BTA : outil open-source d'analyse AD

Philippe Biondi et Joffrey Czarny
philippe.biondi@eads.net
joffrey.czarny@eads.net

Airbus Group Innovations

Résumé La plupart des entreprises, petites ou grosses, utilisent les technologies Microsoft pour fonctionner. L'Active Directory est au cœur de ces systèmes d'information.

Lorsqu'il est question de la sécurité du système d'information, les contrôleurs de domaine Active Directory sont, ou devraient être, au centre des préoccupations, qui sont en temps normal de s'assurer du respect des bonnes pratiques et, lors d'une compromission avérée, d'explorer la possibilité d'un nettoyage du système d'information sans avoir à reconstruire l'Active Directory. Or, peu d'outils permettent de mettre en œuvre cette démarche.

Nous présentons ici BTA, un outil d'audit de bases Active Directory ainsi que notre méthodologie permettant de vérifier l'application des bonnes pratiques et l'absence de modifications malveillantes dans ces bases.

Nous commencerons par décrire les enjeux autour de l'Active Directory, pièce maîtresse de tout système d'information fondé sur les technologies Microsoft et expliciterons le besoin opérationnel amené par ces enjeux.

Nous présenterons et détaillerons ensuite l'outil libre BTA [2] développé pour répondre à ce besoin.

Nous exposerons la méthodologie d'analyse Active Directory que nous utilisons, articulée autour de BTA.

Nous conclurons sur un retour d'expérience.

1 Introduction

1.1 Enjeux

La plupart des entreprises, petites ou grosses, utilisent les technologies Microsoft pour fonctionner. L'Active Directory est au cœur de ces systèmes d'information. Il permet notamment l'authentification des machines et des utilisateurs, assigne et met en application les politiques de sécurité et est utilisé comme base de référence par la plupart des autres applications. De plus certains outils, comme la messagerie (Exchange) ou le partage d'informations (SharePoint), s'intègrent dans l'Active Directory et se

reposent sur lui notamment pour les problématiques d'authentification et d'autorisation.

Lorsqu'il est question de la sécurité du système d'information, les contrôleurs de domaine Active Directory sont, ou devraient être, au centre des préoccupations, qui sont en temps normal de s'assurer du respect des bonnes pratiques et, lors d'une compromission avérée, d'explorer la possibilité d'un nettoyage du système d'information sans avoir à reconstruire l'Active Directory. Or, peu d'outils permettent de mettre en œuvre cette démarche.

Il est essentiel de maîtriser l'ensemble des objets Active Directory, mais également Exchange et SharePoint, et des permissions qui leurs sont attribuées.

Ces dernières années, bon nombre d'entreprises ont malheureusement dû, après une compromission, répondre également à une question : *Dois-je reconstruire mon Active Directory ?* L'obtention des droits d'administration de l'Active Directory étant l'un des objectifs d'une attaque sophistiquée, il est tout à fait envisageable qu'un attaquant tente de se maintenir dans le système en se réservant une ou plusieurs portes dérobées dans celui-ci, par exemple sous forme de privilèges lui permettant de redevenir administrateur de domaine à partir de comptes plus exposés. Il est donc naturel de mettre en cause l'intégrité des bases Active Directory. Cependant, le coût d'une reconstruction d'un AD de zéro est tel que le simple principe de précaution ne suffit pas toujours à justifier cette opération.

1.2 Besoin

Le besoin est donc d'évaluer rapidement l'état d'un Active Directory. Cela inclut, entre autres,

- identifier les résultat des mauvaises pratiques, qui affaiblissent le niveau de sécurité du système d'information (ex : droits administrateurs sur les comptes personnels) ;
- aider au nettoyage (mauvaises pratiques, comptes à l'abandon...) ;
- trouver les anomalies pouvant évoquer une porte dérobée laissée par un attaquant (droits étendus, délégations inexplicables...).

La démarche consiste à passer en revue un grand nombre de points de contrôle tels que :

- la liste des administrateurs de domaine ;
- les délégations ;
- les comptes qui n'ont jamais servi ;
- tous les types de droits pouvant être accordés, en visant l'exhaustivité.

Il est souhaitable de limiter au maximum les interactions avec l'Active Directory, qui est généralement en production. Cependant un grand nombre de requêtes devront être effectuées pour vérifier tous ces points. Il faudra donc privilégier un fonctionnement hors-ligne.

Il est également intéressant de pouvoir passer en revue les différences entre deux photographies d'un même AD à deux moments, par exemple avant et après une compromission.

1.3 Outils existants

L'outil ADperm[1], développé par l'ANSSI, propose également d'extraire les bases des Active Directory afin de les examiner. Mais son approche est exploratoire. L'outil permet de naviguer dans les bases en suivant les relations entre les objets. Cette approche ne répond pas directement au besoin. Elle est plus adaptée pour des opérations de test intrusif, d'analyse post-mortem ou tout autre analyse où on ne sait pas à l'avance ce qu'on cherche et où on se laisse guider par ce qu'on trouve. Elle permet en outre de trouver des points de contrôles intéressants que l'on peut ensuite cristalliser dans BTA. En ce sens, nous pouvons comparer ADperm à un *debugger* et BTA à un framework de tests unitaires et de non-régression. L'approche de ADperm est donc complémentaire de celle de BTA.

Microsoft a développé son propre outil, utilisé uniquement lors de prestations de ses propres consultants (ADSA : Active Directory Security Assessment).

D'autres outils existent et utilisent généralement l'accès LDAP pour donner une vision un peu plus haut niveau que les outils Microsoft tels que `Ldp.exe` et `Dsacl.exe`.

2 BTA

2.1 Présentation

L'outil BTA a donc été développé afin de pouvoir extraire et analyser l'ensemble des informations d'un AD. Il est développé en Python, utilise le moteur MongoDB pour la base de données et tourne sous Linux.

Il est conçu autour du concept de *miners*, qui sont des petits bouts de code ayant chacun le rôle de vérifier un point de contrôle ou une famille de points de contrôle.

BTA transfère chaque base Active Directory dans une base de données. Les *miners* analysent les informations contenues dans cette base de données et génèrent un rapport sous forme d'une structure de données hiérarchique

sommaire, pouvant contenir sections, listes et tables. Ces rapports peuvent être combinés en un seul document. Enfin, ce document peut être traduit en un format de sortie tel que *RestructuredText*, Excel ou CSV.

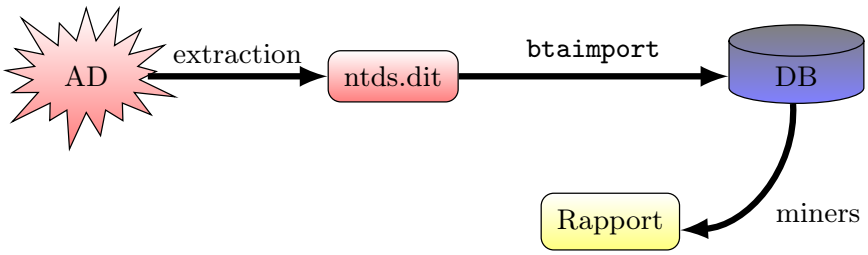

Figure 1. Architecture de BTA, vue de haut

BTA propose également un outil permettant de comparer deux bases de données contenant le même Active Directory à deux instants différents.

Le *framework* est composé de 4 outils :

btaimport Importe la base `NTDS.dit` dans une base MongoDB et post-processing.

btamanage Effectue l'intendance sur les bases NTDS importées (lister, effacer...).

btaminer Appelle les *miners* permettant de requêter les données et propriétés des objets.

btadiff Effectue un différentiel entre deux instances de l'AD intégrées en base.

2.2 Choix techniques

Lecture du fichier `NTDS.dit` Le contenu de la base AD est accessible soit à travers le protocole LDAP, soit à travers l'API d'une bibliothèque Windows [1], soit en effectuant une sauvegarde de son contenu dans un fichier nommé `NTDS.dit`.

Nous avons vu qu'un travail hors-ligne était généralement préférable, aussi nous avons choisi la troisième solution. La procédure de récupération de ce fichier est précisée plus loin, section 3.1.

Le format du fichier `NTDS.dit` est extrêmement complexe [4]. Il n'était pas raisonnable d'implémenter nous-mêmes son analyse puisque cela était déjà fait par la bibliothèque `libesedb` [5]. Un *wrapper* Python a été implémenté pour l'occasion.

Base de donnée Le fichier `NTDS.dit` est une base ESE[1] contenant plusieurs tables telles qu'on peut les imaginer classiquement dans une base de données relationnelle. Cependant, la `libesedb` ne nous permet pas de la requêter comme une base de données, par exemple en recherchant toutes les lignes ayant un champ égal à une valeur donnée. Il est bien possible de le faire à la main, mais les performances seraient médiocres et nous ne profiterions pas d'éventuels index.

Nous utilisons donc la `libesedb` pour lire la totalité des données et les importer dans une base de données que nous savons requêter.

Cependant, une des tables, la `datatable` est généralement constituée de plusieurs milliers de colonnes, ce qui rend impossible l'import dans une base de données relationnelle classique. Ce nombre élevé de colonnes est dû au fait que beaucoup d'applications vont stocker leurs données relatives aux authentifications et aux autorisations dans cette base en rajoutant leurs colonnes (extensions de schéma), qui ne seront utilisées que par les lignes correspondant aux objets gérés par ces applications. Ainsi, la table est creuse et chaque ligne utilise en moyenne une trentaine de colonnes sur les milliers existantes.

Nous avons choisi de nous orienter vers une base de données orientée document, qui ne travaille pas sur des tables mais sur des collections de documents. Chaque document correspond à une ligne et chaque colonne utilisée à un champ du document. Ainsi, nous n'avons pas à expliciter les colonnes non utilisées. Nous avons choisi *MongoDB* simplement car c'était le moteur que nous connaissions le mieux.

Les appels à la base de données ont été abstraits pour permettre un changement à moindre coût, si nécessaire.

2.3 Les informations en entrée

L'ensemble des objets stockés par l'Active Directory est conservé dans le fichier `NTDS.dit`. Ce dernier contient plusieurs tables : `datatable`, `sd_table`, `link_table`... La table `datatable` contient les attributs pour les objets ; la table `sd_table` contient toutes les ACE[2] positionnés sur les objets.

1. *Extended Storage Engine.*
2. Access Control Entry (ACE) ou entrée de contrôle d'accès donnant ou supprimant des droits d'accès à une personne ou un groupe.

La phase d'import consiste à transformer chaque table en une collection [3]. Pour chaque fichier `NTDS.dit`, nous créerons une base contenant une collection par table, ainsi que quelques collections de méta-données.

Cette phase est assez complexe. La `datatable` est un cas particulier. Pour les autres tables, nous connaissons à l'avance les colonnes qui les constituent, leur nom, ainsi que le type de données que nous allons y trouver. Pour la `datatable`, le nombre de colonnes est variable et leur nom est généralement un nom abscons tel que `ATTk135213`. Cependant, chaque colonne est décrite par une ligne dans cette même table. De cette ligne nous pouvons extraire le type, le *common name* ou encore le *LDAP display name* [4] relatifs à cette colonne. Ce dernier fera un bien meilleur nom de colonne. La ligne correspondant à une colonne peut être retrouvée grâce à son *attribute id*. En effet, ce dernier correspond au nombre finissant le nom de la colonne `ATTk135213`. Il est également stocké dans la colonne `ATTc131102` des lignes décrivant les colonnes. Il suffit donc de parcourir cette dernière colonne jusqu'à trouver une ligne ayant l'*attribute id* recherché, ici 135213, et de récupérer dans cette ligne l'*attribute syntax* (colonne `ATTc131104`) et le *LDAP display name* (colonne `ATTm131532`). Cette gymnastique est expliquée plus en détails dans [1].

Ensuite, pour chaque table, toutes les lignes sont lues. Chaque cellule peut être absente, simple ou multi-valuée et encodée ou non. Selon le type, un décodage supplémentaire peut être nécessaire, par exemple pour les SID ou pour les cellules contenant une structure de donnée complexe. Enfin, une dernière traduction peut être nécessaire pour un stockage efficace dans la base de donnée.

L'outil `btaimport` se charge de tout cela. Il suffit au minimum de lui fournir le chemin vers le fichier `NTDS.dit` et l'adresse de la base avec le paramètre `-C`. Cette dernière prend la forme `[IP]:[port]:dbname`, IP et port prenant par défaut les valeurs respectives 127.0.0.1 et 27017 [5]. Ainsi, pour un serveur MongoDB en local, il suffit de lancer :

```
$ btaimport -C ::mabase /chemin/vers/mon/ntds.dit
```

L'opération est relativement longue et peut prendre plusieurs heures.

Dans le cas de plusieurs AD à auditer, pour une forêt d'AD par exemple, nous devons importer plusieurs fichiers. `btaimport` peut travailler sur plusieurs fichiers, du moment qu'on lui fournit, pour chaque fichier,

3. La collection pour MongoDB est l'équivalent de la table pour une base de données relationnelle.

4. Le nom utilisé pour l'interface LDAP.

5. Le port d'écoute par défaut de MongoDB.

l'adresse d'une base. Cela peut se faire en utilisant à la place de `-C` soit l'option `--C-list` soit l'option `--C-from-filename`.

Dans le premier cas, nous avons :

```
$ btaimport  --C-list ::mabase1,::mabase2  \
             ad/base1.ntds.dit ad/base2.ntds.dit
INFO : Going to import ::mabase1       <- ad/base1.ntds.dit
INFO : Going to import ::mabase2       <- ad/base2.ntds.dit
Can I carry on ? (y/n)
```

Dans le deuxième cas, nous devons fournir un programme en notation polonaise inverse qui, appliqué au chemin de chaque fichier, fourni le nom de la base et est appliqué à une chaîne de format pour former l'adresse de la base. Par exemple :

```
$ btaimport  --C-from ::%s "basename rmallext" ad/*dit
INFO : Going to import ::backdoor1  <- ad/backdoor1.ntds.dit
INFO : Going to import ::backdoor2  <- ad/backdoor2.ntds.dit
INFO : Going to import ::clean      <- ad/clean.ntds.dit
Can I carry on ? (y/n)
```

Une description plus détaillée ainsi que la liste des instructions du langage est disponible dans l'annexe A.

Enfin, avec l'option `--multi`, `btaimport` pourra importer plusieurs fichiers en parallèle en utilisant un cœur par fichier. Le CPU étant le facteur limitant sur une machine correctement dimensionnée (*cf.* section 4.1) si l'import d'une base dure 4h, l'import de 4 bases de même taille durera également 4h.

2.4 Les *miners*

Les *miners* sont chargés de vérifier un ou plusieurs points de contrôle et de fournir un rapport sur ce qu'ils ont trouvé. Ils peuvent également appeler d'autres *miners* pour automatiser le contrôle de plusieurs points.

Un *miner* est un objet héritant de la classe `Miner`. Les 5 attributs et méthodes à surcharger sont

- l'attribut `_name_` donnant le nom officiel du *miner* ;
- l'attribut `_desc_` donnant une description succincte ;
- la méthode de classe `create_arg_subparser()` destinée à rajouter les paramètres de la ligne de commande propres au *miner* ;
- la méthode `assert_consistency()` dont le rôle est de vérifier la cohérence de la base sur laquelle le *miner* va travailler et éviter par exemple un faux négatif parce que la base était vide ;

– la méthode `run()` qui va être appelée par `btaminer` une fois les arguments de la ligne de commande analysés, la base de données ouverte et sa cohérence vérifiée, pour lancer les tests.

L'objet *miner* Voici le squelette d'un *miner* minimal ajoutant des options de ligne de commande, ajoutant des tests de cohérence, requêtant la `sd_table` et emplissant le rapport avec le résultat de cette requête ainsi que des valeurs récupérées sur la ligne de commande :

```
from bta.miner import Miner


@Miner.register
class Skel(Miner):
    _name_ = "skeleton"
    _desc_ = "skeleton, list SD id and hashes when id < 50"
    @classmethod
    def create_arg_subparser(cls, parser):
        parser.add_argument("--dummy", help="dummy option")
        parser.add_argument("--dummy_flag", help="dummy flag",
            action="store_true")

    def run(self, options, doc):
        doc.add("Option dummy is %s" % options.dummy)

        table = doc.create_table("my table")
        table.add(["id","hash"])
        table.add()

        for r in self.sd_table.find({"sd_id": {"$lt":50}}):
            table.add([r["sd_id"],r["sd_hash"]])
        table.finished()

    def assert_consistency(self):
        Miner.assert_consistency(self)
        self.assert_field_exists(self.sd_table, "sd_id")
        assert self.datatable.find({"cn":{"$exists":True}}).count()
            > 10, "less than 10 cn in datatable"
```

Les *miners* sont appelés à l'aide de l'outil `btaminer`. Ce dernier va

1. charger tous les *miners* ;
2. les enregistrer en tant que sous-commandes avec leurs paramètres dans l'objet `ArgumentParser` en appelant la méthode de classe `create_arg_subparser()` ;
3. analyser la ligne de commande ;
4. se connecter à la base de données et ouvrir la base choisie ;
5. instancier le *miner* choisi, dont le constructeur va créer tous les attributs référençant les collections utiles aux *miner* : `datatable`, `sd_table`, etc. ;

6. appeler sa méthode `assert_consistency()` ;
7. préparer une structure de document ;
8. appeler sa méthode `run()`.

Requêter la base de données Le but d'un *miner* va être d'extraire certaines données de la base de données MongoDB. Pour définir les requêtes adéquates, la connaissance de la structure des bases d'un AD[1,3] est indispensable .

En guise d'exemple, nous allons essayer de récupérer les membres du group *Admins du domaine*. Ce groupe a le SID *S-1-5-21*domaine-*512*.

Il faut d'abord trouver la valeur de l'attribut *DNT_col* de l'objet correspondant au groupe recherché. Nous allons donc retourner les champs *DNT_col* et *cn* de toute ligne ayant un champ *objectSid* valant *S-1-5-21*domaine-*512*

```
self.datatable.find(
  {"objectSid":"S-1-5-21-1154669122-758131934-2550385761-512"},
  {"DNT_col":1 , "cn":1}
)
{
 "cn" : "Admins du domaine",
 "DNT_col" : 3617
}
```

Une fois cette valeur retournée, il faut chercher les entrées disposant de cette valeur pour l'attribut *link_DNT* dans la table `link_table`. Le résultat de la recherche nous donnera les membres de ce groupe via la valeur de l'attribut *backlink_DNT*

```
self.link_table.find({"link_DNT":3617},{"link_DNT":1, "backlink_DNT":1})
{ "link_DNT" : 3617, "backlink_DNT" : 3563 }
{ "link_DNT" : 3617, "backlink_DNT" : 8789 }
```

Enfin, pour connaitre le nom des membres, il suffit de rechercher la ligne de la `datatable` dont l'attribut *DNT_col* est égal aux valeurs de *backlink_DNT* trouvées.

```
self.datatable.find({"DNT_col":3563})
{ "cn" : "Administrateur",
  "objectSid" : "S-1-5-21-1154669122-758131934-2550385761-500"}

self.datatable.find({"DNT_col":8789},{cn:1} )
{ "cn" : "snorky",
  "objectSid" : "S-1-5-21-1154669122-758131934-2550385761-1154"}
```

Dans cette exemple nous obtenons donc deux utilisateurs *Administrateur* et *snorky* qui sont membres du groupe *Admins du domaine*.

Toutes ces étapes sont donc automatisables et il est possible de rajouter d'autres informations ou filtres permettant d'affiner les résultats. Le rôle du *miner* va donc être d'exécuter ces requêtes et de les mettre en forme dans un rapport. Le *miner ListGroup* effectue ce travail (*cf.* section 3.3).

2.5 Le rapport

Chaque *miner* va consigner ce qu'il a trouvé dans une structure de données hiérarchique constituée de sections, de texte, de tables et de listes. Il est appelé avec ce qui sera pour lui sa racine et va y ajouter ses données, mais cette racine peut être une sous-section d'une structure plus grande.

Cette structure de données a deux modes de fonctionnement : une sortie libre au fur et à mesure que l'analyse s'effectue et une sortie formatée effectuée une fois l'analyse complète.

Dans le premier mode, la sortie est forcément textuelle et pas forcément correctement formatée. En effet, il est par exemple impossible de calculer la bonne largeur pour les colonnes d'une table avant d'avoir récupéré toutes les lignes, aussi des heuristiques sont utilisées.

Dans le deuxième mode, aucune sortie n'est effectuée avant que l'analyse ne soit complète. Puis la structure est transformée dans un format exploitable. Pour l'instant, les formats disponibles sont :

- une sortie reStructuredText ;
- une sortie Excel proposant le document sans les tables ou listes sur une première feuille, puis une feuille par table ou liste ;
- une sortie sous forme d'un fichier zip reprenant les sections document sous forme de sous-répertoires, les parties textuelles sous forme de fichiers textes et les listes et tables sous forme de fichiers CSV.

Pour illustrer la manipulation de cette structure, voici un exemple de méthode `run()` d'un *miner*. La racine de la structure du document pour le *miner* est récupérée dans l'argument `doc`. Nous voyons que nous pouvons ajouter du contenu avec la méthode `add()` et que nous pouvons créer des sous-sections (`create_subsection()`), des tables (`create_table()`) et des listes (`create_list()`).

```
    def run(self, options, doc):
        doc.add("Quelques lignes d'introduction du miner")
        doc.add("Remarques pertinentes sur le contenu de la base")

        s1 = doc.create_subsection("Premiere partie de l'analyse")
        s2 = doc.create_subsection("Deuxieme partie de l'analyse")
        table = s1.create_table("ma table")
```

```
        lst = s2.create_list("ma liste")
        table.add(["decimal","hexadecimal"])
        table.add()
        for i in range(15):
            table.add(["%i"%i, "%x"%i])
        table.finished()
        s1.finished()
        s2.add("Je sais aussi compter a l'envers")
        for i in range(20,0,-1):
            lst.add("%i"%i)
        lst.finished()
        s2.finished()
```

Nous pouvons noter qu'il est possible de créer la deuxième sous-section avant de finir de remplir la première. Cela ne pose aucun souci pour une sortie formatée qui attend la fin de l'analyse. Cependant, pour la sortie au fil de l'analyse, il est nécessaire d'explicitement marquer le fait qu'une sous-structure est complète pour que la sortie puisse passer à la suivante. Cela est fait avec la méthode `finished()`. Dans cet exemple, tant que cette méthode n'est pas appelée sur `s1` et `table`, la sortie de `s2` ne peut pas commencer.

2.6 Confiance dans les résultats

Traçabilité Chaque base de données BTA contient une collection contenant les logs de toutes les modifications effectuées, notamment au moment de l'import. Cela permet de tracer en particulier les dates d'import, les options utilisées, les éventuelles erreurs et les éventuelles réécritures, ainsi que les commandes utilisées pour peupler la base, ou encore de vérifier que la base est bien celle que l'on croit.

La façon d'interpréter certains champs lors de l'import peut évoluer au fur et à mesure du développement du logiciel. Pour garantir que les *miners* seront toujours utilisés sur des bases importées par un `btaimport` compatible, un numéro de version est également stocké et vérifié par les *miners*.

Le *miner* `info` peut afficher ces méta-données :

```
$ btaminer -C ::test -t ReST info

Analysis by miner [info]
========================

Data format version: 1


collections in this database
----------------------------

+-----------------+-------------------+
| name            | number of records |
+=================+===================+
```

```
| system.indexes  | 50                |
| metadata        | 1                 |
| log             | 3                 |
| sd_table        | 94                |
| sd_table_meta   | 4                 |
| link_table      | 54                |
| link_table_meta | 10                |
| datatable       | 3516              |
| datatable_meta  | 1182              |
| category        | 234               |
| domains         | 1                 |
| dnames          | 3515              |
| usersid         | 12                |
| guid            | 6538              |
+-----------------+-------------------+

logs
----

  + logs:
    - 2014-04-13 11:43:07.737000: [u'/v/bta/bin/btaimport', u'-C', u'::test', u'/v/ad/bj8.ntds.dit']
    - 2014-04-13 11:43:43.261000: [u'/v/bta/bin/btaimport', u'-C', u'::test', u'/v/ad/bj8.ntds.dit']
    - 2014-04-13 11:44:36.392000: [u'/v/bta/bin/btaimport', u'-C', u'::test', u'/v/ad/bj8.ntds.dit']
    - actions:
      * 2014-04-13 11:43:07.757000: ERROR: libesedb.so: cannot open shared object file
    - actions:
      * 2014-04-13 11:43:43.261000: ERROR: libesedb.so: cannot open shared object file
    - actions:
      * 2014-04-13 11:44:36.436000: Opened ESEDB file [/v/ad/bj8.ntds.dit]
      * 2014-04-13 11:44:36.436000: Start of importation of [sd_table]
      * 2014-04-13 11:44:36.590000: End of importation of [sd_table]. 94 records.
      * 2014-04-13 11:44:36.590000: Start of creation of metatable for [sd_table]
      * 2014-04-13 11:44:36.593000: End of creation of metatable for [sd_table]
      * 2014-04-13 11:44:36.593000: Start of importation of [link_table]
      * 2014-04-13 11:44:36.638000: End of importation of [link_table]. 54 records.
      * 2014-04-13 11:44:36.638000: Start of creation of metatable for [link_table]
      * 2014-04-13 11:44:36.641000: End of creation of metatable for [link_table]
      * 2014-04-13 11:44:36.641000: Start of importation of [datatable]
      * 2014-04-13 11:44:59.245000: End of importation of [datatable]. 3516 records.
      * 2014-04-13 11:44:59.245000: Start of creation of metatable for [datatable]
      * 2014-04-13 11:44:59.308000: End of creation of metatable for [datatable]
      * 2014-04-13 11:44:59.309000: Starting post-processing
      * 2014-04-13 11:44:59.309000: Start Post-processor: category
      * 2014-04-13 11:44:59.376000: End Post-processor: category
      * 2014-04-13 11:44:59.376000: Start Post-processor: domains
      * 2014-04-13 11:44:59.384000: End Post-processor: domains
      * 2014-04-13 11:44:59.384000: Start Post-processor: dnames
      * 2014-04-13 11:45:09.551000: End Post-processor: dnames
      * 2014-04-13 11:45:09.552000: Start Post-processor: usersid
      * 2014-04-13 11:45:09.565000: End Post-processor: usersid
      * 2014-04-13 11:45:09.565000: Start Post-processor: rightsGuids
      * 2014-04-13 11:45:10.219000: End Post-processor: rightsGuids
      * 2014-04-13 11:45:10.219000: Graceful exit
```

Cohérence Avant que la méthode `run()` d'un *miner* soit appelée, `btaminer` va appeler la méthode `assert_consistency()` afin d'effectuer des test de cohérence. Ceux-ci vont par exemple vérifier que les valeurs des colonnes utilisées par ledit *miner* ont le bon type ou bien vérifier que la base n'est pas vide.

Il faut en effet éviter le faux négatif embarrassant où un *miner* ne retourne aucun résultat non pas parce que le point de contrôle est correct mais parce que la base est tout simplement vide.

Tests unitaires et de non-regression Les tests unitaires et de non-regression sont une grosse lacune de BTA pour l'instant. Nous espérons en ajouter rapidement.

Exhaustivité Les relations entre les champs et leur sémantique sont implicites à l'AD. Il suffit d'oublier une relation pour ne pas voir tout un pan de droits attribués à un objet. Un gros travail dans la compréhension des bases de l'AD est nécessaire.

Il est impossible de garantir l'absence de modifications malveillantes. Cependant, nous pensons qu'une modification, pour être malveillante, devra toucher probablement au moins un point de contrôle que nous vérifions.

3 Méthodologie d'audit

3.1 Extraction du fichier `ntds.dit` de l'AD

La procédure de génération du fichier `NTDS.dit` doit être rigoureusement suivie. L'expérience a montré que lorsqu'elle ne l'était pas ou lorsque des raccourcis étaient pris, le fichier `NTDS.dit` résultant était souvent corrompu.

Pour cela, dans un environnement Windows 2008 ou supérieur, nous recommandons l'utilisation de l'outil `ntdsutil` avec la méthode IFM [6]. Cette méthode permet d'automatiser la duplication du fichier `NTDS.dit` sans se préoccuper du VSS [7] ainsi que de la synchronisation de la base ESE avec `esentutl.exe`.

Voila les actions à mener :

```
C:\>ntdsutil
activate instance ntds
ifm
create full c:\NTDS_saved
quit
quit
```

Voici un exemple concret sur un AD 2008 de test en français. Les commandes à taper sont en gras.

```
C:\> ntdsutil
ntdsutil: activate instance ntds
```

6. Install From Media.
7. Volume Shadow Copy Service.

```
Instance active définie à « NTDS ».
ntdsutil: ifm
ifm : create full c:\NTDS_saved
Création d'une capture instantanée...
Le jeu de captures instantanées {ed7f5b24-82d9-4d30-9ebf-0cf86045d7e9}
a été généré.
Capture instantanée {ca9c5a4c-3450-4bdf-be10-62667f456f38} montée en
tant que C:\$SNAP_201404171405_VOLUMEC$\
La capture instantanée {ca9c5a4c-3450-4bdf-be10-62667f456f38} est déjà
montée.
Initialisation du mode DEFRAGMENTATION...
Base de données source :
C:\$SNAP_201404171405_VOLUMEC$\Windows\NTDS\ntds.dit
Base de données cible  :
c:\NTDS_saved\Active Directory\ntds.dit

Defragmentation  Status (% complete)

0    10   20   30   40   50   60   70   80   90  100
|----|----|----|----|----|----|----|----|----|----|
..................................................

Copie de fichiers de Registre...
Copie : c:\NTDS_saved\registry\SYSTEM
Copie : c:\NTDS_saved\registry\SECURITY
Capture instantanée ca9c5a4c-3450-4bdf-be10-62667f456f38 démontée.
Support IFM créé dans c:\NTDS_saved
ifm : quit
ntdsutil: quit
C:\>
```

Dans un environnement Windows 2003, VSS sera utilisé via l'utilitaire `vssadmin.exe`. Cependant une fois la copie de l'ensemble du répertoire NTDS effectuée, il est important de fermer et de mettre à jour la base ESE en utilisant la commande `esentutl.exe` dans le répertoire où se trouve la copie du fichier `NTDS.dit` (dans notre exemple `c:\NTDS_saved`).

Voici les commandes à taper sur un serveur Windows 2003 :

```
C:\NTDS_saved> vssadmin create shadow /for=c:
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001 Microsoft Corp.

Successfully created shadow copy for 'c:\'
    Shadow Copy ID: {43a58150-a471-49b4-8bc1-1a7d967d32d0}
    Shadow Copy Volume Name: \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1

C:\NTDS_saved> copy
 \\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS .
\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\edb.chk
```

```
\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\edb.log
\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\edbres00001.jrs
\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\edbres00002.jrs
\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\ntds.dit
\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Windows\NTDS\temp.edb
        6 fichier(s) copié(s).

C:\NTDS_saved> esentutl.exe /r edb /d /i /8

Microsoft(R) Windows(R) Database Utilities
Version 5.2
Copyright (C) Microsoft Corporation. All Rights Reserved.

Initiating RECOVERY mode...
   Logfile base name: edb
            Log files: <current directory>
         System files: <current directory>
   Database Directory: <current directory>

Performing soft recovery...

Operation completed successfully in 0.468 seconds.

C:\NTDS_saved> vssadmin delete shadows /for=c:
vssadmin 1.1 - Volume Shadow Copy Service administrative command-line tool
(C) Copyright 2001 Microsoft Corp.

Do you really want to delete 1 shadow copies (Y/N): [N]? Y

Successfully deleted 1 shadow copies.
```

Le fichier `C:\NTDS_saved\NTDS.dit` peut maintenant être récupéré et plus aucune action n'est nécessaire sur l'AD.

3.2 Import du fichier `NTDS.dit` dans la base (`btaimport`)

Une fois le fichier `NTDS.dit` récupéré, la commande `btaimport` va se charger d'importer les données dans une base MongoDB.

Suite à l'import des données de nouvelles collections sont créées dans la base MongoDB afin de précalculer certaines valeurs telles que la liste des catégories de classe de schémas (*schemaIDGUID*), d'objets (*objectGUID*), de droits (*rightsGuid*), qui pourront être utilisées par les *miners* afin de simplifier et d'accélérer certaines requêtes.

```
$ btaimport -C ::test2 mon.ntds.dit
INFO : Going to import ::test2         <- mon.ntds.dit
INFO : Opening [mon.ntds.dit]
INFO : Opening done.
INFO : ### Starting importation of sd_table ###
Importing [test2.sd_table]: 156 / 156
```

```
    --  avg=277.75 rec/s inst=217.11 rec/s
    --  ETA=0:00:00 elapsed=0:00:00
INFO : done.
INFO : Creating metatable
INFO : Importation of sd_table is done.
INFO : ### Starting importation of link_table ###
Importing [test2.link_table]: 1212 / 1212
    --  avg=6327.75 rec/s inst=4767.61 rec/s
    --  ETA=0:00:00 elapsed=0:00:00
INFO : done.
INFO : Creating metatable
INFO : Importation of link_table is done.
INFO : ### Starting importation of datatable ###
INFO : Resolving column names
Scanning for column names: 7500 / 7631
    --  avg=3500.28 recs/s inst=3343.46 recs/s
    --  ETA=0:00:00 elapsed=0:00:02
INFO : All columns found! Ending scan early!
INFO : Resolved 3200 / 3200 columns.
Importing [test2.datatable]: 7631 / 7631
    --  avg=59.37 rec/s inst=59.90 rec/s
    --  ETA=0:00:00 elapsed=0:02:08
INFO : done.
INFO : Creating metatable
INFO : Importation of datatable is done.
INFO : Post-processing: category
INFO : Post-processing: domains
INFO : Post-processing: dnames
INFO : Post-processing: usersid
INFO : Post-processing: rightsGuids
```

3.3 Exécution de *miners* permettant de requêter et corréler les informations en base (`btaminer`)

Lors d'une analyse de sécurité de l'Active Directory, il est intéressant d'auditer les permissions positionnées sur des objets spécifiques, les permissions accordées à des utilisateurs ou bien les attribut d'un objet.

Voici quelques points de contrôle dont la maîtrise a un grand intérêt en matière de sécurité :

- les groupes sensibles ;
- *adminCount* et *AdminSDHolder* ;
- le schéma Active Directory ;
- les droits étendus ;
- les tentatives de connexion infructueuses ;
- les comptes ne s'étant jamais connectés ou ne s'étant pas connectés depuis n mois ;

– l'attribut *userAccountControl*.

Nous pouvons commencer par l'analyse des groupes sensibles : *Administrateurs de l'entreprise*, *Administrateurs du domaine*... Il ne faut cependant pas limiter l'analyse aux les groupes *BUILTIN* de l'Active Directory mais échanger avec les administrateurs de l'Active Directory audité afin d'obtenir le nom des groupes sensibles d'un point de vue métier. Il est intéressant de valider que l'ensemble des membres de ces groupes sensibles sont bien connus par les administrateurs de l'Active Directory. Il est également important de valider les ACE positionnées sur chacun de ces membres afin de savoir si l'un des comptes ne peut pas être compromis par une lacune ou un droit trop permissif.

Pour cela nous utiliserons le *miner ListGroup*, qui effectue deux taches :

– lister l'ensemble des membres ;

– lister les ACE sur chacun des membres.

```
$ btaminer -C::snktest -t ReST ListGroup --match "Admins du domaine"

Analysis by miner [ListGroup]
=============================

List of groups matching [Admins du domaine]

Group Admins du domaine
-----------------------

sid = S-1-5-21-1154669122-758131934-2550385761-512
guid = 8bff35e5-87ff-4d9f-b979-122adf32cdd9
cn = .intra.secu.labz.Users.Admins du domaine
+--------------------+----------+---------------+-----------+
| User               | Deletion | Flags         | Recursive |
+====================+==========+===============+===========+
| snorky             |          | normalAccount |           |
| Administrateur     |          | normalAccount |           |
+--------------------+----------+---------------+-----------+
User snorky (S-1-5-21-1154669122-758131934-2550385761-1154)
+-------------------+---------+--------------------+----------------------+
| Trustee           | Member  | ACE Type           | Object type          |
+===================+=========+====================+======================+
| Admins du domaine | snorky  | AccessAllowedObject | (none)              |
[...]
| Everyone          | snorky  | AccessAllowedObject | User-Change-Password |
| Jean Dupond       | snorky  | AccessAllowedObject | (none)               | <==
| Self              | snorky  | AccessAllowedObject | User-Change-Password |
| Self              | snorky  | AccessAllowedObject | Private-Information   |
| Admins du domaine | snorky  | AccessAllowed       | (none)               |
| Administrateurs   | snorky  | AccessAllowed       | (none)               |
| System            | snorky  | AccessAllowed       | (none)               |
| Everyone          | snorky  | SystemAudit         | (none)               |
| Everyone          | snorky  | SystemAuditObject   | GP-Link              |
| Everyone          | snorky  | SystemAuditObject   | GP-Options           |
+-------------------+---------+--------------------+----------------------+
```

Ainsi il est possible d'identifier que le compte *Jean Dupond* dispose de l'ensemble des privilèges sur le compte *snorky*. L'objectType sert à spécifier selon les cas un droit étendu, une écriture validée, une propriété, une classe d'objet, etc. Quand il est vide comme ici, cela signifie que tous les éléments sont affectés. Mais ces éléments dont il est question dépendent des droits spécifiés dans un champ de l'ACE appelé *AccessMask*. Ici, pour des raisons de visibilité le contenu de l'*AccessMask* n'a pas été affiché, il est cependant impératif de contrôler cette valeur pour connaître les droits spécifiés.

Le contrôle des permissions relatives à *AdminSDHolder*[1, §3.2], permet d'identifier une éventuelle porte dérobée liée à une permission incorrecte ou volontairement positionnée.

Pour analyser la sanité du schéma, plusieurs contrôles sont à effectuer. Premièrement, il faut vérifier le propriétaire du Schéma *Schema Admins* via les valeurs de *nTSecurityDescriptor* (champ *Owner* et droit *WRITE_ OWNER*). Ensuite, pour les objets du schéma, les deux descripteurs de sécurité *nTSecurityDescriptor* et *defaultSecurityDescriptor* doivent être contrôlés. *nTSecurityDescriptor* est le descripteur de sécurité qui s'applique sur l'objet du schéma inspecté, alors que *defaultSecurityDescriptor* n'est présent que pour les objets du schéma décrivant une classe d'objets. Ce dernier contient donc le descripteur de sécurité par défaut affecté aux nouvelles instances de cette classe. Il faut donc contrôler les descripteurs de sécurité affectant les objects du schéma mais également les dates de création et de modification des objets *ClassSchema* et *AttributeSchema*. Les dates doivent être dans les périodes correspondantes à l'extension du schéma (installation Exchange, SCOM, etc.).

Certains droits étendus doivent particulièrement être contrôlés, tel que *User-Force-Change-Password* (type `00299570-246d-11d0-a768-00aa006e0529`), qui permet de réinitialiser le mot de passe d'un compte sans connaître le mot de passe d'origine. Il faut également être attentif au droit *Self-Membership* (type `bf9679c0-0de6-11d0-a285-00aa003049e2`) qui permet de modifier l'appartenance d'un compte à un groupe pour le bénéficiaire de l'ACE. Ci-dessous, un exemple du résultat du *miner ListACE*, permettant d'identifier que le compte *Jean Dupond* a le droit de réinitialiser le mot de passe du compte *Administrateur*.

```
$ btaminer -C::snktest -t ReST ListACE \
           --type 00299570-246d-11d0-a768-00aa006e0529
```

```
Analysis by miner [ListACE]
===========================

+-------------+----------------+---------------------------+
| Trustee     | Subjects       | Object type               |
+=============+================+===========================+
| jean dupond | Administrateur | User-Force-Change-Password |
+-------------+----------------+---------------------------+
```

De plus, l'extraction d'éléments factuels tels que le nombre de tentatives de connexion infructueuses permet par exemple aux administrateurs d'identifier des attaques en cours ou des machines ayant été retirées de l'AD mais tentant toujours de s'authentifier. Connaître les comptes ne s'étant jamais connectés ou ne s'étant pas connectés depuis n mois à l'Active Directory permet aux administrateurs de nettoyer leurs AD en désactivant les comptes inutilisés et donc de réduire la surface d'attaque.

```
$ btaminer -C::snktest -t ReST passwords --last-logon 120

Analysis by miner [passwords]
=============================

+----------------+---------------------------+----------------------+
| name           | lastLogonTimestamp        | userAccountControl   |
+================+===========================+======================+
| Administrateur | 2013-02-04 00:00:28.007000 | accountDisable:False |
| jean dupond    | 2013-02-13 19:19:00.847000 | accountDisable:False |
+----------------+---------------------------+----------------------+
```

L'attribut *UserAccountControl* est particulièrement riche en information car il contient de nombreux indicateurs sur la configuration de chaque utilisateur. Typiquement les comptes de service ont souvent le drapeau *dontExpiredPassword* mais d'autres comptes peuvent également l'avoir. Il est donc intéressant de pouvoir tous les recenser.

```
$ btaminer -C::snktest -t ReST CheckUAC --check dontExpirePassword

Analysis by miner [CheckUAC]
============================

+-------------+------------------------/ /--------+-----------------------------------------------------+
| cn          | SID                     \ \        | Flags                                               |
| Invité      | S-1-5-21-1154669122-7589/ /61-501  | normalAccount, dontExpirePassword, accountDisable   |
| Jean Dupond | S-1-5-21-1154669122-7589\ \61-1178 | normalAccount, dontExpirePassword, passwdNotrequired |
+-------------+------------------------/ /--------+-----------------------------------------------------+
```

Ci-dessous une liste de requêtes pouvant être effectuées au travers de BTA via des *miners* déjà existants et disponibles :

- Vérification sur les membres des Groupes *Domain Admins*, *Enterprise Admins*...
  Pour chaque membre :
  - Vérification du propriétaire de l'objet
  - Vérification des ACEs positionnés sur l'objet *Person/Group*

  `btaminer -C ::$1 ListGroup --match "Domain Admins"`
- Vérification sur *AdminSDHolder*
  - Liste des objets protégés par *AdminSDHolder*
    `btaminer -C::$1 SDProp --list`
  - Vérification des ACE liés à *AdminSDHolder*
    `btaminer -C ::$1 SDProp --checkACE`
- Vérification de la sanité du schéma
  - Listes par date des objets Schema modifiés ou créés
    ```
    btaminer -C ::$1 Schema --timelineAS created
    btaminer -C ::$1 Schema --timelineAS changed
    btaminer -C ::$1 Schema --timelineCS created
    btaminer -C ::$1 Schema --timelineCS changed
    ```
  - Vérification du propriétaire du Schéma
    `btaminer -C ::$1 Schema --owner`
- Vérification sur les droits étendus
  - Vérification sur le droit *User-Force-Change-Password*
    ```
    btaminer -C ::$1 ListACE \
             --type 00299570-246d-11d0-a768-00aa006e0529
    ```
  - Vérification sur le droit *Self-Membership*
    ```
    btaminer -C ::$1 ListACE \
             --type bf9679c0-0de6-11d0-a285-00aa003049e2
    ```
  - Vérification sur le droit *Send-As*
    ```
    btaminer -C ::$1 ListACE \
             --type ab721a54-1e2f-11d0-9819-00aa0040529b
    ```
- Vérification des ACE dont bénéficie le SID
  `btaminer -C ::$1 ListACE --trustee SID`
- Vérification des ACE s'appliquant sur le SID
  `btaminer -C ::$1 ListACE --subject SID`
- Vérification des boites Mails
  `btaminer -C ::$1  MailBoxRights --user SID`
- Liste des comptes qui ne se sont jamais connecté à l'AD
  `btaminer -C ::$1 passwords --never-logged`
- Listes des comptes qui ne se sont pas authentifié sur l'AD depuis 6 mois
  `btaminer -C ::$1 passwords --last-logon 182`

- Nombre de tentatives infructueuses de connexion par compte
  `btaminer -C ::$1 passwords --bad-password-count`
- Liste des comptes disposant d'un flag *UserAccountControl* particulier
  `btaminer -C ::$1 CheckUAC --check accountDisable`
  `btaminer -C ::$1 CheckUAC --check passwdNotrequired`
  `btaminer -C ::$1 CheckUAC --check passwdCantChange`

3.4 Corrélation des informations avec les administrateurs Active Directory

Les Active Directory sont des systèmes très vivants. Ils sont très spécifiques à leur environnement et de nombreuses modifications sont apportées quotidiennement. Il est donc important de corréler les informations collectées avec les administrateurs qui pourront expliquer ou confirmer les écarts avec les bonnes pratiques de sécurité.

3.5 Comparaison de deux instances d'un Active Directory (`btadiff`)

Le dernier module de BTA donne la possibilité de faire des comparaisons d'instances d'un Active Directory à deux instants. Cela permet par exemple de vérifier l'absence de modifications suspectes entre deux moments, ou de surveiller un ou plusieurs objets dans le temps et faire une analyse récurrente de l'Active Directory.

Il prend en paramètre les deux bases que l'on veut comparer. Ici nous avons une base propre (`clean`) et la même base après une compromission (`backdoor1`). Nous demandons également à l'outil de ne pas tenir compte de quelques colonnes qui polluent habituellement la sortie à l'aide de l'option `--ignore-defaults`.

```
$ btadiff --CA ::clean --CB ::backdoor1 --ignore-defaults
===============
Starting diffing sd_table
---------------
AB, 101: [] *sd_refcount['14'=>'15']
AB, 108: [] *sd_refcount['39'=>'41']
A, 229: []
A, 372: []
AB, 423: [] *sd_refcount['3'=>'2']
B, 424: []
B, 425: []
B, 428: []
---------------
Table [sd_table]: 160 records checked, 2 disappeared, 3 appeared, 3 changed
===============
===============
```

```
Starting diffing datatable
---------------
AB, 3586: [DC001] *logonCount['116'=>'117'], *lastLogon['datetime.datetime(20'=>
                                                        'datetime.datetime(20']
AB, 3639: [RID Set] *rIDNextRID['1153'=>'1154']
AB, 8784: [A:[gc]/B:[gc DEL:346bf199-8567-4375-ac15-79ec4b42b270]]
          -showInAdvancedViewOnly, -objectCategory, +lastKnownParent,
          +isRecycled, +isDeleted, *name["u'gc'"=>"u'gc\\nDEL:346bf199-8"],
          *dc["u'gc'"=>"u'gc\\nDEL:346bf199-8"]
AB, 8785: [A:[DomainDnsZones]/B:[DomainDnsZones
                                 DEL:58b2962b-708c-4c93-99ff-0b7e163131f9]]
          -showInAdvancedViewOnly, -objectCategory, +lastKnownParent,
          +isRecycled, +isDeleted,
          *name["u'DomainDnsZones'"=>"u'DomainDnsZones\\nDE"],
          *dc["u'DomainDnsZones'"=>"u'DomainDnsZones\\nDE"]
AB, 8786: [A:[ForestDnsZones]/B:[ForestDnsZones
                                 DEL:87f7d8a2-4d05-48d0-8283-9ab084584470]]
          -showInAdvancedViewOnly, -objectCategory, +lastKnownParent,
          +isRecycled, +isDeleted,
          *name["u'ForestDnsZones'"=>"u'ForestDnsZones\\nDE"],
          *dc["u'ForestDnsZones'"=>"u'ForestDnsZones\\nDE"]
B, 8789: [snorky insomnihack]
B, 8790: [gc]
B, 8791: [DomainDnsZones]
B, 8792: [ForestDnsZones]
---------------
Table [datatable]: 7636 records checked, 0 disappeared, 4 appeared, 5 changed
===============
```

Les lignes commençant par `A` sont celles où un élément est présent dans la base A et pas dans la B, et inversement pour les lignes commençant par `B`. Pour celles commençant par `AB`, nous avons une modification. Dans le cas où la base `A` est antérieure à la base `B`, nous pouvons parler respectivement d'effacement, d'ajout et de modification. La comparaison est naïve et ne permet pas encore de rendre exploitable humainement des gros différentiels.

Ici, sur des petits changements, nous pouvons tout de suite voir, dans la `sd_table`, l'effacement de deux lignes (i.e. deux *security descriptors*), l'ajout de 3, et la modification de 3 pour lesquelles le compteur de référence a changé.

Dans la `datatable` nous pouvons observer :

- l'incrémentation du *logonCount* de l'objet `DC001` ;
- le marquage des objets `gc`, `DomainDnsZones` et `ForestDnsZones` comme effacés grâce à l'ajout de `DEL:...` dans le *CN* ;
- la recréation de nouveaux objets `gc`, `DomainDnsZones` et `ForestDnsZones` ;
- la création d'un objet `snorky insomnihack`.

Nous avons donc tout de suite l'objet et les ACE à étudier de près.

4 Retour d'expérience

4.1 Exigence matérielles

Les ressources de la machine utilisées pour l'import et l'analyse ne sont pas à négliger. Les bases NTDS peuvent être très grosses et contenir des centaines de milliers d'ACE dans la tables `sd_table` et des millions d'objets dans la table `datatable`. Il est donc important de dimensionner la machine sur laquelle l'analyse sera menée. De plus le CPU doit être 64 bits pour permettre à MongoDB de créer ses bases de plusieurs giga-octets. Dans la plupart des cas, un SSD de 500Go a été utilisé avec 12Go de RAM et un Xeon X5450 (3GHz, 4 cœurs, cache L2 de 12 MB).

Ces caractéristiques ont permis d'avoir des temps de traitement satisfaisant. Ainsi une base NTDS de 8 Go contenant plus de 800.000 entrées dans la `datatable` prendra 26 Go en base et l'import durera environ 8 heures 30. Le temps d'exécution des *miners* est négligeable.

4.2 Les problèmes d'import et d'analyse rencontrés

Mauvaise extraction du fichier `NTDS.dit` Le problème le plus souvent rencontré a été l'import du fichier `NTDS.dit`. La méthode citée à la section 3.1 n'était en effet pas toujours suivie par les administrateurs. Le fichier résultant était alors incohérent ou incomplet et l'import échouait lorsqu'il atteignait les lignes corrompues.

Cohérence des objets en base Un autre problème rencontré lors des analyses est souvent la cohérence et la présence des objets définis en base. En effet il arrive que certains objets, comme un utilisateur, soient toujours référencés dans une ACE mais ne sont plus dans la `datatable`, ce qui complexifie l'analyse.

En terme de cohérence, si un Active Directory a été initialement installé en français et a été upgradé au fil de version en anglais, il en résultera des valeurs non cohérentes tels que `SID: S-1-5-21-domaine racine-519` correspondant à `Administrateurs de l'entreprise` (en français) et `SID: S-1-5-21-domaine-512` correspondant à `Domain Admins` (en anglais). Il est donc important de baser les recherches sur des SID et non directement au travers du *Common Name*. Les *miners* doivent également pour chaque requête contrôler la valeur retournée et ne pas tenter d'afficher ou de convertir le CN ou le DN.

4.3 Résultats d'audits

Les analyses ont toutes remonté des problèmes dus à des mauvaises pratiques, comme des comptes d'administration non nominatifs ou encore des comptes dont le mots de passe n'expire jamais.

L'analyse ne peut être effectuée de manière autonome sans interaction avec les équipes AD. Chaque environnement AD est unique, paramétré avec des manières de travail propres à chaque entité et avec son propre historique. De plus, il n'est pas possible de se positionner sur la justesse de l'attribution des droits au sein de l'environnement. Il n'est pas possible de savoir s'il est correct qu'un compte appartienne au groupe *Administrateurs du domaine* sans l'aide d'un administrateur de cet AD. Il est donc essentiel de travailler avec lui afin de comprendre *a minima* l'environnement et surtout de faire valider les résultats.

5 Conclusion

BTA est un outil d'audit destiné à donner en temps contraint un résultat reproductible. Il est complémentaire d'outils exploratoires.

Même hors des problématiques de compromission, ses résultats sont d'une aide précieuse pour nettoyer et durcir les AD audités.

Les développements en cours concernent l'ajout d'un mode LDAP, l'amélioration de la comparaison entre AD et la création de tests unitaires pour les *miners*.

Annexes

A Langage RPN d'édition de noms de fichiers

Comme vu à la section 2.3, lorsqu'on importe plusieurs bases simultanément, il peut être pratique de dériver le nom de la base BTA à partir du nom du fichier contenant la base AD. Un langage de manipulation de chaînes semblait simple, pratique et extensible.

A.1 Définition

Le langage d'édition de noms de fichiers de BTA est un micro-langage RPN destiné à modifier des noms de fichiers pour les transformer en noms de base. Chaque commande du langage dépile un ou plusieurs arguments sur une pile, puis empile le résultat.

En effet, lors de l'import de multiples fichiers, si leur nommage a été fait de manière cohérente, il peut être très rapide de transformer ces noms en noms de bases.

Le langage met donc à notre disposition une liste de commandes d'édition de chaîne. On numérotera les éléments de la pile en partant de 0, 0 étant l'élément du haut de la pile.

`dup :` duplique l'opérande

`drop :` dépile l'opérande

`swap :` intervertit les opérandes de niveau 0 et 1

`basename :` retire tous les caractères précédents le dernier `/` inclus, ce qui a pour effet, sur un chemin, de ne garder que le nom de fichier

`dirname :` retire tous les caractères après le dernier `/` inclus, ce qui a pour effet, sur un chemin, de retirer le nom du fichier

`pathjoin :` concatène les opérandes de niveau 0 et 1, en ajoutant un `/` si nécessaire

`rmext :` retire tous les caractères suivant le dernier `.` inclus, ce qui a pour effet de retirer une éventuelle extension

`rmallext :` retire tous les caractères suivant le premier `.`

`rmheaddir :` retire tous les caractères précédent le premier `/` inclus

`tail :` ne garde de l'opérande de niveau 1 que le nombre de caractères finaux indiqués par l'opérande de niveau 0

`cuttail :` retire de l'opérande de niveau 1 le nombre de caractères finaux indiqués par l'opérande de niveau 0

`head :` ne garde de l'opérande de niveau 1 que le nombre de caractères initiaux indiqués par l'opérande de niveau 0

`cuthead :` retire de l'opérande de niveau 1 le nombre de caractères initiaux indiqués par l'opérande de niveau 0

`extract :` retire de l'opérande de niveau 2 les caractères initiaux indiqués par l'opérande de niveau 1 et les caractères finaux indiqués par l'opérande de niveau 0

`plus :` concatène le niveau 1 et le niveau 0

`replace :` remplace dans l'opérande de niveau 2 toutes les occurrences de l'opérande de niveau 1 par l'opérande de niveau 0

`remove :` retire de l'opérande de niveau 1 toutes les occurrences de l'opérande de niveau 0

`lower :` convertit l'opérande de niveau 0 en minuscules

`upper :` convertit l'opérande de niveau 0 en majuscules

Toute autre chaîne ou expression entre quotes est empilée.

A.2 Exemples

Il est possible de tester une expression en appelant directement le module `bta/tools/RPNedit.py`. Par exemple, en partant d'une pile vide, nous allons empiler la chaîne `abc`, puis la chaîne `def` puis les concaténer :

```
$ python RPNedit.py "'abc' 'def' plus"
==> abcdef
```

Si nous avons trois fichiers `/tmp/ad/ad{1,2,3}.ntds.dit` et que nous souhaitons les importer dans des bases nommées respectivement `audit_ad1`, `audit_ad2` et `audit_ad3`, les opérations que nous avons à faire sont de récupérer uniquement le nom du fichier, de retirer toutes les extensions et de rajouter la chaîne `audit_` au début :

```
$ python RPNedit.py "basename rmallext 'audit_' swap plus" \
         --stack /tmp/ad/*.dit
audit_ad1            <== /tmp/ad/ad1.ntds.dit
audit_ad2            <== /tmp/ad/ad2.ntds.dit
audit_ad3            <== /tmp/ad/ad3.ntds.dit
```

Si maintenant les fichiers se nomment tous `ntds.dit` mais dans des répertoires différents donnant le nom de la base :

```
$ python RPNedit.py "dirname basename 'audit_' swap plus " \
         --stack /tmp/ad/*/ntds.dit
audit_ad1            <== /tmp/ad/ad1/ntds.dit
audit_ad2            <== /tmp/ad/ad2/ntds.dit
audit_ad3            <== /tmp/ad/ad3/ntds.dit
```

Si enfin, nous souhaitons un autre schéma et si seule une partie du nom du répertoire nous intéresse :

```
$ python RPNedit.py "dirname basename 3 cuthead '.' '_' replace upper" \
         --stack /tmp/ad/*/ntds.dit
CLIENT_AAA           <== /tmp/ad/ad.client.aaa/ntds.dit
CLIENT_BBB           <== /tmp/ad/ad.client.bbb/ntds.dit
CLIENT_CCC           <== /tmp/ad/ad.client.ccc/ntds.dit
```

Références

1. G. de Drouas and P. Capillon. Audit des permissions en environnement active directory. In *SSTIC*, 2012.
2. Airbus Group. BTA AD auditing tool, 2013. `https://bitbucket.org/iwseclabs/bta`.
3. Benoît Jousse. The active directory permissions analysis challenge, 2014. `http://blog.cassidiancybersecurity.com/post/2014/03/The-Active-Directory-Permissions-Analysis-Challenge`.
4. Joachim Metz. Extensible Storage Engine (ESE) database file (EDB) format specification. Technical report, Google, 2009-2012. `https://googledrive.com/host/0B3fBvzttpiiSN082cmxsbHB0anc/Extensible%20Storage%20Engine%20(ESE)%20Database%20File%20(EDB)%20format.pdf`.
5. Joachim Metz. Library and tools to access the Extensible Storage Engine (ESE) database file (edb) format., 2012. `https://code.google.com/p/libesedb/`.