Ansible-LockdownRHEL9-CISDocumentation:

MindPoint Group

Sep 23, 2025

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`

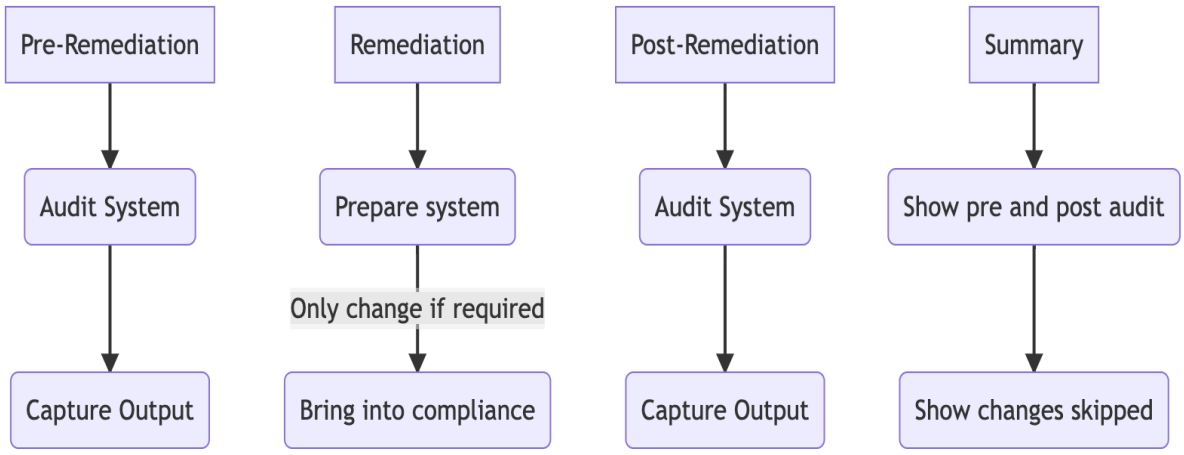
1.4 What is provided?

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.

Baselines are typically designed for the x86_64 architecture. But, with the growing popularity of ARM64/aarch64 systems for running environments, we have begun to extend our support for hardening on ARM as well. This rollout brings certain challenges, particularly regarding the audit component, and it is not officially covered by the baseline provider. However, we strive to provide support for ARM as part of our pipelines.

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. For a more in depth Overview

- *Release Schedule*

1.6.2 Branches

1.6.3 Remediate

- **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.

1.6.4 Audit

- **benchmark_version**

As these are very bespoke for each release the branches are named after the version of the benchmark it is to test. Allowing the playbook to test for he correct settings for that version

- RHEL9-CIS e.g. benchmark_v2.0.0

1.6.5 Demos


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. Ensure you have the correct binary for your architecture examples above are AMD64, but also works on ARM64 (may have bad results with auditd settings)

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
```

(continues on next page)

(continued from previous page)

```
# 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 configured 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

2.7.1 Bash Script Explanation: Goss Security Audit

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

This Bash script runs a **security audit** using **Goss**, a YAML-based testing framework. It is designed to be **Linux OS-agnostic**, configurable, and ensures compliance with **CIS or STIG** benchmarks.

1. Script Metadata and Change Log At the top, the script includes comments detailing changes made over time. This is useful for **tracking updates, fixes, and enhancements**.

2. Benchmark and Audit Variables Understanding variables:

- Uppercase variables are the only ones that should require changes.
- lowercase variables are the ones that are discovered or built from existing.

```
# Goss benchmark variables (these should not need changing unless new release)
BENCHMARK=CIS  # Benchmark Name aligns to the audit
BENCHMARK_VER=1.0.0
BENCHMARK_OS=RHEL9
```

Defines **benchmark name**, **version**, and **target OS**.

```
# Goss host Variables
AUDIT_BIN="${AUDIT_BIN:-/usr/local/bin/goss}"  # location of the goss executable
AUDIT_BIN_MIN_VER="0.4.4"
AUDIT_FILE="${AUDIT_FILE:-goss.yml}"  # default Goss configuration file
AUDIT_CONTENT_LOCATION="${AUDIT_CONTENT_LOCATION:-/opt}"  # Location for audit files
```

Defines **Goss binary location** and **audit file paths**.

3. Help Function

```
Help()
{
  echo "Script to run the goss audit"
  echo
  echo "Syntax: $0 [-f|-g|-m|-o|-v|-w|-h]"
  echo "options:"
  echo "-f     optional - change the format output (options json(default), documentation,
↪ rspecish)"
  echo "-g     optional - Add a group that the server should be grouped with (default␣
↪value = ungrouped)"
  echo "-m     optional - maximum concurrent processes (number, default 50)"
  echo "-o     optional - file to output audit data"
  echo "-v     optional - relative path to the vars file to load (default e.g. $AUDIT_
↪CONTENT_LOCATION/{OS}-$BENCHMARK/vars/$BENCHMARK.yml)"
  echo "-w     optional - Sets the system_type to workstation (Default - Server)"
  echo "-h     Print this Help."
  echo
}
```

Displays **usage instructions** when *-h* is provided.

4. Command-Line Arguments Handling

```
while getopts f:g:m:o:v::wh option; do
  case "${option}" in
    f ) FORMAT=${OPTARG} ;;  # Output format (json, rspecish, etc.)
    g ) GROUP=${OPTARG} ;;   # Defines server group
    m ) MAX=${OPTARG} ;;
    o ) OUTFILE=${OPTARG} ;; # Specifies output file
    v ) VARS_PATH=${OPTARG} ;; # Variables file path
    w ) host_system_type=Workstation ;; # Change system type to Workstation
    h ) Help; exit;; # Show help and exit
    ? ) echo "Invalid option: -${OPTARG}."; Help; exit;; # Invalid option handler
  esac
done
```

Uses *getopts* to process **command-line arguments**.

5. Pre-Checks

```
if [ "$(/usr/bin/id -u)" -ne 0 ]; then
  echo "Script needs to run with root privileges"
  exit 1
fi
```

Ensures the script runs with **root privileges**.

```
if [ "$(grep -Ec "rhel|oracle" /etc/os-release)" != 0 ]; then
  os_vendor="RHEL"
else
  os_vendor="$(hostnamectl | grep Oper | cut -d : -f2 | awk '{print $1}' | tr '[:lower:]
```

(continues on next page)

(continued from previous page)

```
↪')"
fi
```

Detects the **OS vendor**.

6. Audit Variables and File Paths

```
audit_content_version=$os_vendor$os_maj_ver-$BENCHMARK-Audit
audit_content_dir=$AUDIT_CONTENT_LOCATION/$audit_content_version
audit_vars=vars/${BENCHMARK}.yml
```

Defines paths for **storing audit results**.

7. Output File Handling

```
if [ -z "$OUTFILE" ]; then
  export audit_out=${AUDIT_CONTENT_LOCATION}/audit_${host_os_hostname}-${BENCHMARK}-$
↪{BENCHMARK_OS}_${host_epoch}.$format
else
  export audit_out=${OUTFILE}
fi
```

Dynamically sets the output filename based on system details.

8. Pre-Check for Goss Availability

```
if [ -s "${AUDIT_BIN}" ]; then
  goss_installed_version="$($AUDIT_BIN -v | awk '{print $NF}' | cut -dv -f2)"
  newer_version=$(echo -e "$goss_installed_version\n$AUDIT_BIN_MIN_VER" | sort -V | tail␣
↪-n 1)
  if [ "$goss_installed_version" = "$newer_version" ] || [ "$goss_installed_version" = "
↪$AUDIT_BIN_MIN_VER" ]; then
    echo "OK - Goss is installed and version is ok ($goss_installed_version >= $AUDIT_
↪BIN_MIN_VER)"
  else
    echo "WARNING - Goss installed = ${goss_installed_version}, does not meet minimum of
↪${AUDIT_BIN_MIN_VER}"
    export FAILURE=2
  fi
else
  echo "WARNING - The audit binary is not available at $AUDIT_BIN"
  export FAILURE=1
fi
```

Checks if **Goss is installed** and meets the minimum version requirement.

9. Running the Audit

```
echo "Audit Started"
$AUDIT_BIN -g "$audit_content_dir/$AUDIT_FILE" --vars "$varfile_path" --vars-inline "
↪$audit_json_vars" v $format_output > "$audit_out"
```

Executes the **Goss audit** with the specified **configuration file**.

10. Displaying the Audit Results

```
output_summary="tail -2 $audit_out"
format_output="-f $format"

if [ "$format" = json ]; then
   format_output="-f json -o pretty"
   output_summary='grep -A 4 \"summary\": $audit_out'
elif [ "$format" = junit ] || [ "$format" = tap ]; then
   output_summary=""
fi
```

Formats and extracts audit results based on the selected output format.

```
if [ "$(grep -c $BENCHMARK "$audit_out")" != 0 ] || [ "$format" = junit ] || [ "$format"␣
↪= tap ]; then
  eval $output_summary
  echo "Completed file can be found at $audit_out"
  echo "Audit Completed"
else
  echo -e "Fail: There were issues when running the audit, please investigate $audit_out"
fi
```

Checks if the audit ran successfully and notifies the user.

Table 1: **Bash Script Summary**

Feature	Description
Purpose	Runs a Goss-based OS security audit
Supported OS	Linux (RHEL, Oracle, etc.)
Customizable	Output format, grouping and audit file location
Pre-checks	Ensures script runs as **root** and checks Goss
Error Handling	Alerts for missing files and outdated versions

Running goss without script

This assumes you have goss and access to super user privileges.

It is possible to run goss in its raw form, while this is not recommended, for consistency it is added here.

The script discovers and adds extra inline variables to the goss output in the form of the metadata fields as found in the goss.yml This needs to be amended before being able to run in raw form.

- Edit goss.yml remove the lines starting at #metadata and the command tests Vars below

Goss can then be run manually

- full check

```
# {{path to your goss binary}} --vars {{ path to the vars file }} -g {{path to your␣
↪clone of this repo }}/goss.yml --validate
```

example:

```
# /usr/local/bin/goss --vars ../vars/cis.yml -g /home/bolly/rh8_cis_goss/goss.yml␣
↪validate
......FF....FF..............FF...F..FF............F.......................FSSSS......
↪.......FS.F.F.F.F.........FFFFF....
```

(continues on next page)

(continued from previous page)

```
Failures/Skipped:

Title: 1.6.1 Ensure core dumps are restricted (Automated)_sysctl
Command: suid_dumpable_2: exit-status:
Expected
    <int>: 1
to equal
    <int>: 0
Command: suid_dumpable_2: stdout: patterns not found: [fs.suid_dumpable = 0]


Title: 1.4.2 Ensure filesystem integrity is regularly checked (Automated)
Service: aidecheck: enabled:
Expected
    <bool>: false
to equal
    <bool>: true
Service: aidecheck: running:
Expected
    <bool>: false
to equal
    <bool>: true

< ---------cut ------- >

Title: 1.1.22 Ensure sticky bit is set on all world-writable directories
Command: version: exit-status:
Expected
    <int>: 0
to equal
    <int>: 123

Total Duration: 5.102s
Count: 124, Failed: 21, Skipped: 5
```

- running a particular section of tests

```
# /usr/local/bin/goss -g /home/bolly/rh8_cis_goss/section_1/cis_1.1/cis_1.1.22.yml ␣
↪validate
............

Total Duration: 0.033s
Count: 12, Failed: 0, Skipped: 0
```

- changing the output

```
# /usr/local/bin/goss -g /home/bolly/rh8_cis_goss/section_1/cis_1.1/cis_1.1.22.yml ␣
↪validate -f documentation
Title: 1.1.20 Check for removable media nodev
Command: floppy_nodev: exit-status: matches expectation: [0]
Command: floppy_nodev: stdout: matches expectation: [OK]
```

(continues on next page)

(continued from previous page)

```
< -------cut ------- >
Title: 1.1.20 Check for removable media noexec
Command: floppy_noexec: exit-status: matches expectation: [0]
Command: floppy_noexec: stdout: matches expectation: [OK]


Total Duration: 0.022s
Count: 12, Failed: 0, Skipped: 0
```

2.8 Fetch or Copy Audit Files

This section manages how audit output files are collected from managed nodes— either by fetching them to the controller or copying them to a centralized/shared location.

1. Fetch to Controller

```
- name: "FETCH_AUDIT_FILES | Fetch files and copy to controller"
  when: audit_output_collection_method == "fetch"
  ansible.builtin.fetch:
    src: "{{ item }}"
    dest: "{{ audit_output_destination }}"
    flat: true
  failed_when: false
  register: discovered_audit_fetch_state
  loop:
    - "{{ pre_audit_outfile }}"
    - "{{ post_audit_outfile }}"
  loop_control:
    label: "{{ item }}"
  become: false
```

- **Condition:** Runs only if `audit_output_collection_method == "fetch"`.
- **Module:** `fetch` copies files **from the managed node to the Ansible controller**.
- **src:** Points to the audit output file on the managed node.
- **dest:** Directory on the controller where files are saved.
- **flat:** Prevents creating full directory paths under `dest`.
- **failed_when:** Prevents task failure if the file doesn't exist.
- **register:** Stores the result in `discovered_audit_fetch_state`.
- **loop:** Iterates over both `pre_audit_outfile` and `post_audit_outfile`.
- **loop_control.label:** Improves log output for readability.
- **become:** `false` indicates no privilege escalation is used.

2. Copy on Managed Node

```
- name: "FETCH_AUDIT_FILES | Copy files to location available to managed node"
  when: audit_output_collection_method == "copy"
  ansible.builtin.copy:
    src: "{{ item }}"
```

(continues on next page)

(continued from previous page)

```
    dest: "{{ audit_output_destination }}"
    mode: 'u-x,go-wx'
    flat: true
  failed_when: false
  register: discovered_audit_copy_state
  loop:
    - "{{ pre_audit_outfile }}"
    - "{{ post_audit_outfile }}"
  loop_control:
    label: "{{ item }}"
```

- **Condition:** Runs if `audit_output_collection_method == "copy"`.
- **Module:** `copy` transfers files **within the managed node**, to a shared or central path.
- **src/dest:** Source and destination paths on the node.
- **mode:** Sets secure file permissions (*rw*——-).
- **flat:** Ensures output structure is flat.
- **register:** Stores result in `discovered_audit_copy_state`.

3. Show Warning if Fetch/Copy Fails

```
- name: "FETCH_AUDIT_FILES | Warning if issues with fetch or copy"
  when:
    - (audit_output_collection_method == "fetch" and discovered_audit_fetch_state is␣
↪defined and not discovered_audit_fetch_state.changed) or
      (audit_output_collection_method == "copy" and discovered_audit_copy_state is␣
↪defined and not discovered_audit_copy_state.changed)
  block:
    - name: "FETCH_AUDIT_FILES | Warning if issues with fetch_audit_files"
      ansible.builtin.debug:
        msg: "Warning!! Unable to write to localhost {{ audit_output_destination }} for␣
↪audit file copy"
```

- **Purpose:** Emits a warning if no files were transferred via `fetch` or `copy`.
- **Condition:** Based on whether the file transfer actually changed any state.
- **Message:** Informs the user that the output destination on localhost couldn't be written to.

Table 2: **Audit Fetch vs Copy Summary Table**

Feature	Description	Condition
Fetch files to controller	Copies files to control node using *fetch*	audit_output_collection_method == *fetch*"
Copy files on managed node	Copies files locally using *copy*	audit_output_collection_method == *copy*
Error Handling	Displays a warning if file transfer fails	Based on fetch/copy result *changed* status

2.9 Running on Windows

2.9.1 PowerShell Script Explanation: Goss Security Audit

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

Variables can be set within the script

This PowerShell script serves as a wrapper to run an audit on a system using *goss*. It allows users to set custom variables for the audit, including paths for the audit content, binary, and output files.

Parameters

The script supports the following parameters:

- **auditdir (default: *$DEFAULT_CONTENT_DIR*):**
  - Specifies the location where the audit content is stored (e.g., *C:\windows_audit*).
- **binpath (default: *$DEFAULT_AUDIT_BIN*):**
  - Defines the path to the audit binary (e.g., *C:\$DEFAULT_CONTENT_DIR\goss.exe*).
- **varsfile (default: *$DEFAULT_VARS_FILE*):**
  - Allows specifying a variable file containing settings for the audit.
- **group (default: *none*):**
  - Used to categorize the system into a specific group for comparison.
- **outfile (default: *$AUDIT_CONTENT_DIR\audit_$host_os_hostname_$host_epoch.json*):**
  - Defines the output file path for storing the full audit results.

Usage Examples

```
# Run the script with default settings
.\run_audit.ps1

# Specify a custom path for the audit binary
.\run_audit.ps1 -auditbin C:\path_to\binary.exe

# Define a custom audit directory
.\run_audit.ps1 -auditdir C:\somepath_for_audit_content

# Use a specific variables file
.\run_audit.ps1 -varsfile myvars.yml

# Set a custom output file path
.\run_audit.ps1 -outfile C:\audit\output.json

# Assign the system to a group
.\run_audit.ps1 -group webserver
```

Script Functionality

1. Define Default Values

The script sets default values for:

- The benchmark type (*CIS or STIG*).

- The Windows version (*Windows 20XX*).
- The default content directory, audit binary path, and variable file.

2. Validate File Paths
The script verifies the existence of essential files, such as the audit binary and content files. If any file is missing, it displays a warning and exits.

3. Identify Server Type

Using *wmic.exe*, the script determines the server role, which could be:

- Standalone Server
- Member Server
- Primary Domain Controller (PDC)
- Backup Domain Controller (BDC)
- Workstation

4. Collect System Metadata

The script gathers system information such as:

- Machine UUID
- OS Version & Locale
- Hostname
- Epoch time for timestamping output files

5. Run System Audit Commands

Depending on the server type, the script executes:

- *auditpol.exe* to capture audit policies.
- *secedit.exe* for security configuration exports (on standalone servers).
- *gpresult.exe* for Group Policy results (on domain-connected machines).

6. Generate JSON Metadata
The script constructs a JSON object containing system metadata for the audit.

7. Execute the Audit
The script runs the *goss* audit using the collected metadata, storing the results in the specified output file.

8. Output Summary

The script summarizes the audit results:

- If successful, it displays the last few lines of the audit report.
- If failed, it prompts the user to investigate.

Table 3: **PowerShell Script Summary**

Feature	Description
Purpose	Runs a Goss-based OS security audit per parameters
Supported Windows Versions	Standalone Server, Member Server, Primary Domain Controller
Collect System Metadata	OS Version, Hostname, Epoch time
Pre-checks	Verifies the existence of essential audit binary and content files
Error Handling	Alerts for missing files and vars

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
```

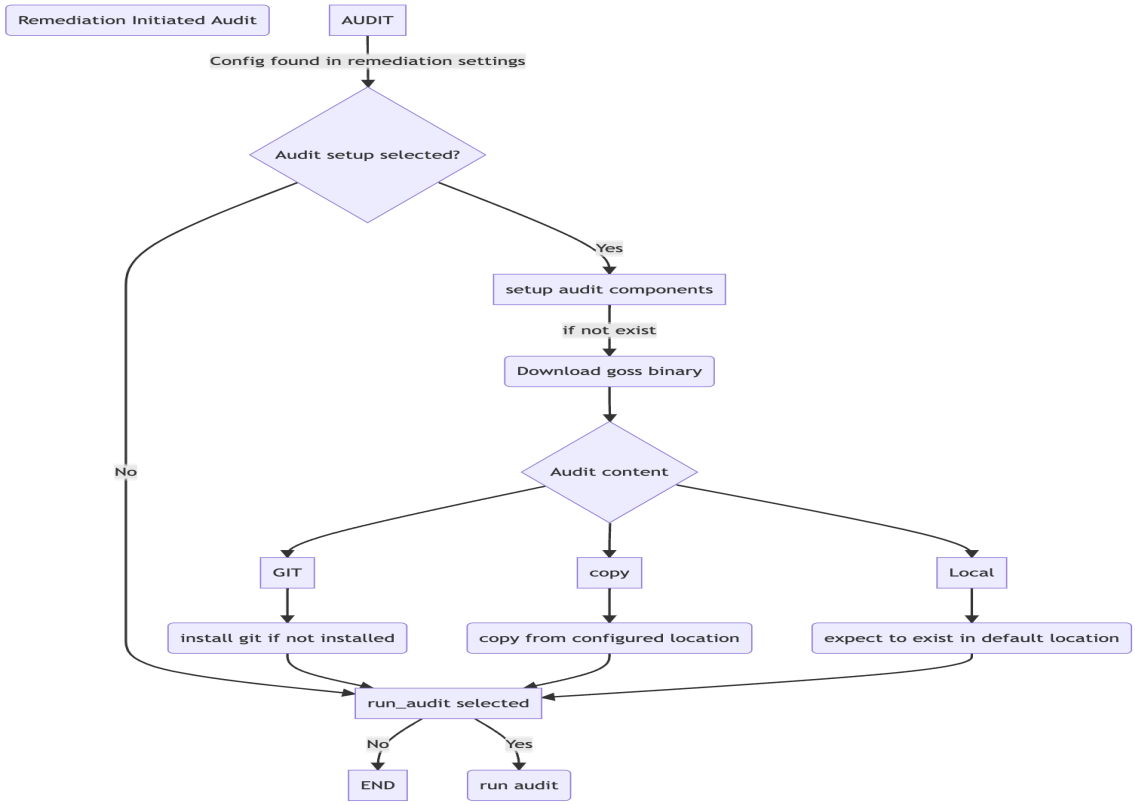
CHAPTER

FOUR

USING AUDIT AND REMEDIATE TOGETHER

Using both Audit and Remediate in a workflow

This is missing the copy remediate settings step



CHAPTER

FIVE

POST HARDENING LOCKDOWN REPORTING VIA ANSIBLE_FACTS

The *etc/ansible/compliance_facts.j2* template metadata and conditions related to hardening performed by the **Ansible Lockdown** role.

Lockdown Ansible_Facts: Creating Custom Compliance Facts

The playbook conditionally creates a custom facts file on managed hosts to document the applied security benchmark and hardening levels.

lockdown_role/tasks/main.yml

```
- name: Add ansible file showing Benchmark and levels applied
  when: create_benchmark_facts
  tags:
    - always
    - benchmark
  block:
    - name: Create ansible facts directory
      ansible.builtin.file:
        path: "{{ ansible_facts_path }}"
        state: directory
        owner: root
        group: root
        mode: 'u=rwx,go=rx'

    - name: Create ansible facts file
      ansible.builtin.template:
        src: etc/ansible/compliance_facts.j2
        dest: "{{ ansible_facts_path }}/compliance_facts.fact"
        owner: root
        group: root
        mode: "u-x,go-wx"
```

Key Components

- **Conditional Execution**: The entire block executes only if the variable `create_benchmark_facts` is set to `true`.
- **Tagging**: The tasks are tagged with `always` and `benchmark`, allowing for selective execution during playbook runs.
- **Directory Creation**: Ensures the existence of the directory specified by `ansible_facts_path` (typically `/etc/ansible/facts.d`), setting appropriate permissions.
- **Facts File Creation**: Uses a Jinja2 template to generate the `compliance_facts.fact` file in the specified directory.

Custom Facts in Role

Ansible allows the use of custom facts to store host-specific information. These facts are typically stored in files within the `/etc/ansible/facts.d` directory on the managed hosts. The facts files can be in JSON or INI format and are loaded automatically during the fact-gathering phase.

Accessing Custom Facts

Once the custom facts are in place and facts have been gathered, they can be accessed in playbooks using the `ansible_local` variable.

```
{{ ansible_local.compliance.benchmark_version }}
```

5.1 Lockdown Facts Example:

Variables Used

- `benchmark_version`: The version of the CIS/STIG benchmark being applied.
- **CIS** `cis_level_1 | cis_level_2`: Booleans that indicate if level 1 or 2 hardening is enabled.
- **STIG** `stig_cat1 | stig_cat2 | stig_cat3`: Indicate whether Category I, II, or III controls were enabled during the hardening process.
- `ansible_run_tags`: List of tags used during the playbook run to identify scope
- `run_audit`: Boolean to indicate if an audit was performed.
- `audit_log_dir`: Path to local audit log directory on the node.
- `post_audit_results`: Captured summary results from post-audit steps.
- `fetch_audit_output`: Boolean flag to indicate whether audit logs were centralized.
- `audit_output_destination`: Destination directory for centralized audit files.

5.1.1 CIS

1. **[lockdown_details]**

- Contains metadata about the CIS benchmark used, run date, and the hardening levels enabled.

```
[lockdown_details]
# Benchmark release
Benchmark_release = CIS-{{ benchmark_version }}
Benchmark_run_date = {{ '%Y-%m-%d - %H:%M:%S' | ansible.builtin.strftime }}

# Hardening levels enabled via variables
level_1_hardening_enabled = {{ rhel9cis_level_1 }}
level_2_hardening_enabled = {{ rhel9cis_level_2 }}

# Tag-based hardening run types (conditional)
{% if 'level1-server' in ansible_run_tags %}
Level_1_Server_tag_run = true
{% endif %}
{% if 'level2-server' in ansible_run_tags %}
Level_2_Server_tag_run = true
{% endif %}
{% if 'level1-workstation' in ansible_run_tags %}
```

(continues on next page)

(continued from previous page)

```
Level_1_workstation_tag_run = true
{% endif %}
{% if 'level2-workstation' in ansible_run_tags %}
Level_2_workstation_tag_run = true
{% endif %}
```

2. **[lockdown_audit_details]**

- Captures audit-specific information if auditing is enabled.

```
[lockdown_audit_details]

{% if run_audit %}
# Audit run
audit_run_date = {{ '%Y-%m-%d - %H:%M:%S' | ansible.builtin.strftime }}
audit_file_local_location = {{ audit_log_dir }}

{% if not audit_only %}
audit_summary = {{ post_audit_results }}
{% endif %}

{% if fetch_audit_output %}
audit_files_centralized_location = {{ audit_output_destination }}
{% endif %}
{% endif %}
```

3. **Output**

```
ansible hosts -i ../inv -m setup -a "filter=ansible_local"
hosts | SUCCESS => {
   "ansible_facts": {
      "ansible_local": {
            "lockdown_facts": {
               "Benchmark_Audit_Details": {
                  "audit_file_location_local": "/opt",
                  "audit_summary": "Count: 798, Failed: 24, Skipped: 6, Duration: 38.824s
↪"
               },
               "Benchmark_Details": {
                  "benchmark_release": "CIS-v2.0.0",
                  "benchmark_run_date": "2025-03-31 - 14:59:43",
                  "level_1_hardening_enabled": "True",
                  "level_2_hardening_enabled": "True"
               }
            }
      },
      "discovered_interpreter_python": "/usr/bin/python3"
   },
   "changed": false
}
```

5.1.2 STIG

1. **[lockdown_details]**

- Contains metadata about the STIG benchmark used, run date, and the hardening levels enabled.

```
[lockdown_details]
# Benchmark release
Benchmark_release = STIG-{{ benchmark_version }}
Benchmark_run_date = {{ '%Y-%m-%d - %H:%M:%S' | ansible.builtin.strftime }}

# If options set (doesn't mean it ran all controls)
cat_1_hardening_enabled = {{ rhel9stig_cat1 }}
cat_2_hardening_enabled = {{ rhel9stig_cat2 }}
cat_3_hardening_enabled = {{ rhel9stig_cat3 }}

# Tag-based hardening run types (conditional)
{% if ansible_run_tags | length > 0 %}
# If tags used to stipulate run level
{% if 'rhel9stig_cat1' in ansible_run_tags %}
Cat_1_Server_tag_run = true
{% endif %}
{% if 'rhel9stig_cat2' in ansible_run_tags %}
Cat_2_Server_tag_run = true
{% endif %}
{% if 'rhel9stig_cat3' in ansible_run_tags %}
Cat_3_Server_tag_run = true
{% endif %}
{% endif %}
```

2. **[lockdown_audit_details]**

- Captures audit-specific information if auditing is enabled.

```
[lockdown_audit_details]

{% if run_audit %}
# Audit run
audit_file_local_location = {{ audit_log_dir }}

{% if not audit_only %}
audit_summary = {{ post_audit_results }}
{% endif %}

{% if fetch_audit_output %}
audit_files_centralized_location = {{ audit_output_destination }}
{% endif %}
{% endif %}
```

3. **Output**

```
ansible hosts -i ../inv -m setup -a "filter=ansible_local"
hosts | SUCCESS => {
   "ansible_facts": {
      "ansible_local": {
```

(continues on next page)

(continued from previous page)


```
            "lockdown_facts": {
               "Benchmark_Audit_Details": {
                  "audit_file_location_local": "/opt",
                  "audit_summary = Count: 979, Failed: 73, Skipped: 22, Duration: 18.411s
↪"
               },
               "Benchmark_Details": {
                  "benchmark_release": "STIG-v2r2",
                  "benchmark_run_date": "2025-03-31 - 14:59:43",
                  "cat_1_hardening_enabled": "True",
                  "cat_2_hardening_enabled": "True",
                  "cat_3_hardening_enabled": "True",
               }
            }
      },
      "discovered_interpreter_python": "/usr/bin/python3"
   },
   "changed": false
}
```

CHAPTER

SIX

RELEASE SCHEDULE

6.1 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.

6.2 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.

6.3 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.

6.4 Branches

Refer to:

- Branches

6.4.1 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

SEVEN

CIS BENCHMARKS

7.1 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	True		
DEBIAN12-CIS	True	True	True		
RHEL8-CIS	True	True	True		
RHEL9-CIS	True	True	True		
RHEL10-CIS	True	True	True		*UNOFFICIAL*
SUSE15-CIS	True	True	True		
UBUNTU18-CIS	True	True	True		
UBUNTU20-CIS	True	True	True		
UBUNTU22-CIS	True	True	True		
UBUNTU24-CIS	True	True	True		

Table 2: CIS Windows Benchmark Availability

Benchmark	Maintained	Remediate	Audit	Release
Windows-10-CIS	True	True	NA	
Windows-11-CIS	True	True	NA	
Windows-2016-CIS	True	True	True	
Windows-2019-CIS	True	True	True	
Windows-2022-CIS	True	True	WIP	
Windows-2025-CIS	True	WIP	WIP	N/A

7.2 Cloud Platforms

Table 3: CIS Platform Benchmark Availability

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

7.3 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

7.4 Archived Roles

Table 5: CIS Retired Benchmark

Benchmark	Maintained	Remediate	Audit	Release
RHEL7-CIS	False	False	False	

CHAPTER

EIGHT

CIS SPECIFIC INFORMATION

8.1 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

NINE

STIG BENCHMARKS

9.1 Operating Systems

Table 1: STIG Linux Benchmark Availability

Benchmark	Maintained	Remediate	Audit	Release
RHEL8-STIG	True	True	True	
RHEL9-STIG	True	True	True	
UBUNTU18-STIG	True	True	True	
UBUNTU20-STIG	True	True	WIP	
UBUNTU22-STIG	True	True	WIP	N/A

Table 2: STIG Windows Benchmark Availability

Benchmark	Maintained	Remediate	Audit	Release
Windows-10-STIG	True	True	False	
Windows-11-STIG	True	True	False	
Windows-2016-STIG	True	True	False	
Windows-2019-STIG	True	True	False	
Windows-2022-STIG	True	True	False	
Windows-2025-STIG	True	WIP	WIP	N/A

9.2 Networking

Table 3: STIG Hardware Benchmark Availability

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

9.3 Applications

Table 4: 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	

9.4 Archived Roles

Table 5: STIG Retired Benchmark

Benchmark	Maintained	Remediate	Audit	Release
RHEL5-STIG	False	False	False	N/A
RHEL6-STIG	False	False	False	N/A
RHEL7-STIG	False	False	False	
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

TEN

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)**

10.1 How to contribute to audit development

10.1.1 Adding code

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

10.1.2 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)**.

10.1.3 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 }}                                               ##␣
↪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 }}
```

(continues on next page)

(continued from previous page)

```
    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.

```
{{ .Vars.section_1 }}
  {{ .Vars.rhelcis8_1_1_1_1 }}
```

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
```

10.1.4 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

10.2 How to Contribute to Remediate Development

10.2.1 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.

10.2.2 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.

- **Keep it clean**
  - Linting

We utilise both:

- ansible-lint
- yamllint

We also run pre-commit checks on all PRs to ensure this is met in requests.

General Layout

- Users should not have to edit tasks, but vars in `defaults/main.yml`. Should be used to override settings in the tasks. If a control has any variability create a variable for that value
- Follow the specific structure listed in the Layout sections
- For controls that install/uninstall packages please use package module and use ansible_facts.packages in the when where required
  - 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
- **Name** - `name:`
  - Each control gets it's own task and that task needs a name. The name format is category | STIG/CIS ID | PATCH or AUDIT | title of control copied from STIG/CIS.
  - 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
- **Mode**
  - To try and ensure idempotency and the changes to permissions should be X of more restrictive - we use symbolic logic e.g. `mode:  'u-x,go-wx'`

Variables

- `defaults/main.yml`
  - Any value that can deviate from the example used in the 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

- `vars/main.yml`
  - While these can be overridden, these will be rare and these are usually set as a default for a value.
- Dynamic Variables (register or set_facts)
  - These are labelled from where/how they are set, This will assist with:
    * reusability across repositories
    * assist in debugging knowing where the value is set

Naming of variables

- `tasks/prelim.yml`
  - These are often used in multiple locations so set in prelim
  - When set here as registered as prelim_description e.g. prelim_apparmor_enforce_status
- `tasks/*/*.yml`
  - Set by registering output and used within that task
  - descriptive naming e.g. discovered_auditd_conf_files
- **Parameters**
  - When using the `command` module to gather information please set `changed_when` and `failed_when` accordingly. 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 `command` over `shell`, except when using (|) in the command as this will cause issues.
  - 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
- **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.

STIG Control Task Layout

- **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
    * Nist values

```
- name: "MEDIUM | RHEL-08-010382 | PATCH | RHEL 8 must restrict privilege elevation to␣
↪authorized personnel."
  when:
    - rhel_08_010382
    - rhel8stig_disruption_high
  tags:
    - RHEL-08-010382
    - CAT2
    - CCI-000366
    - SRG-OS-000480-GPOS-00227
    - SV-237641r646893_rule
    - V-237641
    - NIST800-53R5_CM-7
    - sudo
  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: discovered_sudoers_all_privilege

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

(continues on next page)

(continued from previous page)

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

CIS Control Task Layout

- **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"
  when: rhel8cis_rule_4_1_1_3
  tags:
    - level2-server
    - level2-workstation
    - automated
    - patch
    - auditd
    - grub
    - rule_4.1.1.3
  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: discovered_default_grub_cmdline_linux

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

(continues on next page)

(continued from previous page)

```
      regexp: 'audit=.'
      replace: 'audit=1'
    notify: grub2cfg

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

CHAPTER

ELEVEN

GETTING SUPPORT

11.1 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 early release access, 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.

  target
  https://twitter.com/AnsibleLockdown

11.2 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.

11.2.1 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

11.2.2 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. Furthermore, the run_audit script shows how these variables are created and passed to Goss.

Why do I have different results between x86_64 and AMD64/aarch64 audits?

The two different hardware architectures provide distinct system calls within the OS that auditd can utilize. This is often the source of increased failures compared to x86_64, as they are unable to execute all commands.

My system is impacted when running the audit. How can I restrict its effect?

On both Windows and Linux, you have the ability to limit the number of processes that run at the same time.

This is set using a variable as part of the playbook.

```
audit_max_concurrent
```

Or if running manually using the run_audit script.

```
-m #
```

It is also possible on Linux to change the priority of a process by using nice

- set process priorities

11.2.3 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
- NOTE This Should no longer be an issue with new releases as dependency has been removed

Missing Sudo Password (Linux OS Based)

Many of the CIS Linux OS based 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

TWELVE

GLOSSARY

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

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

CHAPTER

THIRTEEN

USEFUL LINKS

13.1 Support

- Commercial

  Providing:

  - Early Releases
  - Priority big fixes
  - Ticketed support
  - Lockdown Enterprise
  - engage@mindpointgroup.com : Email Enquiry
- Community
  - Discord Invite Link

13.2 Main Content site

- Repositories

13.3 Audit

- Goss Main Site
- Goss Docs

13.4 Remediate

- Ansible Main Site
- Ansible Get Started
- Ansible Docs

13.4.1 Linting

- ansible-lint
- yamllint
- pre-commit