# TPM Genie

## Attacking the Hardware Root of Trust For Less Than $50

# Introduction

**Jeremy Boone**

@uffeux

- Principal Consultant @ NCC Group
- Focus on hardware and embedded systems security
- Previously: 10 years product security @ BlackBerry & Motorola Mobility

# Agenda

1. Overview of Trusted Platform Module
2. Threat Model & Attack Surface
3. Bug Hunting
4. Interposer Design & Build
5. Demo
6. Conclusions


# The Trusted Platform Module

# What is a TPM?

- **A crypto-processor**
- **Trusted Computing Group (TCG) consortium**
- **Multiple versions: TPM v1.2 and v2.0**
- **Used practically everywhere**
  - Servers, laptops, desktops, IoT devices, …
  - Over 2 billion computers use a TPM (allegedly)
  - The U.S. Army and DoD require that every new PC has one

# Functions of a TPM

- **Command-Response protocol w/ 100+ ordinals**

- **Hardware random number generation**
- **Secure (aka on-chip) generation of cryptographic keys**
  - … plus many other crypto primitives

- **Primary use in “Trusted Computing” applications:**
  - Measured Boot
  - Remote Attestation
  - Sealed Storage

# TPM Functions – Measured Boot

- **Each boot stage is hashed (measured) by previous stage**
  - BIOS, MBR, UEFI, kernel command line …
- **Hashes extended into 160-bit Platform Configuration Registers (PCRs)**
  - PCR is a shielded memory space within TPM chip.
  - `PCR[i].new := HASH( PCR[i].old || new_data )`
- **Chain of trust: Code shouldn't be executed until it's been measured.**
  - PCR set can be audited at any point to inspect measurements.
  - Intel Trusted eXecution Technology (TXT)

# TPM Functions – Remote Attestation

- **Prove to authorized remote party that platform is in a specific state.**

- **The basic idea:**
  - Remote party sends nonce to TPM
  - TPM generates a Quote:
    - `Quote = sign( PCR[n..m] + nonce )`
  - TPM returns Quote to remote party
  - Remote party verifies Quote and decides if it can trust host
  - Next steps are application specific
    - Ex: Hand over some kind of secret, such as DRM key

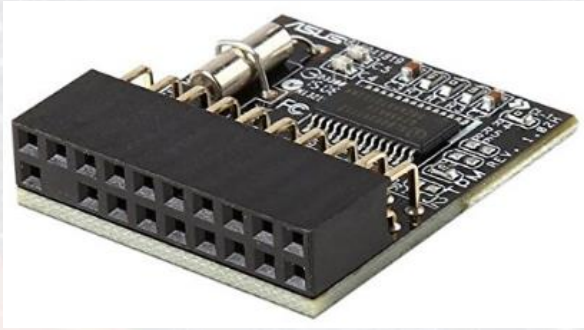
# TPM Functions – Sealed Storage

- **Protects a secret stored in the TPM's non-volatile memory**
  - Ex: Bitlocker or dm-crypt keys
- **Binds the secret to a specific PCR set**
  - `cipher_txt = tpm_seal(   plain_text,  PCRs, [password, locality] )`
  - `plain_txt  = tpm_unseal( cipher_text, PCRs, [password, locality] )`
- **The secret can only be released when the system is in the correct state**


# Attack Surface / Threat Model

# Discrete TPM

- **Manufacturers claim secure type of TPM**
  - Tamper “resistant” against invasive silicon, fault injection and side channel attacks
  - Enter Chris Tarnovsky to prove everyone wrong
- **But… invasive attacks cost $$$**
  - I wanted an attack that works in 5 mins for < $50
- **Threat Model – Those with physical access**
  - Rogue data center employee
  - Supply chain interdiction attacks (NSA ANT-style implant)
  - Evil Maid scenario

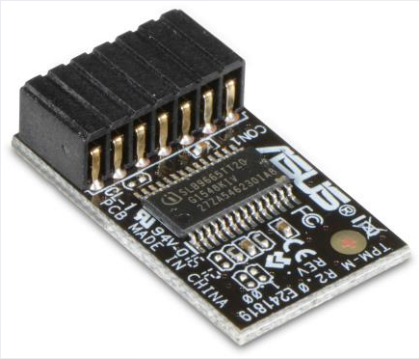


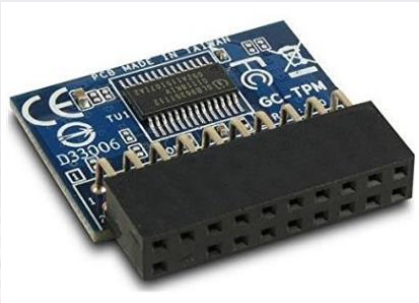
# Discrete TPM Risks

- **Often on a daughter card**
- **Connected to main board via a header**

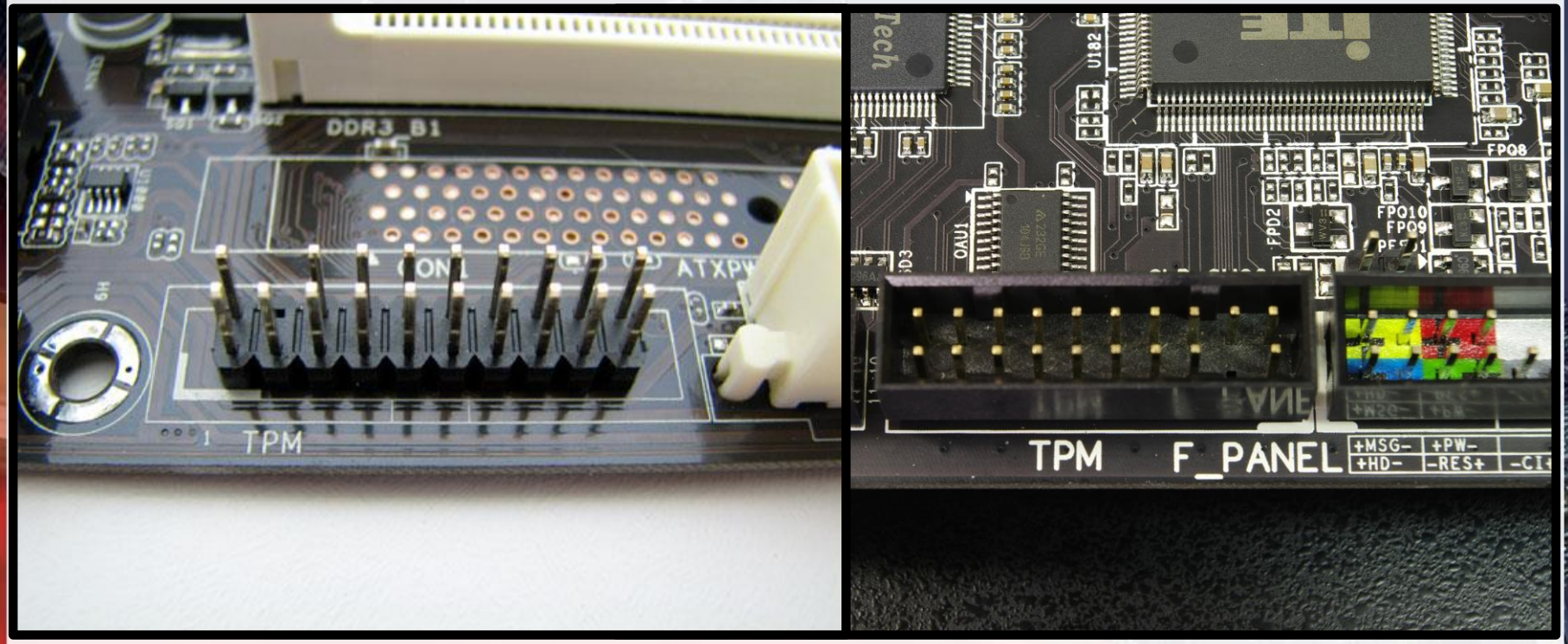
- **Communicate with host via serial bus: I2C, SPI, LPC**

- **Exposes serial bus to tampering**
  - A MITM on the bus can sniff/modify traffic
  - Non-invasive attacks via an “interposer” device
  - No need to cut traces or desolder TPM

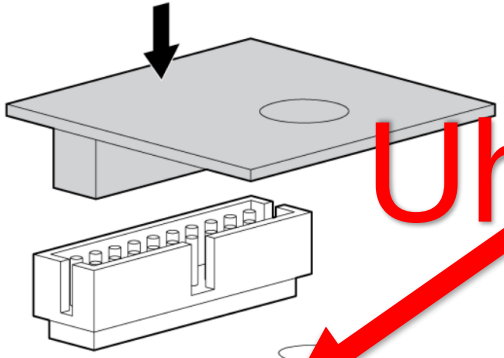




# TPM Headers

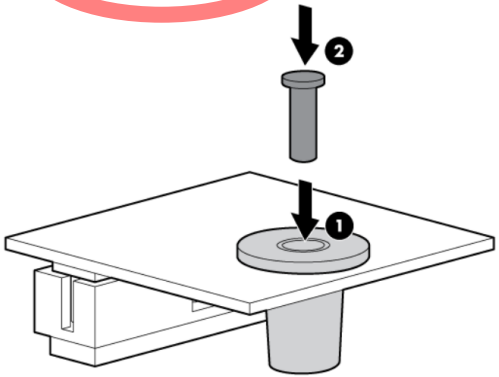


5. Install the TPM board. Press down on the connector to seat the board ("System board components (A-side)" on page 9).

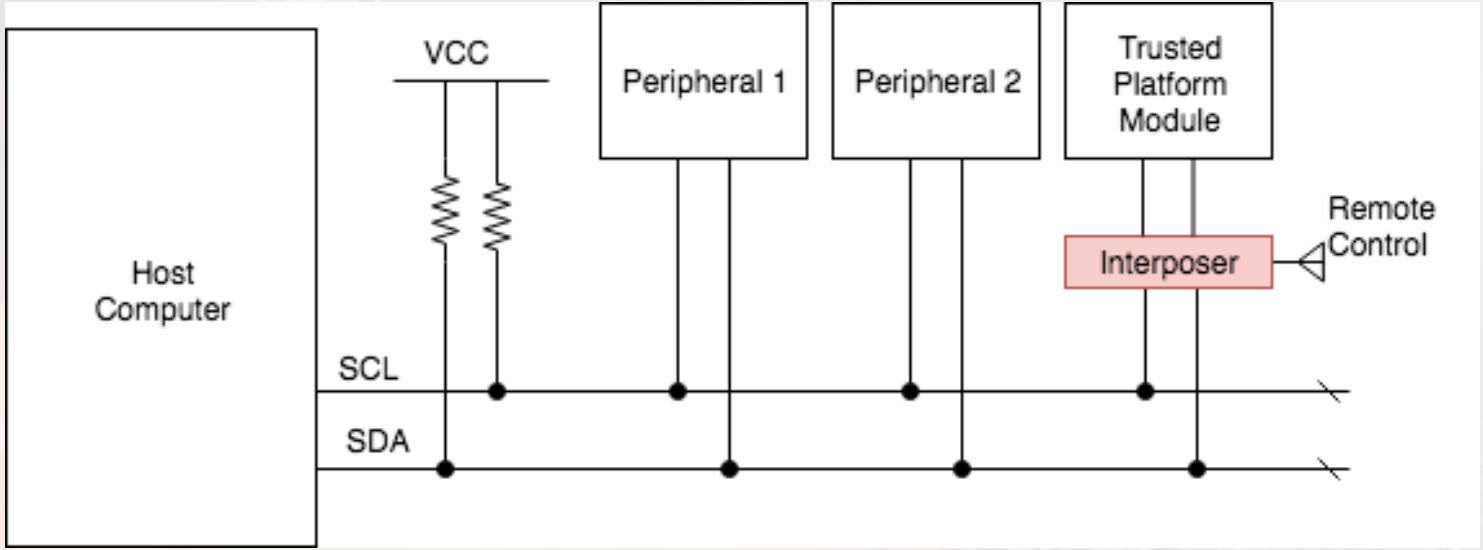


Uhhh…

6. Install the TPM security rivet by pressing the rivet firmly into the system board.




# Serial Bus Interposer

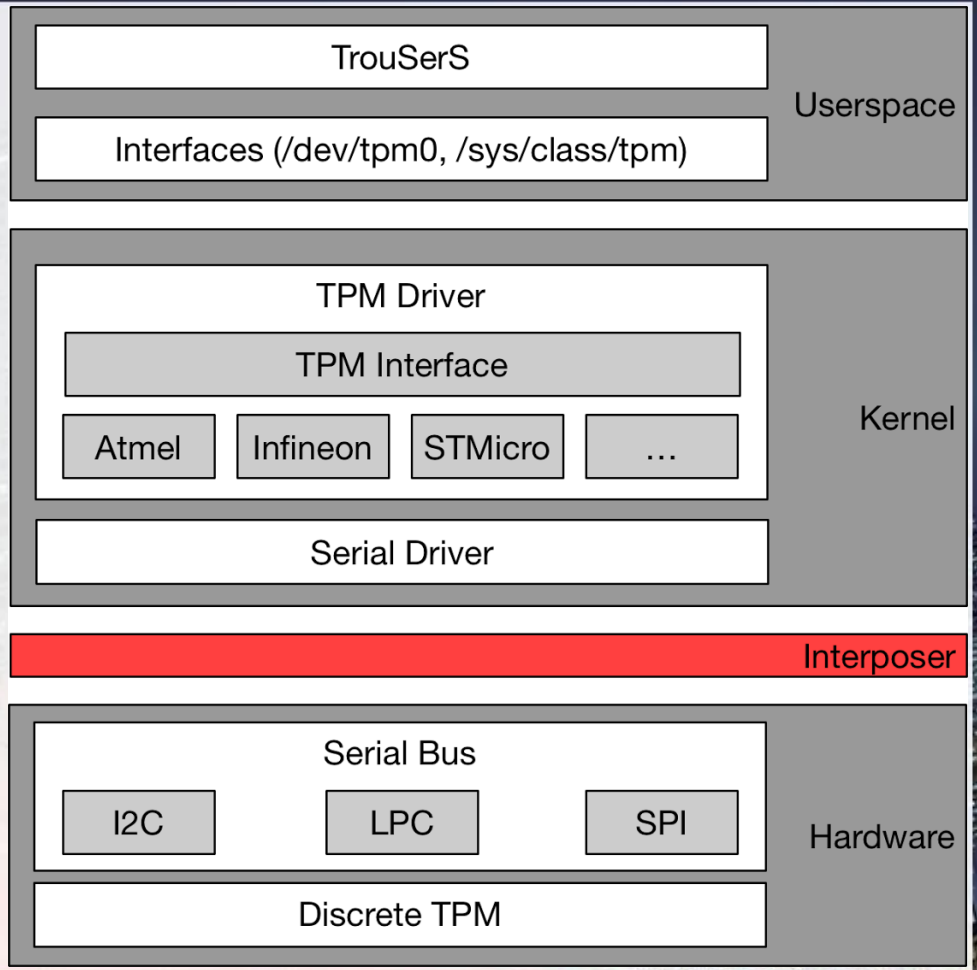



# Hunting For Bugs

# Target Enumeration & Selection

- **Operating Systems:**
  - Linux kernel: Hardware RNG, integrity subsystem
  - Plus other kernels that implement TPM drivers
- **Pre-kernel environments (Bootloader, Legacy BIOS, UEFI):**
  - tboot, coreboot, GRUB, Tianocore EDK2, +more
- **Userspace stuff:**
  - TrouSerS, OpenSSL TPM Engine, +more

# Linux Kernel TPM Driver Architecture



# Kernel Code Review

```
int tpm_get_random(u32 chip_num, u8 *out, size_t max) {
    struct tpm_chip *chip;
    struct tpm_cmd_t tpm_cmd;
    u32 recd, num_bytes = min_t(u32, max, TPM_MAX_RNG_DATA);
    ...
    tpm_cmd.header.in = tpm_getrandom_header;
    tpm_cmd.params.getrandom_in.num_bytes = cpu_to_be32(num_bytes);
    err = tpm_transmit_cmd( chip, &tpm_cmd,
                            TPM_GETRANDOM_RESULT_SIZE + num_bytes );
    ...
    recd = be32_to_cpu(tpm_cmd.params.getrandom_out.rng_data_len);
    memcpy(out, tpm_cmd.params.getrandom_out.rng_data, recd);
    ...
}
```

# Kernel Code Review

```
int tpm_get_random(u32 chip_num, u8 *out, size_t max) {
    struct tpm_chip *chip;
    struct tpm_cmd_t tpm_cmd;
    u32 recd, num_bytes = min_t(u32, max, TPM_MAX_RNG_DATA);
    ...
    tpm_cmd.header.in = tpm_getrandom_header;
    tpm_cmd.params.getrandom_in.num_bytes = cpu_to_be32(num_bytes);
    err = tpm_transmit_cmd( chip, &tpm_cmd,
                            TPM_GETRANDOM_RESULT_SIZE + num_bytes );
    ...
    recd = be32_to_cpu(tpm_cmd.params.getrandom_out.rng_data_len);
    memcpy(out, tpm_cmd.params.getrandom_out.rng_data, recd);
    ...
}
```

# Kernel Code Review

```
int tpm_get_random(u32 chip_num, u8 *out, size_t max) {
    struct tpm_chip *chip;
    struct tpm_cmd_t tpm_cmd;
    u32 recd, num_bytes = min_t(u32, max, TPM_MAX_RNG_DATA);
    ...
    tpm_cmd.header.in = tpm_getrandom_header;
    tpm_cmd.params.getrandom_in.num_bytes = cpu_to_be32(num_bytes);
    err = tpm_transmit_cmd( chip, &tpm_cmd,
                            TPM_GETRANDOM_RESULT_SIZE + num_bytes );
    ...
    recd = be32_to_cpu(tpm_cmd.params.getrandom_out.rng_data_len);
    memcpy(out, tpm_cmd.params.getrandom_out.rng_data, recd);
    ...
}
```

# Kernel Code Review

```
int tpm_get_random(u32 chip_num, u8 *out, size_t max) {
    struct tpm_chip *chip;
    struct tpm_cmd_t tpm_cmd;
    u32 recd, num_bytes = min_t(u32, max, TPM_MAX_RNG_DATA);
    ...
    tpm_cmd.header.in = tpm_getrandom_header;
    tpm_cmd.params.getrandom_in.num_bytes = cpu_to_be32(num_bytes);
    err = tpm_transmit_cmd( chip, &tpm_cmd,
                            TPM_GETRANDOM_RESULT_SIZE + num_bytes );
    ...
    recd = be32_to_cpu(tpm_cmd.params.getrandom_out.rng_data_len);
    memcpy(out, tpm_cmd.params.getrandom_out.rng_data, recd);
    ...
}
```

# Kernel Code Review

```
int tpm_get_random(u32 chip_num, u8 *out, size_t max) {
    struct tpm_chip *chip;
    struct tpm_cmd_t tpm_cmd;
    u32 recd, num_bytes = min_t(u32, max, TPM_MAX_RNG_DATA);
    ...
    tpm_cmd.header.in = tpm_getrandom_header;
    tpm_cmd.params.getrandom_in.num_bytes = cpu_to_be32(num_bytes);
    err = tpm_transmit_cmd( chip, &tpm_cmd,
                            TPM_GETRANDOM_RESULT_SIZE + num_bytes );
    ...
    recd = be32_to_cpu(tpm_cmd.params.getrandom_out.rng_data_len);
    memcpy(out, tpm_cmd.params.getrandom_out.rng_data, recd);
    ...
}
```

# Kernel Code Review

```
int tpm_get_random(u32 chip_num, u8 *out, size_t max) {
    struct tpm_chip *chip;
    struct tpm_cmd_t tpm_cmd;
    u32 recd, num_bytes = min_t(u32, max, TPM_MAX_RNG_DATA);
    ...
    tpm_cmd.header.in = tpm_getrandom_header;
    tpm_cmd.params.getrandom_in.num_bytes = cpu_to_be32(num_bytes);
    err = tpm_transmit_cmd( chip, &tpm_cmd,
                            TPM_GETRANDOM_RESULT_SIZE + num_bytes );
    ...
    recd = be32_to_cpu(tpm_cmd.params.getrandom_out.rng_data_len);
    memcpy(out, tpm_cmd.params.getrandom_out.rng_data, recd);
    ...
}
```

# The Results

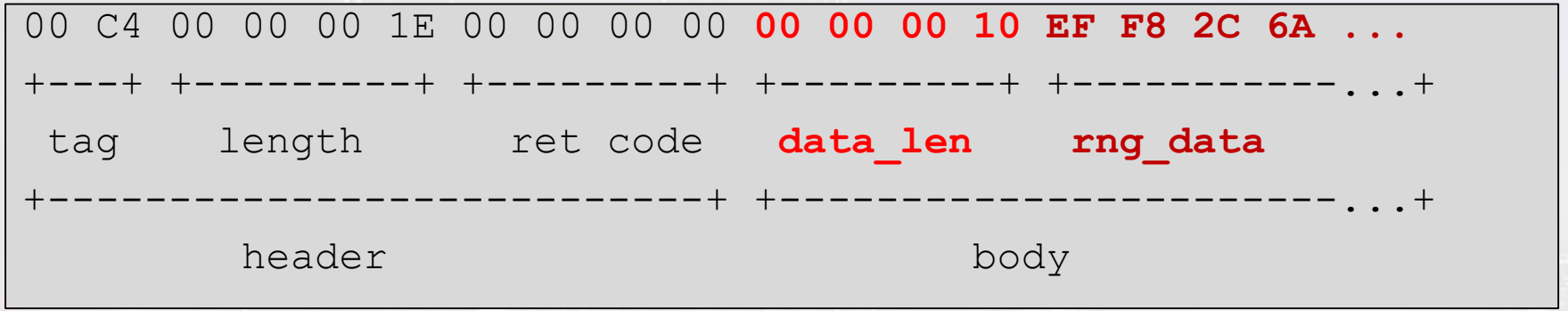
- **Many memory corruption issues:**
  - Linux Kernel: 6
  - U-Boot: 2
  - Coreboot: 1
  - tboot: 13
  - Tianocore EDK2: 10
- **Root cause: Fragile response payload parsing**
- **Some bugs are specific to a TPM chipset vendor (lowest driver layer)**
- **Other bugs affect the generic TPM command/response interface**
  - Both TPM v1.2 and v2.0
  - PCR Read, Get Random, Seal, Unseal, NV Read, Get Capability

# At the Packet Level: Request

```
00 C1 00 00 00 0E 00 00 00 46 00 00 00 10
+---+ +----------+ +----------+ +----------+
 tag    length       ordinal     size_req
+-------------------------------+ +----------+
          header                      body
```

```
tpm_cmd.header.in = tpm_getrandom_header;
tpm_cmd.params.getrandom_in.num_bytes = cpu_to_be32(num_bytes);
tpm_transmit_cmd( chip, &tpm_cmd, TPM_GETRANDOM_RESULT_SIZE + num_bytes );
```

# At the Packet Level: Response




```
tpm_transmit_cmd(chip, &tpm_cmd, 0x1A);
recd = be32_to_cpu(tpm_cmd.params.getrandom_out.rng_data_len);
memcpy(dest, tpm_cmd.params.getrandom_out.rng_data, recd);
```

# At the Packet Level: Response

```
00 C4 00 00 00 1E 00 00 00 00 00 00 FF FF AA AA AA AA ...
+---+ +---------+ +---------+ +---------+ +----------...+
 tag    length      ret code   data_len    rng_data
+-------------------------+ +-----------------------...+
         header                          body
```

```
tpm_transmit_cmd(chip, &tpm_cmd, 0x1A);
recd = be32_to_cpu(tpm_cmd.params.getrandom_out.rng_data_len);
memcpy(dest, tpm_cmd.params.getrandom_out.rng_data, recd);
```


nccgroup
Didn’t The TCG Anticipate This?

# Authorization Sessions

- **HMAC**
  - Appended to command and response packets.
  - Defense against payload tampering.
  - Guarantees Integrity: Packet hasn't been tampered with.
  - Guarantees Authenticity: By knowledge of *shared secret*.
- **Parameter Encryption**
  - Guarantees Confidentiality: Can transmit secrets on the bus w/o exposure.
- **Rolling Nonces**
  - Prevent replay attacks.

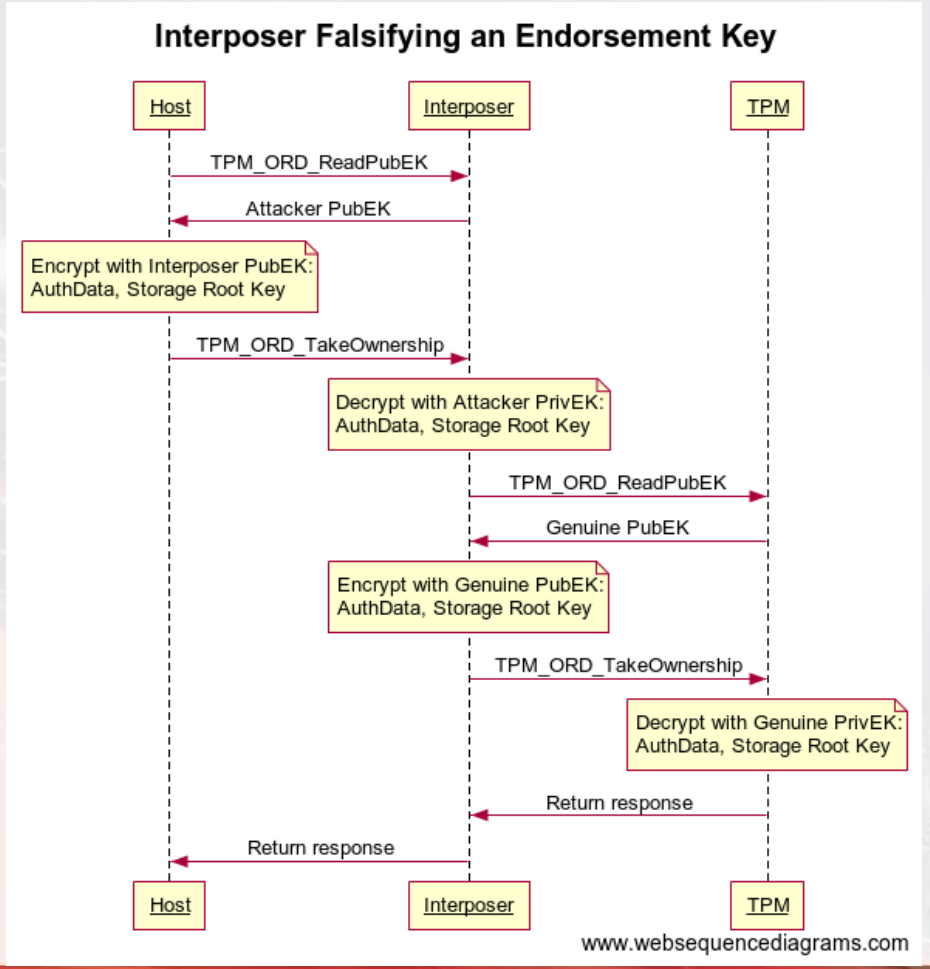
# A Few Problems With That…

- **Authorization Sessions:**
  - TCG specification says auth sessions are optional for many commands.
  - Across the board HMAC not applied to critical commands:
    - Ex: `TPM_ORD_PcrExtend` and `TPM_ORD_GetRandom`
- **HMAC Verification:**
  - tboot `TPM_ORD_Unseal` cmd uses HMAC, but resp HMAC is not verified.
- **Weak Nonces:**
  - tboot uses uninitialized stack memory for nonce.
  - Kernel generates nonce with `TPM_ORD_GetRandom` command.

# Endorsement Key

- **How do I know if I'm speaking to a real TPM or an interposer?**
- **Endorsement Key (EK)**
  - Embedded in TPM.
  - Host can request EK using `TPM_ORD_ReadPubEk` cmd.
- **Used by critical operations**
  - How the Auth Session HMAC shared secret is provisioned.
    - Ex: Taking ownership of TPM (`TPM_ORD_TakeOwnership`)
  - Owner encrypts "AuthData" using PubEK.
  - TPM decrypts "AuthData" with PrivEK.
- **Critically Important:**
  - Must verify the EK Certificate to prove identity of TPM.

## Interposer Falsifying an Endorsement Key



# It Gets Worse…

- We could not find any host-side drivers that actually verify the EK.

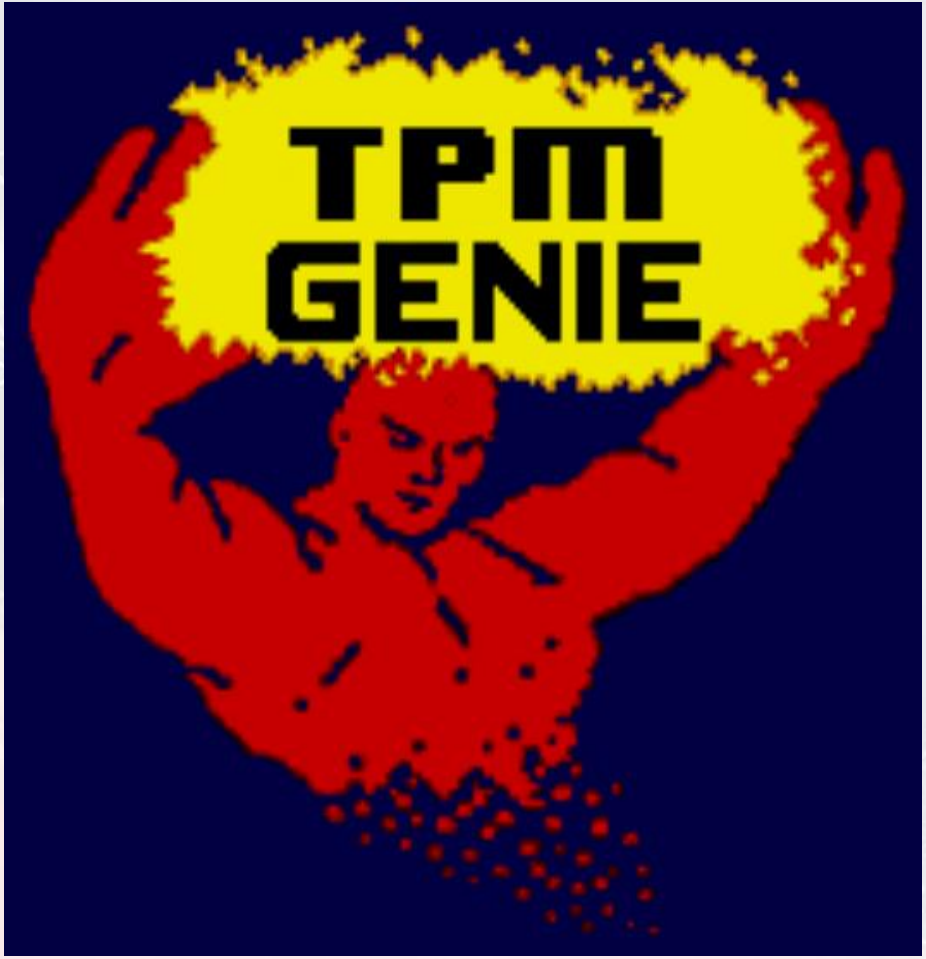
- **Other problems:**
  - Not all manufacturers publish their EK Certs.
  - Some Nuvoton TPMs crash when requesting EK Cert from NVRAM.
  - Some TPMs don’t ship with EK Cert, or allow buyers to provision their own.

- **Impact:**
  - Interposer provides owner with its PubEK
  - Owner sends AuthData to Interposer, encrypted with attacker PubEK
  - Authorization Sessions are defeated.


nccgroup
Designing an Interposer

# Interposer Design

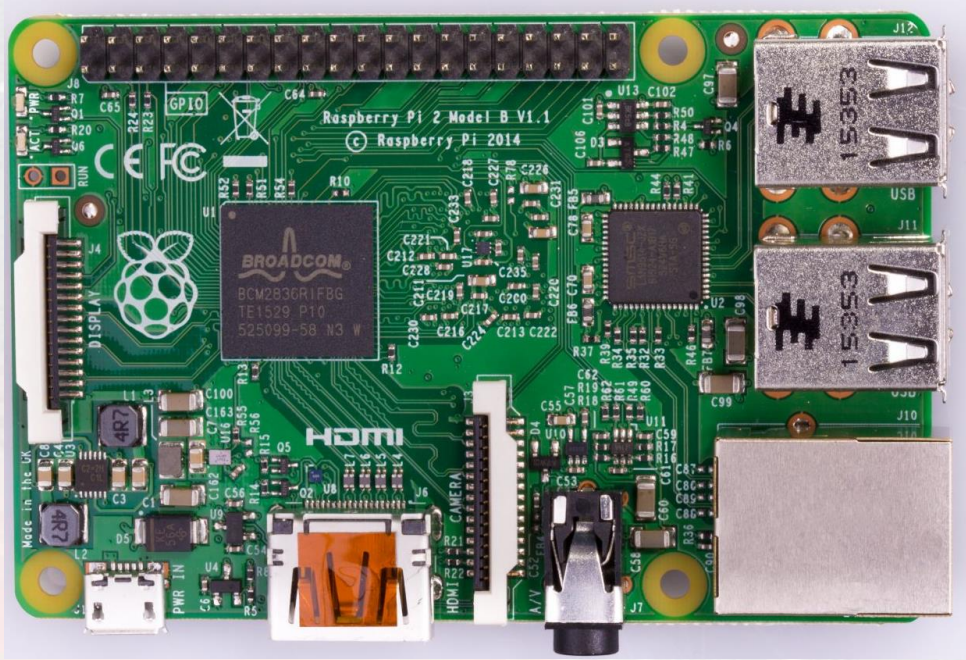
- **Implant in discrete TPM socket**
- **Sit between main board and TPM daughter card**
  - Fool host and TPM into thinking they're talking to each other
- **Allows most traffic to pass through without modification**
- **Modify traffic at opportune time to…**
  - Trigger memory corruption in host response parser.
  - Control PCR Extend operations
  - Control HW RNG output


TPM
GENIE

# Components – Victim Device

- **Raspberry Pi 2 Model B**

- **Why?**
  - Runs Linux
  - Easy to set up
  - Inexpensive
  - Has GPIO header simulating discrete socket



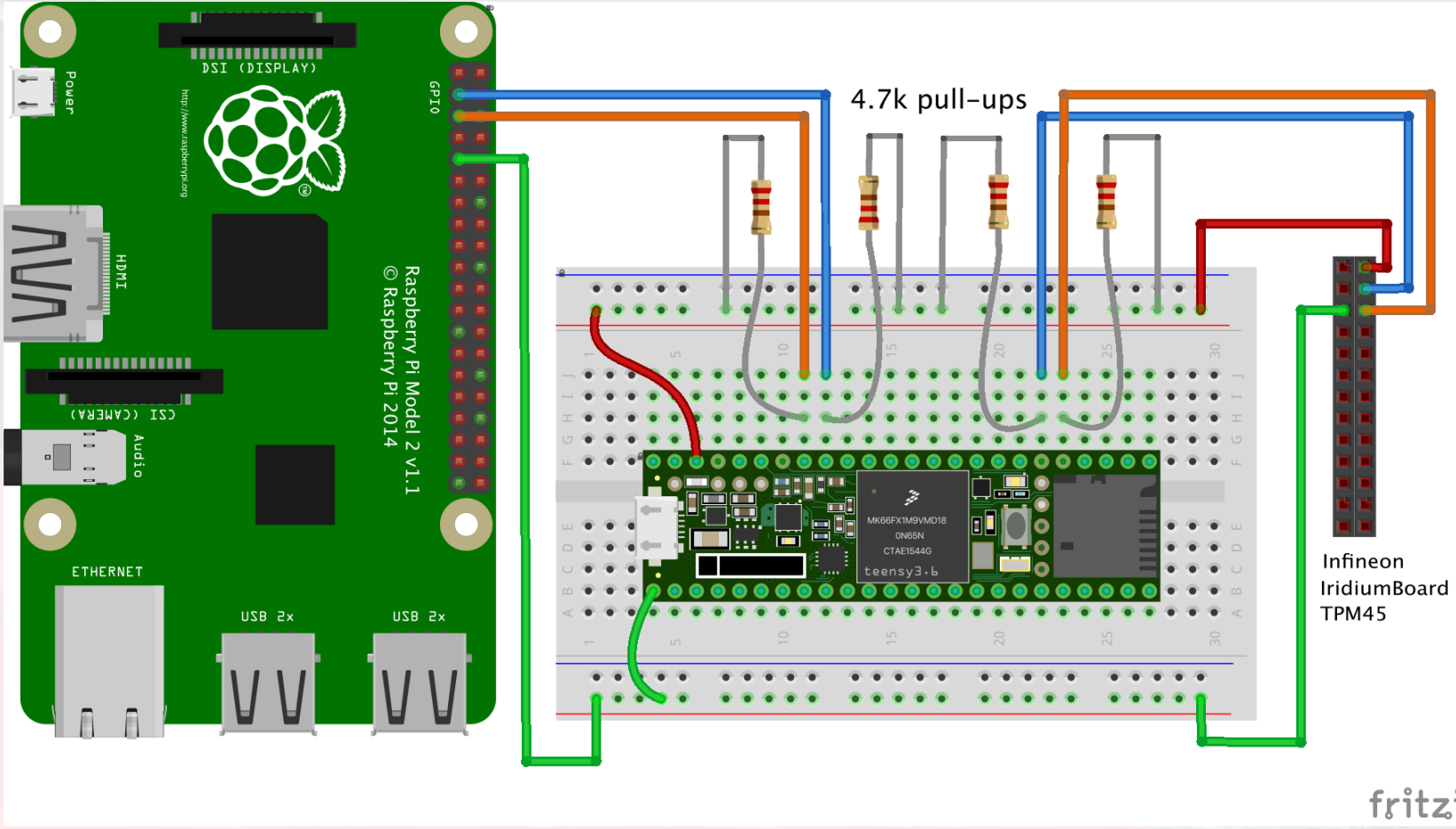
# Components – TPM

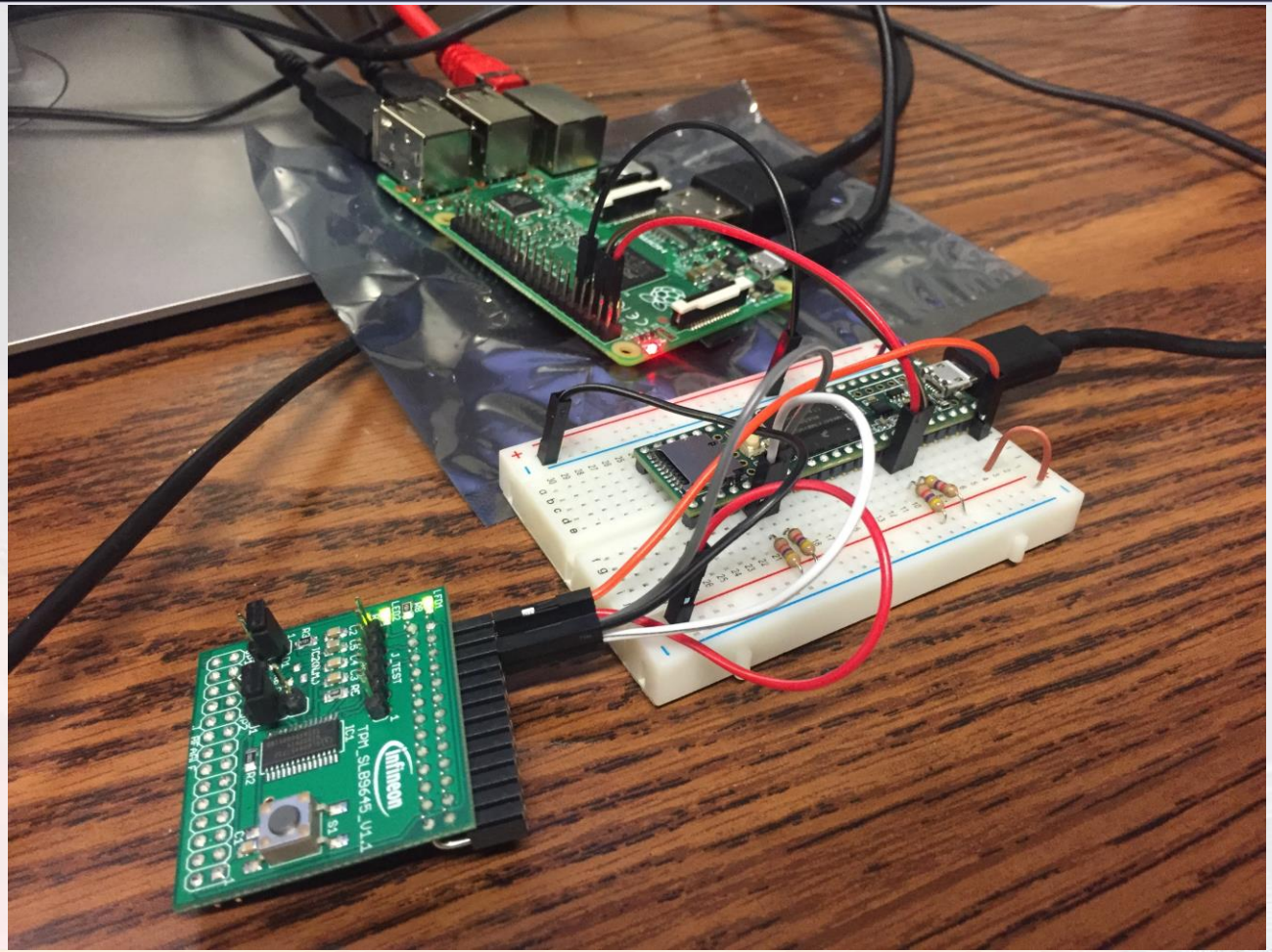
- **Infineon TPM45 Iridium Board**
  - Based on SLB9645
  - TPM v1.2 specification
- **Why?**
  - Inexpensive devkit
  - Designed to work with Raspberry Pi
  - I2C protocol simplifies PoC

# Components – Interposer Device

- **Teensy 3.6 Microcontroller**

- **Why?**
  - Inexpensive MCU
  - Powerful (180 MHz)
  - Arduino simplifies PoC
  - Has multiple I2C busses


DSI (DISPLAY)
Power
GPIO
HDMI
CSI (CAMERA)
Audio
ETHERNET
USB 2x
USB 2x
Raspberry Pi Model 2 v1.1
© Raspberry Pi 2014
4.7k pull-ups
MK66FX1M9VMD18
0N65N
CTAE1544G
teensy3.6
Infineon
IridiumBoard
TPM45
fritzing


Infineon
TPM_SLB9645_V1.4

# TPM Genie Firmware

- **Arduino based:**
  - About 1200 lines of C.
  - Most code is for pretty printing TPM packets on UART console
  - Use `i2c_t3` library for dual I2C bus support
- **Effort:**
  - 10 days to develop initial PoC
  - 15 days to polish up and make not suck
- **Main Challenges:**
  - Delicate timing constraints
  - Raspberry Pi I2C clock stretching bug
  - SDA/SCL cross-talk on messy bread board


nccgroup
Demos

# Impair Hardware RNG

- Interpose `TPM_ORD_GetRandom` command
- **Impact:**
  - Undermine the Linux kernel hardware RNG
  - Weaken system entropy
  - Impair cryptographic operations on the host

```
# dd if=/dev/hwrng count=1 bs=16 | hexdump -C
...
00000000  43 61 6e 53 65 63 57 65  73 74 20 32 30 31 38     |CanSecWest 2018|
```

# Kernel Memory Corruption

- Kernel's `TPM_ORD_GetRandom` parser is fragile
- **Impact:**
  - This ordinal is used by `hw_rng` and `integrity` subsystems.
  - Compromise kernel by returning malformed response packet

```
# dd if=/dev/hwrng count=1 bs=16
<crash>
[65.650713] Unable to handle kernel paging request at virtual address c80255aa
<crash>
```

# PCR Spoofing

- Interpose `TPM_ORD_PcrExtend` command ordinal
- **Impact:**
  - Remote Attestation: Fool trusted remote party into believing that a tampered OS has actually booted into an expected state
  - Sealed Storage: Allow secrets to be unsealed even if the firmware has been tampered with.

```
# python3 pcr_extend.py -i 0 -f data_good
# python3 pcr_extend.py -i 0 -f data_bad
# python3 pcr_read.py -i 0,1

PCR 00: d6 eb c4 e0 4e 16 12 a1 ae 46 5c 51 c0 90 60 8b c5 e6 e1 74
PCR 01: d6 eb c4 e0 4e 16 12 a1 ae 46 5c 51 c0 90 60 8b c5 e6 e1 74
```

# Conclusions

# Conclusions

- **Across the board TPM drivers are buggy**
  - Poor validation of variable-length response structures
- **Design does not defend against local attack vectors**
  - Interposer controls everything on the bus:
    - Register reads/writes, locality, all cmd/resp packet data
  - TCG specification doesn't require Session HMAC for many critical commands.
  - Cannot easily verify the identity of your TPM hardware.
- **Using a TPM can *increase* your system's attack surface**

# Patch Availability

- **Linux Kernel:**
  - Very quick response!
  - Patches for memory corruption bugs now available in v4.16.
  - Plan to backport patches to stable branches: 3.18, 4.4, 4.9, 4,14, and 4.15
  - Unfortunately EK verification and mandatory Auth Session usage is not ready.
- **Coreboot**
  - Unfixed. Report can be found in public bug trackers.
- **U-Boot:**
  - Fixed in master branch.
- **Tianocore EDK2 / tboot**
  - Intel PSIRT has stated that they do not plan on patching the reported vulnerabilities.

# Advice For TCG and TPM Manufacturers

- **Trusted Computing Group**
  - Make Authorization Sessions mandatory for all commands.
  - Publish clear guidance on the need to verify the EK Certificate.
  - Work closely with those that implement TPM drivers to ensure they follow best practices.

- **TPM Chip Manufacturers:**
  - Every TPM must have an Endorsement Certificate.
  - … that is published online so the drivers can verify it.

# Advice for Platform Engineers

- **For now, avoid using discrete TPMs on daughter cards.**

- **Slightly better: Make serial bus access difficult.**
  - Soldered onto mainboard.
  - Cover with a RF shield, under-fill with epoxy resin.
  - Bury serial bus in inner layer of PCB. Difficult because of pull up resistors.

- **Even better: Make bus access much more difficult.**
  - Integrate TPM within the processor.
  - Serial bus is never exposed externally.

# Next Steps / Future Work

- **Improve PoC**
  - Add remote access capability for secret exfiltration
- **Improve hardware**
  - Add support for TPM v2.0, SPI and LPC serial buses
  - Miniaturize, or disguise as legitimate TPM daughter card.
- **Audit more drivers**
  - We only scratched the surface.

# Questions?

https://github.com/nccgroup/TPMGenie

Thanks: Rob Wood, Sultan Qasim Khan,
Ian Robertson, Jason Meltzer, Jon Szymaniak