Concise Summarizer

Export of <https://partyrock.aws/u/onkarr/8gtjzS4R2/Concise-Summarizer>

# App Description

This app allows you to upload a document and receive a concise summary covering all the key points.

# Document Upload

Filename: java-IDS54-J.PreventLDAPinjection-120223-0749-9246.pdf
IDS54-J. Prevent LDAP injection
The Lightweight Directory Access Protocol (LDAP) allows an application to remotely perform operations such as searching and modifying records in
directories. LDAP injection results from inadequate input sanitization and validation and allows malicious users to glean restricted information using the
directory service.
A whitelist can be used to restrict input to a list of valid characters. Characters and character sequences that must be excluded from whitelists—including
Java Naming and Directory Interface (JNDI) metacharacters and LDAP special characters—are listed in the following table.
Characters and Sequences to Exclude from Whitelists
Character Name
' and " Single and double quote
/ and \ Forward slash and backslash
\ Double slashes\*
space Space character at beginning or end of string

Hash character at the beginning of the string

< and > Angle brackets
, and ; Comma and semicolon

* and \* Addition and multiplication operators
  ( and ) Round braces
  \u0000 Unicode characterNULL
* This is a character sequence.
  LDAP Injection Example
  Consider an LDAP Data Interchange Format (LDIF) file that contains records in the following format:
  dn: dc=example,dc=com
  objectclass: dcobject
  objectClass: organization
  o: Some Name
  dc: example
  dn: ou=People,dc=example,dc=com
  ou: People
  objectClass: dcobject
  objectClass: organizationalUnit
  dc: example
  dn: cn=Manager,ou=People,dc=example,dc=com
  cn: Manager
  sn: John Watson

Several objectClass definitions here (omitted)

userPassword: secret1
mail: [john@holmesassociates.com](mailto:john@holmesassociates.com)
dn: cn=Senior Manager,ou=People,dc=example,dc=com
cn: Senior Manager
sn: Sherlock Holmes

Several objectClass definitions here (omitted)

userPassword: secret2
mail: [sherlock@holmesassociates.com](mailto:sherlock@holmesassociates.com)
A search for a valid user name and password often takes the form
(&(sn=<USERSN>)(userPassword=<USERPASSWORD>))

However, an attacker could bypass authentication by using for the field and for the field. Such input would yield everyS\* USERSN \* USERPASSWORD
record whose field began with .USERSN S
An authentication routine that permitted LDAP injection would allow unauthorized users to log in. Likewise, a search routine would allow an attacker to
discover part or all of the data in the directory.
Noncompliant Code Example
This noncompliant code example allows a caller of the method to search for a record in the directory using the LDAP protocol. ThesearchRecord()
string is used to filter the result set for those entries that match a user name and password supplied by the caller.filter
// String userSN = "S\*"; // Invalid
// String userPassword = "*"; // Invalid
public class LDAPInjection {
private void searchRecord(String userSN, String userPassword) throws NamingException {
Hashtable<String, String> env = new Hashtable<String, String>();
env.put(Context.INITIAL\_CONTEXT\_FACTORY, "com.sun.jndi.ldap.LdapCtxFactory");
try {
DirContext dctx = new InitialDirContext(env);
SearchControls sc = new SearchControls();
String[] attributeFilter = {"cn", "mail"};
sc.setReturningAttributes(attributeFilter);
sc.setSearchScope(SearchControls.SUBTREE\_SCOPE);
String base = "dc=example,dc=com";
// The following resolves to (&(sn=S*)(userPassword=\*))
String filter = "(&(sn=" + userSN + ")(userPassword=" + userPassword + "))";
NamingEnumeration<?> results = dctx.search(base, filter, sc);
while (results.hasMore()) {
SearchResult sr = (SearchResult) results.next();
Attributes attrs = (Attributes) sr.getAttributes();
Attribute attr = (Attribute) attrs.get("cn");
System.out.println(attr);
attr = (Attribute) attrs.get("mail");
System.out.println(attr);
}
dctx.close();
} catch (NamingException e) {
// Forward to handler
}
}
}
When a malicious user enters specially crafted input, as outlined previously, this elementary authentication scheme fails to confine the output of the search
query to the information for which the user has access privileges.
Compliant Solution
This compliant solution uses a whitelist to sanitize user input so that the string contains only valid characters. In this code, may containfilter userSN
only letters and spaces, whereas a password may contain only alphanumeric characters.

// String userSN = "Sherlock Holmes"; // Valid
// String userPassword = "secret2"; // Valid
// ... beginning of LDAPInjection.searchRecord()...
sc.setSearchScope(SearchControls.SUBTREE\_SCOPE);
String base = "dc=example,dc=com";
if (!userSN.matches("[\w\s]*") || !userPassword.matches("[\w]*")) {
throw new IllegalArgumentException("Invalid input");
}
String filter = "(&(sn = " + userSN + ")(userPassword=" + userPassword + "))";
// ... remainder of LDAPInjection.searchRecord()...
When a database field such as a password must include special characters, it is critical to ensure that the authentic data is stored in sanitized form in the
database and also that any user input is normalized before the validation or comparison takes place. Using characters that have special meanings in JNDI
and LDAP in the absence of a comprehensive normalization and whitelisting-based routine is discouraged. Special characters must be transformed to
sanitized, safe values before they are added to the whitelist expression against which input will be validated. Likewise, normalization of user input should
occur before the validation step.
Applicability
Failure to sanitize untrusted input can result in information disclosure and privilege escalation.
Automated Detection
Tool Version Checker Description
The Checker Framework 2.1.3 Trust and security errors (see Chapter 8)
Tainting Checker
Parasoft Jtest 2022.2 Protect against LDAP injection
CERT.IDS54.TDLDAP
SonarQube 6.7
S2078
Bibliography
[ ]OWASP 2014a Preventing LDAP Injection in Java

# Document Summary

Here is a concise summary of the document IDS54-J on preventing LDAP injection:

The document explains that LDAP injection vulnerabilities can allow attackers to access restricted directory information by injecting malicious input that is improperly sanitized. To prevent this:

* Use a whitelist to restrict input to only valid characters, excluding metacharacters and LDAP special characters like quotes, slashes, spaces, angle brackets, etc.
* Sanitize user input by matching against the whitelist of allowed characters using a regular expression before using it in LDAP queries.
* For fields like passwords that require special chars, ensure the data is stored sanitized and normalize user input before validation.
* Do not use unsanitized user input directly in LDAP queries.

The document provides noncompliant code examples showing LDAP injection vulnerabilities and compliant solutions that properly validate and sanitize user input against a whitelist before using it in LDAP queries. Preventing LDAP injection by sanitizing untrusted input is critical to avoid information disclosure and privilege escalation.