ntdissector: a swiss-army knife for your NTDS files

Mehdi Elyassa and Julien Legras
mehdi.elyassa@synacktiv.com
julien.legras@synacktiv.com

Synacktiv

Abstract. NTDS files are the central databases of Windows Active Directory environments. They contain secrets and credentials that can be extracted using miscellaneous offensive public tools. However, their capabilities often stop at these secrets and credentials but not more. The *ntdissector* tool was developed to extract and format all the data of these NTDS files by using only Python to be platform-agnostic and easily improved.

1 Introduction

1.1 Context

During an Active Directory password review, we had to output various statistics across different user populations. In that case, the type of users was set in the *extensionAttribute* attribute in the NTDS database. While we had no problem to extract the LM and NT hashes from the NTDS, we were forced to use other tools such as *go-ese* to extract other fields.

The output – JSON files – of the tool was good enough for our needs, but it was not able to decode many attributes and decrypt anything.

At Synacktiv, we use *ldeep* on a regular basis to extract LDAP information and output them as JSON files. The similarity of the outputs and the data eventually led us to the following question: is it possible to create a tool able to output the same data as *ldeep* but directly from a NTDS file?

We will explain in this article our journey in developing such a tool. Some new features introduced after the publication of the two blog posts [2, 3] will also be addressed.

1.2 State of the art

NTDS parsing tools are not new as security auditors need them to perform various audits such as passwords or permissions reviews. Most

of them are dedicated to a specific type of audit such as *pwdump* or *secretsdump* to extract LM and NT hashes or *BTA* to audit the permissions.

We initially worked on *go-ese* to add the missing features but it was not as easy as we thought. Eventually, we found the *dissect* project from Fox-IT: it is developed in Python, it has a *dissect.esedb* module that can be used to convert the NTDS records to JSON. At first, it was quite slow but a good profiling and debugging session resolved the issue. We eventually made a pull request [1] to serialize ESE objects to Python *dict* more efficiently. On a sample NTDS.dit of 1.5GB, the processing time improved by 10 times (from 40 to 4 minutes).

2 Tool's capabilities

At first run, the tool builds various cache files to map out the schema, extract security descriptors, build up links and more broadly speed up any future execution. Therefore, besides converting records to JSON objects, the tool resolves ATT column names to their LDAP or CN naming equivalent. To do so, the *Attribute-Name-LDAP* and *Attribute-Name-CN* attributes of objects from the *attributeSchema* class are saved into a dictionnary, which is persisted with the cache.

The tool also parses the *link_table* to extract the links and backlinks between objects. Moreover, the hierarchy between objects is processed to build up distinguished names.

Additionaly, extracting more than just NT and LM hashes was a major objective behind this project. Our efforts were therefore focused on extracting most of the secrets stored in the NTDS database in order to produce a cross-platform tool that can meet many needs. Today, ntdissector extracts and decrypts the following secrets:

- *DPAPI backup keys*: the backup key is formatted as a PVK key and can be directly used by DPAPI tools, such as *dpapi.py*.
- *Supplemental credentials*: a structure that contains cryptographic hashes for the Digest and Kerberos authentication protocols.
- *LAPS legacy passwords*: plaintext passwords of the local administrators and their associated expiration time.
- *Windows LAPS passwords*: also known as LAPSv2.
- Authentication secrets related to incoming and outgoing domain trusts.

3 Regular NTDS files

This chapter covers the main technical challenges we faced while attempting to format the objects and to get rid of the encryption layers protecting sensitive data.

3.1 DN resolution

As stated earlier, to reproduce the LDAP output, a distinguished name (*DN*) resolution process is implemented in ntdissector. The objects hierarchy is implemented through two attributes:

— A *DNT_col* attribute stores a unique ID identifying the object itself.
— Another *PDNT_col* attribute holds the parent object's ID.

Two other attributes hold the information to format the relative distinguished name (*RDN*) of the object:

— A *RDNtyp_col* attribute contains an ID referencing an attributeSchema object that represents the *DN* type (*CN*, *OU*, *DC*...).
— Another *RDN* attribute contains the name of the object.

These attributes combined to some internal ID resolutions allow formatting the RDN with the `{RDNtyp_col}={RDN}` format.

Regarding the full DN, it is constructed recursively based on the tree structure as follows:

```
{RDNtyp_col}={RDN},{RDNtyp_col N-1}={RDN N-1},{RDNtyp_col N-2}={RDN N-2} ...
# CN=User1,OU=Users,DC=DOMAIN,DC=LOCAL
```

3.2 LAPS v2

In early 2023, Microsoft released a new version of the LAPS solution named *Windows LAPS* [12]. Among the significant changes introduced by this new version, password encryption is now supported to avoid unsecure storage as plaintext in the Active Directory. It mainly relies on the DPAPI-NG mechanism and AES-256. Consequently, the following attributes are introduced in the AD schema by Windows LAPS:

— *msLAPS-Password*
— *msLAPS-EncryptedPassword*
— *msLAPS-EncryptedPasswordHistory*
— *msLAPS-EncryptedDSRMPassword*

— *msLAPS-EncryptedDSRMPasswordHistory*

If encryption is disabled, the *msLAPS-Password* attribute of the computer object stores a JSON object such as:

```
{
  "n":"Administrator",     # Name of the managed local account
  "t":"1d91d7c83e34480",   # UTC password update timestamp
  "p":"<password>"         # Plaintext password
}
```

Otherwise, the content of the *msLAPS-Encrypted** attributes is a blob which uses the structure format defined for the *ms-LAPS-EncryptedPassword* [4] attribute. The latter contains the JSON described above encrypted with a Content Encryption Key (*CEK*) protected via the *MS-GKDI* [6] protocol.

Such protocol, relies on a *Key Distribution Services (KDS) Root Key* [5] to derive the Key Encryption Key (*KEK*) used to encrypt the *CEK*.

Since the *KDS Root Keys* are stored in the NTDS database in *msKds-ProvRootKey* objects, ntdissector implements an offline version of the *MS-GKDI* protocol in order to compute the *KEK* for any given LAPS encrypted password.

```
  // msKds-ProvRootKey.json
   {
    "cn": "0ff68468-a6bf-086c-5c23-2b42fcecb555",
  [...]
    "msKds-KDFAlgorithmID": "SP800_108_CTR_HMAC",
    "msKds-KDFParam": "<HEX>",
    "msKds-PrivateKeyLength": 512,
    "msKds-PublicKeyLength": 2048,
    "msKds-RootKeyData": "<HEX>",
    "msKds-SecretAgreementAlgorithmID": "DH",
    "msKds-SecretAgreementParam": "<HEX>",
  [...]
    "objectCategory":
↪   "CN=ms-Kds-Prov-RootKey,CN=Schema,CN=Configuration,DC=..",
    "objectClass": [
      "msKds-ProvRootKey",
      "top"
    ],
  }
```

Figure 1 represents the decryption process, which is broadly detailed in our blog post [2].

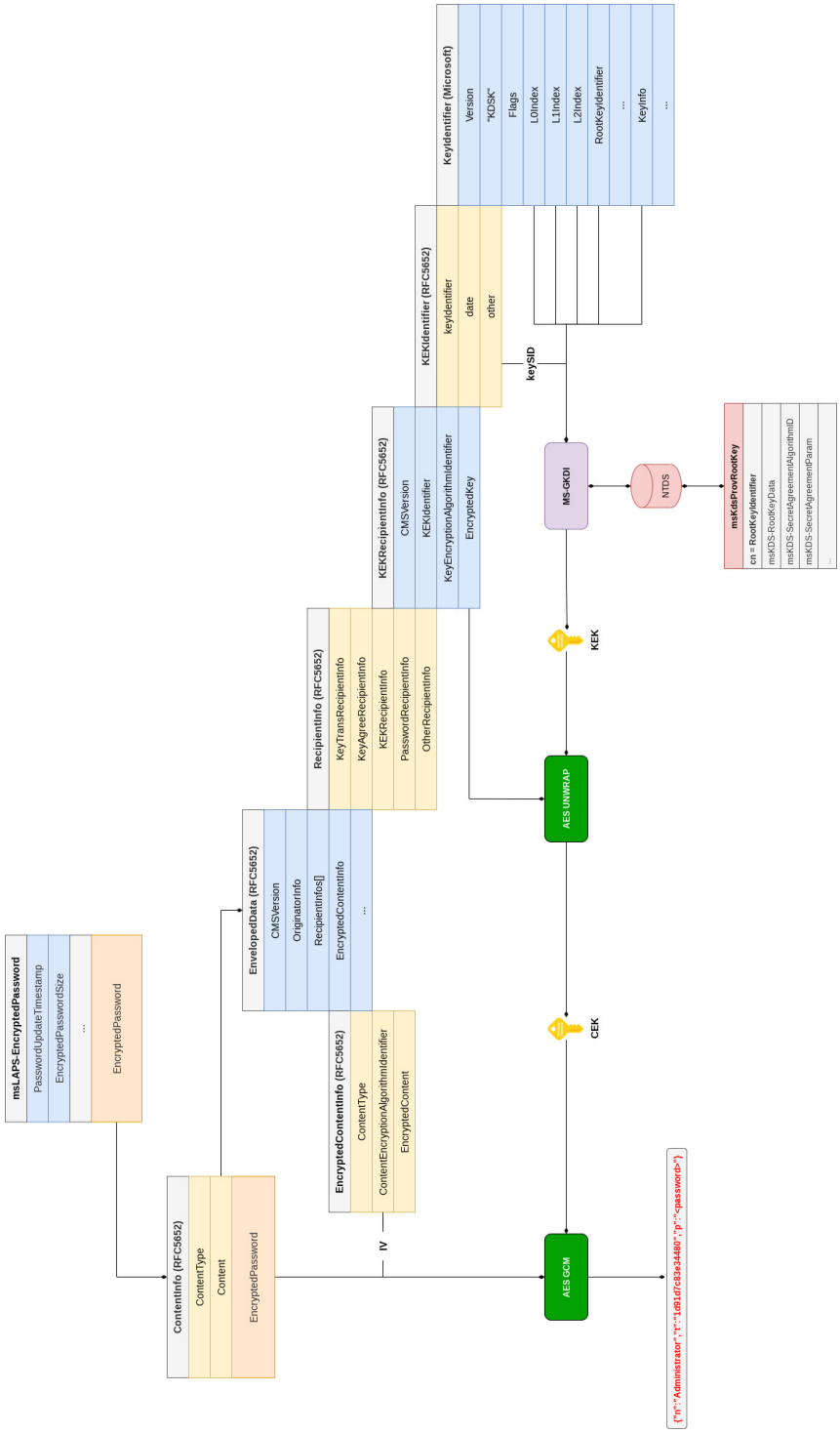

Fig. 1. LAPSv2 password decryption process.

3.3 Trusts

Secrets related to domains trusted by or trusting the local domain are also extracted and formatted by the tool. In particular, the *trustAuthIncoming* and *trustAuthOutgoing* secret attributes defined in objects of the *trustedDomain* class are processed. They contain *LSAPR_AUTH_INFORMATION* [10] structures that hold authentication information either as an RC4 key or a cleartext password, which can be used to compute RC4 or legacy DES keys.

4 ADAM NTDS files

During the development of ntdissector, we took into consideration a particular variant of NTDS databases dubbed ADAM. AD Lightweight Directory Services rely on this format which presents some particularities that caused known tools to fail parsing it.

The first difference resides in the data encryption mechanism. Both formats protect the password hashes and various sensitive information with a Password Encryption Key (PEK). The latter is encrypted with a *SysKey* before being stored in the database.

While a classic NTDS database derives the *SysKey* from four separate keys (*JD*, *Skew1*, *GBG* and *Data*) stored in the SYSTEM hive, the ADAM database, being a standalone instance, relies on a *BootKey* instead, which is assembled from 2 values stored directly in two distinct records: the *rootPekList* as an attribute of the *top* object class and the schemaPekList in the *dMD* object.

```
$ jq '{name, pekList}' ntdissector/out/{top,dMD}.json
{ "name": "$ROOT\_OBJECT$", "pekList": "<HEX>" }
{ "name": "Schema", "pekList": "<HEX>" }
```

The permutations required to compute the *BootKey* are as follows:

```
root_permutation = [2, 4, 25, 9, 7, 27, 5, 11]
schema_permutation = [37, 2, 17, 36, 20, 11, 22, 7]
bootKey = b"".join([rootPekList[i] for i in root_permutation]+
↪  [schemaPekList[i] for i in schema_permutation])
```

Once the key is computed, the credentials can be decrypted by reusing the original decryption routine without the 3DES decryption step. Indeed, the ADAM format relies on a single RC4 encryption layer.

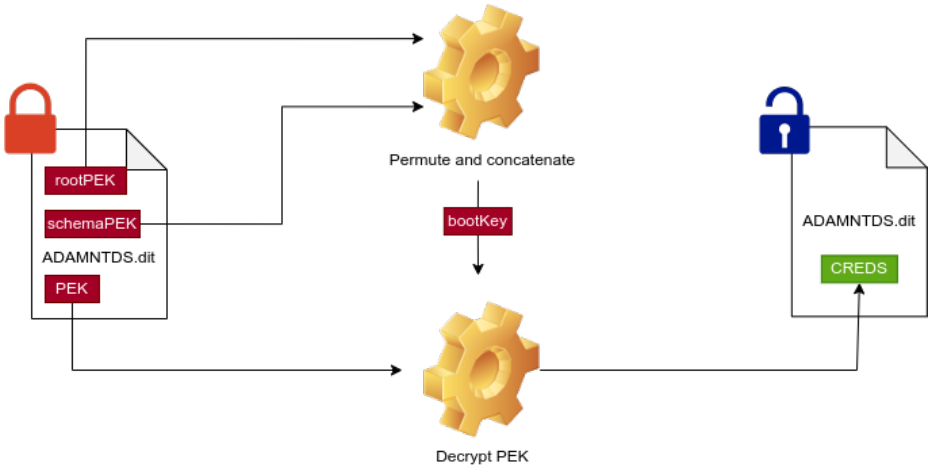

Fig. 2. ADAM NTDS secrets decryption process.

One last difference resides in the kind of structure stored in the *supplementalCredentials* attribute. In a standard NTDS.dit database, this attribute holds the *USER_PROPERTIES* [8] structure. Among other secrets, this structure stores the Kerberos long term keys.

However, in ADAM files, a *WDIGEST_CREDENTIALS* [9] structure is stored in that attribute. In this case, it has little interest since it only seems to store *WDigest* credentials.

5 ntdissector toolset

One of the main goals of *ntdissector* was to provide readable input for other tools. This section describes what can be done with the output of *ntdissector* and the tools it offers.

5.1 secretsdump

The script *user_to_secretsdump.py* provides a simple conversion of JSON files to the usual *secretsdump* output:

```
<username>:<rid>:<LM>:<NT>::: (pwdLastSet=<date>) (status=<status>)
<username>_historyX:<rid>:<LM>:<NT>:::
```

5.2 Bloodhound

Of course, ingesting the data into *Bloodhound* was on the to-do list. A simple converter would be useful for both pentesters who find old NTDS.dit files and for forensic analysis.

The script *convert_to_bloodhound.py* implements such transformation. The documentation of *Bloodhound* is not up-to-date, so most of the work has been inspired by *BloodHound.py* [11] and *SharpHound* [13].

6 Remaining technical challenges

6.1 Compression

Though *dissect.esedb* implements the core of the ESE parsing features, it is not possible at the time of writing to easily decompress some ESE fields that are compressed using the XPRESS10 algorithm. Microsoft implemented this algorithm for their cloud services and they developed their own hardware to improve the decompression in the infrastructure [7].

Unfortunately, the available code cannot be used as-is with Python. *ntdissector* ignores these fields for the moment, contribution to *dissect.esedb* are welcome to add the support of this compression algorithm.

7 Conclusion

The open source tool **ntdissector** was published at `https://github.com/synacktiv/ntdissector`.

References

1. Julien Legras Mehdi Elyassa. Add functions to efficiently serialize records, 2023. `https://github.com/fox-it/dissect.esedb/pull/24`
2. Julien Legras Mehdi Elyassa. Introducing ntdissector, a swiss army knife for your ntds.dit files, 2023. `https://www.synacktiv.com/publications/introducing-ntdissector-a-swiss-army-knife-for-your-ntdsdit-files`
3. Julien Legras Mehdi Elyassa. Using ntdissector to extract secrets from adam ntds files, 2023. `https://www.synacktiv.com/en/publications/using-ntdissector-to-extract-secrets-from-adam-ntds-files`
4. Microsoft. [MS-ADA2]: Attribute ms-LAPS-EncryptedPassword, 2021. `https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-ada2/b6ea7b78-64da-48d3-87cb-2cff378e4597`

5. Microsoft. [MS-GKDI]: Creating a New Root Key, 2021.
https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-gkdi/017840c0-2aca-4abe-9ef5-979046e8a198
6. Microsoft. [MS-GKDI]: Group Key Distribution Protocol, 2021.
https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-gkdi/943dd4f6-6b80-4a66-8594-80df6d2aad0a
7. Microsoft. Project zipline, 2021.
https://github.com/opencomputeproject/Project-Zipline
8. Microsoft. [MS-SAMR]: supplementalCredentials, 2022.
https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-samr/0705f888-62e1-4a4c-bac0-b4d427f396f8
9. Microsoft. [MS-SAMR]: WDIGEST_CREDENTIALS Construction, 2022.
https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-samr/511f65e8-3dbe-4377-b657-30b47dc4f26c
10. Microsoft. [MS-ADTS]: LSAPR_AUTH_INFORMATION, 2024.
https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/dfe16abb-4dfb-402d-bc54-84fcc9932fad
11. Dirk-Jan Mollema. Bloodhound.py.
https://github.com/dirkjanm/BloodHound.py
12. Jay Simmons. By popular demand: Windows LAPS available now!, 2023.
https://techcommunity.microsoft.com/t5/windows-it-pro-blog/by-popular-demand-windows-laps-available-now/bc-p/3805787
13. BloodHound team. Sharphound.
https://github.com/BloodHoundAD/SharpHound