Trusted Computing on Athena

Benjamin Tidor, Eric Mannes, Gaurav Singh

Motivation

In 2005, a number of Athena cluster machines were modified by a student to steal the passwords of anyone who logged in on those machines.[1] Since Athena machines grant root access to all users, it is not difficult to modify the kernel and install a keylogger, or even display a look-alike login screen to capture passwords. Our goal is to prevent these types of attacks by allowing cluster machines to attest to the user that are unmodified before the user logs in.

Design and Implementation

When a user wants to log in to a cluster machine, they use their smartphone (the user agent) to transmit a request to verify that the machine is in an unmodified state to a trusted verifier. The trusted verifier then proxies this request to the cluster machine. The cluster machine replies with a fingerprint of its internal state, which the trusted verifier will check against a list of known good states. If verification succeeds, the trusted verifier will inform the user agent that the workstation is safe to log in to.

User Agent

Before logging in, the user scans a QR code displayed on the cluster workstation. The QR code directs the user agent to a URL of the trusted verifier, e.g.

https://gauravjs.scripts.mit.edu/integrity-controller/v0/login/tpm-project-machine.mit.edu

After performing verification, the trusted verifier replies to the user agent, which displays (a) the hostname of the workstation that was verified and (b) whether or not verification was successful. The user should check that the hostname listed matches the host at which they were attempting to log in, and should look for the green check mark indicating that verification was successful.

In practice, the QR code sticker can be trivially replaced with a fake. The best way for the user to verify that the machine in front of them is the same machine that was verified is to check the hostname displayed onscreen. Unfortunately, this hostname can also be faked with little effort. One potential method for tying physical presence into the online verification process is to display the QR code

[1] http://tech.mit.edu/V125/PDF/V125-N20.pdf

on the monitor and to include a value that changes every few seconds. If this value is included in the dialogue between the trusted verifier and the workstation, the workstation can ensure that the value the user sees matches the value displayed onscreen.

The most sure mechanism of confirming the identity of the workstation is for the user agent to display a map of the cluster and ask the user to indicate their location on the map. As cluster workstations are typically locked in place, this provides relatively high assurance that the correct machine is being verified. We leave the task of mapping the Athena clusters on campus and extending the user agent to display these maps as an exercise to the reader.

Trusted Verifier

The trusted verifier is a central server to be managed by the system administrator (in our case, MIT IS&T). It maintains a listing of each machine's AIK public key and contains the business logic for determining which combinations of PCR hashes represent a valid workstation configuration. The system administrator is responsible for instrumenting known-good machines to determine the set of allowable hashes for each hardware configuration and kernel in use.

The user agent accesses the trusted verifier over HTTP to initiate a request, passing the hostname of the target workstation as a parameter. The trusted verifier then generates a nonce (to prevent replay and pre-computation attacks) and forwards the request to the workstation, which replies with an attestation. The trusted verifier checks the signature on the attestation using the workstation's AIK public key. It then decides whether or not the reported PCRs represent a trusted system configuration and transmits its decision to the client in the HTTP response. Note that our implementation is somewhat incomplete: the task still remains to write a function to compute the composite hash from the PCR values. This is done by the TPM hardware, but the trusted verifier must perform the computation independently, as the AIK signature is made only over the composite hash. Thus, our published implementation loses the flexibility of evaluating each PCR individually, which is needed in a practical deployment. Additionally, we were unable to get tpm-quotetools's tpm_verifyquote, which is poorly documented, to actually verify the signatures. This requires further work.

Workstation

The workstation (public cluster machine) must be built with a Trusted Platform Module (version 1.2). Throughout the boot process, the TPM interacts with cooperating software to measure components of the system and record unforgeable fingerprints in its Platform Control Registers (PCRs). We use a variety of software to create a chain of trust that stretches from the BIOS firmware to the entire contents of the hard disk.

- On boot, a core, immutable piece of code creates a hash of the BIOS configuration and firmware and places a it in PCR 0. It then transfers control to the BIOS.
- The BIOS measures the bootloader and aspects of the hardware configuration, placing its hashes in PCRs 1-3. It then transfers control to the bootloader.
- We installed the TrustedGRUB bootloader, which ultimately extends the chain of trust to the kernel (specifically, the vmlinuz and initrd files). It populates other PCRs with hashes representing the contents of these files, the contents of additional parts of the bootloader, and the commands that were executed during boot, which includes options passed to the kernel.
- The Linux Integrity Management Architecture (IMA) subsystem is a component of the kernel that can be used to measure kernel modules and configuration files as they are loaded. It can also be used to measure core executables such as `ls` that the user expects to be available and safe. We do not include instructions for using IMA, and leave is configuration as an exercise for the reader.

After booting, the workstation initializes the login screen and runs the attestation server, which listens on port 6858 for incoming requests-to-verify issued by the trusted verifier. When such a request arrives, the attestation server reads the values of the PCRs from the TPM and requests a quote -- a signature attesting that the reported PCRs are true and correct. The TPM creates this quote by calculating a composite hash over all PCRs, combining this hash with a verifier-provided challenge nonce, and signing the result with the AIK private key. The AIK private key is securely stored within the TPM hardware and is only used for this purpose: to sign the current PCRs. Using the AIK public key, the trusted verifier can check the signature and thus be confident that the workstation truly is in the reported configuration.

As a final link in the chain of trust, we must ensure that the system has not changed from the configuration that was recorded during the boot process. To do this, we assume that the system will be in its original, trusted configuration so long as no user has yet logged in, and require that the system be restarted after each use. We use PCR 15 to record the login state of the machine: PCR 15 is initialized to zero on boot, and we use a utility invoked via PAM module to fill it with garbage data when any user logs in. The trusted verifier can then require that PCR 15 be zero in order to authorize a login. This design protects against users persisting a keylogger in the kernel or displaying a lookalike login screen. As an optimization that does not require a reboot after every use, we can rely on the attestation server to exclusively bind port 6858 and only send attestations to the trusted verifier when no other user is logged in. We can then assume that the system is safe so long as no user has yet exercised root privileges, and we can delay corrupting PCR 15 to the point where `su` or `sudo` is invoked.

Code

Our code is available at https://github.mit.edu/tpm-project/.

Future Directions

To mitigate the impact of verifying the wrong workstation, we suggest tying authentication into the verification process. One way of doing this would be to have the user agent obtain the TGT and pass it to the workstation after verification succeeds (presumably over an encrypted channel negotiated through a Diffie-Hellman exchange split between the QR code and the network). This ensures that the TGT can only be transmitted to a trusted machine, prevents the user's password from being transmitted over the keyboard-to-chassis channel (where it is relatively easy to install a hardware keylogger), and restricts the window of exploitation to the lifetime of the ticket. Alternatively, our system (which increases confidence in the integrity of the machine) complements krb-agent[2] (which mitigates damage from compromised machines), and the two could potentially be deployed in tandem.

Another area for improvement is making the system more robust against DoS attacks. The servers running on the cluster machines could only accept requests signed by the trusted verifier. This way, only for legitimate requests will the cluster machine spend the time to get the TPM PCRs and have the TPM sign them.

[2] http://css.csail.mit.edu/6.858/2013/projects/mgersh-gurtej-dlaw-jmoldow.pdf