Windows Firewall Lab – Blocking & Allowing Ports

Introduction:

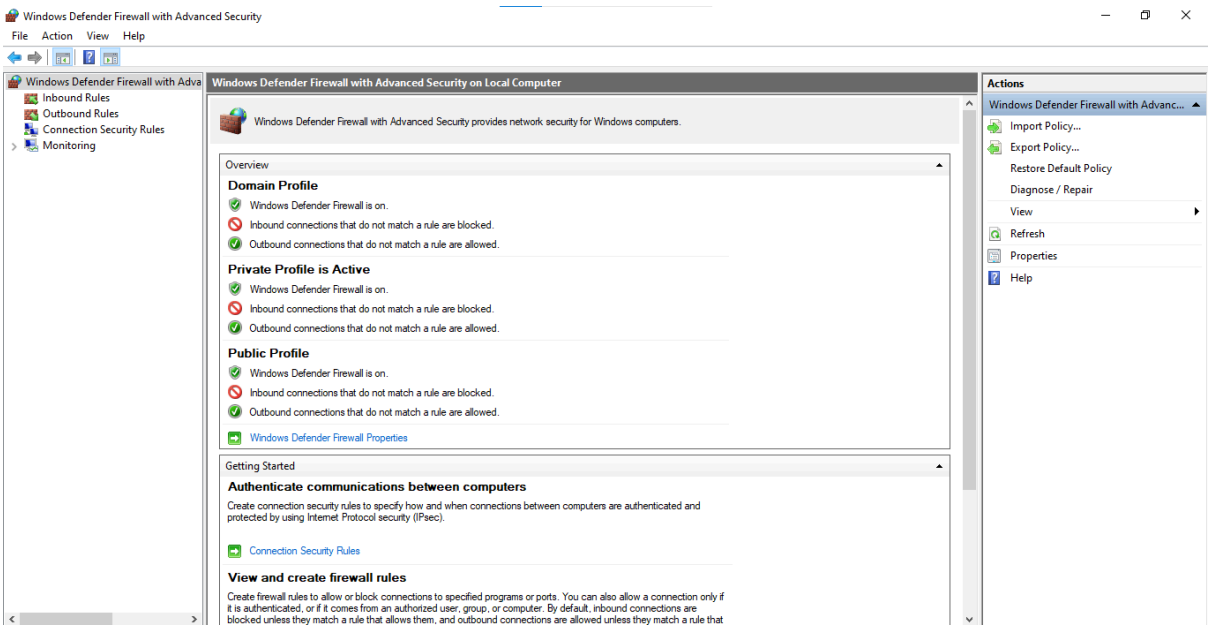
A firewall acts as a **gatekeeper** for your PC. Every incoming or outgoing network request is checked against a set of rules. Each rule has conditions (protocol, port, program, source/destination address) and an action (Allow or Block). Matching packets are either dropped or allowed. If nothing matches, the default profile policy applies.

This lab demonstrates blocking an insecure port (Telnet 23) and optionally allowing a secure port (SSH 22) for controlled access. Steps involved below:

1 Open the Advanced Firewall Console

Steps:

- Press **Win + R**, type wf.msc, press **Enter**
- OR Start → type **Windows Defender Firewall with Advanced Security** → open it



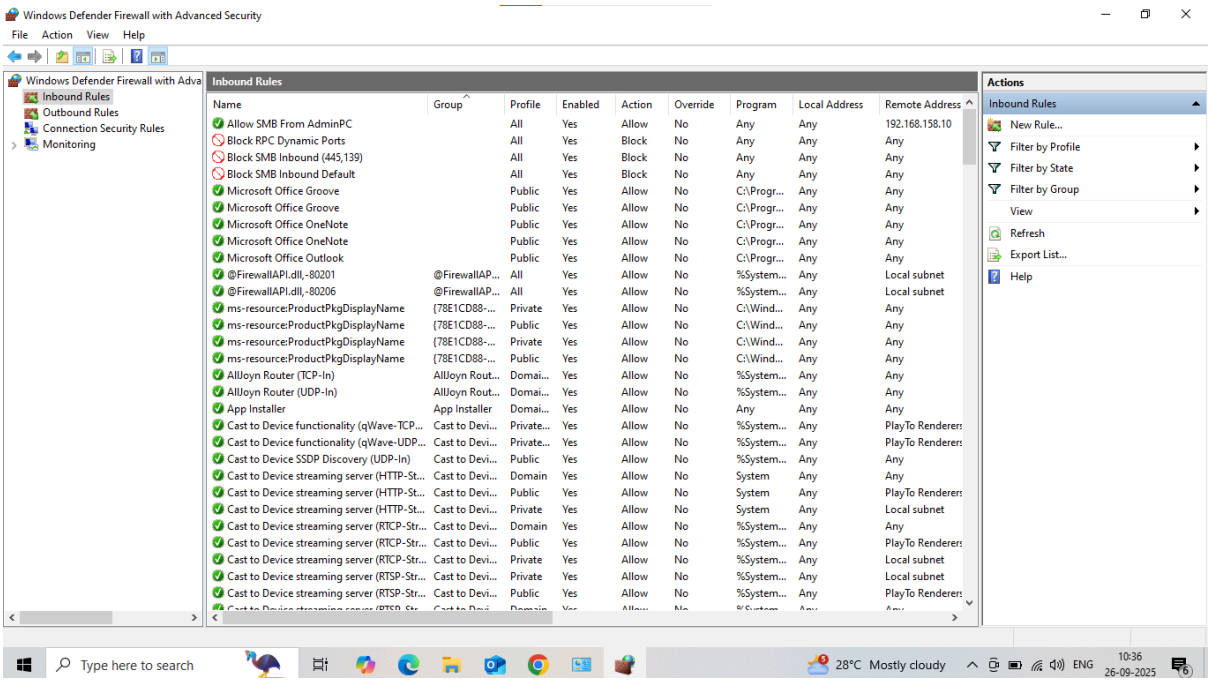
Why: This console provides granular control over ports, programs, profiles, and logging. The standard Settings page is simpler but less flexible.

---

2 Inspect Current Inbound Rules

Steps:

- In the left pane, click **Inbound Rules**
- View rules in the center pane (Name, Enabled, Action, Profile, Local Port, etc.)
- Filter or search for specific rules (e.g., Telnet or port number)

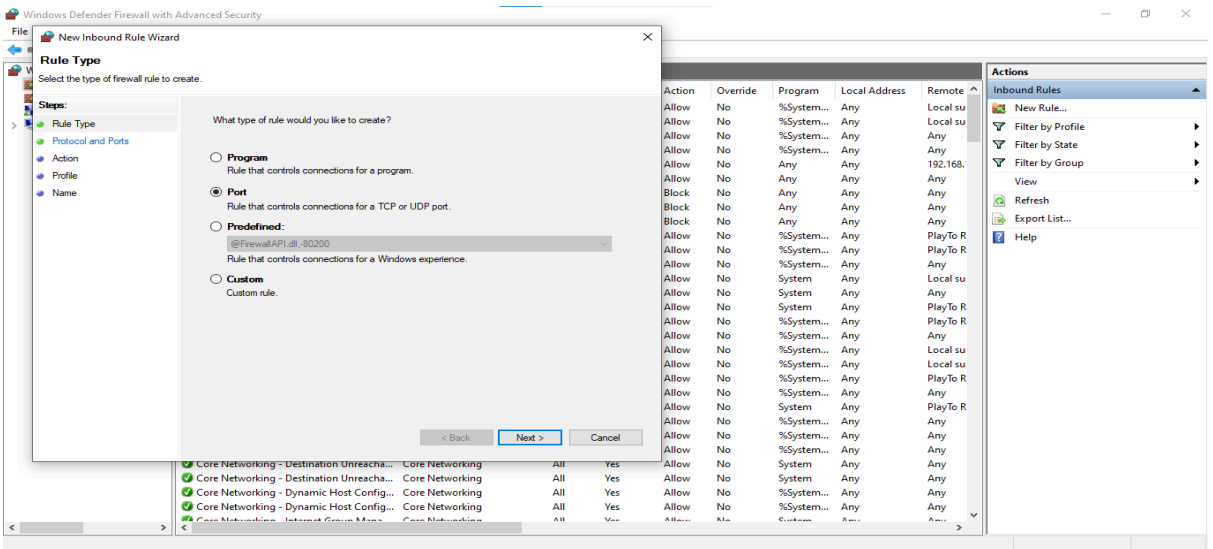


Why: Avoid duplicating rules and identify any existing rules that may override new ones.

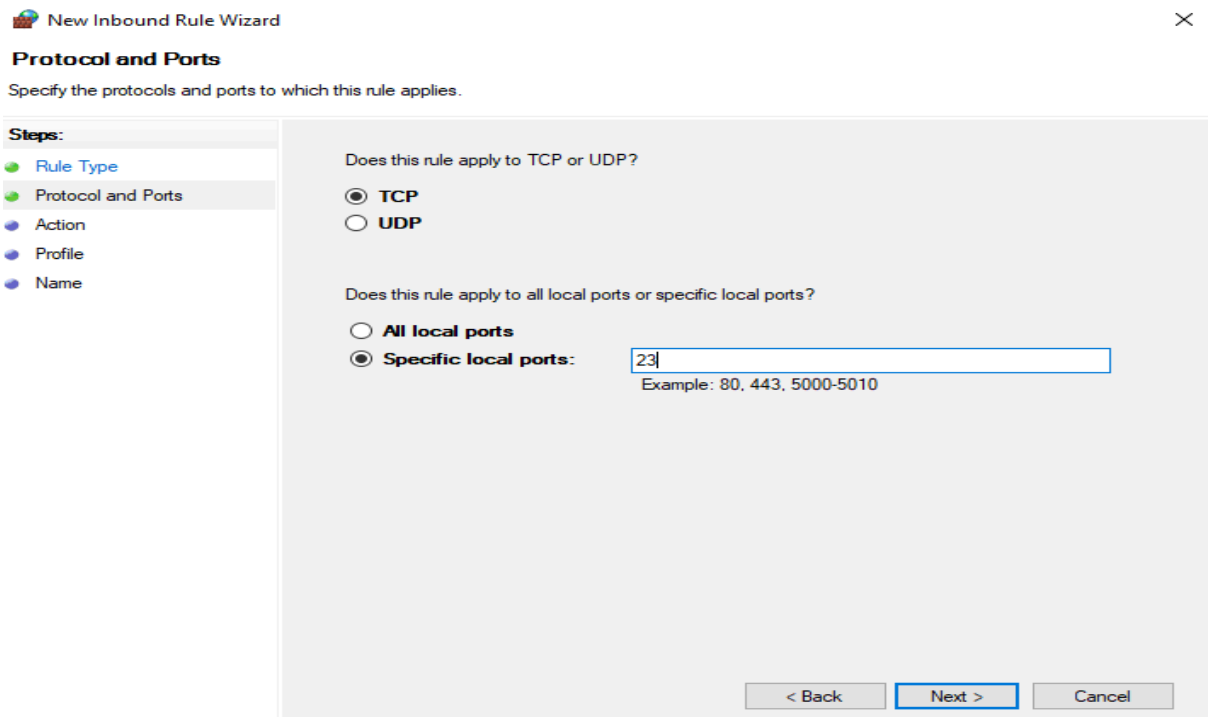
3️⃣ Create a Rule to Block Telnet (TCP Port 23)

Steps:

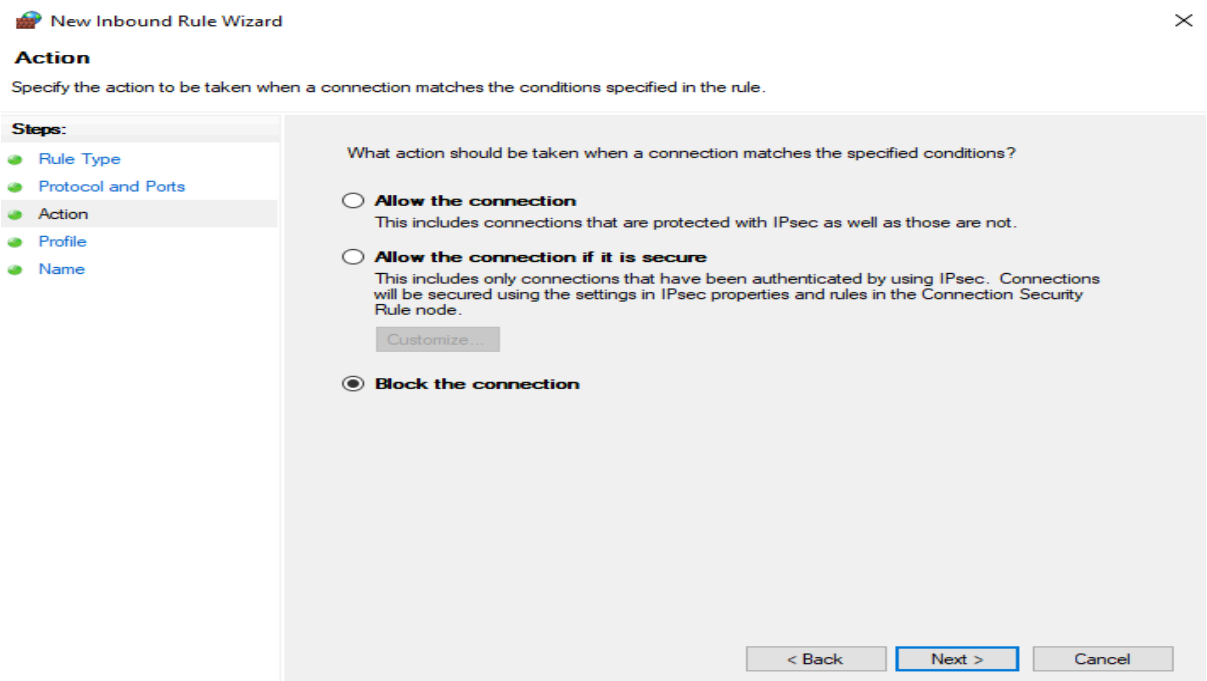
1. In **Inbound Rules**, click **New Rule…**
2. Select **Port** → **Next**



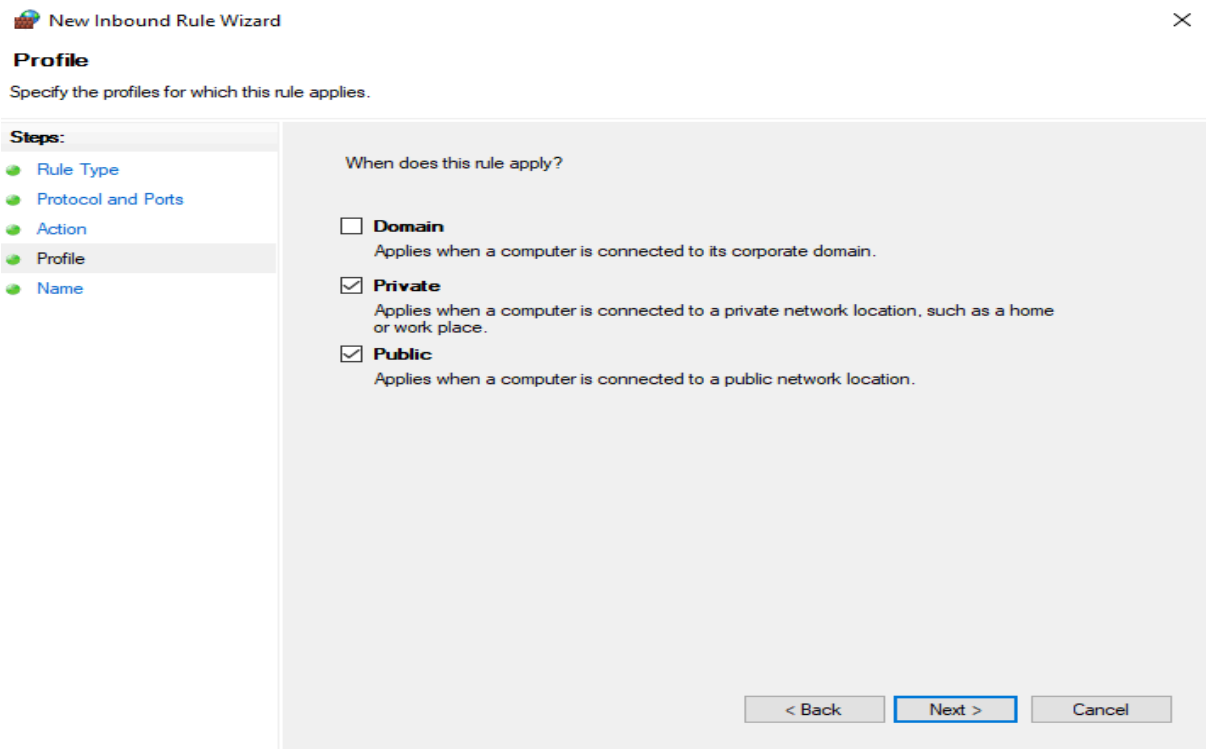
3. Select **TCP** → **Specific local ports:** 23 → **Next**



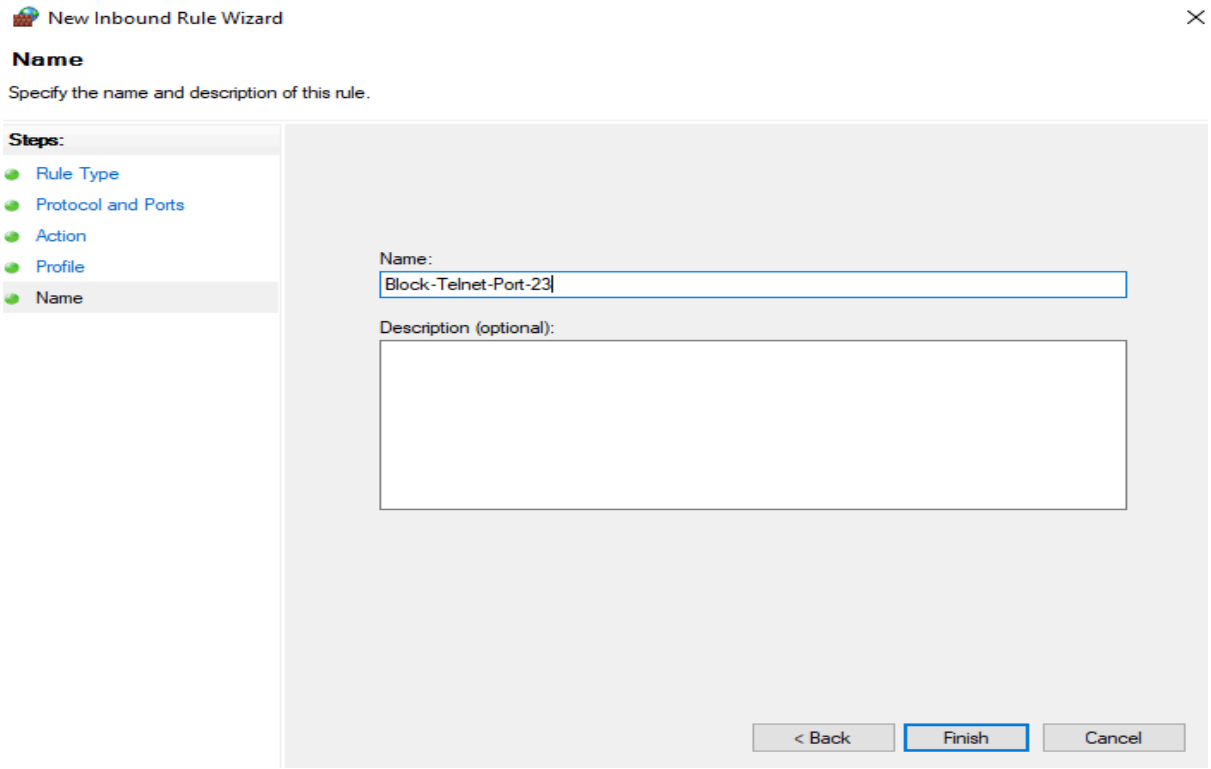
4. Choose **Block the connection** → **Next**



5. Select profiles (Private, Public, Domain if applicable) → **Next**



6. Name the rule **Block-Telnet-Port-23** → **Finish**



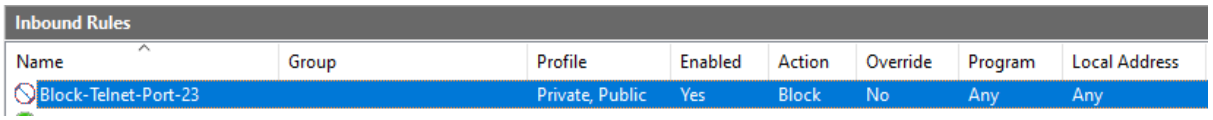
Why: Telnet transmits credentials in plaintext and is insecure. Blocking it reduces the attack surface.

---

4 Verify the Rule

Steps:

- Find **Block-Telnet-Port-23** in Inbound Rules
- Ensure **Enabled = Yes**, **Action = Block**



- Right-click → **Properties** to inspect ports, profiles, and scope

Why: Verification ensures the rule is applied correctly and can be fine-tuned later.

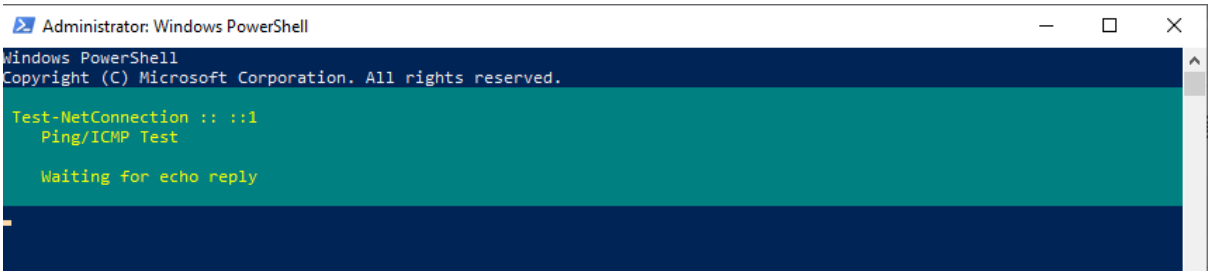
---

5 Test the block (baseline → after comparison)

Important: If no service is listening on port 23, a connection attempt will fail regardless of the firewall. Therefore, it's essential to test **before and after creating the firewall rule** to verify its effect.

Baseline Test (before creating the rule):
Open PowerShell (normal or admin) and run:

Test-NetConnection -ComputerName localhost -Port 23




```
PS C:\Windows\system32> Test-NetConnection -ComputerName localhost -Port 23
WARNING: TCP connect to (::1 : 23) failed
WARNING: TCP connect to (127.0.0.1 : 23) failed


ComputerName            : localhost
RemoteAddress           : ::1
RemotePort              : 23
InterfaceAlias          : Loopback Pseudo-Interface 1
SourceAddress           : ::1
PingSucceeded           : True
PingReplyDetails (RTT)  : 0 ms
TcpTestSucceeded        : False
```

Or, from another machine on the same network:

Test-NetConnection -ComputerName <windows-ip> -Port 23

Interpretation:

- TcpTestSucceeded: True → A service is listening and reachable.
- TcpTestSucceeded: False → Either no service is running or the connection is blocked.

After applying Block-Telnet-Port-23:
Run the same command again.

- If it was **True before** and now **False**, the firewall rule is working.
- If it was **False both times**, no Telnet service was running — the block is correct, but you can't verify it without a listening service.

Alternative manual test using Telnet client:

telnet localhost 23

```
PS C:\Windows\system32> telnet localhost 23
Connecting To localhost...Could not open connection to the host, on port 23: Connect failed
```

- Connection succeeds → service exists
- Connection refused/timeout → blocked or no service

Why: Testing both before and after confirms that the firewall caused the change. Testing only after is ambiguous.

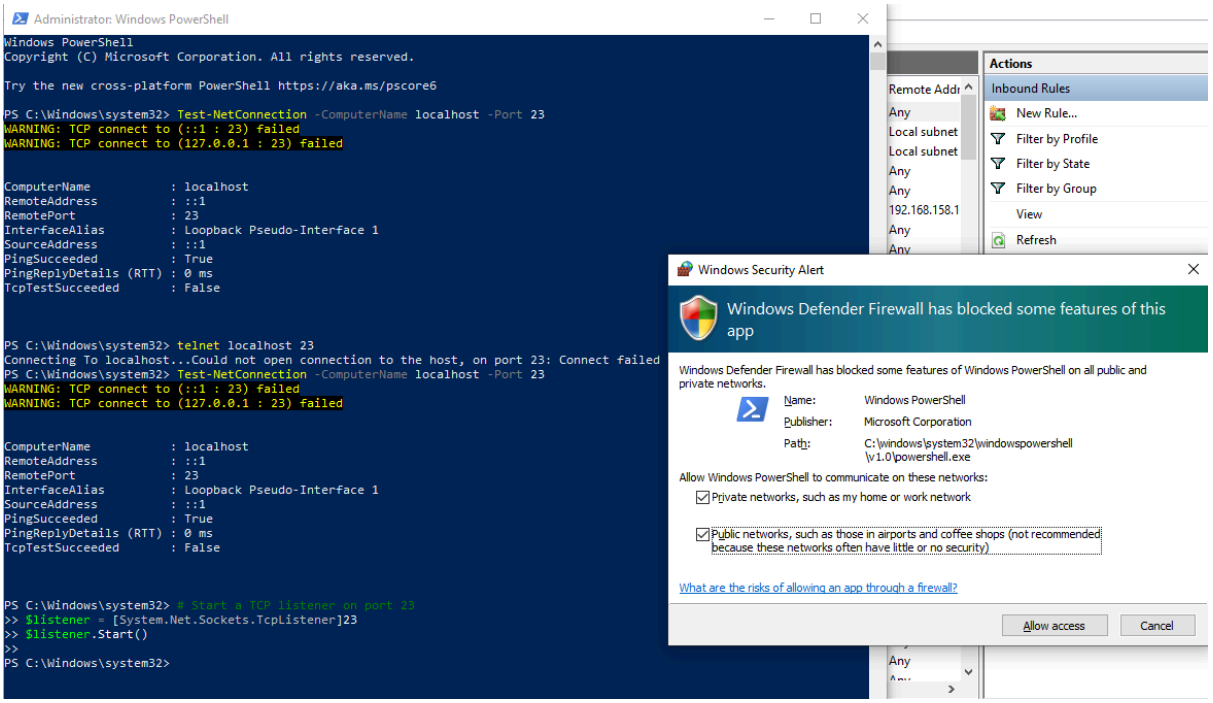
---

6 Optional: Test with a Temporary Local Listener

Note: Never expose a Telnet server to the internet (insecure). For local testing only, you can temporarily run a local listener on port 23:

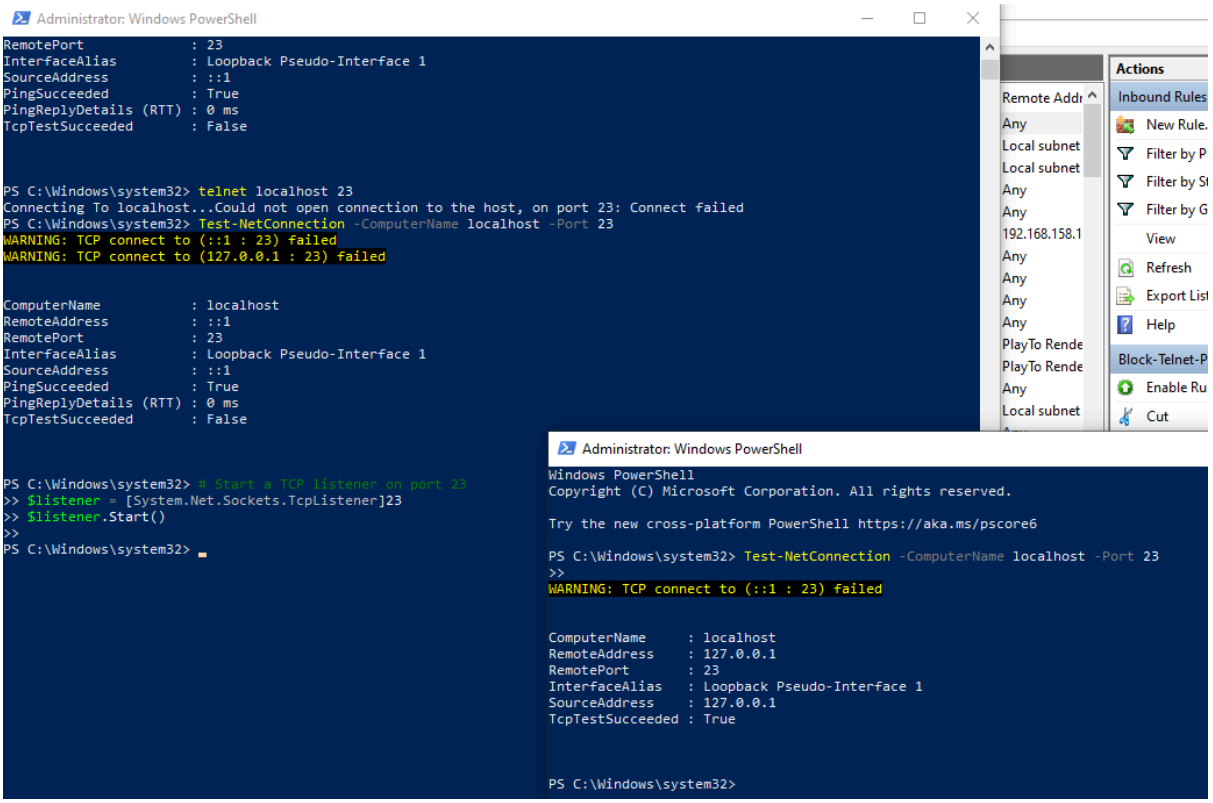
Start listener (PowerShell):

```
$listener = [System.Net.Sockets.TcpListener]23
$listener.Start()
```



Test connectivity in another PowerShell window:

```
Test-NetConnection -ComputerName localhost -Port 23
```



- With no firewall rule → TcpTestSucceeded: True
- With Block-Telnet-Port-23 enabled → TcpTestSucceeded: False

Stop the listener after testing:

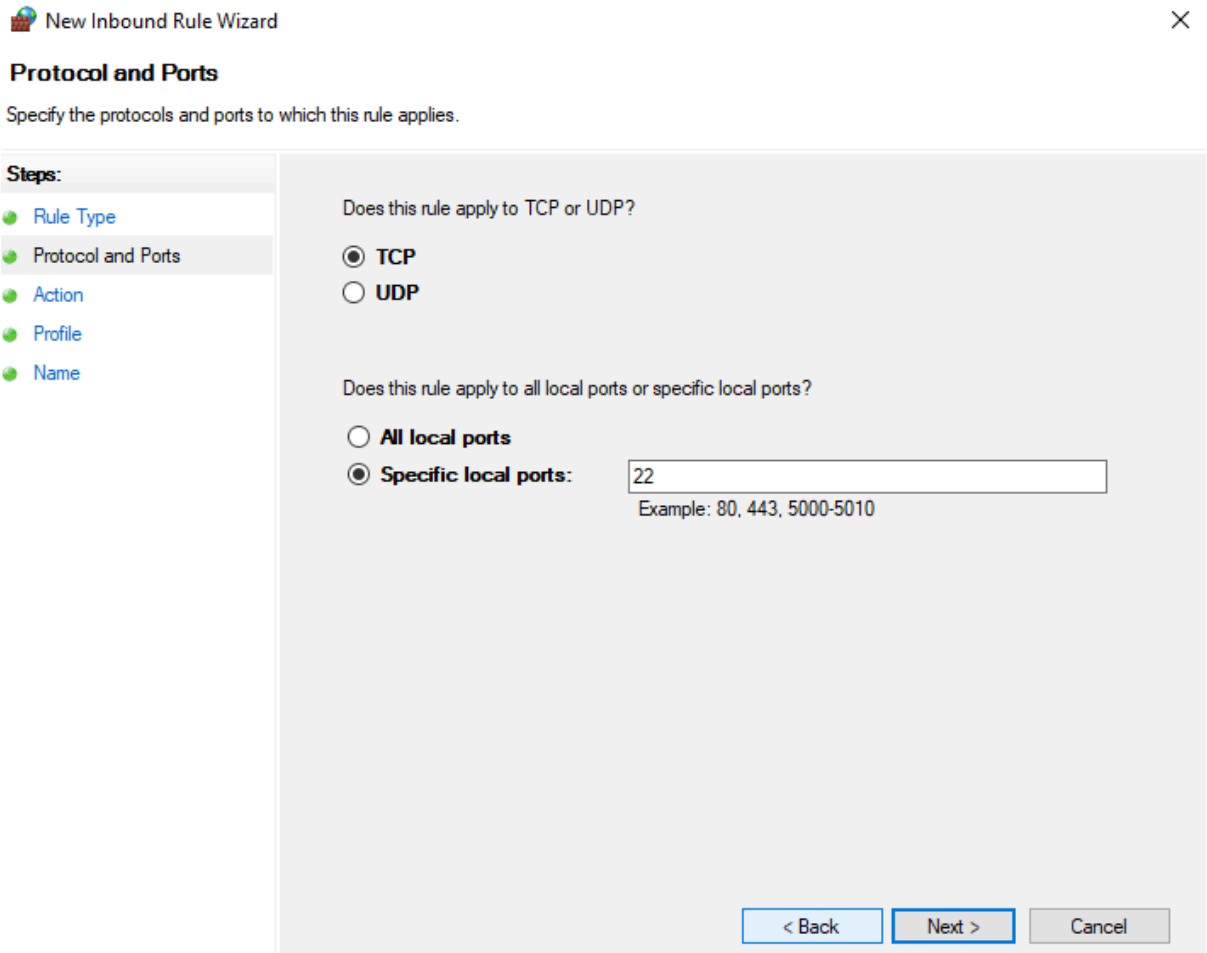
$listener.Stop()

Reminder: After testing, ensure your original firewall settings are restored.

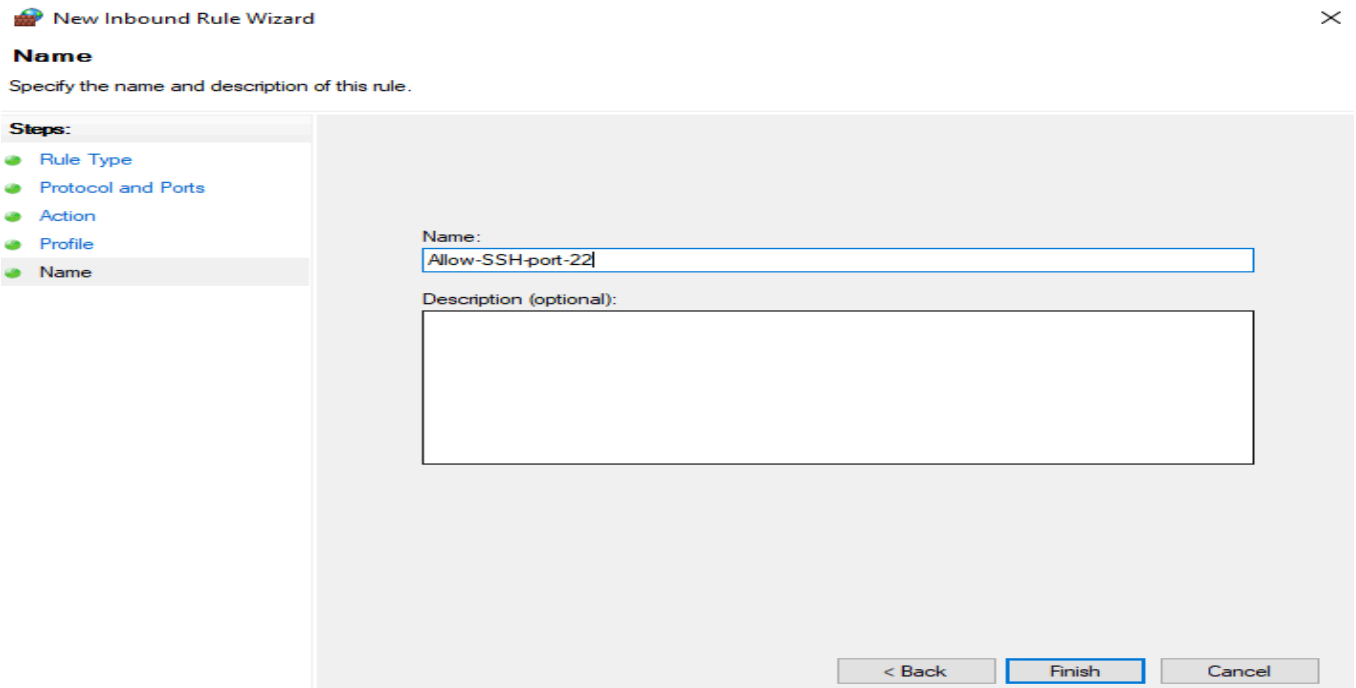
7️⃣ Allow SSH (TCP Port 22)

Steps:

- Inbound Rules → **New Rule…** → Port → TCP → Specific port: 22 → Next



- Choose **Allow the connection** → select profiles → Name: **Allow-SSH-Port-22** → Finish



Inbound Rules

Name	Group	Profile	Enabled	Action	Override	Program
Allow-SSH-port-22		Private...	Yes	Allow	No	Any

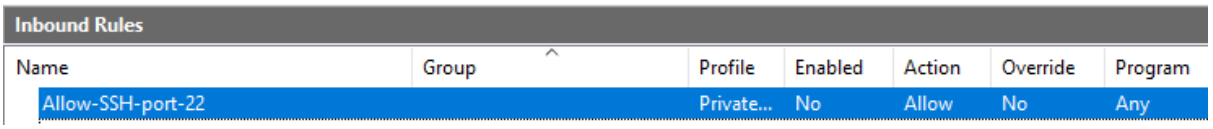
Why: If SSH is needed for remote admin, explicitly allow only required ports to maintain security.

---

8 Disable or Delete Rules

Steps:

- **Disable:** Right-click → **Disable Rule** → temporarily stop rule



Inbound Rules

Name	Group	Profile	Enabled	Action	Override	Program
Allow-SSH-port-22		Private...	No	Allow	No	Any

- **Delete:** Right-click → **Delete** → permanently remove

Why: Disabling is safer for testing; deletion removes rule completely.

---

9 Enable Firewall Logging

Steps:

- Right-click **Windows Defender Firewall with Advanced Security** → **Properties**
- Under each profile (Domain/Private/Public) → **Logging** → **Customize**
- Set **Log dropped packets = Yes**, specify log file path → **OK** → **Apply**

Why: Captures blocked packets for verification and troubleshooting.

---

10 Export Rule Set

Steps:

- In the console top menu → **Action** → **Export Policy…**
- Save as .wfw or CSV for sharing and documentation
- Or else export a csv file with limited columns of data to share in a public repo.

Why: Provides a snapshot of rules for report submission and later use.

Conclusion

- Firewall rules let you **control inbound/outbound traffic** by port, protocol, program, and scope.
- Blocking insecure ports (like Telnet 23) and allowing necessary services (SSH 22) reduces the attack surface.
- Verification through **PowerShell tests or temporary listeners** ensures the rules are functioning correctly.
- Logging and exporting rules provide evidence and help maintain security configurations.
