

skelsec / jackdaw

[Watch](#) 13 [Star](#) 185 [Fork](#) 25

[Code](#) [Issues 1](#) [Pull requests 3](#) [Projects 0](#) [Security](#) [Insights](#)

Dismiss

### Join GitHub today









GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

gather gather gather

[125 commits](#) [5 branches](#) [0 packages](#) [0 releases](#) [2 contributors](#)


[Branch: master ▾](#) [New pull request](#) [Find file](#) [Clone or download ▾](#)

 skelsec	making file enum optional	Latest commit c40e3ac 12 days ago
 jackdaw	making file enum optional	12 days ago
 .gitignore	adding GPO and OU funct	14 days ago
 MANIFEST.in	started UI, lot of improvements	26 days ago
 Makefile	adding GPO and OU funct	14 days ago
 README.md	Update README.md	14 days ago
 jackdaw.py	yaaay new version	3 months ago
 setup.py	making file enum optional	12 days ago

README.md

# jackdaw

gather gather gather



# What is this?

---

Jackdaw is here to collect all information in your domain, store it in a SQL database and show you nice graphs on how your domain objects interact with each-other and how a potential attacker may exploit these interactions. It also comes with a handy feature to help you in a password-cracking project by storing/looking up/reporting hashes/passwords/users.

## Example commands

---

Most of these commands are available already from the webapi, except for the database init.

### DB init

```
jackdaw --sql sqlite:///<full path here>/test.db dbinit
```

### Enumeration

#### Full enumeration with integrated sspi - windows only

```
jackdaw --sql sqlite:///test.db enum 'ldap+sspi://10.10.10.2' 'smb+sspi-ntlm://10.10.10.2'
```

#### Full enumeration with username and password - platform independent

The password is `Passw0rd!`

```
jackdaw --sql sqlite:///test.db ldap 'ldap://TEST\victim:Passw0rd!@10.10.10.2' 'smb+ntlm-  
password://TEST\victim:Passw0rd!@10.10.10.2'
```

#### LDAP-only enumeration with username and password - platform independent

The password is `Passw0rd!`

```
jackdaw --sql sqlite:///test.db ldap 'ldap://TEST\victim:Passw0rd!@10.10.10.2'
```

### Start interactive web interface to plot graph and access additional features

```
jackdaw --sql sqlite:///<FULL PATH TO DB> nest
```

Open `http://127.0.0.1:5000/ui` for the API

Please see the `Building the UI` section further down to learn how to build the UI. Once built:

Open `http://127.0.0.1:5000/nest` for the graph interface (shows the graph, but far from working)

## Features

---

### Data acquisition

---

#### via LDAP

LDAP enumeration phase acquires data on AD info, User, Machine, OU, Group objects which will be represented as a node in the graph, and as a separate table in the DB. Additionally all aforementioned objects' Security Descriptor will be parsed and the ACLs for the DACL added to the DB. This, together with the membership information will be represented as edges in the graph. Additionally custom SQL queries can be performed on any of the aforementioned data types when needed.

#### via SMB

SMB enumeration phase acquires data on shares, localgroups, sessions, NTLM data via connecting to each machine in the domain (which is acquired via LDAP)

#### via LSASS dumps (optional)

The framework allows users to upload LSASS memory dumps to store credentials and extend the session information table. Both will be used as additional edges in the graph (shared password and session respectively). The framework also uses this information to create a password report on weak/shared/cracked credentials.

### via DCSYNC results (optional)

The framework allows users to upload impacket's DCSYNC files to store credentials. This be used as additional edges in the graph (shared password). The framework also uses this information to create a password report on weak/shared/cracked credentials.

### via manual upload (optional)

The framework allows manually extending the available DB in every aspect. Example: when user session information on a given computer is discovered (outside of the automatic enumeration) there is a possibility to manually upload these sessions, which will populate the DB and also the result graph

## Graph

---

The framework can generate a graph using the available information in the database and plot it via the web UI (nest). Furthermore the graph generation and path calculations can be invoked programmatically, either by using the web API (/ui endpoint) or the grph object's functions.

## Anomlaies detection

---

The framework can identify common AD misconfigurations without graph generation. Currently only via the web API.

### User

User anomalies detection involve detection of insecure UAC permissions and extensive user description values. This feature set is expected to grow in the future as new features will be implemented.

### Machine

Machine anomalies detection involve detection of insecure UAC permissions, non-mandatory SMB signing, outdated OS version, out-of-domain machines. This feature set is expected to grow in the future as new features will be implemented.

## Password cracking

---

**The framework is not performing any cracking, only organizing the hashes and the cracking results currently main focus is on impacket and aiosmb's dcsync results INT and LM hashes only!**

Sample porcess is the following:

1. Harvesting credentials as text file via impacket/aiosmb or as memory dumps of the LSASS process via whatever tool you see fit.
2. Upload the harvested credentials via the API
3. Poll uncracked hases via the API
4. Crack them (hashcat?)
5. Upload the results to the framework via the API
6. Generate a report on the cracked/uncracked users and password strength and password sharing

*note form author: This feature was implemented for both attackers and defenders. Personally I don't see much added value on either side, since at the point one obtained the NT hash of a user it's just as good as the password... Nonetheless, more and more companies are performing password strength excercises, and this feature would help them. As for attackers: it is just showing off at this point, but be my guest. Maybe scare management for extra points.*

## Important

---

This project is in experimental phase! This means multiple things:

1. it may crash
2. the controls you are using might change in the future (most likely)
3. (worst part) The database design is not necessary suitable for future requests so it may change. There will be no effort to maintain backwards compatibility with experimental-phase DB structure!

# Technical part

---

## Database backend

---

Jackdaw uses SQLAlchemy ORM module, which gives you the option to use any SQL DB backend you like. The tests are mainly done on SQLite for obvious reasons. There will be no backend-specific commands used in this project that would limit you.

## Building the UI

---

### THIS IS ONLY NEEDED IF YOU INSTALL VIA GIT AND/OR CHANGE SOMETHING IN THE UI CODE

The UI was written in React. Before first use/installation you have to build it. For this, you will need `nodejs` and `npm` installed. Then:

1. Go to `jackdaw/nest/site/nui`
2. Run `npm install`
3. Run `npm run build`

Once done with the above, the UI is ready to play with.

## Kudos

---

"If I have seen further it is by standing on the shoulders of Giants."

### For the original idea

BloodHound team

### For the ACL edge calculation

@dirkjanm (<https://github.com/dirkjanm/>)

### For the awesome UI

Zsolt Imre (<https://github.com/keymandll>)

### For the data collection parts

please see kudos section in `aiosmb` and `msldap` modules

**In case I forgot to mention someone pls send a PR**