Panhandler Documentation

Release 1.0

sp-solutions

Jun 10, 2021

Contents:

CHAPTER 1

Welcome to Panhandler!

Panhandler is a lightweight utility used to aggregate and view or load configuration templates. The primary focus is PAN-OS devices such as the NGFW or Panorama yet may be extended to other elements such as Terraform and 3rd party devices.

Using predefined templates helps fast-track the loading of well known or recommended configurations without extensive searching and scrolling through GUI-click documentation. Each collection of configuration templates are known as *skillets* that are either preloaded into panhandler at runtime or can be manually added as needed.

Skillets can be based on xml, json, text or any other config type used by each device. They are grouped by output action including:

- panos: load into a NGFW and commit
- panorama: load into Panorama and commit
- template: simple text render to the screen
- terraform: deploy infra via Terraform templates
- rest: interact with REST based APIs
- docker: launch docker containers

To load a configuration into a device with panhandler, the user simply has to add the target information for the device to be configured, select the skillet to load, enter the form data, and submit. Panhandler then captures the form data, grabs each configuration element, and loads into the specified device.

1.1 Release History

1.1.1 V4.0

- Released 9-2020

New Features:

- **Skillet Editor** A new UI to edit all aspects of a Skillet.

- **Skillet Creation Tools** This feature allows you to build a skillet from scratch in a number of different ways. For example, you can build a skillet from the differences between two saved configuration files.
- **Improved Terraform Support** Terraform now uses a docker image in the backend, which allows any arbitrary terraform version to be supported. This allows the skillet builder to choose customized docker image containing any version of terraform and supporting libraries.
- **Support for SSH based git repositories** This allows you to use private git repositories as well as push local changes back upstream.

1.1.2 V3.1

- Released 3-2020

New Features:

- **Support for docker type skillets** This brings support for Ansible, Shell scripts, custom binaries, configurable Terraform versions, and more. See github for examples.

1.1.3 V3.0

- Released 2-2020

New Features:

- **New skillet type: pan_validation** This allows PAN-OS configuration file analysis using a jinja language expressions. More example can be found on github.
- **Dynamic UI elements** Allows variables to be shown or hidden based on the value of another variable.
- **New variable types** File uploads, Dynamic lists, new validations and many more.

1.1.4 V2.2

- Released 6-2019

New Features:

- Improved Input validation
- **Python script support with configurable input types.** Script arguments can be passed via cli arguments or as env variables
- **Automatic update detection.** Panhandler will check if you are running the latest and greatest version on startup
- **PAN-OS Skillet debug support** This allows you to verify what is going to be pushed to a PAN-OS device before actually pushing
- **Skillet debug on import** Checks all skillets during repository import for syntax errors
- Collections page now supports filtering and sorting

1.1.5 Example Skillets

Many more examples can be found on Github.

CHAPTER 2

Running Panhandler

The recommended way to run Panhandler is to pull and run the docker container. For Windows users, refer to the Windows installation document.

2.1 Quick Start

The following command will ensure you have the most up to date version of panhandler and will set up all the required ports and volume mounts. This command will also update existing Panhandler containers with the latest released version.

```
curl -s -k -L http://bit.ly/2xui5gM | bash
```

If you don't trust running bit.ly links through Bash, then you can run this variant instead:

```
curl -s -k -L https://raw.githubusercontent.com/PaloAltoNetworks/panhandler/master/ph␣
↪| bash
```

This command will install and or update Panhandler to the latest version using the default values.

If you need special requirements, such as custom volume mounts, non-default username and password, or non-standard ports, you set the following environment variables prior to launching the 'curl' command:

- CNC_USERNAME - Set the default username to login to the application (default paloalto)
- CNC_PASSWORD - Set the default password to login to the application (default panhandler)
- IMAGE_TAG - Set the tag you want to download and install. Possible values: (dev, latest) (default latest)
- DEFAULT_PORT - Set the port the application will listen on for web requests (default 8080)
- FORCE_DEFAULT_PORT- Ensure your desired port will be used regardless of any previously set ports. Possible values are 'true' or 'false'

Note: You must set 'FORCE_DEFAULT_PORT' to 'false' if you change the 'DEFAULT_PORT' to some value other than what was previously set!

2.2 Running the Panhandler Docker Container

If you need to manage the Panhandler container manually:

2.2.1 Using a standard web port

```
docker volume create panhandler_volume
docker run -p 8080:8080 -t -d \
    -v panhandler_volume:/home/cnc_user \
    -v "/var/run/docker.sock:/var/run/docker.sock" \
    -e CNC_USERNAME=paloalto \
    -e CNC_PASSWORD=panhandler \
    --name panhandler paloaltonetworks/panhandler
```

Then access the UI via http://localhost:8080

Changing the values of CNC_USERNAME and CNC_PASSWORD will set the default username and password respectively.

2.2.2 Using an alternate TCP port

If port 8080 is unavailable, you can switch to a different port. This example uses port 9999.

```
docker run -t -p 9999:8080 \
    -v panhandler_volume:/home/cnc_user \
    -v "/var/run/docker.sock:/var/run/docker.sock" \
    -e CNC_USERNAME=paloalto \
    -e CNC_PASSWORD=panhandler \
    --name panhandler paloaltonetworks/panhandler
```

Then access the UI via http://localhost:9999

Note: The -t option for *terminal* allows you to view panhandler output data in the terminal window. This is useful for determining any skillets errors that write to terminal output.

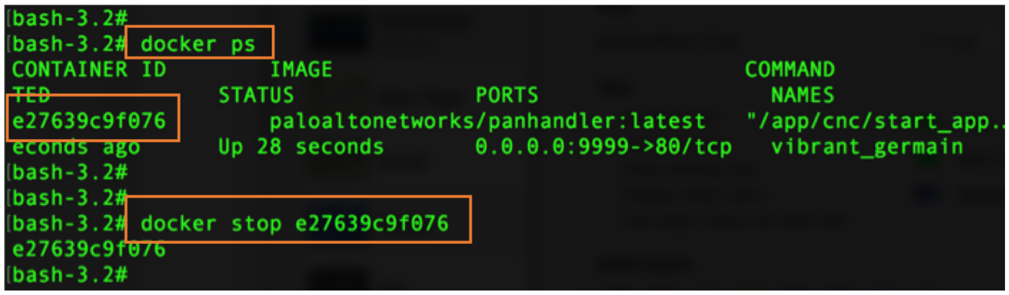
2.2.3 Using Panhandler with TLS

Here is a project that adds TLS to Panhandler: https://github.com/fatofthelan/SecurePanHandler

2.2.4 Stopping the docker container

The docker container runs in the background. You can stop the container by using its container ID.

```
docker ps
docker stop { CONTAINER ID }
```



```
[bash-3.2#
[bash-3.2# docker ps
 CONTAINER ID         IMAGE                             COMMAND
 TED              STATUS              PORTS               NAMES
 e27639c9f076         paloaltonetworks/panhandler:latest   "/app/cnc/start_app..
 econds ago       Up 28 seconds       0.0.0.0:9999->80/tcp   vibrant_germain
[bash-3.2#
[bash-3.2#
[bash-3.2# docker stop e27639c9f076
 e27639c9f076
[bash-3.2#
```

Note: If you need to remove the container, enter *docker rm { CONTAINER ID }* with CONTAINER ID as the ID used to stop. You must stop the container before deleting.

2.3 Building Panhandler

If you want to build panhandler from source (which is not recommended). You will need to update the git submodules, install the pip python requirements for both the app and also CNC, create the local db, and create a local user.

```
git clone https://github.com/PaloAltoNetworks/panhandler.git
cd panhandler
git submodule init
git submodule update
pip install -r requirements.txt
./cnc/manage.py migrate
./cnc/manage.py shell -c "from django.contrib.auth.models import User; User.objects.
↪create_superuser('paloalto', 'admin@example.com', 'panhandler')"
```

2.4 Running Panhandler manually

To start the application on your local machine on port 80:

```
cd panhandler/cnc
celery -A pan_cnc worker --loglevel=info &
manage.py runserver 80
```

To use a different port, supply a different argument to the runserver command above. In this case, the server will start up on port 80. Browse to http://localhost in a web browser to begin. The default login credentials are 'paloalto' and 'panhandler'

2.5 Requirements

Panhandler has been tested to work on Docker version: 18.09.1 (Mac) and 18.09.0 (Linux). Windows users are encouraged to use WSL2.

Please ensure you have the latest docker version installed for the best results.

2.6 Windows Installation

Running panhandler on Windows is possible through docker. The most reliable setup method at this time is to run docker commands directly through PowerShell backed by WSL 2. This process will require multiple reboots so plan accordingly. Other installation methods may not provide appropriate access to the docker daemon from the running panhandler container resulting in certain skillet types not functioning.

2.6.1 Install WSL 2

Begin by installing WSL 2. Microsoft has good documentation on how to do this here:

https://docs.microsoft.com/en-us/windows/wsl/install-win10

If unsure about a Linux distribution to use, choose the latest Ubuntu. Verify you can access WSL 2 before continuing.

2.6.2 Install Docker Desktop

After WSL 2 functionality is verified, install the latest Docker Desktop for Windows using the following tutorial from docker.

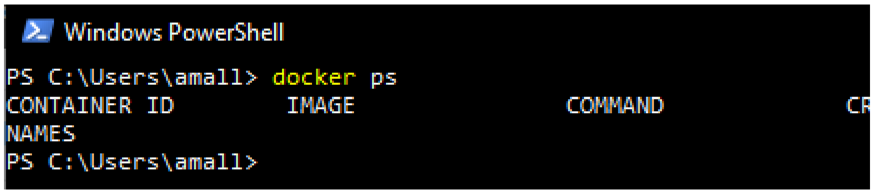
https://docs.docker.com/docker-for-windows/install/

During the install, ensure the following settings:

- Use the WSL2 based engine, using "Hyper-V" may lead to some known problems
- Start Docker Desktop when you login, it will allow panhandler to auto start on boot
- DO NOT select "Expose daemon on tcp://localhost:2375 without TLS"
- DO NOT select "Enable experimental features"
- DO NOT enable "Kubernetes"

Unless the installer states otherwise, these settings can be updated by right clicking the docker icon in your system tray in the bottom right of your Windows screen and selecting "Settings".

Although WSL 2 is required for operation, you will not be using WSL 2 to talk to docker. Open PowerShell and type "*docker ps*" to verify your docker cli is working and able to talk to the docker daemon. You should see output similar to this with no errors. This has to be working before you can proceed.

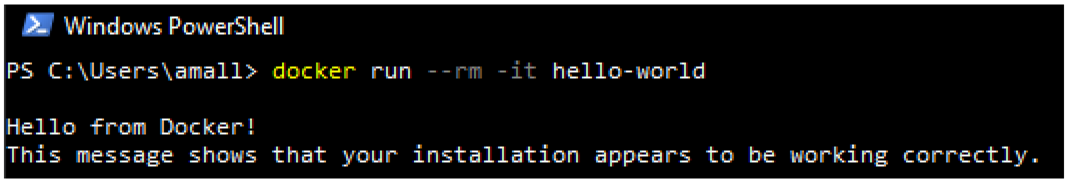


```
PS C:\Users\amall> docker ps
CONTAINER ID        IMAGE               COMMAND             CR
NAMES
PS C:\Users\amall>
```

Another good test to perform to ensure docker is running fine is to run the docker "Hello world" image. From PowerShell type this command:

```
docker run --rm -it hello-world
```

You should get an output similar to this:



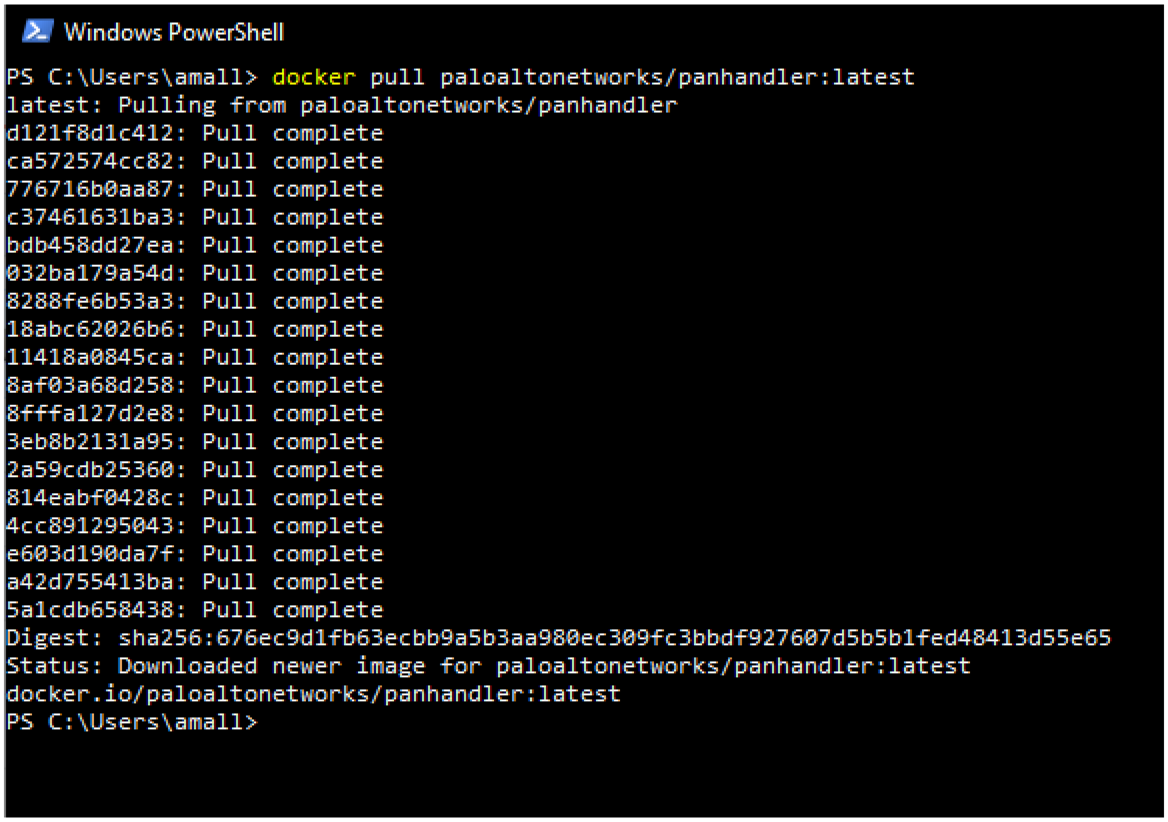
```
PS C:\Users\amall> docker run --rm -it hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.
```

2.6.3 Install Panhandler

At this point, you are ready to install and start panhandler. In PowerShell, issue this command to pull down the latest panhandler image.

```
docker pull paloaltonetworks/panhandler:latest
```

This will take a minute, but you should get output similar to this:



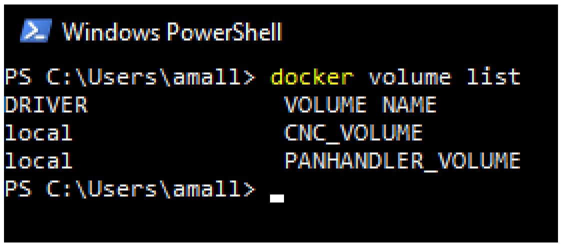
```
PS C:\Users\amall> docker pull paloaltonetworks/panhandler:latest
latest: Pulling from paloaltonetworks/panhandler
d121f8d1c412: Pull complete
ca572574cc82: Pull complete
776716b0aa87: Pull complete
c37461631ba3: Pull complete
bdb458dd27ea: Pull complete
032ba179a54d: Pull complete
8288fe6b53a3: Pull complete
18abc62026b6: Pull complete
11418a0845ca: Pull complete
8af03a68d258: Pull complete
8fffa127d2e8: Pull complete
3eb8b2131a95: Pull complete
2a59cdb25360: Pull complete
814eabf0428c: Pull complete
4cc891295043: Pull complete
e603d190da7f: Pull complete
a42d755413ba: Pull complete
5a1cdb658438: Pull complete
Digest: sha256:676ec9d1fb63ecbb9a5b3aa980ec309fc3bbdf927607d5b5b1fed48413d55e65
Status: Downloaded newer image for paloaltonetworks/panhandler:latest
docker.io/paloaltonetworks/panhandler:latest
PS C:\Users\amall>
```

With the image downloaded, all that's left to do is create the volumes and start panhandler. Docker volumes are virtual storage entities that provides a way to upgrade the image without losing app data. Create the volumes by running these commands:

```
docker volume create CNC_VOLUME
docker volume create PANHANDLER_VOLUME
```

You can verify the volumes have been created by running this command and checking the output matches to the image below:

```
docker volume list
```

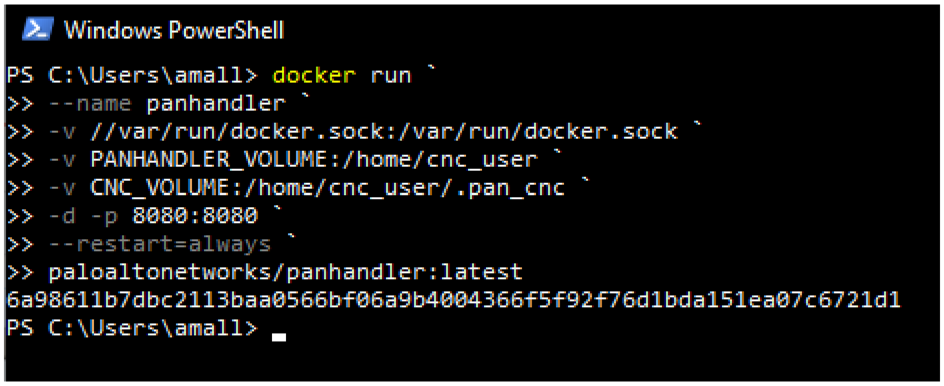


```
PS C:\Users\amall> docker volume list
DRIVER              VOLUME NAME
local               CNC_VOLUME
local               PANHANDLER_VOLUME
PS C:\Users\amall> _
```

Now you can start panhandler by coping this entire command block into PowerShell. This command sets a **restart policy** of *always*, which ensures panhandler will restart with your computer and always run unless you stop it.

```
docker run `
    --name panhandler `
    -v //var/run/docker.sock:/var/run/docker.sock `
    -v PANHANDLER_VOLUME:/home/cnc_user `
    -v CNC_VOLUME:/home/cnc_user/.pan_cnc `
    -d -p 8080:8080 `
    --restart=always `
    paloaltonetworks/panhandler:latest
```

That command will result in a long hash that will serve as the ID for the container, but you can still reference it with the name "*panhandler*".



```
PS C:\Users\amall> docker run `
>> --name panhandler `
>> -v //var/run/docker.sock:/var/run/docker.sock `
>> -v PANHANDLER_VOLUME:/home/cnc_user `
>> -v CNC_VOLUME:/home/cnc_user/.pan_cnc `
>> -d -p 8080:8080 `
>> --restart=always `
>> paloaltonetworks/panhandler:latest
6a98611b7dbc2113baa0566bf06a9b4004366f5f92f76d1bda151ea07c6721d1
PS C:\Users\amall> _
```

After a few seconds, you should be able to access panhandler in your web browser by browsing to:

http://localhost:8080/

The installation process is now complete.

2.6.4 Stopping and Starting Panhandler

If you wish to stop panhandler from running until you restart it, you can do so with the PowerShell command:

```
docker stop panhandler
```

Likewise, this process can be restarted with the command:

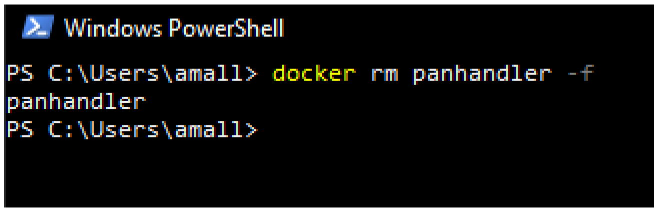
```
docker start panhandler
```

2.6.5 Upgrading Panhandler

Only one more command is required to upgrade panhandler. The process is to delete the old container, update the image, and start a new container.

You can delete the old container, running or stopped, with this command:

```
docker container rm panhandler -f
```



```
Windows PowerShell
PS C:\Users\amall> docker rm panhandler -f
panhandler
PS C:\Users\amall>
```

You then can use the 'docker pull' and 'docker run' commands exactly as they are above to download a newer panhandler image and start it. The volumes you created earlier will be still be available and assigned to the new container if you use the commands as they are.

2.6.6 Troubleshooting Windows Install

If you run into either of the following errors when trying to install a Linux distribution:

> Installing, this may take a few minutes… WslRegisterDistribution failed with error: 0x80370102 Error: 0x80370102 The virtual machine could not be started because a required feature is not installed.

or when trying to run the Docker Desktop GUI:

> Hardware assisted virtualization and data execution protection must be enabled in BIOS.

After verifying that virtualization is enabled in BIOS by opening *Task Manager* > *Performance* > *Virtualization*, please attempt the following steps.

1. If the Windows' Hyper-V feature is totally disabled or not installed, enable Hyper-V by opening the PowerShell as administrator and running the following command:

```
dism.exe /Online /Enable-Feature:Microsoft-Hyper-V /All
```

2. If the Windows' Hyper-Vfeature is enabled and not working, enable Hypervisor with the following command and restart your system:

```
bcdedit /set hypervisorlaunchtype auto
```

3. If the problem persists probably Hyper-V on your system is corrupted, so turn off all Hyper-V related Windows' features under *Control Panel* > *Programs* > *Windows Features*. Restart your system and attempt to enable Hyper-V again.

This troubleshooting guide was found from:

> https://stackoverflow.com/questions/39684974/docker-for-windows-error-hardware-assisted-virtualization-and-data-execution-p

2.7 Switching between Latest and Develop Containers

PanHandler runs in a Docker container, the main build tagged as 'latest'.

There is also a develop branch with new features and updates. Although not the recommended release, some users may want to work with develop and explore new features. Some skillets being developed may also be dependent on newer features.

2.7.1 Updating the Running Latest Version

This script will install or update to the latest 'dev' image for Panhandler. This is recommended for developers or power-users who understand this code may be unstable and not all features may work all the time.

```
curl -s -k -L http://bit.ly/34kXVEn  | bash
```

The following bash script can be copy-pasted into the terminal to stop the PanHandler process, pull the latest, and run again. The example uses port 9999 for web access.

```
export PANHANDLER_IMAGE=paloaltonetworks/panhandler
export PANHANDLER_ID=$(docker ps | grep $PANHANDLER_IMAGE | awk '{ print $1 }')
docker stop $PANHANDLER_ID
docker rm -f $PANHANDLER_ID
docker pull $PANHANDLER_IMAGE
docker run -t -p 9999:80 -t -v $HOME/.pan_cnc:/home/cnc_user/.pan_cnc $PANHANDLER_
↪IMAGE
```

2.7.2 Updating the Running Develop Version

The following bash script can be copy-pasted into the terminal to stop the PanHandler process, pull the develop version, and run again. The example uses port 9999 for web access.

```
export PANHANDLER_IMAGE=paloaltonetworks/panhandler:dev
export PANHANDLER_ID=$(docker ps | grep $PANHANDLER_IMAGE | awk '{ print $1 }')
docker stop $PANHANDLER_ID
docker rm -f $PANHANDLER_ID
docker pull $PANHANDLER_IMAGE
docker run -t -p 9999:80 -t -v $HOME/.pan_cnc:/home/cnc_user/.pan_cnc $PANHANDLER_
↪IMAGE_D
```

2.7.3 Switching from Latest to Develop

These commands still stop the latest main release version then pull down and run the latest develop version. The latest release container will be deleted.

```
export PANHANDLER_IMAGE_M=paloaltonetworks/panhandler
export PANHANDLER_IMAGE_D=paloaltonetworks/panhandler:dev
export PANHANDLER_ID=$(docker ps | grep $PANHANDLER_IMAGE_M | awk '{ print $1 }')
docker stop $PANHANDLER_ID
docker rm -f $PANHANDLER_ID
docker pull $PANHANDLER_IMAGE_D
docker run -t -p 9999:80 -t -v $HOME/.pan_cnc:/home/cnc_user/.pan_cnc $PANHANDLER_
↪IMAGE_D
```

2.7.4 Switching from Develop to Latest

These commands still stop the develop version then pull down and run the latest main release version. The develop version container will be deleted.

```
export PANHANDLER_IMAGE_M=paloaltonetworks/panhandler
export PANHANDLER_IMAGE_D=paloaltonetworks/panhandler:dev
export PANHANDLER_ID=$(docker ps | grep $PANHANDLER_IMAGE_D | awk '{ print $1 }')
docker stop $PANHANDLER_ID
docker rm -f $PANHANDLER_ID
docker pull $PANHANDLER_IMAGE_M
docker run -t -p 9999:80 -t -v $HOME/.pan_cnc:/home/cnc_user/.pan_cnc $PANHANDLER_
↪IMAGE_M
```

When switching between dev and latest clear the cache with the following link:

http://localhost:9999/clear_cache

2.8 Keeping Up to Date

As panhandler is a quickly evolving project with new features added frequently, it is advisable to ensure you update to the latest periodically.

2.8.1 Update Script

The following script is useful to update your version of Panhandler to the latest while retaining all your settings, port mappings, etc.

```
curl -s -k -L http://bit.ly/2xui5gM | bash
```

This script will pull down a bash script that will determine if your version of Panhandler is the latest. If not, it will pull the latest image from Docker Hub, remove the old container and create a new container with the same port mapping as the previous version.

Note: If you are upgrading from a very old Panhandler version, you may need to import Skillet repositories again.

2.8.2 Manually updating the Panhandler Container

Panhandler is primarily distributed as a docker image on Docker Hub. To ensure you have the latest version, check for new releases here. To manually launch a newer version via docker:

```
docker pull paloaltonetworks/panhandler:latest
docker run -p 8080:8080 -t -v $HOME:/home/cnc_user paloaltonetworks/panhandler
```

This will create a container based on the latest image tag. Versioned panhandler images are also available and can be found on Docker Hub.

Note: You must periodically pull new images from Docker hub to ensure you have the latest software with new features and bug fixes.

To ensure you have the most up to date software, perform a docker pull and specify your desired release tag.

```
export TAG=latest
docker pull paloaltonetworks/panhandler:$TAG
docker run -p 8080:8080 -t -v $HOME:/home/cnc_user paloaltonetworks/panhandler:$TAG
```

2.8.3 Ensuring your Panhandler container is using the latest image

If you already have Panhandler running, you may need to use the following commands to first stop the existing container. Note the image tag in the PANHANDLER_IMAGE variable below. You may want to change this to 'latest' or some other specific release tag like '2.2'

```
export PANHANDLER_IMAGE=paloaltonetworks/panhandler:dev
export PANHANDLER_ID=$(docker ps | grep $PANHANDLER_IMAGE | awk '{ print $1 }')
docker stop $PANHANDLER_ID
docker rm -f $PANHANDLER_ID
docker pull $PANHANDLER_IMAGE
docker run -p 8080:8080 -t -v $HOME:/home/cnc_user -d $PANHANDLER_IMAGE
```

2.8.4 Cleaning up old versions

Once you update to a newer version of Panhandler, the older images can still be left around, taking up space on your hard drive. A common best practice is to occasionally remove old images with the following docker command:

```
docker image prune
```

Note: This command may take some time to complete, up to several minutes. The longer it takes, the more space it's saving on your hard drive!

On my system, this command can regularly reclaim over 10GB of space.

Another good command to occasionally run is:

```
docker container prune
```

This will remove all stopped containers and recover their used disk space as well.

CHAPTER 3

Using Panhandler

Once installed and running, use your web browser to access panhandler.

3.1 Access the web portal

For your local device:

> http://localhost:80 (for a standard web port)
>
> http://localhost:9999 (using a defined port, eg. 9999)

The default username and password is: *paloalto* and *panhandler*

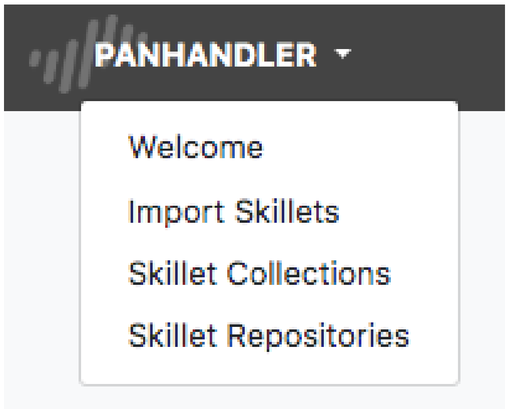
3.2 Set the Configuration Target

Before choosing skillets to load, set the configuration target IP and username/password credentials. This stores the device credentials to be used for API access.

Jump to *Panhandler Environments* to set the environment.

3.3 Choose Skillets to View by Collection

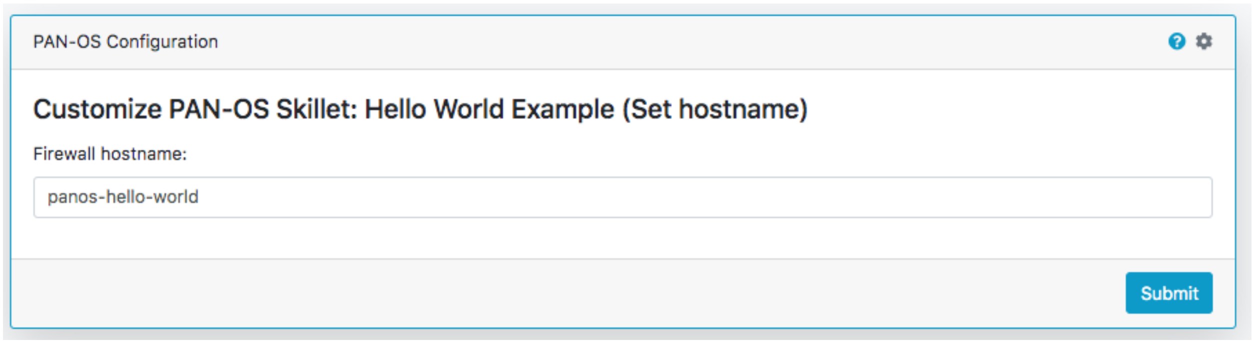
From the main panhandler menu, select *Skillet Collections* to see available Skillet Collections. A collection is a group of Skillets.



Select *Go* on the card for the desired collection to see all Skillets that belong to that collection. Any Skillet builder can create their own collection.

3.4 Select the Skillet to Load

A list of templates will be available to load into your device. Select the desired item and enter the form data.

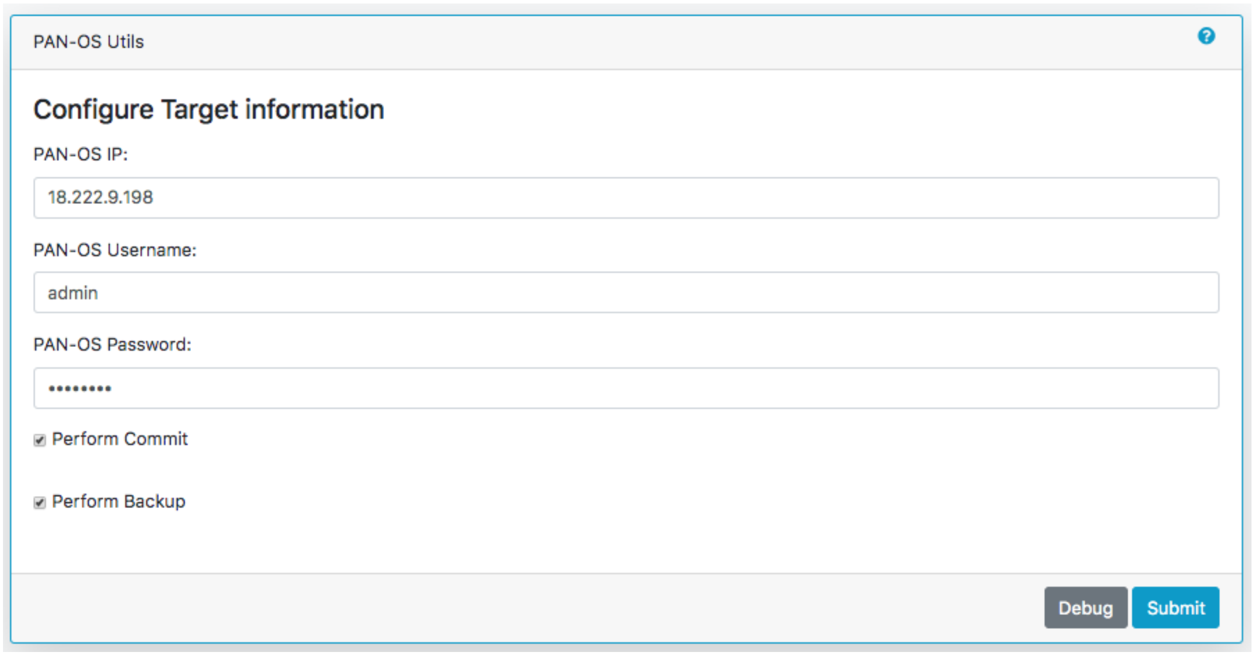

PAN-OS Configuration

Customize PAN-OS Skillet: Hello World Example (Set hostname)

Firewall hostname:

panos-hello-world

The final form will be the target information for API config loading. Confirm the correct values and submit.


PAN-OS Utils

Configure Target information

PAN-OS IP: 18.222.9.198

PAN-OS Username: admin

PAN-OS Password: ••••••••

✓ Perform Commit

✓ Perform Backup

For PAN-OS types, you can choose to check or uncheck the 'Perform Commit' option to push the configuration then do a 'commit' or only push the configuration without a commit.

You can also check or uncheck the 'Perform Backup' option to create a named configuration backup on the device prior to pushing the new configuration. This provides a roll back mechanism should you desire. The named backups will be named with the following format: *panhandler-20190101000000.xml* (panhandler followed by the current timestamp)

Warning: Validate the device type and software version matches the skillet. For example, you will get errors if trying to load a Panorama template into a firewall. There are also cases where you cannot mix sofware versions and loading a v8.1 configuration into a v8.0 device will result in errors.

Warning: Some templates may have dependencies requiring elements to be previously loaded into the system or from other templates. Examples may be certificates, security objects, log forwarding profiles, etc. Check template documentation and look for any specific dependencies.

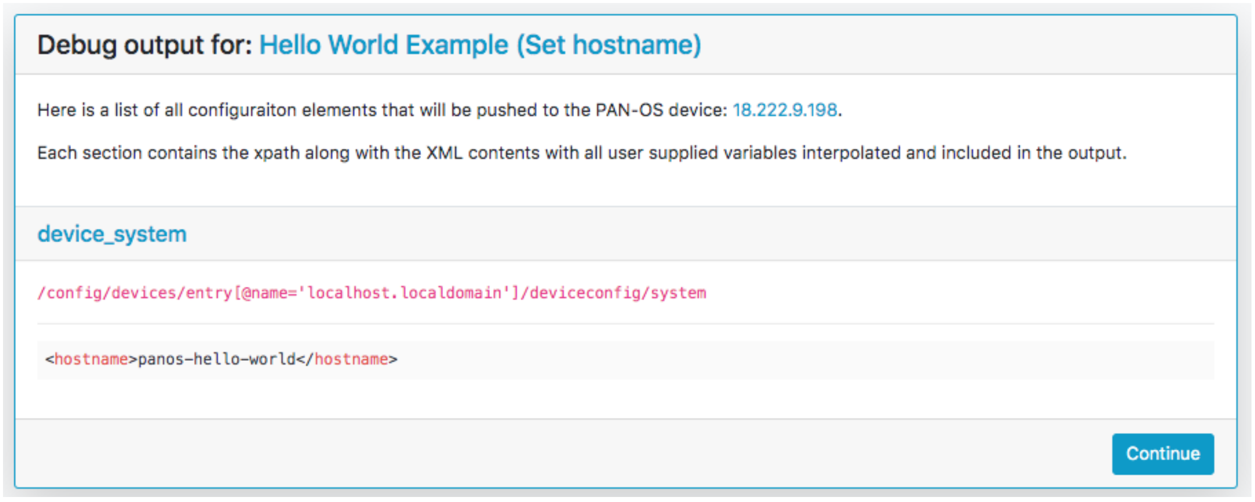
Once the load has completed, you can select another template to load to the same device or choose another Environment to load a configuration to another device.

Note: Commit operations are queued in the background on the device. If you chose to commit the configuration on the edit target screen, then a *Job ID* will be displayed in the success message. You can then use this Job ID to view the status of this commit operation either via a Skillet or on the PAN-OS device directly.

3.5 Understanding what will be pushed

You have two options to examine what configurations will be pushed by a skillet. The first, is to simply uncheck the 'Perform Commit' checkbox. Then you can log into the device and issue a *show config diff* command from the CLI.

You can also select the 'Debug' button from the Edit Target screen. This will display a list of all fully rendered XML snippets and the xpaths where they will be inserted into the configuration heirarchy.



Debug output for: Hello World Example (Set hostname)

Here is a list of all configuraiton elements that will be pushed to the PAN-OS device: 18.222.9.198.

Each section contains the xpath along with the XML contents with all user supplied variables interpolated and included in the output.

device_system

```
/config/devices/entry[@name='localhost.localdomain']/deviceconfig/system
```

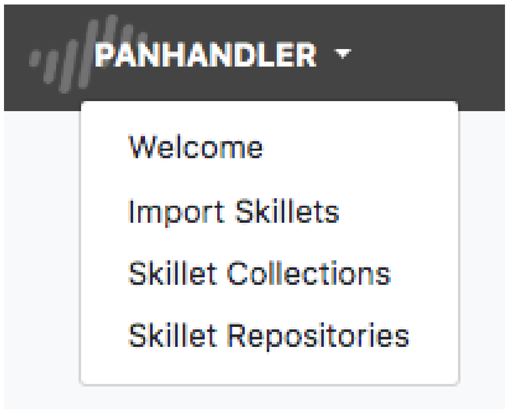
```
<hostname>panos-hello-world</hostname>
```

3.6 Adding a New Skillet Repository

Panhandler is preloaded with a wide set of skillets yet you may still have to manually add skillet repos.

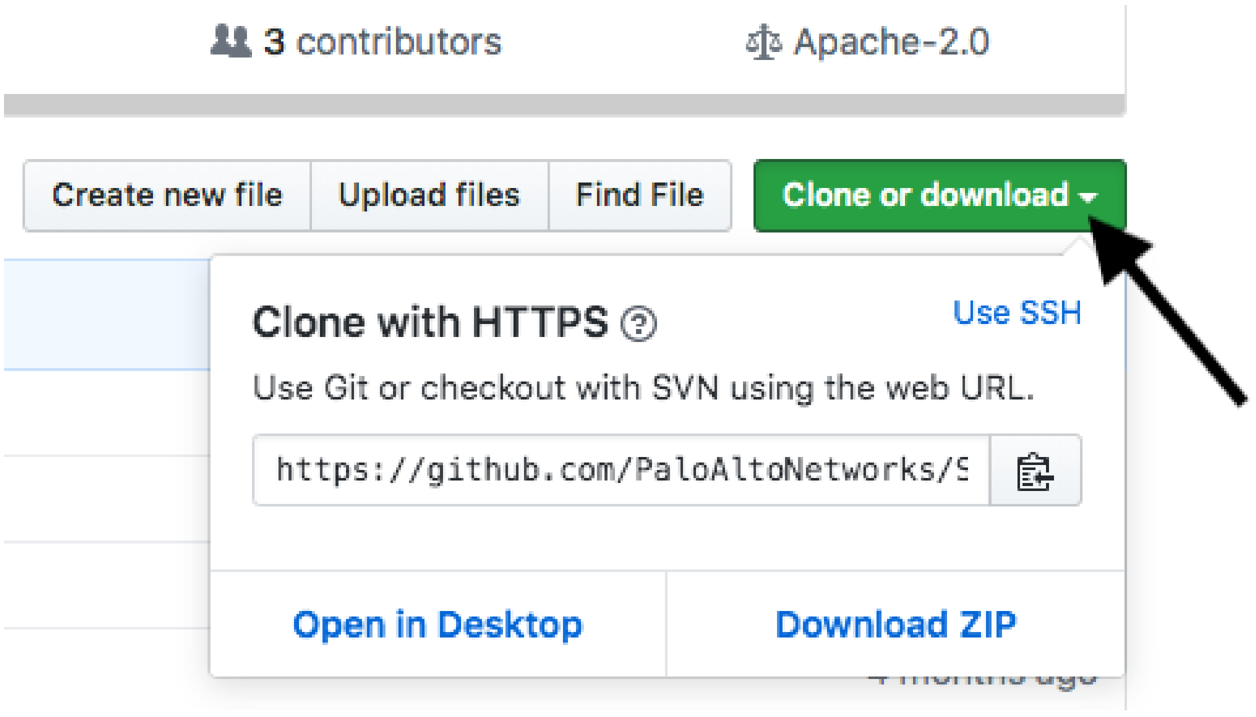
3.6.1 Import a New Skillet

From the main menu, choose *Import Skillets*.



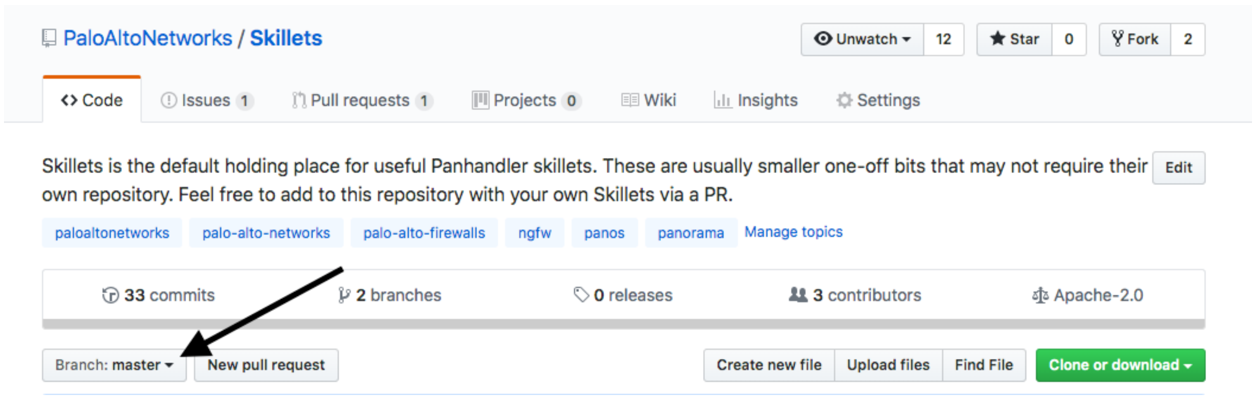
The import repository fields allow you to specify the repo name and URL to import. You may import repositories from any git server, including GitHub, gitlab, gogs, etc.

To import a repository from Github, click on the 'Clone or Download' button and copy the full HTTPS link shown.



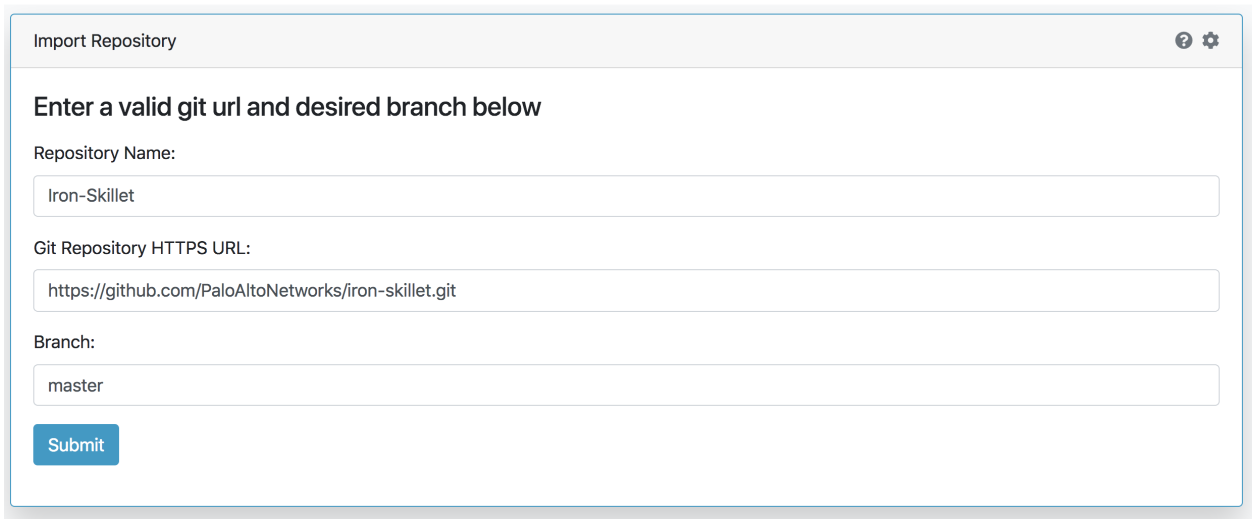
Warning: Private Repositories must use the SSH based URL. You must also import your Panhandler SSH Key into your private repository.

Also, note which branch you want to import. The list of available branches can be found in Github by clicking the 'Branch: master' button on the main page of the repository.



Skillets is the default holding place for useful Panhandler skillets. These are usually smaller one-off bits that may not require their own repository. Feel free to add to this repository with your own Skillets via a PR.

Enter this information in the 'Import Skillets' form to import the repository and gain access to the Skillets contained within.


Import Repository

Enter a valid git url and desired branch below

Repository Name: Iron-Skillet

Git Repository HTTPS URL: https://github.com/PaloAltoNetworks/iron-skillet.git

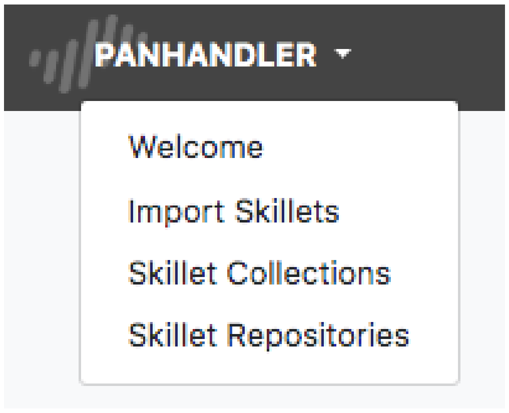
Branch: master

Once successful, you will see the complete list of imported repositories including the newly added repo.

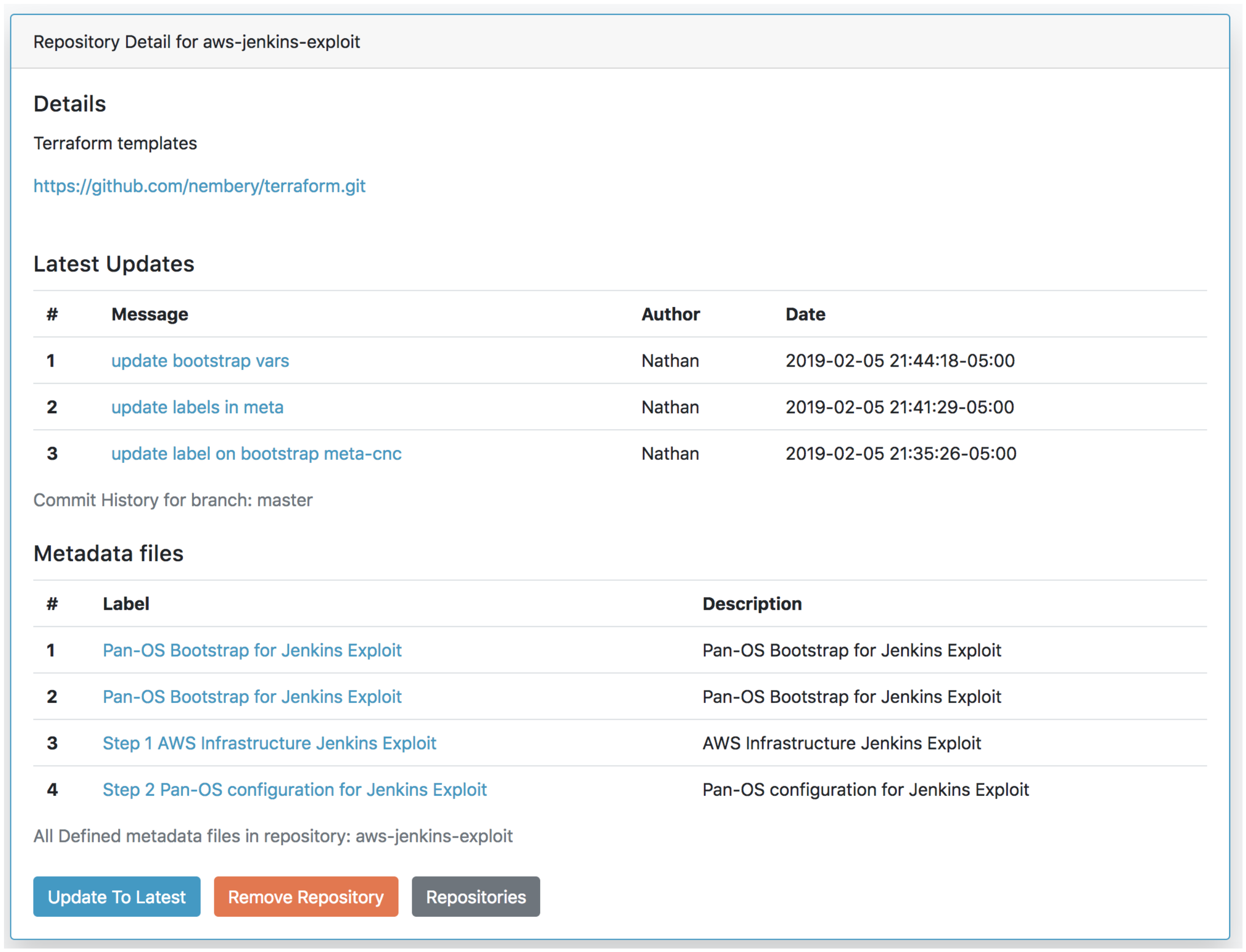
At this stage, going to the *Template Library* will show any additional skillets in their respective categories.

3.6.2 Update a Skillet Repository

From the main menu, choose *Repositories*.



Click on *Details* for the repository of interest.



Repository Detail for aws-jenkins-exploit

Details

Terraform templates

https://github.com/nembery/terraform.git

Latest Updates

#	Message	Author	Date
1	update bootstrap vars	Nathan	2019-02-05 21:44:18-05:00
2	update labels in meta	Nathan	2019-02-05 21:41:29-05:00
3	update label on bootstrap meta-cnc	Nathan	2019-02-05 21:35:26-05:00

Commit History for branch: master

Metadata files

#	Label	Description
1	Pan-OS Bootstrap for Jenkins Exploit	Pan-OS Bootstrap for Jenkins Exploit
2	Pan-OS Bootstrap for Jenkins Exploit	Pan-OS Bootstrap for Jenkins Exploit
3	Step 1 AWS Infrastructure Jenkins Exploit	AWS Infrastructure Jenkins Exploit
4	Step 2 Pan-OS configuration for Jenkins Exploit	Pan-OS configuration for Jenkins Exploit

All Defined metadata files in repository: aws-jenkins-exploit

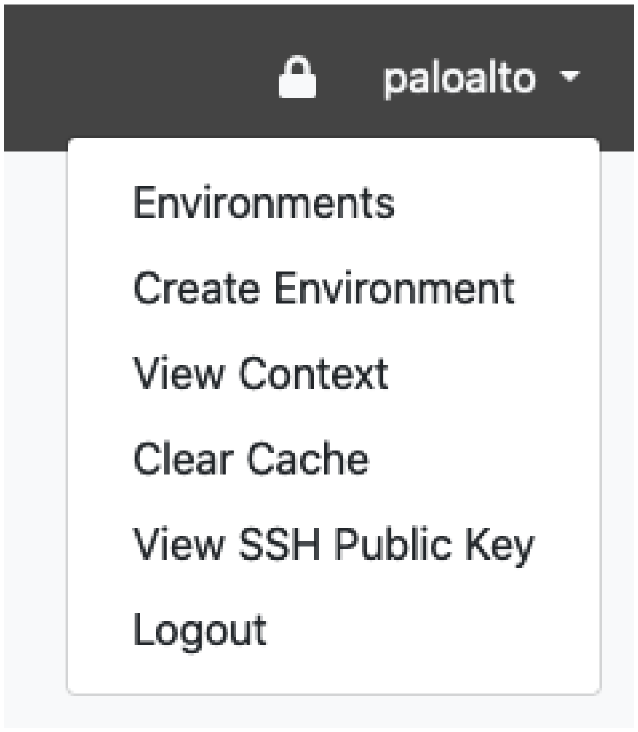
The repo window will show a description of the repo along with the last few content changes.

Choose *Update to Latest* to check for and pull template updates.

Note: *Already up to date* will show that no changes were made to the source skillet and no udpates required.

3.6.3 Using a Private Git Repository

In order to use private repositories, you must first import the Panhandler public SSH key into your upstream repository or account.



Use the ‘View SSH Public Key’ option in the user menu to see the autogenerated key for Panhandler.

Instructions for importing this key into your repository can be found here:

- GitHub
- GitLab
- BitBucket
- Others

Warning: You must use the SSH based git URL when importing your private repository as HTTPS authentication is not supported!

3.7 Panhandler Environments

Often times, it is desirable to store environment specific data outside of a git repository. Panhandler provides a mechanism to do this using ‘Environments’.

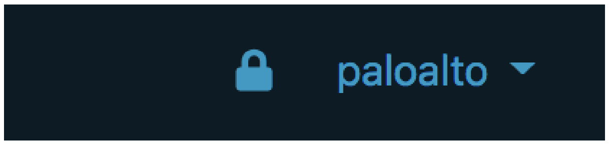
3.7.1 What is an Environment

An environment is a collection of secrets that can be loaded and managed as a unit. For example, you may want to keep all AWS related secrets together in an environment called ‘AWS’. When panhandler displays a web form from a configuration set, any variables from the configuration template that share a name with a secret in the currently loaded environment, that value will be pre-populated.

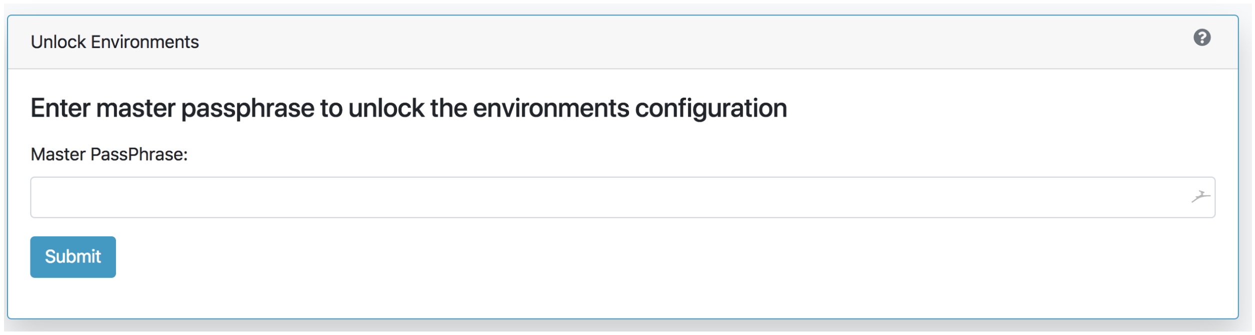
This is especially useful if you have multiple environments such as ‘AWS-QA’, ‘AWS-PROD’, and ‘AWS-DEV’.

3.7.2 Unlocking Environments

To load an environment, click on the 'lock' icon on the right of the navigation bar.



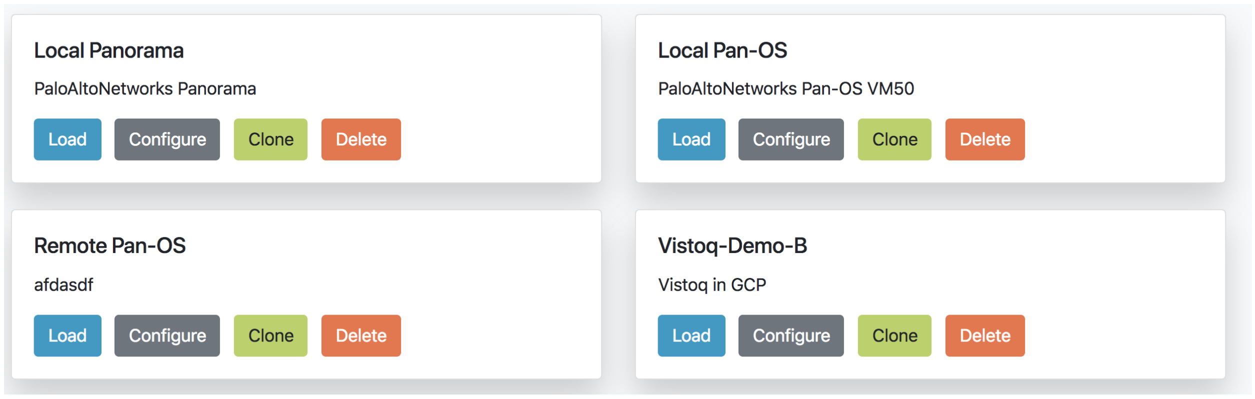
You will be presented with an unlock password dialog. This password will be used to protect any secrets you store in your environments in an encrypted file in your home directory. If this encrypted file does not already exist it will be created and protected with the password you enter here.


Unlock Environments

Enter master passphrase to unlock the environments configuration

Master PassPhrase:

Once unlocked, you can manage your environments by creating new ones, cloning, configuring, or deleting existing ones.


Local Panorama

PaloAltoNetworks Panorama

Local Pan-OS

PaloAltoNetworks Pan-OS VM50

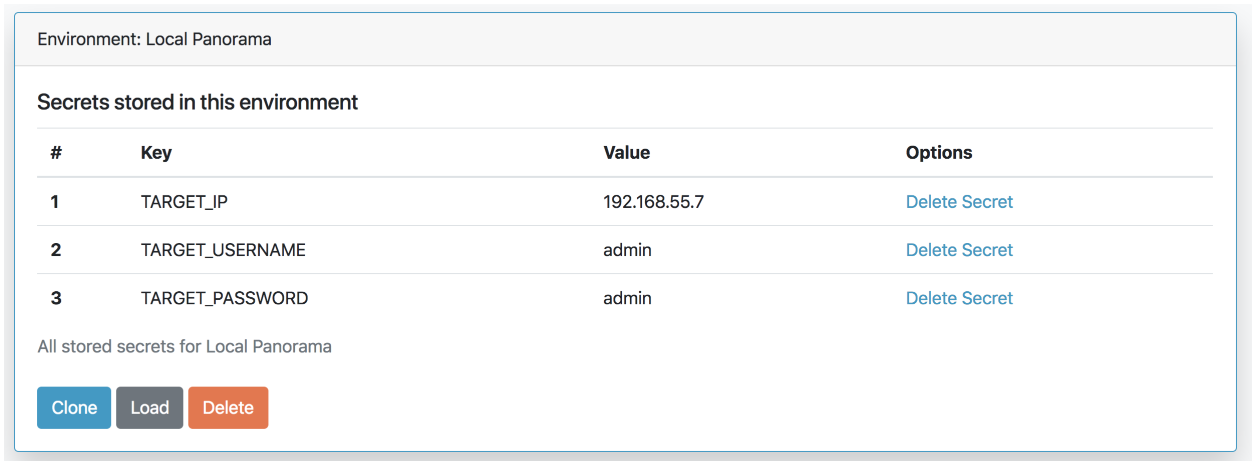
Remote Pan-OS

afdasdf

Vistoq-Demo-B

Vistoq in GCP

Choosing the 'Configure' option on an environment allows you to add, remove, or overwrite secrets stored within them.



Environment: Local Panorama

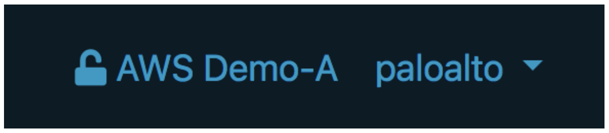
Secrets stored in this environment

#	Key	Value	Options
1	TARGET_IP	192.168.55.7	Delete Secret
2	TARGET_USERNAME	admin	Delete Secret
3	TARGET_PASSWORD	admin	Delete Secret

All stored secrets for Local Panorama

Choosing to 'Load' an environment makes that env available to pre-populate template fields. It will also be available as a 'pop-over' that you can use to copy and paste secrets into template fields. This is useful when you want to store secrets like API_KEYS

Note: Template variables that share the same 'name' as a secret in the currently loaded environment will be pre-populated with the value of that secret. You can find the exact name of a specific variable field by looking at the '.meta-cnc.yaml' file for that form.



CHAPTER 4

Skillets

The heart of Panhandler is the *.skillet.yaml* file. This allows a set of configuration snippets, known as a skillet, to be shared and consumed as a single unit. For example, to configure a default security profile you may need to configure multiple different parts of the PAN-OS configuration. Panhandler allows you to group those different 'pieces' and share them among different devices as a single unit. Often times these configuration bits (affectionately called 'skillets') need slight customization before deployment to a new device. The *.skillet.yaml* file provides a means to templatize these configurations and present a list of customization points, or variables, to the end user or consumer.

4.1 IronSkillet

The very first, and most well known, Skillet is IronSkillet. This was developed as a way to share best practice Day One configurations in an easy to deploy manner without requiring 'a million clicks'.

Much more information about IronSkilet can be found on Readthedocs.

4.2 Basic concepts

In order to add multiple 'bits' of configuration to a device, we need to know the following things:

- XML Configuration fragment with optional variables defined in jinja2 format
- XPath where this xml fragment should be inserted into the candidate configuration
- the order in which these XML fragments must be inserted
- a list of all variables that require user input
- target version requirements. For example: PAN-OS 8.0 or higher

This is all accomplished by adding multiple files each containing an XML configuration fragment and a *.skillet.yaml* file that describes the load order, variables, target requirements, etc.

4.3 YAML syntax

Each *skillet* is structured as a series of files in a single directory. This directory may contain a number of template files (XML, YAML, JSON, etc) and a *.skillet.yaml* file. Note the following:

1. A *.skillet.yaml* file that is formatted with using YAML with the following format:

```
name: config_set_id
label: human readable text string
description: human readable long form text describing this Skillet

labels:
  collection:
    - Example Skillets

variables:
  - name: INF_NAME
    description: Interface Name
    default: Ethernet1/1
    type_hint: text

snippets:
  - xpath: some/xpath/value/here
    name: config_set_knickname
    file: filename of xml snippet to load that should exist in this directory
```

Note: You may also use an 'element' attribute instead of the 'file' attribute if you would rather include the XML fragment inline as opposed to in a separate file.

2. Multiple configuration files. Each should contain a valid template fragment and may use jinja2 variables. These templates may be XML, JSON, YAML, Text, etc. For PAN-OS devices, these are XML fragments from specific stanzas of the PAN-OS device configuration tree.

4.4 Metadata details

Each .skillet.yaml file must contain the following top-level attributes:

- name: unique name of this Skillet
- label: Human readable label that will be displayed in the Panhandler UI
- description: Short description to give specific information about what this Skillet does
- type: The type of skillet. This can be 'panos', 'panorama', 'rest', or others.
- variables: Described in detail below
- snippets: a list od dicts. The required attributes vary according to Skillet tupe

Optional top level attributes:

- depends: List of dicts containing repository urls and branches that this skillet depends on
- **labels: Extensible list of key/value pairs that offers additional, optional, functionality. See here for a** complete list Labels.

Note: Each Metadata file type has it's own format for the 'snippets' section. *file* and *xpath* are only used in *panos* and *panorama* types. Other types such as *template* or *rest* may have a different format.

4.4.1 Skillet Collections

Each Skillet should belong to at least one 'Collection'. Collections are used to group like skillets. SKillets with no *collection* label will be placed in the 'Unknown' Collection.

To configure one or more collections for your Skillet, add a *collection* attribute to the 'labels' dictionary.

```
labels:
  collection:
    - Example Skillets
    - Another Collection
    - Yet another Collection
```

See Labels for a complete list of all labels supported by Panhandler.

4.4.2 Snippet details per Metadata type

Required fields for each metadata type is listed below:

- **panos, panorama, panorama-gpcs**
    - name - name of this snippet
    - cmd - operation to perform. Default is 'set'. Any valid PAN-OS API Command is accepted (set, edit, override, get, show, etc)
    - xpath - XPath where this fragment belongs
    - file - path to the XML fragment to load and parse
    - element - inline XML fragment to load and parse. Can be used in leu of a separate 'file' field

  See Example here: example_panos
- **pan_validation**
    - name - name of the validation test to perform
    - cmd - validate, validate_xml, noop, or parse. Default is validate
    - test - Boolean test to perform using jinja expressions

  See Example here: example_validation
- **template**
    - name - name of this snippet
    - file - path to the jinja2 template to load and parse
    - template_title - Optional title to include in rendered output
- **terraform**
    - None - snippets are not used for terraform

  See Example here: example_terraform

- **rest**
  - name - unique name for this rest operation
  - path - REST URL path component *path: http://host/api/?type=keygen&user={{ username }}&password={{ password }}*
  - operation - type of REST operation (GET, POST, DELETE, etc)
  - **payload - path to a jinja2 template to load and parse to be send as POSTed payload**

    Note: For x-www-form-urlencded this must be a json dictionary

  - **headers - a dict of key value pairs to add to the http headers**

    Note: for example: *Content-Type: application/json*

  See Example here: example_rest and here: example_rest_with_output

- **python3**
  - name - name of the script to execute
  - file - relative path to the python script to execute
  - input_type - Optional type of input required for this script. Valid options are 'cli' or 'env'. This will determine how user input variables will be passed into into the script. The default is 'cli' and will pass variables as long form arguments to the script in the form of *–username=user_input* where *username* is the name of the variable defined in the *variables* section and *user_input* is the value entered for that variable from the user. The other option, 'env' use cause all defined variables to be set in the environment of the python process.

  See Example here: example_python

4.5 Defining Variables for User input

Each skillet can define multiple variables that will be interpolated using the Jinja2 templating language. Each variable defined in the *variables* list should define the following:

1. name: The name of the variable found in the skillets. For example:

```
{{ name }}
```

2. description: A brief description of the variable and it's purpose in the configuration. This will be rendered as the field label in the UI.
3. default: A valid default value which will be used if no value is provided by the user.
4. type_hint: Used to constrain the types of values accepted. May be implemented by additional third party tools. Examples are *text*, *text_field*, *ip_address*, *password*, *dropdown*, and *checkbox*.
5. force_default: The UI will be pre-populated with a value from the loaded environment or with a previously entered value unless this value is set to True. The default is False. Setting to True will ensure the default value will always be rendered in the panhandler UI.
6. required: Determines if a value is required for this field. The default is False.

7. help_text: Optional attribute that will be displayed immediately under the field. This is useful for giving extra information to the user about the purpose of a field.

Note: The variable name must not contain special characters such as '-' or '*' or spaces. Variable names can be any length and can consist of uppercase and lowercase letters (A-Z , a-z), digits (0-9), and the underscore character (_). An additional restriction is that, although a variable name can contain digits, the first character of a variable name cannot be a digit.

4.5.1 Variable Example:

Here is an example variable declaration.

```
- name: FW_NAME
  description: Firewall hostname
  default: panos-01
  type_hint: text
  help_text: Hostname for this firewall.
  allow_special_characters: false
  attributes:
    min: 6
    max: 256
```

See Variables for a complete reference of all available type_hints.

4.6 Hints

4.6.1 Ensuring all variables are defined

When working with a large amount of configuration temlates, it's easy to miss a variable definition. Use this one-liner to find them all.

cd into a skillet dir and run this to find all configured variables:

```
grep -r '{{' . |  cut -d'{' -f3 | awk '{ print $1 }' | sort -u
```

Of, if you have *perl* available, the following may also catch any configuration commands that may have more than one variable defined:

```
grep -r '{{' . | perl -pne 'chomp(); s/.*?{{ (.*?) }}/$1\n/g;' | sort -u
```

4.6.2 YAML Syntax

YAML is notoriously finicky about whitespace and formatting. While it's a relatively simple structure and easy to learn, it can often also be frustrating to work with, especially for large files. A good reference to use to check your YAML syntax is the YAML Lint site.

4.6.3 Jinja Whitespace control

Care must usually be taken to ensure no extra whitespace creeps into your templates due to Jinja looping constructs or control characters. For example, consider the following fragment:

```
<dns-servers>
{% for member in CLIENT_DNS_SUFFIX %}
    <member>{{ member }}</member>
{% endfor %}
</dns-servers>
```

This fragment will result in blank lines being inserted where the ‘for’ and ‘endfor’ control tags are placed. To ensure this does not happen and to prevent any unintentioal whitespace, you can use jinja whitespace control like so:

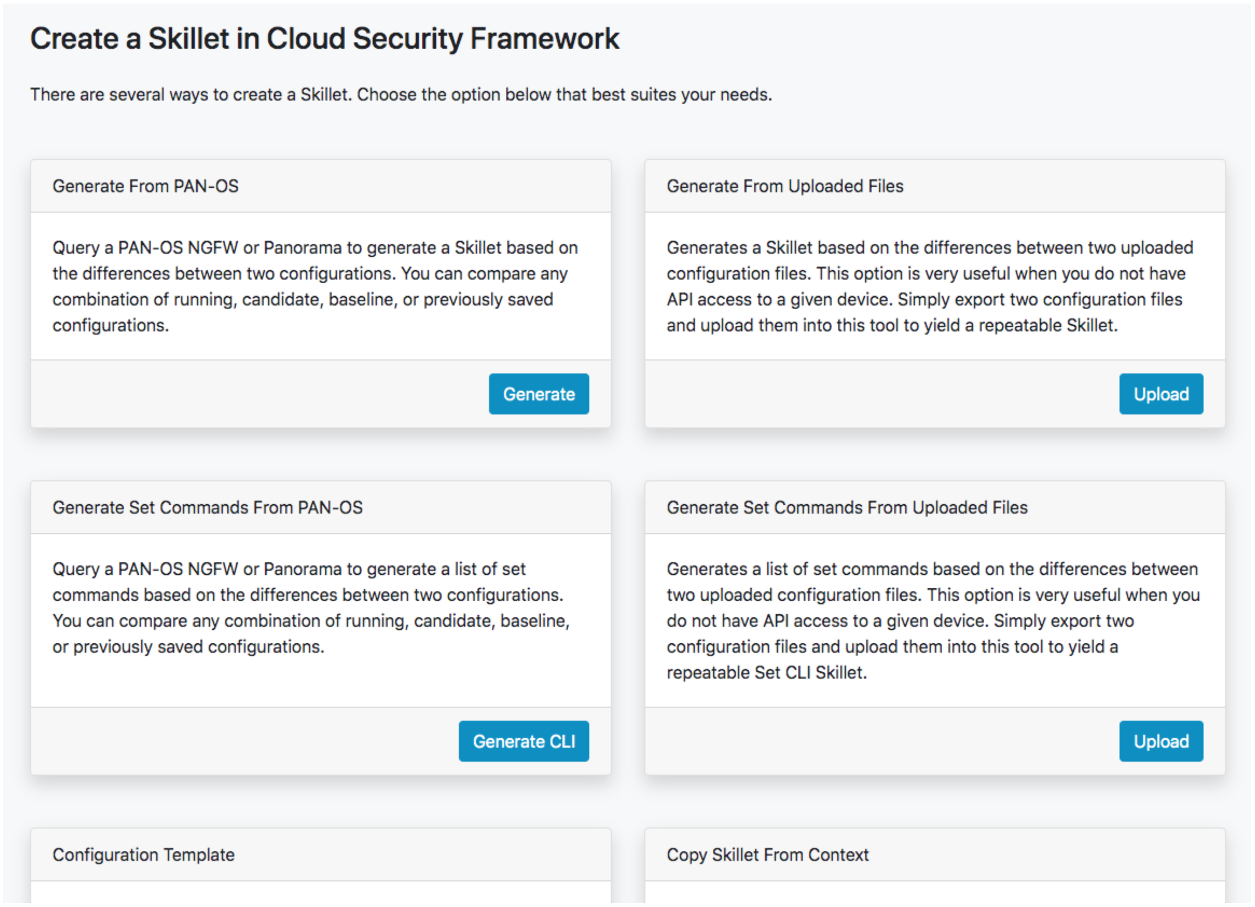
```
<dns-servers>
{%- for member in CLIENT_DNS_SUFFIX %}
    <member>{{ member }}</member>
{%- endfor %}
</dns-servers>
```

Note: Note the ‘-’ after the leading ‘{%’. This instructs jinja to remove these blank lines in the resulting parsed output template.

4.7 Creating and Editing Skillets

In Panhandler 4.0, you now have the ability to generate Skillets dynamically. This feature works by generating the difference between two saved configurations. These configurations can the candidate, running, baseline, or any saved configuration. The currently supported options for skillet generation are:

- Skillet from a running PAN-OS or Panorama instance using saved configurations or the running configuration
- Skillet from two exported configurations
- Set commands from a running PAN-OS or Panorama instance using saved configurations or the running configuration
- Set commands from two exported configurations
- Full Configuration template from a saved configuration



Create a Skillet in Cloud Security Framework

There are several ways to create a Skillet. Choose the option below that best suites your needs.

Generate From PAN-OS

Query a PAN-OS NGFW or Panorama to generate a Skillet based on the differences between two configurations. You can compare any combination of running, candidate, baseline, or previously saved configurations.

Generate From Uploaded Files

Generates a Skillet based on the differences between two uploaded configuration files. This option is very useful when you do not have API access to a given device. Simply export two configuration files and upload them into this tool to yield a repeatable Skillet.

Generate Set Commands From PAN-OS

Query a PAN-OS NGFW or Panorama to generate a list of set commands based on the differences between two configurations. You can compare any combination of running, candidate, baseline, or previously saved configurations.

Generate Set Commands From Uploaded Files

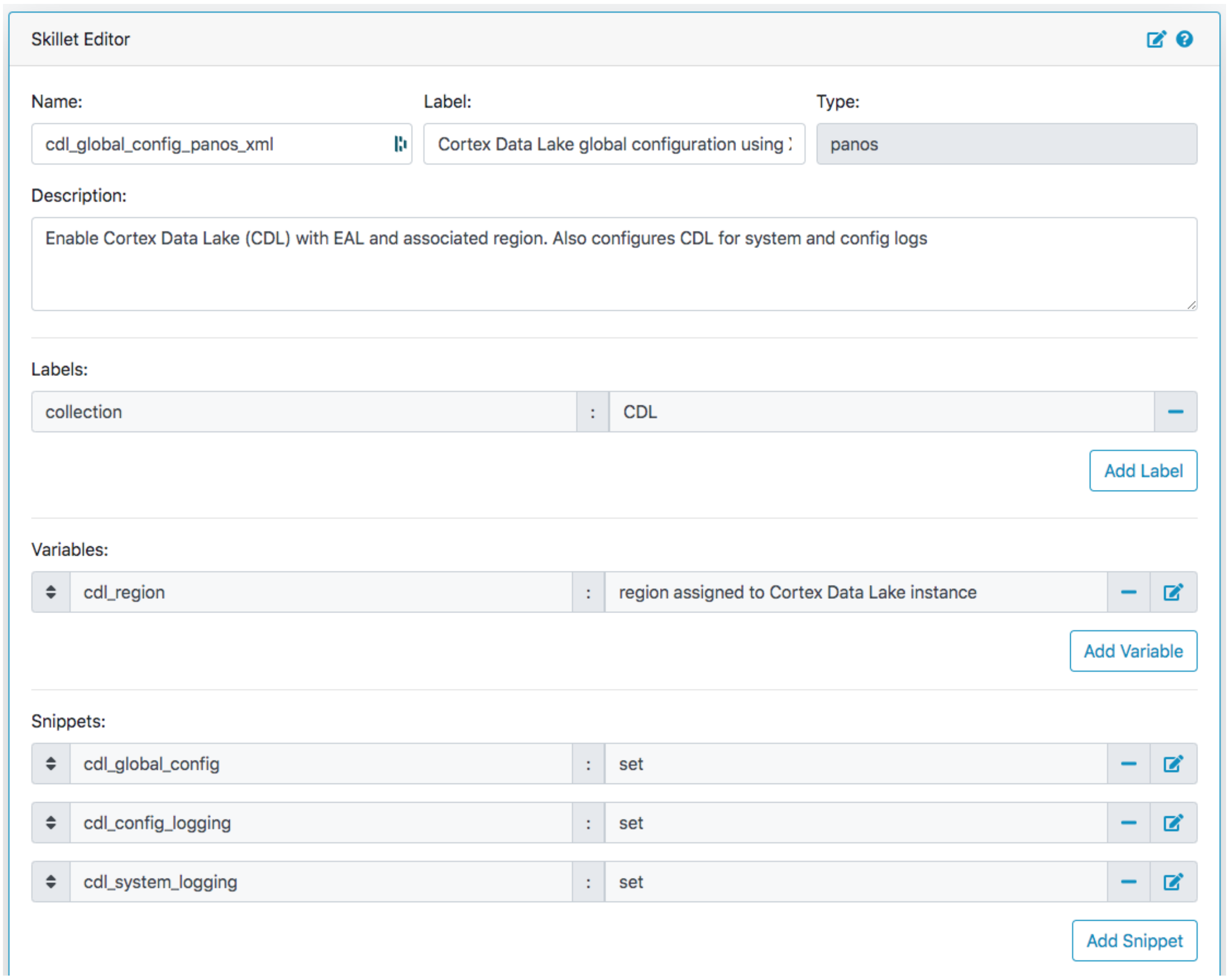
Generates a list of set commands based on the differences between two uploaded configuration files. This option is very useful when you do not have API access to a given device. Simply export two configuration files and upload them into this tool to yield a repeatable Set CLI Skillet.

Configuration Template

Copy Skillet From Context

4.7.1 Skillet Editor

The Skillet Editor allows you to copy, edit, create, and delete Skillets in a local branch of a repository. The Editor allows GUI based editing of all aspects of a Skillet including editing and ordering snippets, dynamically detecting variables, creating and ordering variables, and updating the metadata.


Skillet Editor

Name: cdl_global_config_panos_xml

Label: Cortex Data Lake global configuration using :

Type: panos

Description:

Enable Cortex Data Lake (CDL) with EAL and associated region. Also configures CDL for system and config logs

Labels:

collection	:	CDL

Variables:

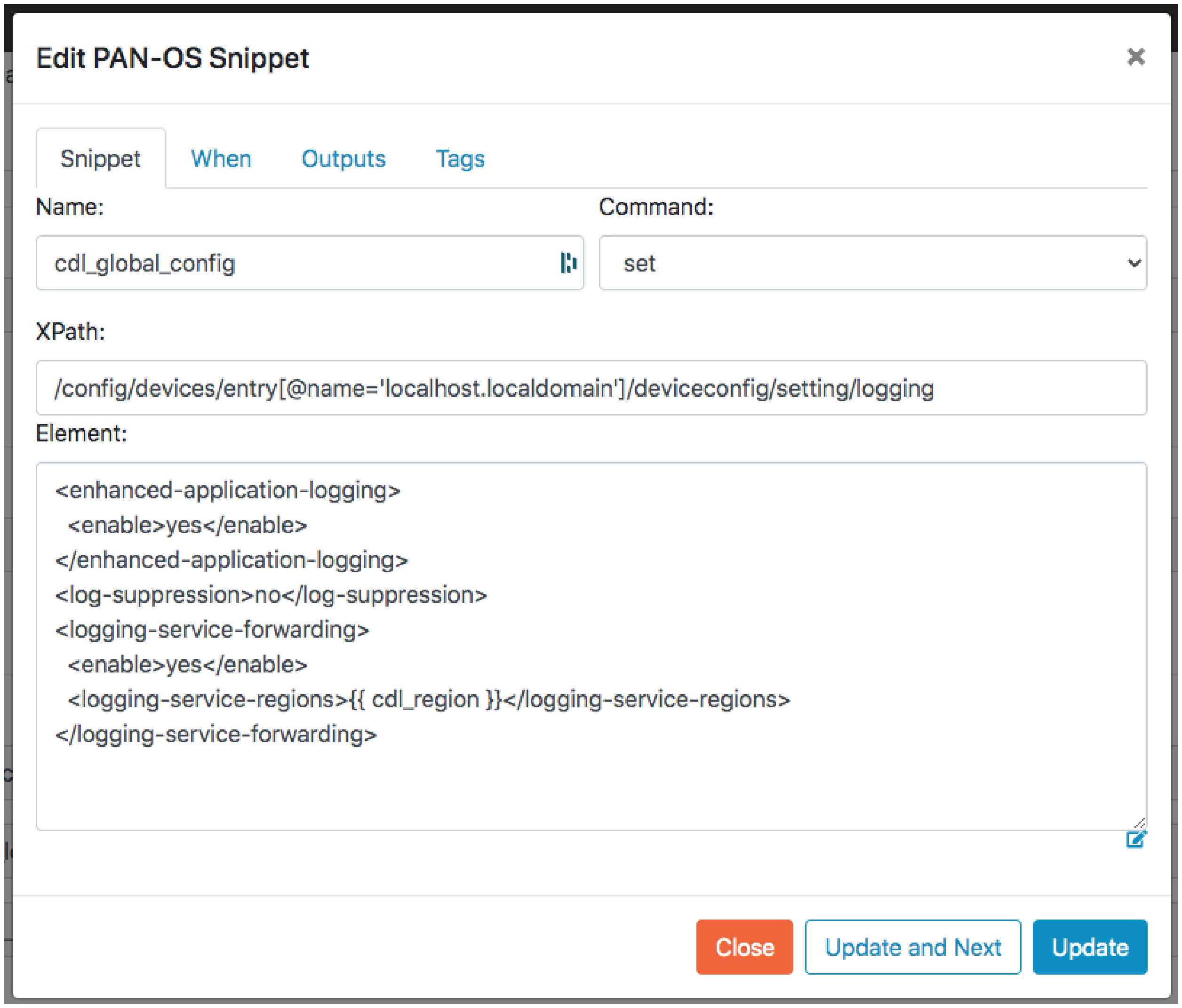
cdl_region	:	region assigned to Cortex Data Lake instance

Snippets:

cdl_global_config	:	set
cdl_config_logging	:	set
cdl_system_logging	:	set

The Skillet Editor currently supports the following skillet types:

- panos
- panorama
- pan-validation
- rest
- template



Name: cdl_global_config

Command: set

XPath: /config/devices/entry[@name='localhost.localdomain']/deviceconfig/setting/logging

Element:

```
<enhanced-application-logging>
  <enable>yes</enable>
</enhanced-application-logging>
<log-suppression>no</log-suppression>
<logging-service-forwarding>
  <enable>yes</enable>
  <logging-service-regions>{{ cdl_region }}</logging-service-regions>
</logging-service-forwarding>
```

4.7.2 Other Tools

If you prefer a CLI experience, check out SLI

For more information, see the Skillet Builder documentation.

4.8 PAN-OS Validation Skillets

PAN-OS Validation skillets are used to check the compliance of a PAN-OS device configuration. They are comprised of a series of 'tests' that each check a specific portion of the configuration. Validation tests can be executed in both 'online' as well as 'offline' mode.

Online mode will query the running configuration of a running NGFW via it's API.

Offline node will execute the tests against an uploaded configuration file. This is especially useful to checking things like configuration backups, or devices where direct API access is not possible.

4.9 Validation Tests

Each test is evaluated using jinja boolean expressions. This means each test can only result in a pass or fail. In order to perform simple logical operations on the XML configuration, it must first be converted into variables that can be passed to the jinja templating engine. Once the variables have been captured, we can test each one of them with some logical operation.

4.9.1 Variable Capturing

Panhandler will automatically inject the 'config' variable into the validation skillet context to simplify capturing additional variables from it. The 'config' variable is the 'running' configuration from the target device, or an uploaded configuration from the user. In either case, the 'config' variable will always be present for validation skillets.

The following example shows variable capturing:

```
- name: parse config variable and capture outputs
    cmd: parse
    variable: config
    outputs:
      # create a variable named 'zone_names' which will be a list of the attribute
↪'names' from each zone
      # note the use of '//' in the capture_pattern to select all zones
      # the '@name' will return only the value of the attribute 'name' from each
↪'entry'
      - name: zone_names
        capture_pattern: /config/devices/entry/vsys/entry/zone//entry/@name
      # note here we can combine an advanced xpath query with 'capture_object'. This␣
↪will capture
      # the full interface definition from the interface that contains the 'ip_to_find
↪' value
      - name: interface_with_ip
        capture_object: /config/devices/entry/network/interface/ethernet//entry/
↪layer3/ip/entry[@name="{{ ip_to_find }}"]/../..
```

This example captures two variables from the config: 'zone_names' and 'interface_with_ip'. The 'parse' cmd type informs Panhandler that this step is going to pass the variable named in the 'variable' attribute to the output. The 'outputs' attribute will then determine what specific parts of this variable we want to capture. The value of the 'outputs' attribute is a list of dicts. Each dict represents one new variable that will be captured. The two options for what you want to capture are 'capture_pattern' and 'capture_object'. Both types will query the 'config' variable using an XPATH expression. The main difference is in how the results of that query are processed and returned.

4.9.2 Capture Pattern

The 'capture_pattern' attribute will try to intelligently interpret the results of the XPATH query. This is most useful as in the above when you would like to return a list of element attributes, or a list of element text values.

In the above example, the variable 'zone_names' will be a list with the following:

```
zone_name = [
  "trust",
  "untrust",
  "dmz"
]
```

4.9.3 Capture Object

The 'capture_object' attribute will convert the returned XML into an dictionary object using the python 'xmltodict' library. This is especially useful when you want to perform a large number of tests on the same basic part of the config. This allows you to 'capture' one part of the config, then perform logic against lots of different parts of it.

In the example above, the variable 'interface_with_ip' will have the value:

```
interface_with_ip = {
  "layer3": {
    "ip": {
      "entry": {
        "@name": "10.10.10.10/24"
      }
    }
  }
}
```

4.9.4 Validation Testing

Once you have captured the various variables you want to test, use the 'validate' cmd type.

For example:

```
- name: zones_are_configured
  cmd: validate
  label: Ensure at least one zone is Configured
  test: zone_names is not none
  documentation_link: https://iron-skillet.readthedocs.io/en/docs_dev/viz_guide_panos.
↪html#device-setup-management-general-settings
```

The 'test' attribute uses the jinja expression language to perform a boolean test on the supplied expression. In this example, if zone_names is defined and has a value, then the test will pass.

4.9.5 A more complex example

This example is slightly more complex and uses a number of features to accomplish this compliance check:

```
- name: device_config_file
  cmd: parse
  variable: config
  outputs:
    # capture all the xml elements under statistics-service for later evaluation
    - name: telemetry
      capture_object: /config/devices/entry[@name='localhost.localdomain']/
↪deviceconfig/system/update-schedule/statistics-service

- name: telemetry_fully_enabled
  label: enable all telemetry attributes
  test: |
    (
    telemetry | element_value('statistics-service.application-reports') == 'yes'
    and telemetry | element_value('statistics-service.threat-prevention-reports') ==
↪'yes'
    and telemetry | element_value('statistics-service.threat-prevention-pcap') == 'yes
↪'
```

(continues on next page)

(continued from previous page)

```
    and telemetry | element_value('statistics-service.passive-dns-monitoring') == 'yes
↪'
    and telemetry | element_value('statistics-service.url-reports') == 'yes'
    and telemetry | element_value('statistics-service.health-performance-reports') ==
↪'yes'
    and telemetry | element_value('statistics-service.passive-dns-monitoring') == 'yes
↪'
    and telemetry | element_value('statistics-service.file-identification-reports')␣
↪== 'yes'
    )
  fail_message: telemetry should be enabled for all attributes
  documentation_link: https://iron-skillet.readthedocs.io/en/docs_dev/viz_guide_panos.
↪html#device-setup-telemetry-telemetry
```

Here, we first capture the XML elements found under 'statistics-service' if any are found. This is then converted into a variable object with the name 'telemetry'. The 'telemetry' object when fully configured will have the following structure:

```
telemetry = {
  "statistics-service": {
    "application-reports": "yes",
    "threat-prevention-reports": "yes",
    "threat-prevention-pcap": "yes",
    "threat-prevention-information": "yes",
    "passive-dns-monitoring": "yes",
    "url-reports": "yes",
    "health-performance-reports": "yes",
    "file-identification-reports": "yes"
  }
}
```

To facilitate a simple syntax to check this, custom jinja filters have been developed including 'element_value'. We use 'element_value' here to return the value found at a specific 'path' inside the object. The 'path' is a '.' or '/' separated list of attributes to check.

```
# this will evaluate to true in this case because the path 'statistics-service.
↪application-reports' exists
# and the value found therein is equal to the desired value of 'yes'
telemetry | element_value('statistics-service.application-reports') == 'yes'
```

For more information about all available custom filters and their example uses, see the list of filters documentation here.

4.10 PAN-OS Validation Examples

To get a sense of all that is possible, here are a couple of complete examples.

CIS Benchmarks will validate a PAN-OS device for CIS compliance.

STIG Benchmarks will validate a PAN-OS device for STIG compliance.

4.11 Hints, Tips, Tricks

4.11.1 Start with a Pass

Because you often need to know the structure of the configuration and the resulting objects, it is always a good idea to start with a fully configured PAN-OS NGFW that will 'pass' the validation test you are writing.

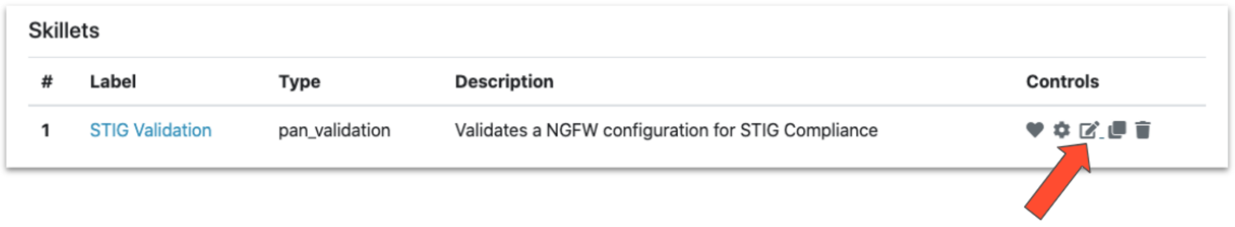
4.11.2 Use Tools to explore the config

You can also use the **'Skillet Builder'_** tools found on github here: https://github.com/PaloAltoNetworks/skilletbuilder. These are a set of Skillets designed to aid in building Skillets and especially Validation Skillets. Start with an example validation skillet from here: https://github.com/PaloAltoNetworks/skilletlib/tree/master/example_skillets and copy the contents in the 'Skillet Test Tool'. This will allow you to quickly test various capture patterns and run different types of test quickly. It will also show you the structure of the XML snippets and objects returned from your XPATH queries.

4.12 Creating and Debugging Validation Skillets

Panhandler allows you to edit and debug validation skillets using the Skillet Editor. See *Creating and Editing Skillets*.

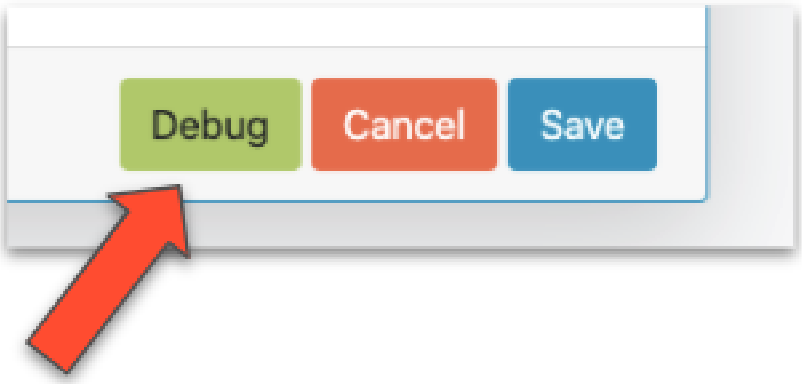
From the repository details page, click the 'edit' control for the Skillet you want to edit.



Skillets

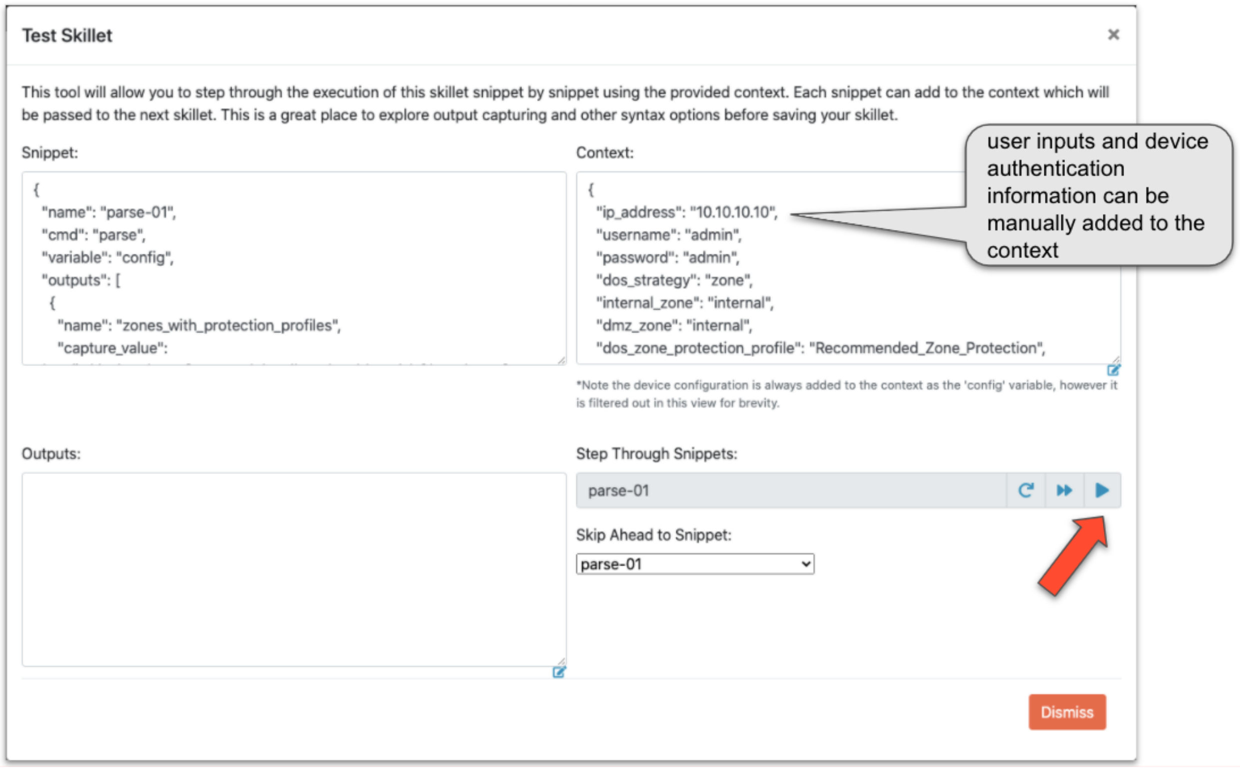
#	Label	Type	Description	Controls
1	STIG Validation	pan_validation	Validates a NGFW configuration for STIG Compliance	

At the bottom of the Skillet Editor, click the 'Debug' button to enter the Skillet Debugger.



4.13 Skillet Debugger

The Skillet Debugger allows you to step through each snippet and see the context between steps. This is especially useful to understand the various captures and filters available.



This tool will allow you to step through the execution of this skillet snippet by snippet using the provided context. Each snippet can add to the context which will be passed to the next skillet. This is a great place to explore output capturing and other syntax options before saving your skillet.

```
{
 "name": "parse-01",
 "cmd": "parse",
 "variable": "config",
 "outputs": [
  {
   "name": "zones_with_protection_profiles",
   "capture_value":
```

```
{
 "ip_address": "10.10.10.10",
 "username": "admin",
 "password": "admin",
 "dos_strategy": "zone",
 "internal_zone": "internal",
 "dmz_zone": "internal",
 "dos_zone_protection_profile": "Recommended_Zone_Protection",
```

*Note the device configuration is always added to the context as the 'config' variable, however it is filtered out in this view for brevity.

To use the debugger, manually enter Device connection information into the Context input. You may also edit any defined variables here that may impact the skillet logic.

Note: Ensure the context input is valid JSON.

Click the 'play' button to execute the next snippet. The 'Outputs' will show the returned value from the snippet. The 'Context' will also contain all captured values as well. This allows you to quickly experiment with various capture_pattern, capture_list, capture_value, and filter_items options.

You may also use the 'Skip Ahead to Snippet' in order to test a specific snippet execution.

Note: Be sure you understand what variables a snippet requires in the context when skipping ahead. In some cases, you'll need play the snippets in order to get the proper context values in place.

4.13.1 Manual Debugging with SLI

SLI is a command line interface to skilletlib and offers a great way to test and discover all the various features of skillets.

SLI makes it easy to quickly verify XPath queries, capture queries, and so on.

```
# Test and output a capture_list that displays names of all decryption policies
sli capture list  "/config/devices/entry[@name='localhost.localdomain']/vsys/entry/
↪rulebase/decryption/rules/entry/@name"

# Same as above, except this command will store the output to the default context in␣
↪the variable "decryption_rules"
sli capture -uc list "/config/devices/entry[@name='localhost.localdomain']/vsys/entry/
↪rulebase/decryption/rules/entry/@name" decryption_rules

# Capturing an object works similar to capturing a list
sli capture object "/config/devices/entry[@name='localhost.localdomain']/vsys/entry/
↪rulebase/decryption"

# Capturing an expression allows further processing on data already stored in the␣
↪context
sli capture -uc expression "decryption_rules | json_query('[].entry[].category.
↪member[]')"

# Windows requires an additional escape character on double quotes, a ` is required␣
↪in addition to the \
sli capture -uc expression "decryption_obj | json_query('decryption.rules.entry[].\`
↪"@name\`"')"
```

SLI is available on Pypi.org and can be easily installed like this:

```
pip install sli
```

4.13.2 Manual Debugging with Python

In some cases, it may be desirable to use Python or a debugger like PyCharm or pdb for building your validation skillet. Here is an example python script that will load a config file from the local filesystem and run a skillet. You may use the 'filter_snippets' option to only run specified snippets as desired.

```
import json

import click

from skilletlib.skilletLoader import SkilletLoader


@click.command()
@click.option("-c", "--config_file", help="Local Config File", type=str, default=
↪"config.xml")
@click.option("-d", "--skillet_dir", help="Skillet Directory", type=str, default=".")
@click.option("-f", "--snippet_filter", help="Snippet Filter Type", type=str, default=
↪"")
@click.option("-s", "--snippet_filter_value", help="Snippet Filter Value", type=str,␣
↪default="")
def cli(config_file, skillet_dir, snippet_filter, snippet_filter_value):
    sl = SkilletLoader()
    skillets = sl.load_all_skillets_from_dir(skillet_dir)
    d = skillets[0]

    context = dict()
    with open(config_file, 'r') as config:
```

(continues on next page)

(continued from previous page)

```
        context['config'] = config.read()

    if snippet_filter != "":
        context['__filter_snippets'] = {
            snippet_filter: snippet_filter_value
        }

    out = d.execute(context)

    print('=' * 80)
    print(json.dumps(out, indent=4))
    print('=' * 80)


if __name__ == '__main__':
    cli()
```

The above requires 'click' and 'skilletlib' to be installed. The output will contain all captured values and filtered items in the 'outputs' key.

```
pip install click
pip install git+https://github.com/PaloAltoNetworks/skilletlib.git@develop
↪#egg=skilletlib
```

For more information, see the Skillet Builder documentation.

CHAPTER 5

Example Skillet

In this example, we will create a skillet that allows the user to customize a single variable. Of course, finding the correct XML and XPath information is not at all obvious. However, there are many tools available to assist with this such as SLI and Skillet Builder.

5.1 XML Fragment

First, we'll extract the parts of the configuration that comprise this 'unit' of configuration changes (a skillet). For example, this portion of the configuration describes the log-settings we would like to modify:

```
<system>
    <match-list>
     <entry name="dhcp-log-match">
        <send-syslog>
            <member>mgmt-interface</member>
        </send-syslog>
        <filter>(eventid eq lease-start)</filter>
      </entry>
    </match-list>
</system>
<syslog>
    <entry name="mgmt-interface">
        <server>
            <entry name="mgmt-intf">
                <transport>UDP</transport>
                <port>514</port>
                <format>BSD</format>
                <server>{{ MGMT_IP }}</server>
                <facility>LOG_USER</facility>
            </entry>
        </server>
    </entry>
</syslog>
```

Notice here we have defined one variable: *MGMT_IP*. This will allow the user to insert their own management ip when deploying.

5.2 Skillet file

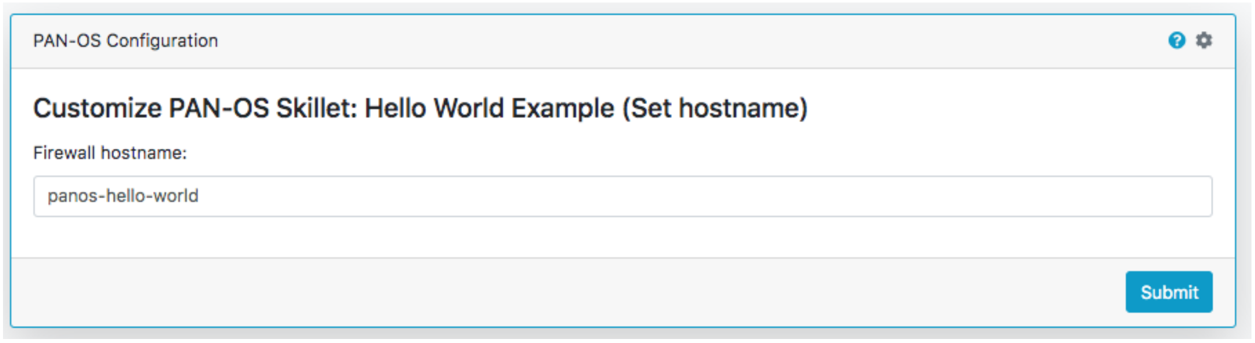
The skillet file itself is a YAML file with a suffix of *.skillet.yaml*. You may also prefix the filename, for example: *example.skillet.yaml*. See YAML Syntax for complete details.

```yaml
name: example_log_setting
label: Log Setting Example
description: Example log setting to configure syslog
type: panos
extends:

labels:
  service_type: userid

variables:
  - name: MGMT_IP
    description: NGFW management IP address
    default: 192.168.0.1
    type_hint: ip_address

snippets:
  - name: log_settings
    cmd: set
    xpath: /config/shared/log-settings
    file: log_settings.xml
```

In this file, we give some basic information about what this skillet will do, what configuration bits will be applied, and what variables the user can customize. Notice in the 'variables' section, we specify a variable entry with a 'name' that matches the variable defined in the XML fragment. The 'snippets' section will inform Panhandler where in the configuration this fragment should be inserted (xpath) and where to find the fragment (file).

5.3 Rendered Form

This *example.skillet.yaml* will produce the following web form in Panhandler:


PAN-OS Configuration

Customize PAN-OS Skillet: Hello World Example (Set hostname)

Firewall hostname:

panos-hello-world

CHAPTER 6

More Example Skillets

6.1 Example Skillets by Type

example_panos

example_rest

example_rest_with_output

example_python

example_terraform

example_validation

6.2 Example Skillets by Feature

example_complex_validation

example_when

example_panos_get

6.3 External Skillet Repositories

Here is a couple of Git repositories that contain numerous example Skillets.

Palo Alto Networks Skillets.

SkilletLib is a library for parsing and executing Skillets in third party applications and tooling. The SkilletLib repository also has many useful examples.

Palo Alto Networks World Wide CE team has a great collection of Skillets on Github.

Many other Skillets may be found on Github as well using the Skillets topic.

CHAPTER 7

When things go wrong

Sometimes you may hit bugs or other unexpected behaviours. This page should give you some information about how to recover your environment in the event something goes sideways.

7.1 Ensuring you have the latest

New releases almost always feature mostly bugfixes. As such, if you encounter a problem, you should first update Panhandler to the latest version. The recommended way to do that is to run the installer script:

```
curl -s -k -L http://bit.ly/2xui5gM | bash
```

7.2 Restarting the docker container

Some types of problems can be fixed by restarting the container

```
# get the docker ID of the panhandler container
docker ps | grep panhandler

# use the 12 digit ID in the restart command
docker restart 3fd4e2c78557

# or as a one-linter
docker ps | grep panhandler | awk '{ print $1 }' | xargs -n 1 docker restart
```

7.3 Clearing the cache

If you are seeing inconsistent data in the UI after a failed git import or some other error condition, this can indicate the cache is out of date. Since the cache survives a docker restart, you may need to manually perform a clear. To clear the cache navigate to the following URL: http://127.0.0.1:8080/clear_cache

Note: You may need to change the port number above to match your environment

7.4 Cancelling a Task

Some skillets use a background task to perform it's action. If this task appears to be looping or stuck, you can cancel the task by navigating to the following URL: http://127.0.0.1:8080/cancel_task

7.5 Removing a Repository

If for some reason, panhandler cannot load a repository, or crashes on the repository details page, you may need to manually remove the repository. The recommended way to start the panhandler container is to create a volume mount from your $HOME directory. This ensure all persistent data will be stored in $HOME/.pan_cnc/panhandler. To manually remove a repository, open a shell and navidate to $HOME/.pan_cnc/panhandler/repositories and use the *rm -rf* command to remove it completely. You will then need to clear the cache as noted above.

7.6 Troubleshooting Docker Skillets

Docker Skillets require communication with the docker daemon on your host machine via a special bind mount. First, ensure you have the proper bind mount configured on your Panhandler container:

```
docker inspect panhandler -f "{{ .HostConfig.Binds }}"
```

Ensure you see the */var/run/docker.sock:/var/run/docker.sock* in the returned list. If you do not have the docker.sock listed in the output, ensure you the docker run command you are using includes the parameter: *-v /var/run/docker.sock:/var/run/docker.sock*.

If you find that the volume is properly mounted, but you still cannot execute docker Skillets, you may need to adjust the permissions and group mappings inside the container. Panhandler includes tool to simplify this for you:

```
docker exec -u root -it panhandler /app/cnc/tools/create_docker_group.sh
```

7.7 The hammer approach

If none of the above things work, you may need to remove everything and start over. The installer script can be used to do this and should be your first option:

```
curl -s -k -L http://bit.ly/2xui5gM | bash
```

Or, you may perform these steps manually. First, stop the container

```
# as a one-linter
docker ps | grep panhandler | awk '{ print $1 }' | xargs -n 1 docker stop
```

Next, remove all persistent data

```
# be careful with this one!
rm -rf $HOME/.pan_cnc/panhandler
```

Update to the latest docker image and create a new container

```
docker pull paloaltonetworks/panhandler:latest
docker run -t -v $HOME:/home/cnc_user -p 8080:8080 paloaltonetworks/panhandler:latest
```

7.8 File a bug

If you need to perform any of the above steps, then this is bug. Please file a bug report with as much detail as possible here: https://github.com/paloaltonetworks/panhandler/issues

CHAPTER 8

About

Panhandler is a tool to manage and share PAN-OS configuration sets called *Skillets*. A configuration set can be a full device configuration, or a set of configuration elements. Panhandler allows you to import git repositories that contain one or more of these configuration templates. Each template contains a set of configuration elements and variables that can be customized for each deployment. Variables are presented in an auto-generated web form for an operator to complete. Once complete, the template is rendered and pushed to a PAN-OS device.

Skillets allow an architect or builder to create a fully customizable configuration set with the correct level of abstraction for their organization's needs. For example, the PAN-OS GUI may offer 20 different options for a given feature, however, in your organization you may want to standardize on 18 of those and allow customization of only 2 options. Skillets allow you to do just that.

For more information about Skillets, see the Live community page.

For more information about building Skillets, see the Skillet Builder Documentation.

CHAPTER 9

Disclaimer

This software is provided without support, warranty, or guarantee. Use at your own risk.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search