1. Intro

Djedefre is a tool for documenting your network.

Djedefre consists of 3 parts:

— the database

— scan scripts

— the GUI

The scan scripts fill the database. The GUI displays the network in the database and allows the user to change certain values. The database is created when the GUI is first started.

Djederfre is not a stable release. GUI parts may change, database layout can change and functionality will be added. Some of the changes will not yet have been documented.

Some time ago, there was a tool called Cheops, that automated network discovery and gave a nice drawing of the network. Cheops is now abandonware. Djedefre was pharaoh after Cheops. The main difference between Cheops and Djedefre is, that Djedefre assumes that the network is managed. This means that there is access to at least some servers, routers and switches.

2. Using Djedefre

2.1 Installing

From `"https://github.com/ljmdullaart/djedefre"` download at least

— `djedefre.pl`

— the directory `scan_scripts`

— the directory `images`

Make sure that the following Perl modules are installed:

— `Tk` - `Tk::JBrowseEntry`

— `File::Slurp`

— `File::Slurper`

— `File::HomeDir`

— `DBD/SQLite.pm` - `List/MoerUtils.pm` - `Sort/Naturally.pm`

Make sure that the following programs are installed:

— `ipcalc`

— `grepcidr`

— `fping`

2.2 First start

After installing or unpacking,

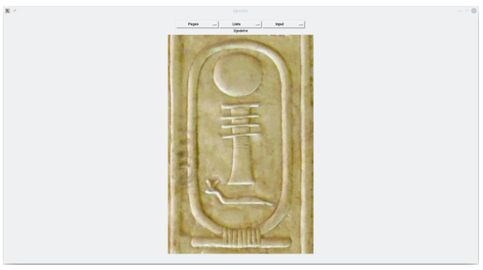
— Move to the directory where `Djedefre` is installed.

— Make sure that there is a directory `database` and create it if it is not there

— Create the database with `perl djedefre_create_db.pl`

— start the GUI with `./dedefre.pl &`

You will be greeted with a cartouche depicting the hieroglyphs for Djedefre with a number of menus above it.



The buttons give access to functionality:

— Pages contains a list of pages with network drawings

— Lists provides listings of the network

— Input provides various methods for data entry

At this point, when you select the page "top" from the Pages menu, you will get an empty drawing.

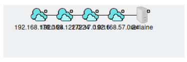
2.3 Start scanning

To get some information into the database, scan-scripts are used. These scan-scripts can be found in the directory `scan_scripts`.

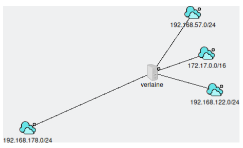
The best start would be to scan the local system. Launch the script `scan_local_system.sh` in the `scan_scripts` directory in a separate `xterm` window with:

```
bash scan-scripts/scan_local_system.sh database/djedefre.pl
```

When the scan is finished, the plot page "top" will show the local system and the subnets that it is connected to.

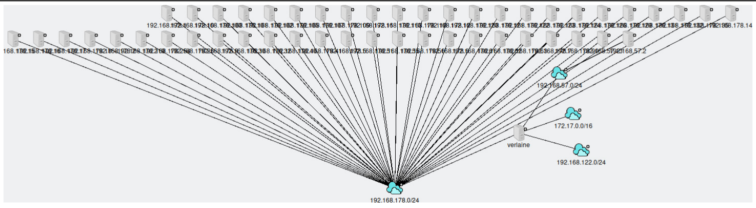


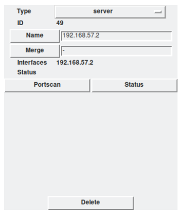
With the left mouse button, you can pick-up the server and the subnets and place them anywhere on the canvas that you like. When a server or subnet is selected, an info-window opens on the right displaying some information about the selected item.



As there are now subnets in the database, a scan of the subnets is possible. Run the script `scan_subnet.sh` with:

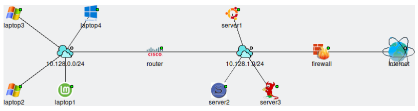
```
bash scan-scripts/scan_subnet.sh database/djedefre.pl
```





Type server
ID 49
Name 192.168.57.2
Merge -
Interfaces 192.168.57.2
Status

When you have scanned the network, placed all devices and subnets, set the correct type and name, your network may look something like this:



2.4 Scanning the network

You may have noticed that the scan of subnets did not provide a complete list of all the servers on the network. This is because the subnet scan uses `fping` to scan the subnet. Typically, for example, Windows 10 does not reply to a ping, so no Windows 10 machines will show up. An ARP scan however will detect those machines if they're on the same layer 2 segment.

There are a number of scan scripts available. These scripts try to find out what the network lay-out is, what the systems are. Although the scripts can run in any sequence, the fastest discovery seems to be:

— local_system

— subnet

— arp

— access

— type

— server

— remote_system

— cisco

— dns

— vbox

— internet

— database_integrity

— status

The scan scripts are in the directory `scan_scripts` and they are called `scan_xxxxx.sh` with `xxxxx` meaning the name of the scan.

2.4.1 local_system

Adds the local system to the network. Also adds the interfaces and the subnets that the system is connected to.

2.4.2 subnet

Scans all the known subnets for servers. Uses `sudo fping` for the scan.

2.4.3 access

Scan all the known systems for ssh access. It must be password-less login. Three users are used:

— current user: should be the default

— `root` : should be available only is a very closed environment, because it is unsafe. Some IoT things require this.

— `admin` : same remark as `root` But some devices need this, for example older Qnap NASses.

2.4.4 arp

The ARP table is read from every device that has ssh access. Interfaces that have an IP address and where the MAC-id is not 00:00:00:00:00:00 are added. For interfaces without host, a server is created.

2.4.5 type
Tries to determine the type of servers. This works reasonably well for servers that have ssh access.

2.4.6 server
For the servers that have ssh access, try to appropriate all the interfaces.

Also, if the host name is still the IP address, rename the server to its host name.

2.4.7 remote_system
For all the servers that have ssh access, add the networks that they are connected to.

2.4.8 cisco
Cisco IOS devices do not have a standard shell. The Cisco scan scans devices that have the type "cisco" and does a "server" scan and a "remote_system" scan on these devices.

2.4.9 dns
For all interfaces, set the host name to the host name that DNS has given it. Also, for all servers where the name is an IP address, set the host name to the corresponding name in DNS.

2.4.10 vbox
Determine for all VirtualBox managers which machines are running on it.

2.4.11 status
Determines the status of the servers. In its simple form, it tests whether the server responds to a ping.

If under `scan_scripts` there is a script`status_`*name*`.sh`, where *name* is the name of the server, then that script is used to determine the status of the server. If the script has an exit code 0, then the server is up.

2.4.12 internet
Uses `traceroute` to determine where the break-out to the Internet is. The last IP address that is in the database table "interfaces" is considered as belonging to the server that has the interface to the Internet.

3. Using_djedefre

3.1 Objects, pages, lists

Djedefre adiminstrates objects.

4. Details

5. Database

Most fields are only filled-in if they are known

5.1 interfaces

id	integer primary key autoincrement	ID of the interface
macid	string	MAC-ID
ip	string	IP address
hostname	string	
host	integer	
subnet	integer	
access	string	
connect_if	integer	
port	integer	

5.2 subnet

id	integer primary key autoincrement
nwaddress	string
cidr	integer
xcoord	integer
ycoord	integer
name	string
options	string
access	string

5.3 server

id	integer primary key autoincrement
name	string
xcoord	integer
ycoord	integer
type	string
status	string
last_up	integer
options	string
ostype	string
os	string
processor	string
devicetype	string
memory	string

5.4 command

id	integer primary key autoincrement
host	string
button	string
command	string

5.5 details

id	integer
type	string
os	string

5.6 pages

id	integer primary key autoincrement
page	string
tbl	string
item	integer
xcoord	integer
ycoord	integer

5.7 switch

id	integer primary key autoincrement
switch	string
server	integer
name	string
ports	integer

5.8 l2connect

id	integer primary key autoincrement
vlan	string
from_tbl	string
from_id	integer
from_port	integer
to_tbl	string
to_id	integer
to_port	integer

5.9 config

id	integer primary key autoincrement
attribute	string
item	string
value	string

5.10 cloud

id	integer primary key autoincrement
name	string
vendor	string
type	string
xcoord	integer
ycoord	integer
service	string

5.11 dashboard

id	integer primary key autoincrement
server	string
type	string
variable	string
value	string
color1	string
color2	string

5.12 nfs

id	integer primary key autoincrement
server	string
export	string
client	string
mountpoint	string

6. The source code

If I want to change anything, I find myself first digging through the source code before I can even begin to add new functionality. This chapter should help finding my way through the code easier.

6.1 TK frame hiërarchy

Most os the actual work in the frames is done in the frame `subframe` which is destroyed and recreated, depending on what is displayed. The `subframe` is set within the `main_frame` because sometimes, destroying and re-creating the `main_frame` causes resizing of the main window and/or flickering.

- — main_window
  - — button_frame
    - — unnamed button "List netork"
    - — unnamed button "Plot network"
    - — unnamed button "Options"
    - — unnamed button "exit"
  - — unnamed label with textvariable $Message
  - — main_frame
    - — subframe: is destroyed and recreated, depending on what to display
      - — On start-up:
        - — unnamed label "Dedefre"
        - — Photo image with djedefre.gif
      - — As network-plot:
        - — nw_info_frame
        - — nw_button_frame
        - — nw_frame
      - — As Options:
        - — Label "Options"
        - — opt_scan_frame
      - — As List network:
        - — list_main_frame

6.1.1 On start-up

The subframe contains:

- — unnamed label "Dedefre"
- — Photo image with djedefre.gif

Not much interaction is possible.

6.2 Modules

6.2.1 *Nwdrawing*

6.2.1.1 Synopsis

Nwdrawing creates a network drawing. The drawing consistes of a drawing and an information field.

The drawing places objects on the map and lines are drawn. Objects may be moved and the lines will automatiically follow. If an object is selected, the information field will be updated for the selected object.

6.2.1.2 Typical use

`nw_del_objects();`	delete all the existing objects, if any
`nw_del_lines();`	delete all the existing lines, if any
`nw_objects(@nw_obj);`	create new objects for the drawing
`nw_lines(@nw);`	create new lines for the drawing
`nw_frame($parent_frame);`	create the drawing in the parent_frame
`nw_callback ('callback-type', \&callback_function);`	Set call-backs for the different call-back types

6.2.1.3 Objects

Objects are placed in an array of hashes. The objects are created with `nw_objects(@nw_obj);` where `@nw_obj` is the array of hashes. The hashes contain the following fields:

push @nw_obj, {		
`newid => $id*$qobjtypes+$objtsubnet,`	always present	must be unique in the array
`id => $id,`	always present	is the id that is used in the callback
`x => $x,`	always present	x-coordinate of the object
`y => $y,`	always present	y-coordinate of the object,
`logo => 'subnet',`	always present	logo used for the object
`name => $name,`	always present	name used for the object
`table => 'subnet',`	always present	table where the object is stored. (table, id) is unique in the array
`pages => @pageslist,`	always present	list of pages where the object is displayed. `push @{$l3_obj[$i]{pages}},'pagename'`
`nwaddress=> $nwaddress,`	present if table=subnet	network address of the subnet
`cidr => $cidr`	present if table=subnet	CIDR-bits of the subnet
`color => $color`	present if table=subnet	color for the subnet.
`status => $status,`	present if type=server	status of the server ('up', 'down','excluded')
`options => $options,`	present if type=server	Possible options, separated by ';'; only `vboxhost:$id` is used.
`ostype => $ostype,`	present if type=server	os-type
`os => $os,`	present if type=server	detailed OS data
`processor => $processor,`	present if type=server	processor type if known
`memory => $memory,`	present if type=server	quantity of memory
`devicetype => $devicetype,`	present if type=server	device-type (server, network, nas, ...)
`interfaces=> @if_list`	present if type=server	list of strings "interface-name ipv4address
`vendor => $vendor,`	present if type=cloud	vendor/provider of the service
`service => $service`	present if type=cloud	service provided via this cloud
}		

6.2.1.4 Lines

Lines are an array of hashes. The lines are set with `nw_lines(@l3_line);`

The hashes contain the following:

```
push @l3_line, {
   from    => $newid1,
   to      => $newid2,
   type    => $color
}
```

6.2.1.5 Call-back

Call-back functions are called if something important changes in the network drawing. Callbacks are set with a call to `nw_callback($type,$func)`. `$func` is a function, for example `\&functionname`. `$type is from the following table:`

type	**arguments**	**what**
color	table,id. color	The color of id in table is changed to color
delete	table,id, name	The object with the id=id must be deleted from the table *table*.
devicetype	table,id,tpchoice	The devicetype id in table is set to tpchoice
merge	table,id,name,merge	Mergs id=id with merge. Merge can be an ID, a name etc.
move	table,id,cx,cy	Move object with id to cx, cy.
name	table,id, name	Give the object id a new name
page	table,id,name,action,page	action='add' or 'del'. Remove or add the object from a page.
type	table,id,tpchoice	Set the object's type to tpchoice.

table is the table in the database. This may be subnet,server, cloud etc.

6.2.2 Multilist

Multilist provides a ascollable table from which an item can be selected.

6.2.2.1 Typical use

`ml_new($parent,$height,$pack_side)`	Create a multilist in `$parent` and pack on the `$pac_side`
`ml_colwidth(@width)`	set the column-widths in the multilist
`ml_colhead(@headers)`	set the labels of the column-headers
`ml_create()`	create the multilist
`ml_insert(@content)`	Add content to the multilist
`ml_callback(\&subroutine)`	Add a call-back for selection
`ml_destroy`	

6.2.2.2 Callback

The subroutine in the callback is called with the value in the first column as argument. Typically, this should be some unique ID.

6.2.3 dje_db

The database interface module. All interfacing with the database should go through this module. This will make it easier to change database when that is necessary.

The module contains a number of basic routines for interfacing with the database:

— `connect_db` : connect to the database

— `db_dosql` : execute an SQL query on the database

— `db_getrow` : get 1 row of results; returns an empty array if there are no more rows

And a number of subs that copy entire tables to arrays:

— `db_get_interfaces`

— `db_get_subnet`

— `db_get_get_server`

— `db_get_db_get_l2`

6.2.3.1 Typical use

`connect_db`	connect to the databas
`db_dosql (` query `)`	do a query on the database
`@row=db_getrow()`	get a row of output

Note that this sequence is not re-entrant; `db_getrow()` will always give a row for the latest `db_dosql` that was executed.

CONTENTS