10

Onboarding and Managing Linux Servers

In this chapter, you will learn about extending **Microsoft Defender for Endpoint** (**MDE**) to Linux, the most popular server OS type. You are indeed reading this correctly: you can now get antimalware protection from *Microsoft* for *Linux* servers. You've already explored how to onboard other desktop and server OSs to MDE, so by extending this to your Linux server estate, you'll reap the benefits of a central **endpoint detection and response** (**EDR**) and antimalware system to defend against threats and respond to incidents.

The process for Linux servers is similar to macOS but has differences you'll need to be aware of due to the nature of how Linux server distributions operate and are managed. Therefore, in this chapter, to help you master onboarding and managing Linux servers, we'll explore the following:

- Options available for onboarding
- How to customize the protection settings for your Linux servers, including controlling the levels of protection you want, exclusions, schedules, and other important settings

IMPORTANT NOTE

To be clear, we are referring to x64 and x86_64 servers only: client Linux distributions (Linux on the desktop) are not supported. The server distribution must be on the supported distribution list:

- *Amazon Linux 2*
- *CentOS 6.7+ or 7.2+*
- *Debian 9+*
- *Fedora 33+*
- *Oracle Linux 7.2+ or 8+*
- **Red Hat Enterprise Linux** *(***RHEL***) 6.7+, 7.2+, or 8+*
- **SUSE Linux Enterprise Server** *(***SLES***) 12+*
- *Ubuntu 16.04+ LTS*

There is no graphical interface for the Linux version of MDE; you are purely working with commands and/or orchestration tools.

You now know whether your Linux distribution is supported, so let's kick off your deployment by exploring how you onboard the server.

Onboarding Linux

If your servers are in Azure or managed with Azure Arc, Microsoft Defender for Cloud is an onboarding option, just as with Windows Server. When you onboard devices using Microsoft Defender for Cloud, it is branded as Microsoft Defender for Servers. Deployment of the client and onboarding to your specific MDE instance is configured automatically this way.

Configuration management tools such as Ansible, Chef, and Puppet can be used to deploy and onboard but are not mandatory: an onboarding script is also available. In this chapter, we will focus on using the script and Microsoft Defender for Cloud.

Script onboarding

The process for onboarding MDE on Linux differs by Linux variation. We can group them into three groups:

- Amazon Linux 2, CentOS, Fedora, and Oracle Linux all follow the RHEL path
- Ubuntu and Debian Linux use the same process
- SLES 12+ sits on its own

Among other things, the commands executed by the shell differ due to the different tools available. RHEL and its variants will utilize **`yum`** for installation; Ubuntu and Debian's version of this is **`apt-get`**; while SLES uses its **`zypper`** command.

What all methods have in common is the general process: configuring software repositories for a **channel**, then installing the application, and using a Python script to onboard.

The channels available are **`insiders-fast`**, **`insiders-slow`**, and **`prod`**. These correspond to update cadence. If you are averse to preview features, stick to **`prod`**. It is probably worth running **`insiders-fast`** on at least some of your estate so that you can play with new toys as they become available.

The good news is that to simplify manual onboarding, Microsoft maintains a Bash script: **`mde_install.sh`**. You can obtain this at **github.com/microsoft/mdatp-xplat/blob/master/linux/installation/mde_installer.sh**.

Compatible with all three variants described previously, the script identifies which distribution of Linux you are onboarding and performs the appropriate commands. This is much simpler, and, therefore, less risky than executing the many steps individually yourself. It is this book's recommended path that any manual installations be completed using the script.

DOWNLOADING SCRIPTS FROM THE INTERNET

Always be cautious when executing scripts downloaded from the internet. Verify that you have downloaded them from the correct source and taken *measures to confirm that the source has not been compromised or otherwise invalidated since the original address was published.*

Here's how to get started with `mde_install.sh` and onboarding devices with it:

1. Visit the Microsoft 365 Defender portal and navigate to **Settings** | **Endpoints** | **Onboarding**.
2. Select **Linux Server** as the OS to start onboarding and proceed to click **Download onboarding package**. This is the file unique to our tenant's onboarding process, and what the `mde_install.sh` script will refer to.
3. You will need to get this to a location accessible by the server. One method I often use is to download from my browser, then use the download history to get the URL, which remains accessible by tools on Linux such as `wget` or `curl` for a few minutes, then copy it to the server clipboard.

Here's an example for `wget`:

```
wget
https://onboardingpackagescusprd.blob.core.windows.net/linuxserverscript/abc123/WindowsDefenderATP(
skoid=acb123
```

And here's one for `curl`:

```
curl
https://onboardingpackagescusprd.blob.core.windows.net/linuxserverscript/abc123/WindowsDefenderATP(
skoid=acb123 –output WindowsDefenderATPOnboardingPackage.zip
```

4. You will notice the onboarding package is a ZIP file rather than the more common `TAR.GZ` or `TAR.BZ2` Linux archive types. You may want to unzip it prior to getting it to the server, or you can install the `unzip` command and run the following:

   ```
   unzip WindowsDefenderATPOnboardingPackage.zip
   ```

The result is the following file, which we reference when we run the onboarding script: `MicrosoftDefenderATPOnboardingLinuxServer.py`.

5. Download the `mde_installer.sh` installation and onboarding script from GitHub for execution on your Linux server. One way of doing this might be `wget` or `curl`.

Here's how you'd do it using `wget`: `wget https://raw.githubusercontent.com/microsoft/mdatp-xplat/master/linux/installation/mde_installer.sh`

And you'd run this command if using `curl`: `curl https://raw.githubusercontent.com/microsoft/mdatp-xplat/master/linux/installation/mde_installer.sh -o mde_installer.sh`

6. Give the script executable permissions with **chmod**:

```
chmod +x mde_installer.sh
```

7. You can now run the Bash script, referencing the onboarding file. There are a few options for the script that we'll need to specify too. The following is an example of an installation command, followed by an explanation of the options:

```
sudo ./mde_intaller.sh –-min_req --install –-channel prod --onboard MicrosoftDefenderATPOnboa
```

Let's review the options you can pass to the script and when you should use them:

- **--channel** controls which of the three update cadences this server's agent will receive. This is not the update frequency of signatures: it is the ability to opt into previews. Available values are **prod**, **insiders-slow**, and (default) **insiders-fast**.
- **--clean**, for SUSE only, will tidy up by removing the **packages-microsoft-com** repository.
- **--debug** mode can be used in troubleshooting scenarios but not normal usage.
- **--help** lists the options available in the current version of the script, including their short versions, such as **-c** for **--channel** and **-o** for **--onboard**.
- **--http-proxy**, **--https-proxy**, and **--ftp-proxy** are used when you need to specify a proxy for these.
- **--install** does what it says on the tin and is mandatory to proceed with the installation.
- **--min-req** should always be used as this confirms and enforces minimum requirements prior to installation. For example, confirm there is enough memory before proceeding; otherwise, exit the script.
- **--onboard** also does what it says on the tin and must point to the script you downloaded from the Microsoft 365 Defender portal.
- **--passive-mode** is an option if another security tool that handles endpoint protection antimalware, but you still want to send telemetry to MDE.
- **--remove** can be used if the time comes to uninstall MDE.
- **--skip_conflict** should only be used if there are other security products running on the system and you are willing to accept the consequences (risk of conflicts and performance hit) of running them side by side with MDE. The script will otherwise find CrowdStrike (**falcon-sensor**), Carbon Black (**cbsensor**), McAfee (**MFEcma**), or Trend Micro (**ds_agent**) services running on the system and exit.
- **--tag** sets a tag for the device. **GROUP** must also be specified (for now, the only supported key name) as well as a value, such as **UbunutuServers**—for example: **--tag GROUP UbuntuServers**.
- **--upgrade** forces an agent update, if available.
- **--verbose** is an obvious one and is always recommended too so that you see the full output onscreen about progress.

- `--version` will report the version of the script—for example, 0.5.4. Alternatively, you can look at the script in an editor and find `SCRIPT_VERSION`. Generally, you should download the latest version each time you onboard a new system.
- `--yes` automates the script further by not prompting the administrator for responses, so for ease of use, this one is recommended.

A successful script running with the `verbose` option will output `Onboarded: true`, and confirm other options you passed to the script.

Microsoft Defender for Cloud onboarding

Microsoft Defender for Servers is the enhanced security feature of Microsoft Defender for Cloud that can automatically onboard Linux **virtual machines** (**VMs**) in Azure and Azure Arc into MDE. The steps for enabling this automatic provisioning are the same as those detailed in the *Windows Server* section of *Chapter 4*, with the notable difference that the extension is `MDE.Linux` rather than `MDE.Windows`. The Log Analytics agent also differs. Ubuntu, for example, receives **OMSAgentForLinux** rather than Windows' **MicrosoftMonitoringAgent**.

Azure Arc onboarding

Linux servers not hosted in Azure can be extended to Microsoft Defender for Servers by using Azure Arc, just as with on-premises or third-party cloud Windows servers, as explained in *Chapter 4*. The steps in Azure, including required permissions (roles), are the same for interactive configuration as those detailed in the *Windows Server* section of that chapter, with the following changes specific to Linux:

- As you progress through the wizard to add a single server, choose **Linux** within the **Server Details** section of the **Resource details** page
- The script you are presented with at the end of the wizard uses Bash, rather than PowerShell
- Execute the script on the server with root permissions. Here's an example:

```
chmod +x OnboardingScript.sh



sudo ./OnboardingScript.sh
```

As with the Windows process, the script will prompt you to visit microsoft.com/devicelogin to input a registration code. Once entered, the Linux server becomes manageable as an Azure Arc-enabled server. If you have enabled Microsoft Defender for Cloud, it will provision MDE on the server as part of Microsoft Defender for Servers.

Managing Linux protection settings

With MDE now deployed to your Linux servers, it's time to focus on customizing its protection settings. This section will educate you on the fundamentals of how Linux settings are deployed, how to manage scanning and remediation, exclusion control, and how updates can be scheduled.

As in other chapters, our focus here is enterprise deployment. While the **`sudo mdatp config`** command is available for individual *hands-on keyboard* servers, just as it was for macOS, we're going to focus on the configuration profile file. This file is how settings are controlled centrally.

Understanding MDE configuration profile files

Settings are deployed to Linux servers with a configuration profile, like the type of profile you learned about for macOS in the previous chapter. The difference for Linux is the format: JSON.

The good news is that JSON files are easier to read than macOS's XML files but maintain a similar structure, so you'll get used to them in no time. You build the configuration profile by once again specifying *keys* (setting names) that have different *values* (your chosen options). Unlike macOS XMLs, you don't have to specify data types such as string, integer, and Boolean, but we continue to have nested keys such as **`exclusions`**, which is nested within **`antivirusEngine`**.

The JSON file must be published to the server exactly as **`/etc/opt/microsoft/mdatp/managed/mdatp_managed.json`**. For macOS, this book encourages the use of Intune to distribute the configuration profile file. Linux servers cannot be managed by Intune, so you should use an orchestration tool such as Puppet, Ansible, or Chef.

What does a Linux configuration profile file look like? Let's check out an example, then dissect it so that you understand exactly what's going on:

```
{
    "antivirusEngine":{
        "enforcementLevel":"real_time",
        "behaviorMonitoring":"enabled",
        "scanArchives":true,
        "maximumOnDemandScanThreads":1,
        "exclusionsMergePolicy":"admin_only",
        "threatTypeSettingsMergePolicy":"admin_only",
        "threatTypeSettings":[{
            "key":"potentially_unwanted_application",
            "value":"block"
            } ]
    },
    "cloudService":{
        "enabled":true,
        "cloudBlockLevel":"high",
        "diagnosticLevel":"optional",
        "automaticSampleSubmissionConsent":"all",
        "automaticDefinitionUpdateEnabled":true
        }
    }
```

We'll once again make some comparisons to the macOS XML configuration file you learned about in the last chapter. Notice that we no longer have dictionaries or any **Payload** prefixed keys to worry about (as those are specific to macOS). Instead, we get a very easy-to-interpret JSON file. Let's analyze it some more.

The JSON opens and closes with *curly brackets* or *braces*: **{ }**. These are also used to open and close the different groups of settings (*nested preferences* or *dictionaries*) such as **antivirusEngine** and **cloudService**. You can also see that square brackets, **[ ]**, are used to open and close nested values (arrays) such as **threatTypeSettings**.

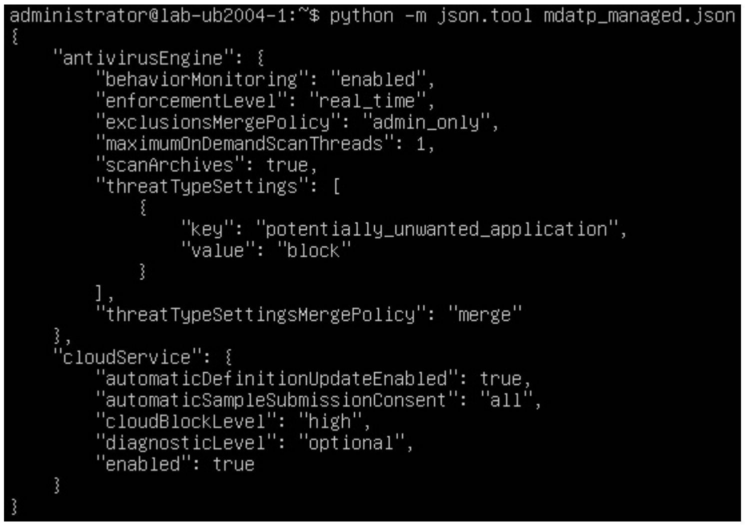
The following extract from the example JSON can be used to explain how we format individual settings:

```
"automaticSampleSubmissionConsent":"all",
"automaticDefinitionUpdateEnabled":true
```

Each setting (*key*) is called in double quotation marks, **" "**, followed by a colon, **:**, then its value. Values should also be in double quotation marks unless they are integers or Booleans (**true** or **false**). Also note that if a setting is not the last at its level, it has a comma at the end.

One configuration file is deployed to each server we want to control the settings of. If you want to be sure it's formatted correctly prior to deployment, you can use **python -m json.tool mdatp_managed.json**, as depicted in the following screenshot. If the JSON is invalid, you'll see a **No JSON object could be decoded** error message.

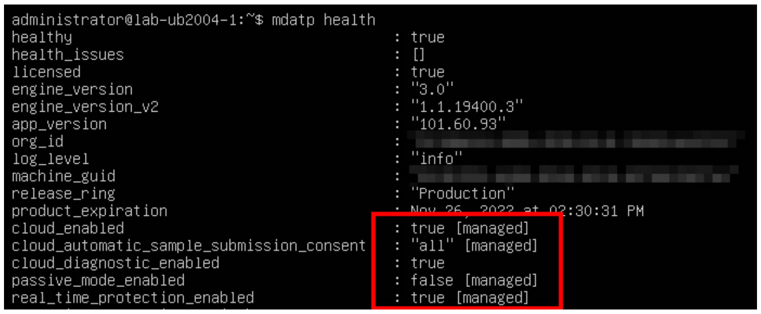
You can also use an online tool such as jsonlint.com:



```
administrator@lab-ub2004-1:~$ python -m json.tool mdatp_managed.json
{
    "antivirusEngine": {
        "behaviorMonitoring": "enabled",
        "enforcementLevel": "real_time",
        "exclusionsMergePolicy": "admin_only",
        "maximumOnDemandScanThreads": 1,
        "scanArchives": true,
        "threatTypeSettings": [
            {
                "key": "potentially_unwanted_application",
                "value": "block"
            }
        ],
        "threatTypeSettingsMergePolicy": "merge"
    },
    "cloudService": {
        "automaticDefinitionUpdateEnabled": true,
        "automaticSampleSubmissionConsent": "all",
        "cloudBlockLevel": "high",
        "diagnosticLevel": "optional",
        "enabled": true
    }
}
```

Figure 10.1 – Output of the JSON validator

After your orchestration tool has pushed **`mdatp_managed.json`** to **`/etc/opt/microsoft/mdatp/managed/mdatp_managed.json`**, you can verify that the settings are successfully applied and in a managed state by executing **`mdatp health`**. The settings in your JSON will have the **`[managed]`** suffix for some settings, as you can see in the next screenshot. You don't need to restart MDE for a pushed file to apply:



```
administrator@lab-ub2004-1:~$ mdatp health
healthy                                     : true
health_issues                               : []
licensed                                    : true
engine_version                              : "3.0"
engine_version_v2                           : "1.1.19400.3"
app_version                                 : "101.60.93"
org_id                                      : [redacted]
log_level                                   : "info"
machine_guid                                : [redacted]
release_ring                                : "Production"
product_expiration                          : Nov 26, 2022 at 02:30:31 PM
cloud_enabled                               : true [managed]
cloud_automatic_sample_submission_consent   : "all" [managed]
cloud_diagnostic_enabled                    : true
passive_mode_enabled                        : false [managed]
real_time_protection_enabled                : true [managed]
```

Figure 10.2 – mdatp health output confirming the managed settings

ONLY ONE CONFIGURATION FILE PER SERVER

There's a big difference in managing Linux server settings when compared to something such as Group Policy for Windows: files aren't merged. Therefore, if different servers require different settings, you'll need multiple files scoped specifically to the exact requirements, rather than mixing and matching pushed files.

You now understand how configuration profiles are structured, where they must be pushed to, and how to validate them. In the coming sections, you'll learn about the different settings available for configuration profiles so that you can get MDE for Linux configured to your own requirements. The first settings we'll review are those that control scanning and remediation.

Scanning and remediation

In the example in the *Understanding MDE configuration profile files* section of this chapter, you saw the two dictionaries that contain preferences for scanning and remediation: **`antivirusEngine`** and **`cloudService`**. These settings will look familiar to anyone who has already configured macOS configuration profile files.

First up, let's look at some of the most significant settings you need to know about within **`antivirusEngine`**.

`enforcementLevel` is used to control the running mode of MDE, and we have three values. The default is **`real_time`**, which keeps the device protected by scanning files as they're accessed. Alternatively, **`on_demand`** only scans files when a scan is started on a schedule or manually. Lastly, **`passive`** enables passive mode, which keeps EDR telemetry gathering capabilities and updates, but means MDE will not do any remediation. Passive mode is generally reserved for use during migrations or if an alternative product performs remediation and you only want more data in Microsoft 365 Defender.

`behaviourMonitoring` is disabled by default but you may want to change this to **`enabled`** for highly secure environments. While it may create more false positives—a risk with all heightened antivirus security changes—you will benefit from MDE's event correlation between processes and events that may be indicative of threats.

Another setting disabled by default you may want to consider changing to **`enabled`** is **`enableFileHashComputing`**. This doesn't pose a risk of false positives but may have performance implications, as now MDE will calculate the hash of all files that it scans. The reason you'd use this is to improve the reliability of indicators configured in the Microsoft 365 Defender portal.

If performance is a concern, consider changing the integer for **`maximumOnDemandScanThreats`**. This setting is called *the degree of parallelism for on-demand scans*, which isn't like setting a percentage of CPU to use, but rather the number of threads available during scanning. This defaults to **`2`**, so you can downgrade it to **`1`** or increase it up to **`64`**.

In our chapters on Windows and macOS, you will have seen advice about local policy merging. By default, this is allowed (**`merge`**) for the **`exclusionsMergePolicy`** and **`threatTypeSettingsMergePolicy`** settings but it's recommended you don't allow merge (**`admin_only`**). This means that exclusions and settings for the handling of threat types can only be controlled by the configuration profile you centrally deploy: not the settings a local administrator specifies to the MDE application.

Now, we'll check out the **`cloudService`** settings for configuration profiles.

One of the most important features of MDE is cloud-delivered protection. Within the profile file, we can force-enable this (it's on by default) by setting the **`enabled`** key to **`true`**. This is a dependency for another key feature: cloud block level. Managed with the **`cloudBlockLevel`** key, it can have **`normal`**, **`moderate`**, **`high`**, **`high_plus`**, or **`zero_tolerance`** values. These roughly translate to the level at which you'll tolerate the cloud protection service returning false-positive results, with **`normal`** returning few but at the cost of poorer protection, and **`zero_tolerance`** most likely to return false positives but with the security benefit of not allowing any unknown programs to execute. There's no right or wrong option here, but generally speaking, your high-value, high-risk assets should have the greatest cloud block level possible. Ideally, you would consider starting higher up the chain and lowering only if absolutely necessary. In some scenarios—where time is precious, and you need to get MDE deployed in a hurry with minimal productivity risk (for example, licensing deadlines when migrating to MDE)—you could do the opposite: start low and increase when you can set time aside to manage any problems as a result.

As with macOS, you have **`automaticSampleSubmission`** and **`automaticDefinitionUpdateEnabled`** keys, which this book recommends be set to **`true`**.

The configuration profile isn't used to schedule scans with MDE for Linux. Instead, you'll use Linux's built-in scheduling capabilities. Let's have a look at doing this with **crontab**.

While a full overview of using **crontab** is beyond the scope of this book, you edit files that contain times and the commands to be executed. Orchestration tools for Linux such as Chef, Puppet, and Ansible will let you manage this centrally. For MDE, scheduling means you'll update your **crontab** file to specify when you want to scan and what type of scan.

You can run a full scan, custom scan, or quick scan. A quick scan targets the locations most commonly seen to contain malware as determined by Microsoft. In the following example, we have a line in our **crontab** file to schedule a full scan every Sunday at 4 a.m.:

```
0 4 * * 0 /bin/mdatp scan full >/dev/null 2>&1
```

In the next example, a quick scan is scheduled daily for 5 p.m. This scan also creates a log file:

```
0 17 * * * /bin/mdatp scan quick >> /home/user/logs/mde-cron-scan.log
```

In our final example, we're using a custom scan every day at 4 p.m. This scan also ignores exclusions:

```
0 16 * * * /bin/mdatp scan custom --path /bin/lobapp --ignore-exclusions >/dev/null 2>&1
```

With scanning and remediation settings—including using **crontab** to schedule—now understood, let's have a look at how we can create some exclusions for those.

Exclusions

If you've read the chapters on MDE for Windows and macOS, you can probably predict the first thing this book will say about exclusions for Linux: avoid them as much as you can. Exclusions are a bypass to protection capabilities that may benefit performance or support requirements but can be leveraged by attackers to sail right through your defenses.

With that disclaimer out of the way, we can have a look at using our configuration profile to manage exclusions. Note that these apply to the local endpoint protection engine rather than EDR alerts.

Before going any further, you should read the *Exclusions* section in the last chapter, *Chapter 9, Onboarding and Managing macOS*. This is because the types of exclusions and their capabilities are consistent across macOS and Linux: both support extensions, folders, files and processes, and wildcards in the same way. What differs is how these exclusions are formatted, due to the difference in XML and JSON.

Exclusions are controlled by a dictionary key called **`exclusions`** that is a nested preference of **`antivirusEngine`**. Let's check out a full example:

```
{
    "antivirusEngine": {
        "exclusionsMergePolicy": "admin_only",
        "exclusions": [{
                "$type": "excludedPath",
                "isDirectory": false,
                "path": "/bin/script1.sh"
            },
            {
                "$type": "excludedPath",
                "isDirectory": true,
                "path": "/home/*/LobApp"
            },
            {
                "$type": "excludedFileExtension",
                "extension": ".xml"
            },
            {
                "$type": "excludedFileName",
                "name": "companyapp"
            }
        ]
    }
}
```

Hopefully, you agree that it's reasonably easy to read and convert to your own needs, but let's consider a few key things to note.

We begin with the **`exclusions`** key, then **`exclusionsMergePolicy`** set to **`admin_only`**. This value means that only exclusions specified in the configuration file are honored; others set locally won't apply. This is recommended so that you can centrally secure exclusion management, but you can alternatively use **`merge`**.

Next, there's an array of exclusions. The example has four different types of exclusions, but you could have multiple types of the same exclusion, each with its own place in the array. For example, if you have another two filenames to exclude, you will have something like the following:

```
                {
                "$type": "excludedFileName",
                "name": "onefile"
            },
            {
                "$type": "excludedFileName",
                "name": "anotherfile"
            }
```

Note that a comma must be appended to the curly bracket that ends each item in the array except the last item.

excludedPath is used for both directories and full file paths, with **isDirectory** being **true** or **false**, letting MDE know which.

When designing your configuration profile file, remember the order of preference for exclusions: full file paths are most preferred due to how specific they are; extensions and filenames are least preferred as they are least specific.

In addition to the **exclusions** configuration options, there are **allowedThreats** options. Let's look at an example of this code in the configuration file and then explain it:

```
{
    "antivirusEngine":{
        "allowedThreats":[
            "PUA:MacOS/Adload.L!MTB"
            ]
    }
}
```

allowedThreats lets you specify an array of specific threats that MDE will ignore. In the example, **PUA:MacOS/Adload.L!MTB** gets a pass. Approach this with caution, and generally only after a confirmed false positive has been identified, though review its need regularly.

If you're going to create exclusions, the least you can do is keep MDE up to date! So, let's see how that works for Linux.

Managing updates

Signature updates, also referred to as **threat intelligence** (**TI**), are updated automatically several times each day by MDE on Linux. If you have command-line access, you can also invoke those updates manually with **mdatp definitions update**, which pulls those updates (only) from the internet directly. The application itself, however, requires you to configure updates.

Earlier in this chapter, you learned about **crontab** to schedule different scan types. With MDE for Linux, if we want to force updates at specific times and intervals, we use it again. A good approach for update frequency is a ring-based approach: some servers update more frequently than others, and after you've confirmed there have been no problems, other servers can get those updates, less frequently.

This time, we'll invoke different commands depending on what type of Linux OS we've got MDE installed on. This is because Red Hat-based OSs have a different update tool (**yum**) than SUSE OSs (**zypper**), which are different than Debian OSs (**apt-get**). Make sure you know which command your OS uses before updating **crontab**.

In our first example, we'll take on Debian OSs, which include Ubuntu. This line in **`crontab`** will have our MDE update (if available) each day at 1 a.m. and output to a log file:

```
* 1 * * * sudo apt-get install --only-upgrade mdatp >> home/user/logs/mde-cron-update.log
```

Now, we'll consider OSs that use **`yum`**, such as Red Hat and CentOS. This example will try to update every 6 hours on weekends, with no log file:

```
* */6 * * 0,6 sudo yum update mdatp -y >/dev/null 2>&1
```

Lastly, let's look at a SUSE example that runs every 5 days at midnight:

```
* 0 */5 * * sudo zypper update mdatp >/dev/null 2>&1
```

You can use these examples to start building out your own **`crontab`** requirements for MDE's application updates, depending on the Linux distributions in your scope.

UPDATE CHANNELS

In the Onboarding Linux section of this chapter, you learned about **`insiders-fast`**, **`insiders-slow`**, *and* **`prod`** *update channels. If you want to test out early versions of the MDE software for Linux as part of your update management, you specify these in the configuration.*

With your server now receiving regular MDE software updates, we'll cover one final Linux setting that you'll have seen in other OS chapters: **potentially unwanted application** (**PUA**) protection.

PUAs

A gray area of antimalware protection is PUA protection. What makes PUAs gray is they are not clearly malicious. They might have some legitimate use cases, but they would generally raise eyebrows, particularly in an enterprise environment. Examples might include red/blue/purple team tools, cryptominers, and so on.

PUA protection can run in three modes on Linux servers:

- Completely disabled, which is achieved with the **`off`** value
- Audit mode, which doesn't perform any remediation but records what would have been prevented; achieved using the **`audit`** value
- Block mode, which fully enforces PUA protection and is achieved with the **`block`** value

These values are specified in a configuration profile in the **`threatTypeSettings`** dictionary. Let's check out an example:

```
{
    "antivirusEngine": {
        "threatTypeSettingsMergePolicy": "admin_only",
        "threatTypeSettings": [{
            "key": "potentially_unwanted_application",
            "value": "block"
        }]
    }
}
```

Firstly, you can see another policy merge setting: **`threatTypeSettingsMergePolicy`**. Again, our options are **`admin_only`** or **`merge`**, which allow local settings to be ignored or respected. Generally, you'll want **`admin_only`** to make sure this is only managed by whoever controls your configuration profile file, rather than local server users too. Next, against **`value`**, you choose **`block`**, **`audit`**, or **`off`**.

You're now aware of how to manage **`threatTypeSettings`** in your Linux configuration profile, which also concludes our overall review of managing the configuration profile. Let's head on to summarizing what you've learned in this chapter.

Summary

This chapter taught you how to onboard Linux servers into MDE. You learned that client OSs are unsupported but that many server distributions are supported. We explored the use of a Microsoft-provided script to onboard them, including its various options for customized deployments. You also found out that Microsoft Defender for Servers, part of Microsoft Defender for Cloud, provides automatic onboarding for Linux servers, and that you can extend this to on-premises or third-party clouds using Azure Arc.

In the next chapter, we explore leaving the server world and focus on the most portable devices: iOS and Android.

Questions

Now that you understand the onboarding processes for MDE, you can test your knowledge with the following questions:

1. Which of the following Linux distributions are supported? Choose all that apply.
    1. Oracle Linux 6.7
    2. Oracle Linux 7.2
    3. Oracle Linux 8
    4. Amazon Linux 2
2. Which of these should you expect to see on an Ubuntu server onboarded using Microsoft Defender for Cloud? Choose all that apply.
    1. **MicrosoftMonitoringAgent**
    2. **MDE.Cloud**

   3. **MDE.Linux**
   4. **OMSAgentForLinux**
3. You want to pilot MDE on some Linux servers purely for endpoint telemetry information and not antimalware. To achieve this, which of the following options can be passed to the script when onboarding devices?
   1. `--pilot`
   2. `--pilot-mode`
   3. `--passive-mode`
   4. `--passive`
4. True or false: There is a GUI version of MDE available for client Linux distributions such as Ubuntu.
   1. True
   2. False
5. Which of the following would you be required to process if adding an individual Linux server to Azure Arc with the generated script? Choose all that apply.
   1. An Azure AD **testing procedure specification** (**TPS**) report
   2. An Azure AD trusted location
   3. An Azure AD application registration
   4. An Azure AD device login registration code

Further reading

There may be some specific scenarios and news regarding MDE for Linux that this book has not discussed. You can find useful information on examples of these with the following links:

- Microsoft's announcement of the general availability of MDE for Linux, at the time called Defender ATP, is a milestone in the business's journey beyond protecting only Windows: techcommunity.microsoft.com/t5/microsoft-defender-for-endpoint/microsoft-defender-atp-for-linux-is-now-generally-available/ba-p/1482344
- The `mdatp-xplat` repository on GitHub hosts diagnostic tools, scripts, and installation scripts (referenced in this chapter), and is available here: github.com/microsoft/mdatp-xplat