



[dirkjanm / BloodHound.py](#) Public

Notifications

Fork 328

Star 1.9k

Code

Issues 23

Pull requests 8

Projects

Security

Insights

[BloodHound.py](#) / [bloodhound](#) / [ad](#) / [domain.py](#)

749 lines (659 loc) · 32.7 KB

Code

Blame

Raw

```
1 #####
2 #
3 # Copyright (c) 2018 Fox-IT
4 #
5 # Permission is hereby granted, free of charge, to any person obtaining a copy
6 # of this software and associated documentation files (the "Software"), to deal
7 # in the Software without restriction, including without limitation the rights
8 # to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
9 # copies of the Software, and to permit persons to whom the Software is
10 # furnished to do so, subject to the following conditions:
11 #
12 # The above copyright notice and this permission notice shall be included in all
13 # copies or substantial portions of the Software.
14 #
15 # THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
16 # IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
17 # FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
18 # AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
19 # LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
20 # OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
21 # SOFTWARE.
22 #
23 #####
24 from __future__ import unicode_literals
25 import logging
26 import traceback
```

```
27     import codecs
28     import json
29
30     from uuid import UUID
31     from dns import resolver
32     from ldap3 import ALL_ATTRIBUTES, BASE, SUBTREE, LEVEL
33     from ldap3.core.exceptions import LDAPKeyError, LDAPAttributeError, LDAPCursorError, LDAPNoSuchObjectError
34     from ldap3.protocol.microsoft import security_descriptor_control
35     # from impacket.krb5.kerberosv5 import KerberosError
36     from bloodhound.ad.utils import ADUtils, DNSCache, SidCache, SamCache, CollectionException
37     from bloodhound.ad.computer import ADComputer
38     from bloodhound.enumeration.objectresolver import ObjectResolver
39     from future.utils import intervalvalues, iteritems, native_str
40
41     """
42     Active Directory Domain Controller
43     """
44     class ADDC(ADComputer):
45         def __init__(self, hostname=None, ad=None):
46             ADComputer.__init__(self, hostname)
47             self.ad = ad
48             # Primary LDAP connection
49             self.ldap = None
50             # Secondary LDAP connection
51             self.resolverldap = None
52             # GC LDAP connection
53             self.gcldap = None
54             # Initialize GUID map
55             self.objecttype_guid_map = dict()
56
57         def ldap_connect(self, protocol='ldap', resolver=False):
58             """
59             Connect to the LDAP service
60             """
61             logging.info('Connecting to LDAP server: %s' % self.hostname)
62
63             # Convert the hostname to an IP, this prevents ldap3 from doing it
64             # which doesn't use our custom nameservers
65             q = self.ad.dnsresolver.query(self.hostname, tcp=self.ad.dns_tcp)
66             for r in q:
67                 ip = r.address
68
69             ldap = self.ad.auth.getLDAPConnection(hostname=self.hostname, ip=ip,
70                                                    baseDN=self.ad.baseDN, protocol=protocol)
71
72             if resolver:
73                 self.resolverldap = ldap
```

```
73         else:
74             self.ldap = ldap
75         return ldap is not None
76
77     def gc_connect(self, protocol='ldap'):
78         """
79         Connect to the global catalog
80         """
81         if self.hostname in self.ad.gcs():
82             # This server is a Global Catalog
83             initial_server = self.hostname
84         else:
85             # Pick the first GC server
86             try:
87                 initial_server = self.ad.gcs()[0]
88             except IndexError:
89                 logging.error('Could not find a Global Catalog in this domain!\'
90                             \' Resolving will be unreliable in forests with multiple domains')
91                 return False
92         try:
93             # Convert the hostname to an IP, this prevents ldap3 from doing it
94             # which doesn't use our custom nameservers
95             logging.info('Connecting to GC LDAP server: %s' % initial_server)
96             q = self.ad.dnsresolver.query(initial_server, tcp=self.ad.dns_tcp)
97             for r in q:
98                 ip = r.address
99         except (resolver.NXDOMAIN, resolver.Timeout):
100             for server in self.ad.gcs():
101                 # Skip the one we already tried
102                 if server == initial_server:
103                     continue
104                 try:
105                     # Convert the hostname to an IP, this prevents ldap3 from doing it
106                     # which doesn't use our custom nameservers
107                     logging.info('Connecting to GC LDAP server: %s' % server)
108                     q = self.ad.dnsresolver.query(server, tcp=self.ad.dns_tcp)
109                     for r in q:
110                         ip = r.address
111                         break
112                 except (resolver.NXDOMAIN, resolver.Timeout):
113                     continue
114
115         self.gcldap = self.ad.auth.getLDAPConnection(hostname=self.hostname, ip=ip, gc=True,
116                                                     baseDN=self.ad.baseDN, protocol=protocol)
117         return self.gcldap is not None
118
```

110


```
676         logging.warning('Could not find a global catalog server, assuming the primary D
677             'If this gives errors, either specify a hostname with -gc or di
678         self._gcs = self._dcs
679     else:
680         logging.warning('Could not find a global catalog server. Please specify one wit
681
682     try:
683         kquery = query.replace('pdc','dc').replace('_ldap','_kerberos')
684         q = self.dnsresolver.query(kquery, 'SRV', tcp=self.dns_tcp)
685         # TODO: Get the additional records here to get the DC ip immediately
686         for r in q:
687             kdc = str(r.target).rstrip('.')
688             logging.debug('Found KDC for enumeration domain: %s' % str(r.target).rstrip('.'))
689             if kdc not in self._kdcs:
690                 self._kdcs.append(kdc)
691                 self.auth.kdc = self._kdcs[0]
692     except resolver.NXDOMAIN:
693         pass
694
695     if self.auth.userdomain.lower() != ad_domain.lower():
696         # Resolve KDC for user auth domain
697         kquery = '_kerberos._tcp.dc._msdcs.%s' % self.auth.userdomain
698         q = self.dnsresolver.query(kquery, 'SRV', tcp=self.dns_tcp)
699         for r in q:
700             kdc = str(r.target).rstrip('.')
701             logging.debug('Found KDC for user: %s' % str(r.target).rstrip('.'))
702             self.auth.userdomain_kdc = kdc
703
704     return True
705
706
707 ✓ def get_domain_by_name(self, name):
708     for domain, entry in iteritems(self.domains):
709         if 'name' in entry['attributes']:
710             if entry['attributes']['name'].upper() == name.upper():
711                 return entry
712     # Also try domains by NETBIOS definition
713     for domain, entry in iteritems(self.nbdomains):
```



```
714         if domain.upper() == name.upper():
715             return entry
716     return None
717
718
719 ✓ def get_dn_from_cache_or_ldap(self, distinguishedname):
720     try:
721         linkentry = self.dncache[distinguishedname.upper()]
722     except KeyError:
723         use_gc = ADUtils.ldap2domain(distinguishedname).lower() != self.domain.lower()
724         qobject = self.objectresolver.resolve_distinguishedname(distinguishedname, use_gc=use_gc)
725         if qobject is None:
726             return None
727         resolved_entry = ADUtils.resolve_ad_entry(qobject)
728         linkentry = {
729             "ObjectIdentifier": resolved_entry['objectid'],
730             "ObjectType": resolved_entry['type'].capitalize()
731         }
732         self.dncache[distinguishedname.upper()] = linkentry
733     return linkentry
734
735     """
736     Active Directory Domain
737     """
738 ✓ class ADDomain(object):
739 ✓     def __init__(self, name=None, netbios_name=None, sid=None, distinguishedname=None):
740         self.name = name
741         self.netbios_name = netbios_name
742         self.sid = sid
743         self.distinguishedname = distinguishedname
744
745
746     @staticmethod
747     def fromLDAP(identifier, sid=None):
748         dns_name = ADUtils.ldap2domain(identifier)
749         return ADDomain(name=dns_name, sid=sid, distinguishedname=identifier)
```