**Executive Summary**

In this case study, the task was to configure internal firewall rules on a Windows server within the Wonderville Town network using PowerShell. The goal was to automate the setup of firewall rules and ensure that these rules could be deployed across multiple hosts. This case study focuses on configuring firewall rules to secure communication between three machines: a Windows server, a Windows 10 machine, and an Ubuntu device. The goal was to ensure that only authorized services like SSH, HTTP, HTTPS, and RDP were allowed while blocking all other traffic. The firewall was configured using PowerShell to implement these security measures, and the rules were tested for effectiveness. The report includes an overview of the technical solution, evidence from testing, and recommendations for deploying similar configurations in organizational environments.  
  
**Network topology:**

A screenshot of a computer

Description automatically generated

**Technical Solution**

The configuration steps were carried out using PowerShell on a Windows Server acting as the gateway, and a Windows 10 machine and Ubuntu device, which were connected to the server.

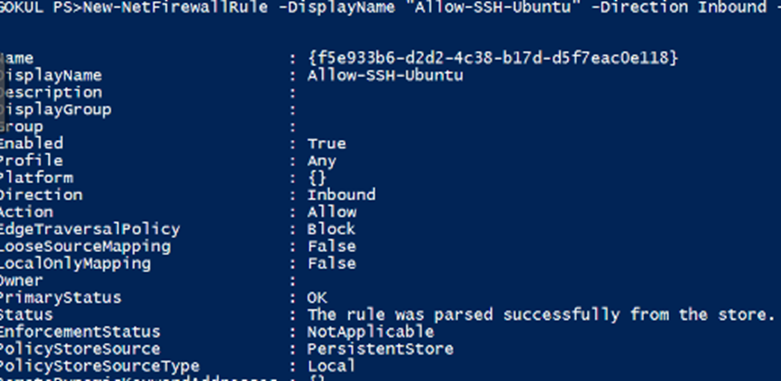
The firewall configuration was automated using PowerShell commands. The steps involved were:  
**Enabling the Firewall Profiles**: We enabled all the firewall profiles (Domain, Public, and Private) using the following PowerShell command:

**Set-NetFirewallProfile -Profile Domain, Public, Private -Enabled True**  
 A screenshot of a computer

Description automatically generated

**SSH Rule:**

A firewall rule was created to allow SSH traffic from specific IP addresses (between Ubuntu and the server). (*see attachment SSH.txt below for the full script*)



**HTTP and HTTPS Rules:**

Two firewall rules were configured to allow HTTP (port 80) and HTTPS (port 443) traffic. (*see attachment HTTP.txt and HTTPS.txt below for the full script*)

A computer screen shot of a blue screen

Description automatically generatedA screen shot of a computer

Description automatically generated

**RDP Rule**:

A rule was created to allow Remote Desktop (RDP) on port 3389 for the Windows Server and Windows 10 machine. (*see attachment RDP.txt below for the full script*)

A computer screen shot of a blue screen

Description automatically generated

**Blocking All Other Traffic**:

A rule was implemented to block any other traffic that did not match the allowed service rules. (*see attachment block all other inbound traffic.txt below for the full script*)

A computer screen shot of a blue screen

Description automatically generated

**Verifying the Rules:** The firewall rules were verified by listing all the created rules:

**Get-NetFirewallRule | Format-Table -Property Name, DisplayName, Enabled, Direction, Action, Profile** A computer screen shot of a blue screen

Description automatically generated

**Evidence**

All tests were conducted in a virtual lab environment provided by the Cyber Range.

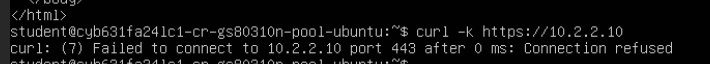
**SSH Test**  
Successfully connected from Ubuntu to the Windows Server using SSH. The connection was validated by running the command ssh student@<IP>

A computer screen with white text

Description automatically generated

**HTTP/HTTPS Test**  
HTTP traffic was tested using curl and verified using a web browser in windows 10.  
 A screenshot of a computer

Description automatically generated

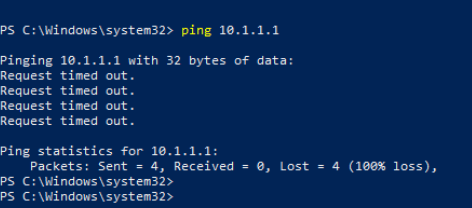
The HTTPS test indicated this will try to establish a connection to port 443 even without an SSL certificate. If the connection is refused but you get a response, it means the port is open, and the firewall rule works. If the connection times out, the port might not be accessible.  


**RDP Test**  
Remote Desktop was verified by successfully connecting from the Windows 10 machine to the Windows Server through Remote desktop connection. Where it asked for username and password.

A computer screen shot of a computer

Description automatically generated

**Block All Traffic Test**  
After implementing the block-all rule, all traffic apart from the allowed services (SSH, HTTP, HTTPS, and RDP) was blocked. This was confirmed using ping tests and failed connections on unallowed ports.



**Recommendations**

The firewall configuration effectively secures the server by only allowing necessary traffic for specific services. This setup provides a robust security layer for small organizational networks where internal resources must be protected from external threats.

**Benefits**

* + Protects critical services like SSH and RDP from unauthorized access.
  + Reduces the attack surface by blocking unnecessary traffic.
  + Centralizes control using PowerShell automation for easy firewall configuration.

**Challenges**

* + Proper configuration of HTTPS would require SSL certificates.
  + Monitoring and logging are needed to track attempts to bypass the firewall
  + Some services like HTTPS require additional configuration before they can be fully enabled.
  + Periodic review of firewall rules might be needed as the organization’s access control policies evolve

**Case Reflection**

During this case study, I learned how to configure and troubleshoot firewall rules using PowerShell. The importance of properly testing rules became evident when testing HTTPS, as the lack of SSL configuration blocked the service despite the rule being in place. I faced challenges while verifying RDP and learned the significance of service dependencies. Overall, this experience improved my understanding of host security management in a virtualized environment.

**Assumptions:**

• All remote machines have similar access needs and firewall policies, which may not always hold in real scenarios.

• HTTPS services would behave the same as HTTP without extra configurations, which proved to be incorrect.

**Attachments**

PowerShell Script for SSH, HTTP, HTTPS, RDP, and blocking all other traffic.