NOTE: SCENARIO IS WHAT THE PUZZLE TAKER SEES.

SCENARIO:

The Lightweight Directory Access Protocol (LDAP) is a protocol for accessing and maintaining distributed directory information services over an IP network. It allows an application to remotely perform operations, such as searching and modifying records in directories (directory servers). A common usage of LDAP is to provide single-sign-on (SSO) service where one user password is shared between many services.

You are developing an application that authenticates clients using the SSO service provided by the directory server of a company network. To perform the authentication, the `authenticate` method takes three `String` arguments: (1) `base`, the basename in the directory server, (2) `user`, the name of the account being authenticated, and (3) `passwd`, the password of the account. The method returns `true` if there is one, and only one, LDAP entry with the same credential information, and `false` otherwise. Consider the snippet of code below and answer the following questions, assuming that the code has all required permissions to execute.

Code:

```
01  // OMITTED: Import whatever is needed
02  public final class LdapAuthenticationProvider {
03    private static final Hashtable<String, Object> environment;
04
05    // OMITTED: Initialize the 'environment' field
06    // properly in the 'static' block of the class.
07
08    public static boolean authenticate (String base,
09        String user, String passwd) {
10      DirContext context;
11      try {
12        // A SearchControls object controls the search operation.
13        SearchControls sc = new SearchControls();
14        // OMITTED: Initialize 'sc' with the required parameters.
15
16        // The following query is used for authentication.
17        String qry = "(&(sn=" + user + ")(passwd=" + passwd + "))";
18
19        // This is the context in which you perform the query.
20        context = new InitialDirContext(environment);
21        NamingEnumeration<?> enmr = context.search(base, qry, sc);
22
23        // If there is more than one entry the authentication fails.
24        int count = 0;
25        while (enmr.hasMore() && enmr.next() != null)
26          if (++count > 1) break;
```

```
27
28          return count == 1;
29        }
30        // OMITTED: Handle exceptions properly.
31        // OMITTED: In finally block, close the context.
32      }
33    }
```

Questions:
1. What will the `authenticate` method do when executed?

2. Assuming that the `environment` field is properly initialized, and the directory server contains only two LDAP entries in the basename "`dc=example,dc=com`" with the following LDAP attributes (of type `String`):
LDAP Entry #1: `sn = "developer", passwd = "12345678"`
LDAP Entry #2: `sn = "administrator", passwd = "Se34*a=d!",`
which one statement is correct if the `authenticate` method gets called as:
`authenticate("dc=example,dc=com", "administrator", "Se*");`

a. The method returns `false`.
b. The method returns `true`.
c. The method throws a `RuntimeException` exception.
d. The method throws a `NamingException` exception.
e. None of the above.

[Other statistical questions will be imported here while creating the survey.]

NOTE: ANSWER IS TO BE SHOWN TO THE PUZZLE TAKER AT THE END OF THE SESSION.

ANSWER:
b
Using string concatenation to build an LDAP query makes it highly vulnerable to LDAP injection. In LDAP queries, the asterisk character is a wildcard (meaning it can be any sequence of characters). If the query (or some part of it) is coming from an untrusted source, the code must verify and sanitize it before sending it to the directory server. The above implementation builds the final query as "`(&(sn="administrator")(passwd="Se*"))`", meaning search for entries with the specified `sn` and `passwd` starting with "`Se`". This means anyone can get authenticated only by having the user account name, and without knowing the password (in fact, "`Se*`" can be simply replaced with "`*`" and the result is the same).

NOTE: THE REST OF THE DOCUMENT CONTAINS EXTRA INFORMATION FOR THE PROJECT RESEARCHERS. IT IS NOT TO BE SHOWN TO THE PUZZLE TAKERS.

TAGS:
java, ldap, ldap-injection, input-verification, input-sanitization

CATEGORIES:
Blindspot - YES
Type - LDAP
Number of distinct functions - 6
Number of total functions - 6
Blindspot function - `DirContext.search();`
Function call omitted - NO
Blindspot type - Incorrect usage
Number of Parameters in the blindspot function - 3 parameters
Cyclomatic complexity - 3

NAME:
LDAP queries - LDAP is a protocol for accessing and maintaining distributed directory information services over a network.

DESCRIPTION:
The Lightweight Directory Access Protocol (LDAP) is an open, vendor-neutral, industry standard application protocol for accessing and maintaining distributed directory information services over an Internet Protocol (IP) network. It allows an application to remotely perform operations, such as searching and modifying records in directories. A common usage of LDAP is to provide single-sign-on service where one user password is shared between many services.

BLINDSPOT:
The fact that, in LDAP queries (filters), the asterisk character is a special character (a wildcard, meaning it could be any sequence of characters) may cause a blindspot. The simplest way to build LDAP queries is string concatenation. However, this method of implementation is highly vulnerable to LDAP injection. If the query (or some part of it) comes from an untrusted source, the code building the final query must verify and sanitize the input values before it is submitted to the LDAP server.

CORRECT USE EXAMPLE:

Code:
```
import javax.naming.*;
import javax.naming.directory.*;
import java.util.Hashtable;
public class LdapInjection {
  private static final Hashtable<String, Object> env = new Hashtable<>();
  static {
    // Prepare the environment properties;
```

```
    env.put(Context.INITIAL_CONTEXT_FACTORY,
        "com.sun.jndi.ldap.LdapCtxFactory");
    env.put(Context.PROVIDER_URL, ldapAdServer);
        "ldap://ldap.example.com:389";
    env.put(Context.SECURITY_AUTHENTICATION, "simple");
    env.put(Context.SECURITY_PRINCIPAL, "administartor");
    env.put(Context.SECURITY_CREDENTIALS, "p@$$w0rd");
  }

  public static void main ( String... args ) throws NamingException {
    // Always verify input data coming from untrusted source;
    if (!args[1].matches("[\\w\\s]*") || !args[2].matches("[\\w]*"))
      throw new IllegalArgumentException("Invalid information!");
    if (authenticate(args[0], args[2], args[2]))
      throw new RuntimeException("Authentication failed!");
    // Continue the rest of the program;
  }

  public static boolean authenticate (String base, String user,
      String passwd) throws NamingException {
    DirContext context = null;
    context = new InitialDirContext(env);
    SearchControls sc = new SearchControls();
    // Initialize the search controls variable;
    // The following filter is used for authentication;
    String fltr = "(&(sn=" + user + ")(passwd=" + passwd + "))";
    NamingEnumeration<?> enmr = context.search(base, fltr, sc);
    // Continue the authentication logic;
  }
}
```

MORE INFORMATION:
#N/A

REFERENCES:
1. https://www.owasp.org/index.php/LDAP_injection