**Security through Templates**

**Milan, 16 dicember 2022**

# Automation through templates

On real life networks, we possibly have to deal with hundreds or even thousands of routers and switches. They can usually be logically grouped, reflecting the purpose for which they have been bought and configured.

Vendors have different approaches about configurations: some offers both GUI and CLI, some offer a CLI, some other only a GUI. Management ‘brains’ can usually ‘see’ devices, and offer the possibility to define templates. Some of them can even be quite complex and be written through ‘jinja’ syntax (a simple programming language), and possibly containing references to variables or ‘metadata’ that need to be defined per-device.

Both Fortimanager and Cisco EPN/Prime/DNA (by the way, why so many different softwares ?) offer such a way to manage devices. It’s not just a matter of speeding up provisioning and deploy, it’s also a matter of maintaining configuration consistency through thousands of devices, avoiding the usual problems related to people who modify configurations on the fly and forget to follow common procedures. As time goes by, entropy increases and spurious configuration will start magically appearing/disappearing on a few configurations, and such entropy will increase as time passes by. Just joking about it …

A picture containing text, person

Description automatically generated

# Device profiles

Which devices belong to which profile should be a well-known information. Let’s suppose that we want to analyze a group of configurations, retrieving a bunch of ‘common’ configuration lines, that will be part of our template. We could potentially ask such an information to chatGPT, to produce a Python script to retrieve a template from a group of configuration files, using AI and libraries like Sklearn or Tensorflow. If we think about it, one of the tasks that AI could accomplish, is the classification problem. Finding common patterns, even on complex things like video or images, and tell you what’s inside it. So one possible application for such a neural network, could be that of reading 2000 configuration files, and provide as an outcome a number of configuration templates, classifying every device as belonging to a specific ‘profile’. Doable, but definitely **too complex**. You can always save this to sell such a product to a final customer, even though there is no AI at all. You can even try to put some ‘blockchain’ in it, even though this word has lost some traction in the last months.

So let’s start from the fact that the device’s profile has been already decided and can be taken as an input parameter. The target is that of analyzing ‘X’ configurations (‘X’ being a number) belonging to the same profile, and provide a ‘**maximum likelihood**’ template. How could we do it ? To be honest, it’s probably easier than what you would expect. For example, we could simply COUNT the number of occurrences of the same commands, and retrieve our ‘template proposal’, for example the output could be something like the following:

|  |  |
| --- | --- |
| **Command** | **Hits** |
| no service pad | 656 |
| service tcp-keepalives-in | 655 |
| service tcp-keepalives-out | 655 |
| service timestamps debug datetime msec localtime | 655 |
| service timestamps log datetime msec localtime | 654 |
| service password-encryption | 655 |
| no service dhcp | 656 |
| logging buffered 10000 informational | 652 |
| no logging console | 646 |
| enable secret | 638 |
| username Backup secret 5 | 603 |
| aaa new-model | 656 |
| aaa group server radius CUSTOMER | 656 |
| aaa group server radius CUSTOMER  server-private 10.10.22.11 key 7  server-private 10.10.22.12 key 7 | 654 |
| aaa authentication login default group CUSTOMER local | 656 |
| aaa authentication enable default group CUSTOMER enable | 649 |
| aaa authorization exec default group CUSTOMER if-authenticated | 656 |
| aaa accounting system default start-stop group CUSTOMER | 656 |
| aaa session-id common | 656 |
| clock timezone CET 1 0 | 654 |
| power-supply dual | 654 |
| system mtu routing 1500 | 656 |
| no ip domain-lookup | 655 |

Supposing we have a total of 656 configurations, very likely the above commands should be present on ALL devices. If they are not, it’s probably a mistake. On such analysis, we should take care of the fact that some lines are specific for every site, and we should modify them to properly handle them, for example lines related to local usernames:

**username cisco password 7 <omitted>**

**username backup secret 5 <omitted>**

… shouldn’t be considered as such (they would look all different on each device), but should be truncated to:

**username cisco password 7**

**username backup secret 5**

Since we are also interested in CLEANING up the configurations and the spurious lines that have been introduced and configured for any reason without using a template approach, also the ‘less likelihood’ lines could be listed. Of course, there should be a manual analysis here to decide what is right to be there, and what is not.

|  |  |
| --- | --- |
| **Command** | **Hits** |
| logging host 10.10.1.2 | 4 |
| logging host 10.10.5.9 transport udp port 5 | 2 |
| logging source-interface Vlan1100 | 2 |
| logging source-interface Vlan150 | 2 |
| login on-failure log | 2 |
| login on-success log | 2 |
| cdp run | 2 |

## Skipping lines

Some lines should be completely skipped independently from the content, for example the following (unless one specific vrf is configured everywhere, possibly with the same rd values everywhere even though it’s a suboptimal approach):

**vrf definition <vrf>**

rd <AS>:<value>

**!**

**address-family ipv4**

**route-target export <AS>:<rt>**

**route-target import <AS>:<rt>**

**exit-address-family**

**!**

Interfaces configurations COULD be all skipped, but maybe for certain profiles the Gi0 is always used for management purposes and should be configured in the same way everywhere. A description should always be present. An ip address should be there too.

To perform all these checks, **regular expressions** SHOULD be used as a powerful way to filter strings. For example if we are dealing with MPLS\_PE profiles, the bgp configurations should be the same, not only for global parameters but also for the route-reflectors configurations. This can be all handled without many troubles.

Being even more flexible, a dictionary COULD be used to have different regexp for skipping/replacing lines as needed. Everything could be retrieved from an excel file, or be saved/retrieved in json/yaml format from a text file.

## Fixing what is wrong

Once we got such information, we can split the ‘to do’ actions in two:

* a command (those belonging to the first column in the below table) that should be there, is not there in the devices’ configuration
* a command is there, but shouldn’t be there or should be there with a different configuration parameter (remove/clean/change the command)
* a specific command is there on a few devices, and must remain there

From such an analysis, you can fill up another column that could be parsed by a script, which would be responsible to synchronize the configurations as requested (connecting to the target devices via CLI). Just to provide an example of what we’ve explained above:



Beware that with the keyword **‘$(var\_name)‘** (in the above picture you can find a $(hostname) example) we are referring to a ‘VARS’ excel sheet where the per-device variables are defined:



Simple ip subnets operations are allowed, like for example the following:

**$(mgmt\_ip) + 1 = 10.10.0.3 + 1 = 10.10.0.4**

Such an approach could be useful in case we need to add a static route pointing to a specific interface, writing the next-hop ip as the interface’s ip address + 3 (for example).

## Why Security by Template

Most security commands on switches/routers, but also some functional ones, are just commands. Thus having a template that has been checked for any missing command and parameter, or having at least a **unique ‘source of truth’** that gets updated every time a security breach happens (yes, if you find anything before you’re welcome), can prevent bad things from happening.

Just a few examples of security commands, related to access through the console or the line vty:

|  |
| --- |
| line con 0$ |
| line con 0@@@session-timeout 5$ |
| line con 0@@@exec-timeout 5 0$ |
| line con 0@@@logging synchronous level all$ |
| **line con 0@@@no exec$** |
| line con 0@@@stopbits 1$ |
| line vty 0 4$ |
| line vty 0 4@@@session-timeout 5$ |
| **line vty 0 4@@@access-class ssh\_permit in$** |
| line vty 0 4@@@exec-timeout 5 0$ |
| line vty 0 4@@@logging synchronous level all$ |
| **line vty 0 4@@@transport input ssh$** |
| **line vty 0 4@@@transport output none$** |
| line vty 5 15$ |
| line vty 5 15@@@session-timeout 5$ |
| **line vty 5 15@@@access-class ssh\_permit in$** |
| line vty 5 15@@@exec-timeout 5 0$ |
| line vty 5 15@@@logging synchronous level all$ |
| **line vty 5 15@@@transport input ssh$** |
| **line vty 5 15@@@transport output none$** |

Having any of the above commands missing or changed respect to the target template, is a potential security hole in your network. ‘ssh\_permit’ is an access-list that could be checked too for consistency among the devices belonging to the same profile.

### Configuration files indentation

Templates are configuration lines on their own. How can we check for the presence of a line, that must be included as a ‘son’ of another one ?

**ip access-list standard ssh\_permit**

**permit <subnet 1> <mask 1>**

**permit <subnet 2> <mask 2>**

**permit <subnet 3> <mask 3>**

**!**

**ip access-list standard another\_acl**

**permit <subnet 4> <mask 4>**

**permit <subnet 2> <mask 2> 🡨 same subnet/mask, different access-list**

**permit <subnet 5> <mask 5>**

**!**

How can we distinguish between the first and the second access-list ? What we could do is choosing a ‘line separator’ that can’t virtually be never found in the configurations (even the ‘\n’ character), and consider the lines as follows:

**ip access-list standard ssh\_permit@@@permit <subnet 1> <mask 1>**

**ip access-list standard ssh\_permit@@@permit <subnet 2> <mask 2>**

**ip access-list standard ssh\_permit@@@permit <subnet 3> <mask 3>**

**ip access-list standard another\_acl@@@permit <subnet 4> <mask 4>**

**ip access-list standard another\_acl@@@permit <subnet 2> <mask 2>**

**ip access-list standard another\_acl@@@permit <subnet 5> <mask 5>**

… in this way, if we want to check for the presence of <subnet 2>/<mask2> inside access-list ssh\_permit, we can perfectly do it, excluding the chance for false positives.

# Generating and managing templates

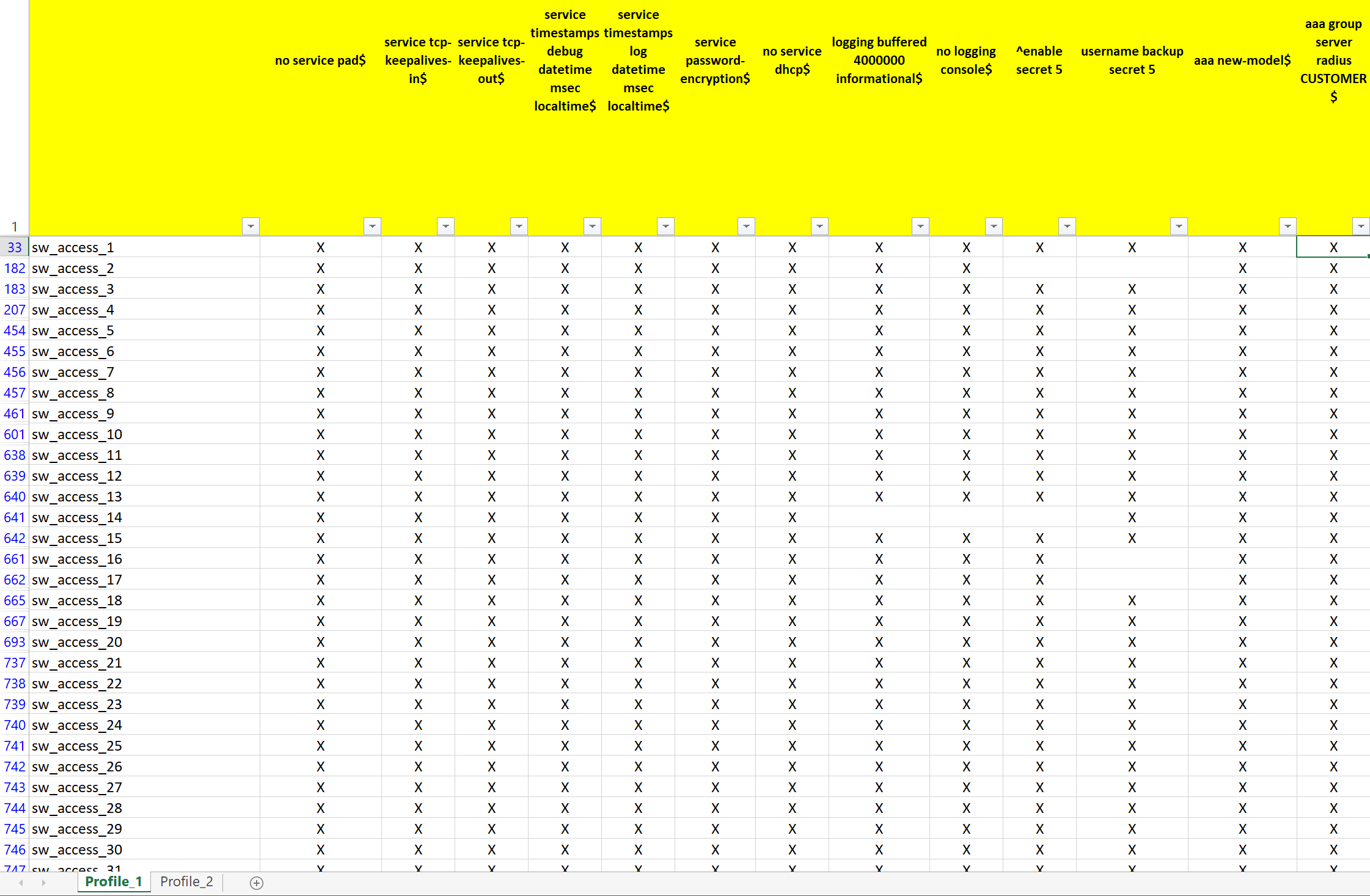
There are three steps that an hypothetical Python script should manage:

1. analyze the available devices’ configurations by profile, and create an output of ‘most likelihood’ template commands, together with the ‘less likelihood’ template commands. The ‘fix command’ column could contain the command that would need to be configured in case it is missing (another command could be for the commands to be removed/changed). For example the ‘enable secret 5’ command could have ‘enable secret 0 my\_clear\_pwd’ in the ‘add command’ column (they are not necessarily always the same, even if in most cases they will).

Calendar

Description automatically generated with medium confidence

1. create an excel file where the rows are the devices, and the columns are a list of the commands produced in the previous step. Every cell will contain an ‘x’ if the command is present, will be empty otherwise.



1. read the above excel file and fix the problems: if there are missing commands, configure them, if there are wrong commands remove/change them. Yes … some attention should be put on the ORDER to avoid problems, but most of the times it’s gonna work (unless you change the routing, you can’t cut your presently used connection to the device).

This could be an example of the list of the devices, that could be used to fix the configurations with the discovered problems. The connection profile should be used by a Python class to connect to the targets, using the correct transport mode (e.g. you connect to a proxy with ssh and from there you could telnet or ssh to the target device).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conn profile** | **Transport** | **Device** | **Mgmt ip** | **Config profile** |
| ssh\_proxy1 | ssh | sw\_site1 | 10.2.3.4 | access\_switch |
| ssh\_proxy1 | telnet | PE\_site\_1 | 10.1.4.5 | mpls\_PE |

# Python script

Do it as an exercise 😊. It’s a good way to start learning !! I will probably provide a version of such a script in the future, publishing it on my github’s page:

<https://github.com/ricky-andre>