry's foremost authority in this area.

Software developers, quality assurance professionals, corporate security auditors and security practitioners use WebInspect products throughout the application lifecycle to identify security vulnerabilities that would otherwise go undetected by traditional measures such as automated application testing tools, network firewalls, intrusion detection systems, or manual code reviews. The security assurance provided by WebInspect helps Fortune 500 companies and organizations in regulated industries — including financial services, health care and government — protect their sensitive data and comply with legal mandates and regulations regarding privacy and information security.

SPI Dynamics is privately held with headquarters in Atlanta, Georgia



About the WebInspect Product Line

The WebInspect product line ensures the security of your entire network with intuitive, intelligent, and accurate processes that dynamically scan standard and proprietary web applications to identify known and unidentified application vulnerabilities. WebInspect products provide a new level of protection for your critical business information. With WebInspect products, you find and correct vulnerabilities at their source, before attackers can exploit them.

Whether you are an application developer, security auditor, QA professional or security consultant, WebInspect provides the tools you need to ensure the security of your web applications through a powerful combination of unique Adaptive-Agent™ technology and SPI Dynamics' industry-leading and continuously updated vulnerability database, SecureBase™. Through Adaptive-Agent technology, you can quickly and accurately assess the security of your web content, regardless of your environment. WebInspect enables users to perform security assessments for any web application, including these industry-leading application platforms:

- IBM WebSphere
- Macromedia ColdFusion
- Lotus Domino
- Oracle Application Server
- Macromedia JRun
- BEA Weblogic
- Jakarta Tomcat

About the Author

As a senior research and development engineer at SPI Dynamics, Sacha Faust is responsible for researching new techniques for web auditing, conducting source code reviews to find vulnerabilities, and securing web applications. He may be reached via e-mail at sfaust@spidynamics.com.

Contact Information

SPI Dynamics
115 Perimeter Center Place
Suite 270
Atlanta, GA 30346

Telephone: (678) 781-4800
Fax: (678) 781-4850
Email: info@spidynamics.com
Web: www.spidynamics.com



Appendix A: LDAP References

For more information on LDAP, refer to these additional resources.

RFC 1960 - A String Representation of LDAP Search Filters

<http://www.ietf.org/rfc/rfc1960.txt>

LDAP Overview

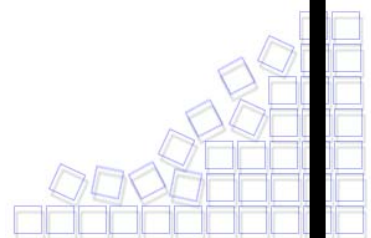
by Bruce Greenblatt, http://www.directory-applications.com/ldap3_files/frame.htm

Understanding LDAP

by IBM, <http://www.redbooks.ibm.com/redbooks/SG244986.html>

LDAPMAN web site

<http://ldapman.org/>



Appendix B: Further Reading

Introduction to LDAP Security

http://www.severus.org/sacha/docperso/intro_to_ldap_tisc.htm

SunOne schema – good link for objectclass definitions

<http://docs.sun.com/source/816-6699-10/objclass.html>

Understanding and Deploying LDAP Directory Services

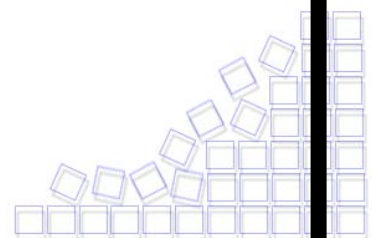
by Tim Howes, Timothy A. Howes, Mark C. Smith, Gordon S. Good

ISBN: 0-672-32316-8

LDAP: Programming Directory-Enabled Applications With Lightweight Directory Access Protocol

by Tim Howes, Mark Smith.

ISBN: 1-57870-000-0



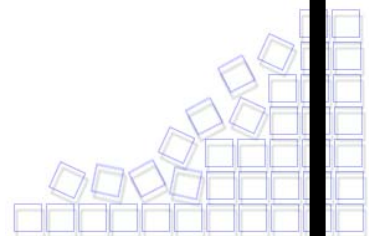
Appendix C: Example Source Code

This is the source code of the ldap-search.asp file used in this whitepaper.

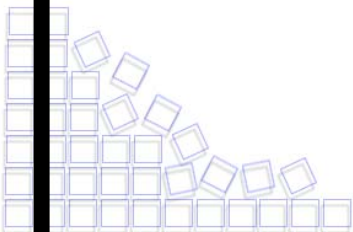
```

<html>
<body>
<%@ Language=VBScript %>
<%
    Dim userName
    Dim debug
    Dim filter
    Const LDAP_SERVER = "ldaptest.spilab.com" 'you need to point to your ldap server
    debug = False
    if( Request.QueryString("debug") <> "" ) then
        debug = CBool(Request.QueryString("debug"))
    end if
    userName = Request.QueryString("user")
    if( userName = "" ) then
        Response.Write("<b>Invalid request. Please specify a valid user name</b><br>")
        Response.End()
    end if
    if( debug ) then
        Response.Write("userName = " + userName + "<br>")
    end if
    filter = "(uid=" + CStr(userName) + ")"          ' searching for the user entry
    if( debug ) then
        Response.Write("filter = " + filter + "<br>")
    end if
    Call PerformSearch(filter)
    Sub PerformSearch( filter )
        Dim ldapObj
        'Creating the LDAP object and setting the base dn
        Set ldapObj = Server.CreateObject("IPWorksASP.LDAP")
        ldapObj.ServerName = LDAP_SERVER
        ldapObj.DN = "ou=people,dc=spilab,dc=com"
        'Setting the search filter
        ldapObj.SearchFilter = filter
        'Setting the attributes we are looking for
        ldapObj.AttrCount = 3
        ldapObj.AttrType(0) = "cn"
        ldapObj.AttrType(1) = "mail"
        ldapObj.AttrType(2) = "telephoneNumber"
        if( debug ) then
            Response.Write("search sase = " & ldapObj.DN & "<br>")
            Response.Write("ldap search filter = " & ldapObj.SearchFilter & "<br>")
            Dim searchAttrStr
            For i = 0 To ldapObj.AttrCount -1
                if( i = 0 ) then          ' for cleaner output
                    searchAttrStr = "ldap search attributes = " &

```



```
        ldapObj.AttrType(i)
                else
                    searchAttrStr = searchAttrStr & " , " & ldapObj.AttrType(i)
                end if
            Next
        if( i > 0 ) then
            Response.Write(searchAttrStr & "<br>" )
        end if
    end if
    ldapObj.Search
    'Showing the user information
    While ldapObj.NextResult = 1
        Response.Write("<p>")
        Response.Write("<b><u>User information for : " + ldapObj.AttrValue(0) +
"</u></b><br>")
        For i = 0 To ldapObj.AttrCount -1
            Response.Write("<b>" + ldapObj.AttrType(i) + "</b> : " +
ldapObj.AttrValue(i) + "<br>" )
        Next
        Response.Write("</p>")
    Wend
End Sub
%>
</body>
</html>
```



Appendix D: LDAP Search Filter Syntax

This was obtained from [RFC 1960](#)

```

<filter> ::= '(' <filtercomp> ')'
<filtercomp> ::= <and> | <or> | <not> | <item>
<and> ::= '&' <filterlist>
<or> ::= '|' <filterlist>
<not> ::= '!' <filter>
<filterlist> ::= <filter> | <filter> <filterlist>
<item> ::= <simple> | <present> | <substring>
<simple> ::= <attr> <filtertype> <value>
<filtertype> ::= <equal> | <approx> | <ge> | <le>
<equal> ::= '='
<approx> ::= '~='
<ge> ::= '>='
<le> ::= '<='
<present> ::= <attr> '='*
<substring> ::= <attr> '=' <initial> <any> <final>
<initial> ::= NULL | <value>
<any> ::= '*' <starval>
<starval> ::= NULL | <value> '*' <starval>
<final> ::= NULL | <value>

```

I also recommend taking a look at

http://msdn.microsoft.com/library/default.asp?url=/library/en-us/netdir/adsisearch_filter_syntax.asp

