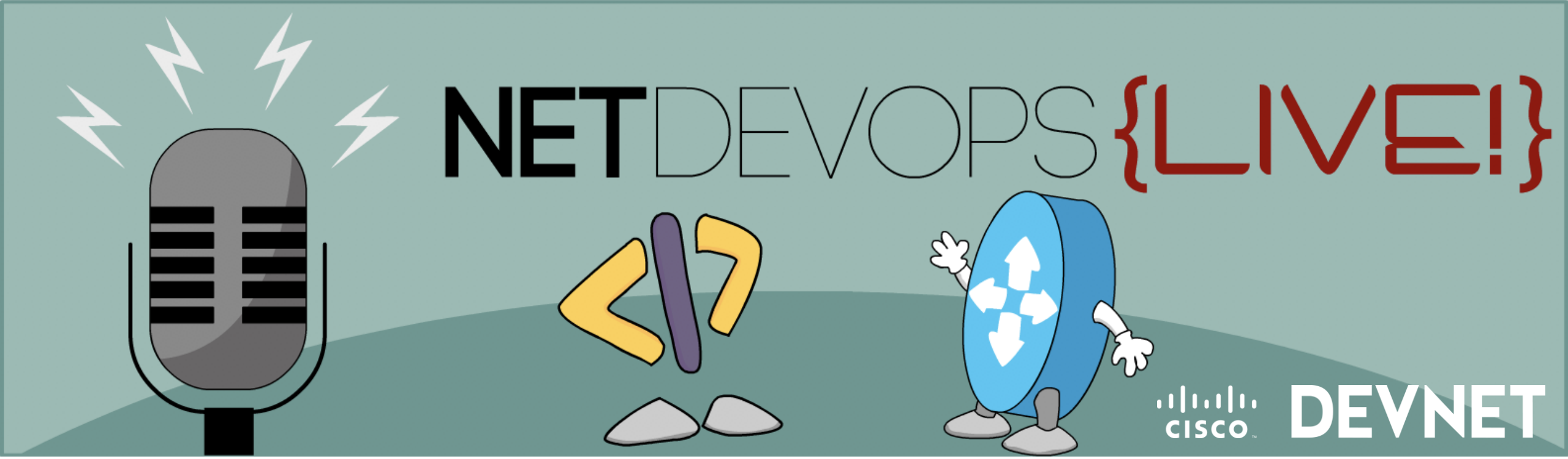


Automate Safely, NetDevOps Security Strategies

Hank Preston, ccie 38336 R/S
NetDevOps Engineer, DevNet Sandbox
Twitter: @hfpreston

Season 2, Talk 10

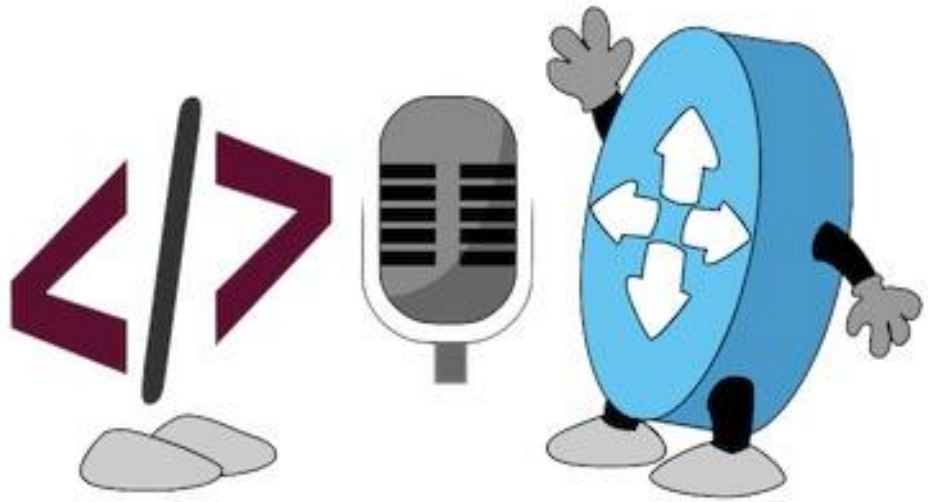
https://developer.cisco.com/netdevops/live

Before we start...

- ✓ You are never “secure”
- ✓ Security is a continuum
- ✓ You **must** trust something/someone
- ✓ Following are some “easy” suggestions you can implement today.

What are we going to talk about?

- Don't let just anyone in… API Access Control
  - Excuse me… who are you? Authentication
  - Look but no touch… Role Based Access
  - Don't even think about trying… ACLs and APIs
- Spilling the beans on secrets
  - Caring about the environment… variables
  - Securing your valuables in a vault
- Just one more thing…
  - Version control safely with .gitignore



Security Continuum

Don’t let just anyone in…
API Access Control

APIs and Authentication

- API access no different from CLI access
- Know who's accessing
- Ensure unique credentials, no community admins

```
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:35438 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:36516 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:37590 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:38664 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:39738 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:40816 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:41890 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:42964 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:44038 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:45116 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:46190 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:47264 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:48338 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:49416 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:50490 and was authorized for netconf over ssh.
```

These logs aren't terribly helpful are they... 🤔

APIs and Authentication

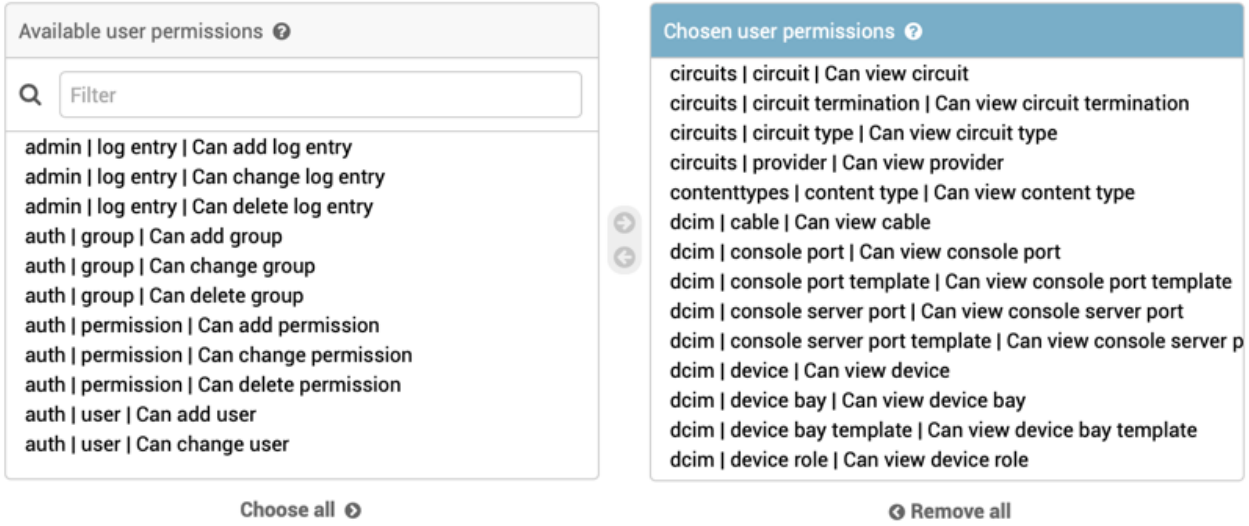
- API access no different from CLI access
- Know who's accessing
- Ensure unique credentials, no community admins

```
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'jpatterson' authenticated successfully from 10.24.22.241:65216 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'developer' authenticated successfully from 64.103.26.44:52682 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'jbutcher' authenticated successfully from 10.24.22.241:49698 and was authorized for netconf over ssh. E
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'gbullien' authenticated successfully from 10.24.22.241:49777 and was authorized for netconf over ssh. E
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'tclancy' authenticated successfully from 10.24.22.241:49831 and was authorized for netconf over ssh. Ex
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'jrtolkien' authenticated successfully from 10.24.22.241:49891 and was authorized for netconf over ssh.
%DMI-5-AUTH_PASSED: R0/0: dmiauthd: User 'gmartin' authenticated successfully from 10.24.22.241:49940 and was authorized for netconf over ssh. Ex
```

Much better, now we know who to talk to 😁

Principal of Least Privilege

- Many times automation about “read only” actions
- Use accounts authorized for only privileges needed
- Different APIs and Applications will handle authorization and privilege differently



https://en.wikipedia.org/wiki/Principle_of_least_privilege

Demo: NETCONF Read Only Access

- IOS XE default NETCONF behavior
  - Only Priv15 allowed access
- NETCONF Access Control Model (NACM) manages all access
- Let's allow PRIV01 users read-only access

```
<config>
 <nacm xmlns="urn:ietf:params:xml:ns:yang:ietf-netconf-acm">
  <rule-list>
   <name>only-get</name>
   <group>PRIV01</group>
   <rule>
    <name>deny-edit-config</name>
    <module-name>ietf-netconf</module-name>
    <rpc-name>edit-config</rpc-name>
    <access-operations>exec</access-operations>
    <action>deny</action>
   </rule>
   <rule>
    <name>allow-get</name>
    <module-name>ietf-netconf</module-name>
    <rpc-name>get</rpc-name>
    <access-operations>exec</access-operations>
    <action>permit</action>
   </rule>
   <rule>
    <name>allow-models</name>
    <module-name>*</module-name>
    <access-operations>read</access-operations>
    <action>permit</action>
   </rule>
  </rule-list>
 </nacm>
</config>
```

IOS XE NACM Config Guide

Code Example on GitHub

Protect that Management Plane

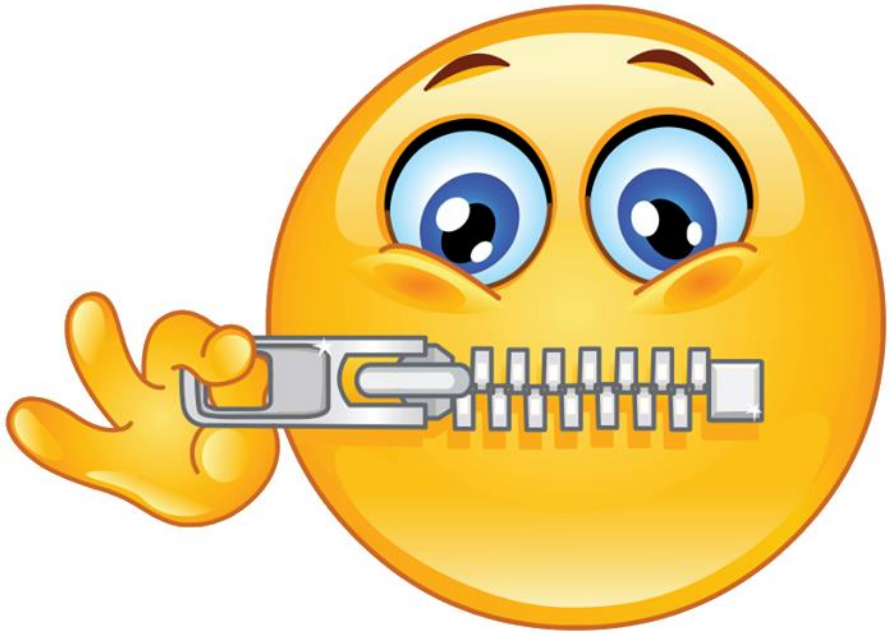
- Do the simple things
  - ACLs on VTY Lines
  - ACLS for HTTP Server
  - Disable “insecure” telnet and http
- Strive for the harder things
  - Key based authentication
  - Trusted Certificates and Keys

```
ip access-list standard MGMT-ACCESS
 permit 10.10.20.20
 permit 10.10.20.21
 deny   any log

no ip http server
ip http access-class ipv4 MGMT-ACCESS
ip http authentication local
ip http secure-server

line vty 0 4
 access-class MGMT-ACCESS in
 transport input ssh
```

Spilling the beans on secrets

Q: What is a secret?
A: Anything you don't want someone else to know!

- Credentials (Users/Passwords)
- IP Addresses
- Tokens
- DNS Names
- Really anything



So this is really not good...

```
# DevNet Always-On IOS XE Sandbox Device
ios_xe1 = {
        "address": "ios-xe-mgmt.cisco.com",
        "netconf_port": 10000,
        "restconf_port": 9443,
        "ssh_port": 8181,
        "username": "root",
        "password": "D_Vay!_10&"
    }
```

```
# DevNet Always-On Sandbox ACI APIC
apic = {
    "host": "https://sandboxapicdc.cisco.com",
    "username": "admin",
    "password": "ciscopsdt",
    "port": 443
    }
```

We (DevNet Sandbox) actually want all this info to be public.

But you probably don't for your environment.

So what do you do?

```
# DevNet Always-On IOS XE Sandbox Device
ios_xe1 = {
        "address": "ios-xe-mgmt.cisco.com",
        "netconf_port": 10000,
        "restconf_port": 9443,
        "ssh_port": 8181,
        "username": "root",
        "password": "D_Vay!_10&"
      }
```

```
# DevNet Always-On Sandbox ACI APIC
apic = {
    "host": "https://sandboxapicdc.cisco.com",
    "username": "admin",
    "password": "ciscopsdt",
    "port": 443
    }
```

The 12 Factor App
III. Config

https://12factor.net/config

III. Config

Store config in the environment

An app's *config* is everything that is likely to vary between deploys (staging, production, developer environments, etc). This includes:

- Resource handles to the database, Memcached, and other backing services
- Credentials to external services such as Amazon S3 or Twitter
- Per-deploy values such as the canonical hostname for the deploy

Apps sometimes store config as constants in the code. This is a violation of twelve-factor, which requires **strict separation of config from code**. Config varies substantially across deploys, code does not.

A litmus test for whether an app has all config correctly factored out of the code is whether the codebase could be made open source at any moment, without compromising any credentials.

Note that this definition of "config" does **not** include internal application config, such as `config/routes.rb` in Rails, or how code modules are connected in Spring. This type of config does not vary between deploys, and so is best done in the code.

Another approach to config is the use of config files which are not checked into revision control, such as `config/database.yml` in Rails. This is a huge improvement over using constants which are checked into the code repo, but still has weaknesses: it's easy to mistakenly check in a config file to the repo; there is a tendency for config files to be scattered about in different places and different formats, making it hard to see and manage all the config in one place. Further, these formats tend to be language- or framework-specific.

The twelve-factor app stores config in *environment variables* (often shortened to *env vars* or *env*). Env vars are easy to

*An **environment variable** is a dynamic-named value that can affect the way running processes will behave on a computer.*

https://en.wikipedia.org/wiki/Environment_variable

Exploring Environment Variables

- Part of EVERY operating system
- And you've probably used them without knowing it
  - $HOME
  - $PATH
  - $PS1
- Create your own
  - Linux/macOS – **export VAR=VALUE**
  - Windows – **set VAR=VALUE**

For example only… terrible way to connect to a device without a password 😬

```
(std) ~\ $ echo $HOME
/Users/hapresto

(std) ~\ $ echo $PATH
/Users/hapresto/ncs47/bin:/Users/hapresto/virtualen
vs/std/bin...

(std) ~\ $ echo $PS1
(std) \W\ $

(std) ~\ $ export NETDEVOPS_LIVE="Is Awesome"
(std) ~\ $ echo $NETDEVOPS_LIVE
Is Awesome

(std) ~\ $ export USERNAME=cisco
(std) ~\ $ export PASSWORD=cisco

(std) ~\ $ sshpass -f <(printf '%s\n' $PASSWORD)\
           ssh $USERNAME@172.16.30.53

core1#
```

Demo: Environment Variables and Python

- It's very easy to access Environment Variables from Python
- Use the **os** standard library
- Leverage **os.getenv()** to safely retrieve values
  - Check for **None** to ensure data exists

```
# From Bash
export SBX_ADDRESS=10.10.20.48
export SBX_USER=developer
export SBX_PASS=C1sco12345

ipython # Starting Python

import os

os.environ

os.environ["SBX_ADDRESS"]
os.environ["SBX_USER"]
os.environ["SBX_PASS"]

os.environ["SBX_MISSING"]

address = os.getenv("SBX_ADDRESS")
missing = os.getenv("SBX_MISSING")

missing is None
```

Code Example on GitHub

Demo: “sourcing” files for easy ENV VARS

- Manually “exporting” in bash error prone and time consuming
- Use source env_file to quickly read them into active terminal
- Assumes you “trust” local computer
  - Data stored clear text
  - Don’t commit these files to source control (more on this later)

```
$ cat src_env
# Setting ENV VARS for DevNet Sandbox
export SBX_ADDRESS=10.10.20.48
export SBX_NETCONF_PORT=10000
export SBX_SSH_PORT=8181
export SBX_RESTCONF_PORT=9443
export SBX_USER=developer
export SBX_PASS=C1sco12345

$ source src_env

$ echo $SBX_SSH_PORT
8181

$ python env_var_demo1.py
Here are the interfaces from device ios-
xe-mgmt-latest.cisco.com
GigabitEthernet1
GigabitEthernet2
GigabitEthernet3
```

src_env.template on GitHub env_var_demo1.py on GitHub

Environment Variables Used in Other Tools Too

Ansible Lookup Filter for Variables

```
# host_vars/device.yml
ansible_connection: network_cli
ansible_network_os: ios
ansible_host: "{{ lookup('env', 'SBX_ADDRESS') }}"
ansible_user: "{{ lookup('env', 'SBX_USER') }}"
ansible_password: "{{ lookup('env', 'SBX_PASS') }}"
ansible_become: yes
ansible_become_method: enable
ansible_become_password: |
  "{{ lookup('env', 'SBX_PASS') }}"
```

Ansible Docs | host_vars/device.yml on GitHub

pyATS/Genie Testbed ENV Syntax

```
# Genie Testbed File
testbed:
 name: netdevops-security
 tacacs:
   username: "%ENV{SBX_USER}"
 passwords:
   tacacs: "%ENV{SBX_PASS}"
   enable: "%ENV{SBX_PASS}"

devices:
 csr1000v-1:
  os: iosxe
  type: iosxe
  connections:
   defaults:
    class: unicon.Unicon
   ssh:
    protocol: ssh
    ip: "%ENV{SBX_ADDRESS}"
    port: "%ENV{SBX_SSH_PORT}"
```

pyATS/Genie Docs | genie-testbed.yaml on GitHub

How to share across teams

- VERY insecure to commit a **src_env** file with secrets to source control
- But you need this info wherever you execute code…
- So commit **src_env.template**
- Document in README how to setup for project

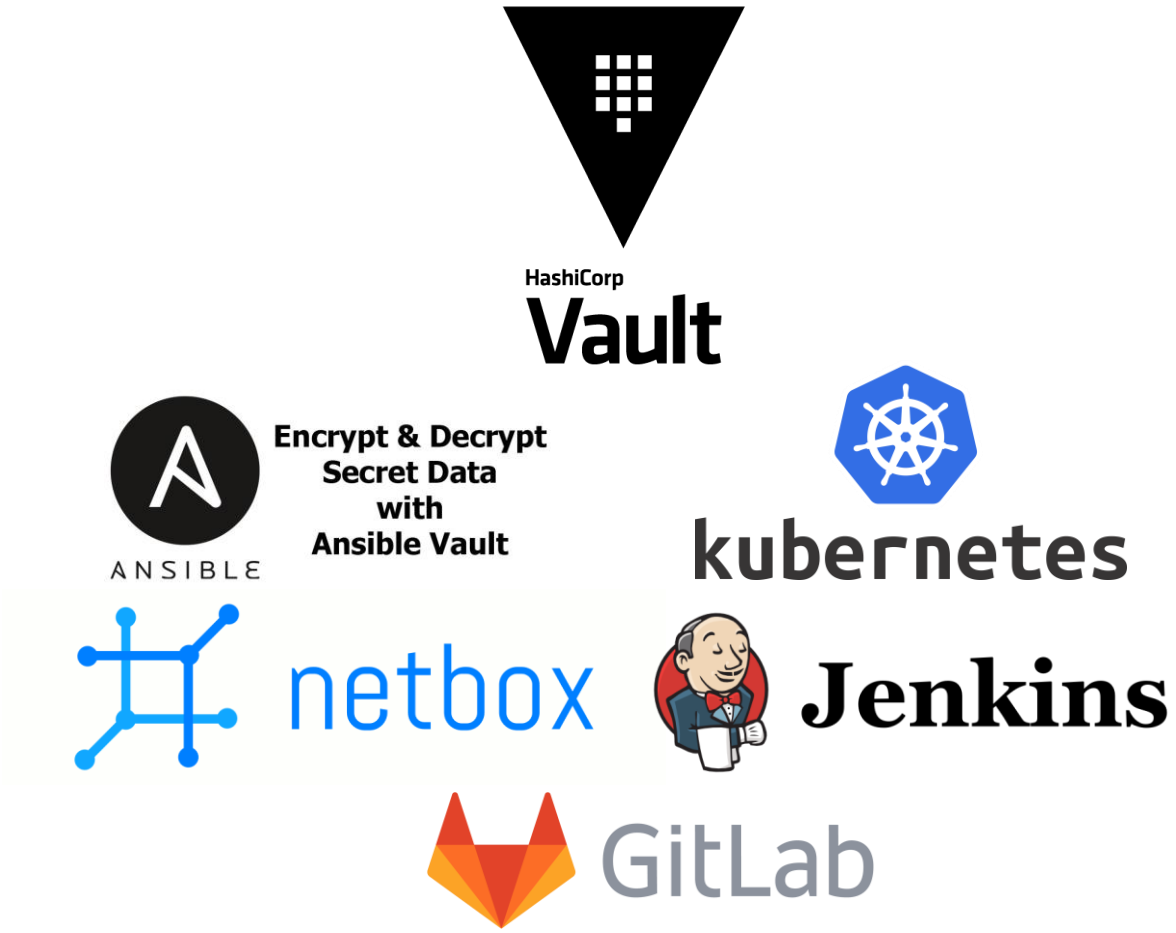
```
# Copy file to src_env
# Complete with "secrets"
# 'source src_env' before running

export SBX_ADDRESS=
export SBX_NETCONF_PORT=
export SBX_SSH_PORT=
export SBX_RESTCONF_PORT=
export SBX_USER=
export SBX_PASS=
```

src_env.template on GitHub

Environment Variables not secure enough… how about a “secret vault”?

What is a secret store/vault?

- An additional technology layer/application
- Store encrypted versions of secrets
- Requires some authentication to retrieve unencrypted values
- Can be a standalone application or a feature of another system
- Variety of potential options available



Demo: Ansible Vault Secured Variables

- **ansible-vault** included with Ansible
- Creates encrypted version of files or strings
  - Safe to store in source control
- Encrypted/Decrypted with password or file
  - Introduces need to distribute/share the vault "key"

```
# device.insecure.yml
ansible_host: 10.10.20.48
ansible_port: 8181
ansible_user: developer
ansible_password: C1sco12345
```

```
$ ansible-vault encrypt --output=device.yml\
   device.insecure.yml
New Vault password:
Confirm New Vault password:
Encryption successful
```

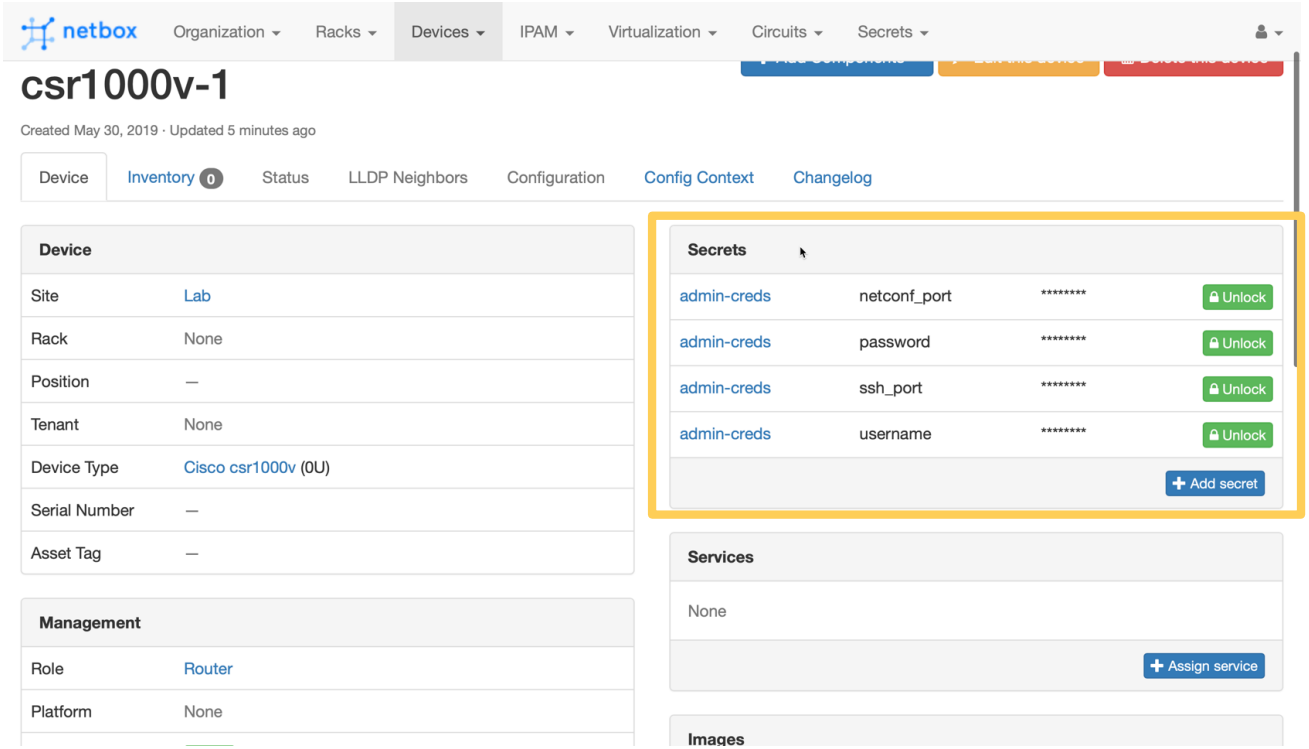
```
# device.yml
$ANSIBLE_VAULT;1.1;AES256
346264323130353937353764363232643263623532613332663465363066366162323966323164
373437333835386137633537326430653265303139353036 0a643134373339373033643634393
353261643038633339323365376365366233643330363935636461343336646438313437656130
6233306533...
```

```
$ ansible-playbook --ask-vault-pass playbook.yml
Vault password:
```

device.insecure.yml on GitHub

Demo: Storing Secrets in NetBox

- NetBox is a ”Source of Truth” for network many automation teams
- Supports securely storing secrets along with device info
- Robust abilities to limit access to secrets by user and role
- Encrypted with private/public keys



Demo: Retrieving Secrets from NetBox for Scripts

- NetBox provides API and language libraries (Python, Go, etc)
- Use these to retrieve secret details at execution time
- Must provide API Token and SSH Private Key to access

```
from getpass import getpass
import pynetbox

device_name = input("What is your device name?\n")
nb_url = "http://localhost:8081"
nb_token = getpass("What is your NetBox API Token?\n")
private_key_file = input("Where is your Private Key for Secret Access?\n")

nb = pynetbox.api(nb_url, token=nb_token,
            private_key_file=private_key_file)
nb_device = nb.dcim.devices.get(name=device_name)

address = nb_device.primary_ip.address

username = nb.secrets.secrets.get(device_id=nb_device.id,
                        name="username")
password = nb.secrets.secrets.get(device_id=nb_device.id,
                        name="password")
netconf_port = nb.secrets.secrets.get(device_id=nb_device.id,
                          name="netconf_port")
```

vault_netbox_demo.py on GitHub

Just one more thing...

Protect yourself with .gitignore

- **.gitignore** files tell git what to ignore (duh 🤦‍♂️)
  - Place file in any directory, or at root of repository
  - Commit your ignore files!
- There are standard ”ignores”
  - **venv/**
  - ***.pyc**
- But you can add your own as well
  - Clear text secret files
  - Certificates and tokens

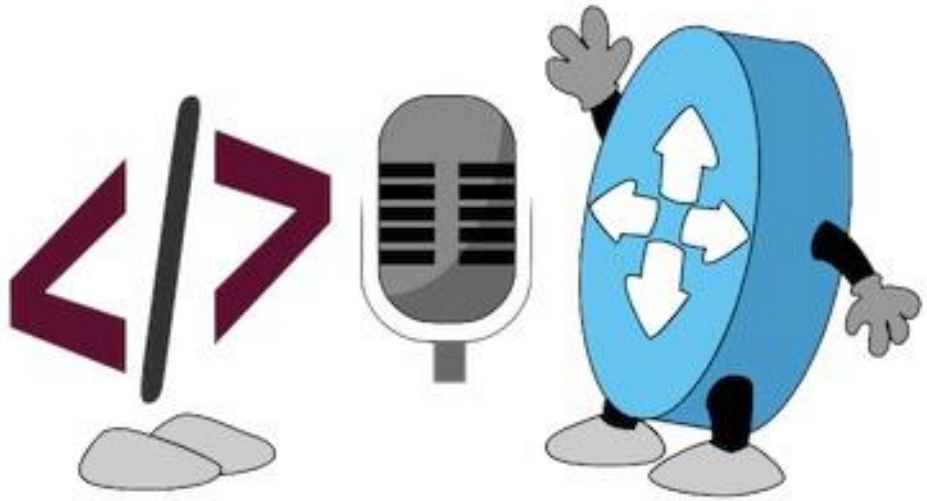
```
# .gitignore Samples
*insecure*
src_env
secrets*
credentials
id_rsa
*key*
```

.gitignore on GitHub

Summing up

What did we talk about?

- Don’t let just anyone in… API Access Control
  - Excuse me… who are you? Authentication
  - Look but no touch… Role Based Access
  - Don’t even think about trying… ACLs and APIs
- Spilling the beans on secrets
  - Caring about the environment… variables
  - Securing your valuables in a vault
- Just one more thing…
  - Version control safely with .gitignore



Webinar Resource List

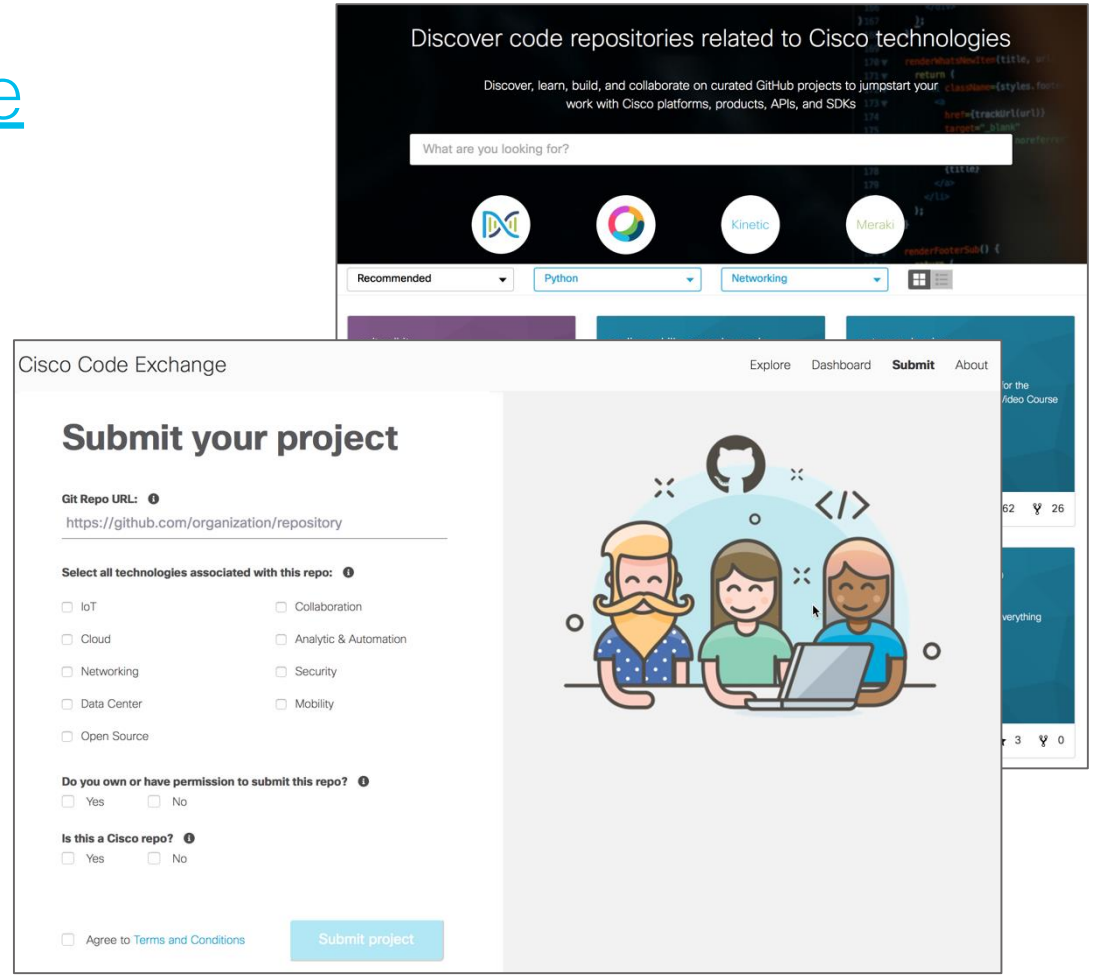
- Docs and Links
  - NETCONF Access Control Model on IOS XE Docs
  - pyATS/Genie ENV Documentation
  - Ansible Lookup ENV Documentation
  - Ansible Vault Documentation
  - NetBox Secrets Documentation
- Learning Labs
  - Guided Walkthrough of Demos http://bit.ly/ndl-security-lab
- DevNet Sandboxes
  - IOS XE on CSR Latest Code Always On Sandbox http://cs.co/sbx-iosxe-latest
- Code Samples
  - https://github.com/hpreston/netdevops_demos -> netdevops-security folder

NetDevOps Live! Code Exchange Challenge

developer.cisco.com/codeexchange

Submit a network automation project to Code Exchange that uses either Environment Variables or a "Vault" model for storing secrets.

Example: Create an src_env.template file that users can use to create a src_env for a lab device. Then read in that data using the os library in your script.



Looking for more about NetDevOps?

- NetDevOps on DevNet
  developer.cisco.com/netdevops
- NetDevOps Live!
  developer.cisco.com/netdevops/live
- NetDevOps Blogs
  blogs.cisco.com/tag/netdevops
- Network Programmability Basics Video Course
  developer.cisco.com/video/net-prog-basics/

Got more questions? Stay in touch!

Hank Preston

hapresto@cisco.com
@hfpreston
http://github.com/hpreston

CISCO DEVNET

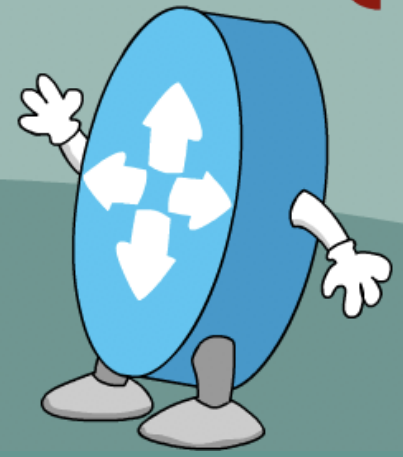
LEARN CODE INSPIRE CONNECT

developer.cisco.com

@CiscoDevNet
facebook.com/ciscodevnet/
http://github.com/CiscoDevNet

NETDEVOPS {LIVE!}

cisco DEVNET



https://developer.cisco.com/netdevops/live

@netdevopslive