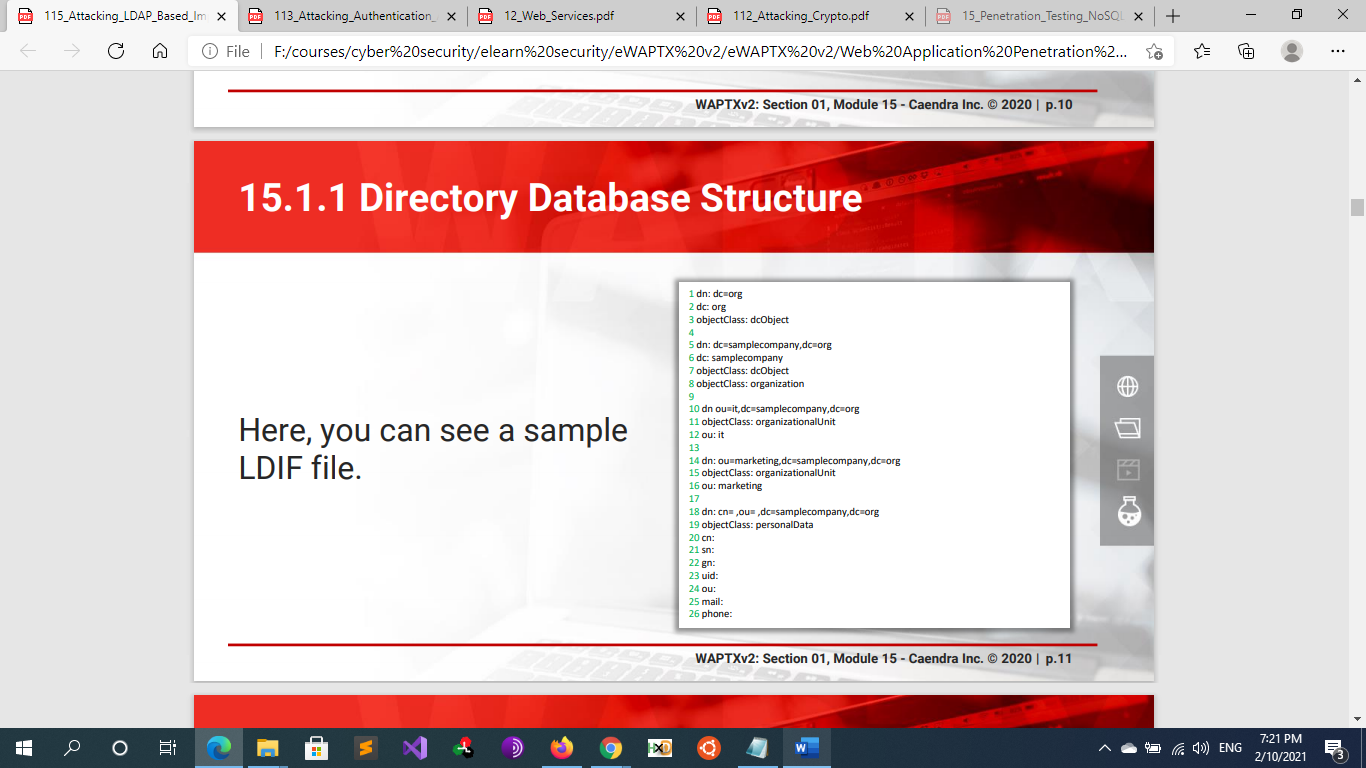
**LDAP injections**

* **Overview**
* **Detecting LDAP injection vulnerability**
* **LDAP injection Authentication Bypass**
* **Blind LDAP Injection**
* **Preventing LDAP injection**

**-------------------------------------------------------------------------------------------------------**

* **LDAP injections** 
  + **Overview**
    - **What is LDAP?**
      * Lightweight Directory Access Protocol (LDAP) is an open-standard protocol for both querying and manipulating X.500 directory services. The LDAP protocol runs over Internet transport protocols, such as TCP. (LDAP) is used to access directory services over a network.
        + Directory service is a database-like virtual storage that holds data in specific hierarchical structure. LDAP structure is based on a tree of directory of entries.
        + A directory is a hierarchically organized data store that may contain any kind of information but is commonly used to store personal data such as names, telephone numbers, e-mail addresses, and job functions.
      * LDAP is object oriented, thus every entry in an LDAP directory services is an instance of an object and must correspond to the rules fixed for the attributes of the object. LDAP can not only query objects from a directory database, it can also be used for management and authentication**. Note that LDAP is just the protocol to access Directory Service, not a storage mechanism itself**
      * Sample databases that use directory structure is Microsoft Active Directory (where LDAP is often used in authentication process) or the less known OpenLDAP.
        + You are most likely to encounter LDAP being used in corporate intranet-based web applications, such as an HR application that allows users to view and modify information about employees
    - **Directory Database Structure**
      * Objects in directory databases accessed via LDAP are stored in LDIF which stands for LDAP Data Interchange Format. LDIF defines directory content as a set of records, one record for each object (or entry). It also represents update requests, such as Add, Modify, Delete, and Rename, as a set of records, one record for each update request.
      * A directory database can support LDIF by defining its assumptions in a LDIF file. It can be a plaintext file simply containing directory data representation as well as LDAP commands. They are also used to read, write, and update data in a directory
      * **LDIF File Example:**
      * 
        + Lines 1-3: We are defining the toplevel domain "org".
        + Lines 5-8: Next, we are defining the subdomain “samplecompany", for example “samplecompany.org".
        + Lines 10-16: We define two organization units (ou): it and marketing.
        + Lines 18-26: We then add objects to the domain „samplecompany.org" and assign attributes with values

For example, „sn” stands for „surname”, „cn” stands for canonical name (or first name), while „mail” is a placeholder for an email address.

Each directory services database might have different default attributes..

* + - **What is LDAP Injection?**
      * LDAP Injection is an attack technique used to exploit web sites that construct LDAP statements from user-supplied input.
    - **LDAP Syntax**
      * LDAP as a protocol has its own structure for querying the back-end database. It utilizes operators like the following:
        + "=" (equal to)
        + | (logical or)
        + ! (logical not)
        + & (logical and)
        + \* (wildcard) – stands for any string or character
      * These operators are used in larger expressions (LDAP queries). Below you can find exemplary LDAP queries.
        + (cn=John) will fetch personal entries where canonical name is „John”
        + (cn=J\*) will fetch personal entries where canonical name starts with „J”, as a wildcard is placed in the query
    - **LDAP Search query filters** 
      * Each LDAP query uses one or more search filters, which determine the directory entries that are returned by the query. Search filters can use various logical operators to represent complex search conditions. The most common search filters you are likely to encounter are as follows:
      * **Simple match conditions** match on the value of a single attribute. For example, an application function that searches for a user via his username might use this filter:
        + (username=value)
      * **Disjunctive queries** specify multiple conditions, any one of which must be satisfied by entries that are returned. For example, a search function that looks up a user-supplied search term in several directory attributes might use this filter:
        + (|(cn=searchterm)(sn=searchterm)(ou=searchterm))
      * **Conjunctive queries** specify multiple conditions, all of which must be satisfied by entries that are returned. For example, a login mechanism implemented in LDAP might use this filter:
        + (&(username=daf)(password=secret))
  + **Detecting LDAP injection vulnerability**
    - Try entering just the \* character as a search term. This character functions as a wildcard in LDAP, but not in SQL. If a large number of results are returned, this is a good indicator that you are dealing with an LDAP query.
    - Try entering a number of closing brackets: )))))))))) This input closes any brackets enclosing your input, as well as those that encapsulate the main search filter itself. This results in unmatched closing brackets, thus invalidating the query syntax. If an error results, the application may be vulnerable to LDAP injection. (Note that this input may also break many other kinds of application logic, so this provides a strong indicator only if you are already confident that you are dealing with an LDAP query.
    - Try entering various expressions designed to interfere with different types of queries, and see if these allow you to influence the results being returned. The cn attribute is supported by all LDAP implementations and is useful to use if you do not know any details about the directory you are querying. For example:
      * )(cn=\*
      * \*))(|(cn=\*
      * \*))%00
  + **LDAP injection Authentication Bypass**
    - **Null bind Authentication Bypass**
      * Some LDAP servers authorize NULL Bind: if null values are sent, the LDAP server will proceed to bind the connection, and the PHP code will think that the credentials are correct. To get the bind with 2 null values, you will need to completely remove this parameter from the query or from the body of the message. If you keep something like username=&password= in the URL or the body if it’s a post request, these values will not work, since they won't be null; instead, they will be empty. So you need to send a request without the username=&password= in the request body or url to be a Null value
      * LDAP uses the wildcard \* character very often, to match any values. This can be used for match everything \* or just substrings (for example, **adm\*** for all words starting with **adm**).
    - **First step know the query structure by trying to generate an error**
      * \*)
      * )(cn=\*
      * \*))(|(cn=\*
      * \*))%00
        + We can get rid of the end of the filter, using a NULL BYTE (encoded as **%00**).
      * Ex : ERROR : Invalid LDAP syntax : (&(uid=\*))(userPassword=111))
      * The original query was
        + $filter =” (&(uid=”.$\_POST[‘uid’].”)(userPassword=”.$\_POST[‘pass’].”))”;
    - **Rebuild the query**
      * Once we knew the query we start to rebuild it to bypass the authentication
      * Username= \*)(|(userPassword=\*
      * Password= anything)
  + **Blind LDAP injection**
    - Suppose that an attacker can infer from the server responses, although the application does not show error messages, the code injected in the LDAP filter generates a valid response (true result) or an error (false result). The attacker could use this behavior to ask the server true or false questions. These types of attacks are named “Blind Attacks”. Blind LDAP Injection attacks are slower than classic ones but they can be easily implemented, since they are based on binary logic, and they let the attacker extract information from the LDAP Directory
    - To directly query an LDAP server, the attacker needs to know (or guess) the attribute names so they can be specified in a filter. Blind LDAP injection is a more advanced exploitation technique for extracting unknown information by sending multiple requests and checking server responses to determine if the query is valid. Combined with additional optimizations and automation, this allows attackers to obtain information using a series of yes/no questions: a valid server response means “yes”, and a server error means “no”. Effective blind injection
    - **Steps of Exploitation**
      * **Attribute discovery**
        + Attackers can query a variety of likely attributes and monitor server responses. If an attribute exists, the server will return a valid response. Otherwise, an error or empty response is returned. Let’s say an application unsafely constructs an AND filter to retrieve users, such as:

(&(userID=John Doe)(objectClass=user))

* + - * + If the attacker can manipulate the user ID value, they can inject code like the following to check if user objects in this directory have a department attribute:

(&(userID=John Doe)(department=\*))(objectClass=user))

* + - * + If the department attribute exists (and John Doe is a valid user ID), the server will return a valid response. Otherwise, the attacker can try other attribute names.
      * **Booleanization**
        + Once an attribute name is known, the attacker can send a series of requests containing wildcards and/or comparison operators to determine specific attribute values. Again, only two server responses are considered, so booleanization is the process of transforming the search process into a series of true/false tests.
        + Let’s say the department attribute from the previous example exists. To discover the department name, the attacker can start by injecting the following code to check the first letter:

(&(userID=John Doe)(department=a\*))(objectClass=user))

* + - * + A valid server response means that a department starting with the letter “a” exists. The attacker can continue the process for ab\*, ac\*, and so forth, to discover subsequent characters. For numeric values, the operators <= (less than or equal to) and >= (greater than or equal to) can be used to go through the likely value space
      * **Character set reduction**
        + To minimize the number of requests, attackers can use multiple wildcards to find out which characters are present anywhere in the target value. For example, a valid server response for the following injection:

(&(userID=John Doe)(department=\*x\*))(objectClass=user))

* + - * + means that a department name containing the letter “x” exists. If an error or empty response is returned, the attacker can eliminate this character from the scan. This can greatly reduce the number of requests needed to find the target value
    - **Resources**
      * https://www.netsparker.com/blog/web-security/ldap-injection-how-to-prevent/
  + **Preventing LDAP injection**
    - If it is necessary to insert user-supplied input into an LDAP query, this operation should be performed only on simple items of data that can be subjected to strict input validation. The user input should be checked against a white list of acceptable characters, which should ideally include only alphanumeric characters. Characters that may be used to interfere with the LDAP query should be blocked, including ( ) ; , \* | & = and the null byte. Any input that does not match the white list should be rejected, not sanitized.