Beyond Anti Evil Maid

Protecting hardware from early boot attacks

Matthew Garrett

Security Developer at CoreOS

<mjg59@coreos.com>

@mjg59


CoreOS

How can you trust your computer?

When do you trust your computer?

Your computer must be trusted before you enter any secrets

Can the software asking for your password be trusted?

Can the kernel running the software asking for your password be trusted?

Can the bootloader that loaded the kernel running the software asking for your password be trusted?

Can the firmware that launched the bootloader that loaded the kernel running the software asking for your password be trusted?

Can the CPU executing the firmware that launched the bootloader that loaded the kernel running the software asking for your password be trusted?

Trusted Computing

Trusted Platform Module (TPM)

Measurement

SHA1 of next boot component

Extend Platform Configuration Register

$$PCR_N = SHA1(PCR_O||hash)$$

Three ways to get a specific PCR value:

Break SHA1

Trigger execution of arbitrary code through unmeasured data

Perform exactly the same sequence of writes to the PCR

But how do we read these values?

An untrustworthy kernel could just lie…

Remote Attestation

Cryptographic communication between TPM and remote system

Local OS can’t interfere with remote attestation

Need to trust remote machine

Need network access

How does the remote machine communicate back to you?

We’re asserting that the TPM is trustworthy

TPMs can control data release based on PCR state

TPMs can “seal” data to PCR state

Data is encrypted with a TPM-specific key

Encrypted blob contains PCR values

TPM decrypts blob. If PCRs don’t match, TPM gives error.

Disk decryption key?

(Someone can still steal your laptop)

Disk encryption key + passphrase?


Password (/dev/sda3):********_

```
Call Trace:
 [<c041b7f2>] iounmap+0x9e/0xc8
 [<c053480d>] agp_generic_free_gatt_table+0x2e/0x9e
 [<c0533991>] agp_add_bridge+0x1a8/0x26f
 [<c05439eb>] __driver_attach+0x0/0x6b
 [<c04e6bf4>] pci_device_probe+0x36/0x57
 [<c0543945>] driver_probe_device+0x42/0x8b
 [<c0543a2f>] __driver_attach+0x44/0x6b
 [<c054344a>] bus_for_each_dev+0x37/0x59
 [<c05438af>] driver_attach+0x11/0x13
 [<c05439eb>] __driver_attach+0x0/0x6b
 [<c0543152>] bus_add_driver+0x64/0xfd
 [<c04e6d22>] __pci_register_driver+0x47/0x63
 [<c040044d>] init+0x17d/0x2f7
 [<c0403dee>] ret_from_fork+0x6/0x1c
 [<c04002d0>] init+0x0/0x2f7
 [<c04002d0>] init+0x0/0x2f7
 [<c0404c3b>] kernel_thread_helper+0x7/0x10
 =======================
Code: 78 29 8b 44 24 04 29 d0 8b 54 24 10 c1 f8 05 c1 e0 0c 09 f8 89 02 8b 43 0c
 85 c0 75 08 0f 0b 9c 00 77 c8 61 c0 48 89 43 0c eb 08 <0f> 0b 9f 00 77 c8 61 c0
 8b 03 f6 c4 04 0f 85 a5 00 00 00 a1 0c
EIP: [<c041bd49>] change_page_attr+0x19a/0x275 SS:ESP 0068:c14f7ec0
 <0>Kernel panic - not syncing: Fatal exception
 _
```

User has no way of knowing whether system is trustworthy

Anti Evil Maid

Joanna Rutkowska

Encrypt a phrase known only to the user

Print during boot process or store on USB stick

Don’t see the phrase? Don’t enter any secrets

Problems:

If always printed, easy to steal and mimic

If booted from USB, relies on user to remember

(And theft is still possible)

Exposing the secret is suboptimal

How about a dynamic exposure?

Shared secret with dynamic component

Where have we seen this before?

Time-based One-Time Passwords

Easily available

Well understood by users

Seal the TOTP seed to the TPM

Print a QR code, allow user to enrol it

Print 6 digit number on boot

If number on phone matches, safe to enter secrets

Problems:

Secret exists in RAM

IOMMU

(Still disabled on many free OSes)

Rethink

TPM keys can be restricted to PCR state

Ask TPM to sign current time of day

Verify that signature matches

Requires new software, less familiar to users

Generate key outside the TPM, import it into device, encrypt time of day, hash it, extract six digits

But you can then ask the TPM to give up the key

Simplicity is a virtue

TPMs have GPIO pins

Access to GPIO pins can be restricted to PCR state

OS attempts to write to LED at boot

If LED lights up, system is good

Vulnerable to trivial hardware attacks

NFC communication between TPM and device holding secret

Device verifies exchange with TPM, automatically passes secret back

What attacks are we always vulnerable to?

Management-Engine based attacks

Attacking the TPM itself

If hardware isn’t trustworthy, we have no mechanism to build trust

Incremental improvements in security are still improvements in security

Code:
https://github.com/mjg59/shim
https://github.com/mjg59/grub
https://github.com/mjg59/tpmtotp