June 27, 2017–Draft

1. Background

Windows Server 2016 introduced security features to protect VMs running on compromised hosts, where attackers can execute software as a privileged user. These security features are grouped into two main categories.

- Guarded fabric
- Shielded VMs

Guarded fabric is a collection of nodes cooperating to protect shielded VMs running under Hyper-V. These nodes are divided into a **Host Guardian Service** and one or more **guarded hosts**. Guarded hosts execute shielded VMs subject to authorization by a Host Guardian Service. For example, when a guarded host starts a shielded VM, it interacts with the Host Guardian Service, which utilizes remote attestation to confirm that the node is trusted. If so it releases a key enabling the shielded VM to be started. If a guarded host is compromised, the VM will no longer start since it cannot obtain the appropriate credentials for unlocking the VM.

A **shielded VM** is a virtual machine whose resources are protected (during execution and while at rest). Each resource is protected by one or more of the following methods:

- **Digital signing** (UEFI primary boot loader)
- **Remote attestation** (PCR measurements)
- **TPM-sealing** (disk partition keys)
- **Data encryption** (disk partitions, swap devices)
- **Process and memory isolation** (VM memory and Hyper-V execution state)

A shielded VM is represented as a virtual disk file with two kinds of partitions:

- **EFI System Partition** (**ESP**) – unencrypted partition, which contains:
    o Digitally-signed primary boot loader
    o TPM-sealed disk partition keys
- **Encrypted disk partitions** – encrypted with the disk partition keys

A shielded VM is the result of a three-stage process.

- **Preparation** – installs and configures an OS onto a virtual disk file.
- **Templatization** – converts a virtual disk file into a **shielded template**.
- **Provisioning** – creates one or more shielded VMs from a shielded template.

The lifecycle of a VM is depicted in the following diagram.
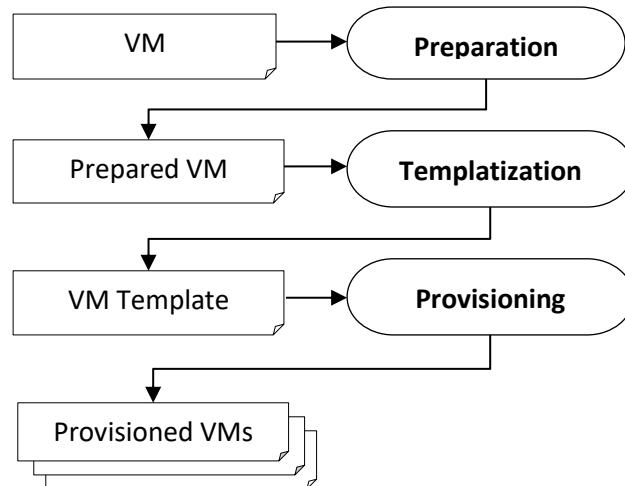
Once shielded VMs are provisioned, they may be deployed into the guarded fabric.

Support for shielding Windows VMs has been available since the launch of Windows Server 2016. Starting in Windows Server 2016 R3, this feature will be available for Linux VMs as well. This feature is called **Linux Shielded VMs** or just **LSVM**. The procedures for deploying guarded fabric and for templatizing and provisioning shielded VMs are the same. But preparing an OS image for templatization varies from Windows to Linux. **This paper focuses on preparing a Linux image to participate in this environment.**

For more background on guarded fabric and deployment of shielded VMs, please refer to the following resources:

- [Guarded fabric and Shielded VMs Overview](#)
- [Creating a Shielded Template](#)
- [Provision a Shielded VM](#)

2. Overview of LSVM

Linux Shielded VMs (LSVM) provides tools to enable Linux VMs to run as shielded Hyper-V guests. LSVM provides two main tools:
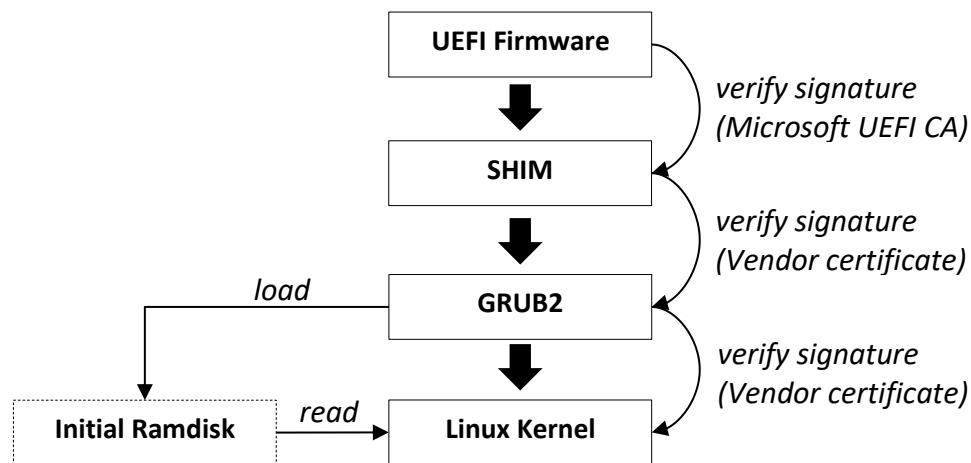
- **LSVMLOAD** – a primary boot loader for booting from an encrypted boot partition.
- **LSVMPREP –** a program that prepares the Linux environment for *templatization* and *provisioning*.

The **LSVMLOAD** boot loader is an EFI program that (1) employs TPM to unseal passphrases for unlocking the boot and root partitions, (2) sets up a virtual mapping that transparently decrypts and encrypts disk I/O requests (on the fly) from downstream EFI programs, (3) injects the disk passphrases into a non-persistent memory resident copy of the initial ramdisk, and (4) loads and executes a stock Linux Shim (SHIM), which executes GRUB2 to kick off the boot process. LSVM was designed to be as non-intrusive as possible. For example, **LSVMLOAD** operates without any modification to the SHIM, GRUB2, or the kernel. Step 2 makes this possible by setting up a virtual I/O mapping to redirect non-encrypted requests (from the SHIM and GRUB2) to the encrypted boot partition. Through this mapping, the SHIM and GRUB2 appear to be executing on an unencrypted boot partition. The boot process is described in detail later.

The **LSVMPREP** program prepares a Linux VM for the *templatization* and *provisioning* stages described in **Chapter 1**. The **LSVMPREP** program is run manually within a booted Linux image. It performs the following steps: (1) verifies that the root partition is encrypted with a well-known key, (2) encrypts the boot partition with a well-known key, (3) copies the SHIM and GRUB2 to the encrypted boot partition, (4) patches the initial ramdisk to unlock the boot partition and root partition using passphrase files injected by LSVMLOAD while booting (this replaces user interaction), (5) installs the **LSVMLOAD** boot loader onto the EFI system partition. Note that the passphrase file referred to in Step 4 is never persisted to disk. Rather it is injected into the memory blocks that are managed by the virtual disk mapping described above.
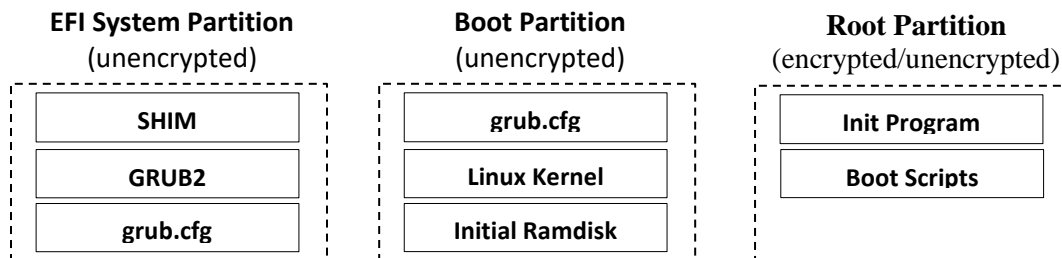
The Linux Boot Process

This section describes the existing Linux boot process or boot process. This process is the same whether Linux is booting under a Hypervisor or on bare metal. The following diagram depicts the Linux boot chain.

UEFI Firmware

verify signature (Microsoft UEFI CA)

SHIM

verify signature (Vendor certificate)

GRUB2

verify signature (Vendor certificate)

load

Initial Ramdisk — *read* → Linux Kernel

This boot chain comprises four distinct stages: (1) The UEFI firmware finds the primary boot loader (SHIM) on the unencrypted ESP (EFI system partition), verifies that it was signed by the **Microsoft UEFI CA** (Hyper-V), and if so, executes it. (2) The SHIM finds the secondary boot loader (GRUB2) on the unencrypted ESP, verifies that it was signed by the vendor certificate, and if so, executes it. (3) GRUB2 reads its primary configuration file from the unencrypted ESP and loads it to determine the location of the unencrypted boot partition. It then loads its secondary configuration file from that partition. From that configuration, it selects a Linux kernel image (**vmlinuz**) and initial ramdisk (**initrd**) to be booted. GRUB2 then asks the SHIM to verify that the kernel is signed by the vendor certificate. If so, it loads the initial ramdisk into memory and executes the kernel against that ramdisk. (4) The kernel executes the init program in the initial ramdisk. Eventually the initial ramdisk attempts to mount the real root partition, which requires user interaction to obtain a passphrase if it is encrypted.
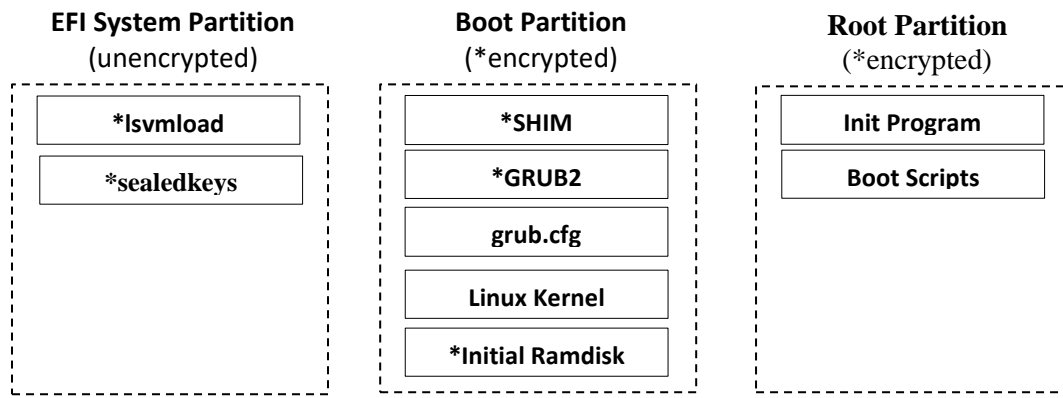
Before discussing how LSVM modifies the boot process, it is important to understand how the boot components are organized across the various disk partitions. The following diagram depicts this layout.

EFI System Partition
(unencrypted)

- SHIM
- GRUB2
- grub.cfg

Boot Partition
(unencrypted)

- grub.cfg
- Linux Kernel
- Initial Ramdisk

Root Partition
(encrypted/unencrypted)

- Init Program
- Boot Scripts

A separate boot partition is needed when the root partition is encrypted. Otherwise GRUB2 would have to interactively request a passphrase to decrypt the root partition to obtain the kernel and initial ramdisk. Then GRUB2 would have to propagate the passphrase to the initial ramdisk or allow the initial ramdisk to request the passphrase for a second time.

The LSVM Boot Process

Before **LSVMLOAD** ever boots the system, the layout of the disk has been modified by the preparation, templatization, and provisioning process (changes are depicted with an asterisk).

EFI System Partition (unencrypted)	Boot Partition (*encrypted)	Root Partition (*encrypted)
*lsvmload	***SHIM**	**Init Program**
*sealedkeys	***GRUB2**	**Boot Scripts**
	grub.cfg	
	Linux Kernel	
	*Initial Ramdisk	

The disk layout is modified as follows: (1) the root partition is encrypted, (2) the boot partition is encrypted, (3) the disk passphrases are TPM-sealed and stored on the ESP (**sealedkeys**), (4) **LSVMLOAD** is installed on the ESP, (5) the SHIM is copied to the boot partition, (7) GRUB2 is copied to the boot partition, and (8) the initial ramdisk configuration is modified and the initial ramdisk is recreated (for a complete list of component paths, please see **Appendix A**).

The LSVM boot process is depicted by the following diagram.
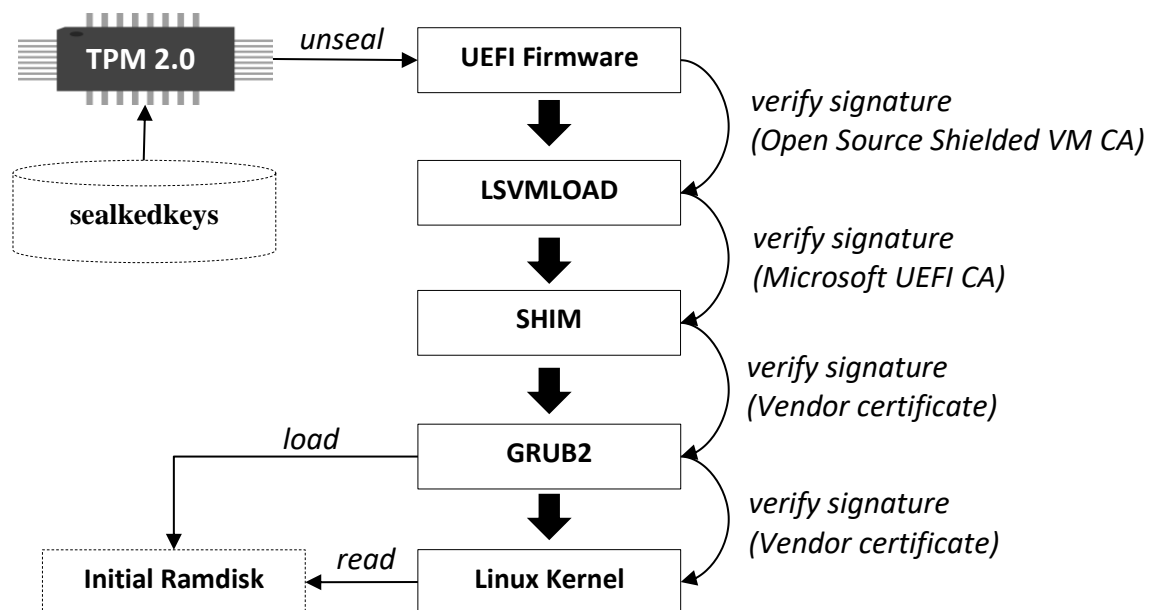
This diagram is like the original one with four key changes: (1) **LSVMLOAD** is now the primary boot loader, responsible for executing the SHIM, (2) The UEFI firmware now validates the primary boot loader using the Microsoft-issued **Open Source Shielded VM CA**, (3) **LSVMLOAD** obtains the disk encryption keys by unsealing a TPM-sealed blob (**sealedkeys**), (4) the **SHIM** and **GRUB2** and their configurations now reside on the encrypted boot disk, and (5) the initial ramdisk obtains the disk encryption passphrases from flat files (injected by **LSVMLOAD**). From **LSVMLOAD** down, the boot process is apparently identical from the point of view of the **SHIM** and **GRUB2** (by using the virtual disk mapper). For a detailed description of the LSVM boot process, please see **Appendix B**.

3. Preparing a Linux Image

This chapter describes how to prepare a Linux image for the templatization stage. The resulting image will be used as the input to the templatization phase described in **Chapter 1**. This process comprises three steps (1) creating a virtual hard disk from an ISO, (2) installing the Linux Operating System, and (3) preparing the image.

Creating a Virtual Hard Disk from an ISO

The first step is to create a virtual hard disk (VHDX) using the Hyper-V manager. First, we must choose an ISO for this new VHDX, which depends on the distribution of course, but for the purposes of this tutorial we will show how to create a virtual hard disk the following Ubuntu ISO:

```
ubuntu-16.04.2-server-amd64.iso
```

Begin by opening the **Hyper-V Manager** and then right-click on the host machine name in the left pane. Navigate the menu to:

New → Virtual Machine…

This opens the **New Virtual Machine Wizard** shown below:

Click **Next**.

Type in a name for the new VHDX and click the **Next** button, which opens the window shown below.

Select **Generation 2** (to create a Generation 2 virtual hard disk) and click **Next**, which opens the following Window.

New Virtual Machine Wizard

✖

Assign Memory

Before You Begin
Specify Name and Location
Specify Generation
Assign Memory
Configure Networking
Connect Virtual Hard Disk
    Installation Options
Summary

Specify the amount of memory to allocate to this virtual machine. You can specify an amount from 32 MB through 12582912 MB. To improve performance, specify more than the minimum amount recommended for the operating system.

Startup memory: 1024 MB

☑ Use Dynamic Memory for this virtual machine.

ⓘ When you decide how much memory to assign to a virtual machine, consider how you intend to use the virtual machine and the operating system that it will run.

< Previous   Next >   Finish   Cancel

You can change the **Startup memory** setting or accept the default. Click **Next**.

Select a network adapter to use or leave blank and configure later. Click **Next**.

This window creates a new virtual hard disk. Just accept the defaults and click **Next**.

Select **Install an operating system from a bootable image file** and then specify the path of the Linux ISO. Click **Next**.
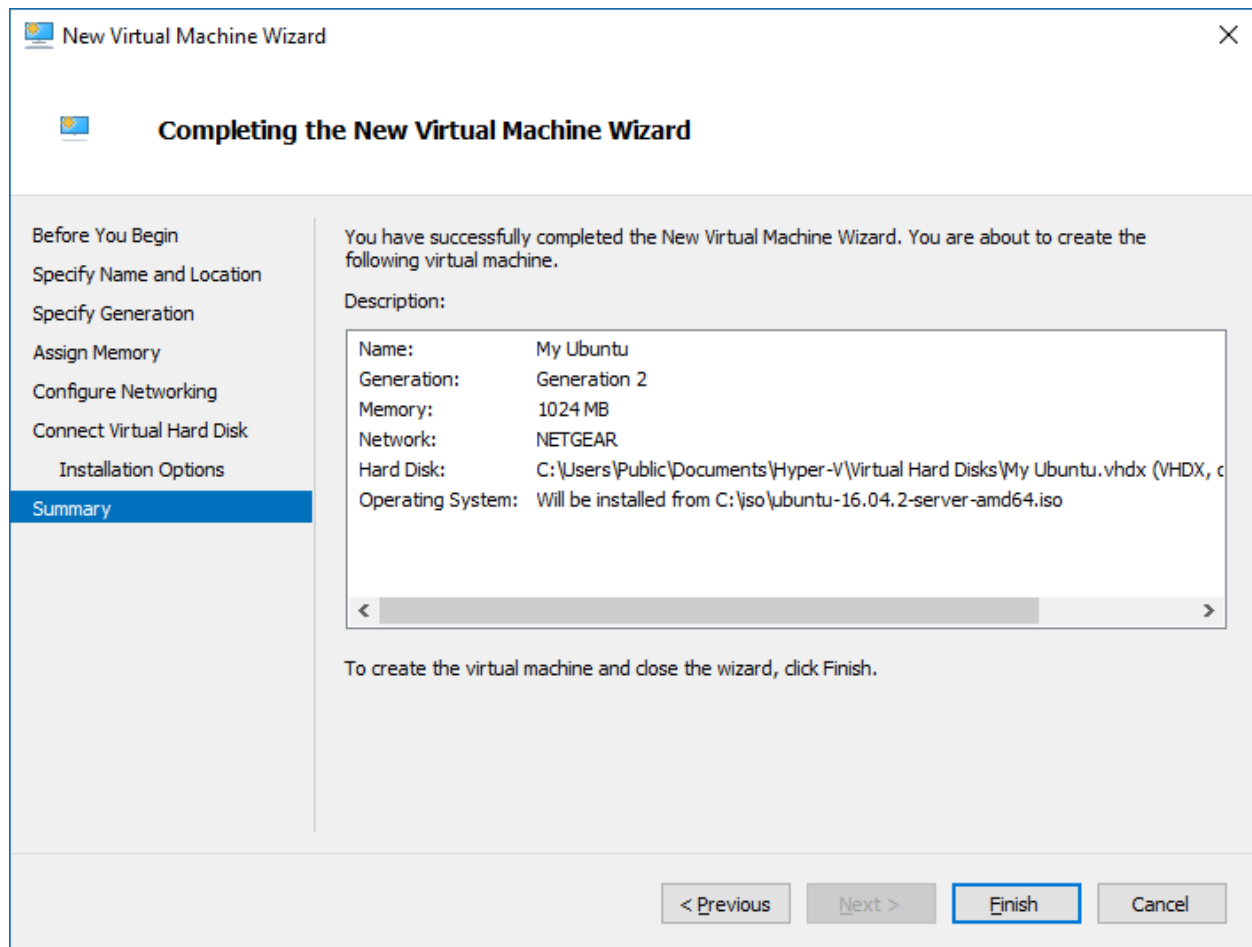
Review the information in the **Description** window and click **Finish**. This closes the New Virtual Machine Wizard.
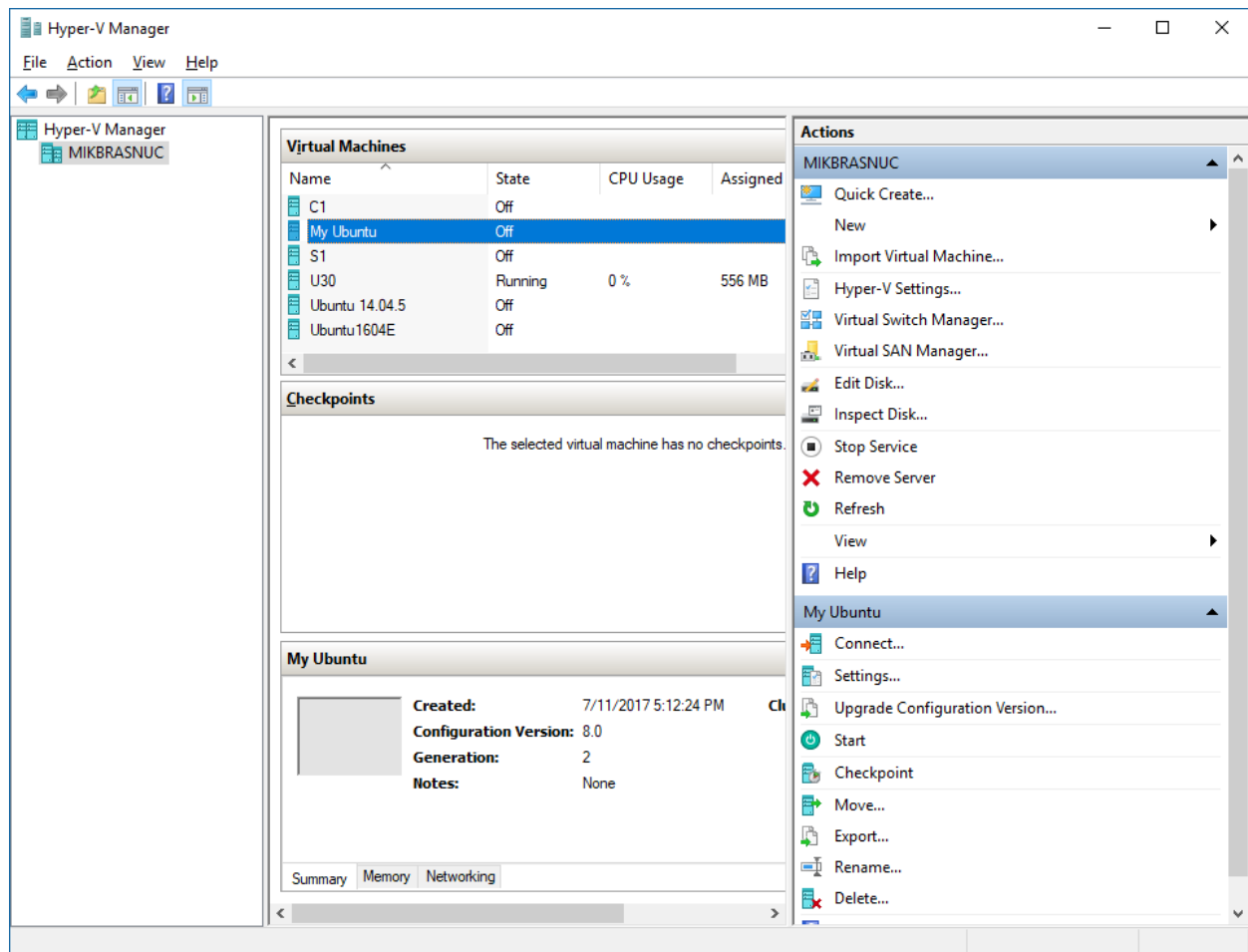
On the Hyper-V Manager window, right-click on the **My Ubuntu** virtual machine (highlighted above) and then select:

Action→ Settings…

Which opens the following window.
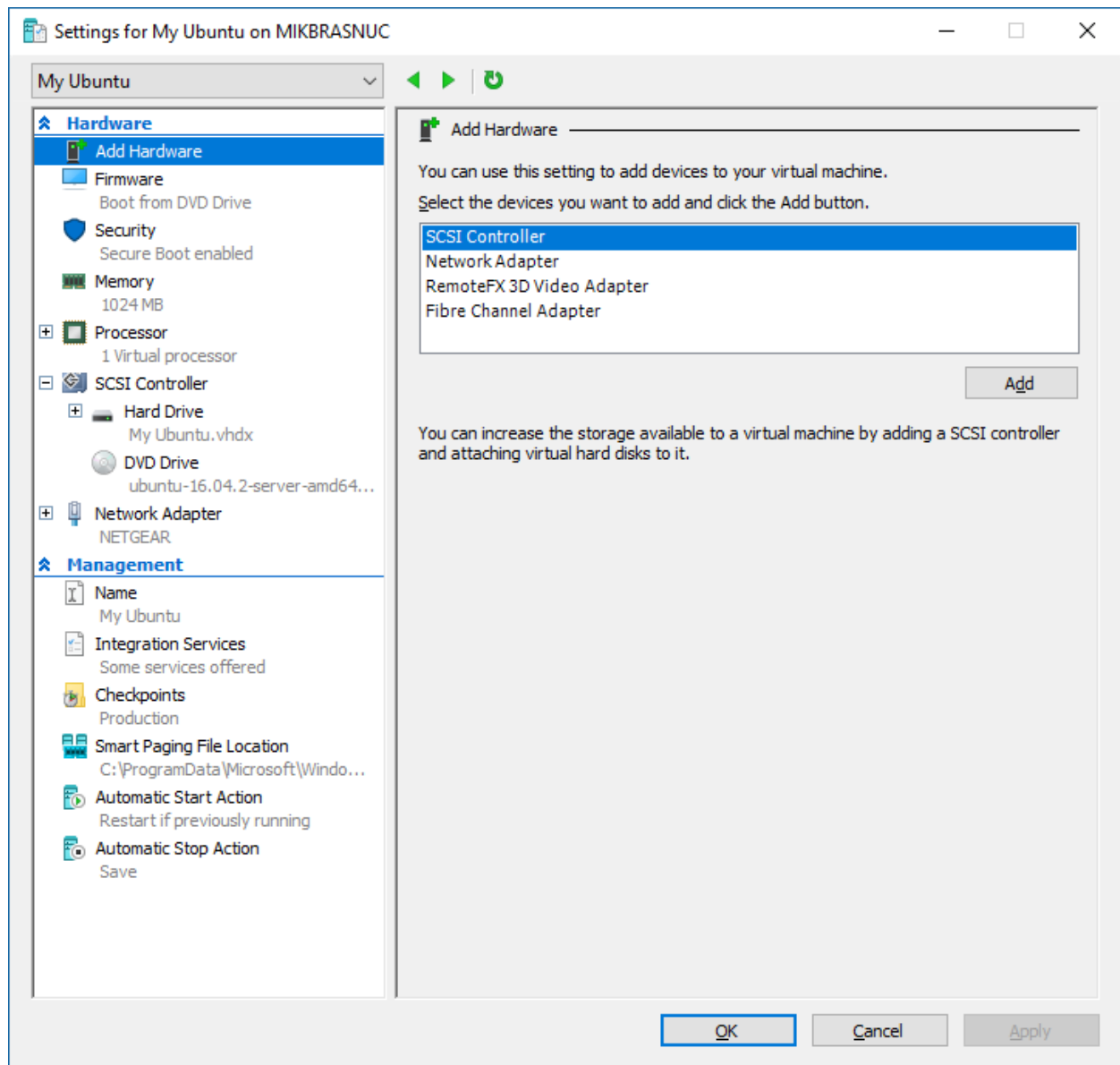
In the left pane select **Security**, which opens the following right pane.
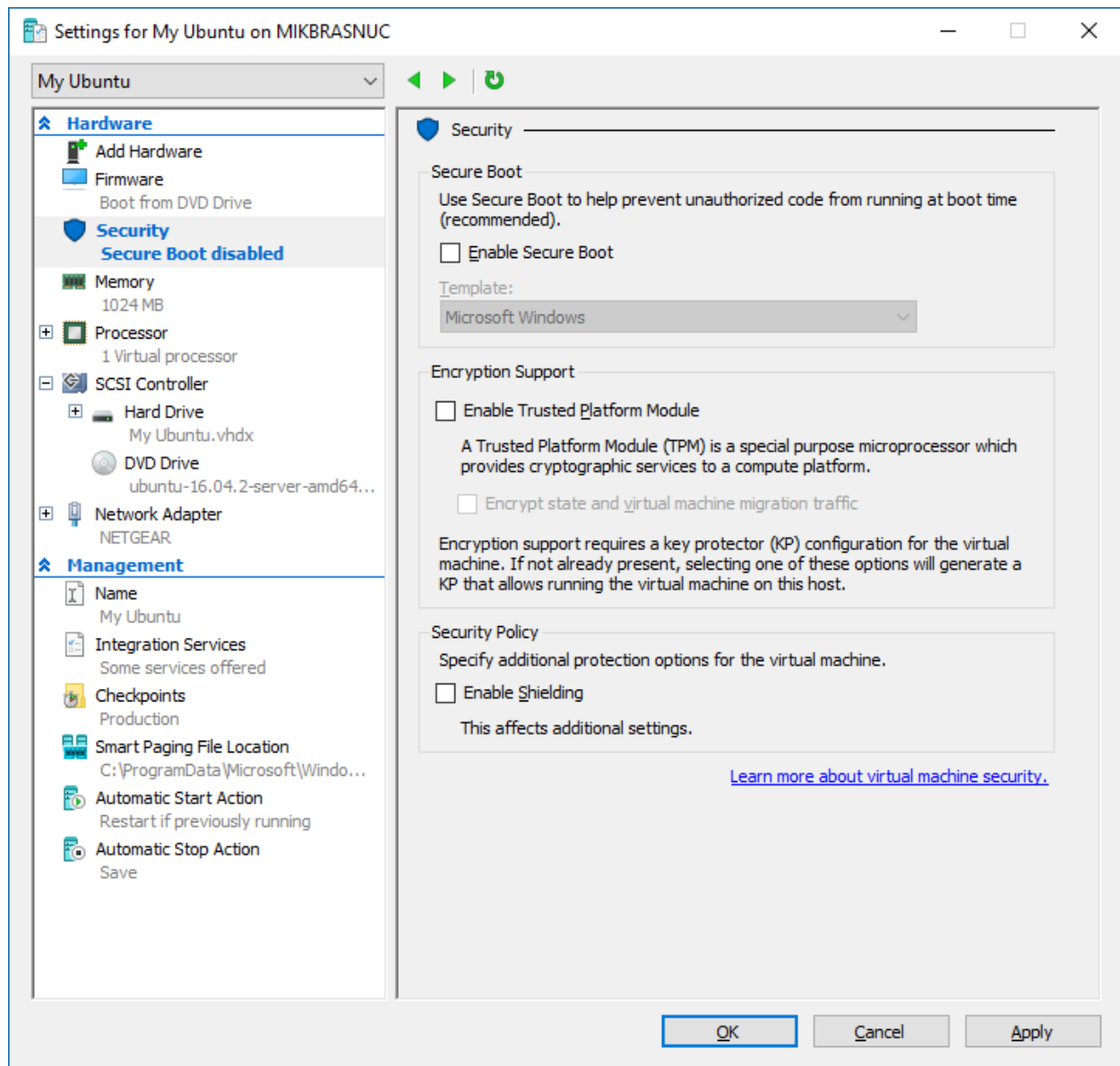
Unselect **Enable Secure Boot** (for now), and click **Ok**.

This completes the create of the VHDX file.

Installing the Linux Operating System

The next step is to install the Linux Operating System. Open the Hyper-V Manager window as shown below.
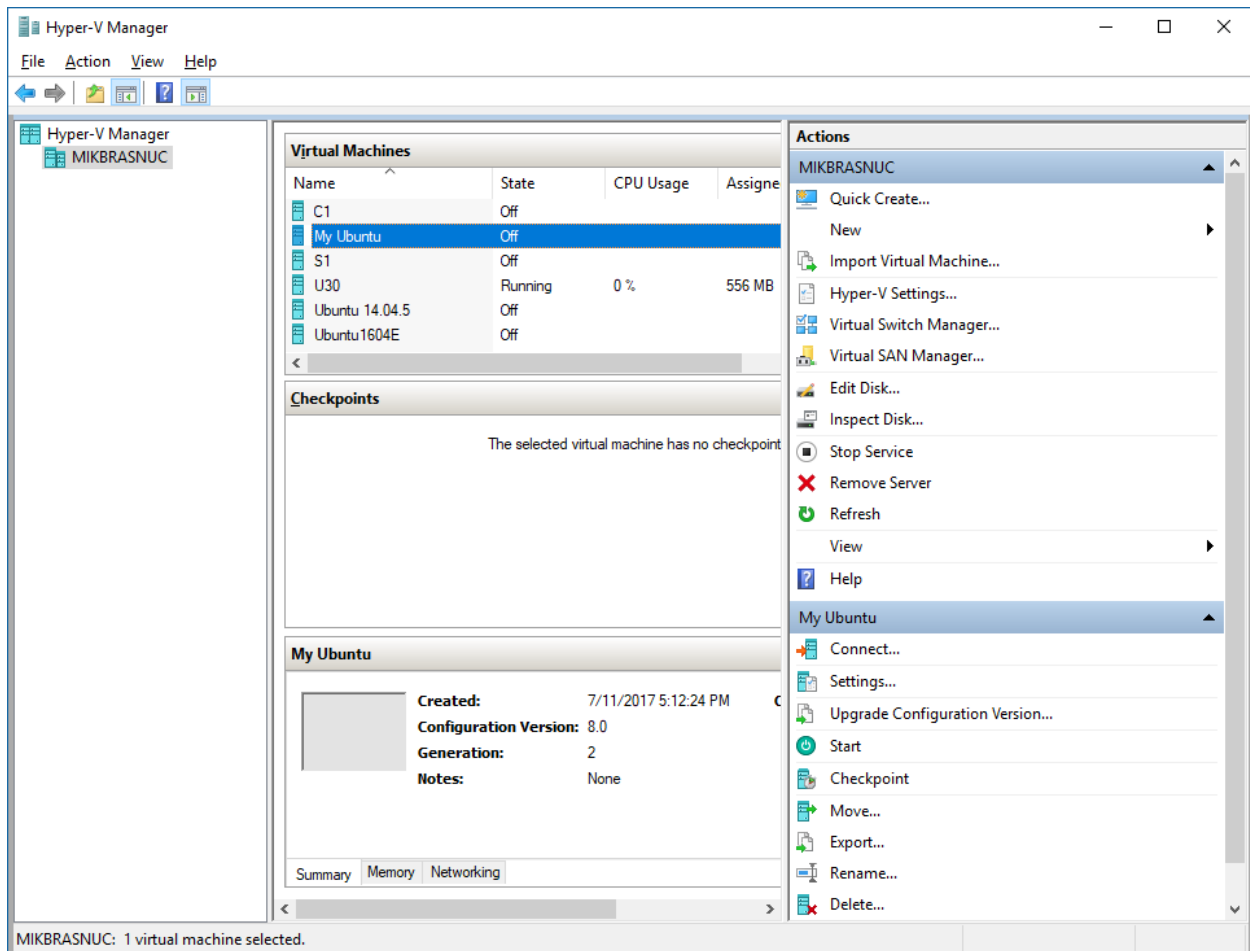
Right click on the **My Ubuntu** and select **Start**. And then right click on **My Ubuntu** and select **Connect**. This opens the console window.

Use the console Window to install the given distribution. Installation varies from distribution to distribution, but here are some common requirements for the resulting distribution:

- Must be enabled for UEFI boot (must have the `/boot/efi/EFI/boot`).
- Must have an **unencrypted** boot partition mounted on /boot
- Must have an **encrypted** root partition whose passphrase is "passphrase".

Ubuntu: when picking a partitioning scheme, select

```
Guided – use entire disk and set up encrypted LVM
```

Once the installation is complete, proceed to the next section.

Preparing the Linux Image

Preparing the Linux image consists of running the LSVMPREP command on a booted Linux image. Boot into the Linux image and obtain the LSVMTOOLS package.

```
lsvmtools-1.0.0-x86_64.tar.gz
```

Next install the LSVMTOOLS with the following commands.

```
# tar zxvf lsvmtools-1.0.0-x86_64.tar.gz
# cd lsvmtools-1.0.0
# ./install
```

Finally run the **LSVMPREP** command.

```
# cd /opt/lsvmtools-1.0.0
#./lsvmprep
Checking /boot: encrypting
Command successful.
mke2fs 1.42.13 (17-May-2015)
Creating filesystem with 497664 1k blocks and 124440 inodes
Filesystem UUID: 8d652a9c-3847-433f-bf14-1623f443d3c9
Superblock backups stored on blocks:
        8193, 24577, 40961, 57345, 73729, 204801, 221185, 401409

Allocating group tables: done
Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

Patched /etc/crypttab
Patched /etc/fstab
Creating /sbin/lsvmtool
Creating /boot/efi/EFI/boot/bootx64.efi
Creating /boot/efi/EFI/boot/lsvmconf
Creating /boot/lsvmload/shimx64.efi
Creating /boot/lsvmload/grubx64.efi
Creating /boot/grub/grub.cfg
Patching /usr/share/initramfs-tools
Patching /etc/initramfs-tools
Updating initial ramdisks
```

The LSVMPREP program performs the following tasks.

- Encrypts the boot partition with a well-known passphrase: "passphrase".
- Patches /etc/crypttab and /etc/fstab to mount the LUKS-encrypted boot partition.
- Installs LSVMLOAD: /boot/efi/EFI/boot/bootx64.efi.
- TPM-seals the passphrases and stores them on the ESP.
- Copies the **SHIM** from the ESP to the encrypted boot partition.

- Copies **GRUB2** from the ESP to the encrypted boot partition.
- Relocates **grub.cfg** from the ESP to the encrypted boot partition.
- Patches the configuration of the initial ramdisk (initrd) to read passphrases from flat files injected by **LSVMLOAD**.
- Regenerates the initial ramdisks and recreates **grub.cfg**.
- Applies UEFI dbx variable updates (for black listing of compromised loaders).

Once these steps are complete, the image is ready for the templatization step.

4. Templatizing and Provisioning the Linux VM

The templatizing and provision process is similar for Windows and Linux. Refer to Windows documentation for this process.

Appendix A – Location of Boot Components

Location	Description
`/boot/efi/EFI/boot/bootx64.efi`	LSVMLOAD boot loader
`/boot/efi/EFI/boot/lsvmconf`	LSVMLOAD boot loader configuration file
`/boot/efi/EFI/boot/sealedkeys`	TPM-sealed disk passphrases
`/boot/LSVMLOAD/shimx64.efi`	Relocated shimx64.efi
`/boot/LSVMLOAD/grubx64.efi`	Relocated grubx64.efi
`/boot/grub/grub.cfg`	GRUB2 configuration file
`/boot/initrd.img-*`	Initial ramdisk
`/boot/vmlinuz-*`	Linux kernel

Appendix B –LSVM Boot Steps

1. The UEFI firmware locates **LSVMLOAD** (`/boot/efi/EFI/boot/bootx64.efi`) and verifies that it has been signed by the **Microsoft UEFI CA**. If so, the UEFI firmware executes **LSVMLOAD**.
2. **LSVMLOAD** reads its configuration file (`/boot/efi/EFI/boot/lsvmconf`) to obtain the partition identifiers for the root and disk partitions.
3. **LSVMLOAD** verifies that the TPM-chip is present and aborts if not.
4. **LSVMLOAD** performs certain TPM measurements (Linux scenario measurements).
5. **LSVMLOAD** unseals the passphrases (`/boot/efi/EFI/boot/sealedkeys`) for the encrypted boot and root partitions. If this fails, **LSVMLOAD** obtains the passphrases through interactive input.
6. **LSVMLOAD** caps the TPM's PCR-11 (preventing downstream components from unsealing the passphrases.
7. **LSVMLOAD** attempts to unlock the LUKS-encrypted **boot** partition with the boot passphrase.
8. **LSVMLOAD** attempts to unlock the LUKS-encrypted **root** partition with the root passphrase.
9. **LSVMLOAD** decrypts the specialization file (if any) and copies it to the encrypted root partition.
10. **LSVMLOAD** sets up a virtual mapping to intercept disk I/O requests from the **SHIM** and **GRUB2** and transparently redirect them to the encrypted boot disk.
11. **LSVMLOAD** creates an unencrypted pseudo EXT2 partition that maps to the original encrypted boot partition.
12. **LSVMLOAD** sets up a virtual VFAT ramdisk to intercept disk I/O requests intended for the ESP.
13. **LSVMLOAD** writes a **grub.cfg** file into the VFAT ramdisk. This configuration file contains the partition identifier for the pseudo-partition created in Step 11.
14. **LSVMLOAD** loads the initial ramdisk from the boot partition into memory and into the virtual mapping.
15. **LSVMLOAD** injects the passphrases for root and boot partitions into the memory resident copy of the initial ramdisk. Theses blocks are never synced to disk and so these secretes are not persisted on the encrypted boot partition.
16. **LSVMLOAD** loads the **SHIM** into memory and verifies that it has been signed by the Microsoft UEFI CA and executes it if so.
17. The **SHIM** attempts to read **GRUB2** from the ESP into memory. These read requests are intercepted by **LSVMLOAD**, which transparently redirects reads on this file to the encrypted boot partition.
18. The **SHIM** verifies that GRUB2 was signed by the vendor certificate, and if so executes it.
19. **GRUB2** attempts to read its configuration file (**grub.cfg**) on the ESP but is transparently redirected to the VFAT ramdisk that was set up by **LSVMLOAD**.

20. **GRUB2** reads the boot disk partition ID from its configuration (which is the pseudo-partition set up by the **LSVMLOAD**).
21. **GRUB2** reads its secondary configuration file (`/boot/grub/grub.cfg`) from the boot partition. These reads are intercepted by **LSVMLOAD** and redirected to the encrypted boot partition.
22. **GRUB2** reads the initial ramdisk (`/boot/initrd-*`) from the boot partition. Again, these reads are intercepted by **LSVMLOAD**.
23. **GRUB2** reads the kernel (`/boot/vmlinuz-*`) from the boot partition. Again, these reads are intercepted by **LSVMLOAD**.
24. **GRUB2** asks the **SHIM** to verify that the kernel is signed by the vendor certificate and if so, executes the kernel against the initial ramdisk image.