Scanning Your Ansible Playbooks

In this chapter, you will learn how to scan your Ansible playbooks using two third-party tools: Checkov and KICS. Both are open source and can help you identify and fix common configuration issues within your Ansible code, such as syntax errors, misconfigurations, hardcoded secrets, and deployment problems, which could lead to potential breaches.

By the end of this chapter, you will have done the following:

* Installed and run Checkov and KICS scans on our Ansible playbooks
* Reviewed the results and reports generated during the scans
* Fixed any issues detected during the scans

The chapter covers the following topics:

* Why scan your playbooks?
* Docker overview and installation
* Exploring Checkov
* Exploring KICS

Technical requirements

Rather than installing the tools locally, we will use Docker to execute the scans; there will be a little detail on how to install Docker later in the chapter. Additionally, we will be scanning a variation of the playbook we wrote in [*Chapter 11*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/11), *Highly Available Cloud Deployments*; this can be found in the repository at https://github.com/PacktPublishing/Learn-Ansible-Second-Edition/tree/main/Chapter13.

Why scan your playbooks?

While we have been taking a sensible approach to deploying our cloud resources in previous chapters, many of the guardrails we have put in place have all been ones I have learned through experience and by applying a little common sense.

For example, when launching a virtual machine resource in either Microsoft Azure or Amazon Web Services, we have been locking down the SSH or RDP service to the host’s public IP address, which is running Ansible; up until now, this has been your local machine rather than just opening SSH or RDP to the world by using 0.0.0.0/0 as the source address, which is the CIDR notation for *“allow all.”*

This is not a problem for the workloads we have been working on; having a virtual machine exposed directly to the internet with its management port open for everyone to access is not considered best practice, as it will expose you to brute-force attacks, which, if they are successful, will not only lead to that machine being compromised; it could also act as a gateway to the rest of your network and other associated resources such as databases and storage.

I would class the preceding example as common sense, but as we launch more and more cloud services using our playbooks, how can we ensure that we are following best practices for services that maybe we haven’t had much experience with outside of getting something up and running? How can we put some guardrails in place to stop us from doing something before resources are deployed?

This is where the two tools we will look at in this chapter come in; they are designed to scan your playbooks, look at the configuration, and compare them to their best practice policies. Eventually, in [*Chapter 15*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/15), *Using Ansible with GitHub Actions and Azure DevOps*, we will build one of the two tools into our deployment pipelines, but for now, we are going to look at the tools and run them locally using Docker.

Docker overview and installation

Docker, the platform that made containers popular, is both an open source and commercial solution that enables you to package all of the elements of your application, including libraries and other dependencies, alongside your own code in a single, easy-to-distribute package; this means that we won’t need to download and install all of the prerequisites for the tools that we will be running in this chapter or need to compile the tools from source to get working executables for our system.

To follow the example in this chapter, you must install **Docker Desktop** on your host.

Installing Docker Desktop on macOS

To install Docker Desktop on macOS, follow these three steps:

1. Choose the appropriate installer for your Mac’s architecture:
   1. For ARM64 (Apple Silicon), use https://desktop.docker.com/mac/main/arm64/Docker.dmg.
   2. For AMD64 (Intel Macs), use https://desktop.docker.com/mac/main/amd64/Docker.dmg.
2. After downloading, open the Docker.dmg file by double-clicking it. In the opened window, drag the Docker icon to your Applications folder to install Docker Desktop. It will be installed at /Applications/Docker.app.
3. To launch Docker, navigate to the Applications folder and double-click on **Docker**; this will start **Docker Desktop**.

When you first launch Docker Desktop, it will walk you through the remaining installation steps and run in the background once complete.

Installing Docker Desktop on Windows

To install Docker Desktop on Windows, follow these instructions:

1. Download the Docker Desktop Installer for Windows from this link: https://desktop.docker.com/win/main/amd64/Docker%20Desktop%20Installer.exe.
2. Run the downloaded **Docker Desktop Installer.exe** by double-clicking on it. Docker Desktop will be installed at the default location C:\Program Files\Docker\Docker.
3. During the installation, you may be prompted to choose whether to use **WSL 2 (Windows Subsystem for Linux 2)** or **Hyper-V** as the backend. Select the **Use WSL 2 instead of Hyper-V** option, as we have used this throughout the book to run Ansible.
4. Follow the on-screen instructions provided by the installation wizard to authorize the installer and complete the installation process.
5. Once the installation is completed, click **Close** to finish the setup.

From here, you can open Docker Desktop from the start menu, and it will run in the background.

Installing Docker Desktop on Linux

If you are running a Linux Desktop, the instructions will differ slightly depending on your Linux distribution; for detailed instructions, see https://docs.docker.com/desktop/linux/install/.

Now, with Docker Desktop installed, we can look at the first of the two tools we will look at.

Exploring Checkov

Checkov is an open source static code analysis tool maintained by Prisma Cloud designed for **infrastructure-as-code** (**IaC**).

It helps developers and DevOps teams identify misconfigurations in their files before deployment to cloud environments. By scanning the code for tools such as Terraform, CloudFormation, Kubernetes, and others, including Ansible, Checkov checks against best practices and compliance guidelines, ensuring your infrastructure deployments are secure, efficient, and compliant with industry standards before it is deployed.

Important note

You may have noticed that Ansible is mentioned as “others” in the preceding description; that is because support for Ansible was only just introduced at the time of writing this in early 2024. Because of this, while we will be looking at Checkov during this chapter, we will not be going into as much detail about Checkov or the second tool, Kics.

Before we run our scan, we need a playbook; open your terminal and check out the scan GitHub repository by running the following:

$ git clone git@github.com:PacktPublishing/Learn-Ansible-Second-Edition.git

CopyExplain

This repository contains a copy of the final playbook code from [*Chapter 11*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/11), *Highly Available Cloud Deployments*.

Now that we have the code checked out, we can download the Checkov container image. To do this, we need to pull it from Docker Hub by running the following command:

$ docker image pull bridgecrew/checkov:latest

CopyExplain

This will download the image from https://hub.docker.com/r/bridgecrew/checkov, and with it downloaded, we can now scan our playbook code.

To run the scan, issue the following commands:

$ cd Learn-Ansible-Second-Edition/Chapter13

$ docker container run --rm --tty --volume ./:/ansible --workdir /ansible bridgecrew/checkov --directory /ansible

CopyExplain

Before we review the results, let’s quickly break down the command that we have just run:

* docker container run executes a new Docker container.
* --rm instructs Docker to remove the container after it exits automatically.
* --tty allocates a pseudo-TTY, which makes the scan output readable to our session.
* --volume ./:/ansible mounts the current directory, defined as ./, to the /ansible path inside the container.
* --workdir /ansible sets the working directory inside the container to /ansible.
* bridgecrew/checkov specifies the Checkov Docker image we have just pulled from the Docker Hub.
* --directory /ansible instructs Checkov to scan files in /ansible; it is not part of the Docker command but is sending instructions to the Checkov binary, which is the default entry point for our container to run the scan. If we had Checkov installed locally, then this would be the equivalent to running the checkov --directory /ansible command.

Now that we have broken down the command used to run the scan, we can look at the output of the scan itself, starting with the overview:

ansible scan results:

Passed checks: 5, Failed checks: 3, Skipped checks: 0

CopyExplain

As you can see, we have more passes than failed checks, which is a good start; the next section of the output details the checks, starting with the following passes:

Check: CKV\_ANSIBLE\_2: "Ensure that certificate validation isn't disabled with get\_url"

     PASSED for resource: tasks.ansible.builtin.get\_url.download wp-cli

     File: /roles/wordpress/tasks/main.yml:11-17

CopyExplain

Our first pass checks to see if we are instructing the ansible.builtin.get\_url module to bypass certificate validation when connecting to an HTTPS site to download content.

The next four passes are for the two times our playbook uses the ansible.builtin.apt module:

Check: CKV\_ANSIBLE\_5: "Ensure that packages with untrusted or missing signatures are not used"

     PASSED for resource: tasks.ansible.builtin.apt.update apt-cache and upgrade packages

     File: /roles/stack-install/tasks/main.yml:5-13

Check: CKV\_ANSIBLE\_5: "Ensure that packages with untrusted or missing signatures are not used"

     PASSED for resource: tasks.ansible.builtin.apt.update cache and install the stack packages

     File: /roles/stack-install/tasks/main.yml:27-33

CopyExplain

The first pair of passes ensures that we are not installing any packages that are not correctly signed. The second pair of passes also checks for the same thing:

Check: CKV\_ANSIBLE\_6: "Ensure that the force parameter is not used, as it disables signature validation and allows packages to be downgraded which can leave the system in a broken or inconsistent state"

     PASSED for resource: tasks.ansible.builtin.apt.update apt-cache and upgrade packages

     File: /roles/stack-install/tasks/main.yml:5-13

Check: CKV\_ANSIBLE\_6: "Ensure that the force parameter is not used, as it disables signature validation and allows packages to be downgraded which can leave the system in a broken or inconsistent state"

     PASSED for resource: tasks.ansible.builtin.apt.update cache and install the stack packages

     File: /roles/stack-install/tasks/main.yml:27-33

CopyExplain

However, this time, the check ensures that we are not using the force parameter, which, as you can see from the description, disables signature checks and can also leave our APT database in a little bit of a state if things go wrong.

Now, we move on to the failures; the first failure is the one we called out as the example when we spoke about why you would want to use the tools we are covering in this chapter:

Check: CKV\_AWS\_88: "EC2 instance should not have public IP."

     FAILED for resource: tasks.amazon.aws.ec2\_instance.Create the temporary ec2 instance

     File: /roles/ec2tmp/tasks/main.yml:53-75

      Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/public-policies/public-12

           62 |     network:

           63 |       assign\_public\_ip: "{{ ec2.public\_ip }}"

CopyExplain

So, what gives? As you may recall from [*Chapter 11*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/11), *Highly Available Cloud Deployments*, the instance we are launching is only temporary and accessible while the playbook is running. However, Checkov doesn’t know this, so it rightly calls it out and, as you can see, provides details on why this is via the guide URL, which, for this check, is https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/public-policies/public-12.

Moving on to the next failure in the scan, we see the following:

Check: CKV\_AWS\_135: "Ensure that EC2 is EBS optimized"

     FAILED for resource: tasks.amazon.aws.ec2\_instance.Create the temporary ec2 instance

     File: /roles/ec2tmp/tasks/main.yml:53-75

     Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-general-policies/ensure-that-ec2-is-ebs-optimized

CopyExplain

In this case, Checkov believes a parameter is missing from the amazon.aws.ec2\_instance block when we launch the temporary EC2 instance. It is recommended that the parameter ebs\_optimized is set to true rather than keeping the value as false, which is the default for the parameter.

The final failure in the scan output is as follows:

Check: CKV2\_ANSIBLE\_2: "Ensure that HTTPS url is used with get\_url"

     FAILED for resource: tasks.ansible.builtin.get\_url.download wp-cli

    File: /roles/wordpress/tasks/main.yml:11-17

           11 | - name: "download wp-cli"

           12 |   ansible.builtin.get\_url:

           13 |     url: "{{ wp\_cli.download }}"

           14 |     dest: "{{ wp\_cli.path }}"

CopyExplain

As Checkov is doing static code analysis, it isn’t designed to check for the contents of variables. Because the policy checks that we are providing a secure URL (that is, https://domain.com/ in the url section of the task) it fails, as it is just seeing the {{ wp\_cli.download }} variable name rather than the contents of the variable.

If you are keeping count, that makes two of the three failed checks false positives; for the first failure, we can accept the risk, as we know the machine is only temporary, and we know that we are locking the EC2 instance down to trusted IP addresses.

For the third failure, we can confirm that the contents of the {{ wp\_cli.download }} variable is a secure URL, as it is https://raw.githubusercontent.com/wp-cli/builds/gh-pages/phar/wp-cli.phar.

The second failure is the only one we need to look at; let’s take a look at the tasks, starting with the Amazon.aws.ec2\_instance one.

Here, we need to add two things; the first thing is a comment to instruct Checkov that we accept the risk being highlighted by the CKV\_AWS\_88 policy, and then we need to set ebs\_optimized to true.

The following code shows the updates I have made to roles/ec2tmp/tasks/main.yml; everything below the name parameter remains as is:

- name: "Create the temporary ec2 instance"

  amazon.aws.ec2\_instance:

    # checkov:skip=CKV\_AWS\_88:"While a public IP address is assigned to the instance, it is locked down by the security group and the instance is temporary."

    ebs\_optimized: true

    name: "{{ ec2\_tmp\_name }}"

CopyExplain

As you can see, instructing Checkov to skip a check is straightforward; the comment is split into four parts:

* # is the standard syntax for starting a comment in a YAML file
* checkov: instructs Checkov to pay attention to the contents of the comment
* skip=CKV\_AWS\_88: instructs Checkov to skip the CKV\_AWS\_88 check when it runs
* "While a public IP address is assigned to the instance, it is locked down by the security group and the instance is temporary." is the suppress comment that will appear in the output when we run the scan

The next line in the update task implements the recommendation that we set the ebs\_optimized parameter to true.

Now, we move on to the second task, which we need to update, and can be found in roles/wordpress/tasks/main.yml. Here, we just add a comment to make Checkov skip CKV2\_ANSIBLE\_2:

- name: "download wp-cli"

  ansible.builtin.get\_url:

    # checkov:skip=CKV2\_ANSIBLE\_2:"The URL passed in the variable is secured with SSL/TLS protocol."

    url: "{{ wp\_cli.download }}"

    dest: "{{ wp\_cli.path }}"

CopyExplain

If you are following along, the repository contains a branch called checkov; with the preceding detailed changes applied, you can switch to it by running the following:

$ git switch chapter13-checkov

CopyExplain

Then, we can re-run the scan using the following command:

$ docker container run --rm --tty --volume ./:/ansible --workdir /ansible bridgecrew/checkov --directory /ansible

CopyExplain

I can see that my changes have both suppressed and resolved the three failures:

ansible scan results:

Passed checks: 6, Failed checks: 0, Skipped checks: 2

CopyExplain

We have the pass for CKV\_AWS\_135:

Check: CKV\_AWS\_135: "Ensure that EC2 is EBS optimized"

     PASSED for resource: tasks.amazon.aws.ec2\_instance.Create the temporary ec2 instance

      File: /roles/ec2tmp/tasks/main.yml:53-77

      Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/aws-general-policies/ensure-that-ec2-is-ebs-optimized

CopyExplain

We also have the two false positives now showing:

Check: CKV\_AWS\_88: "EC2 instance should not have public IP."

     SKIPPED for resource: tasks.amazon.aws.ec2\_instance.Create the temporary ec2 instance

     Suppress comment: "While a public IP address is assigned to the instance, it is locked down by the security group and the instance is temporary."

     File: /roles/ec2tmp/tasks/main.yml:53-77

     Guide: https://docs.prismacloud.io/en/enterprise-edition/policy-reference/aws-policies/public-policies/public-12

CopyExplain

For the second one, we have the following:

Check: CKV2\_ANSIBLE\_2: "Ensure that HTTPS url is used with get\_url"

     SKIPPED for resource: tasks.ansible.builtin.get\_url.download wp-cli

     Suppress comment: "The URL passed in the variable is secured with SSL/TLS protocol."

     File: /roles/wordpress/tasks/main.yml:11-18

CopyExplain

As you can see, our comments are visible for all to see.

So, returning to the call-out at the start of the section, why have we covered this tool if Checkov doesn’t have full coverage for Ansible? As you can see from the output of the scan of our playbook, while there is not much coverage now, each new release brings additional Ansible policies. Hence, as time goes on, coverage should only get more robust, and hopefully, we will bring this promising tool in line with the second tool we will look at: **KICS**, or to give it its full title, **keeping infrastructure as code secure**.

Exploring KICS

KICS is another static code analysis tool, and like Checkov, it is open source. It is designed to help you find common misconfiguration issues, potential compliance issues, and even security vulnerabilities within your IaC code. It ships with support for Kubernetes, Docker, AWS CloudFormation, Terram, and, of course, Ansible, which we will be focusing on in this chapter.

KICS is designed to be easy to install, understand, and integrate into CI/CD pipelines. It includes over 2,400 customizable rules and is built for extensibility, allowing for the easy addition of support for new IaC tools and updates to existing integrations.

KICS is maintained and supported by **Checkmarx** specialists in software application security testing, meaning that KICS has a good pedigree.

Running the scan

Let’s dive straight in. If you haven’t already, check out the example repository using the following command:

$ git clone https://github.com/russmckendrick/Learn-Ansible-Second-Edition-Scan.git

CopyExplain

Now, we can pull the latest container image from Docker Hub (https://hub.docker.com/r/checkmarx/kics) by using the following command:

$ docker image pull checkmarx/kics:latest

CopyExplain

Change to the folder containing our Ansible playbook:

$ cd Learn-Ansible-Second-Edition-Scan

CopyExplain

Then run the scan itself:

$ docker container run --rm --tty --volume ./:/ansible checkmarx/kics scan --path /ansible/

CopyExplain

As you can see, the docker command follows the same pattern we discussed when we ran Checkov up until where we pass the options to the KICS binary; here, we instruct KICS to run scan against the --path /ansible/, which is the directory we have mounting from our host machine inside the container using the --volume option.

Reviewing the results

Now, let’s take a look at the result of the scan; KICS presents its output, which is slightly different from Checkov in that the initial output is designed to give real-time information on the scan itself:

Scanning with Keeping Infrastructure as Code Secure v1.7.12

Preparing Scan Assets: Done

Executing queries: [------------------------------] 100.00%

Files scanned: 33

Parsed files: 32

Queries loaded: 292

Queries failed to execute: 0

CopyExplain

Let’s now work through the various results and group them by severity levels.

Info and low-severity results

The first result highlights potentially risky file permissions for the files we create (using templates) or copy:

Risky File Permissions, Severity: INFO, Results: 5

Description: Some modules could end up creating new files on disk with permissions that might be too open or unpredictable

Platform: Ansible

Learn more about this vulnerability: https://docs.kics.io/latest/queries/ansible-queries/common/88841d5c-d22d-4b7e-a6a0-89ca50e44b9f

CopyExplain

It then goes on to list all the affected files; here is a snippet of the first few:

[1]: ../../ansible/destroy.yml:52

     051:     - name: "Delete the group\_vars/generated\_aws\_endpoints.yml file"

     052:       ansible.builtin.file:

     053:         path: "group\_vars/generated\_aws\_endpoints.yml"

CopyExplain

Here is another:

[2]: ../../ansible/destroy.yml:81

     080:     - name: "Delete the group\_vars/generated\_efs\_targets.yml file"

     081:       ansible.builtin.file:

     082:         path: "group\_vars/generated\_efs\_targets.yml"

CopyExplain

Moving on to the next issue, we see the following:

Unpinned Package Version, Severity: LOW, Results: 1

Description: Setting state to latest performs an update and installs additional packages possibly resulting in performance degradation or loss of service

Platform: Ansible

Learn more about this vulnerability: https://docs.kics.io/latest/queries/ansible-queries/common/c05e2c20-0a2c-4686-b1f8-5f0a5612d4e8

CopyExplain

Again, here is a sample of where it has spotted the issue:

[1]: ../../ansible/roles/stack\_install/tasks/main.yml:8

      007:     name: "\*"

      008:     state: "latest"

      009:     update\_cache: true

CopyExplain

The next and final low-scoring result is as follows:

EFS Without Tags, Severity: LOW, Results: 1

Description: Amazon Elastic Filesystem should have filesystem tags associated

Platform: Ansible

Learn more about this vulnerability: https://docs.kics.io/latest/queries/ansible-queries/aws/b8a9852c-9943-4973-b8d5-77dae9352851

CopyExplain

Here are the details:

[1]: ../../ansible/destroy.yml:75

      074:     - name: "Delete the EFS File System"

      075:       community.aws.efs:

      076:         name: "{{ efs\_name }}"

CopyExplain

Let’s quickly review the low-scoring ones before moving on to the one result with a medium score.

So, the first result was, **“Some modules could end up creating new files on disk with permissions that might be too open or unpredictable”**. It called out 11 places within our playbook where this could be an issue, so we should look at resolving these.

First off, if you ran the full scan, you will have noticed that three of the results are from the destroy.yml file.

Given that these tasks are removing files, we don’t care about the file permissions here. So, rather than adding the permissions to the individual tasks, we should instruct KICS not to run the check across the whole file.

To do this, we need to add the following comment at the very top of the file:

# kics-scan disable=88841d5c-d22d-4b7e-a6a0-89ca50e44b9f

CopyExplain

Next, we have ansible.builtin.template in roles/efs/tasks/main.yml. Rather than skip the test, I added the permissions using the mode key:

- name: "Generate the efs targets vars file"

  ansible.builtin.template:

    src: "targets.j2"

    dest: "group\_vars/generated\_efs\_targets.yml"

    mode: "0644"

CopyExplain

The final result is for the ansible.builtin.get\_url module used by the task, which downloads wp-cli in the roles/wordpress/tasks/main.yml file.

When reviewing the code, it looked like the following:

- name: "Download wp-cli"

  ansible.builtin.get\_url:

    url: "{{ wp\_cli.download }}"

    dest: "{{ wp\_cli.path }}"

CopyExplain

This was immediately followed by the following:

- name: "Update permissions of wp-cli to allow anyone to execute it"

  ansible.builtin.file:

    path: "{{ wp\_cli.path }}"

    mode: "0755"

CopyExplain

Here, KICS highlights that we can set the mode as part of ansible.builtin.get\_url, which means we do not have to do it separately, so I added the following to the download task:

    mode: "0755"

CopyExplain

Then, I removed the second task. That clears the file permission issues KICS reported.

The next LOW score says, **“Setting state to latest performs an update and installs additional packages possibly resulting in performance degradation or loss of service”**.

This appears in roles/stack-install/tasks/main.yml, where the task uses ansible.builtin.apt to update the installed images, as this task is only called when we bootstrap our temporary EC2 instance and we made allowances for changes to the PHP version in the main playbook. I think it is safe to accept this as a false positive, so we can tell KICS not to run the test on this file by adding the following to the very top of roles/stack-install/tasks/main.yml:

# kics-scan disable=c05e2c20-0a2c-4686-b1f8-5f0a5612d4e8

CopyExplain

This leaves us with **“Amazon Elastic Filesystem should have filesystem tags associated”**; the task it is complaining about is in destroy.yml, so the lack of tags does matter.

Let’s exclude the check from being run. To do this, we need to append it to the end of the comment we already added, which means the comment at the end of destroy.yml now reads this:

# kics-scan disable=88841d5c-d22d-4b7e-a6a0-89ca50e44b9f,b8a9852c-9943-4973-b8d5-77dae9352851

CopyExplain

When appending IDs, please ensure that a comma separates them; otherwise, KICS will read them as one string. Finally, we have the high-severity results.

High-severity results

Luckily, here we have just two problems called out across four tasks, starting with the following:

EFS Not Encrypted, Severity: HIGH, Results: 2

Description: Elastic File System (EFS) must be encrypted

Platform: Ansible

Learn more about this vulnerability: https://docs.kics.io/latest/queries/ansible-queries/aws/727c4fd4-d604-4df6-a179-7713d3c85e20

CopyExplain

These are the two tasks:

[1]: ../../ansible/roles/efs/tasks/main.yml:25

     024: - name: "Create the EFS File System"

     025:   community.aws.efs:

CopyExplain

The second is in the destroy.yml file:

[2]: ../../ansible/destroy.yml:77

     076:     - name: "Delete the EFS File System"

     077:       community.aws.efs:

CopyExplain

I think you can probably guess how we are going to resolve the second one; let’s get it to ignore the test in destroy.yml:

# kics-scan disable=88841d5c-d22d-4b7e-a6a0-89ca50e44b9f,b8a9852c-9943-4973-b8d5-77dae9352851,050f085f-a8db-4072-9010-2cca235cc02f,727c4fd4-d604-4df6-a179-7713d3c85e20

CopyExplain

For roles/efs/tasks/main.yml, the recommendation is to enable encryption, so let’s take that advice:

- name: "Create the EFS File System"

  community.aws.efs:

    encrypt: true

    name: "{{ efs\_name }}"

CopyExplain

As you can see from the preceding snippet, we have added the encrypt parameter and set it to true.

The next issue highlighted by KICS also has to do with EFS filesystem encryption:

EFS Without KMS, Severity: HIGH, Results: 2

Description: Amazon Elastic Filesystem should have filesystem encryption enabled using KMS CMK customer-managed keys instead of AWS managed-keys

Platform: Ansible

Learn more about this vulnerability: https://docs.kics.io/latest/queries/ansible-queries/aws/bd77554e-f138-40c5-91b2-2a09f878608e

CopyExplain

The results are for the same files as the previous issue, so we will append the ID to the list of checks to disable at the top of the destroy.yml file.

Given that this is just a demo environment, I am happy to accept the potential risk of not using a customer-managed key vault to store my own managed encryption keys; so, in this instance, I will add the following:

# kics-scan disable=bd77554e-f138-40c5-91b2-2a09f878608e

CopyExplain

I’ll do so at the very top of the roles/efs/tasks/main.yml file. If this were a fixed production environment, then I would have added a role to launch and maintain AWS Key Management Service (https://aws.amazon.com/kms/) as part of the deployment.

The results summary

The final part of the rules gives an overview of everything we have covered, which, for the initial scan with none of the fixes in place, is as follows:

Results Summary:

HIGH: 4

MEDIUM: 0

LOW: 2

INFO: 5

TOTAL: 11

CopyExplain

Re-running the scan

As before, there is a branch containing all of the updated files we discussed and implemented in the previous section; to change to it, run the following:

$ git switch chapter13-kics

CopyExplain

You can then run the scan again using this:

$ docker container run --rm --tty --volume ./:/ansible checkmarx/kics scan --path /ansible/

CopyExplain

This should now return a clean bill of health:

Scanning with Keeping Infrastructure as Code Secure v1.7.13

Preparing Scan Assets: Done

Executing queries: [------------------------------] 100.00%

Results Summary:

HIGH: 0

MEDIUM: 0

LOW: 0

INFO: 0

TOTAL: 0

CopyExplain

As you can see, no problems are being reported now.

Output files

Before we finish the chapter, there is one more thing that we should quickly discuss about KICS: its ability to output a report in various file formats.

If you were to re-run the scans against the main and kics branches but using the following command, then you will notice that a file called results.html appears in your repo folder:

$ docker container run --rm --tty --volume ./:/ansible checkmarx/kics scan --path /ansible/ --report-formats "html" --output-path /ansible/

CopyExplain

As you can see, we are passing in two new flags; the first, --report-formats, tells KICS to output a report as an html file, and the second, --output-path, lets KICS know where to save the report file; in our case, as we are running KICS in a container that needs to be a location within the container that persists, once the container has finished running, the container will automatically be removed along with any files written.

When running the command against the main branch, which does not contain any of the fixes, we applied the header of the report, which looks like the following:

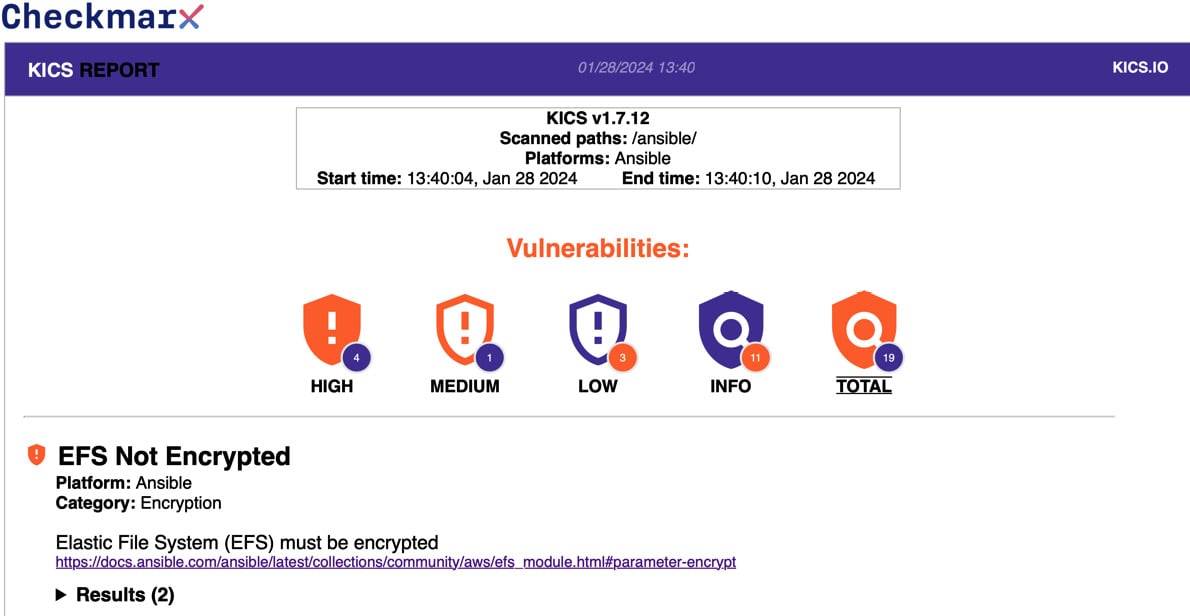


Figure 13.1 – Viewing the report showing issues

Then, re-running the scan against the KICS branch updates it to the following:

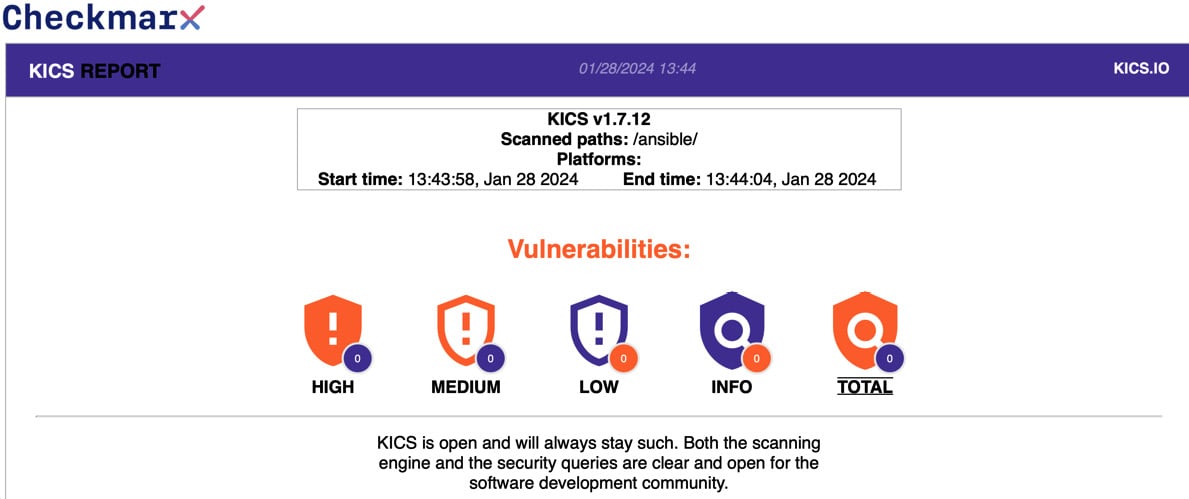


Figure 13.2 – A clean bill of health

You can also output in PDF, JSON, and other standard reporting formats. As you can see, this is a little more digestible than reading the output of the command line report we covered in the previous section.

We will put these reports to good use when we get to [*Chapter 15*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/15), *Using Ansible with GitHub Actions and Azure DevOps*, as we will publish the results as part of our pipeline runs.

Summary

In this chapter, we have covered two tools that we can add to our workflows, and we manually ran scans against the playbook we developed in [*Chapter 11*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/11), *Highly Available Cloud Deployments*. As mentioned in the chapter, Checkov's support for Ansible is relatively new, so it has a different coverage to KICS. However, I am sure you agree that both tools worked well.

Important note

There is one elephant in the room, though; even without the same coverage level, both tools came up with slightly different results, so you should never rely on them 100% to fully secure your deployments. Think of them as trusted colleagues reviewing your code for anything obvious that stands out as being an issue rather than a security-focused cloud platform architect with a working knowledge of your workload who dictates precisely what measures you should take when deploying your infrastructure in a secure way fully.

As already mentioned at the end of the previous section, we will be revisiting KICS in [*Chapter 15*](https://subscription.packtpub.com/book/cloud-and-networking/9781835088913/15), *Using Ansible with GitHub Actions and Azure DevOps*. Before we get there, now that we have looked at how we can review and secure our playbook code, we can now look at how we can secure our workload by quickly applying security best practices to the host operating systems that we are targeting using Ansible.