BloodHound.py / bloodhound / ad / domain.py

dirkjanm Implement Kerberos auth method and NTLM fallback for SMB 56feb26 · 2 years ago History

```
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, LD
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, CollectionExcept
37  from bloodhound.ad.computer import ADComputer
38  from bloodhound.enumeration.objectresolver import ObjectResolver
39  from future.utils import itervalues, 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):
```

```
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          if resolver:
72              self.resolverldap = ldap
73          else:
74              self.ldap = ldap
75          return ldap is not None
```

```
                                    generator=True)

        entriesNum = 0
        for entry in entries:
            # Ensure systemFlags entry is not empty before running the naming context c
            if not entry['attributes']['systemFlags']:
                continue
            # This is a naming context, but not a domain
            if not entry['attributes']['systemFlags'] & 2:
                continue
            entry['attributes']['distinguishedName'] = entry['attributes']['nCName']
            entriesNum += 1
            # Todo: actually use these objects instead of discarding them
            # means rewriting other functions
            d = ADDomain.fromLDAP(entry['attributes']['nCName'])
            # We don't want to add our own domain since this entry doesn't contain the
            # which we need later on
            if entry['attributes']['nCName'] not in self.ad.domains:
                self.ad.domains[entry['attributes']['nCName']] = entry
                self.ad.nbdomains[entry['attributes']['nETBIOSName']] = entry

        # Store this number so we can easily determine if we are in a multi-domain
        # forest later on.
        self.ad.num_domains = entriesNum
        logging.info('Found %u domains in the forest', entriesNum)

    def get_cache_items(self):
        self.get_objecttype()
        self.get_forest_domains()
        self.gc_connect()
        sidcache = {}
        dncache = {}
        for nc, domain in self.ad.domains.items():
            logging.info('Processing domain %s', domain['attributes']['name'])
            query = '(|(&(objectCategory=person)(objectClass=user))(objectClass=group)(
            entries = self.search(query,
                                  use_gc=True,
                                  use_resolver=True,
                                  attributes=['sAMAccountName', 'distinguishedName', 's
                                  search_base=nc,
                                  generator=True)
            for lentry in entries:
                resolved_entry = ADUtils.resolve_ad_entry(lentry)
                cacheitem = {
                    "ObjectIdentifier": resolved_entry['objectid'],
                    "ObjectType": resolved_entry['type'].capitalize()
                }
                sidcache[resolved_entry['objectid']] = cacheitem
```

```
356                     dncache[ADUtils.get_entry_property(lentry, 'distinguishedName').upper()
357             return dncache, sidcache
358
359         def get_groups(self, include_properties=False, acl=False):
360             properties = ['distinguishedName', 'samaccountname', 'samaccounttype', 'objects
361             if include_properties:
362                 properties += ['adminCount', 'description', 'whencreated']
363             if acl:
364                 properties += ['nTSecurityDescriptor']
365             entries = self.search('(objectClass=group)',
366                                   properties,
367                                   generator=True,
368                                   query_sd=acl)
369             return entries
370
371         def get_gpos(self, include_properties=False, acl=False):
372             properties = ['distinguishedName', 'name', 'objectGUID', 'gPCFileSysPath', 'dis
373             if include_properties:
374                 properties += ['description', 'whencreated']
375             if acl:
376                 properties += ['nTSecurityDescriptor']
377             entries = self.search('(objectCategory=groupPolicyContainer)',
378                                   properties,
379                                   generator=True,
380                                   query_sd=acl)
381             return entries
382
383         def get_ous(self, include_properties=False, acl=False):
384             properties = ['distinguishedName', 'name', 'objectGUID', 'gPLink']
385             if include_properties:
386                 properties += ['description', 'whencreated']
387             if acl:
388                 properties += ['nTSecurityDescriptor']
389             entries = self.search('(objectCategory=organizationalUnit)',
390                                   properties,
391                                   generator=True,
392                                   query_sd=acl)
393             return entries
394
395         def get_containers(self, include_properties=False, acl=False, dn=''):
396             properties = ['distinguishedName', 'name', 'objectGUID', 'isCriticalSystemObjec
397             if include_properties:
398                 properties += ['description', 'whencreated']
399             if acl:
400                 properties += ['nTSecurityDescriptor']
401             entries = self.search('(&(objectCategory=container)(objectClass=container))',
402                                   properties,
403                                   generator=True,
404                                   query_sd=acl,
405                                   search_base=dn)
406             return entries
407
408         def get_users(self, include_properties=False, acl=False):
```

Files

d65eb61

Go to file

- bloodhound
  - ad
    - __init__.py
    - authentication.py
    - computer.py
    - domain.py
    - dumpntlm.py
    - structures.py
    - trusts.py

BloodHound.py / bloodhound / ad / domain.py ↑ Top

Code | Blame | 749 lines (659 loc) · 32.7 KB | Raw

```
395         def get_containers(self, include_properties=False, acl=False, dn=''):
414
415             if include_properties:
416                 properties += ['servicePrincipalName', 'userAccountControl', 'displayName',
417                                'lastLogon', 'lastLogonTimestamp', 'pwdLastSet', 'mail', 'ti
418                                'description', 'userPassword', 'adminCount', 'msDS-AllowedTo
419                                'whencreated', 'unicodepwd', 'scriptpath']
420                 if 'unixuserpassword' in self.objecttype_guid_map:
421                     properties.append('unixuserpassword')
422             if acl:
423                 properties.append('nTSecurityDescriptor')
424
425             # Query for GMSA only if server supports it
426             if 'msDS-GroupManagedServiceAccount' in self.ldap.server.schema.object_classes:
427                 query = '(|(&(objectCategory=person)(objectClass=user))(objectClass=msDS-Gr
428             else:
429                 logging.debug('No support for GMSA, skipping in query')
430                 query = '(&(objectCategory=person)(objectClass=user))'
```

utils.py
enumeration
lib
__init__.py
__main__.py
.editorconfig
.gitignore
Dockerfile
LICENSE
README.md
bloodhound.py
createforestcache.py
setup.py

```
            query = "(&(objectCategory=person)(objectClass=user))"
        entries = self.search(query,
                              properties,
                              generator=True,
                              query_sd=acl)
        return entries


    def get_computers(self, include_properties=False, acl=False):
        """
        Get all computer objects. This purely gets them using LDAP. This function is us
        or used to create a cache in case of computer enumeration later on.
        """
        properties = ['samaccountname', 'userAccountControl', 'distinguishedname',
                      'dnshostname', 'samaccounttype', 'objectSid', 'primaryGroupID',
                      'isDeleted']
        if include_properties:
            properties += ['servicePrincipalName', 'msDS-AllowedToDelegateTo', 'sIDHist
                           'lastLogon', 'lastLogonTimestamp', 'pwdLastSet', 'operatingS
                           'operatingSystemServicePack']
            # Difference between guid map which maps the lowercase schema object name a
            if 'ms-DS-Allowed-To-Act-On-Behalf-Of-Other-Identity'.lower() in self.objec
                properties.append('msDS-AllowedToActOnBehalfOfOtherIdentity')
            if self.ad.has_laps:
                properties.append('ms-mcs-admpwdexpirationtime')
        if acl:
            # Also collect LAPS expiration time since this matters for reporting (no LA
            if self.ad.has_laps:
                properties += ['nTSecurityDescriptor', 'ms-mcs-admpwdexpirationtime']
            else:
                properties.append('nTSecurityDescriptor')
        entries = self.search('(&(sAMAccountType=805306369))',
                              properties,
                              generator=True,
                              query_sd=acl)

        return entries

    def get_computers_withcache(self, include_properties=False, acl=False):
        """
        Get all computer objects. This is the function used when we do computer enumera
        items are cached first. Calls the function with generator above to create the l
        """

        entries = self.get_computers(include_properties, acl)

        entriesNum = 0
        for entry in entries:
            entriesNum += 1
            # Resolve it first for DN cache
            resolved_entry = ADUtils.resolve_ad_entry(entry)
            cacheitem = {
                "ObjectIdentifier": resolved_entry['objectid'],
                "ObjectType": resolved_entry['type'].capitalize()
            }
            self.ad.dncache[ADUtils.get_entry_property(entry, 'distinguishedName', '').
            # This list is used to process computers later on
            self.ad.computers[ADUtils.get_entry_property(entry, 'distinguishedName', ''
            self.ad.computersidcache.put(ADUtils.get_entry_property(entry, 'dNSHostname

        logging.info('Found %u computers', entriesNum)

        return entries

    def get_memberships(self):
        entries = self.search('(|(memberof=*)(primarygroupid=*))',
                              ['samaccountname', 'distinguishedname',
                               'dnshostname', 'samaccounttype', 'primarygroupid',
                               'memberof'],
                              generator=False)
        return entries

    def get_sessions(self):
        entries = self.search('(&(samAccountType=805306368)(!(userAccountControl:1.2.84
                              ['homedirectory', 'scriptpath', 'profilepath'])
```

```
505             return entries
506
507         def get_childobjects(self, dn, use_resolver=True):
508             entries = self.search('(|(objectClass=container)(objectClass=organizationalUnit
509                                   attributes=['objectSid', 'objectClass', 'objectGUID', 'di
510                                   search_base=dn,
511                                   search_scope=LEVEL,
512                                   use_resolver=use_resolver)
513
514             return entries
515
516         def get_trusts(self):
517             entries = self.search('(objectClass=trustedDomain)',
518                                   attributes=['flatName', 'name', 'securityIdentifier', 'tr
519                                   generator=True)
520             return entries
521
522         def prefetch_info(self, props=False, acls=False, cache_computers=False):
523             self.get_objecttype()
524             self.get_domains(acl=acls)
525             self.get_forest_domains()
526             if cache_computers:
527                 self.get_computers_withcache(include_properties=props, acl=acls)
528
529         def get_root_domain(self):
530             return ADUtils.ldap2domain(self.ldap.server.info.other['configurationNamingCont
531
532
533     """
534     Active Directory data and cache
535     """
536     class AD(object):
537
538         def __init__(self, domain=None, auth=None, nameserver=None, dns_tcp=False, dns_time
539             self.domain = domain
540             # Object of type ADDomain, added later
541             self.domain_object = None
542             self.auth = auth
543             # List of DCs for this domain. Contains just one DC since
544             # we query for the primary DC specifically
```

```
676                     logging.warning('Could not find a global catalog server, assuming t
677                                     'If this gives errors, either specify a hostname wi
678                     self._gcs = self._dcs
679                 else:
680                     logging.warning('Could not find a global catalog server. Please spe
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).rs
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.lowe
724                 qobject = self.objectresolver.resolve_distinguishedname(distinguishedname,
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)
```