Network-based Root of Trust for Installation

Joshua Schiffman, Thomas Moyer, Trent Jaeger and Patrick McDaniel

Systems and Internet Infrastructure Security Laboratory
Computer Science and Engineering Department, Pennsylvania State University

Administrators of large data centers often require network installation mechanisms, such as disk cloning over the network, to manage the integrity of their machines. However, network-based installation is vulnerable to a variety of attacks, including compromised machines responding to installation requests with malware. To enable verification that running machines were installed correctly, we propose a network-based Root of Trust for Installation (netROTI), an installer that binds the state of a system to its installer and disk image. Our evaluation demonstrates that a netROTI installation adds about 8 seconds overhead plus 3% of image download time to a standard network install and thwarts many known attacks against the installation process.

1 Introduction

Data centers and large enterprises often use network installation and monitoring to manage the integrity of their deployed systems. This enables administrators to focus on hardening fewer systems that are cloned over the network to a multitude of machines. Once installed, remote monitoring tools and services can be used to automate the process of detecting anomalies in system behavior, intrusions, and illicit modifications to the filesystem. These mechanisms aim to provide *trusted distribution* where by an administrator can verify a system was installed as intended and that nothing has secretly modified the system since installation [10]. Without trusted distribution, it is difficult to ensure the ongoing correctness of a system at runtime.

Unfortunately, network-based installation introduces new challenges to the already difficult process of verifying system installation. The most common method of initializing a network installation is to load a bootstrap program over the network, thus eliminating the need for installation media like optical disks. However, the addition of network access opens the possibility for malicious parties to compromise installer code or corrupt the disk image in flight. Even after installation, malicious modifications can compromise security-critical files, which may be difficult to detect in specialized files like configuration scripts that lack a well known correct state. In the presence of such subversive code and hard to verify data, monitoring tools may be tricked into reporting that nothing wrong has happened. Ultimately, a method is need for proving to an administrator that a system has been securely installed and not been modified since that installation.

To achieve this goal, we propose a network-based Root of Trust for Installation (netROTI), an installation method that links a filesystem to its installer and the disk image used prior to configuration. If an administrator trusts their installer and disk image, then they can trust systems booted from filesystems derived from such an installation. Using the netROTI approach, administrators can configure their hardened images, and run the netROTI to install all of their machines automatically. They can then receive a proof from each machine, showing if it was booted from a compromised filesystem. An implementation of the netROTI for a Eucalyptus [14] cloud environment adds only an 8 second fixed overhead plus 3% of image download time to the network installation process, and verification can be automated for the administrator. The result is that secure network installation, even over an untrusted network, can be automated.

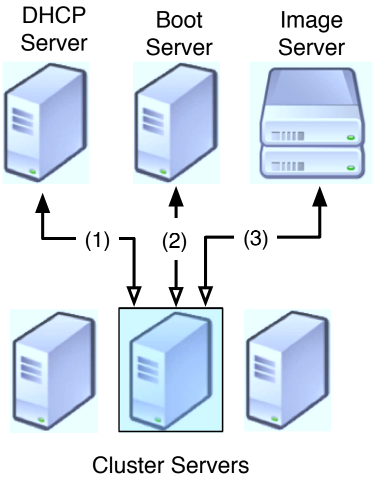

Figure 1: A network install bootstrapped via PXE Boot. The client system in blue loads the PXE Boot firmware, which (1) initiates a DHCP request on the local subnet to setup basic networking and locate a Boot Server. After obtaining this address, (2) the client requests a Network Boot Program (NBP) from the Boot Server and executes it. The bootloader may request additional files from the Boot Server such as modules and an initial ramdisk to setup the installer client environment. Finally, (3) the installer connects to the Image Server and begins transferring the disk image to the target system's hard disk. After configuring the image, the system reboots.

2 Network Boot Installation

In this section, we first describe the process of network installation. Next, we discuss the possible attacks on this procedure and security guarantees required for a trusted installation.

2.1 Current Network Installation

Companies and universities with large system deployments install and maintain their systems differently from the typical desktop user. While individual systems are typically installed using optical disk or a USB drive, the long installation process, need for physical media, and specific customizations for that environment make it impractical to use the same approach for hundreds of machines. Instead, network-based installation techniques using customized automated installer images or disk cloning are used to rapidly upgrade out-of- date systems or restore compromised servers to their proper state.

From speaking with administrators in several large companies and our own university, which supplies computing resources for over 40,000 students, we found the most common disk cloning tools to be Symantec's Norton Ghost [2], Acronis True Image [6], or custom designed tools that use a variety of free and open source utilities. Other services automate installation tasks like Microsoft's Windows Deployment Services [13] and Rocks [5]. These tools function by loading a client at boot time over the network that connect to a management server download files like a pre-configured disk image or installer programs. This reduces deployment time and allows administrators to harden a single installation and replicate it among systems that perform similar tasks like VMMs in a cloud or employee workstations.

While there are several methods of bootstrapping an installer client, one of the most common methods is the Preboot Execution Environment (PXE) [3]. Figure 1 briefly illustrates a network install using the PXE protocol. First, the system to be imaged (we call the client) boots into the PXE firmware (usually loaded by the BIOS from the NIC's firmware). Next, the client starts the protocol by (1) broadcasting a

DHCPDISCOVER request on port 67 with additional PXEClient extension tag. A DHCP or ProxyDHCP server responds with a DHCPOFFER on port 68 providing an IP address and a list of Boot Servers. The client then (2) sends a DHCPREQUEST to a Boot Server and gets a DHCPACK message with the file name of a Network Boot Program (NBP) it retrieves from the Boot Server via TFTP. The client then executes the NBP, which may request additional files such as a kernel or modules required for the client's hardware.

The installer client is setup by the NBP either using files download from the Boot Server or by retrieving them using protocols like NFS or HTTP. Finally, the client (3) contacts the Image Server and requests a disk image. After the image has been written to the hard drive, the client performs additional configuration steps like setting the hostname and networking. Finally, the machine reboots into the newly imaged OS.

2.2 Attacks on Network Installation

Ensuring the correct operation of systems within large installations like a data center, requires the administrators to be able to prove their systems have been installed and configured with high integrity code and data. While the techniques mentioned above automate installation, they do not do enable administrators to verify whether the system has booted from a properly installed filesystem. Potential attacks on the installation procedure or modification of systems later could corrupt a server and lead to a host of attacks from within the data center. We now consider some of these attacks and then discuss the guarantees that must be satisfied to ensure the a server has booted from a high integrity installation.

The first place a server can be corrupted is during installation. In the process described in Figure 1, the client system could potentially load a malicious PXE firmware from the NIC installed during a previously compromised state. Other attacks have been demonstrated [4] that allow remote attackers to compromise NIC firmware over the network. In either case, such firmware could lead to direct attacks on the system's memory. Another vector for attack exists when the PXE client searches for the Boot Server. Since the PXE client relies on information from local DHCP or Proxy DHCP servers, a compromised server acting as a rogue DHCP server on the local subnet could trick the client into downloading a malicious NBP and install a rootkit. At the network level, an attacker could modify data sent on the wire to the client if unencrypted or perform an man-in-the-middle attack to tamper with the installation. Even after installation, a system may be vulnerable. Numerous attacks exist that place rootkits or make malicious changes to the filesystem that persist even after a system reboots.

2.3 Securing Network Boot Installation

To secure network installation, it is necessary to show the installed system is derived from the expected origins, installer, and disk image. While not everyone may trust the installer or disk image, those that do would be willing to work with such a system if it could be verified. In this case, we envision that large data center administrators would be able to leverage such trust because they specify the installer and disk images that can be loaded.

In order to verify a system's installation, a method is needed for accurate measurement and reporting. Recent work in trusted computing has examined the challenge of building trust in commodity systems. Trusted hardware such as the Trusted Platform Module (TPM) and extensions added to Intel and AMD processors offer various trust primitives. Using this hardware support, systems can generate *attestations* of a platform's critical code and data, which remote parties can verify. Parno *et al*'s survey [15] examines the broad range of applications to which researchers have applied these trust primitives.

Verifying installation is not very useful by itself, however, as the machine will be immediately rebooted after installation and may be rebooted multiple times before any subsequent re-installation. Thus, any network installation must enable verification that a system was booted from an expected installation. Thus, our method enables an administrator to verify that the filesystem at boot time is linked to the installation origins, the installer and disk image. We note that this does not prevent the system from coming under runtime attacks, such as buffer overflows, but these attacks will be detected if they modify the filesystem on the next reboot.

Any secure network installation must be practical. A key question is whether the installed filesystem is sufficiently stable to enable such a verification. In an initial experiment [9], we found that only three files of privileged VM system configuration were modified dynamically during its execution. Also, manual updates to systems are prohibited in our approach, as they are ad hoc. For administrators, a clean, automated install is preferable to manual modifications anyway. An administrator may push specific updates to all their systems automatically, but these cannot be linked to the installer. We envision that software on the installed system can extend the install-time proofs, if authorized.

3 The netROTI Method

We now introduce netROTI, a network-based installation method that links the resulting system verifiably to a particular source. We first define our trust and threat models to establish the scope of our solution and then detail how the netROTI augments the installation process. Finally, we describe the protocol used to verify the filesystem's origin.

3.1 Trust and Threats for Designing a netROTI

For our design, we assume a trust model where the physical hardware is safe from attack and is implemented correctly. We also trust there exists an administrator or software provider with the authority to deem particular code and data (e.g., the installer and disk image) as trustworthy. While we make no assumptions that such trust is placed correctly, our goal is to prove that a system is linked to a particular origin certified by one or more authorities. Thus, a verifier can determine their trust in a system based on its trust in the ability of authorities to certify their systems. We also trust the data center administrators and hence not addressing insider attacks. Finally, we do not consider attacks on the cryptographic algorithms used nor attacks on the PKI or authentication procedures like direct anonymous attestation [8] to establish identities.

For our threat model, we consider an attacker that can modify or inject data on the network, is able to impersonate various services, and compromise other hosts on the network. These attacks could lead to the client loading a malicious installer, installing a vulnerable or malicious disk image, or compromise of device firmware. The attacker can change the contents of the client's disk after installation and perform attacks on the running system. Reporting attacks on the system's runtime state is outside the scope of this work, but the netROTI does provide a root of trust for detecting these security violations by giving a proof of the systems initial integrity at boot time.

3.2 The netROTI Overview

The netROTI approach is a network-based system installation method that cryptographically links the installed filesystem with the installer and source used in the installation. Figure 2 illustrates each phase of the installation procedure. The *preinstall phase* highlighted in green is a trusted, manual step requiring the administrator to configure the client to boot from the network and to prepare the client's Root of Trust for Measurement (RTM) used to record and report critical code and data. Since the tasks in this phase are performed manually, we trust them axiomatically. Next, the client gathers the necessary files to install from the network in the *gather phase*, which is shown in red because it need not be run by trusted code and is unmeasured. Once collected, the system enters the subsequent blue phases, which contribute to building a *ROTI proof* linking the installer and image to the client's filesystem. The *bootstrap phase* initializes a secure execution environment for the installer after the RTM measures it. The installer downloads and measures

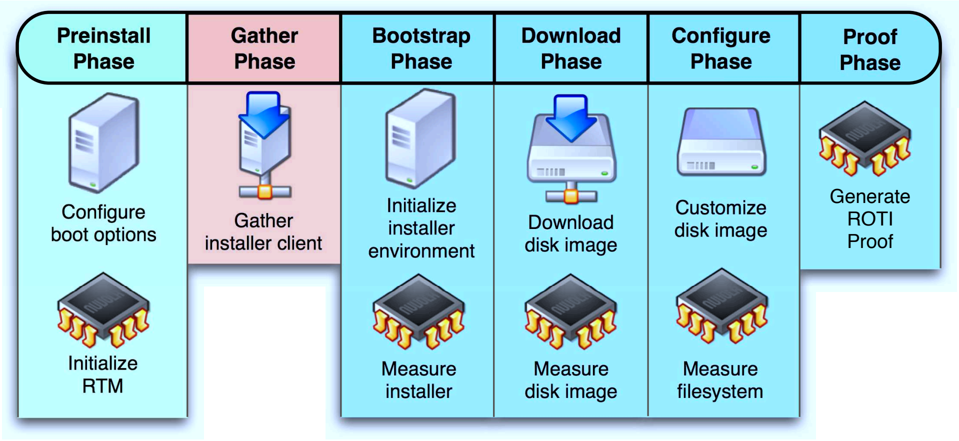

Figure 2: Timeline of the installation process. The administrator configures the client in the preinstall phase. The client then downloads the installer and bootstraps a secure environment, which measures the installer. Next, the client downloads, measures, and configures a disk image to place on the local disk. Finally, the resulting filesystem is measured and a proof of the system's Root of Trust for Installation (ROTI proof) is generated.

the image to be installed in the *download phase*. Next, it configures and measures the resulting filesystem in the *configure phase*. In the final *proof phase*, the RTM generates a ROTI proof later used by the system at runtime to produce attestations, which a verifier can use to identify the installer and disk image used to configure the filesystem from which that client booted.

3.3 The netROTI Installation Phases Detailed

Each of the netROTI installation phases has a specific goal and tasks to achieve that goal.

Preinstall Phase The preinstall phase is a manual process carried out by the system administrator to prepare the the client system for installation. This phase is needed to configure components that enable generation of ROTI proofs. To prepare the system, the administrator configures the BIOS to boot from the network and installs the RTM with the keys necessary for it to identify this client uniquely.

Gather Phase The goal of this phase is to retrieve the installation image and installer. We need not measure this phase as we will start the install from a known state using only these inputs starting the next phase. First, the client machine loads the network boot firmware to obtain network access and locate the Boot Server. It then retrieves an NBP that downloads the additional installer files, the installer kernel, a ramdisk containing the installer code, and a bootstrap program that sets up a secure environment for the installer.

Bootstrap Phase Since the previous install phase performed unmeasured operations, there is a possibility that malicious code may have been loaded into memory. Therefore, we must establish a clean starting point for measuring subsequent operations in the installation process. The bootstrap phase achieves this through a CPU-supported technique called *late launch* that takes a piece of code, records it in the RTM, and effectively reboots the system before executing the code in a region of protected memory. This memory protection prevents attacks from potentially malicious resident code loaded before the installer and from external devices that have direct access to memory. Once the installer kernel is launched, it measures the installer's ramdisk, unpacks it into memory, and begins the next phase. We color this phase blue to indicate the installer code and data are measured before being executed.

Download Phase After the installer has been initialized, it enters the download phase. The goal is to retrieve and measure the disk image before installing it. First, the local system's basic networking and partition table are prepared to enable a disk image to be retrieved and written to a clean disk. The disk image is also measured into the RTM so that a verifier can later identify the trustworthiness of the downloaded disk image. This helps detect attacks on the disk image while in transit and from compromised or rogue image servers.

Configure Phase In the configure phase, the downloaded disk image is specialized to the target system. This includes setting up networking, filesystem tables, devices, security policies, SSH host keys, etc. The installer also generates signing keys used by the RTM for generating attestations and the ROTI proof. We describe this in more detail in Section 4.2. Next, the system's startup scripts are modified to measure the root filesystem at boot time. This filesystem manifest is included in attestations so the verifier can inspect how the filesystem has been modified.

Proof Phase In the final, proof phase, the installer generates a *ROTI proof* that ties the final installed filesystem to the installer and disk image for verification at runtime that the client is derived from such inputs. The ROTI proof is a signed tuple $R = \text{Sign}(F, D, I)_{K^-}$, where K^- is a private key that identifies the physical machine and is endorsed by its RTM. This tuple acts as a proof showing that a machine possessing K^- was specifically configured by I using disk image D to produce filesystem F. A verifier can inspect the ROTI proof to determine if it believes the combination of I and D results in a trustworthy installation (i.e., F). Upon completion of the ROTI proof, the system reboots into its newly installed filesystem. In the next subsection, we describe how the ROTI proof is used by a verifier to check the integrity of the client at runtime.

3.4 Verification

Once the installation procedure is complete, the system is ready for verification. A successful validation proves to a verifier the particular installer and disk image that was used to install the current filesystem. The verifier can then make a trust decision whether the combination of installer and disk image came from a source they trust.

Figure 3 illustrates the validation procedure. During boot, the initial ramdisk (initrd) is loaded and measures the root filesystem to generate a manifest of hashes for each file. Next, the system starts a network facing *attestation daemon* to handle attestation requests. When a remote verifier wishes to inspect the proving system, it sends a nonce to the attestation daemon that builds an attestation and returns it to the verifier. The attestation is a signed statement $A = \text{Sign}(R, F', N)_{K^-}$, where R is the ROTI proof, F' is the current filesystem manifest, N is the nonce, and K^- is the private key of the system endorsed by its RTM. The verifier checks that the signatures on both the A and R are from keys endorsed by the RTM of the proving system. Verifying the identities of keys is outside the scope of this paper, but we assume a PKI is maintained by the administrator deploying the systems. Once it has established the identities of the signing keys, the verifier assesses whether it trusts the installer and disk image in R to produce a trustworthy installation. If it does, it then compares F' and F in R to see how they differ. If no security critical files have changed, the verifier can assume the current system booted into a filesystem that was produced by an installer and disk image the verifier trusts.

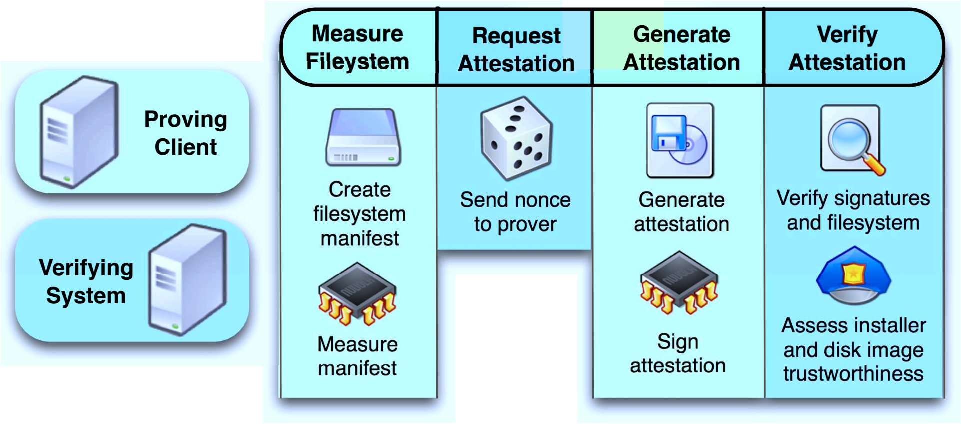

Figure 3: Timeline of the validation process. The filesystem on the proving client is measured to generate an attestation of the installed and current filesystems, which can be checked by the verifying system.

4 Implementing the netROTI

We now describe our proof of concept netROTI implementation. We first introduce some background on the trusted computing components we use and then we give an overview of the installation process and then detail the key components.

4.1 Trusted Computing Primitives

Two key mechanisms we employ in the netROTI are TPM attestations and the new dynamic root of trust. The TPM is a secure coprocessor attached to the motherboard that provides several features. These includes NVRAM for storing cryptographic secrets and a set of *platform configuration registers* (PCRs) that stores arbitrary data measurements. The TPM also contains a public key pair called the Endorsement Key (EK) that uniquely identifies the TPM device and associated client. Using the EK, the TPM can generate and certify other keys. The main runtime mechanisms of the TPM are: 1) *TPM Extend*, which adds a measurement to a PCR by hashing the measurement value with the current PCR value to form a hash chain and 2) *TPM Quote*, where the TPM produces a signed statement over a specified set of PCRs and a nonce from a second party. This quote allows a remote verifier to examine the state of the system (represented by the measurements in the PCRs) at the time the nonce is generated. The TPM uses a signing key called the Attestation Identity Key (AIK) derived from the EK to sign the quote, which effectively identifies the quote as coming from the platform containing the TPM. A quote combined with the list of measurements associated with the PCRs' current state is called an *attestation*. To verify an attestation, a remote verifier validates that the measurements correspond to acceptable operation (e.g., a trusted installer load) and that the series of measurements results in the provided PCR values.

Producing meaningful attestations requires establishing a *root of trust* guaranteeing measurements are taken by a trustworthy entity on the system. A computer boots into what is called a *static root of trust for measurement* (SRTM) because the normal boot process is largely fixed. Normally, the BIOS takes a measurement of its firmware and option ROM chips before passing control to the bootloader, which is expected to continue the measurement chain by measuring itself, the kernel and so on. This chain rooted in the normal boot procedure gives a verifier a view of how the system booted. However, there are several known attacks on the SRTM, such as TPM reset attacks that enable an attacker to reset a PCR and insert false measurements.

To address these issues, new secure virtualization architectures like AMD's Secure Virtual Machine

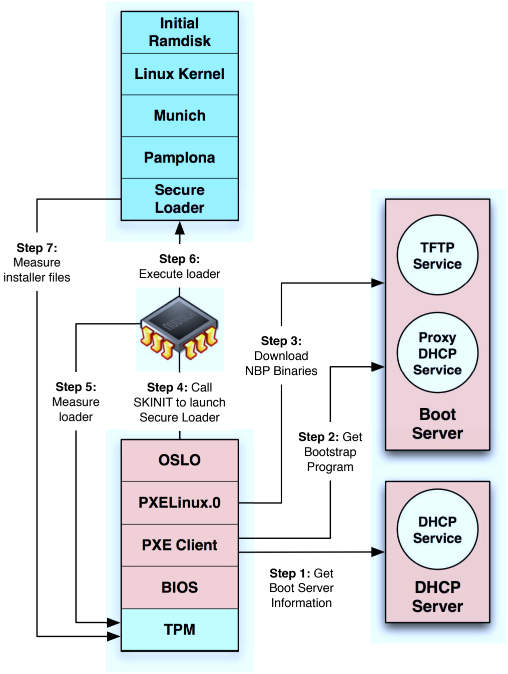

Figure 4: After the client follows the PXE protocol and contacts the Boot Server, it downloads the `oslo` bootloader's three binaries, the installer kernel, and initial ramdisk. `oslo` performs the AMD DRTM operation, SKINIT, which causes the CPU to clear the DRTM registers, extend PCR 17 with a hash of the Secure Loader Block (SLB), and begins executing the SLB, which measures the rest of the downloaded files and extends it to PCR 19. `pamplona` undoes the DRTM memory protection and `munich` launches the kernel to being installation.

(SVM) [7] and Intel's Trusted Execution Technology enable a machine to form a *dynamic root of trust for measurement* (DRTM) by effectively rebooting the system and executing a piece of code called a secure loader in a memory protected region safe from code loaded before the loader was executed. The CPU first sets a special group of DRTM PCRs to a specific value that only the CPU can set and then measures the loader before it is started. This prevents malicious code from imitating the DRTM process by inserting secure loader's code measurement in to the PCR. The DRTM is useful for bootstrapping security critical code like a VMM kernel when the security of the system is difficult to assess at boot time.

4.2 Installation

We created our netROTI as a series of scripts that automates the installation process packed into an 11 MB ext2 ramdisk downloaded along with a modified Linux 2.6.18 kernel and the OSLO [11] bootloader. OSLO is a specialized bootloader that implements the DRTM functionality in AMD processors that we use to launch the installer kernel in a secure environment. Before installation begins, the administrator configures the BIOS to boot from the PXE firmware. The TPM must also be cleared of any previous administrative passwords and keys, so the installer can create its own. This corresponds to the preinstall phase in our design. In the gather phase, the begins the PXE protocol to obtain the location of the Boot Server. The client then downloads the `pxelinux.0` NBP via TFTP, which automatically retrieves the OSLO bootloader, Linux kernel, and installer ramdisk.

The system then enters the bootstrap phase illustrated by Figure 4. First, the NBP constructs a multiboot header indicating the address where the the installer files are located in memory and executes the OSLO bootloader. OSLO consists of three ELF binaries that perform separate stages of the DRTM process. The first binary prepares the system for the DRTM process by shutting down all but the primary CPU core and loads the second stage binary into the secure loader block (SLB). The AMD DRTM instruction, SKINIT, is invoked with the entry point address of the SLB as its only argument. The CPU then sets the DRTM PCRs in the TPM to -1, sets the device exclusion vector (DEV) to enable memory protection for the SLB, and sends a measurement of the SLB to the TPM. Finally, the CPUs jumps to the entry point in the SLB that measures the installer Linux kernel, ramdisk, and boot parameters in the multiboot header, restarts the other CPU cores, and disables the DEV protection. Finally, it launches the third stage binary that imitates a normal GRUB bootloader and launches the Linux kernel.

Once the installer has been bootstrapped, the ramdisk is unpacked into memory and the installation script is executed. This sets up basic support for devices like console and networking and starts the download phase by running a `partimage` client. This contacts a preconfigured `partimaged` server, verifies its SSL certificate against the CA certificate in the ramdisk, establishes an SSL connection, and downloads the disk image. The image is measured and then written to the hard disk.

In the configure phase, the installer scripts configures machine specific files including updating the UDEV rules for new hardware, networking configurations, firewall rules, fstab entries, creating a swap partition, regenerating SSH host keys, and so on. The TPM is then setup with new administrative credentials and a fresh AIK is generated. The TPM also endorses the AIK by creating a certificate that signs the AIK's public key with the TPM's EK. The installer also installs a simple network-facing python service we wrote that acts as the attestation daemon, which services requests for attestations. The initial ramdisk on disk is then modified to generate a manifest of the filesystem every time it boots. This manifest contains a hash of every file and is used to create attestations.

In the proof phase, the installer measures the files on disk that have changed or been added since the disk image was written. A list of SHA1 hashes for each file is stored on the disk. The final step generates the ROTI proof by producing a TPM quote with PCRs containing every measurement taken during the installation process. This quote is signed with the newly created AIK from the configure phase. We use a hash of the system's hostname as the nonce since we are not concerned with the freshness of the quote, as we only care that the ROTI proof correctly identifies the installer and disk image for this system. We tar and gzip the quote with the file manifest and list of measurements taken during installation to create the final ROTI proof file.

4.3 Verification

We now describe the verification protocol between a system installed by the netROTI (the proving system) and a remote verifier. Before the protocol begins, the proving system boots into its initial ramdisk. It then executes the measurement script inserted during installation. This script generates a manifest of the entire filesystem with corresponding hashes for each file. The system resumes the boot process and starts the attestation daemon. When a verifier sends a nonce to the daemon, the daemon takes a SHA1 hash of the nonce and requests a TPM quote signed by the TPM's AIK. The quote's PCRs contains a hash of the filesystem manifest taken at boot time and the nonce. The quote is then returned to the verifier with the ROTI proof file (corresponding to R in the attestation A from Section 3.4), the filesystem manifests taken at boot (F') and during installation (F), and the AIK's certificate (the verifier has the nonce N already).

Upon receipt of the attestation, the verifier first validates the signatures of the quote and ROTI proof. It then checks that the AIK's certificate is signed by the expected TPM's EK. Next, the verifier assesses the trustworthiness of the installer and disk image by extracting them from the ROTI proof and matching them against a list of acceptable measurements. If these are found to be trustworthy, the filesystem manifests are compared to see if any files have changed since installation. If no security critical files are modified, then verifier accepts the proving system as having booted into a filesystem installed by a trusted installer and disk

Type	Operation	Time (seconds)
Install	Download and Write Disk Image	64.000
Install	Configuration	18.644
	Sub-total	82.644
netROTI	netROTI Configuration	6.740
netROTI	Measure Image	1.900
netROTI	Generate TPM Quote	0.890
netROTI	Measure Modified Files	0.390
	Sub-total	9.920
Optional	TPM Setup	45.400
Optional	Generate AIK	11.220
	Sub-total	56.620
	Total Install	**149.184**

Table 1: Breakdown of the installation time averaged over ten installations of a Eucalyptus cloud node.

image.

5 Evaluation

We constructed a proof of concept netROTI installer to assess the overall impact it has on network installation and how it address attacks on the installation process.

5.1 Performance

To evaluate the overhead our netROTI installer imposes on the installation procedure, we performed ten installations of a Eucalyptus cloud node's disk image across ten systems. We created an image to be installed by manually configuring an Ubuntu server cloud in our Eucalyptus on a Dell PowerEdge M605 blade with 8-core 2.3GHz Opteron CPUs and 16GB of RAM on a quiescent gigabit network. We then created a 387 MB gzipped disk image of the 1.3 GB filesystem. Table 1 shows the times for each operation performed during installation. Normal installation took 82.644 seconds or 55.34% of the overall time. The disk image related operations (e.g., downloading, writing, and measuring the image) are a function of the disk image's size, which can be improved through more efficient compression algorithms. In particular, we found our hardware could perform SHA1 hashes at 132 MB/s, which resulted in a 1.9 second disk image measurement time.

TPM related operations are inherently slow due to the speed of the TPM's bus (33 MHz) and its low power design. While netROTI specific operations added additional time to the install, two operations, namely generating a new AIK and TPM setup, account for 37.95% of that overhead. We note that the function of these steps are to create keys that could be reused across multiple installations as long as the encrypted public portions of the AIK and SRK (created in the TPM setup operation) are retained during reinstallation. Thus, an administrator could copy those encrypted files and redistribute them in the installer or have the installer copy them from the local disk before overwriting it. Ultimately, we find the overhead due to the netROTI to be a fixed cost of about 8 seconds plus about a 3% overhead for measuring the image when the optional TPM setup and AIK creation steps are reused from previous installations.

Attack Type	**OSLO**	**Tripwire**	**Bitlocker**	**netROTI**
Rootkits before install	Yes	No	No	Yes
Malicious installer code	Yes	No	No	Yes
Malicious disk image	No	No	No	Yes
Modified data after install	No	Yes	Partial	Yes
Runtime attacks	No	No	No	No

Table 2: A comparison of several mechanisms' ability to detect or prevent several classes of attack on an installation.

5.2 Security Evaluation

Table 2 lists a comparison of several security mechanisms and their ability to handle a range of attacks on the network installation process. In addition to our netROTI design, we consider the OSLO bootloader alone, the filesystem auditing tool Tripwire [12], and the Windows Bitlocker file encryption scheme [1]. OSLO uses the DRTM process to both measure malicious installer code and defeat rootkits the system might boot into before installation, but it is unable to address attacks beyond that. Tripwire is an auditing tool that creates a digitally signed log of the installed filesystem that administrators can query to detect changes. This prevents attacks that change the disk contents after installation, but cannot guarantee anything about the filesystem during installation. Bitlocker encrypts the filesystem and optionally uses the TPM to verify that the early boot phase has not been modified before decrypting the disk. While this prevents offline attacks, modifications to the disk after decryption are not prevented. The netROTI uses OSLO for protection against rootkits and to record malicious installers. It also measures disk images before installation and uses the ROTI proof combined with boot-time filesystem measurements to detect changes. However, none of these approaches directly address attacks on the installed system at runtime.

The key advantage of the netROTI over these other approaches is its ability to provide an attestation of not only the filesystem, but the installation environment that produced it. While the other solutions in our comparison prevent attacks at various stages of the installation process, none of them can speak for the trustworthiness of the installer that produced them. By using secure hardware to measure before using each critical component during installation, a verifiable proof can be created of the filesystem's origin.

6 Conclusion

In this article, we introduced the netROTI, a method for performing network installation so that the resulting filesystem can be traced back to the exact installer and disk image that produced it. The netROTI leverages the protection of trusted computing features in modern CPUs to bootstrap a dynamic root of trust in an installer downloaded over the network and record all steps of the installation procedure to produce a ROTI proof. Using this proof, a verifier can inspect whether the guarantees of trusted distribution have been achieved. Our evaluation demonstrated the netROTI protects against a variety of attacks on the installation process and introduces only minimal overhead when optimizations are taken into account.

7 Acknowledgements

This work was supported by NSF grants CNS-0931914 and CNS-0627551. We would also like to thanks Luke St. Clair for his efforts in designing Root of Trust Installer, Erik Steigler and Mark Tamminga for their insights into network installation, and Bernhard Kauer for his design of and assistance using OSLO.

References

[1] BitLocker Drive Encryption: Technical Overview. http://technet.microsoft.com/en-us/windowsvist

[2] Norton Ghost. http://www.symantec.com/norton/ghost.

[3] Preboot Execution Environment (PXE) Specification. http://www.intel.com/design/archives/wfm/dow

[4] The Jedi Packet Trick takes over the Deathstar. http://www.alchemistowl.org/arrigo/Papers/Arrigo

[5] Rocks Clusters. http://www.rocksclusters.org/wordpress/, August 2010.

[6] Acronis. True Image. http://www.acronis.com/homecomputing/products/trueimage/index.htm

[7] Processor-based virtualization, amd64 style. http://developer.amd.com/documentation/articles/pa

[8] E. Brickell, J. Camenisch, and L. Chen. Direct anonymous attestation. In *11th Conference on Computer and Communications Security*, pages 132–145, New York, NY, USA, 2004. ACM.

[9] L. S. Clair, J. Schiffman, T. Jaeger, and P. McDaniel. Establishing and sustaining system integrity via root of trust installation. In *Proceedings of the 2007 Annual Computer Security Applications Conference*, pages 19–29, Dec. 2007.

[10] R. P. Gallagher. A Guide to Understanding Trusted Distribution in Trusted Systems. http://www.fas.org/irp/nsa/rainbow/tg008.htm, 1988.

[11] B. Kauer. Oslo: improving the security of trusted computing. In *16th USENIX Security Symposium*, pages 1–9, Berkeley, CA, USA, 2007. USENIX Association.

[12] G. H. Kim and E. H. Spafford. The design and implementation of tripwire: a file system integrity checker. In *Conference on Computer and Communications Security*, pages 18–29, New York, NY, USA, 1994. ACM.

[13] Microsoft. Windows Deployment Services. http://msdn.microsoft.com/en-us/library/aa967394.a

[14] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *9th International Symposium on Cluster Computing and the Grid*, pages 124–131, Washington, DC, USA, 2009. IEEE Computer Society.

[15] B. Parno, J. M. McCune, and A. Perrig. Bootstrapping trust in commodity computers. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2010.