Linux Shielded VMs Overview

September 26, 2017

Key technologies: guarded fabric, shielded VMs, hypervisor, UEFI, EFI, TCG, measured boot, secure boot, boot loader, digital signing, TPM 2.0, AES encryption, disk encryption, LUKS, EXT2, EXT3, VFAT, COFF, SHA-256, I/O virtualization, Microsoft Windows, Hyper-V, OpenSSL, TCG.

1. Introduction

This paper discusses the **Linux Shielded VMs** (LSVM) project whose aim is to protect Linux VMs running on potentially compromised nodes. A ***compromised node*** is a computer that has been hacked so that an attacker may run software in privileged mode. In a cloud computing environment, customers build virtual machines that embed confidential data, which are deployed into a cloud, which is managed by a third party. The customer may regard this environment as untrusted since (1) the cloud may also be running virtual machines designed by competitors and possibly hackers and (2) the whole cloud is an ongoing target of attack.

There are many ways a hacker can attack a virtual machine after taking control of the hosting node. A full description of these attacks is beyond the scope of this paper, but here are some common ones.

- A hacker may access the virtual disk file, either by reading secrets from this file or by injecting malicious software to take control of the virtual machine.
- A hacker may access the virtual machine's execution state, such as memory pages, swapped memory, or virtual machine configuration settings.
- A hacker may completely replace the hypervisor so that the virtual machine is running under the full control of malware.

For the purposes of this paper, we employ the terms guest and host. The ***guest*** is a virtual machine executing on a given node, and the ***host*** is the node that executes (i.e., hosts) that virtual machine (including its hypervisor). As we will see in the next chapter (when discussing Guarded Fabric), the term host is expanded to include the entire ***hosting environment***, which refers to a set of nodes working together to shield virtual machines from attack.

So how can virtual machines be protected from attack? Solutions must be provided for both the host and the guest. Fortunately, the host-side solutions were well hashed out by the Windows shielded VM work, discussed in the next section. And the architecture of the guest-side solutions is already addressed for Microsoft Windows. This leaves us to focus on improvements to the Linux guest that will allow it to leverage this existing security infrastructure.

2. Guarded Fabric and Windows Shielded VMs

Windows Server 2016 introduced security features to protect Windows VMs (guests) 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 start up. 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)

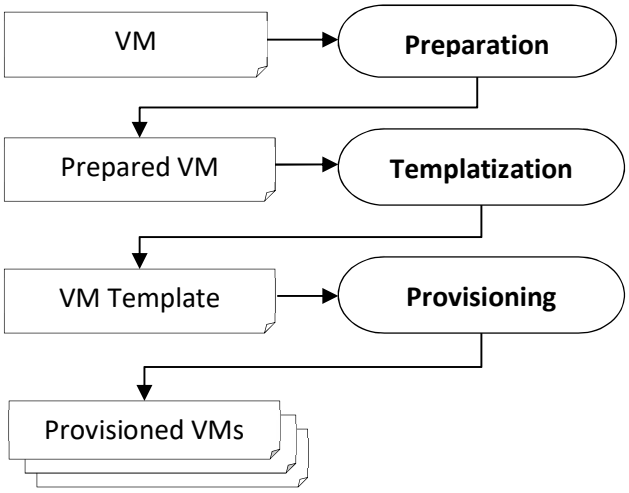
A shielded VM is represented as a virtual disk file with two kinds of partitions:

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

A shielded VM is the product 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.

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

3. Linux Shielded VMs (LSVM)

Linux Shielded VMs (***LSVM***) provide tools to enable Linux VMs to run as shielded Hyper-V guests. LSVM provides two main tools:

- **LSVMLOAD** – a primary boot loader for booting an encrypted boot partition.
- **LSVMPREP –** a program that prepares the Linux environment for *templatization* and *provisioning*.

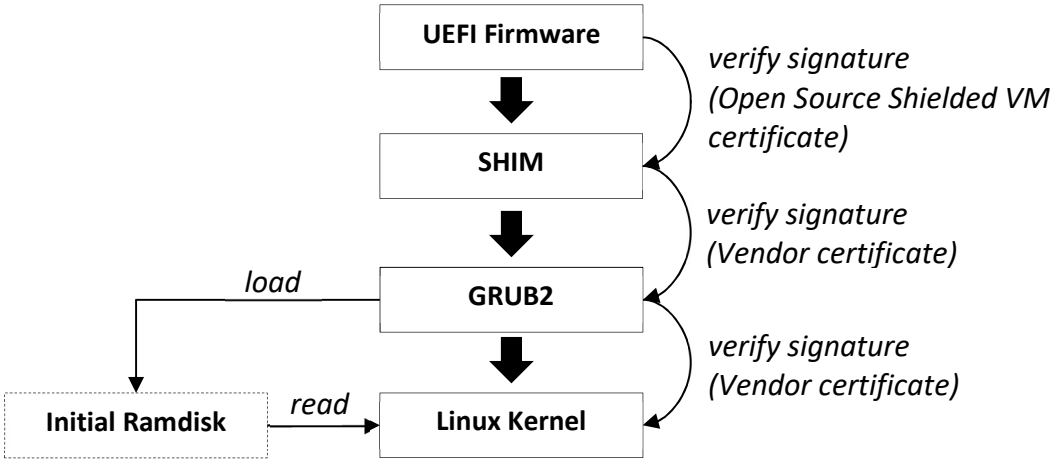
The **LSVMLOAD** boot loader is an EFI program that (1) employs TPM to unseal passphrases for unlocking the boot and root partitions, (2) sets up a virtual mapping that transparently decrypts and encrypts disk I/O requests (on the fly) from downstream EFI programs, (3) injects the disk passphrases into 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 2**. 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 memory blocks that are managed by the virtual disk mapping described above.

The Linux Boot Process

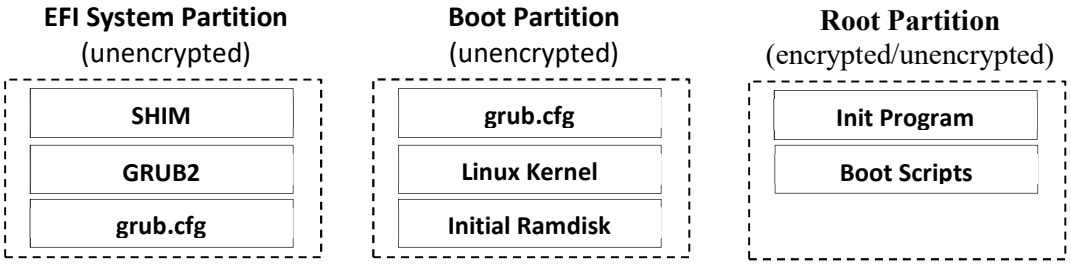
This section describes the existing Linux boot process or boot process. This process is the same whether Linux is booting under a Hypervisor or on bare metal. The following diagram depicts the Linux boot chain.



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 **Open Source Shielded VM** certificate (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.

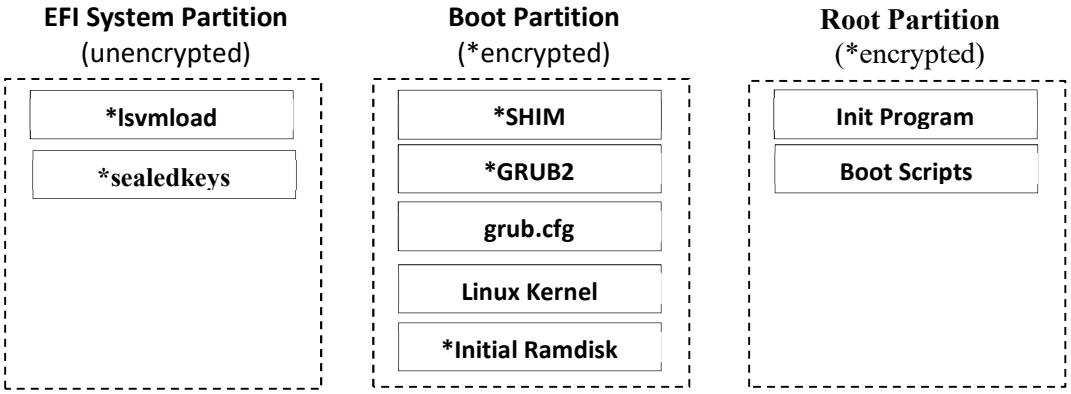
Before discussing how LSVM modifies the boot process, it is important to understand how the boot components are organized across the various disk partitions. The following diagram depicts this layout.



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

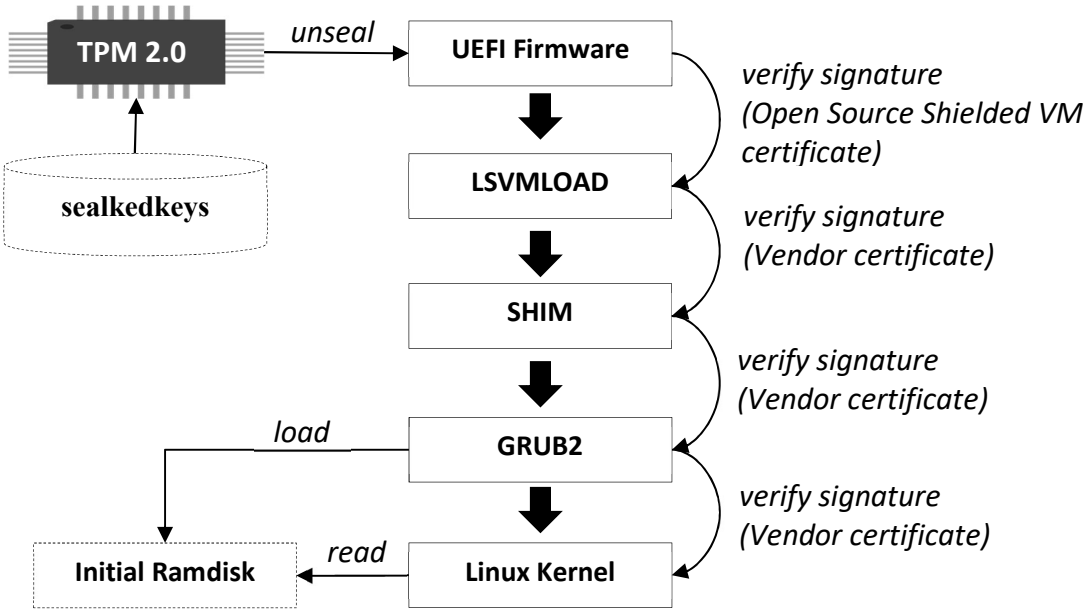
Before **LSVMLOAD** ever boots the system, the layout of the disk has been modified by the preparation, templatization, and provisioning process (changes are depicted with an asterisk).



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 encrypted boot partition, (6) GRUB2 is copied to the encrypted boot partition, and (7) the initial ramdisk configuration is modified and the initial ramdisk is regenerated from this configuration.

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).

4. The Pieces of LSVMLOAD

LSVMLOAD is a complex EFI program that required new implementations of the following pieces for EFI:

- **TPM 2.0 client stack** (to communicate with EFI and Intel driver)
- **COFF image loader** (to load the SHIM, GRUB2, or the Linux kernel)
- **Certificate verification** (validate signed images)

- **EXT2 driver for EFI** (process contents of boot partition)
- **VFAT2 driver for EFI** (ESP processing)
- **VFAT RAM file system** (I/O virtualization)
- **LUKS implementation** (disk encryption)
- **CPIO library** (processing of initial ramdisk)
- **GUID Partition Table library** (partition identification)
- **Disk block manager** (to support EXT2 and VFAT file systems)
- **I/O mapping layer** (transparent translation between encrypted and decrypted disks)