Using TPMs to cryptographically verify devices at scale

Matthew Garrett
Tom D’Netto

Open Source Summit, North America 2019
About us

Matthew Garrett
Low-level security
@mjg59

Tom D’Netto
Security engineer & Go enthusiast
@twitchyliquid64
The trust problem and remote access

Trusted Platform Modules (TPMs)

Remote attestation

Deploying remote attestation with open source

Questions
The Fundamental Problem

How do we trust that a device is secure when all we see are bytes on the wire?

How do we know that traffic isn’t being generated by an attacker after compromising a device?
Existing solutions are inadequate

Without cryptographic assurances as to the state of the device and its credentials, circumventing controls on a compromised device remains straightforward:

- **Endpoint detection agents** can be disabled
- **Remote logging** traffic can be faked or replayed
- **On-system identity keys** can be stolen and used from an attacker-controlled system

![Diagram showing compromised traffic being indistinguishable from legitimate activity at a Policy Enforcement Point before reaching a Networked Resource]
The tech that makes this possible

Trusted Platform Modules
TPM Overview

If we can’t trust an operating system, what can we trust?

Trusted Platform Modules allow us to segregate that trust into a separate hardware component

- Store a trusted record of what the system booted
- Securely store keys and/or secrets
TPM Measurements

- Each component in the boot chain “measures” the next
- Each measurement is pushed to the TPM
- The TPM can provide a digest of the measurements
- But how do we establish trust in the TPM?
TPM Identity: Endorsement Keys and Certs

- Each TPM has one or more unique Endorsement Keys
- Each EK has an associated certificate
- The certificate can be chained back to the TPM manufacturer

**Endorsement Key**
- RSA or ECC key pair

*The Endorsement Key uniquely identifies the TPM*

**Endorsement Certificate**
- <Endorsement key info>
- Issuer information: ...
- Signed by: ...
- ...

*The Endorsement Certificate relates the TPM to its manufacturer*
Attestation Keys

- AKs allow attestation to be separated from the EK
- Through a complicated dance, the TPM with a specific EK can prove it generated a AK
- The AK is used to sign the quote, allowing the quote to be tied to the TPM

TPM proves AK is co-resident with EK
AK signs platform state (PCRs)
Attestation (quote) transmitted
Remote Attestation

Solving trust in remote access

Google Cloud
Components of a remote attestation deployment

On-system agent - Runs on each device
  - Collects measurements, attests them using the TPM, and reports them back

Verification service - Runs on your infrastructure
  - Verifies attestations, and extracts data points from the measurements for decision making

Access proxy - Runs on your infrastructure
  - Proxies all remote access to resources
  - Makes allow/deny access decisions based on data points extracted by the verification service
An end-to-end example

Device Boots
- The processor records the hash of platform firmware into the event log
- Platform firmware records further measurements
- Each boot component records the hash of the next stage before loading it
- OS boot components record properties about themselves and their components

At the end of boot
- A log of all events exists in memory, which should match the PCR hashes stored in the TPM.

The device attests its state to the verification service
- The agent transmits the event log, along with a signature from the TPM over the PCR hashes (and a nonce)
- The verification service replays the log to compute final PCR values, checking them against the signed PCRs
- Everything checked out, so the log is processed to extract data points for decisions making

Access control decisions are made using extracted data points:
- For instance, we can restrict production access to a device which booted a specific OS with secure boot, and OS events demonstrate that our security monitoring agents started successfully
Making access decisions

There are a wealth of data points we can use, for instance:

<table>
<thead>
<tr>
<th>Data point</th>
<th>Expected value</th>
<th>Known state of device</th>
</tr>
</thead>
<tbody>
<tr>
<td>Secure Boot enabled</td>
<td>true</td>
<td>Early-boot is harder to compromise</td>
</tr>
<tr>
<td>ELAM drivers list (windows-only)</td>
<td>Hash/path of endpoint security agent (ie: <code>\Windows\system32\drivers\CrowdStrike\CSBoot.sys</code>)</td>
<td>Security monitoring agent was loaded</td>
</tr>
<tr>
<td>Loader image hash</td>
<td>Hash of bootloader/config</td>
<td>Device booted an expected OS (ie: the one containing our security controls)</td>
</tr>
</tbody>
</table>

This can be non-trivial - for hash-based data points, reliably determining the correct set of hash values is hard.
Open source tooling
Go-attestation

Available at:
https://github.com/google/go-attestation

Implements:
- All TPM primitives for an attestation client on Windows/Linux, TPM 1.2 or TPM 2.0.
- Server-side verification primitives & event log parsing
- Demo is WIP in PR #69

package attest

import "github.com/google/go-attestation/attest"

Package attest abstracts TPM attestation operations.

Index

Variables
- func ParseEKCertificate(ekCert []byte) (*x509.Certificate, error)
  type AIK
    - func (k *AIK) ActivateCredential(tpm *TPM, in EncryptedCredential) (secret []byte, err error)
    - func (k *AIK) AttestationParameters() AttestationParameters
    - func (k *AIK) Close(t *TPM) error
    - func (k *AIK) Marshal() ([]byte, error)
    - func (k *AIK) Quote(tpm *TPM, nonce []byte, alg HashAlg) (*Quote, error)
  type AIKPublic
    - func ParseAIKPublic(version TPMVersion, public []byte) (*AIKPublic, error)
  type ActivationParameters
    - func (p *ActivationParameters) Generate() (secret []byte, err error)
Questions?