Community Experience Distilled

Unleash the power of Proxmox VE by setting up a dedicated virtual environment to serve both containers and virtual machines.

Rik Goldman

Proxmox VE 4.1 provides an open source, enterprise virtualization platform on which to host virtual servers as either virtual machines or containers.

This book will support your practice of the requisite skills to successfully create, tailor, and deploy virtual machines and containers with Proxmox VE 4.1.

Following a survey of PVE’s features and characteristics, this book will contrast containers with virtual machines and establish cases for both. It walks through the installation of Proxmox VE, explores the creation of containers and virtual machines, and suggests best practices for virtual disk creation, network configuration, and Proxmox VE host and guest security.

Throughout the book, you will navigate the Proxmox VE 4.1 web interface and explore options for command-line management.

Who this book is written for

This book is intended for server and system administrators and engineers who are eager to take advantage of the potential of virtual machines and containers to manage servers more efficiently and make the best use of resources, from energy consumption to hardware utilization and physical real estate.

What you will learn from this book

- Install and configure Proxmox VE 4.1
- Download container templates and virtual appliances
- Create and host containers based on templates
- Create and host virtual machines
- Optimize virtual machine performance for common use cases
- Apply the latest security patches to a Proxmox VE host
- Contrast PVE virtual machines and containers in order to recognize their respective use cases
- Secure Proxmox VE hosts as well as virtual machines and containers
- Assess the benefits of virtualization with regard to budgets, server real estate, maintenance, and management time

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 7 'Securing Proxmox VE'
- A synopsis of the book’s content
- More information on Learning Proxmox VE
About the Author

Rik Goldman had 18 years of professional IT experience and 17 years of teaching experience when he became the director of technology and a teacher of advanced computing at Chelsea School in 2012.

Throughout his 10 years at the university, he concentrated on literary computing, new media, humanities computing, and virtuality. At first, Rik supported his studies by developing institutional websites and database applications; eventually, however, he became the administrator of Solaris and Irix servers for West Virginia University's Center for Literary Computing, a lab committed to the study of electronic texts, virtuality, and digital composition and rhetoric.

In the classroom, Rik's commitment to authentic teaching and learning as well as his advocacy of social justice and equity have placed him at the vanguard of technology education. Working with and learning from his students, he has overseen projects that have provided real solutions for school infrastructure, data management, and programming. His many accomplishments reveal an educator who strives to provide authentic opportunities for learning and engagement, but his true legacy lies in what he has engendered in his students: a desire for knowledge, a critical urge, and an analyst's zeal for complex abstractions. Through this work with students and his responsibilities as a systems administrator, Rik has enjoyed a productive preoccupation with virtualization technologies and their impact on popular culture.

Since his full-time adoption of Red Hat 5 at home, he has been committed to GNU/Linux and the underlying philosophies that have made it so successful. Consequently, he is a passionate advocate of open source and free software. Together with his students, he has contributed to the success of a myriad open source endeavors by developing documentation, writing code, and mentoring communities of young developers from around the world.

In his free time, Rik enjoys reading literature, exploring critical theory, listening to records, and traveling to concerts with his family.
"There is a double spooking the world, the double of abstraction. The fortunes of states and armies, companies and communities depend on it. All contending classes - the landlords and farmers, the workers and capitalists - revere yet fear the relentless abstraction of the world on which their fortunes yet depend. All the classes but one. The hacker class."

"The virtual is the true domain of the hacker. It is from the virtual that the hacker produces ever-new expressions of the actual. To the hacker, what is represented as being real is always partial, limited, perhaps even false. To the hacker there is always a surplus of possibility expressed in what is actual, the surplus of the virtual. This is the inexhaustible domain of what is real without being actual, what is not but which may be. To hack is to release the virtual into the actual, to express the difference of the real."

– McKenzie Wark, A Hacker Manifesto

Not so many years ago, it would’ve taken three computers to author this book efficiently on the go. Virtualization, however, has made it possible to write without the obscene hassle of dragging about so much baggage. Virtualization has reduced labor and energy expenditure and maximized productivity and discretionary time during the writing and production of this book.

Abstraction liberates us from material constraints, leaving in their place the privilege of nostalgia – tractor-fed edge strips, darkroom chemicals, printing presses and type-set trays, overflowing money bags with dollar signs, and of course, cramped server rooms.

Through server virtualization, the abstraction of computing resources from physical systems has overturned data centers and radically upset the traditional and repetitive routines of system engineers and administrators in favor of efficiency, conservation, lowered expenditure, secure systems, and the simple deployment of automation to complete repetitive tasks.

Proxmox VE has been a pioneering agent in this rapid revolution since the 2008 release of version 1.0 – the first hypervisor to support both virtual machines and containers.
With version 4.2 in the works, and the industry's fascination finally fixed on the realization of a container revolution, Proxmox VE still provides an open source, enterprise virtualization solution with premium support that enjoys tremendous international popularity—even as competing brands have scrambled to roll out container solutions just in time.

This book is packed with introductory concepts and best practice techniques for experienced Linux users eager to take advantage of bleeding edge virtualization strategies and practices with Proxmox VE.

This book explores the benefits of two of these complementary virtualization technologies, containers and virtual machines, so you'll be forearmed to make informed and deliberated choices regarding the best paths for virtualizing your data center.

What this book covers

Chapter 1, Proxmox VE Fundamentals, outlines Proxmox VE’s features and distinguishing characteristics and briefly compares and contrasts virtual machines and containers.

Chapter 2, Installing Proxmox VE, goes through the Proxmox VE installation process after covering Proxmox VE’s hardware requirements and discussing minimal and optimal hardware specifications.

Chapter 3, Creating Containers, starts with a primer on containers and their uses before providing a walkthrough of the container creation processes, including choosing and downloading an OS or virtual appliance template.

Chapter 4, Creating Virtual Machines, first elaborates on the functional differences between virtual machines and suggests prospective use cases and the inherent benefits and drawbacks of full virtualization. It then walks through the process of creating and configuring virtual machines intended for Microsoft Windows Server and Fedora Server.

Chapter 5, Working with Virtual Disks, compares and contrasts virtual hard disk options, including disk image types, virtual bus/interfaces, and cache types.

Chapter 6, Networking with Proxmox VE, contrasts common virtual Ethernet adaptor options provided by Proxmox VE and works to articulate use cases for each.

Chapter 7, Securing Proxmox VE, enumerates strategies for mitigating security threats to virtualized datacenters in general, and Proxmox VE hosts and guests in particular.
Securing Proxmox VE

“Abstraction may be discovered or produced, may be material or immaterial, but abstraction is what every hack produces and affirms….To hack is to produce or apply the abstract to information and the possibility of new worlds”

– A Hacker Manifesto, McKenzie Wark

“Putting it bluntly, virtualization is deception.”

– Data Center Virtualization Essentials, Gustavo Alessandro Andrade Santana

“The enemy knows the system being used…”

– Shannon's Maxim

“Security through obscurity is not an answer.”

– Information Security: Principles and Practices, Merkow and Breithaupt

“Containers have quickly become a popular cloud-optimization strategy for enterprises, however, what do we really know about the security implications?”

– Kowsik Guruswamy

The end goal of this chapter is to support you in mitigating threats to the security of your Proxmox VE infrastructure.

We start by enumerating and articulating the potential benefits of virtualization on infrastructure security.

However, these benefits must not be relied on unconditionally or discussed uncritically; we must qualify them here.
Moreover, we must expose the potential security risks virtualization can inject into the infrastructure.

Ultimately, this chapter commits to providing strategies for mitigating security threats to our Proxmox VE hosts and guests.

Security assurance is, of course, a sprawling field, and often includes not only threat mitigation, but also policy making, monitoring, incident response, and forensics. Our focus here is exclusively on mitigating vulnerabilities specific to Proxmox VE hosts.

Towards that end, then, the goals of security are defined here in accordance with tradition:

- Maintaining the *confidentiality* of a system
- Assuring its *integrity*
- Providing consistent *availability* of services

![Security triad](image)
The preceding illustration suggests that these three points are not isolated: the whole picture consists of three mutual relationships: one between confidentiality and integrity; one between confidentiality and availability; and, finally, one between integrity and availability. Too much or too little emphasis on one point distorts our Platonic triangle, a symbolic representation of our impossible ideal.

Given our end goal—mitigating threats—this chapter proceeds along the following vector:

- Examining the potential security rewards of virtualization
- Interrogating those rewards and exploring the potential vulnerabilities virtualization introduces
- Acting directly to mitigate threats

**Security benefits of virtualization**

Introducing well-planned, deliberate, and well-executed virtualization into an infrastructure delivers some very compelling security benefits.

> “The abstraction of IT resources that masks the physical nature and boundaries of those resources…”

– Virtualization as defined by Gartner's IT Glossary ([http://www.gartner.com/it-glossary/virtualization](http://www.gartner.com/it-glossary/virtualization)).

Let's be clear about one thing with regard to this common trope representing virtualization as a deceptive masquerade: security through obscurity does not work. The use of secrecy for the design or implementation of a system to provide security is a failing proposition. In enumerating the security benefits of virtualization, this section purposefully avoids suggesting that abstraction and the obfuscation it permits are an effective security strategy.

We'll see as the chapter develops that none of the security rewards promised by virtualization advocates can be realized without a good understanding of networking, systems administration, type 2 hypervisors, VMs, and containers. The following is also required:

- Rigorous planning based in part on defense in depth
- Flawless realization of those plans
- Excellent management throughout the lifecycle of all guests.
Given all of the preceding, there are very clear security benefits to virtualization with Proxmox VE:

- Reduction of the physical attack surface
- VM isolation
- Ability to restore to prior states
- Hardware abstraction
- Network segmentation support
- Encapsulation and portability
- Physical security
- Fine privilege control
- Integrated firewalls

**Attack surface reduction**

Moving to a virtual infrastructure reduces your physical attack surface in accordance with your virtual machine density. The more physical servers we convert to Proxmox VE guests, in conjunction with how densely we pack guests onto our Proxmox VE hosts, the fewer servers there are to ensure protection from potentially devastating physical attacks, for example.

Virtualization has an inherent potential to reduce the attack surface of an infrastructure in several ways; we'll focus here on how it reduces the number of physical hosts providing services.
Where we were once protecting, say, 15 machines from physical attack, there might be one or two physical Proxmox VE hosts to protect, for example.

We have to be critical here and realize this can open up new vulnerabilities and frustrating problems to resolve.
Securing Proxmox VE

The Proxmox VE host becomes monolithic, so:

- If there's an unpatched vulnerability in Debian 8, PVE, or KVM and QEMU, the confidentiality, integrity, and availability of all the virtual machines hosted by that instance are also threatened.
- If an attacker gains physical access to the PVE host or hosts, the security and sanctity of all guests, both containers and virtual machines, are most certainly in doubt. If that's troubling, consider too that your snapshots and backups may have disappeared.

There's no doubt virtualization reduces the overall attack surface of your infrastructure, and in several ways; however, as articulated previously, this does not relieve us of any burdens, it just gives those burdens higher stakes while making them less complex to address.

Isolation

Virtualization encourages isolation. One VM doesn't naturally affect another VM, even if it is on the same host.

This tendency toward isolation suggests that destructive malware infecting a virtual server won't necessarily escape and spread to other virtual servers, even when they share a host.

If there's an oversight however, such as naively sharing data between two guests, or (worse) sharing data between a guest and the host, attacks can be devastating. Therefore, resist the temptation to create file shares among VMs on PVE; for the sake of security, do not share files between a Proxmox VE host and any of its guests.

Here the importance of documenting your infrastructure and writing a well-deliberated and well-enforced security policy is clear.

Availability of prior states

In the event an attack against a Proxmox VE guest succeeds, the guest can be rolled back to a prior state via backups or snapshots, effectively minimizing the time it takes to recover from an attack. (Note, however, that any data or information that changed between the time of the selected backup and the moment it is restored will be lost during a rollback.)
Rolling back involves applying a former version of a guest's storage file to the virtual machine or container.

So, it's logical that the ability to restore to a prior state isn't an unconditional advantage of virtualization, even if it's an integrated feature of Proxmox VE; snapshots and backups must be integral to the well-thought out and executed lifecycle of the PVE guests and part of a well-defined and enforced policy.

**Hardware abstraction**

A fundamentally compelling characteristic of full virtualization is the abstraction of a computer from the physical hardware.

Imagine that a Proxmox VE guest with a well-organized trove of information collected over months is compromised and its hard drive destroyed in an attack that would have rendered physical storage unsalvageable—destruction of storage on a virtual disk does no damage to the physical storage that hosted it. When prior states are available to restore from, a snapshot or a backup can be restored onto the same hardware.

Without condition, damage dealt to virtual components has no effect on the physical host. This is an inherent reward for abstraction.
Segmentation

If an infrastructure is to be virtualized with Proxmox VE, take advantage of the network segmentation technologies it supports, such as VLAN tags, bridges, IPs masquerading with NAT, and per-guest firewalls. Use these technologies to make VMs or containers available to a limited population of users with legitimate business needs that call for a limited level of access.

Visualizing segmentation and trust zones with Proxmox VE
To make the most of this valuable potential, think rigorously about sufficient trust zones as you plan your infrastructure.

A trust zone is a network segment within which data flows relatively unrestricted, while data flowing in and out of the trust zone is subject to stronger restrictions.

**Open vSwitch**

As an alternative to the bridges, bonds, and VLAN interfaces native to Linux, Proxmox VE supports Open vSwitch.

Open vSwitch is an open source, software implementation of a distributed, multilayer switch. It's production-ready and designed with virtualization specifically in mind.

To learn more about Open vSwitch, including features and potential drawbacks, visit its website at http://openvswitch.org. To learn more about Proxmox VE's support of Open vSwitch, visit the Proxmox wiki at https://pve.proxmox.com/wiki/Open_vSwitch.

**Encapsulation and portability**

As described in Chapter 5, Working with Virtual Disks, with full virtualization all information on virtual servers, including boot disks, is saved as a file; this is an example of encapsulation.

Invaluably, encapsulation serves the portability of the VM; even as an attacker works to compromise a PVE host, its guests can be moved, without halting, to another host in a cluster. Live migration—migrating an active guest from one PVE host to another in the same cluster—helps assure availability even in the case of an ongoing attack against a host.

**On Clustering and High Availability**

For more on these topics, see Mastering Proxmox, Proxmox High Availability, and Proxmox Cookbook from Packt Publishing.
Physical security

When an attacker gains access to a physical server, the availability, confidentiality, and integrity it should demonstrate are absolutely in doubt.

To put it bluntly (and to appropriate a phrase from Scott Culp), if an attacker has physical access to your Proxmox VE host, it's not your host anymore.

As described previously, a very powerful benefit of virtualization is that the number of physical hosts decreases as the services they provided move to VMs or containers, effectively reducing the attack vectors, since the fewer machines there are to find physical space for, the fewer there are to gain illegitimate access to.

Nevertheless, include in your policy the procedures by which the PVE hosts, the storage, and the intermediate distribution frame (that is, the wiring closet) are to be physically secured against illegitimate access, and enforce the policy consistently and without exception; this means you've mitigated a potentially devastating threat.

Fine privilege control

With defense in depth and the principle of least-privilege in mind, consider the fine control of user access and restrictions that's realized when each service is moved to a discrete VM or container—in contrast to keeping multiple services running on a single physical server.
In a virtual infrastructure, a user privileged to access one service is not explicitly privileged to another as we can imagine they would be if the services were shared on a single hardware server. We can restrict access to Debian, finely define a user's role in relation to each VM and container via the PVE web interface, and further refine privileges on the guest OS level and the application level.

Some layers to consider in the practice of "defense in depth"

By default, the Proxmox VE management interface is authorized through GNU/Linux's default authentication system (PAM), and root is the only user. However, the authentication system for the web interface can be changed from PAM to PVE's system, to Active Directory, or to LDAP.

Whatever the authentication mechanism, specific users can be assigned different roles, or privileges, for each individual VM or container.
PVE User Management
To learn more about PVE's user management features, see the user management page of the Proxmox wiki: https://pve.proxmox.com/wiki/User_Management.

There are a host of predefined roles for users or groups that ship with PVE; and we can create new roles with different privileges and restrictions as necessary.

PVE firewall features
Proxmox VE provides flexible firewall features based on iptables.

These features can be configured via the administration interface or the command line to provide several layers of protection, as this allows rulesets to accept and reject traffic per guest server, per PVE host, and for an entire cluster.

To learn more about the Proxmox VE firewall, visit the official documentation at https://pve.proxmox.com/wiki/Proxmox_VE_Firewall, where detailed configuration examples are available.

It's rather critical that PVE be protected by a firewall.

Proxmox VE 3.4 relies on the following ports:

- 8006 (web interface)
- 85 (pvedaemon—configured to listen only on 127.0.0.1)
- 5900-5999 (VNC web console)
- 22 (sshd; used for cluster actions as well as a means to access a remote shell)
- 5404, 5405 UDP (CMAN multicast—if you run a cluster)

Proxmox VE 4.0 saw some changes to Proxmox VE's port usage:

- 8006 (web interface)
- 85 (pvedaemon-configured to listen only on 127.0.0.1)
- 5900-5999 (VNC web console)
- 3128 (SPICE console)
- 22 (SSH access-now optional)
- 111 (rpcbind)
- 5404, 5405 UDP (corosync multicast if you run a cluster)
Use your firewall experience to restrict access to these ports from subnets and IP ranges that don't have a legitimate need to access them.

Aggravated vulnerabilities

Virtualization's potential security benefits are certainly compelling, but many are quite conditional and altogether they are certainly no panacea.

Moreover, virtualization introduces new threats to an infrastructure—threats that otherwise either wouldn't be a concern at all or are exacerbated by virtualization.

This section calls attention to vulnerabilities that are historically problematic for virtual infrastructures:

- Denial of service attacks
- VM escape and hyper jumping
- Server sprawl
- Growing complexity

Denial of service attacks

Denial of service (DoS) attacks come in a wide variety of flavors. However, the immediate intent is the same: overwhelming a network, and its administrators, by generating large amounts of illegitimate traffic.

Distributed denial of service (DDoS) and DoS attacks are cheap, effective, and increasingly common. On the surface, they seem to be most effective at rendering services unavailable or unusable. More insidious, perhaps, is that, by keeping administrators' hands full as they cope with illegitimate traffic, other attacks can be launched without attracting their immediate attention.
Unfortunately, DoS attacks are particularly powerful against virtual infrastructures, wherein overwhelmed virtual hosts will certainly threaten the availability of virtual guests and the services they provide.

Visualizing a DDoS attack

In addition, research published in 2013 found that DoS attacks are significantly more potent where virtual machines are involved:

“[Under] DoS attack, the performance of a web server hosted in a VM can degrade by up to 23%, while that of a nonvirtualized server hosted on the same hardware degrades by only 8%. Even with relatively light attacks, the file system and memory access performance of hypervisor-based virtualization degrades at a much higher rate than their nonvirtualized counterparts.”


Clearly, in some areas, we may take solace in some security benefits of virtualization; however, DoS and DDoS attacks are one threat we cannot turn away from.

When migrating services from physical machines to Proxmox VE guests, work to define and deploy not only preventative measures, but also rapid response protocols. This calls for the implementation of monitoring and alert systems, as well as a firewall configuration that deliberately takes such attacks into consideration.
Each type of DoS attack has its own array of detection and mitigation strategies. To mitigate against SYN flooding attacks, for example, see section three of http://tools.ietf.org/html/rfc4987.

**VM escape and hyper jumping**

*Virtual machine escape* occurs when an attacker successfully “breaks out” of a virtual machine and interacts with the host operating system.

In a similar vein, *VM jumping*, or hyper jumping as it is sometimes referred to, is the process of gaining illegitimate access to a virtual machine via another virtual machine.

Presumably encapsulated and isolated environments, virtual machines run operating systems that shouldn’t know that they are virtualized; there should be no way to break out of the virtual machine to interact directly with the parent hypervisor. For the same reasons, it should be impossible to illegitimately access a virtual machine through another virtual machine.

VM escape exploits are particularly devastating since the hypervisor controls the execution of all of the virtual machines and containers on the host. Consequently, an attacker that can gain access to the hypervisor can then win control over every guest running on the PVE host; since the hypervisor is between the physical hardware and the guest operating system, a successful VM escape will enable attackers to simply circumvent security controls implemented on the virtual machine.

VM escapes and hyper jumping should be an intellectual exercise, a fascinating theoretical problem. Unfortunately, that's simply not the case.

During the production of this book, for example, several VM escape vulnerabilities have emerged. Perhaps the vulnerability that captured the most attention was the one dubbed VENOM by researchers (http://venom.crowdstrike.com/):

“VENOM, CVE-2015-3456, is a security vulnerability in the virtual floppy drive code used by many computer virtualization platforms. This vulnerability may allow an attacker to escape from the confines of an affected virtual machine (VM) guest and potentially obtain code-execution access to the host. Absent mitigation, this VM escape could open access to the host system and all other VMs running on that host, potentially giving adversaries significant elevated access to the host’s local network and adjacent systems."
“Exploitation of the VENOM vulnerability can expose access to corporate intellectual property (IP), in addition to sensitive and personally identifiable information (PII), potentially impacting the thousands of organizations and millions of end users that rely on affected VMs for the allocation of shared computing resources, as well as connectivity, storage, security, and privacy.”

Unfortunately, the many “computer virtualization platforms” invoked by the disclosure included all QEMU-based platforms; this includes Proxmox VE.
In the case of VENOM, a patch was available from Debian the same day the vulnerability was disclosed. A PVE administrator simply had to upgrade, in Debian fashion, and then restart guest virtual machines:

```
apt-get clean
apt-get update
```

After a shutdown and restart of all VMs on the Proxmox VE host, the vulnerability was gone. There was no need to even reboot the PVE host.

The bug was introduced unknowingly in the QEMU source when the QEMU Floppy Drive Controller was introduced in 2004.

So it seems virtualization's celebrated isolation is not absolute.

From VENOM, we can learn some direct preventative actions that can be taken to mitigate emerging VM escape vulnerabilities:

- Keep Proxmox VE and Debian routinely patched. There are a variety of ways to automate the patching process; we'll walk through one method in the next section. Since Proxmox VE is patched through the same mechanism as Debian, patches to both are applied simultaneously.
- Patch operating systems and applications running on virtual machines and containers. On Debian and Ubuntu guests, use the apt tool; on Microsoft Windows and Server guests, set a reasonable Windows Update policy to ensure urgent updates are applied.
- Don't install virtual machine features you do not need. Doing so increases your attack surface unnecessarily. Be particularly attentive to what virtual devices are attached to your VM; if you don't need a virtual optical drive or floppy disk drive on the VM, either deliberately avoid installing them or remove them from the VM when you no longer need them.
- Avoid running software and services that are not essential to your guests' primary roles.
- However, weigh the benefits of running endpoint security software on a virtual machine; in his September 2015 article, “The Curious Case of the Escaping Virtual Machine,” Bunmi Sowandi suggests that such software will detect malicious code trying to run in a VM before it has a chance to “escape”. (http://vmturbo.com/about-virtualization/the-curious-case-of-the-escaping-virtual-machine/)
As we learned from VENOM, the best protection against VM escape and hyper jumping exploits is routine and well-thought out patch management.

**Virtualization sprawl**

In the context of virtual infrastructure, sprawl refers to the tendency of virtual servers to proliferate faster than administrators can properly manage. Sprawl encourages poor management decisions, hasty undeliberated action, sloppy configuration mistakes, and missed opportunities to mitigate threats.

From a security perspective, therefore, virtualization sprawl presents dire problems as administrators miss security patches, fail to harden services, and perhaps expose the network, the hypervisors, and storage nodes unnecessarily.

A helpful article on the Hewlett Packard site suggests some best practices to reduce the impact of sprawl effectively. Like many security issues explored in this chapter, the suggested solution is excellent planning, deliberated deployment, and writing and enforcing a security policy that includes VM lifecycle management. The article ([http://h30499.www3.hp.com/t5/Grounded-in-the-Cloud/5-ways-to-get-control-of-virtualization-sprawl/ba-p/6170959](http://h30499.www3.hp.com/t5/Grounded-in-the-Cloud/5-ways-to-get-control-of-virtualization-sprawl/ba-p/6170959)) puts particular emphasis on the following:

- Whenever possible, create VMs and containers from “golden images” that include patches, patch policy, audit policies, software, and software policies
• Proactively update policy-based enforcement on VMs as well as VM templates (and containers and container templates)

• Systematically manage the lifecycle and compliance of virtual servers from end to end (including, for example, routine snapshots or backups, and applying patches and upgrades)

The article is insistent on conveying two messages successfully:

• A well-thought-out security policy, which administrators can realistically comply with, is absolutely essential for keeping sprawl in check
• Tasks that can be automated must be automated

Virtualization sprawl encourages disorder; tame it with automation driven by a well-thought-out policy that’s followed in the practice of daily administration.
At war with complexity

“A network architecture should be as clean and simple to understand as it can be. It should be possible to briefly…draw a few simple pictures to illustrate that design....”

“Having a clear understanding of the traffic flow on your network puts you in control of it. Not understanding the network puts you at the mercy of its vagaries.”

– The Practice of System and Network Administration, 2E 2007

Given that virtualization encourages sprawl and given that secure virtual infrastructures demand segmentation, it's not entirely surprising that virtualization encourages problematic network complexity.

As network complexity goes up, so too does the pain that administrators suffer, since they must keep accurate documentation, troubleshoot connectivity problems on the fly, and sometimes provide actionable information for third-party support providers.

The “chaos approach” encouraged by infrastructure virtualization is not a reliable model to use in a network where the availability of every component matters (The Practice of System and Network Administration).

To limit network complexity, consider that the campus' network architects, engineers, and administrators should all be able to sketch, without aids, the key features and basic structure of the network topology.

According to Limoncelli, Hogan, and Chalup, the network architecture is neither clean enough, comprehensible enough, nor simple enough if this network map can't be relatively easily rendered. Maps of the physical and logical networks should absolutely be part of the system documentation and revised to reflect any modifications from the previous topology.

Taking the risk of sounding redundant squarely in the face, the best way to tame network complexity is to ensure the network can be explained and diagrammed, logically and physically, without the support of additional resources. If that's not the case, re-evaluate the architecture and see where it can be simplified without sacrificing security.
Taking action

If you're not yet virtualizing infrastructure, or you're not otherwise in a position to develop a strategic security policy, there're tactics you can take in the meantime to mitigate some threats to your Proxmox virtual environment:

- Secure the bootloader
- If possible, lock down the BIOS/UEFI
- Absolutely prohibit remote access to Proxmox VE's user interfaces
- Disable root access via SSH; consider prohibiting sudo access as well
- Use Fail2ban to prevent brute-force attacks
- Rely on key-based SSH authentication
- Maintain security patches for Proxmox VE and its guests
- Consider an enterprise support subscription

The practical procedures that follow are a strong (and immediate) complement to the less concrete strategies articulated previously.

This concluding section thus walks through these immediate tactical mitigation objectives to provide immediate support as you come to terms with Proxmox VE.

Protecting the boot process

In this section, we work to assure that OS and application-level authentication isn't rendered useless by an attacker with physical access who can thoroughly bypass these mechanisms.
We can think of booting as a four-stage process:

- The PC is turned on, and the BIOS is loaded.
- The BIOS runs the code in the MBR of disk 0.
- The MBR code loads and runs the code in the bootsector of the active partition.
- The bootsector code looks for and runs bootmgr from the root of its partition.

During this process, the system can be vulnerable:

- An unsecured BIOS can be directed to boot from an attacker's storage device, allowing them to compromise the confidentiality and integrity of data stored on the Proxmox VE host and interfere with the availability of the services and virtual servers the host has intended to provide.
- By manipulating an unsecured bootloader, attackers can gain root access to a machine and compromise its confidentiality, integrity, and availability.

Using either method, the attacker effectively owns the machine. Let's do our best to lock down the BIOS/UEFI on our hosts and GRUB 2.0, the bootloader for Proxmox VE 3.4 and 4.1.

Locking down the bootloader

OS-level authentication restrictions can be very simply defeated on an otherwise secure Proxmox VE machine by manipulating GRUB 2, Proxmox VE's bootloader. See, for example, http://linuxconfig.org/ubuntu-14-04-lost-password-recovery, wherein the process is fully articulated. The gist of an attack looks like this:

1. Reboot and enter the GRUB 2 menu immediately after startup.
2. Modify the boot options.
3. Boot the system based on your modifications.
4. Change the root password of the system.
5. Shutdown and restart.
6. Login with the new password.

For an experienced GNU/Linux administrator, it's likely a familiar process; it's identical to how we reset a lost root password.

GRUB 2 offers extensive customization, and with it, the power to disable access to GRUB 2 options generally, as well as to specific menu options.

We will walk through the universal protection of GRUB 2 menu entries to enable access by a single superuser with an unencrypted password. This will prohibit attackers as well as sloppy or disgruntled colleagues from editing a GRUB entry or accessing a GRUB command line.

To follow this procedure, you must:

- Have root privileges
- Determine a superuser name and password to be used (we will use the name admin and the password pve)
- Edit GRUB configuration files from the command line
- Update the GRUB 2 configuration with the update-grub command

Let's get started:

1. Log into a shell on your Proxmox VE host; if PVE is configured with an IP address of 192.168.1.200, access a shell via SSH from a workstation on the same network:

```bash
ssh root@192.168.1.200
```
2. Open `/etc/grub.d/00_header` using a plaintext editor; this example uses `nano` as the editor:

   ```
nano /etc/grub.d/00_header
   ```

3. Append the following lines to the bottom of the text:

   ```
cat << EOF
set superusers="admin"
password admin pve
EOF
   ```

4. Save the document and exit the editor; in `nano`, use `Ctrl + X`, then `Y`, and then `Enter` to return to a shell prompt.

5. Open `/etc/grub.d/10_linux` in `nano`:

   ```
nano /etc/grub.d/10_linux
   ```

6. Seek the following group of lines in `/etc/grub.d/10_linux`:

   ```
echo "menuentry '$(echo "$title" | grub_quote)' ${CLASS} \$menuentry_id_option 'gnulinux-\$version-\$type-\$boot_device_id' " | sed "s/^/$submenu_indentation/"
else
  echo "menuentry '$(echo "$os" | grub_quote)' ${CLASS} \$menuentry_id_option 'gnulinux-simple-\$boot_device_id' " | sed "s/^/$submenu_indentation/"
   ```

7. In the last line, insert `--unrestricted` between `${CLASS}` and `\$menuentry`; the resultant line appears as follows:

   ```
echo "menuentry '$(echo "$os" | grub_quote)' ${CLASS} --unrestricted \$menuentry_id_option 'gnulinux-simple-\$boot_device_id' " | sed "s/^/$submenu_indentation/"
   ```

8. Save the revised document and exit the editor; in `nano`, use `Ctrl + X` to exit, press `Y` to confirm, and then press `Enter` to return to a shell prompt.

9. Finally, prompt GRUB 2 to reconfigure itself based on the changes:

   ```
update-grub
   ```

When you reboot, you should find that PVE will start normally if left uninterrupted. However, if you try to edit a menu entry, boot from a submenu, or access a GRUB command line, you should find that you're required to authenticate.
For more on GRUB security, visit the following links:
https://help.ubuntu.com/community/Grub2/Passwords

As the article at http://opensourceforu.efytimes.com/2013/03/playing-hide-and-seek-with-passwords/ points out, this password can still be bypassed by configuring BIOS/UEFI to boot from the attacker's boot device. If your hardware allows, you probably want to secure this first stage of the boot process so would-be malefactors won't be able to finagle the Proxmox VE host to boot from their own devices.

**Locking down BIOS/UEFI**

By securing the bootloader, GRUB 2, we can prevent a user from bypassing OS security and gaining root privileges on the Proxmox VE host.

However, attackers can still simply bypass even the bootloader's security by booting instead from their own media. From there, they can mount the machine's secondary storage and make immediate decisions for you about its confidentiality and integrity. If an attacker is particularly deliberate, he/she can install a cunning means to access the machine remotely at a later date.

To mitigate this threat, we can, depending on our firmware, password protect the boot device settings in either the BIOS or UEFI.

Since there's an unwieldy array of BIOS and UEFI firmware vendors, we'll articulate a vision for what we'd like to do, and then hope our systems will cooperate.

The objective is to manipulate the BIOS/UEFI so it operates as follows:

- Allows the system to cold-boot without any interruption
- Requires authentication to change the boot device
- Prohibits entering the setup manager without authentication

This configuration can be tricky and is largely contingent on the BIOS/UEFI vendor.
Ideally, a machine with BIOS will allow you to do the following:

1. Enter the BIOS configuration when starting your PC. After power on, press the prompted key to enter **BIOS Setup Utility**; sometimes, it's the F key, *Delete*, or *ESC*. On some Lenovo machines, it's *Enter*. In the case of VMWare's PhoenixBIOS, F2 is used to access the setup utility (access to this interface is what we want to make impossible or frustrating for an attacker):

   PhoenixBIOS 4.0 Release 6.0
   Copyright 1985-2001 Phoenix Technologies Ltd.
   All Rights Reserved
   Copyright 2000-2013 VMware, Inc.
   VMware BIOS build 397

   639K System RAM Passed
   2047M Extended RAM Passed
   ATAPI CD-ROM: VMware Virtual IDE CDROM Drive
   Mouse initialized

   Press F2 to enter SETUP, F12 for Network Boot, ESC for Boot Menu

   PhoenixBIOS, included with VMware Workstation, uses F2 to enter setup, F12 for network boot, and ESC for the boot menu.

2. Once you enter **SETUP**, navigate to a security tab. On the machines I've accessed, setting the administrator password will require the same credentials of anyone trying to enter the setup utility. On the PhoenixBios for VMware Workstation virtual machines, I can still boot without authenticating. So far so good.
3. However, when the preceding screenshot invites users to press F12 for network boot or Esc for the boot menu, they'll find these are unrestricted. In this case, any attempt to lock down BIOS for security is really quite devastated.

4. If available, navigate to the boot settings in the SETUP utility. If the option is available in the Boot menu, disable devices that aren't used and that you don't want to use, such as network boot and boot from optical drives or USB devices.

5. Again, depending on availability, navigate to boot priority and set the boot priority of the remaining devices. Some systems allow you to use the D key to disable devices.

Ideally, your BIOS setup tool allows you to disable boot devices or password protect the boot device menu as well as the SETUP tool itself. Furthermore, it'll hopefully let you set a supervisor password without prompting you to authenticate to boot. If the boot process is interrupted to ask for authentication, there's the potential for a lot of unnecessary, and perhaps unplanned, down-time for the services users are relying on.

The settings recommended previously were possible in about half of the physical machines I surveyed for the chapter.

If your machine does provide for this ideal scenario, realize that, at best, losing the BIOS passwords means you've lost your license to reconfigure or troubleshoot the physical machine. In case of a lost password, you'll need to research how to bypass the security that you once felt was keeping bad guys out. The reset process is awkward, inconsistent, a time sink, and may not work.

It's essential, therefore, to keep a safe copy of any credentials you use to restrict access to BIOS or UEFI.

To learn more about securing the specific BIOS/UEFI system for your Proxmox VE host, look for the documentation from the computer or BIOS manufacturers or from community users on the Web.

If protecting the boot process to your satisfaction is not possible with the BIOS or UEFI you're stuck with, compensate by making sure access to the physical host is absolutely secure with alarmed door locks, key passes that log ingress and egress, and so on.

**Secure Boot and Proxmox**

**TIP**

Proxmox VE does not support the Secure Boot feature of UEFI.
Hardening the OS and hypervisor

The objective here is to ensure the security of both Proxmox VE (3.4 and 4.0) and Debian, its underlying operating system. Because Proxmox VE and Debian are inextricably bound, it's appropriate to address them together.

Prohibit remote access to the hypervisor

Pundits specifically committed to secure virtual infrastructures are insistent on this point: remote access to the hypervisor must be forbidden.

This directive must be qualified: it's absolutely appropriate it's absolutely appropriate to run a Proxmox VE host heedlessly (sans display) and to access it from another workstation on the same LAN via both SSH and the web-based administration interface.

What you want to avoid at all costs is making PVE ports, particularly 22 and 8006, public-facing and accessible to the Internet. Unless VPN is configured, Proxmox VE absolutely should not be available from outside the LAN.

OpenVPN is an open-source package for providing VPN services; if you're considering a VPN solution, read more about OpenVPN at https://wiki.debian.org/OpenVPN.

Harden SSH

Proxmox VE is designed to have two access alternatives:

- Access to a command line interface via SSH
- Access via the Web-based administrative interface

SSH must be an available option so administrators can make configuration changes to the underlying operating system. Moreover, as we saw in Chapter 3, Creating Containers and Chapter 4, Creating Virtual Machines, we may choose to take care of a significant amount of Proxmox VE administrative tasks via the command line.

So SSH can’t be disabled outright in the name of security. However, in the name of security assurance, we must fine-tune the configuration of SSH to mitigate threats, whether from disgruntled or sloppy colleagues or outside attacks.
Our objectives here are as follows:

- Disabling direct root account access via SSH
- Mitigating brute-force password attacks against SSH
- Limiting access by IP
- Using encrypted keys rather than passwords to authenticate over SSH

**Disabling root account access via SSH**

This procedure is critical and absolutely necessary. First, let's create our own accounts with which to log in. We'll then use the new account to log in, escalate privileges using the `su -` command, and then follow a simple procedure to disable the root from logging in through SSH.

Once this procedure is complete, we'll log in using the new account for the foreseeable future; to perform a procedure that requires root's privileges, we'll simply use the `su -` command to temporarily escalate the privileges of this user account, by following these steps:

1. Choose a username and password you intend to use to administer Proxmox VE via SSH.
2. Via SSH, log in as the root using the credentials you created during the installation of Proxmox VE.
3. Use the adduser command, followed by the username you've chosen, to create a new account:
   ```
   adduser rgoldman
   ```
4. Follow the prompts to create and confirm the new account's password.
5. Enter the full name you'd like associated with the new account.
6. You may choose to ignore the remaining prompts for additional information, such as office number, address, and telephone number.
7. Press Y and Enter to confirm the creation of the new user.

```
root@DevOps:--# adduser rgoldman
Adding user `rgoldman' ...
Adding new group `rgoldman' (1000) ...
Adding new user `rgoldman' (1000) with group `rgoldman' ...
Creating home directory `/home/rgoldman' ...
Copying files from `/etc/skel' ...
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
Changing the user information for rgoldman
Enter the new value, or press ENTER for the default
  Full Name []: Rik Goldman
  Room Number []:
  Work Phone []:
  Home Phone []:
  Other []:
Is the information correct? [Y/n] y
root@DevOps:--#
```

Creating a new user within a command line prompt

8. Use the exit command to close the SSH session.
9. Reconnect to Proxmox VE using the new account name:

   `ssh rgoldman@192.168.1.200`

10. Use the su - command to escalate privileges and enter the root's password to continue.
11. Edit `/etc/ssh/sshd_config` using nano:

   `nano /etc/ssh/sshd_config`

12. Seek the following section using the arrow keys or the PageDown key:

   ```
   #LoginGraceTime 2m
   #PermitRootLogin no
   #StrictModes yes
   #MaxAuthTries 6
   ```
13. In this file, the # symbol signifies that the line is a comment and is not to be considered as part of the SSH daemon's configuration. Let's remove the # symbol from the second line to enable the directive:

    PermitRootLogin no

14. Save and exit the revised configuration file: Ctrl + X, Y, Enter.
15. We can restart the SSH daemon without it affecting our session:

    /etc/init.d/ssh restart

15. Now, let's test to ensure root can no longer log in. Start by entering exit to leave the su mode, and exit again to close SSH.
16. Start a new SSH session as the root:

    ssh root@192.168.1.200

Access should be denied, without the opportunity to enter a password.

**Preventing brute-force attacks against SSH**

As long as SSH daemon is configured to use password authentication, it's vulnerable to brute-force password attacks. One mitigation strategy is to install and configure Fail2ban, a powerful tool designed to detect attacks on a service and ban the offending IP address from which the attacks originate for a predefined period. Fail2ban effectively increases the cost in resources and the time attackers have to invest to continue a brute force password attack.

To make use of Fail2ban, follow this procedure:

1. Login via SSH and your new user account.
2. Escalate privileges:

       su -

3. Install Fail2ban:

       apt-get update && apt-get install -y fail2ban

4. Copy the default configuration called jail.conf to a new file called jail.local:

       cp /etc/fail2ban/jail.conf /etc/fail2ban/jail.local
5. Open the new file in an editor such as nano:

```
nano /etc/fail2ban/jail.local
```

6. Confirm Fail2ban's configuration for SSH in the following stanza:

```
[ssh]
enabled   = true
port      = ssh
filter    = sshd
logpath   = /var/log/auth.log
maxretry  = 6
```

7. Save the file and exit nano with Ctrl + X, Y, Enter.

8. If any changes were necessary, restart Fail2ban:

```
/etc/init.d/fail2ban restart
```

Fail2ban can also be configured to protect PVE's web-based administration interface from brute-force attacks.

First, add the following to `/etc/fail2ban/jail.local`:

```
[proxmox]
enabled = true
port    = 8006, https
filter   = proxmox
logpath = /var/log/daemon.log
maxretry = 3
bantime = 3600 # 1 hour
```

Then, create the filter by entering nano
```
/etc/fail2ban/filter.d/proxmox.conf at the command line.
```

Enter the following lines:

```
[Definition]
failregex = pvedaemon\[.*authentication
failure;
rhost= <host> user=.* msg=.* ignoreregex =
```

Save the file and exit nano with Ctrl + X, Y, Enter.

Restart Fail2ban to activate the new configuration:

```
/etc/init.d/fail2ban restart
```
With Fail2ban configured as described previously, failure to authenticate successfully three times in a row in the web interface will prohibit the client from connecting again for a full hour:

**Unable to connect**

Firefox can't establish a connection to the server at 172.16.118.128:8006.

- The site could be temporarily unavailable or too busy. Try again in a few moments.
- If you are unable to load any pages, check your computer's network connection.
- If your computer or network is protected by a firewall or proxy, make sure that Firefox is permitted to access the Web.

Try Again

After three unsuccessful attempts to log in to the PVE web interface
More on Fail2ban
To learn more about Fail2ban and how it works, visit http://www.fail2ban.org/wiki/index.php/Main_Page.
To learn more about Fail2ban and Proxmox VE, visit the Proxmox Wiki at https://pve.proxmox.com/wiki/Fail2ban.

Relying on key-based authentication
Another way to secure SSH access to a Proxmox VE server is to rely on key-based authentication instead of password authentication. The advantage to this authentication method is that you can disable password authentication altogether and not have to worry about the strength of legitimate users' passwords. Another benefit is that you can use the same key to authenticate to any number of SSH servers.

To make use of this feature, we'll start by generating an SSH key pair. Once you have a public and private key that can be used to authenticate, we'll place the public key on the PVE host so that we can use SSH key authentication to log in. Once these two procedures are complete, you'll check to see whether you can log in to your PVE host.

In the examples that follow, rgoldman is used as a placeholder for the username, while 192.168.1.200 is used as a placeholder for PVE's IP address. Please replace each as appropriate.

1. On a workstation on the same LAN as the PVE host, generate an SSH key pair:

   `ssh-keygen`

2. Confirm the key location at the first prompt by pressing Enter:

   Generating public/private rsa key pair.
   Enter file in which to save the key (/home/rgoldman/.ssh/id_rsa):

3. At the next prompts, you may choose to create and confirm an optional passphrase; using a passphrase will prevent an attacker from accessing the PVE host from your workstation. On the other hand, you'll have to enter the passphrase every time you wish to authenticate with the key:

   Created directory '/home/rgoldman/.ssh'.
   Enter passphrase (empty for no passphrase): 
   Enter same passphrase again: 
4. Look for confirmation that the key has been created—you should see an output to the terminal similar to the following:

    Your identification has been saved in /home/rgoldman/.ssh/id_rsa.
    Your public key has been saved in /home/rgoldman/.ssh/id_rsa.pub.
    The key fingerprint is:
    The key's randomart image is:
    +---[ RSA 2048]----+
    |   ..o   |
    |    o. o  |
    |   ..S    |
    |    o o.  |
    | =o.+o    |
    | . ==++.. |
    | o==++.   |
    +----------+

    An output similar to the preceding example confirms that you have a public and private key that you can use to authenticate.

    Next, let's place the public key on the PVE host so that you can use the SSH key authentication to login.

5. Confirm the presence of the keys by listing the contents of ~/.ssh/; ensure that both id_rsa and id_rsa.pub are present in the results:

    ls ~/.ssh/

6. Next, push the new public key to the PVE host using the ssh-copy-id tool with the following syntax (assuming the username is rgoldman and the PVE host is at 192.168.1.200):

    ssh-copy-id rgoldman@192.168.1.200

7. That's it. Now confirm you can login without a password:

    ssh rgoldman@192.168.1.200

If you were able to login to your PVE account using SSH without a password, you have successfully configured SSH key-based authentication for your account.
Securing Proxmox VE

The image that follows illustrates the transaction:

![SSH Key Authentication Diagram]

Note, however, that your password-based authentication mechanism is still active; the SSH server is still exposed to brute-force or social engineering attacks.

The following steps will disable password-based authentication on your server. Certainly, don't proceed here if you plan to access PVE from other hosts but haven't copied the key yet (or if you are not yet confident with key-based authentication):

1. Using SSH, log in to the PVE host from a local workstation.
2. Escalate privileges:
   ```
   su -
   ```
3. Open the SSH daemon configuration using `nano` or another plaintext editor:
   ```
   nano /etc/ssh/sshd_config
   ```
4. Browse the contents of the file for the following directive:
   ```
   PasswordAuthentication yes
   ```
5. Change the directive so it is as follows:
   ```
   PasswordAuthentication no
   ```
6. Save the file and close the editor with `Ctrl + X, Y, Enter`.
7. Restart the SSH daemon:
   ```
   /etc/init.d/ssh restart
   ```
We can now access Proxmox VE via SSH without worrying about the strength of our passwords or being vulnerable to brute force attacks or social engineering attacks.

Note, however, it's imperative when we rely on key-based authentication that we keep our keys absolutely secure. This means, in part, remembering consistently to lock our workstations and protect our workstation accounts with strong passwords.

Learn more about SSH
To learn more about SSH and authentication, consider visiting the following resources:
https://www.debian-administration.org/article/530/SSH_with_authentication_key_instead_of_password
https://www.digitalocean.com/community/tutorials/how-to-configure-ssh-key-based-authentication-on-a-linux-server

Managing patches
As we discovered, the patch for the VENOM exploit was available for Debian the same day the exploit was made public; PVE administrators simply had to update and upgrade and restart PVE guests to eliminate the threat. This should drive home the importance of applying security patches not only for the Proxmox VE host, but also for its guests, whether they are containers or virtual machines.

However, routinely applying patches for several machines is tedious no matter the assurance it provides.

For the PVE hosts and Ubuntu or Debian guests, there are several tools to relieve the tedium. Finding the sweet spot between fully automated upgrades with minimal interactivity and doing due diligence to ensure patch candidates won't disrupt operations is where the magic happens.

In this section, we'll configure a tool called unattended-upgrades to routinely apply only security upgrades. We'll leave other patches to our best judgement.
Use the APT tool to install unattended-upgrades:

```
su -
apt-get update
apt-get install -y unattended-upgrades
```

Then, configure the automated security upgrades:

1. The default configuration file for the unattended-upgrades package is at `/etc/apt/apt.conf.d/50unattended-upgrades`; let's take a look at it:

```
nano /etc/apt/apt.conf.d/50unattended-upgrades
```

2. Look for the following stanza (the // characters preceding some lines effectively comment out directives so they are ignored):

```
// Automatically upgrade packages from these (origin:archive) pairs
Unattended-Upgrade::Allowed-Origins {
    "${distro_id}:${distro_codename}-security";
    "${distro_id}:${distro_codename}-updates";
    "${distro_id}:${distro.codename}-proposed";
    "${distro_id}:${distro_codename}-backports";
}
```

3. Ensure that the line with "${distro_id}:${distro_codename}-security"; is not commented out. This directive signals the utility to allow unattended-upgrades from security repositories, and only from security repositories.

4. Exit the configuration file by pressing Ctrl + X, Y, and Enter to preserve any changes you may have made.

5. Next, let's check the configuration of `/etc/apt/apt.conf.d/20auto-upgrade` to ensure that Debian is configured to periodically update package lists and that unattended-upgrades are enabled:

```
nano /etc/apt/apt.conf.d/20auto-upgrade
```

6. Ensure that the following lines appear in the file as they do here:

```
APT::Periodic::Update-Package-Lists "1";
APT::Periodic::Unattended-Upgrade "1";
```

7. With unattended-upgrades configured, de-escalate privileges by entering `exit` in the terminal.
Changes made by the unattended-upgrades utility are logged in /var/log/unattended-upgrades/unattended-upgrades.log.

The same package can be used to automate security patch applications for Proxmox VE guests running Ubuntu, Debian, or LinuxMint.

To learn more about the unattended-upgrades package, see the following documentation:
https://wiki.debian.org/UnattendedUpgrades

**Enterprise subscriptions**

Though PVE may be open source, Proxmox Server Solutions (the company behind Proxmox VE) very strongly encourages users to invest in subscriptions for Proxmox VE (https://www.proxmox.com/en/proxmox-ve/support):

“A Proxmox VE subscription is the easy and affordable solution to get access to the Proxmox VE Enterprise repository, to stable software updates and security enhancements, as well as to technical support services. A subscription helps you to run Proxmox VE with confidence in your company.”

“By combining great open source software with quality-assured services and support the Proxmox VE Subscription helps you to deploy and maintain the best stable and secure open source virtualization environment”

Security wise, there is a strong advantage to obtaining a subscription: Proxmox Server Solutions provides subscribers with access to the enterprise repository, which provides stable and “enhanced” security updates.

In contrast, users of the pve-no-subscription repositories have access to patches that are perhaps more cutting-edge, but also less stable.

Another benefit of a Proxmox subscription is access to dedicated, professional support. This is far from a commentary on community support, which has in all cases been fantastic in my experience. However, subscription support will track a ticket and promise prompt solutions. In a production environment, this can certainly make a critical difference.
Securing Proxmox VE

There are four subscription plans available: premium, standard, basic, and community (this last plan does not offer a support plan, but it does offer access to the enterprise repository). Plans are priced per month per CPU socket (https://www.proxmox.com/en/proxmox-ve/pricing).

“Subscriptions are licensed per physical server and CPU socket. In a Proxmox VE Cluster, you need to have the same subscription level for all your servers. Subscription period is one year from purchase date.”

Technical support is provided to subscribers via a web and email-based customer portal (in English or German).

Community support, by contrast, is available via the public support forum (https://forum.proxmox.com) or via IRC (the ##proxmox channel on the Freenode network).
A video tutorial is available to guide subscribers in uploading a subscription key to Proxmox VE and installing new updates at https://www.proxmox.com/en/training/video-tutorials/item/install-updates.

From a security perspective then, the pve-no-subscription repository is characterized on the site as offering patches that are not quite stable enough for production, while the enterprise repository promises enhanced security patches. You'll have to make a deliberate choice for your use case.

If you're moving services that are not mission-critical to a virtual infrastructure with Proxmox VE, perhaps community support and the pve-no-subscription options will work well for you. Otherwise, give all due consideration to an appropriate subscription option.

**Summary**

The security and information that assurance administrators can realistically provide is clearly never as exhaustive as it is exhausting.

In the first section of this chapter, you learned that hardware virtualization has inherent security benefits.

However, you also learned that many promising benefits are undermined if they're not supported by thorough planning of the virtual infrastructure, explicit policy-making up front, and a flawless deployment, all followed by unflagging policy enforcement and ongoing virtual server lifecycle management.

We then outlined threats that are either unique to virtualized infrastructures or aggravated in the context of virtualization. Each point was followed by either concrete action to mitigate the threat or links to resources for more details on addressing a potential problem.

We concluded with concrete, step-by-step remedies that could be initiated immediately, even as you continue to explore and assess Proxmox VE.
Where to buy this book

You can buy Learning Proxmox VE from the Packt Publishing website. Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.