Tuan Tran

MAINTAINING LINUX REPOSITORIES FOR CLOSED NETWORK INFRASTRUCTURE

Bachelor's / Master's thesis

Bachelor of Engineering

Information Technology

2023

South-Eastern Finland
University of Applied Sciences

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering
Author(s)	Tuan Tran
Thesis title	Maintaining Linux Repositories for closed network infrastructure
Commissioned by	Mipro Oy
Year	2023
Pages	64 pages, 2 pages of appendixes
Supervisor(s)	Matti Juutilainen

ABSTRACT

The thesis touched upon the realm of Linux repository management and focused on delivering a comprehensive explanation and guide for constructing, configuring and securing a Red Hat Enterprise Linux repository. The solution involved elevating powerful software such as dnf, reposync and httpd. The process was recorded in a systematic approach to creation while mitigating potential vulnerabilities.

Starting with a breakdown of the concept of a software marketplace in the modern-day choice, the study moved to clarify Linux repository structure and the risk of resorting to traditional methods. A conclusion was made that there was a need to develop an independent local repository that would sync content from the internet to feed onto recipient systems.

The research progresses to network architecture, focusing on network members and security principles like firewalls and integrity calculation. A thorough step-by-step guide was conducted on how to set up the repository from scratch. The process involved network resource placement, web server configuration, repository preparation and synchronisation. Meanwhile, intricate topics such as SSL and package security were also explored.

Furthermore, the solution also included the same process for setting up a client system to retrieve content from the homemade repository, considering both types: recipient and mirror setup. Finally, the project emphasises the idea of automation of the processes to promote sustainability and negate end-user periodical intervention.

The study contributed insights and practical production value for building and administrating a secure Linux repository. Thereby, it presented significant value in cybersecurity practices in sensitive Linux environments. The discovery presented a blueprint for organisations to opt for a centralised method of deploying systems with internet access limitations.

Keywords: repository, Red Hat Enterprise Linux, local, reposync, httpd, dnf

CONTENTS

1 INTRODUCTION

The existence of a repository is not new; it is the most common way to manage and distribute applications in an eco-system of a software brand. The most common repositories are Google Play Store by Google, App Store by Apple, and Microsoft Store by Microsoft.

Each differs in how they contain, manage, and deploy applications for the end devices. Nevertheless, they share a common goal: centralising the installation, updating and discovering software conveniently. This solution also guarantees the source of the applications, and the entire process is secure at every step.

On the other hand, this solution also helps developers and creators publish software securely while helping distribute and advertise it to the mainstream. Together, the existence of a repository helps connect publishers and developers to the consumer in the most centralised way while ensuring the safety and inclusiveness of end devices.

Linux repository is no different story. Each Linux distro comes with its repository with a collection of software and dependencies to support the applications. This thesis, however, will mainly focus on the Red Hat Enterprise Linux repository.

In this case, the familiar pattern has been a sponsored provider for the repository directly from the brand. However, in many cases, there is a need to set up new systems entirely offline. There are many reasons for this, but the most common ones are for security policy compliance or due to the sensitivity production value of the system. On the other hand, there is also the usage for systems that are remote and therefore need a portable solution to be deployed there.

Therefore, this thesis aims to develop a workflow to overcome this challenge. In this case, it will be an internal repository in a Demilitarized Zone network. On one end, it will have access to the internet through a firewall that will allow secure connections to the provider. Conversely, it will grant access to the internal network with systems needing software installation and updates.

The goal is to achieve a consistent setup so that any system introduced into the internal network will have access to the repository to startup as soon as possible. Nevertheless, a secondary goal is to achieve a sustainable solution so that old systems can access compatible software versions. In contrast, new systems can access the latest updates and security patches.

2 BACKGROUND

This chapter will discuss many working theories behind the tools used in this project. It will cover software options that enable the repository to come to life and hardware equipment that has crucial roles in protecting the members of the project.

2.1 Linux Repository and packages distribution

While Windows and Mac OS repository serves as a catalogue of applications and focuses more on the advertisement of the application itself in terms of feature and functionality for the end user, Linux's approach to the repository, however, is much more complex. They put their attention towards dependency resolution, versioning, and packaging aspect. These are all critical factors in achieving package installation and management on Linux.

Linux utilises a packet management system that facilitates installation, update, configuration, and removal tasks. Different distributions adopt distinct methods for managing packages. For instance, Ubuntu, a Debian-derived Linux distribution, uses APT, Advanced Package Tool, as their choice for application management. Conversely, RHEL, a Fedora-based Linux distribution, selected dnf (Dandified YUM) as their favoured package manager. Both systems share patterns like package structure, dependency resolution and repository management (Reselman 2022).

Nonetheless, their task execution methods differ substantially. APT holds packages in DEB file format, while dnf uses RPM. In contrast to operating

systems like Windows or Mac, which utilise EXE and APP files to install applications as all-in-one packages, Linux needs dependencies for packages to function (Reselman 2022).

For Ubuntu, a control file within the package file stores metadata. It contains information about the package, name, version number, dependencies, and a brief description of the package. In RHEL's case, this data is stored in a spec file containing build instructions and scripts (Reselman 2022).

Dependency handling is crucial for package installation since the requested package typically lacks the necessary tools. The spec file in dnf encompasses all required dependencies and their version numbers. The dnf algorithm processes dependency queries recursively, resolving conflicts and issues individually and in real-time during package installation.

On the other hand, APT's method is much more advanced and is based on a directed acyclic graph (DAG). In this case, it handles complex dependency orders while recommending alternative packages on the off chance that conflicts may arise. When it comes to compatibility, APT outshines dnf by providing a more comprehensive range when it comes to dependencies' versions, making it more flexible, all the while allowing dependencies to update to newer versions if necessary. Contrary to dnf's method, APT's method is transaction-based, meaning it resolves all dependencies simultaneously, which can be proven more efficient.

Finally, these two package managers' techniques of repository indexing are also different. With so many packages, there must be an efficient way of sorting and querying. From the perspective of APT, a package list file oversees the repository metadata. The file contains information about all the packages in the repository, which includes names, versions, and dependencies. When a user queries, APT looks up this list to determine the availability and ties to it. By updating this file, the repository is ensured to have the latest information.

dnf's angle on this matter is a repodata directory. It contains metadata about each package rather than a whole list of packages. The directory includes each package's name, version, dependencies, and other relevant information. To put the difference between these two's practices, APT will have to update the entire list of new information when a package is updated. At the same time, dnf will only require that specific package metadata be updated. This directory approach makes dnf better at metadata retrieval in terms of efficiency, as only needed packages are looked up.

In summary, although both package managers serve the same purpose, they exhibit differences in their underlying processes. This thesis will concentrate on the workings and procedures behind dnf. The package manager facilitates repository configuration, searching, installation, updates, and clean-up, offering a streamlined and automated means of application acquisition, ultimately simplifying and abstracting the process for end-users.

2.2 Legacy methods and vulnerabilities

Traditionally when deploying RHEL systems, they will require software installation and updates. So how would one use to do it? There are generally two methods.

First, the system establishes an internet connection to access RHEL's Content Delivery Network (CDN). Utilising a global provider ensures that the system can consistently obtain the most recent updates from the provider, eliminating dependence on regional mirrors, as with other Linux distributions. Additionally, Red Hat's CDN delivers secure connections via HTTPS and SSL, enhancing the safety and reliability of users during installation or updating processes (Red Hat Customer Portal n.d).

Alternatively, the external storage method comes into play in a sensitive system with a policy that dictates a zero-internet requirement. The same result can be achieved as the previous approach by downloading packages to external storage and importing/installing them manually into the system. However, there are

caveats. As discussed, dependency is the main factor in the application's installation.

In many cases, there will be compatibility issues. Hurdles will lead to back-and-forth acquirement of dependency or even straight-up incompatibility with the OS, which is inconsistent and hinder the workflow significantly. Not to mention, this method introduces uncontrollable variables regarding safety. The package may contain malware and other harmful code if downloaded from an unknown and untrustful source. Furthermore, this approach makes it impractical to update the packages in the future as there is no official way of keeping track of the versions of the installed packages.

Therefore, there was a need to introduce a new method that would overcome the disadvantages mentioned above while also complying with security policy. This is where the local software marketplace comes into play, as it will nullify many threats discussed in this chapter.

2.3 Network architecture

When designing a dedicated repository, the network topology must first be looked at as it involves the functionality and security of the repository as well as the clients themselves. This is why a demilitarised zone network (DMZ) was chosen for the topology.

In this case, a DMZ network is a secure solution that allows controlled access to external destinations, such as the internet. The network is usually situated between an internal network (where the client systems will be) and an external network where potential threads may lie. The goal of the DMZ is to allocate a secure buffer zone that negates live access to the internal network from external locations (Fortinet n.d).

DMZ is designed to promote a high sense of security and control regarding external access to the private network. By placing shared resources in the buffer zone, they can dictate which services are available outside and protect the

internal resources from direct exposure to the internet. This can elevate integrity and prevent data breaches and security incidents that would violate the policy of the client systems in the internal network.

DMZ configuration consists of a firewall that serves two functions, one that filters traffic between the internal network and the buffer zone (Figure 1). The other sifts the content between the external network and the buffer zone. By isolating resources like this, each member is ensured only to have a minimum traffic allowance through the firewalls. Connections will only come down to compulsory links (Okta 2023).

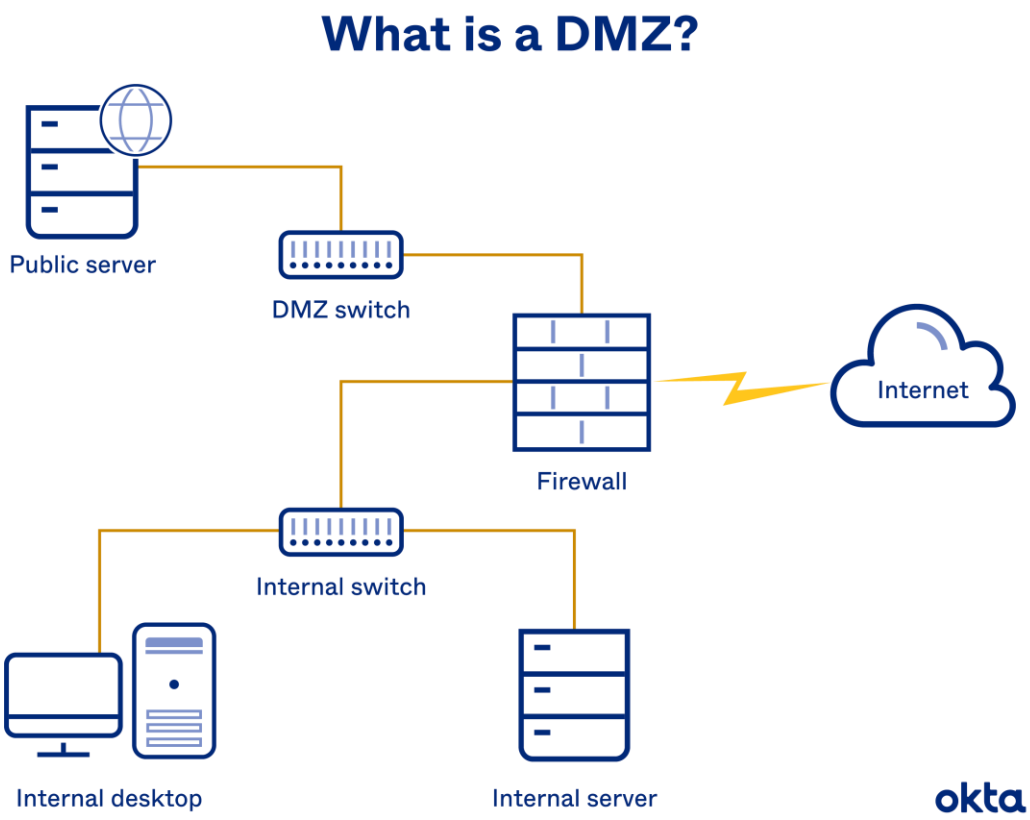


Figure 1. Typical DMZ topology (Okta 2023)

The DMZ plays the most crucial role in network security, providing a secure environment for external-facing servers and services. This is the most effective way to store sensitive resources from unauthorised access while delivering necessary services to internal clients.

2.4 Software

This sub-chapter covers all the software that is used for the repository. It comprises operating systems, web servers, package managers and automated scripts.

2.4.1 Red Hat Enterprise Linux

Red Hat Enterprise Linux got its inspiration from the Fedora Linux distribution and is currently under the care of Red Hat Inc. The product is highly nominated for enterprise adaptations and mission-critical applications and services. It involves many software and tools, production-grade software, development kits and desktop applications.

A notable feature of Red Hat Enterprise Linux is its subscription-based support for enterprise customers. The company distributes software updates, security patches, and technical assistance through an extensive range of subscription models, catering to various clients such as Fortune 500 companies, government agencies, and educational institutions (Red Hat n.d).

In summary, customers put their faith in Red Hat because they believe in security and are often entrusted with missions that require such. Their solution caters towards virtualisation, cluster and high-availability model. Beyond that, the project's solid and energetic community contributes to its growth. This is why this operating system is regarded as mature and widely favoured above other flavours.

2.4.2 dnf

As discussed before, starting from version 8, dnf was the desired package management tool for RHEL, succeeding the Yum package manager.

The primary purpose of this arrangement is to create a software suite that promotes speed and efficiency in package management, excelling in complex

transactions with improved dependency resolution. Dnf operates on a plugin structure, which facilitates expandability in its functionality.

dnf supports many different repositories, allowing users to install packages flexibly from diverse sources. Furthermore, it adeptly manoeuvres tasks like upgrades, updates and removal of packages (Fedora Project n.d).

dnf, known for being a versatile package manager, presents a practical and automated methodology for managing software packages on RHEL. This addition is a vital factor in the operating system and even more important in creating the remote repository.

2.4.3 yum-utils and reposync

Even though dnf is used as a package manager, the fact that the tool supports plugins as part of its infrastructure allows it to introduce other components, such as yum-utils. This tool supplies utilities for additional functionality to manage repositories on an RHEL system.

In this thesis, however, reposync is the main focus, a command-line tool provided by yum-utils. Its primary purpose is to mirror a dnf repository. This will allow users to download all the packages and metadata in the remote repository to a local system (Man7 n.d).

The tool is commonly used to synchronise and maintain a local repository of packages on a system instead of having to be dependent on a remote repository. This is extremely useful in this case, where client systems are required to have as minimal contact with the outside world as much as possible. On another spectrum, it is also helpful for clients that are unable or unreliable to have updates via the internet.

To break down the process of a reposync execution, it first downloads the repository metadata from the remote repository. This metadata contains details

about available packages, dependencies and updates. After that, it will start downloading all the resources.

The beneficial factor of the sustainability of reposync is its ability to maintain the repository efficiently. When checking with the remote repository's metadata, it will only download newer or missing packages if a local repository already exists. This critical feature allows the local repository to be built and updated for the latest revisions.

2.4.4 httpd

httpd, or Hypertext Transfer Protocol Daemon, is a web server software that hosts web pages and other content. It is commonly tied to Apache HTTP Server, one of the most popular web servers in the world.

The web server is niftily configurable to meet the requirements of different applications. It supports a versatile range of programming languages and frameworks. Security-wise, it also complies with various authentication mechanisms.

This software listens for incoming HTTP requests from clients and responds with the desired resource, ranging from HTML pages, images, scripts and files. In this case, the main feature that will be utilised is hosting files for the repository so the client can access them. httpd will serve as a file server for the repository, containing all the packages and metadata.

2.5 Security

While this topic does not actively reflect the operation of the homemade repository, it is still worth discussing because the elements involved, such as client systems, are sensitive to vulnerabilities. Therefore, to cater to such segments, it is vital that the environment and the members are at least equipped with some defence mechanisms.

2.5.1 firewalld

Firewalld is the software firewall included in the operating system of Red Hat. It will be the first layer of network security in the topology. The security measure is easy to use while allowing rules to be processed dynamically.

This software uses zones as its trustworthy level definition for network connections. A zone is a collection of configured rules deciding what traffic is permitted or negated.

For instance. the default 'public' zone will allow outgoing traffic to the internet but denies external traffic. Meanwhile, the 'trusted' site will agree to connections from trusted networks and services. Because of this, the 'trusted' zone will tend to be applied on interfaces facing the internal network.

Firewalld rules are very flexible. They allow the creation of traffic filters based on criteria such as source/destination IP addresses, port numbers and even protocols.

The 'nftables' framework is used to put the rules into practice. It guarantees that the network traffic is effectively processed while keeping up with speed factors. Without excluding, Firewalld also supports various services, such as network address translation and port forwarding. This feature help deploys internal services accessible from the internet.

Firewalld is a powerful and flexible yet easy-to-use firewall management solution. It aims to deliver a simple and intuitive method for rules implementation on the RHEL system. For the dedicated repository, however, the main focus will be firewall rules setup that would let necessary traffic in and out of the DMZ.

2.5.2 Physical firewall

A physical firewall helps the filtering process of traffic even deeper compared to the software firewall provided by the operating system. It is configured to filter

incoming/outgoing IP addresses, domain names, ports and protocols. They also support logging functions that detect and respond to potential threats beyond this scope.

When configured correctly, a physical firewall can help prevent malicious attacks and block out any unnecessary connections that were an original part of the system. Physical firewalls have extensive features beyond filtration, such as scheduling when policies kick in or packet capturing for troubleshooting.

In many cases, the firewall also acts as a gateway for the network because it has capabilities to handle routing as well. Since these studies are separated by networks in different subnets, it is essential to have one endpoint that handles routing traffic between each zone. Furthermore, suppose the firewall is going to act as a router. In that case, it is indispensable to mention that they also support features like Dynamic Host Configuration Protocol (DHCP) for distributing IP addresses to clients in subnets.

When combining traffic filtering and routing functionality, network zones can be established so that they are isolated from each other and propagate mandatory traffic between them. A firewall is a powerful tool that will help segment network traffic and provide values such as integrity and security to the operation.

2.5.3 gpgcheck

One of many tasks that reposync and dnf handle is the process of checking for package integrity cause otherwise, how does one make sure whatever package is being downloaded or mirrored is safe or in its original state? This is where gpgcheck (GNU privacy guard check) comes into play. It is a security measure ensuring the authenticity and integrity of downloaded software packages.

As for how GPG generally works, a digital signature is born when a package is signed with GPG and can be compared against the package later to determine if it produces the same value. The digital signature is constructed using the private

key of the maker. Later on, the public key would be used when necessary to verify.

When a package is downloaded from a repository, the GPG check process compares the digital signature against the provider's public key by default. What is assuring here is that in the case of RHEL, the public key is already included in the system, so there is no risk of key tampering or having to import public keys. If the signature matches, the package is only then deemed safe.

GPG is an important security feature and one of many security gatekeepers in this local repository solution. With the process involved, the provided service for client systems is secure and untampered despite multiple jumps.

3 REPOSITORY SETUP/CONFIGURATION

This chapter will demonstrate the practical installation and implementation of the solution.

It is important to note that this thesis will work on the basis that an RHEL 8.7 system already exists. Other variables, such as OS installation and hardware, will not be mentioned as they depend on the environment and requirements.

3.1 DMZ placement

As stated above, a DMZ network topology will be used in this setup. This chapter will discuss resource placement, specifically the buffer zone and the internal network.

First, the repository will be placed in the buffer zone between the two firewalls' connections. This will secure two functions. The server will be able to retrieve content from the internet, more specifically, packages and updates from the official repository provided by Red Hat.

Second, the machine will serve the Red Hat clients in the internal network. This will guarantee the safety of customers' devices as they will not be exposed to the public internet while also being able to get the latest packages and updates from the local repository.

By distributing resources this way, the integrity of any node is in good hands. This approach allows flexibility between security policy and software compatibility.

3.2 Network topology

Now that the barriers of this system are clarified, a physical and logical representation of the local network will be further presented and discussed for this specific case study.

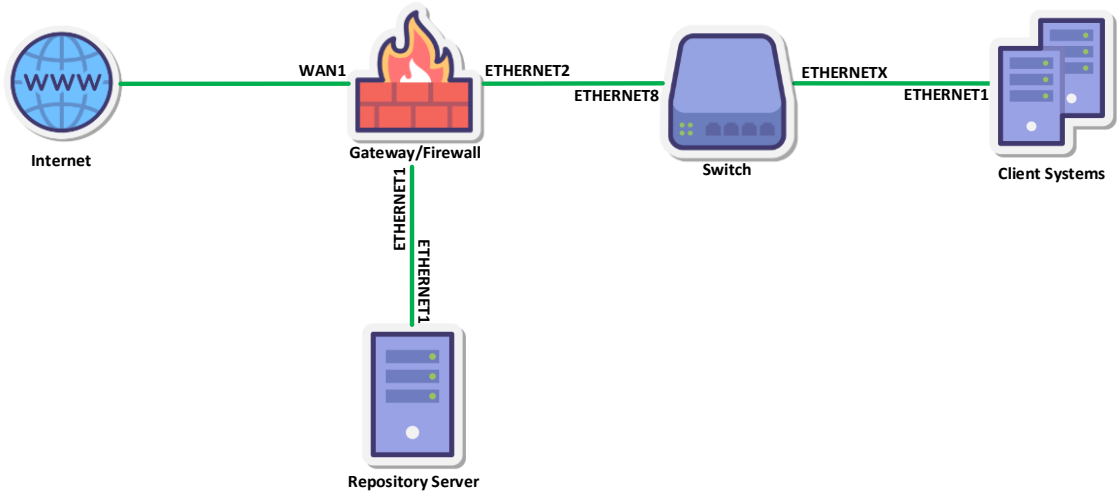


Figure 2. The physical topology of the network

Figure 2 describes the layer two topology of the network in this scenario. For firewall equipment, a Fortinet device will be used. It is capable of a multitude of features that will be discussed later in firewall configuration. A dedicated port on the firewall (WAN1) will be dedicated to the external network. For the sake of abstracting, this port will be considered an internet port.

Meanwhile, the repository server will be plugged directly into the firewall via port ETHERNET 1 of the firewall. On the other hand, the client systems will gain entry through the firewall under a switch using the port ETHERNET2. The reasoning

for this is to alleviate the number of ports used directly on the limited firewall. Since the firewall rules for the client systems will be inherently the same for the whole subnet, it will be much more logical to deploy it under a switch.

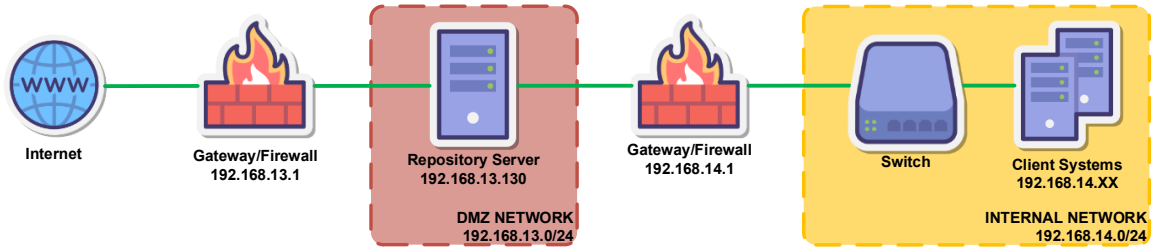


Figure 3. Layer 3 topology of the network

Figure 3 depicts the routing breakdown of the network. In this demonstration, the DMZ network subnet will be 192.168.13.0/24, and the internal network will be 192.168.14.0/24. Each subnet will have the first IP addresses act as their gateway. Both networks will also be able to rent addresses thanks to DHCP provided by the firewall. This is helpful because it reduces the extra work required for systems to be introduced to the internal network later.

3.3 Red Hat license registration

To begin with, Red Hat license activation grants access to the public repository provided by Red Hat. To activate the machine, the **subscription-manager** command is used.

First, log in to the account using the **register** option, then enter a Red Hat account username and password (Figure 4):

```
[repo@repository ~]$ subscription-manager register
You are attempting to run "subscription-manager" which requires
administrative
privileges, but more information is needed in order to do so.
Authenticating as "root"
Password:
Registering to: subscription.rhsm.redhat.com:443/subscription
Username: atutr
Password:
The system has been registered with ID: 1e5da6f9-355e-4f23-a11e-15bc4b594ba2
The registered system name is: repository
```

Figure 4. Red Hat system registration demonstration

An ID representing the activation tag is omitted, which can also be seen on the Red Hat Customer Portal, along with the system's name (Figure 5).

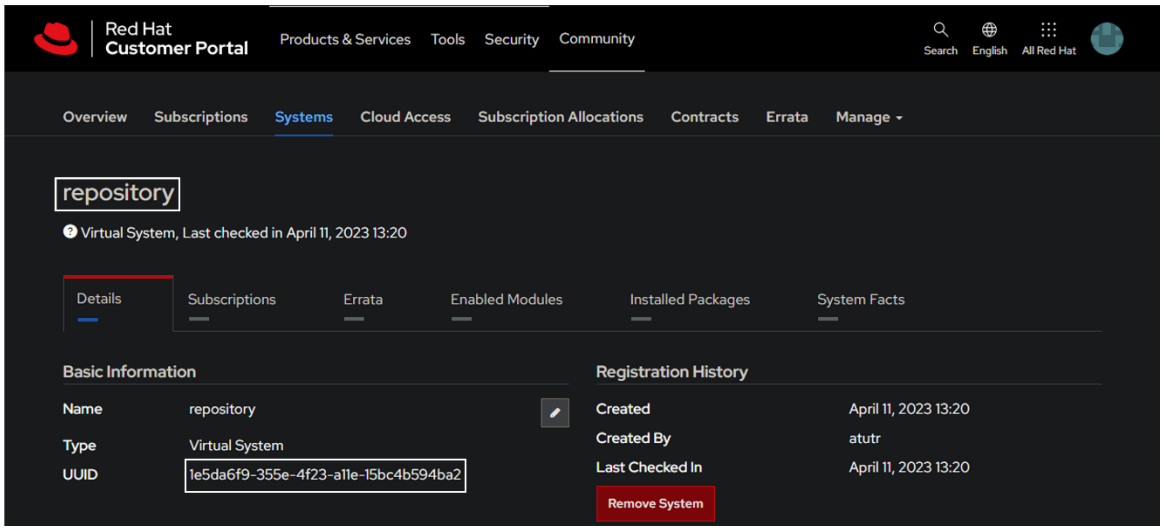


Figure 5. Red Hat Customer Portal Systems page

To activate the system, use the **attach** option with the **--auto** sub-option. The **--auto** sub-option attaches the most suitable license for the system depending on the subscriptions available on the account:

```
[repo@repository ~]$ subscription-manager attach --auto
You are attempting to run "subscription-manager" which requires
administrative
privileges, but more information is needed in order to do so.
Authenticating as "root"
Password:
Installed Product Current Status:
Product Name: Red Hat Enterprise Linux for x86_64
Status:       Subscribed
```

Figure 6. Red Hat system license activation demonstration.

The **Subscribed** output from the Status section indicates that the process was successful (Figure 6). For the sake of demonstration, a self-support trial license was used in this project. It will grant access to the BaseOS and AppStream repositories, the two fundamental sources needed for Red Hat to perform installation and update.

3.4 Software update

To see the repositories that are available in the system, the **dnf** command is used with the **repolist** option:

```
[repo@repository ~]$ sudo dnf repolist
Updating Subscription Management repositories.
repo id                           repo name
rhel-8-for-x86_64-appstream-rpms  Red Hat Enterprise Linux 8 for x86_64 -
AppStream (RPMs)
rhel-8-for-x86_64-baseos-rpms     Red Hat Enterprise Linux 8 for x86_64 -
BaseOS (RPMs)Complete!
```

Figure 7. Using dnf repolist to display all enabled repositories

By default, activating the system enables two repositories (Figure 7): BaseOS (rhel-8-for-x86_64-baseos-rpms) and AppStream (rhel-8-for-x86_64-appstream-rpms). These two sources allow the system to retrieve updates and install extra software and packages. This also enables the machine to perform system updates; keeping the server up to date with security patches is essential.

To update the system, the **dnf** command is used with the **upgrade** option and **-y** sub-option to bypass yes/no prompts:

```
[repo@repository ~]$ sudo dnf upgrade -y
[sudo] password for repo:
Updating Subscription Management repositories.
Last metadata expiration check: 0:12:26 ago on Tue 11 Apr 2023 06:43:08 PM
EEST.
Dependencies resolved.
...
Complete!
```

Figure 8. Using dnf to update the system

Depending on the hardware, this process may take a while. But a **Complete!** output will indicate that the task was successful (Figure 8).

3.5 Web server configurations

Httpd, the web server hosting application, is installed along with **mod_ssl,** a plugin that provides SSL support, to host the repository using the **dnf** command with the **install** sub-option:

```
[repo@repository ~]$ sudo dnf install httpd mod_ssl -y
[sudo] password for repo:
Updating Subscription Management repositories.
Last metadata expiration check: 0:26:59 ago on Tue 11 Apr 2023 06:43:08 PM
EEST.06:43:08 PM EEST.
Dependencies resolved.
...
Complete!
```

Figure 9: Installing httpd using dnf

Once again, the installation is relatively simple, and the final output indicates a successful attempt (Figure 9).

Next on the list is to create an HTTPS connection for the web server. Typically, an SSL/TLS certificate from a trusted certificate authority (CA) would be used.

This certificate verifies a domain name against a higher level of witness, proving that it is a safe and secure destination.

However, because the repository is in an isolated network, there is no need for a domain name, which would require some form of name resolution later. Nevertheless, a certificate's security features are still heavily needed. Therefore, a self-signed certificate will be created for the system.

```
[repo@repository ~]$ sudo openssl req -x509 -nodes -days 365 -newkey
rsa:2048 -keyout /etc/pki/tls/private/repository.key -out
/etc/pki/tls/certs/repository.crt
[sudo] password for repo:
Generating a RSA private key
.................................+++++
.....................................................................
.........+++++
writing new private key to '/etc/pki/tls/private/repository.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:
State or Province Name (full name) []:
Locality Name (eg, city) [Default City]:
Organisation Name (eg, company) [Default Company Ltd]:
Organisational Unit Name (eg, section) []:
Common Name (eg, your name or your server's hostname) []:
Email Address []:
```

Figure 10. Using OpenSSL to generate a self-signed SSL/TLS certificate

OpenSSL is used to create this self-signed SSL/TLS certificate. It has 365 days of validity from the creation date. The **-nodes** option specifies not to encrypt the private key, which has a length of 2048 bits. The certificate will only be used to

establish HTTPS connections with clients, as other functionality of SSL will not be utilised.

In Figure 10, when requested to make a certificate, many prompts displayed are used to propagate information for the certification, such as company name and email address. It is optional, and all of them can be left blank.

To tell httpd to use this certificate, its configuration is located in **/etc/httpd/conf.d/ssl.conf** requires some modification:

```
Listen 192.168.13.130:443 https
...
<VirtualHost 192.168.13.130:443>
...
SSLCertificateFile /etc/pki/tls/certs/repository.crt
SSLCertificateKeyFile /etc/pki/tls/private/repository.key
...
</VirtualHost>
```

Figure 11. Modifying httpd configuration to use SSL certificate

The modified content describes the location of the certificate and the key created a while ago (Figure 11). The adjustment also made it so the web server only listens to one specific IP address: 192.168.13.130. The HTTP method is still available, so the configuration must be altered to prevent it from responding. The global structure is located at: **/etc/httpd/conf/httpd.conf**

```
...
~~Listen 80~~
...
```

Figure 12. Modifying httpd global configuration to turn off HTTP

The HTTP method is prevented from working by deleting or commenting out the listen to port 80 line (Figure 12). This helps protect the client system from accidentally using this angle, which may harm the connection's authenticity.

3.6 Repository configuration

The repository structure will now be formed, starting with folders to store the packages. The web server's root folder should be the hosting location of the local repository; in this case, it is **/var/www/html**. This helps avoid modifying the configuration of the web server.

To create folders in the directory and any parent directories that have not existed, use the **mkdir** command with the -p option (Figure 13):

```
[repo@repository ~]$ sudo mkdir -p /var/www/html/rhel/87/basesos
[sudo] password for repo:
[repo@repository ~]$ sudo mkdir -p /var/www/html/rhel/87/appstream
[repo@repository ~]$
```

Figure 13. Creating a repository folder in the web server.

There will be no output on successful completion. To break down the logic of the repository structure, the folder tree of the web server and the spine of the repository can be seen below (Figure 14):

```
var/
├─ www/
│  ├─ html/ \\This folder is the document root of our web server
│  │  ├─ rhel/ \\This folder indicates Linux distribution
│  │  │  ├─ 87/ \\This folder dictates the distribution version
│  │  │  │  ├─ baseos \\BaseOS repository
│  │  │  │  ├─ appstream \\AppStream repository
│  │  ├─ alma/ \\Example: Alma Linux repository in the future
│  │  │  ├─ 86/ \\Example: Alma Linux version 8.6 in the future
```

Figure 14. Folder structure explanation of the repository

Finally, the web server needs to be started using the **service** command and the **start** option (Figure 15):

```
[repo@repository ~]$ sudo service httpd start
[sudo] password for repo:
Redirecting to /bin/systemctl start httpd.service
```

Figure 15. Starting the web server using the service command

Using the **systemctl** and the **enable** command, **httpd.service**, the web server is always permitted to start when the system starts (Figure 16). This allows the repository to be available at all times without intervention from end users.

```
[repo@repository ~]$ sudo systemctl enable httpd.service
Created symlink /etc/systemd/system/multi-user.target.wants/httpd.service → /usr/lib/systemd/system/httpd.service.
```

Figure 16. Enabling the web server service to start on boot

This step concludes the preliminary setup of the repository web server. This addition allows the server to become a hosting solution for the packages and files found in a repository.

3.7 Repository cloning

As mentioned, **reposync,** part of the **yum-utils** tool pack, will be used. To install yum-utils, **dnf** will be put into action (Figure 17):

```
[repo@repository ~]$ sudo dnf install yum-utils -y
[sudo] password for repo:
Updating Subscription Management repositories.
Last metadata expiration check: 1:01:58 ago on Wed 12 Apr 2023 09:25:16 AM EEST.
...
Installed:
  yum-utils-4.0.21-14.1.el8.noarch

Complete!
```

Figure 17. yum-utils installation

Dnf is modified to increase the number of parallel downloads as well as use the fastest mirror to speed up the process:

```
[repo@repository ~]$ sudo echo “max_parallel_downloads=20” >>
/etc/dnf/dnf.conf
[repo@repository ~]$ sudo echo “fastestmirror=true” >> /etc/dnf/dnf.conf
```

Figure 18. Modifying the dnf configuration to support faster download and choosing the fastest mirror

The **max_parallel_downloads** parameter accepts a maximum value of 20. Therefore, this environment will integrate that option to maximise as much bandwidth as possible (Figure 18).

The **fastestmirror** parameter will use the fastest location in Red Hat's content delivery network. This ensures the operation is as efficient as possible (Figure 18).

Finally, **reposync** is used to start cloning the public repositories, and the table bellows explain the option that will be used (Table 1):

Table 1. Reposync's options explained (Man7 n.d)

Parameter	Description
-g, --gpgcheck	Packages that fail the GPG signature check are removed
--delete	Remove local packages that do not exist on the remote repository
--releasever	Dictate Linux version compatibility when downloading packages
-m, --downloadcomps	Download and unzip comps.xml
--norepopath	Do not use the repository ID as the name when saved
--repo	Pick a specific repository to perform the operation
-p, --download-path	Where to store the cloned repository
--download-metadata	Download the remote repository metadata

While some parameters are self-explanatory, as seen in Table 1, some should be at least mentioned here for clarification. The **--norepopath** option prevents the cloning process from using the repository ID (Figure 7) as the directory name. Instead, the folder hosting the repository has already been created and will be

nominated as the folder name instead (Figure 13). This is helpful as the repository ID, by nature, is lengthy, making later processes more tedious.

The **–downloadcomps** option retrieves the **comps.xml** file from the remote repository, which contains group definitions of packages. This allows users to install bundles of applications by purpose rather than installing every application manually.

As discussed in previous chapters, the **--download-metadata** option gets the metadata from the remote repository, which indexes the repository. dnf uses this information to know which packages are available and what dependencies are needed.

Finally, the repository is ready to start being cloned:

```
[repo@repository ~]$ sudo reposync --norepopath -g -m --delete --download-
meta --releasever 8.7 -p /var/www/html/rhel/87/baseos --repo=rhel-8-for-
x86_64-baseos-rpms
[sudo] password for repo:
Updating Subscription Management repositories.
Red Hat Enterprise Linux 8 for x86_64 - B  13 kB/s | 4.1 kB     00:00
Red Hat Enterprise Linux 8 for x86_64 - B  18 MB/s | 160 MB     00:09
comps.xml for repository rhel-8-for-x86_64-baseos-rpms saved
(1/13036): libksba-1.3.5-7.el8.i686.rpm    76 kB/s | 142 kB     00:01
(2/13036): glib2-fam-2.56.4-1.el8.x86_64. 7.0 kB/s |  15 kB     00:02
(3/13036): libnfsidmap-2.3.3-14.el8.x86_6  35 kB/s | 121 kB     00:03
(4/13036): opa-fm-10.7.0.0.145-2.el8.x86_ 513 kB/s | 1.4 MB     00:02
(5/13036): python3-talloc-2.1.14-3.el8.x8 7.9 kB/s |  26 kB     00:03
...
(13033/13036): kpatch-patch-4_18_0-425_10 104 kB/s |  32 kB     00:00
(13034/13036): kpatch-patch-4_18_0-425_3_ 112 kB/s |  35 kB     00:00
(13035/13036): kpatch-patch-4_18_0-425_13 123 kB/s |  24 kB     00:00
(13036/13036): kernel-modules-4.18.0-425. 393 kB/s |  33 MB     01:26
\\The task will stop here for a while to perform gpgcheck
...
[repo@repository ~]$
```

Figure 19. Cloning an 8.7 BaseOS Red Hat repository using reposync

The process will take a while, depending on the collection of software. One thing to note is that even after finishing, reposync will take time to handle gpgcheck on every downloaded package (Figure 19). This is critical, as it is the first measure to ensure the packages are not tampered with and infect any systems beyond this.

AppStream repository will be the next in line to be cloned; the case remains the same:

```
[repo@repository ~]$ sudo reposync --norepopath -g -m --delete --download-
meta --releasever 8.7 -p /var/www/html/rhel/87/appstream --repo=rhel-8-for-
x86_64-appstream-rpms
[sudo] password for repo:
Updating Subscription Management repositories.
Red Hat Enterprise Linux 8 for x86_64 - A  15 kB/s | 4.5 kB     00:00
Red Hat Enterprise Linux 8 for x86_64 - A 7.8 MB/s |  53 MB     00:06
Last metadata expiration check: 0:00:03 ago on Thu 13 Apr 2023 08:32:57 AM
EEST.
Red Hat Enterprise Linux 8 for x86_64 - A  11 MB/s | 136 MB     00:12
comps.xml for repository rhel-8-for-x86_64-appstream-rpms saved
(1/7851): python3-prettytable-0.7.2-14.el 149 kB/s |  44 kB     00:00
(2/7851): hunspell-fur-0.20050912-16.el8. 382 kB/s | 134 kB     00:00
(3/7851): perl-Mozilla-CA-20160104-7.el8.  78 kB/s |  15 kB     00:00
(4/7851): perl-ExtUtils-Install-2.14-4.el 217 kB/s |  46 kB     00:00
(5/7851): perl-MailTools-2.20-2.el8.noarc 482 kB/s | 113 kB     00:00
...
(7848/7851): dotnet-apphost-pack-7.0-7.0. 9.6 MB/s | 4.0 MB     00:00
(7849/7851): dotnet-apphost-pack-6.0-6.0. 1.5 MB/s | 4.0 MB     00:02
(7850/7851): dotnet-sdk-7.0-7.0.105-1.el8 4.1 MB/s |  88 MB     00:21
(7851/7851): dotnet-sdk-6.0-6.0.116-1.el8 1.6 MB/s |  77 MB     00:48
\\The task will stop here for a while to perform gpgcheck
...
[repo@repository ~]$
```

Figure 20. Cloning an 8.7 AppStream Red Hat repository using reposync

At this point, the repository is almost complete. While this case study only focuses on the Red Hat repository, repsoync can be used for other repositories from Linux distributions like Alma and Rocky.

3.8 Security review

This chapter will discuss the security measures implemented to protect the environment and its nodes. While they do not actively affect the operation of this mission, they do contribute factors that help reinforce and strengthen the network as well as the systems involved.

3.8.1 Firewall configuration

Firewalls located in the DMZ must filter out traffic thoroughly as it is the last line of defence and the most effective mechanism in this topology. Therefore, it is worth discussing the rules that should be applied and the exclusions. The desired traffic control would be broken down into two segments to put it into perspective.

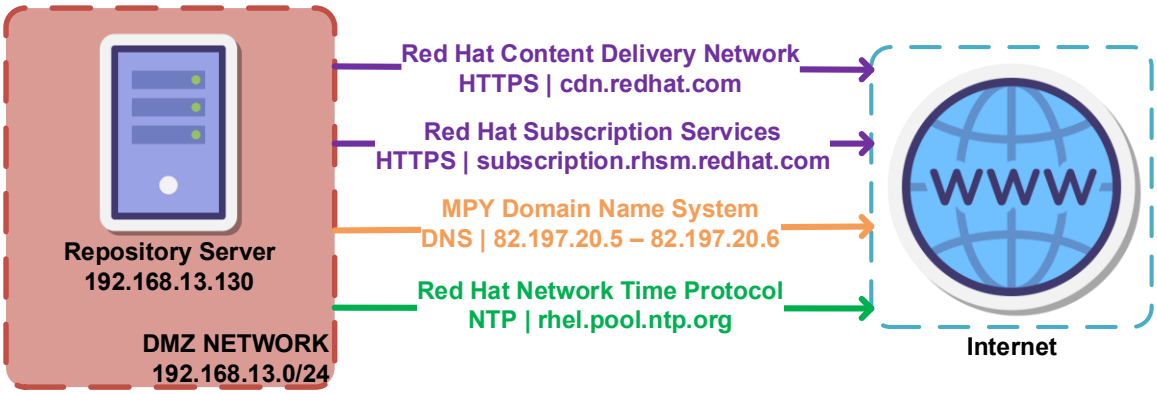


Figure 21. Security Policies between DMZ and external network

Firstly, to and from the external network, unnecessary ports and IP ranges must be restricted (Figure 21), as this is the biggest threat to the security of this project because of direct exposure to the internet. Only Red Hat Online Services are allowed access to the repository to enforce this correctly. As for Domain Name System (DNS), the ones from this project's internet provider, MPY, will be used. Moreover, for Network Time Protocol (NTP), Red Hat will be the sponsor of choice here.

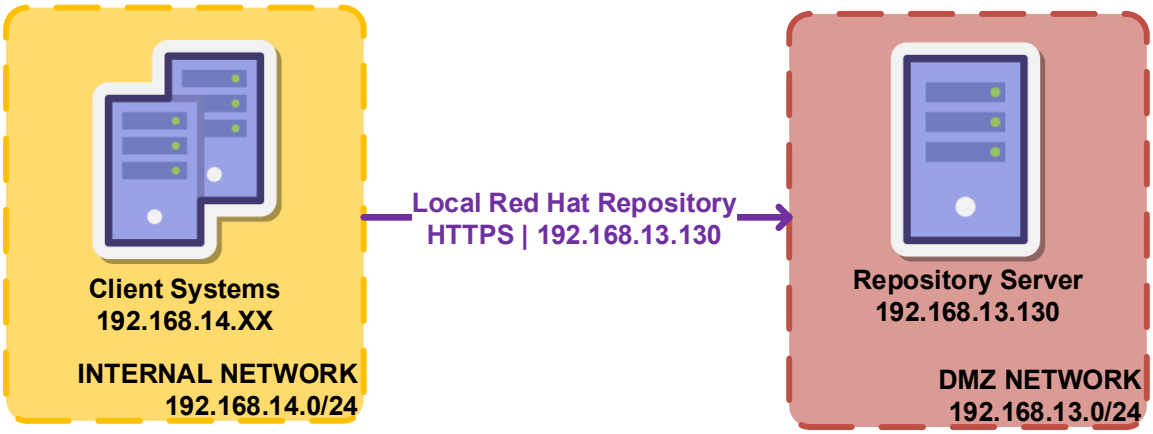


Figure 22. Security policies between internal and DMZ network

Secondly, the connections between the internal network and the buffer zone are also essential (Figure 22). The internal network will host client systems that require access to the repository to retrieve system updates and packages. To achieve maximum security, the clients only needed access to the repository through web server access. This translates to only the IP address of the repository being allowed, as well as HTTPS access.

Any other connections not mentioned here will automatically be considered denied for the sake of security and thoroughness. In later sections, these settings will be discussed on how they are put into reality with the software and physical firewall configuration.

3.8.2 Software firewall

Red Hat Linux uses firewalld as their built-in firewall for their operating system. As referenced before, the firewall must be configured with rules according to sub-chapter 3.8.1. For clarification, the interface will be renamed according to the network zone.

```
[repo@repository ~]$ ip a show
...
2: DMZ: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group
default qlen 1000
    link/ether 00:0c:29:a2:a4:55 brd ff:ff:ff:ff:ff:ff
    altname enp3s0
    inet 192.168.13.130/24 brd 192.168.13.255 scope global dynamic
noprefixroute DMZ
       valid_lft 1124sec preferred_lft 1124sec
    inet6 fe80::20c:29ff:fea2:a455/64 scope link noprefixroute
       valid_lft forever preferred_lft forever
```

Figure 23. Two network interfaces are named according to their direction

The repository interface is now identified as DMZ with an IP address of 192.16.13.130 in the 24 netmask (Figure 23). Firewalld segregates traffic by implementing zones, which then contain the network interface that inherits the rules of a zone. To see current active zones, the **firewall-cmd** tool is used:

```
[repo@repository ~]$ sudo firewall-cmd --get-active-zones
[sudo] password for repo:
public
  interfaces: DMZ
```

Figure 24. firewalld public zone and its interfaces

By default, firewalld attaches every interface to the public zone (Figure 24). A new area must be created that would accommodate its own rules:

```
[repo@repository ~]$ sudo firewall-cmd --permanent --new-zone=DMZ
[sudo] password for repo:
success
```

Figure 25. Creating the INTERNET firewall zone

One thing to note is, firewalld does not save changes after reboot without the **--permanent** option (Figure 25). This option is necessary to retain any changes made in the current session. The DMZ interface can now be placed into the DMZ zone:

```
[repo@repository ~]$ sudo firewall-cmd --zone=DMZ --change-interface=DMZ --
permanent
[sudo] password for repo:
The interface is under control of NetworkManager, setting zone to 'DMZ'.
success
```

Figure 26. Adding the INTERNAL interface into the INTERNAL zone

The process separates the DMZ interface from the public default zone and relocates it to the newly-made DMZ zone (Figure 26). The reasoning behind this is that default zone also comes with their own default rules. Therefore to minimise the amount of traffic allowed, a new area is the best practice in this case.

```
[repo@repository ~]$ sudo firewall-cmd --reload
[sudo] password for repo:
success
```

Figure 27. Putting firewall changes into effect

To make the rules go into commission, firewalld must be refreshed using the **--reload** option (Figure 27). This process will put the changes that were made into effect.

```
[repo@repository ~]$ sudo firewall-cmd --get-active-zones
[sudo] password for repo:
DMZ
  interfaces: DMZ
```

Figure 28. Active zones now show the interfaces in their respective zones.

The correct interface has now been allocated to its corresponding zone (Figure 28). The built-in firewall implements a traffic filter by explicitly allowing services, ports and sources:

```
[repo@repository ~]$ sudo firewall-cmd --zone=DMZ --list-all
[sudo] password for repo:
DMZ (active)
  target: default
  icmp-block-inversion: no
  interfaces: DMZ
  sources:
  services:
  ports:
  protocols:
  forward: no
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

Figure 29. Listing status and regulations of the INTERNAL firewall zone

As seen in Figure 29, no services, ports, or sources are currently allowed. This is, by default, the most secure state where no connections are allowed. However, for these client systems to be reachable, the web server needs to be enabled to be accessible, as well as the DNS and NTP service (Figure 30):

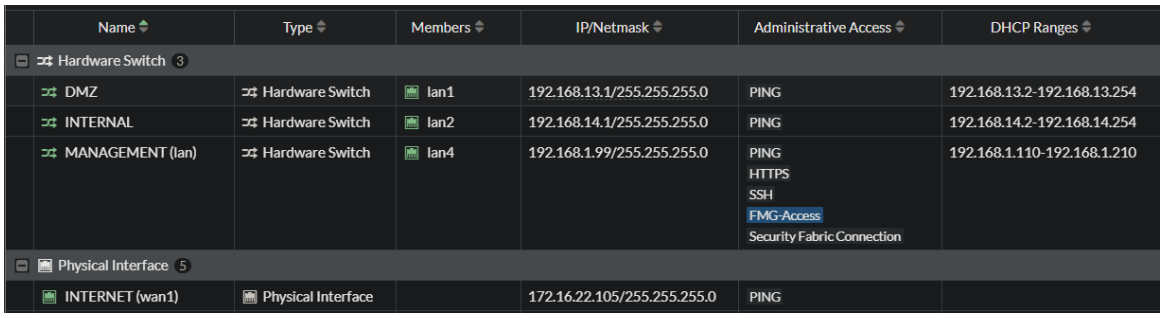
```
[repo@repository ~]$ sudo firewall-cmd --zone=INTERNAL --permanent --add-
service=https –add-service=dns -add-service=ntp
[sudo] password for repo:
success
```

Figure 30. Adding HTTPS service to the INTERNAL firewall zone

Again, for the firewall to take effect, firewalld must be reloaded(Figure 27). The goal of the software firewall is to filter as much as possible without interfering with this solution's operation. Therefore, additional policies such as IP addresses and domain names will be considered during the physical firewall configuration. At this point, the software firewall is essentially constructed. This will now only allow mandatory services.

3.8.3 Physical firewall

The physical equipment used in this case study is a Fortinet firewall. It features many of the functions discussed in sub-chapter 2.5.2. This device is a flexible variable in this demonstration as it is also subjected to change to other solutions. It is critical to understand that this chapter will only break down the finalised firewall configuration that reflects the policies discussed in sub-chapter 3.8.1. To begin with, the network hosting capability will be demonstrated according to the topology in sub-chapter 3.2:



Name	Type	Members	IP/Netmask	Administrative Access	DHCP Ranges
Hardware Switch 3					
DMZ	Hardware Switch	lan1	192.168.13.1/255.255.255.0	PING	192.168.13.2-192.168.13.254
INTERNAL	Hardware Switch	lan2	192.168.14.1/255.255.255.0	PING	192.168.14.2-192.168.14.254
MANAGEMENT (lan)	Hardware Switch	lan4	192.168.1.99/255.255.255.0	PING HTTPS SSH FMG-Access Security Fabric Connection	192.168.1.110-192.168.1.210
Physical Interface 5					
INTERNET (wan1)	Physical Interface		172.16.22.105/255.255.255.0	PING	

Figure 31. Interfaces configuration of the firewall

Figure 31 depicts the setup of virtual hardware switches and interfaces on the firewall. Specifically, physical ports lan1 and lan2 are converted to switch ports for the DMZ and INTERNAL network. The hardware switch/virtual router is assigned with the first address of the two subnets.
These ports then become the gateway for the members plugged into these ports. Because the firewall is also the router for these networks, limited administrative access to the gateway is allowed so that only the ICMP protocol can test network connectivity. As for DHCP, the virtual switch also acts as an address-distribution server. This convenience allows the client systems to acquire IP addresses and other information, such as the gateway, reducing redundant integration into the network.

Name	Details	Interface	Type	Ref.
IP Range/Subnet 8				
DMZ GATEWAY	192.168.13.1/32		Address	0
FABRIC_DEVICE	0.0.0.0/0		Address	0
FIREWALL_AUTH_PORTAL_ADDRESS	0.0.0.0/0		Address	0
MPY DNS	82.197.20.5 - 82.197.20.6		Address	1
REPOSITORY	192.168.13.130/32		Address	4
SSLVPN_TUNNEL_ADDR1	10.212.134.200 - 10.212.134.210		Address	2
all	0.0.0.0/0		Address	4
none	0.0.0.0/32		Address	0
Interface Subnet 2				
DMZ address	192.168.13.0/24	DMZ	Address	0
INTERNAL address	192.168.14.0/24	INTERNAL	Address	1
FQDN 9				
RHEL NTP	rhel.pool.ntp.org		Address	1
Red Hat CDN	cdn.redhat.com		Address	1
Red Hat Subscription Services	subscription.rhsm.redhat.com		Address	1
gmail.com	gmail.com		Address	1
login.microsoft.com	login.microsoft.com		Address	1
login.microsoftonline.com	login.microsoftonline.com		Address	1
login.windows.net	login.windows.net		Address	1
wildcard.dropbox.com	*.dropbox.com		Address	0
wildcard.google.com	*.google.com		Address	1

Figure 32. Mandatory addresses and domain names declaration

Address and domain name declarations are depicted in Figure 32. This process dictates what kind of source to attach to the policy for filtering. In the section of IP ranges, there are records of MPY's DNS servers at 82.197.20.5 and 82.197.20.6, as well as the repository's IP address at 192.168.13.130. Even though the repository is already included in the DMZ subnet entry, it is fundamental to seclude it as a dedicated entry to limit as little flowing traffic as possible. Therefore, the repository has its own entry at 192.168.13.130/32

As for the domain names, the services provided by Red Hat are recorded. In this case, the NTP service is reachable at rhel.pool.ntp.org, while the subscription service is found at subscription.rhsm.redhat.com. Furthermore, finally, the most important entry is the Red Hat CDN which will provide access to the public repository. It is available at cdn.redhat.com. The principal services must be declared before employing them into policies.

Name	Source	Destination	Schedule	Service	Action	NAT
DMZ → INTERNET (wan1) 4						
all	all	all	always	ALL	ACCEPT	Enabled
NTP	REPOSITORY	RHEL NTP	always	NTP	ACCEPT	Enabled
DNS	REPOSITORY	MPY DNS	always	DNS	ACCEPT	Enabled
Red Hat Online Services	REPOSITORY	Red Hat CDN Red Hat Subscription Services	always	HTTPS	ACCEPT	Enabled
INTERNAL → DMZ 1						
Local Repository	INTERNAL address	REPOSITORY	Repository Access	HTTPS	ACCEPT	Enabled
MANAGEMENT (lan) → INTERNAL 1						
CLIENT ACCESS	all	all	always	ALL	ACCEPT	Enabled
Implicit 1						
Implicit Deny	all	all	always	ALL	DENY	

Image 1. Policy integration in the firewall configuration

Finally, policy enforcement is the last step (Image 1). This section determines what rules are adhered to traffic passing through the firewall. Every other bit of implementation from before can be assembled to form regulations for the traffic. Before starting, it is crucial to understand that the firewall will always have a complete denial policy for other types of traffic that are not declared. This policy guarantees that only the mandatory traffic will be able to make it through the firewall. To begin with, the direction from the DMZ network to the INTERNET will be examined.

There will be three policies, and the first one is the NTP service that allows travel from the repository to the Red Hat NTP service. The protocol, in this case, is NTP. Secondly, the DNS policy allows movement from the repository to the internet provider services; this protocol is also DNS. Moreover, finally, the most important policy is the flow from the repository to Red Hat online services, which includes the subscription manager and the public repository. Because this is internet functionality, it is important to specify that only HTTPS is allowed for the best security scenario.

In the second direction, from the internal network to the DMZ, only the main policy will be looked at: traffic from the internal subnet to the repository. In the same situation as before, only HTTPS is allowed in this policy, as its only purpose is to download content from the repository for the client systems.

A standard setting in all of these policies is the support for NAT. Because every node mentioned here is located on a different network, there is a need for some form of representation when crossing network zones. Therefore, this option will be enabled for every policy. The network is now secured and monitored with these policies configured and enabled.

3.8.4 Package integrity check

As discussed before, gpgcheck needs a key to compute against the value of the package. In cases of direct cloning like this, where an RHEL system is cloning an RHEL repository, the key is already located on the system, which means external keys are not required to be downloaded. This can be proven as follow:

```
[rhel-8-for-x86_64-baseos-rpms]
name = Red Hat Enterprise Linux 8 for x86_64 - BaseOS (RPMs)
baseurl =
https://cdn.redhat.com/content/dist/rhel8/$releasever/x86_64/baseos/os
enabled = 1
gpgcheck = 1
gpgkey = file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
sslverify = 1
sslcacert = /etc/rhsm/ca/redhat-uep.pem
sslclientkey = /etc/pki/entitlement/2658196918229211024-key.pem
sslclientcert = /etc/pki/entitlement/2658196918229211024.pem
sslverifystatus = 1
metadata_expire = 86400
enabled_metadata = 1
```

Figure 33. BaseOS repository configuration in dnf

Red Hat public repository configuration can be found in **/etc/yum.repos.d/redhat.repo**. The repository ID is written in brackets in the file, indicating the section for that specific repository settings (Figure 33). Here, gpgcheck is enabled and the local location of the key. This means the entire gpgcheck process and its material are digested locally.

The protocol can be verified even in the reposync process (Figure 19 and Figure 20) in a modified version that would include debugging information using the **–d** option with a level 10 debug:

```
[repo@repository ~]$ sudo reposync --norepopath -g -m --delete --download-
meta --releasever 8.7 -p /var/www/html/rhel/87/baseos --repo=rhel-8-for-
x86_64-baseos-rpms -d 10
[sudo] password for repo:
Updating Subscription Management repositories.
Red Hat Enterprise Linux 8 for x86_64 - B  13 kB/s | 4.1 kB     00:00
Red Hat Enterprise Linux 8 for x86_64 - B  18 MB/s | 160 MB     00:09
comps.xml for repository rhel-8-for-x86_64-baseos-rpms saved
(1/13036): libksba-1.3.5-7.el8.i686.rpm    76 kB/s | 142 kB     00:01
...
(13036/13036): kernel-modules-4.18.0-425. 393 kB/s |  33 MB     01:26
Using rpmkeys executable at /bin/rpmkeys to verify signatures
Cleaning up.
```

Figure 34. gpgcheck happens at the end of the cloning procedure

In Figure 34, rpmkeys is used to compute against the signatures of the downloaded packages, ensuring that the downloaded packages are up to standard. In fact, this demonstration also happens later when packages are used for installation:

```
[main]
gpgcheck=1
installonly_limit=3
clean_requirements_on_remove=True
best=True
skip_if_unavailable=False
max_parallel_downloads=20
fastestmirror=True
```

Figure 35. dnf configuration

In dnf's configuration file, located at **/etc/dnf/dnf.conf**, the gpgcheck option is enabled by default (Figure 35). Therefore, any installation using dnf or yum will involve product value comparison to ensure integrity. Using the **--debuglevel**

option with a value of 10, an example of an installation can be observed with this procedure involved:

```
[repo@repository ~]$ sudo dnf install --debuglevel=10 createrepo
[sudo] password for repo:
...
Transaction Summary
================================================================================
Install  3 Packages

Total download size: 274 k
Installed size: 580 k
Is this ok [y/N]: y
Downloading Packages:
(1/3): drpm-0.4.1-3.el8.x86_64.rpm           259 kB/s |  68 kB     00:00
(2/3): createrepo_c-0.17.7-6.el8.x86_64.rpm  320 kB/s |  89 kB     00:00
(3/3): createrepo_c-libs-0.17.7-6.el8.x86_64 412 kB/s | 116 kB     00:00
--------------------------------------------------------------------------------
-Total                                        958 kB/s | 274 kB     00:00
Using rpmkeys executable at /bin/rpmkeys to verify signatures
...
```

Figure 36. dnf installation performing signature verification

When installing software as a demonstration, after the packages are downloaded, rpmkeys once again is requested to perform signature confirmation (Figure 36).

3.8.5 File/Folder ownership complications

One caveat arose that affects the system integrity is the interference between the web server and reposync's operation. When a reposync operation commences, any files and folder it creates are owned by the user root and group root:

```
f: /var/www/html/rhel/87/baseos/repodata/repomd.xml
dr-xr-xr-x root root /
drwxr-xr-x root root var
drwxr-xr-x root root www
drwxr-x--- root root html
drwxr-x--- root root rhel
drwxr-x--- root root 87
drwxr-x--- root root baseos
drwxr-x--- root root repodata
-rw-r----- root root repomd.xml
```

Figure 37. File/folder ownership and permission of the repository folder structure

In Figure 37, the root user owns every folder and file spanning from the root of the webserver to the metadata file in the Base OS repository. Therefore, when trying to reach the web server for this file, a permission issue surfaced:

```
[client@client ~]$ curl -k
https://192.168.13.130/rhel/87/baseos/repodata/repomd.xml
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
</body></html>
```

Figure 38. Requesting repomd.xml returns a 403 error

This is because the web server's configuration, located at httpd global configuration (/etc/httpd/conf/httpd.conf), specifies that httpd operates as the user apache of group apache (Figure 39). Therefore, as the user apache cannot access the files and folder requested, it will return an error of 403, indicating a file permission hurdle (Figure 38).

```
# If you wish httpd to run as a different user or group, you must run
# httpd as root initially and it will switch.
#
# User/Group: The name (or #number) of the user/group to run httpd as.
# It is usually good practice to create a dedicated user and group for
# running httpd, as with most system services.
#
User apache
Group apache
```

Figure 39. httpd global configuration states which user to operate on

One solution is to change the file/folder ownership to apache after every reposync run:

```
f: /var/www/html/rhel/87/baseos/repodata/repomd.xml
dr-xr-xr-x root   root /
drwxr-xr-x root   root var
drwxr-xr-x root   root www
drwxr-x--- apache root html
drwxr-x--- apache root rhel
drwxr-x--- apache root 87
drwxr-x--- apache root baseos
drwxr-x--- apache root repodata
-rw-r----- apache root repomd.xml
```

Figure 40. repository file/folder ownership belongs to apache and group root

This solution allows the web server access to the files (Figure 40). Even though this is considered acceptable, it will not be desirable to regard this as sustainable since end-users must modify it after every reposync run. Therefore, a group ownership solution is heavily preferred instead:

```
[repo@repository ~]$ sudo chgrp -R apache /var/www/html/
[sudo] password for repo:
```

Figure 41. Changing group ownership of the web server files/folder

By changing the ownership of any files and folder within the /var/www/html directory to the apache group (Figure 41), the user apache also inherits any permission that the group provides:

```
f: /var/www/html/rhel/87/baseos/repodata/repomd.xml
dr-xr-xr-x root root   /
drwxr-xr-x root root   var
drwxr-xr-x root root   www
drwxr-x--- root apache html
drwxr-x--- root apache rhel
drwxr-x--- root apache 87
drwxr-x--- root apache baseos
drwxr-xr-x root apache repodata
-rw-r--r-- root apache repomd.xml
```

Figure 42. Web server structure now owned by apache group

In Figure 42, every folder/file is now owned by the apache group. Finally, the remote client can now have access to the content while reposync can independently operate without having to mingle with permissions:

```
[client@client ~]$ curl -k
https://192.168.13.130/rhel/87/baseos/repodata/repomd.xml -I
HTTP/1.1 200 OK
Date: Sat, 15 Apr 2023 17:36:54 GMT
Server: Apache/2.4.37 (Red Hat Enterprise Linux) OpenSSL/1.1.1k
Last-Modified: Sat, 15 Apr 2023 17:35:08 GMT
ETag: "1045-5f963610d14a5"
Accept-Ranges: bytes
Content-Length: 4165
Content-Type: text/xml
```

Figure 43. HTTP status code output using curl to retrieve web server content

Finally, the web server returns a status code 200, indicating a successful attempt (Figure 43). With this adjustment in mind, the repository configuration is considered finished.

3.9 Repository update automation

Of course, over time, Red Hat's public repository will have newer updates and packages provided by Red Hat itself. Keeping the local repository up-to-date is equally important, as that is half its purpose. One solution is to log in periodically

to the repository and manually run the reposync task, which is incredibly tedious. Not to mention, it ruins the idea of this solution which is self-sustainability.

Therefore, the workaround here is to create a system service and a timer service that would run every Sunday around nighttime to avoid downtime during weekdays. The system service will allow itself to run it minimally while supporting logging in case it needs to be reviewed. Meanwhile, the timer service will run the system service on a designated schedule.

```
[Unit]
Description=Sync RHEL 8.7 BaseOS and AppStream repositories
After=network.target

[Service]
Type=simple
ExecStart=/bin/bash -c '/usr/bin/reposync --norepopath -g -m --delete --
download-meta --releasever 8.7 -p /var/www/html/rhel/87/baseos --repo=rhel-
8-for-x86_64-baseos-rpms & \
/usr/bin/reposync --norepopath -g -m --delete --download-meta --releasever
8.7 -p /var/www/html/rhel/87/appstream --repo=rhel-8-for-x86_64-appstream-
rpms & \
wait'
User=root

[Install]
WantedBy=multi-user.target
```

Figure 44. System service that updates the BaseOS and AppStream Repository

Figure 44 is a system service configuration file created and named reposync-rhel87.service located in /etc/systemd/system. These parameters in the configuration file can be explained in the table below:

Table 2. System service options explained to

Parameter	Description
Description	Human-readable information about the service
After=network.target	Advice the service manager to start only after the system has network capability
Type=simple	Service turns active mode as soon as ExecStart commits and terminates when it finished
ExecStart	Command to run with the service
User=root	Runs the service as root
WantedBy=multi-user.target	Starts the service when the system is ready for multi-user access.

From Table 2, a few crucial options should be discussed thoroughly to understand the service better. Firstly, running the service with the **network.target** option allows the automation only to proceed after the server has achieved network functionality. This is important because the operation depends on network communication heavily. Secondly, running the service as root is critical because, as demonstrated in sub-chapter 3.8.5, the process requires the root user to perform the reposync operation successfully.

In the content of the system service, the ExecStart parameter creates a shell session that executes a list of strings containing the repository sync options. The '&' symbol runs the command in the background, while the 'wait' option ensures the service remains active until all background processes are completed before marking the system service.

When making new services, systemctl needs to be reloaded to reflect upon the new services (Figure 45):

```
[repo@repository ~]$ sudo systemctl daemon-reload
[sudo] password for repo:
```

Figure 45. Reloading systemctl to discover new or changed services

In order to use the system services, the **systemctl** tool is invoked, which is used to control **systemd** services. The tool can start, stop and check the status of services. The conversion from a command line tool to a system service makes the process output minimally possible. Not to mention, it also supports checking logs via **journalctl**.

```
[repo@repository ~]$ sudo systemctl start reposync-rhel87.service
[sudo] password for repo:
```

Figure 46. Starting reposync system service using systemctl

The system service can be commenced using the start option with the systemctl tool (Figure 46). In this example, the local repository already exists and already has every package it can find in the remote repository:

```
[repo@repository ~]$ sudo systemctl status reposync-rhel87.service --no-
pager
● reposync-rhel87.service - Sync RHEL 8.7 BaseOS and AppStream repositories
   Loaded: loaded (/etc/systemd/system/reposync-rhel87.service; disabled;
vendor preset: disabled)
   Active: active (running) since Tue 2023-04-18 16:12:23 EEST; 4min 5s ago
 Main PID: 23738 (bash)
    Tasks: 3 (limit: 23469)
   Memory: 2.4G
   CGroup: /system.slice/reposync-rhel87.service
           ├─23738 /bin/bash -c /usr/bin/reposync --norepopath -g -m --dele…
           ├─23739 /usr/libexec/platform-python /usr/bin/reposync --norepop…
           └─44236 rpmkeys --checksig --root / --verbose --define=_pkgverif…
Apr 18 16:12:46 repository bash[23739]: [SKIPPED] libsmbclient-4.16.4-6.…ded
Apr 18 16:12:46 repository bash[23739]: [SKIPPED] kernel-modules-extra-4…ded
Apr 18 16:12:46 repository bash[23739]: [SKIPPED] NetworkManager-ovs-1.4…ded
Apr 18 16:12:46 repository bash[23739]: [SKIPPED] selinux-policy-mls-3.1…ded
```

Figure 47. Checking the status of system service

The system service immediately skipped downloading and moved to the GPG check protocol (Figure 47). There are traces of this in the list of processes. In this case, it is process ID 23739 belonging to reposync and process ID 43236 owned by rpmkeys. The process can be seen running in the Active field.

```
[repo@repository ~]$ sudo systemctl status reposync-rhel87.service --no-
pager
[sudo] password for repo:
● reposync-rhel87.service - Sync RHEL 8.7 BaseOS and AppStream repositories
   Loaded: loaded (/etc/systemd/system/reposync-rhel87.service; disabled;
vendor preset: disabled)
   Active: inactive (dead)
...
Apr 18 16:16:41 repository systemd[1]: reposync-rhel87.service: Succeeded
```

Figure 48. System service finished running reposync

When checking the status later, instead of a report from bash (the shell process that executes the reposync task), a message from systemd was omitted that the system service had successfully completed the task (Figure 48). The Active field now shows inactive status instead.

One thing to note is that while this does not necessarily involve the functionality of systemctl but rather reposync itself, it is still a great time to mention it due to its convenience when combined. reposync, when stopped, stores download that has already been done. If the system service were to get interrupted, starting the system service again would continue at the last download point. However, it will not help with any continuity of the GPG check process.

At this point, the reposync task has been automated successfully. However, to achieve a regular update for the repository, a timer service needs to be created that would trigger the system service every Sunday around nighttime.

```
[Unit]
Description=Sunday Repository Update

[Timer]
OnCalendar=Sun *-*-* 04:00:00
Unit=reposync-rhel87.service

[Install]
WantedBy=timers.target
```

Figure 49 Timer service configuration

Figure 49 describes the configuration of a timer service named **reposync.timer** and located in **/etc/systemd/system**. This system service indicates that it would perform the reposync-rhel87 system service every Sunday at four, as shown above.

```
[repo@repository ~]$ sudo systemctl enable reposync.timer --now
[sudo] password for repo:
Created symlink /etc/systemd/system/timers.target.wants/reposync.timer →
/etc/systemd/system/reposync.timer.
```

Figure 50. Starting the timer service immediately and on boot

To start the service timer on boot, the **enable** option is used along with the sub-option **--now** to start the service immediately (Figure 50). The status of the service timer can be observed in the following method:

```
[repo@repository ~]$ sudo systemctl status reposync.timer
[sudo] password for repo:
● reposync.timer - Sunday Repository Update
   Loaded: loaded (/etc/systemd/system/reposync.timer; enabled; vendor pres>
Active: active (waiting) since Tue 2023-04-18 17:24:37 EEST; 7s ago
  Trigger: Sun 2023-04-23 04:00:00 EEST; 4 days left

Apr 18 17:24:37 repository systemd[1]: Started Sunday Repository Update.
```

Figure 51. Status of timer service

Looking at the well-being of the service, it is currently active, and the next update will happen on Sunday 23rd of April at four in the morning (Figure 51)

On April 23rd, at around 4, the status of the reposync system service can be checked:

```
[repo@repository ~]$ sudo systemctl status reposync-rhel87.service
[sudo] password for repo:
● reposync-rhel87.service - Sync RHEL 8.7 BaseOS and AppStream repositories
   Loaded: loaded (/etc/systemd/system/reposync-rhel87.service; disabled;
vendor preset: disabled)
   Active: inactive (dead)
...
Apr 23 04:08:11 repository systemd[1]: reposync-rhel87.service: Succeeded.
```

Figure 52. reposync operating according to schedule.

Figure 52 shows the last known status of the service, indicating that the system service was performed successfully at 4:08 in the morning on April 23rd. Eight minutes of operation only accounted for no new packages or updates downloaded. In reality, this value may change depending on the public repository differences. The local repository's sustainability is guaranteed from what was achieved and will update itself periodically.

4 CLIENT SETUP/CONFIGURATION

This chapter will focus on the systems in the internal network. These are client systems that will seek content from the local repository for software and updates. This demonstration is based on the grounds that a Red Hat Linux system already exists. The client machines' configuration and hardware will not be taken into account as they depend on the scenario and requirements they were requested.

4.1 Recipient configuration

First, a repository configuration must be imported so that dnf or yum can recognise it. A resemblance of this was Figure 33, which was Linux Red Hat public repository configuration. A new repo file will be created and called **local.repo** in **/etc/yum.repos.d/** and explain what most of the options are:

```
[BaseOS]
name=Red Hat Enterprise Linux 8.7 BaseOS
enabled=1
gpgcheck=1
sslverify=0
baseurl=https://192.168.13.130/rhel/87/baseos
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release

[AppStream]
name=Red Hat Enterprise Linux 8.7 AppStream
enabled=1
gpgcheck=1
sslverify=0
baseurl=https://192.168.13.130/rhel/87/appstream
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release
```

Figure 53. Remote repository configuration

In this version of the repository configuration, the URL points toward the local repository web server instead of the public repository provided by Red Hat (Figure 53). The table below explains what other options impact the operation:

Table 3. Repository configuration parameters explained (Reselman 2022.)

Parameter	Description
[xxxxxx]	Repository ID
name	Human readable name for the repository
enabled	Disable/enable the repository
gpgcheck	Perform GPG check during installation
sslverify	Perform SSL verification when downloading
baseurl	Repository location (accepts both URL and local directory)
gpgkey	GPG key location (accept both URL and local file)

In Table 3, it is vital to point out that sslverify is a process enabled by default. When performed, the function checks the SSL certificate of the remote repository to see if it belongs to a list of trusted certificate authorities to deem the connection secure or not. Traditionally in a public-facing environment, where references

beyond this point are impossible to monitor, this process protects against man-in-the-middle (MITM) attacks. If the protocol fails, dnf will also fail to connect to the remote repository (Figure 54).

```
  - Curl error (60): Peer certificate cannot be authenticated with given CA
certificates for https://192.168.13.130/rhel/87/baseos/repodata/repomd.xml
[SSL certificate problem: self signed certificate]
Error: Failed to download metadata for repo 'BaseOS': Cannot download
repomd.xml: Cannot download repodata/repomd.xml: All mirrors were tried
```

Figure 54. DNF fails to connect to the remote repository due to certification problems

However, in the current setup, the connections span between the internal network and the DMZ. Every step along the path is monitored thoroughly, with no blind spots. This risk is heavily mitigated. As justified before, one primary purpose of SSL is to verify domain names. However, the connection has to be encrypted and secured, which is SSL's secondary purpose. Therefore, in order to make the link possible, SSL verification will be disabled, according to Figure 53.

```
[client@client ~]$ sudo dnf makecache
[sudo] password for repo:

Red Hat Enterprise Linux 8.7 BaseOS           205 kB/s | 4.1 kB     00:00
Red Hat Enterprise Linux 8.7 AppStream        539 kB/s | 4.5 kB     00:00
Metadata cache created.
```

Figure 55. dnf caching process is succesful

When disabled, dnf will skip the process while still allowing it to connect to the remote repository, all the meanwhile making the connection secure and encrypted. Using dnf's makecache option, the remote repository can be seen working by performing a caching process (Figure 55). dnf will then download metadata for all the repositories on the system. This process stores information related to packages, dependencies, and updates.

4.2 Mirror configuration

One of the main benefits of having a local repository like this is to mirror it to other portable systems. These systems can then be deployed to different sites that cannot have reliable/secure internet connection. This would provide them with security updates, which is essential for many systems that have lost touch with system updates for a long time.

Using the same configuration demonstrated in sub-chapter 4.2, 3.5 and 3.6, data can be cloned from the local repository instead of the public one. With this shortcut, the download time is significantly reduced as network connections exist within the infrastructure and can operate at a much higher speed.

Assuming a mirror system exists with the configuration mentioned above, reposync can be used to reach the local repository instead:

```
[mirror@mirror ~]$ sudo time reposync --norepopath -m --delete --download-
meta --releasever 8.7 -p /var/www/html/rhel/87/baseos --repo=BaseOS -n
[sudo] password for repo:
Red Hat Enterprise Linux 8.7 BaseOS     479 kB/s | 4.1 kB     00:00
Red Hat Enterprise Linux 8.7 BaseOS     109 MB/s | 160 MB     00:01
comps.xml for repository BaseOS saved
(1/1803): libassuan-2.5.1-3.el8.x86_64.rpm    2.5 MB/s |  83 kB     00:00
...
(1803/1803): kernel-modules-4.18.0-425.19.2.e  78 MB/s |  33 MB     00:00
1.98user 11.88system 0:18.30elapsed 75%CPU (0avgtext+0avgdata
131176maxresident)k
0inputs+3120448outputs (0major+29072minor)pagefaults 0swaps
```

Figure 56. A reposync process from the local repository with elapsed time

For comparison purposes, only the newest packages will be cloned using the **-n** option and skipping the GPG check process, as it is CPU-bound to eliminate unpredictable variables. The time command will also be used, which reports how long a function took. In Figure 56, it took 18 seconds to synchronise from the local repository.

```
[repo@repository repo]# sudo time reposync --norepopath -m --delete --
download-meta --releasever 8.7 -p /var/www/html/rhel/87/baseos --repo=rhel-
8-for-x86_64-baseos-rpms -n
[sudo] password for repo:
Updating Subscription Management repositories.
Red Hat Enterprise Linux 8 for x86_64 - BaseO  14 kB/s | 4.1 kB     00:00
Red Hat Enterprise Linux 8 for x86_64 - BaseO 8.0 MB/s | 160 MB     00:19
comps.xml for repository BaseOS saved
(1/1803): libedit-3.1-23.20170329cvs.el8.i686 341 kB/s | 106 kB     00:00
...
(1803/1803): kernel-modules-4.18.0-425.19.2.e 5.1 MB/s |  33 MB     00:06
101.49user 52.12system 5:03.57elapsed 50%CPU (0avgtext+0avgdata
145528maxresident)k
77360inputs+3120984outputs (0major+33699minor)pagefaults 0swaps
```

Figure 57. A reposync process from Red Hat's public repository with elapsed time

Contrary to the mirroring client, the local repository took 5 minutes and 3 seconds to retrieve the same content from Red Hat's public repository. This goes to show the immense benefit that the project brings, shaving at least 5 minutes already on the most minimal comparison. If the same comparison is performed, but at the grand scheme of 13000 packages, the size of the local BaseOS repository, instead of 1800 packages, the differences would be much more exponential.

4.3 Result

Overall, a stable state has been achieved where the client systems can install updates and any packages they require for their permission. For example, if the client systems need the PostgreSQL database, it can quickly install it from the local repository:

```
[client@client ~]$ sudo dnf install postgresql
[sudo] password for client:
Last metadata expiration check: 6:22:07 ago on Mon 17 Apr 2023 07:34:52 AM
EEST.
Dependencies resolved.
================================================================================
Package     Arch    Version                                    Repository Size
================================================================================
Installing:
 postgresql  x86_64  10.23-1.module+el8.7.0+17280+3a452e1f   AppStream  1.5
Installing dependencies:
 libpq       x86_64  13.5-1.el8                                AppStream  198
Enabling module streams:
 postgresql          10
Transaction Summary
================================================================================
Install  2 Packages

Total download size: 1.7 M
Installed size: 6.2 M
Is this ok [y/N]:
```

Figure 58. Installing PostgreSQL database from a local depository

Using dnf to install the PostgreSQL database, the package belongs to the local AppStream repository (Figure 58). However, in case there is a need to install a specific version of the package from a compatibility perspective:

```
[client@client ~]$ sudo dnf list --showduplicates java-1.8.0-openjdk
[sudo] password for client:
Last metadata expiration check: 3:07:11 ago on Mon 17 Apr 2023 01:58:15 PM
EEST.
Available Packages
java-1.8.0-openjdk.x86_64       1:1.8.0.201.b09-2.el8          AppStream
java-1.8.0-openjdk.x86_64       1:1.8.0.212.b04-1.el8_0        AppStream
java-1.8.0-openjdk.x86_64       1:1.8.0.222.b10-0.el8_0        AppStream
java-1.8.0-openjdk.x86_64       1:1.8.0.222.b10-1.el8          AppStream
java-1.8.0-openjdk.x86_64       1:1.8.0.232.b09-0.el8_0        AppStream
java-1.8.0-openjdk.x86_64       1:1.8.0.232.b09-2.el8_1        AppStream
...
java-1.8.0-openjdk.x86_64       1:1.8.0.342.b07-2.el8_6        AppStream
java-1.8.0-openjdk.x86_64       1:1.8.0.345.b01-1.el8_6        AppStream
java-1.8.0-openjdk.x86_64       1:1.8.0.345.b01-5.el8          AppStream
java-1.8.0-openjdk.x86_64       1:1.8.0.352.b08-2.el8_6        AppStream
java-1.8.0-openjdk.x86_64       1:1.8.0.352.b08-2.el8_7        AppStream
java-1.8.0-openjdk.x86_64       1:1.8.0.362.b09-2.el8_7        AppStream
```

Figure 59. Listing every available java-1.8.0-openjdk version available

Using dnf with the list option, every java-1.8.0-openjdk version can be found available on the local repository (Figure 59). As explained in the previous chapters, this information is organised thanks to the metadata of the repository keeping track of all the versions. The purpose of meticulously picking out a version like this is usually for environments more sensitive to software updates.

Therefore, in a typical scenario, if a specific version of java-1.8.0-openjdk would work well with the system's operation, it is equally vital that the local repository can provide packages that would guarantee such. For demonstration, if java-1.8.0-openjdk with a version of 1.8.0.342.b07-2 works better than the latest version, then it can be requested explicitly during installation using dnf:

```
[client@client ~]$  sudo dnf install java-1.8.0-openjdk-1:1.8.0.342.b07-
2.el8_6.x86_64
[sudo] password for client:
Last metadata expiration check: 3:40:55 ago on Mon 17 Apr 2023 01:58:15 PM
EEST.
Dependencies resolved.
================================================================================
Package                 Arch   Version                   Repo      Size
================================================================================
Installing:
 java-1.8.0-openjdk      x86_64 1:1.8.0.342.b07-2.el8_6 AppStream 348 k
Installing dependencies:
 alsa-lib                x86_64 1.2.7.2-1.el8           AppStream 496 k
...
Transaction Summary
================================================================================
Install  38 Packages

Total download size: 42 M
Installed size: 142 M
Is this ok [y/N]:
```

Figure 60. Installing a specific version of java-1.8.0-openjdk

Using dnf, specific versions of packages can be enquired to be installed on a system. In this case, version 1.8.0.342.b07-2 was requested, and the installation will continue normally. In another example, an old system running Red Hat Enterprise Linux will be updated from 8.5 to 8.7:

```
[client@oldclient client]# cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.5 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.5"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.5 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8::baseos"
HOME_URL="https://www.redhat.com/"
DOCUMENTATION_URL="https://access.redhat.com/documentation/red_hat_enterpris
e_linux/8/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.5
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.5"
```

Figure 61 Old Client running Red Hat Enterprise Linux 8.5

In this example, the old client runs Red Hat Enterprise Linux 8.5 (Figure 61). This can be seen in the os-release file located in the etc folder. The file gives information about system distribution, version, and build. Using dnf with the update option, the old client can be upgraded to RHEL 8.7:

```
[client@oldclient client]# dnf update -y
[sudo] password for client:
Last metadata expiration check: 0:02:24 ago on Mon 17 Apr 2023 07:28:23 PM
EEST.
Dependencies resolved.
================================================================================
Package              Arch   Version                     Repo       Size
================================================================================
Installing:
 kernel               x86_64 4.18.0-425.19.2.el8_7      BaseOS    8.9 M
Upgrading:
 NetworkManager       x86_64 1:1.40.0-6.el8_7           BaseOS    2.3 M
 NetworkManager-libnm x86_64 1:1.40.0-6.el8_7           BaseOS    1.9 M
...
Complete!
```

Figure 62 Using dnf to upgrade the old client to RHEL 8.7

In Figure 62, a kernel installation is introduced in the process. This indicates that the operating system will be upgraded from version 8.5 to 8.7. In the next section, other packages will be updated to the latest version, which is part of the BaseOS repository hosting RHEL 8.7 packages. After it is finished, the result can be seen in the following output:

```
[client@oldclient client]# cat /etc/os-release
NAME="Red Hat Enterprise Linux"
VERSION="8.7 (Ootpa)"
ID="rhel"
ID_LIKE="fedora"
VERSION_ID="8.7"
PLATFORM_ID="platform:el8"
PRETTY_NAME="Red Hat Enterprise Linux 8.7 (Ootpa)"
ANSI_COLOR="0;31"
CPE_NAME="cpe:/o:redhat:enterprise_linux:8::baseos"
HOME_URL="https://www.redhat.com/"
DOCUMENTATION_URL="https://access.redhat.com/documentation/red_hat_enterpris
e_linux/8/"
BUG_REPORT_URL="https://bugzilla.redhat.com/"

REDHAT_BUGZILLA_PRODUCT="Red Hat Enterprise Linux 8"
REDHAT_BUGZILLA_PRODUCT_VERSION=8.7
REDHAT_SUPPORT_PRODUCT="Red Hat Enterprise Linux"
REDHAT_SUPPORT_PRODUCT_VERSION="8.7"
```

Figure 63. The old client is now running RHEL 8.7

When rechecking the os-release file, the system reflected the update and now identified itself as Red Hat Enterprise Linux 8.7 (Figure 63). These results will now conclude this project's practical tasks.

5 CONCLUSION

In the end, the desired result was achieved. Looking at the repository's point of view, a few things are noted. The local repository is now a local software marketplace. It provides the same functionality as a public repository regarding the catalogue of applications and the latest updates. Even the most imperative component, security verification, can be replicated just like an internet installation. That is, in fact, the essence of what this homemade solution was expected to replicate. Even better is the automation of syncing with the public repository periodically. This process allows the local repository to keep itself up to date.

On the other hand, many security criteria were cleared from the client system's perspective. First, the systems are secluded in their own network with no direct connection to the internet. This factor satisfies many standards and requirements from customers. Secondly, the systems have access to a local repository that provides the latest security updates as well as any extra software it may need for its operation.

Finally, from the abstract point of view, a network solution is formed that segregates members according to their roles. This alone already quenches many network risks. However, extra measures were committed for traffic filtration that limits the maximum amount of communications between nodes as well as the internet. Not to mention systems even come with their own defence instruments to help further negate hazards.

Overall, the idea and concept of an independent source for software update and installation is not limited to just Red Hat Enterprise Linux. However, the theory and manoeuvres that were implemented can also be applied to many other solutions. It is crucial to extract the content down to the purpose level, such as repository syncing, web server hardening and network circulation filtration. By covering every angle thoroughly and the notion behind the steps performed here, it can be transferred into future projects requiring the same criterion. This project is a foundation of theory and application that can also be applied to other hosting complications that may arise.

REFERENCE

Fedora Project. n.d. Using the DNF software package manager. Web page. Available at: https://docs.fedoraproject.org/en-US/quick-docs/dnf/ [Accessed 17 May 2023]

Firewalld. n.d. Available at: https://firewalld.org/ [Accessed 17 May 2023]

Fortinet. n.d. DMZ Networks. Web page. Available at: https://www.fortinet.com/resources/cyberglossary/what-is-dmz [Accessed 16 May 2023]

Fortinet. n.d. What is a Hardware Firewall? Hardware vs. Software Firewalls. Web page. Available at: https://www.fortinet.com/resources/cyberglossary/hardware-firewalls-better-than-software# [Accessed 18 May 2023]

Java T Point. n.d. How to Install Httpd in Ubuntu? Web page. Available at: https://www.javatpoint.com/how-to-install-httpd-in-ubuntu [Accessed 17 May 2023]

Gab-Momoh, R. 2021. What are the differences between dnf and apt package managers?. Tech Direct Archive. Web page. Available at: https://techdirectarchive.com/2021/11/19/the-differences-between-dnf-and-apt-package-managers/ [Accessed 16 May 2023]

Lauber S. 2020. RPM and GPG: How to verify Linux packages before installing them. Red Hat. Web page. Available at: https://www.redhat.com/sysadmin/rpm-gpg-verify-packages# [Accessed 18 May 2023]

Man7. n.d. reposync(1) — Linux manual page. Available at: https://man7.org/linux/man-pages/man1/reposync.1.html# [Accessed 17 May 2023]

Okta. 2023. DMZ Network: What Is a DMZ & How Does It Work?. Okta. Web page. Available at: https://www.okta.com/identity-101/dmz/ [Accessed 16 May 2023]

Red Hat. n.d. Customers. Web page. Available at: https://www.redhat.com/en/customers [Accessed 17 May 2023]

Red Hat Customer Portal. n.d. Chapter 4. Content Delivery Network (CDN) Structure. Web page. Available at: https://access.redhat.com/documentation/en-us/red_hat_satellite/6.2/html/architecture_guide/chap-red_hat_satellite-architecture_guide-content_delivery_network_structure [Accessed 16 May 2023]

Red Hat Customer Portal. n.d. Chapter 10. Managing Services with systemd. Web page. Available at: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/chap-managing_services_with_systemd [Accessed 19 May 2023]

Reselman, B. 2022. Moving from apt to dnf package management. Red Hat Developer. Web page. Available at: https://developers.redhat.com/articles/2022/10/07/move-apt-dnf-package-management# [Accessed 16 May 2023]

Reselman, B. 2022. What's inside an RPM .repo file? Red Hat Developer. Web page. Available at: https://developers.redhat.com/articles/2022/10/07/whats-inside-rpm-repo-file# [Accessed 16 May 2023]

LIST OF FIGURES