UpGuard™

HANDBOOK

10 Steps for Configuring a New Server

The Server Checklist

That's a nice new Linux server you got there...

OVERVIEW

That's a nice new Linux server you got there... it would be a shame if something were to happen to it. It might run okay out of the box, but before you put it in production, there are 10 steps you need to take to make sure it's configured securely. The details of these steps may vary from distribution to distribution, but conceptually they apply to any flavor of Linux. By checking these steps off on new servers, you can ensure that they have at least basic protection against the most common attacks.

1. USER CONFIGURATION

The very first thing you're going to want to do, if it wasn't part of your OS setup, is change the root password. This should be self-evident, but can be surprisingly overlooked during a routine server setup. The password should be at least 8 characters, using a combination of upper and lowercase letters, numbers and symbols. You should also set up a password policy that specifies aging, locking, history and complexity requirements if you are going to use local accounts. In most cases you should disable the root user entirely and create non-privileged user accounts with sudo access for those who require elevated rights.

2. NETWORK CONFIGURATION

One of the most basic configurations you'll need to make is to enable network connectivity by assigning the server an IP address and hostname. For most servers you'll want to use a static IP so clients can always find the resource at the same address. If your network uses VLANs, consider how isolated the server's segment is and where it would best fit. If you don't use IPv6, turn it off. Set the hostname, domain and DNS server information. Two or more DNS servers should be used for redundancy and you should test nslookup to make sure name resolution is working correctly.

3. PACKAGE MANAGEMENT

Presumably you're setting up your new server for a specific purpose, so make sure you install whatever packages you might need if they aren't part of the distribution you're using. These could be application packages like PHP, MongoDB, ngnix or supporting packages like pear. Likewise, any extraneous packages that are installed on your system should be removed to shrink the server footprint. All of this should be done through your distribution's package management solution, such as yum or apt for easier management down the road.

4. UPDATE INSTALLATION AND CONFIGURATION

Once you have the right packages installed on your server, you should make sure everything is updated. Not just the packages you installed, but the kernel and default packages as well. Unless you have a requirement for a specific version, you should always use the latest production release to keep your system secure. Usually your package management solution will deliver the newest supported version. You should also consider setting up automatic updates within the package management tool if doing so works for the service(s) you're hosting on this server.

5. NTP CONFIGURATION

Configure your server to sync its time to NTP servers. These could be internal NTP servers if your environment has those, or external time servers that are available for anyone. What's important is to prevent clock drift, where the server's clock skews from the actual time. This can cause a lot of problems, including authentication issues where time skew between the server and the authenticating infrastructure is measured before granting access. This should be a simple tweak, but it's a critical bit of reliable infrastructure.

6. FIREWALLS AND IPTABLES

Depending on your distribution, iptables may already be completely locked down and require you to open what you need, but regardless of the default config, you should always take a look at it and make sure it's set up the way you want. Remember to always use the principle of least privilege and only open those ports you absolutely need for the services on that server. If your server is behind a dedicated firewall of some kind, be sure to deny everything but what's necessary there as well. Assuming your iptables/firewall IS restrictive by default, don't forget to open up what you need for your server to do its job!

7. SECURING SSH

SSH is the main remote access method for Linux distributions and as such should be properly secured. You should disable root's ability to SSH in remotely, even if you disabled the account, so that just in case root gets enabled on the server for some reason it still will not be exploitable remotely. You can also restrict SSH to certain IP ranges if you have a fixed set of client IPs that will be connecting. Optionally, you can change the default SSH port to "obscure" it, but honestly a simple scan will reveal the new open port to anyone who wants to find it. Finally, you can disable password authentication altogether and use certificate based authentication to reduce even further the chances of SSH exploitation.

8. DAEMON CONFIGURATION

You've cleaned up your packages, but it's also important to set the right applications to autostart on reboot. Be sure to turn off any daemons you don't need. One key to a secure server is reducing the active footprint as much as possible so the only surface areas available for attack are those required by the application(s). Once this is done, remaining services should be hardened as much as possible to ensure resiliency.

9. SELINUX AND FURTHER HARDENING

If you've ever used a Red Hat distro, you might be familiar with SELinux, the kernel hardening tool that protects the system from various operations. SELinux is great at protecting against unauthorized use and access of system resources. It's also great at breaking applications, so make sure you test your configuration out with SELinux enabled and use the logs to make sure nothing legitimate is being blocked. Beyond this, you need to research hardening any applications like MySQL or Apache, as each one will have a suite of best practices to follow.

10. LOGGING

Finally, you should make sure that the level of logging you need is enabled and that you have sufficient resources for it. You will end up troubleshooting this server, so do yourself a favor now and build the logging structure you'll need to solve problems quickly. Most software has configurable logging, but you'll need some trial and error to find the right balance between not enough information and too much. There are a host of third-party logging tools that can help with everything from aggregation to visualization, but every environment needs to be considered for its needs first. Then you can find the tool(s) that will help you fill them.

CONCLUSION

Each one of these steps can take some time to implement, especially the first time around. But by establishing a routine of initial server configuration, you can ensure that new machines in your environment will be resilient. Failure to take any of these steps can lead to pretty serious consequences if your server is ever the target of an attack. Following them won't guarantee safety-- data breaches happen-- but it does make it far more difficult for malicious actors and will require some degree of skill to overcome.