# Ansible-LockdownRHEL9-CISDocumentation:

**MindPoint Group**

**Nov 20, 2023**

# INTRODUCTION

CHAPTER

ONE

# AUTOMATED SECURITY BENCHMARK - AUDITING AND REMEDIATION

## 1.1 MindPoint Group's (MPG) Ansible-Lockdown Overview

Our ReadtheDocs elaborates the resources, significance and objective of using our Automated Security Benchmark for auditing and remediation of system security. Our MPG Ansible roles can be applied to systems to improve security posture, meet compliance requirements, and deploy without disruption after due diligence. Security hardening is achieved through the use of industry-recognized benchmarks CIS and DISA STIG, which provide open-source licensed configurations to bring systems into security compliance. The content delivered consists of an audit component based on GOSS that scans a host for compliance and a remediate component that can be run centrally using an **Ansible Deployment Server** to bring host(s) into compliance. Our open-source development/release process composes of MPG's Ansible-Lockdown GitHub main/devel branches and ansible-galaxy updates that aligned with new benchmark versions.

## 1.2 Why should this role be applied to a system?

There are **three** main reasons to apply this role to systems:

- **1. Improve security posture**

  The configurations from the adopted benchmark add security and rigor around multiple components of an operating system, including user authentication, service configurations, and package management. All of these configurations add up to an environment that is more difficult for an attacker to penetrate and use for lateral movement.

- **2. Meet compliance requirements**

  Some deployers may be subject to industry compliance programs, such as PCI-DSS, HIPAA, ISO 27001/27002, or NIST 800-53. Many of these programs require hardening standards to be applied to systems.

- **3. Deployment without disruption**

  Security is often at odds with usability. The role provides the greatest security benefit without disrupting production systems. Deployers have the option to opt out or opt in for most configurations depending on how their environments are configured.

## 1.3 What is security hardening?

Based upon industry recognized benchmarks and best practices, using leading products to enable highly adjustable configurations to bring your systems/platforms into security compliance.

- Open-Source (MIT licensed)
  - Community supported as standard
  - Enterprise support available
- Configuration-as-code
  - Assist in bringing your systems/platform into compliance through the use of Ansible
  - Audit your current system/platform using GOSS
- Highly configurable to work with your systems

The content delivered is based upon either one of the two major contributors to the security best practices in the IT industry.

- **Center for Internet Security** (CIS):
  - A global IT community of experts helping to build, document sets of benchmarks to produce industry best security practices.
  - CIS Benchmarks are vendor agnostic, consensus-based security configuration guides both developed and accepted by government, business, industry, and academia.

**or**

- **Security Technical Implementation Guide** (STIG):
  - From the Defense Information Systems Agency (DISA)
  - The STIG is released with a public domain license and it is commonly used to secure systems at public and private organizations around the world.

---

**Note:** Both CIS and STIG are well-known and respected benchmarks created for the industry to assist in achieving recognized compliance (e.g. PCI DSS, HIPAA, SOC2, NIST) and adopting security best practices.

---

### 1.3.1 CIS Overview

#### What is CIS?

**Center for Internet Security**

**CIS** is home to the Multi-State Information Sharing and Analysis Center® (MS-ISAC®), the trusted resource for cyber threat prevention, protection, response, and recovery for U.S. State, Local, Tribal, and Territorial government entities, and the Elections Infrastructure Information Sharing and Analysis Center® (EI-ISAC®), which supports the rapidly changing cybersecurity needs of U.S. elections offices.

#### What do the CIS roles do?

The roles follow the CIS provided guide (benchmark) released for the OS/platform/application. Each guide is different, some have in excess of 200 controls and apply to various parts of an OS/platform/application. Each guide is updated regularly by CIS.

---

**Note:** CIS is often used if there is absence for an appropriate released STIG version.

---

#### Control Severities

Controls are divided into groups based on the following properties:

- **Level 1** The majority of control are based at this level. These controls have are considered to have a low impact to a system. By implementing these controls is considered low to medium risk of disruption.
- **Level 2** These controls are considered high risk with a chance of system disruption if implemented.

---

**Note:** Along with severities it also shows the severity for servers vs workstations. You can have a control that is more severe for servers than workstations or vice-versa. We tag each task with the full level, level1-server/level1-workstation

---

---

**Note:** All of the default configurations are found within is_container.yml and this is where they should be adjusted. Do not adjust within the tasks themselves - remediation - `defaults/main.yml` - audit

- standalone `vars/CIS.yml`
- combined `vars/[system_hostname].yml`

---

### 1.3.2 STIG Overview

#### What is STIG?

Sometimes referred to as DISA STIG. DISA STIG refers to an organization (DISA — Defense Information Systems Agency) that provides technical guides (STIG — Security Technical Implementation Guide).

#### What do the STIG roles do?

This role follows the Security Technical Implementation Guide (STIG) released for the OS/Platform/application. Each guide is different, some have in excess of 200 controls and apply to various part of an OS but each guide is updated regularly by (DISA).

---

**Note:** DISA is part of the United States Department of Defense.

---

### Control Severities

Controls are divided into groups based on the following properties:

- **High (CAT I)** These controls have a large impact on the security of a system. They also have the largest operational impact to a system and deployers should test them thoroughly in non-production environments.
- **Medium (CAT II)** These controls are the bulk of the items in the STIG and they have a moderate level of impact on the security of a system. Many controls in this category will have an operational impact on a system and should be tested thoroughly before implementation.
- **Low (CAT III)** These controls have a smaller impact on overall security, but they are generally easier to implement with a much lower operational impact.

---

**Note:** All of the default configurations are found within - remediation - `defaults/main.yml` - audit

- standalone `vars/STIG.yml`
- combined `vars/[system_hostname].yml`

---

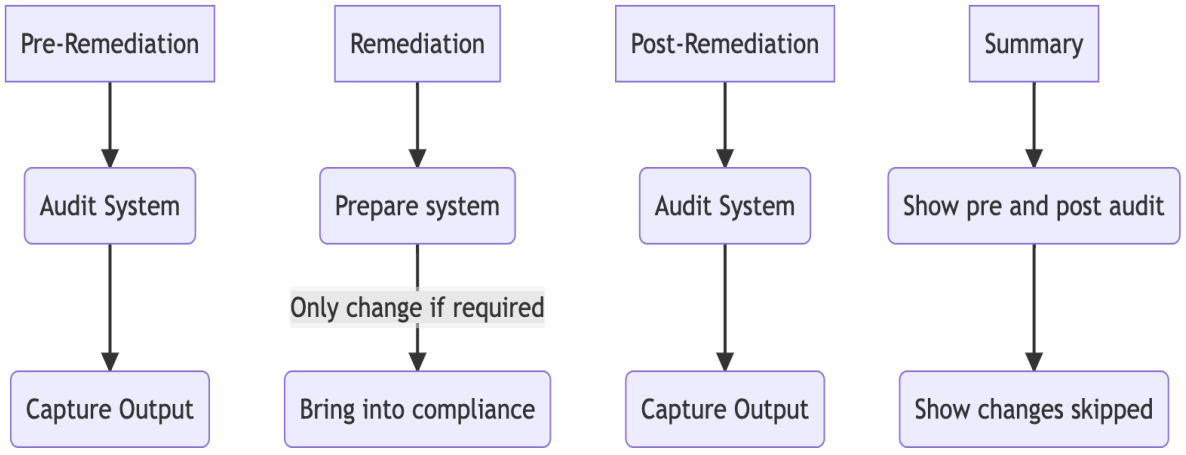
## 1.4 What is provided?

The content provided is open-source licensed configurations to assist in achieving or auditing compliance to one of the benchmark providers listed above.

This consists of two components:

- **Audit**
  - Runs a small single binary on the system written in GO called GOSS.
  - Enables you to very quickly scan your host and output the status of compliance for your host.
- **Remediate**
  - Has the ability to run from a central location using the configuration management tool ansible.
  - Can assist with bringing your host into compliance for the relevant benchmark.

Both can be run alone or in conjunction with each other.



## 1.5 How is this written?

We analyze each configuration control from the applicable benchmark to determine what impact it has on a live production environment and how to best implement a way to audit the current configuration and how to achieve those requirements using Ansible. Tasks are added to the role that configure a host to meet the configuration requirements. Each task is documented to explain what was changed, why it was changed, and what deployers need to understand about the change.

Deployers have the option to enable/disable every control that does not suit their environments needs. Every control item has an associated variable that can be used to switch it on or off.

Additionally, the items that have configurable values, i.e. number of password attempts, will generally have a corresponding variable that allows for customization of the applied value. It is imperative for each deployer to understand the regulations and compliance requirements that their organization and specific environments are responsible for meeting in order to effectively implement the controls in the relevant benchmark.

## 1.6 Development Process

### 1.6.1 Lifecycle of releases and branches

While Remediate and Audit are individually managed processes, nevertheless, some of the content is linked. There are occasions where both need updating or just one of them.

As a rule, our goal is to abide to the following lifecycle process for branches and releases that include ansible-galaxy sync updates. Being community, we have direct customer requests and requirements that take priority in releases.

### 1.6.2 Branches

- **devel branch**
  - Staging area for bug fixes, PRs and new benchmarks.
  - Default and working development branch.
  - We aim to get majority of PRs merged to devel between 2-4 weeks.
  - Community Collaboration PR (Pull Request) Branch.
- **main branch** *(The release branch)*
  - We merge from devel to main branch.
  - main branch is dependent on the severity and impact of issues closed.
  - Routinely, a release alignment is every 8-12 weeks (sometimes much quicker).
  - Once a new CIS/STIG benchmark gets released, we aim to merge the new tagged release (2-4 weeks).
  - The major releases are sourced and linked to ansible-galaxy Roles.

### 1.6.3 Demos


lockdown
ENTERPRISE


LONDON
INFRASTRUCTURE
AS CODE
MindPoint
GROUP SM
ANSIBLE
Ansible Lockdown: Mark Bolwell
Automating audit and compliance benchmarks

CHAPTER

TWO

# AUDIT



## 2.1 Overview

Ansible remediation for security benchmarks now utilizes an open-source go binary called goss to audit the system.

Ensuring consistency in checks by using the same settings and controls that have been enabled in the remediation steps, are the same ones checked by the audit.

## 2.2 Considerations

- The audit runs using the host only compute resources (memory/cpu).
- Please be aware this may have adverse effect running on a heavily utilized system.
- Please consider shared resources when running on many hosts simultaneously.

It can be run in two ways:

- Enabled to run as part of the ansible playbook and is setup to capture pre remediation and post remediation states. Using the same configured variables as used in remediation See Using Audit and Remediate at the same time
- Standalone script
  - run_audit.sh (Linux (shell))
  - run_audit.ps1 (Windows(powershell))

## 2.3 Currently Enabled Playbooks

- CIS Benchmarks
- STIG Benchmarks

## 2.4 Setup auditing as standalone

It is presumed that you have the script downloaded and the audit content ready from source control or your own configured location.

The following requirements are needed (OS dependant)

- Super user or permissions to run privilege commands
  - Linux sudo can work
  - Windows ability to run security audits and query group or local policy.
- goss binary appropriate for the OS
  - The binary must be accessible by the OS so the scripts can run smoothly with appropriate rights.
    - The expected path for the binary is found inside the relevant OS script and can be adjusted as required.
  - Linux
    - Binary
    - Checksum
  - Windows
    - Binary
    - Checksum

---

**Note:** The binary only needs to be accessible to the host with ability to use. The relevant script needs to be adjust to point to the path of the binary.

---

## 2.5 Running the Audit Only as part of remediate playbook

It is possible to just run the audit on some playbooks (being rolled out across them all). This is a variable set

```
audit_only: true
```

This will run the audit based on the same release as the playbook and will then stop. Extra variables also enable the ability to copy back the audit output to the control node and create a directory structure.

```
# As part of audit_only
# This will enable files to be copied back to control node
fetch_audit_files: false
# Path to copy the files to will create dir structure
audit_capture_files_dir: /some/location to copy to on control node
```

## 2.6 Defining the audit

Each script runs against a configures variables file found in the content location in

```
{downloaded content}/vars/{benchmark}.yml
```

These are the variables that configure which controls are run along with some configurable settings during an audit.

Each script has the ability for you to set several variables depending on your environment requirements. e.g. locations on where to find binary or output locations

There are also switch options to allow you to run a couple of these benchmarks at one time.

Script runtime options

- The group option allows a meta field that can be assigned against the report for use in the analysis if servers are under the same group.

If more than one server group is analyzed, groups can be separated with commas.

- The full audit report has the saved output filename and location information.

## 2.7 Running on Linux

- Script
  - run_audit.sh (found in content directory)

Understanding variables:

- Uppercase variable are the only ones that should need changing
- lowercase variables are the ones that are discovered or built from existing.

script variables example:

```
AUDIT_BIN="${AUDIT_BIN:-/usr/local/bin/goss}"  # location of the goss executable
AUDIT_FILE="${AUDIT_FILE:-goss.yml}"  # the default goss file used by the audit provided␣
↪by the audit configuration
AUDIT_CONTENT_LOCATION="${AUDIT_CONTENT_LOCATION:-/var/tmp}"  # Location of the audit␣
↪configuration file as available to the OS
```

script help

```
Script to run the goss audit

Syntax: ./run_audit.sh [-f|-g|-o|-v|-w|-h]
options:
-f     optional - change the format output (default value = json)
-g     optional - Add a group that the server should be grouped with (default value =␣
↪ungrouped)
-o     optional - file to output audit data
-v     optional - relative path to the vars file to load (default e.g. /var/tmp/RHEL7-
↪CIS/vars/CIS.yml)
-w     optional - Sets the system_type to workstation (Default - Server)
-h     Print this Help.
```

(continues on next page)

(continued from previous page)

```
Other options can be assigned in the script itself
```

## 2.8 Running on Windows

- Script
  - run_audit.ps1 (found in content directory)

Variables can be set within the script

**Variables for Audit**

```
$DEFAULT_CONTENT_DIR = "C:\remediation_audit_logs"  # This can be changed using cli
$DEFAULT_AUDIT_BIN = "$DEFAULT_CONTENT_DIR\goss.exe"  # This can be changed using cli␣
↪option
```

**script help**

```
NAME
    C:\remediation_audit_logs\Windows-2019-CIS-Audit\run_audit.ps1

SYNOPSIS
    Wrapper script to run an audit


SYNTAX
    C:\remediation_audit_logs\Windows-2016-CIS-Audit\run_audit.ps1 [[-auditbin] <String>
↪] [[-auditdir] <String>]
    [[-varsfile] <String>] [[-group] <String>] [[-outfile] <String>] [<CommonParameters>]


DESCRIPTION
    Wrapper script to run an audit on the system using goss.
    This allows for bespoke variables to be set


PARAMETERS
    -auditbin <String>

    -auditdir <String>
        default: $DEFAULT_CONTENT_DIR
        Ability to change the location of where the content can be found
        This is where the audit content is stored
        e.g. c:/windows_audit

    -varsfile <String>
        default: $DEFAULT_VARS_FILE
        Ability to set a variable file defined with the settings to match your␣
↪requirements

    -group <String>
```

(continues on next page)

(continued from previous page)

```
        default: none
        Ability to set a group that the system belongs to
        Can be used when matching similar system in that same group

    -outfile <String>
        default: $AUDIT_CONTENT_DIR\audit_$host_os_hostname_$host_epoch.json
        Ability to set an outfile to send the full audit output to
        Requires path to be set.
        e.g. c:/windows_audit_reports

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug,
        ErrorAction, ErrorVariable, WarningAction, WarningVariable,
        OutBuffer, PipelineVariable, and OutVariable. For more information, see
        about_CommonParameters (http://go.microsoft.com/fwlink/?LinkID=113216).

    -------------------------- EXAMPLE 1 --------------------------

    PS C:\>./run_audit.ps1

    ./run_audit.ps1 -auditbin c:\path_to\binary.name
    ./run_audit.ps1 -auditdir c:\somepath_for_audit_content
    ./run_audit.ps1 -varsfile myvars.yml
    ./run_audit.ps1 -outfile path\to\audit\output.json
    ./run_audit.ps1 -group webserver
```

CHAPTER

# THREE

# REMEDIATE

This role is part of the Ansible Lockdown project and can be used as a standalone role or it can be used along with other Ansible roles, playbooks, or collections.



**Warning:** It is strongly recommended to run the role in a test environment first. There are controls that could introduce breaking changes. Check mode might not always catch these changes. The best way to confirm how the role will change your system is to fully test.

## 3.1 Requirements

This documentation assumes that the reader has completed the steps within the

- Ansible installation guide.

and has a good understanding of using ansible

- Ansible User Guide.

**Note:** The role requires elevated privileges and must be run as a user with `sudo` access. The example above uses the `become` option, which causes Ansible to use sudo before running tasks.

## 3.2 Installation

The recommended installation methods for this role are `ansible-galaxy` (recommended) or `git`.

### 3.2.1 Using `ansible-galaxy`

The easiest installation method is to use the `ansible-galaxy` command that is provided with your Ansible installation:

The general format is ansible-galaxy install git+|url to repo|, below is an example with RHEL8-CIS

```
ansible-galaxy install git+https://github.com/ansible-lockdown/RHEL8-CIS.git
```

The `ansible-galaxy` command will install the role into `/etc/ansible/roles/` and this makes it easy to use with Ansible playbooks.

### 3.2.2 Using `git`

Start by cloning the role into a directory of your choice, this example uses ~/.ansible/roles which can be changed out for any folder. That folder needs to be empty:

To clone and create a folder with the same name as the repo:

```
cd ~/CIS_Roles
git clone https://github.com/ansible-lockdown/RHEL8-CIS.git
```

To clone and put the files from the repo into a specific folder, folder does need to be empty:

```
mkdir ~/CIS_Roles
git clone https://github.com/ansible-lockdown/RHEL8-CIS.git ~/CIS_Roles
```

Ansible looks for roles in `~/.ansible/roles` by default.

If the role is cloned into a different directory, that directory must be provided with the `roles_path` option in `ansible.cfg`. The following is an example of a `ansible.cfg` file that uses a custom path for roles:

```
[DEFAULTS]
roles_path = /etc/ansible/roles:/home/myuser/custom/roles
```

With this configuration, Ansible looks for roles in `/etc/ansible/roles` and `~/custom/roles`.

## 3.3 How to Use

### 3.3.1 On It's Own

This role can be used on it's own as a role. The file `site.yml` is the included file to point to. This role does not include an inventory file for hosts since that is too site specific, that will need to be managed locally. Below are examples of how to run in various scenarios

CLI - Notice the reference to site.yml

```
cd roles
ansible-playbook -i hosts -e '{ "rhel8stig_cat2_patch":false,"rhel8stig_cat3_patch
↪":false }' ./RHEL8-STIG/site.yml'
```

Tower Steps

### 3.3.2 With Existing Playbooks

This role works well with existing playbooks. The following is an example of a basic playbook that uses this role:

```
---

- hosts: servers
  become: yes
  roles:
    - role: RHEL8-CIS
      when:
        - ansible_os_family == 'RedHat'
        - ansible_distribution_major_version | version_compare('8', '=')
```

### 3.3.3 Variables and the Role

The role is fully customizable by setting the variables provided in the `defaults/main.yml` file. These variables range in usage from toggling entire sections (CIS), categories (STIG), general groups (GUI related), individual controls, localized settings, etc. There are comments around these variables that have a description of what the variable does, what the value options are, and what controls are associated with the variable. Variables are also listed in order of appearance in the execution of the role, variables used early in the are listed earlier in the file. Variables in this location are also very low in precedence, here is the official list of variable precedence. This means they are over-written very easily via extra vars

This role has been written with ease of use in mind, which means it's written in a way that requires as little user interaction as possible. No need to modify any tasks at all!

### 3.3.4 Using and Modifying Variables Directly (`defaults/main.yml`)

This is the most basic way to make the change. The file has all of the available variables along with comments on what task the variable is for, a description on what the variable is, and the formatting for the value in the variable. Just update the values as needed

### 3.3.5 Modifying Variables with Extra-Vars

This is where the power of using variables via `defaults/main.yml` come into play. Anywhere you can use or set an extra var is place you can set these variables.

CLI In-Line setting (Set to only run STIG CAT1)

```
ansible-playbook -i host_file -e '{ "rhel8stig_cat2_patch":false,"rhel8stig_cat3_patch
↪":false }' ./RHEL8-STIG/site.yml
```

### 3.3.6 Using Tags

Each control is tagged with various pieces of information about the control to allow for more refined use with skipping or running controls. For STIG this includes all of the ID's, CIS has the level2 data, and both have info related to what the control relates to. For example all controls related to SSH will have the `ssh` tag.

STIG Example:

```
tags:
  - RHEL-08-040137
  - CAT2
  - CCI-001764
  - SRG-OS-000368-GPOS-00154
  - SV-244546r809339_rule
  - V-244546
  - fapolicy
```

CIS Example:

```
tags:
    - level1-server
    - level1-workstation
    - automated
    - patch
    - dhcp
    - rule_2.2.5
```

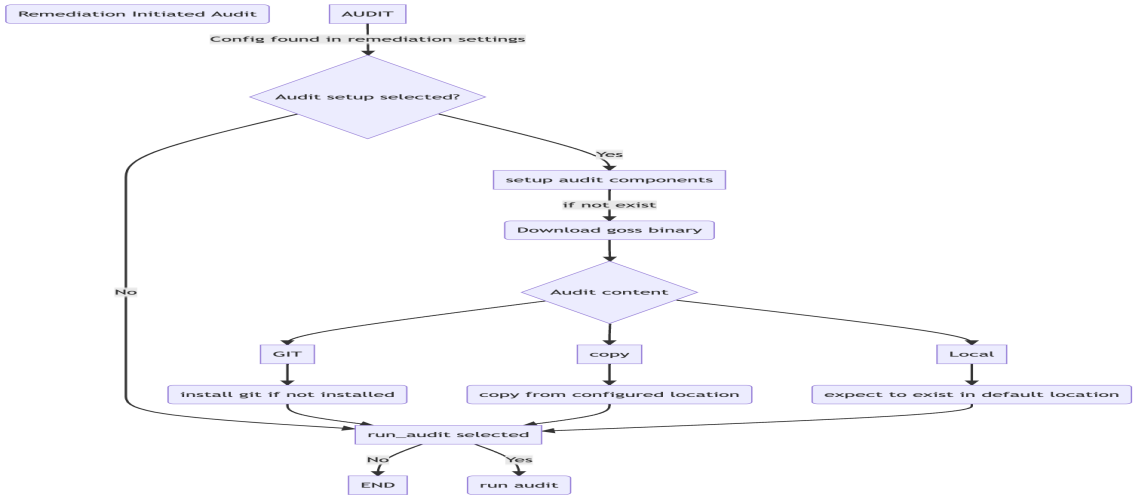
CHAPTER

FOUR

# USING AUDIT AND REMEDIATE TOGETHER

Using both Audit and Remediate in a workflow

This is missing the copy remediate settings step



CHAPTER

FIVE

# RELEASE SCHEDULE

## 5.1 Vendor Benchmark Schedule

### 5.1.1 CIS

CIS (Center for Internet Security) releases developed security configurations and best practices that the community validates. There is no official schedule, although CIS tends to release new or updated benchmarks in a draft format, which individuals can subscribe to prior to an official release.

### 5.1.2 STIG

STIG (Security Technical Implementation Guide) releases are developed by vendors in conjunction with requirements from DISA. STIG release schedule is managed and released on a quarterly schedule.

### 5.1.3 Playbook Releases

We aim to release remediate benchmarks within 30 days of vendor release to subscribers, this is subject to the number of changes, with new benchmarks often taking significantly longer. Public releases will be approximately three months after subscriber release.

### 5.1.4 Example

Table 1: Example Release Schedule for updated benchmarks

| Benchmark | Vendor Release | Subscriber release | Public Release |
|---|---|---|---|
| RHEL8-STIG | 27th January | 27th February | 27th May |

Becoming a subscriber is simple, visit https://lockdownenterprise.com for more information.

CHAPTER

SIX

# CIS BENCHMARKS

## 6.1 Operating Systems

Table 1: CIS Linux Benchmark Availability

| Benchmark | Maintained | Remediate | Audit | Release |
|---|---|---|---|---|
| AMAZON2-CIS | True | True | True | |
| AMAZON2023-CIS | True | True | True | |
| DEBIAN11-CIS | True | True | WIP | N/A |
| RHEL7-CIS | True | True | True | |
| RHEL8-CIS | True | True | True | |
| RHEL9-CIS | True | True | True | |
| UBUNTU18-CIS | True | True | True | |
| UBUNTU20-CIS | True | True | True | |
| UBUNTU22-CIS | True | True | True | |

Table 2: CIS Windows Benchmark Availability

| Benchmark | Maintained | Remediate | Audit | Release |
|---|---|---|---|---|
| Windows-10-CIS | True | True | WIP | N/A |
| Windows-11-CIS | True | True | WIP | N/A |
| Windows-2016-CIS | True | True | True | |
| Windows-2019-CIS | True | True | True | |
| Windows-2022-CIS | True | True | WIP | |

## 6.2 Cloud Platforms

Table 3: CIS Platform Benchmark Availability

| Benchmark | Maintained | Remediate | Audit | Release |
|---|---|---|---|---|
| AWS-Foundations | True | True | False | |
| Azure-CIS | True | True | False | |

## 6.3 Applications

Table 4: CIS Application Benchmark Availability

| Benchmark | Maintained | Remediate | Audit | Release |
|---|---|---|---|---|
| Apache-2.4-CIS | True | True | False | |
| Postgres-12-CIS | True | True | False | |
| Kubernetes1.6.1-CIS | True | True | False | |
| NGINX-CIS | True | WIP | WIP | N/A |

## 6.4 Archived Roles

None currently

CHAPTER

SEVEN

# CIS SPECIFIC INFORMATION

## 7.1 Advanced Options

**Note:** These are advanced options and required a greater understanding of all aspects of implementation.

- auditd_exclusion:

auditd logs can fill up very quickly with the default CIS options to log every privileged commands. Whether scanners/automation or and job that needs to run against a system with privilege access. e.g.sudo

There is the ability to change this for specific users to exclude anything in user space. This will still capture login/logout and sshd process but anything else will be excluded for that user. This can be enabled with the following (this needs to be set in an alternate variable location):

```
allow_auditd_uid_user_exclusions: true
```

Then a list of applicable users can be added to the exclusions. e.g.

```
rhel8cis_auditd_uid_exclude:
- ansible
- vagrant
```

CHAPTER

EIGHT

# STIG BENCHMARKS

## 8.1 Operating Systems

Table 1: STIG Windows Benchmark Availability

| Benchmark | Maintained | Remediate | Audit | Release |
|---|---|---|---|---|
| Windows-10-STIG | True | True | False | |
| Windows-2016-STIG | True | True | False | |
| Windows-2019-STIG | True | True | False | |
| Windows-2022-STIG | True | WIP | WIP | N/A |

## 8.2 Networking

Table 2: STIG Hardware Benchmark Availability

| Benchmark | Maintained | Remediate | Audit | Release |
|---|---|---|---|---|
| Cisco-IOS-L2S | True | True | False | |

## 8.3 Applications

Table 3: STIG Application Benchmark Availability

| Benchmark | Maintained | Remediate | Audit | Release |
|---|---|---|---|---|
| Apache-2.4-STIG | True | True | False | |
| TOMCAT-9-STIG | True | True | False | |
| Windows_Advance_Firewall-STIG | True | True | True | |
| KUBERNETES-STIG | True | True | False | |

## 8.4 Archived Roles

Table 4: STIG Retired Benchmark

| Benchmark | Maintained | Remediate | Audit | Release |
|---|---|---|---|---|
| RHEL5-STIG | False | False | False | N/A |
| RHEL6-STIG | False | False | False | N/A |
| Windows-2008R2-Member-Server-STIG | False | False | False | N/A |
| Windows-2012-Member-Server-STIG | False | False | False | N/A |
| Windows-2012-Domain-Controller-STIG | False | False | False | N/A |
| Postgres-9-STIG | True | True | False | N/A |

CHAPTER

NINE

# CONTRIBUTING

Commonly with GitHub open-source repositories and projects, community contributions are via GitHub Issues and Pull Requests.

We do ask the following:

- Respect the community's time and effort that gets put into feedback and support.
- Provide as much information as possible with issues or pull request.
- Pull Requests:
  - **signed-off-by (user is provided)**
  - **signed-by (gpg key signed)**

## 9.1 How to contribute to audit development

### 9.1.1 Adding code

We strive to maintain a set of standards we would like to achieve for all code written.

### 9.1.2 Considerations

- Keep it as simple as possible to aid investigation and debug.
- Ensure it aligns with the remediate portion **(the remediate and audit repos are intrinsically linked when combined)**.

### 9.1.3 Layout

- Each control should be in its own file.
- Use variables wherever possible.
- Some controls were associated; possibly grouped.
  - Multiple stages test (config and running tests)
  - Similar tests (filesystem mount options)
- Structure should be where appropriate

```
|- control group e.g. section_1
|-- grouped controls
|---- control test
```

**e.g.**

```
|- section_1
|-- cis_1.1
|--- cis_1.1.x.yml
```

### Content

- Each task requires the following to be included:
  - Title
  - At least one control variable (whether control ID is to be run or not).
    - If only one test add the variable prior to to the goss_module itself
  - Use appropriate goss_modules before using command
    - There are circumstances when command allows discovery/filters results

For a full list of goss and how to use the goss_modules (tests). Goss Docs

**Example**

*Basic test*

```
{{ if .Vars.rhel9cis_level_1 }}
  {{ if .Vars.rhelcis9_1_1_10 }}
command:
  usb-storage:
    title: 1.1.10 | Disable USB Storage
    exit-status: 0
    exec: "modprobe -n -v usb-storage | grep -E '(usb-storage|install)'"
    stdout:
    - install /bin/true
    meta:
      server: 1
      workstation: 2
      CIS_ID: 1.1.10
      CISv8:
      - 10.3
      CISv8_IG1: true
      CISv8_IG2: true
      CISv8_IG3: true
  {{ end }}
{{ end }}
```

**Breakdown**

```
{{ if .Vars.rhel9cis_level_1 }}                                                ##␣
↪if rhel9cis_level_1 is true
  {{ if .Vars.rhelcis9_1_1_10 }}                                               ##␣
```

(continues on next page)

(continued from previous page)

```
↪if rhelcis9_1_1_10 is true
command:                                                                    ##␣
↪goss_module
  usb-storage:                                                              ##␣
↪unique name associated with the command
    title: 1.1.10 | Disable USB Storage                                     ##␣
↪title  {{ control id }}| {{ control title }}
    exit-status: 0                                                          ##␣
↪Options for goss_module
    exec: "modprobe -n -v usb-storage | grep -E '(usb-storage|install)'"    ##␣
↪Options for goss_module
    stdout:                                                                 ##␣
↪Options for goss_module
    - install /bin/true                                                     ##␣
↪Options for goss_module
    meta:                                                                   ##␣
↪Meta data used for reporting (see metadata)
      server: 1
      workstation: 2
      CIS_ID: 1.1.10
      CISv8:
      - 10.3
      CISv8_IG1: true
      CISv8_IG2: true
      CISv8_IG3: true
  {{ end }}                                                                 ##␣
↪Close if statement
{{ end }}                                                                   ##␣
↪Close if statement
```

**Variable precedence**

Variables should be added higher in the test based on the level of impact.

### Metadata

This is added to the audit benchmark for reference across compliance requirements. There are two levels of metadata:

- **audit metadata** - this is general system information and audit information.
- **control metadata** - this is added to every audit control and is specific to each control.

**Audit Metadata** *(required)*

- These are items set/discovered about the system within the script set via vars in the script.
- Referenced in the goss.yml file.

**Contains:**

Table 1: Discovered audit variables

| Variable Title | Script variable name | Purpose |
|---|---|---|
| host_machine_uuid: | {{ .Vars.machine_uuid }} | discovered UUID of system (used as unique identifier) |
| host_epoch: | {{ .Vars.epoch }} | epoch time that script initiated (part of output filename) |
| host_os_locale: | {{ .Vars.os_locale }} | system locale (TZ) |
| host_os_release: | {{ .Vars.os_release }} | OS version (e.g. 7) |
| host_os_distribution: | {{ .Vars.os_distribution }} | OS distribution ( e.g. rhel) |
| host_hostname: | {{ .Vars.os_hostname }} | hostname |
| host_system_type: | {{ .Vars.system_type }} | |
| | Linux | Server/Workstation Manually set (default server) |
| | Windows | pulled from regkey and set |

**Special Variables**

- **host_automation_group:** {{ .Vars.auto_group }}
  - Used to group like systems when reporting
  - If run via remediate, it uses host group memberships
  - If run via script, it is an optional value or null

**Control Metadata** (required)

- This consists of data found in the benchmark documentation
- This potentially changes with each release update (this will need to be correct for the release being worked on)

**CIS Specific**

This contains the following:

- `server:` cis level options: (1|2)
- `workstation:` cis level: (1|2|NA)
- `CIS_ID:` control reference
- `CISv8:` list of associated groups the control is associated with
- `CISv8_IG1:` Boolean (if meets that association)

```
meta:
  server: 1
  workstation: 1
  CIS_ID: 1.1.1.1
  CISv8:
```

(continues on next page)

(continued from previous page)

```
  - 4.8
  CISv8_IG1: false
  CISv8_IG2: true
  CISv8_IG3: true
```

**STIG Specific**

All can be found in the details of the control itself

- `Cat:` the category control is associated with either (1|2|3)
- `CCI:` Common identifier is found in the STIG documentation
- `Group_Title:` is the associated group that particular control is a part of
- `Rule_ID:` changes with every iteration of the control details
- `STIG_ID:` control ID known by STIG
- `Vul_ID:` vulnerability identifier

```
meta:
  Cat: 1
  CCI:
  - CCI-001494
  - CCI-001496
  - CCI-002165
  - CCI-002235
  Group_Title: SRG-OS-000257-GPOS-00098
  Rule_ID: SV-204392r646841_rule
  STIG_ID: RHEL-07-010010
  Vul_ID: V-204392
```

### 9.1.4 Gotchas

If you have two tasks that refer to the same file or command (this is currently the unique identifier used in goss), it will only give you the result of one test (the last one ran).

e.g.

```
file:
 /etc/selinux/config:
   title: 1.6.1.4 | Ensure the SELinux mode is not disabled | config
   exists: true
   contains:
   - '/^SELINUX( |)=( |)(enforcing|permissive)/'
   - '!/^SELINUX( |)=( |)disabled/'
   meta:
     server: 1
     workstation: 1
     CIS_ID:
     - 1.6.1.4
     CISv8:
     - 3.3
     CISv8_IG1: true
```

(continues on next page)

(continued from previous page)

```
    CISv8_IG2: true
    CISv8_IG3: true
```

and

```
file:
/etc/selinux/config:
  title: 1.6.1.5 | Ensure the SELinux mode is enforcing | config
  exists: true
  contains:
  - '/^SELINUX( |)=( |)enforcing/'
  - '!/^SELINUX( |)=( |)disabled/'
  meta:
    server: 2
    workstation: 2
    CIS_ID:
    - 1.6.1.5
    CISv8:
    - 3.3
    CISv8_IG1: true
    CISv8_IG2: true
    CISv8_IG3: true
```

**Only one will give you results**

## 9.2 How to Contribute to Remediate Development

### 9.2.1 Remediate Code Summary

We strive to maintain a set of standards and consistency for all code that is written. Please review the formatting of previously written controls and follow the formatting currently in place. We do have a style guide on writing controls that document standard linting, parameter use/order, tagging, etc.

### 9.2.2 Remediate Code Considerations

- **Keep it as simple as possible** - This is to aid investigation and debugging. Overly complicated tasks/controls are harder to troubleshoot when things go wrong.
- **Readability is key** - This means the most clever way to write something might lead to the task being complicated and hard to read. - For example if you have a multi-axis loop, with vars spread across the role that reference tasks of their own, with jinja filters on top of json filters to do something in one task instead of having three tasks to accomplish the same thing. (Please create the three tasks for readability)
- **Controls only do what the control ask for** - Our Audit tool and Remediate are intrinsically linked when combined. Things like variables and how the search is executed in our Audit rely on the Remediate being correct when run from the Remediate playbook. - Scanners also use the Fix Text and/or intent of the control (sometimes the Fix Text has mistakes…) to check for compliance. If you deviate from this, scanners find false positives. - There should be no extra security settings set (even if they are good ideas to set). These roles expect to only set what is defined in the STIG or CIS benchmarks. If other security settings are set, it can cause confusion.

### 9.2.3 Remediate Code Layout

#### General Layout

- Each control should be it's own task. If it needs to be multiple tasks create a block for those tasks, see example in the Layout sections
- Users should not have to edit tasks, but vars in `defaults/main.yml`. If a control has any variability create a variable for that value
- Follow the structure listed in the Layout sections
- For controls that install/uninstall packages please use package module and use ansible_facts.packages in the when
  - The package module will use the default package manager for that system, for example `yum` (RHEL 7), `dnf` (RHEL 8+), `apt` (Ubuntu), etc.

  This allows for a more unified use of tasks between all roles - The use of `ansible_facts.packages` will skip if it does not need to run and add efficiency to run time

#### STIG Control Task Layout

- **Name** - `name:` - Each control gets it's own task and that task needs a name. The name format is category | STIG ID | PATCH or AUDIT | title of control copied from STIG. - PATCH or AUDIT - This is in reference to the task making a change (PATCH) or not making a change (AUDIT). Tasks that don't make changes are one that

  generally gather information to be used later
- **Block** - If there are more than one task to complete a control please keep those in a single task but using a block format. - When using blocks the steps should have a pipe ( | ) after the title followed by a description of what that task is doing in the block.
- **Module** - This is just the module being used to execute that task, nothing special here
- **Parameters** - When using the `shell` module to gather information please set `changed_when` and `failed_when` to false. This will cause the task to always run and register which always creates the variable registering. The action parts of the task that use that var should handle the var if a fail occurred during the `command` or `shell` function - Please always use `shell` over `command`. The `command` module is being phased out, but also for consistency please use `shell` - When using modules that have alias's, please do not use the alias since those are often not part of the module documentation - When using modules that require a path type of parameter please use that first - When using modules that regex, please use that second after path where applies
- **Registers** - Please adhere to the exiting format of registered variable names - Dynamic variables are variables set in-line of a task, rhel_08_010382_sudoers_all in the example below. Please follow the all lower case using underscores instead of dashes STIG ID followed by a descriptive name. This helps keep variable names from overlapping in this role or other playbooks/collections you use this role with.
- **Variables** - Any value that can deviate from the example used in the STIG fix text. For example tasks that ask to set permissions X or more restrictive that the permissions value should be a variable - Any value that has multiple value options should be a variable - These variables should be defined in the `defaults/main.yml` file - These variables should be formatted as role name followed by the descriptor, rhel8stig_disruption_high for example
- **With_items** - When using loops please use with_items for consistency. We know `loop` has much of the same functionality as `with_items` but for consistency we would like `with_items` since it covers all uses - Please use `loop_control` on wordy loops - Please put the loop list below the `with_items` like in the example
- **When** - *(Please use when statements on all controls)* - The control should have the when set to run when the var for the individual task toggle set to true. That toggle is the STIG ID, all lower case with underscores instead of dashes - When you are outside of the block please stack the `when` values under the `when` call, see example below

for clarification. - When you are inside of the block you can use use single line for `when` and value in a single `when` instance. If there are and/or whens please stack those under the when - Please do not use empty compares, if you are basing your task run off an empty var please use | length > 0 or | length == 0.

- **Tags** - All controls must have tags, but the individual tasks in the block do not get tags. See the example below for clarification - The tags are in this specific order:
  - STIG ID copied from the STIG
  - Category
  - CCI value (NIST group ID)
  - Security Group ID
  - Rule ID
  - Vulnerability ID
  - Descriptor of what the task is involved with. For example ssh, selinux, pamd, gui, etc. This tag is always lowercase

```
- name: "MEDIUM | RHEL-08-010382 | PATCH | RHEL 8 must restrict privilege elevation to␣
↪authorized personnel."
  block:
      - name: "MEDIUM | RHEL-08-010382 | AUDIT | RHEL 8 must restrict privilege␣
↪elevation to authorized personnel. | Get ALL settings"
        ansible.builtin.shell: grep -iws 'ALL' /etc/sudoers /etc/sudoers.d/* | cut -d":"␣
↪-f1 | uniq | sort
        changed_when: false
        failed_when: false
        register: rhel_08_010382_sudoers_all

      - name: "MEDIUM | RHEL-08-010382 | PATCH | RHEL 8 must restrict privilege␣
↪elevation to authorized personnel. | Remove format 1"
        ansible.builtin.lineinfile:
            path: "{{ item }}"
            regexp: 'ALL ALL=(ALL) ALL'
            state: absent
            validate: '/usr/sbin/visudo -cf %s'
        with_items:
            - "{{ rhel_08_010382_sudoers_all.stdout_lines }}"
        when: rhel_08_010382_sudoers_all.stdout | length > 0

      - name: "MEDIUM | RHEL-08-010382 | PATCH | RHEL 8 must restrict privilege␣
↪elevation to authorized personnel. | Remove format 2"
        ansible.builtin.lineinfile:
            path: "{{ item }}"
            regexp: 'ALL ALL=(ALL:ALL) ALL'
            state: absent
            validate: '/usr/sbin/visudo -cf %s'
        with_items:
            - "{{ rhel_08_010382_sudoers_all.stdout_lines }}"
        when: rhel_08_010382_sudoers_all.stdout | length > 0
  when:
      - rhel_08_010382
      - rhel8stig_disruption_high
```

(continues on next page)

(continued from previous page)

```
  tags:
      - RHEL-08-010382
      - CAT2
      - CCI-000366
      - SRG-OS-000480-GPOS-00227
      - SV-237641r646893_rule
      - V-237641
      - sudo
```

### CIS Control Task Layout

- **Name** - `name:` - Each control gets it's own task and that task gets a name. The name format is Control Number | PATCH or AUDIT | Title copied from CIS control
- **Block** - If there is more than one task to complete a control please those in a single task but using a block format, example below. - When using blocks the steps should have a pipe ( | ) after the title followed by a description of what that task is doing in the block.
- **Module** - This is just the module being used to execute that task, nothing special here
- **Parameters** - When using the `shell` module to gather information please set `changed_when` and `failed_when` to false. This will cause the task to always run and register which always creates the variable registering. The action parts of the task that use that var should handle the var if a fail occurred during the `command` or `shell` function
  - Please always use `shell` over `command`. The `command` module is being phased out, but also for consistency please use `shell`
  - When using modules that have alias's, please do not use the alias since those are often not part of the module documentation
  - When using modules that require a path type of parameter please use that first
  - When using modules that regex, please use that second after path where applies
- **Registers** - Please adhere to the exiting format of registered variable names - Dynamic variables are variables set in-line of a task, rhel8cis_4_1_1_1_3_grub_cmdline_linux in the example below. Please follow the all lower case standard using underscores instead of periods/dots with benchmark name followed by the CIS control number and finally a descriptive name. This helps keep variable names from overlapping in this role or other playbooks/collections you use this role with.
- **Variables** - Any value that can deviate from the example used in the STIG fix text. For example tasks that ask to set permissions X or more restrictive that the permissions value should be a variable - Any value that has multiple value options should be a variable - These variables should be defined in the `defaults/main.yml` file - These variables should be formatted as role name followed by the descriptor, rhel8stig_disruption_high for example
- **With_items** - When using loops please use with_items for consistency. We know `loop` has much of the same functionality as `with_items` but for consistency we would like `with_items`

**since it covers all uses**

  - Please use `loop_control` on wordy loops
  - Please put the loop list below the `with_items` like in the example

- **When** - *(Please use when statements on all controls)* - The control should have the when set to run when the var for the individual task toggle set to true. That toggle is the STIG ID, all lower case with underscores instead of dashes - When you are outside of the block please stack the `when` values under the `when` call, see example below

for clarification. - When you are inside of the block you can use use single line for `when` and value in a single `when` instance. If there are and/or whens please stack those under the when - Please do not use empty compares, if you are basing your task run off an empty var please use | length > 0 or | length == 0.

- **Tags** - All controls must have tags, but the individual tasks in the block do not get tags. See the example below for clarification - The tags are in this specific order:
  - Server Level
  - Workstation Level
  - Automated or Manual. This is from the CIS control in the benchmark documentation and is their assessment of the control being able to be automated or a manual control.

  If we automate or don't automate the control itself we use the value from the benchmark itself here - Patch or Audit. Does the overall task make any changes or just audit/message out - Descriptor of what the task is involved with. For example ssh, selinux, pamd, gui, etc. This tag is always lowercase - Number of the control. The format is rule_< the number>, rule_4.1.1.3 for example

```
- name: "4.1.1.3 | PATCH | Ensure auditing for processes that start prior to auditd is␣
↪enabled"
  block:
      - name: "4.1.1.3 | AUDIT | Ensure auditing for processes that start prior to␣
↪auditd is enabled | Get GRUB_CMDLINE_LINUX"
        ansible.builtin.shell: grep 'GRUB_CMDLINE_LINUX=' /etc/default/grub | sed 's/.$//
↪'
        changed_when: false
        failed_when: false
        check_mode: no
        register: rhel8cis_4_1_1_3_grub_cmdline_linux

      - name: "4.1.1.3 | PATCH | Ensure auditing for processes that start prior to␣
↪auditd is enabled | Replace existing setting"
        ansible.builtin.replace:
            path: /etc/default/grub
            regexp: 'audit=.'
            replace: 'audit=1'
        notify: grub2cfg
        when: "'audit=' in rhel8cis_4_1_1_3_grub_cmdline_linux.stdout"

      - name: "4.1.1.3 | PATCH | Ensure auditing for processes that start prior to␣
↪auditd is enabled | Add audit setting if missing"
        ansible.builtin.lineinfile:
            path: /etc/default/grub
            regexp: '^GRUB_CMDLINE_LINUX='
            line: '{{ rhel8cis_4_1_1_3_grub_cmdline_linux.stdout }} audit=1"'
        notify: grub2cfg
        when: "'audit=' not in rhel8cis_4_1_1_3_grub_cmdline_linux.stdout"
  when:
      - rhel8cis_rule_4_1_1_3
  tags:
      - level2-server
      - level2-workstation
      - automated
      - patch
      - auditd
```

(continues on next page)

(continued from previous page)

```
    - grub
    - rule_4.1.1.3
```

CHAPTER

# TEN

# GETTING SUPPORT

## 10.1 Contact Us

- Discord Community Discussions : Our Discord channels are direct community communications for all aspects of the Ansible-Lockdown repositories.
- MindPoint Group Official Subscription : If your organization requires you to have support, signed releases, or validated testing, those options are available.
- G2 : Read and Write our reviews at G2
- Twitter@AnsibleLockdown : Get the latest news from our Twitter feed on everything Ansible Lockdown.

## 10.2 MindPoint Group Official Site and Services

- MindPoint Group Official Website : If your organization has general questions regarding Cybersecurity or our Organization.
- engage@mindpointgroup.com : via Email
- LinkedIn : Our official LinkedIn Business Page
- Twitter@MindPointGroup : Get the latest news from our Twitter feed on everything MindPointGroup.

### 10.2.1 Known Issues

#### Audit

#### Remediate

**Cloud init fails** - Bug 1839899

**Affects**

- RHEL8-CIS - 1.1.3.3
- RHEL8-STIG - RHEL-08-040134

**All SELinux related modules are currently broken on RHEL8.6 with epel packages active.** - Bug 2093589

**Affects**

- Version-Release number of selected component (if applicable):
- REHL8.6
- Ansible 5.4
- Python 3.8

**Multiple missing Python 3.8 libraries to support normal Ansible playbook tasks.** - Bug 2093105

**Affects**

- Version-Release number of selected component (if applicable):
- ansible-core-2.12.2-3.1.el8.x86_64
- ansible-5.4.0-2.el8.noarch
- python38-3.8.12

### 10.2.2 Audit - FAQ

#### Why does goss run manually fail?

example.

```
goss -g goss.yml -v
```

Goss is designed to run from the scripts passing discovered variables into Goss for metadata. Without these values being set, Goss will fail. These metadata variables can bee seen towards the end of the goss.yml file. Futhermore, the run_audit script shows how these variables are created and passed to Goss.

### 10.2.3 Remediate - FAQ

#### Missing "jmespath" Fatal Error

```
fatal: [ansible]: FAILED! => {"msg": "You need to install \"jmespath\" prior to running␣
↪json_query filter"}
```

This can occur during a playbook run on certain operating systems when patching takes place as part of the playbook due to the way python is implemented.

- You Need to install jmespath : A great article and explanation written by Discord community member baassssiiee

#### Missing Sudo Password (Linux OS Based)

Many of the CIS Linux OS based (section 5.x) roles remove the ability for sudoers to use the NOPASSWD option to enable elevated privileges.

The user (unless root) that is running the playbook on the target should have a password set and the playbook run accordingly to pass the become_password.

All repositories should now run a pre-req check to ensure that the user does have a password set and will fail if this is not setup.

CHAPTER

ELEVEN

# GLOSSARY

The following is a list of Glossaries used in the documentation:

- Ansible RedHat Glossary
- Ansible RedHat ReadTheDocs
- Cyber.Mil Acronyms

CHAPTER

# TWELVE

# USEFUL LINKS

## 12.1 Main Content site

- Repositories

## 12.2 Audit

- Goss Main Site
- Goss Docs

## 12.3 Remediate

- Ansible Main Site
- Ansible Get Started
- Ansible Docs