TC 101: Roadmap

The challenge
- potential failures of my PC and other devices
- threat model

Dimensions of the solution
- what is trust about?
- principles for engineering trust into a system

Building blocks
- hardware and software components
- requirements of a trusted OS

Existing Technology
- components
- ecosystem
The challenge: 

*Shall I trust this?*
Man charged over Woolwich murder

A second man, Michael Adebolajo, is charged with the murder of Drummer Lee Rigby in Woolwich, south-east London.

Protestors pack Istanbul square

Thousands of people flood into Istanbul's Taksim Square as not police withdraw, the latest step in days of protests sparked by plans to develop a park in the Turkish city.

In pictures: New clashes
Possible approach to trusted computing?
Scenario: running \textit{gcc}

- checking that I installed the right \textit{gcc} package
- could use a package manager...
- could do the steps manually

\textbf{objective:} am I installing the ‘right’ gcc
$ wget http://ftp.uk.debian.org/debian/pool/main/g/gcc-defaults/gcc_4.4.5-1_i386.deb

$ md5sum gcc_4.4.5-1_i386.deb
5a48cd3518113e654a4fbec1e69f6ea4 gcc_4.4.5-1_i386.deb

is this actually making an MD5 sum of the nominated file?
$ wget http://ftp.uk.debian.org/debian/pool/main/g/gcc-defaults/gcc_4.4.5-1_i386.deb

$ md5sum gcc_4.4.5-1_i386.deb
5a48cd3518113e654a4fbec1e69bfea4 gcc_4.4.5-1_i386.deb

- is this actually making an MD5 sum of the nominated file?
- is /usr/bin/md5sum the same binary that was installed originally?
- does the supplied binary correctly compute MD5 sums?
- am I executing /usr/bin/md5sum?
- is the shell behaving as I expect?
Regress

- verifying the correct operation of the package relies on the correct operation of the utility
- verifying the verifier relies on the correct operation of the shell
- verifying the correct operation of the OS relies on ... lower level stuff
- what am I really relying on?

$ which gcc
/usr/bin/gcc

$ md5sum /usr/bin/gcc
72efc7342fd5bc5c99bc3d97aef3ef

$ /usr/bin/md5sum /usr/bin/gcc
a664b65ed7b67aa48978a12dc13dd7a5

Read Reflections on Trusting Trust, By Ken Thompson
Layering our security

- layered approaches pervade our computer systems architecture
  - commoditization is key
- we have many approaches to security ...
  - different at each layer
- correctness and verification have their place
  - but are insufficient alone
- malware operation:
  - change the machine code
  - change the context (calling stack, etc., interrupt services)
  - change the configuration
Threat model

- Software-based attacks, delivered by network
  - using published APIs; exploiting vulnerabilities

- Software-based attacks, delivered by other devices, with physical access
  - peripherals, boot media, ...

- Limited hardware attack
  - “things that involve opening the case”

- Full, in-depth hardware attack
  - “major lab facility”
Threat model

- **Software-based attacks, delivered by network**
  - fully in-scope: significant for almost all internet users

- **Software-based attacks, delivered by other devices, with physical access**
  - largely in scope, but have regard to the duration of the access

- **Limited hardware attack**
  - largely out of scope

- **Full, in-depth hardware attack**
  - the classical well-funded adversary – will succeed
Dimensions of the solution: What characteristics are needed in order to trust a system?
The Trusted Computing Base (TCB) of computer system is “the totality of protection mechanisms within it, including hardware, firmware, and software, the combination of which is responsible for enforcing a computer security policy.” [Orange Book]

• what should be the design goals for the TCB?
“Attack surface” - PC

- pre-boot (BIOS, UFEI)
- firmware, option ROMs
- management functions (SMM, SMI)
- OS kernel
- kernel-mode drivers
- application software and drivers
- peripherals
- etc.
Steps to Trust

Graeme Proudler says it is safe to trust something when:

- it can be unambiguously identified, and
- it operates unhindered, and
- the user has first-hand experience of consistent, good, behaviour
- or the user trusts someone who vouches for consistent, good, behaviour.
Steps to Trust

Graeme Proudler says it is safe to trust something when:

An entity can be trusted if it always behaves in the expected manner for the intended purpose.

(TCG 2004)
trust 1. (I) /information system/ A feeling of certainty (sometimes based on inconclusive evidence) either (a) that the system will not fail or (b) that the system meets its specifications (i.e., the system does what it claims to do and does not perform unwanted functions).

(See: trust level, trusted system, trustworthy system. Compare: assurance.)
trusted computer system  1. (I) /information system/
A system that operates as expected, according to design and policy, doing what is required – despite environmental disruption, human user and operator errors, and attacks by hostile parties – and not doing other things [NRC98]. (See: trust level, trusted process. Compare: trustworthy.)

trustworthy system  1. (I) A system that not only is trusted, but also warrants that trust because the system’s behavior can be validated in some convincing way, such as through formal analysis or code review. (See: trust. Compare: trusted.)
Words like ‘Trust’ get some people very excitable. Our objective here is not to explore the philosophy, sociology, or psychology of trust, merely to use the term as a shorthand for a particular collection of reasonably well-defined technical concepts.

Beware

Read Why Trust is bad for Security
By Dieter Gollman
Nomenclature

Some people now regret the name *Trusted Computing*:

| *Trustworthy Computing* could be a better title, | or maybe *Trustable Computing* | but it’s too late to change. |

Which is the better name?
Building Trust in a System

- we want to gain maximum benefit from the *hard-to-alter* characteristics of hardware
- need a cost-effective engineering solution
- we want to factorize the *trusted computing base*
- focus for now on the identification and unhindered operation of systems
Kinds of solution

- All code in ROM/gates;
  - does the same thing forever

- Code in firmware;
  - can be updated

- Critical functionality in ROM, firmware;
  - general code in software

- Fully (re-) programmable system
A goal

- We want to achieve hardware-like trust characteristics in a software programmable system.
  - implement hardware-based roots of trust
    - control secret keys
    - control platform identity
  - build chains of trust which indicate/manage what software is running
    - report *platform state* reliably
    - and/or launch only white-listed software
Extending a Trust Perimeter

- **limited-functionality hardware, (relatively) highly assured**
- TPM, Smartcard, crypto processor., UICC ..

- **more general-purpose code execution platform**
- uses secure element and other hardware to deliver evidences of trustworthiness (identity, normal operation...)

* I may be stretching terminology a little
Roots and Chains

- **root of trust**: also called *trust anchor*: A component that must always behave in the expected manner, because its misbehaviour cannot be detected. The complete set of Roots of Trust has at least the minimum set of functions to enable a description of the platform characteristics that affect the trustworthiness of the platform. [TCG]

- **chain of trust**: Iterative means to extend the trust boundary from the root(s) of trust, in order to extend the collection of trustworthy functions. Implies/entails transitive trust. [TCG, paraphrased]
Making this manageable

- resets and updates
- supply chain management
- points of control: who decides what’s trustworthy?
- using crypto: key management and global secrets
- commoditization vs specialized solutions
- respect free markets/competition law
- verification, proof, assurance, evidence
- privacy
Building Blocks:
What kind of tools do we have?
Protected Storage: identity and more

- protected storage for keys
- prevent secret key from being exported
- use key for controlled purposes
  - encryptions, decryptions, signatures, proof of possession ...
  - subject to access control
- one key – uniquely identify the device holding it
  - (and the platform, if the device is permanently attached)
  - might be bad for privacy
- multiple keys and indirection solve this problem
- extended functionality allows hierarchy of keys/keychain
  - including multiple identities, sub-trees for different users/processes/purposes
- all insufficient alone: bad software could invoke all this functionality
  - but cannot, say, clone the device/platform
controlling the platform boot is a good way to ensure normal operations

immutable root of trust checks pre-boot before passing control

pre-boot checks loader before passing control

etc.

what’s wrong with this?
Key management is tied up with state management

- access to certain keys will be tied to particular states
  - updating ‘good’ reference values
  - managing keys for platform identity
  - preventing one layer from impersonating another
  - doctrine of defence in depth implies we should not simply assume that trusted software is trustworthy
### Achieving Trusted Execution

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>separate trusted co-processor</strong></td>
<td>- critical code run in a separate contained environment</td>
</tr>
<tr>
<td><strong>dual (or multi-) mode processor</strong></td>
<td>- critical functions, I/O etc., only available in trusted mode; typically trusted mode manages the boot process</td>
</tr>
<tr>
<td><strong>measured boot</strong></td>
<td>- make a tamper-proof record of the boot process, so higher level code has evidence of the functionality it relies upon</td>
</tr>
<tr>
<td><strong>secure boot</strong></td>
<td>- check each element contributing to the boot process, and halt (or enter a special state) if a bad one is found</td>
</tr>
<tr>
<td><strong>late launch</strong></td>
<td>- special processor instructions put the platform into a controlled state, then transfers control to assured code, achieving strong isolation for this</td>
</tr>
</tbody>
</table>
Deciding what to launch

- launch-time checking of a whitelist
  - cryptographic hash of object code
- digital signatures on object code
- limits on the installation of code
  - app store model

- this doesn’t inhibit malware which launches within the ‘good’ code’s execution environment
  - e.g. return-oriented programming
- time-of-check-time-of-use (TOCTU) problem
  - is the code which is executing the same code that I checked?
Hardware Access Lock: ratchet approach
(IBM 4758)

- special ratchet register set at zero at system reset
- can only be incremented (up to a maximum value)
- read/write permissions for some memory blocks determined by ratchet value
  - higher value ~ lower trust ~ less access

- reset starts ‘Miniboot 0’ – resides in ROM
- termination of Miniboot 0 increments register and calls Miniboot 1 (can check its integrity first)
- termination of Miniboot 0 increments register and calls OS Kernel (say).
- etc.
TCG approach to Trustworthy state for a computing platform

- essence is to take a **measurement** (a *cryptographic hash*) of each component which contributes to the platform state
  - firmware, kernel, library, application binary, configuration file, etc.
- use those measurements as the basis of deciding if the platform is in a state I trust
  - the same state as last time it booted
  - using components without known vulnerabilities
  - etc.
Building a record of platform state

- is my application untainted?
- is the environment in which it runs untainted?
- how to obtain measurements?
Building a record of platform state

- concept is to have each component in the chain be *measured* by the preceding one
Trusted Computing Group
PC Architecture: Roots of Trust

**Root of Trust for Measurement (RTM)**
- Initiates process of recording what software is running
- E.g. implement in BIOS: in an immutable or securely updatable component

**Root of Trust for Storage (RTS)**
- Implements shielded locations: registers with special integrity or confidentiality characteristics
- Implement in Trusted Platform Module (TPM): in hardware for tamper-proofing

**Root of Trust for Reporting (RTR)**
- Using cryptography to give assurances to third parties
- Built from keys burned into TPM at manufacture time
Remarks

- this process gives us a measured boot process
  - any component in the chain can verify the measurements about the components below/before it by querying the TPM
  - implies transitive trust
- considerable complexity
  - around 200 measurements in a typical boot to a general-purpose operating system
- Question: how could we convert this to a trusted boot?

Read On the Feasibility of Remote Attestation for Web Services
By John Lyle and Andrew Martin
Record of platform state can *attest* that state to other code on the platform.

careful thought needed regarding how to make use of this

It can also be used to demonstrate to a third party that the platform is in a particular state.

this is *remote attestation*
Existing Technology:

*What can I pick up and use?*
Missing piece
Role of TPM in measurement

Provides tamper-proof store for measurements

- we can record additional measurements, but not overwrite old ones, nor undo their recording, without a platform reset

Needs to provide authenticated reports, too

- consider an application asking a TPM driver to report on the platform state
- driver could lie to application, unless TPM output is itself strongly authenticated
- how to communicate this to the desktop user?
TPM: Core Functionality

- **non-volatile storage:**
  - storage root key (SRK)
  - endorsement key (EK)
  - monotonic counters

- **volatile storage:**
  - other keys
  - context, authentication sessions, secure transport sessions, ... 
  - *platform configuration registers* (PCRs)

- **computational functions**
  - crypto, (‘true’) random numbers, key generation

- **shielded locations – protected capabilities**
  - *deliberately not a Turing Machine!*
  - controlled interface to keys and PCRs
Platform Configuration Registers (PCRs)

- Implement trustworthy storage of measurements
- Output of hash function
  - Hash of program code
  - Hash of configuration file
  - Hash of password (if needed)

extend(i, v) :=

\[ pcr[i] \leftarrow \text{hash}(pcr[i], v) \]
Reading Platform configuration registers (PCRs)

- PCRs cannot be directly written, but can be read.

**PCR read operation:**

\[ TPM \text{PCRRRead}(n) := \text{output} \ PCR[n] \]

**PCR 'quote' operation, for nonce \(i\):**

\[ TPM\text{Quote}([n_1, \ldots n_m, i, auth, k]) := \text{output} (\{PCR[n_1], \ldots PCR[n_m], i\}_{key(k)}) \]

“give me a signed, current record of the contents of the PCRs nominated”
Attestation

PCR *Quote* operation enables us to build protocols which *attest* a platform’s state to a third party.

Nonce value allows third party to know that quote is ‘fresh’.

Signing key needs to be certified as belonging to a TPM and/or platform, to the third party’s satisfaction.

This attestation of binaries raises many problems/issues as well: often we would prefer *semantic* attestation.
The Endorsement Key (EK) is held in the TPM:

- gives the platform a unique identity
  - the EK is typically* fixed for the lifetime of the platform
- asserts the platform credentials:
  - secret key is the proof of possession for endorsement credentials, conformance, etc.
- Secret part of EK must not be known outside the TPM
  - but the specification allows it to be generated outside the chip, during manufacture
- These features give rise to significant challenges for manufacture, supply chain management, and provisioning.

TPM v1.2 detail

EK is a 2048-bit RSA key pair
Embedding

- Anything could *claim* to be a TPM, or a trusted platform
  - *root of trust for reporting* is intended to substantiate claims
- To trust the platform we need
  - assurance that it contains a correctly-implemented TPM
  - evidence that the embedding of that TPM within the platform conforms to an evaluated design
- Here is a role for
  - platform manufacturers
  - third-party accreditation
  - *digital certificates*
Attestation Identity Key (AIK)

- Solution to privacy problem is to allow the platform to have arbitrarily many *attestation identity keys* (AIKs)
- Process for signing these involves EK — so can check platform credentials
  - run a protocol with a *"Privacy" CA*
- In use, the AIK has no reference to EK
  - but would generally assert that this AIK belongs to *a* TPM
- Each AIK is strongly bound to the platform, and protected by the root of trust for storage (RTS)
- Alternative is *Direct Anonymous Attestation* (DAA)
  - advanced zero-knowledge protocol
  - resource-intensive; optional implementation
Hierarchy of Keys

- a storage root key (SRK) generated and held in TPM – (re-)initialized by “take ownership” (v1.2)
- private part cannot be extracted
  - can be used to decrypt (see below) only
- key blobs can be encrypted for storage in untrusted locations
- TPM implements ‘LoadKey’ operation to import an encrypted blob, and hold it in temporary store
- so TPM can protect an arbitrarily large collection of keys
Migratable and non-migratable keys

Would want to migrate keys:
- to permit *group* use of keys, in some applications
- practicality and compatibility with non-TPM software
- users move!
- because hardware doesn’t last forever (*)

Would not want to migrate keys:
- to have confidence that some keys are forever bound to a particular TPM
- desirable that some keys are *guaranteed* to exist in only one place (at any one time, anyway)
- danger of keys being available outside the control of *any* TPM

(*) hardware doesn’t last forever
Sealed Storage

- combine cryptographic capabilities with PCRs to give a novel capability: sealed storage
  - ‘sealing’ operation nominates a key, target PCR values, and some data
    - target values need not be current PCR values
  - result is a blob which can
    - only be unsealed only the TPM which sealed it
    - can be unsealed only if the current PCR values match the target PCR values
TPM summary

- headline features:
  - shielded locations / protected capabilities
  - key storage hierarchy
  - platform configuration registers
  - crypto support; key generation; random numbers
  - other functions
TPM and TEE

- TPM supports measured boot
  - extend PCRs with hash of each component
  - different PCRs defined for different boot phases in a PC
- TPM supports late launch
  - high-numbered PCRs reserved for recording hashes of late launched environment
  - major vendors implement processor/chipset capabilities to enable this
- Mobile TPM uses secure boot
  - mobile platforms have special requirements
What is late-launch good for?

- launch a secure hypervisor
  - use it to measure a virtual machine; gives trusted virtualization
  - many complexities around I/O and drivers
- launch a tiny OS/‘shim’/execution environment for a particular task
  - key verification; signature-making; etc.
  - none of your normal OS functionality available
  - see Flicker, TrustVisor
- other?
ARM TrustZone®

- aims for programmable environment which protects confidentiality and integrity of assets
- partition hardware and software resources into two worlds
- minimize content of secure world
- extensions allow a single core to time-slice between worlds
- transfer is via monitor mode – carefully controlled entry point
- can run whole secure OS in secure world

Trust Zone Tier 1

- aims to secure keypad and screen – for entering PINs, etc.
- when application requests payment, secure kernel is invoked
- secure boot via boot ROM to secure OS, then main OS
- master key and SIM (secure element) interface block tied to secure mode
Trust Zone Tier 3

- tier 2 adds DRM capability through address space controller (TZASC)
- hardware crypto acceleration for performance
- other key management, DMA control, etc., brought into the security regime
GlobalPlatform

- cross-industry standards effort
- covers multiple embedded applications on secure chip technology
- its trusted execution environment is designed as a separate zone, distinct from the rich OS of the main platform
- “considerably lower cost than an secure element”
Some other technologies

- **secure UFEI**
  - allows a secure boot through digital signatures
  - Windows 8 makes use of this (as well as TPM measurements, higher up the stack)

- **ChromeOS**
  - uses the TPM for key storage
  - implements a secure boot without critical use of the TPM
Trusted Network Connect

- whether wired or wireless – *network access control* (NAC) is a significant requirement
- TNC designed to allow access to the network to be based on attestation
- with virtual networks (VPN, VLAN) different VMs can be routed separately *based on attestation data*
- High-grade NAC with potentially low overheads.
Reflection
Trusted Computing Critics

- Early proposals for trusted computing platforms met many criticisms
  - some were clearly ill-founded
  - some have been addressed in current designs
    - widely acknowledged and accepted
  - some may persist
    - accepting that many technologies have both ‘good’ and ‘bad’ uses
  - scope for DRM was a cause célèbre
    - but look at our threat model...

- Criticisms deserve to be taken seriously:
  - Richard Stallman: Can you Trust your Computer?
Who decides what to trust?
- should be platform/system *owner*, not vendor, etc.
  - private individual or corporate IT
  - may delegate this decision
- but how to ensure users are not tricked/phished, etc.?
- *AppStore* model appears to be highly effective in controlling the spread of malware.

Highly desirable to have no master key; no global secrets
- most of the designs we shall explore this week have this property

Platform identity is highly sensitive
- see discussion of privacy protections
Attacks

- Few known attacks against the functional spec
  - intention is that there should be *no* software attacks
- TPM v1.1 had a reset attack in the PC platform
- In just about *every* system, the TPM is *not* going to be the weakest point
  - attacks are presently of largely theoretical interest
- Attacks against the secret keys are expected to require large physics lab capabilities
  - one attack (Feb 2010) achieved this
  - Question: what is the impact of such an attack?
- *Question: besides attacking the TPM itself, how else might we compromise the operations it is intended to protect?*
Threat model revisited

- The threat landscape has evolved considerably since this kind of trusted platform was first proposed.
- We should keep asking:
  - Are we looking in the right place?
  - Adopting these technologies will shift the attacks to the next-weakest spot: how well protected is it?
- see: Rolf Oppliger and Ruedi Rytz, *Does Trusted Computing Remedy Computer Security Problems?*
  - it would be timely to write an updated paper
Sober judgement

- Making anything but the smallest delta on a commodity platform price is very hard to justify
  - Incorporating TPM (see below) was designed to be no more than a $5 delta on the price of a platform; these chips currently cost less than $1.
- How much security do you get for $5?
  - Compare with the (very high) costs of bespoke national-security-grade components.
- *But smart use of technology can magnify the investment immensely:*
  - compare with easily-implemented strong cryptography.
  - *Cost to use vs Cost to break* can be very favourable.
Summary

- The Challenge
- Dimensions of the solution
- Building Blocks
- Existing Technologies
- Clear thinking needed
Andrew Martin, MA, DPhil, CEng, MBCS, CITP
CDT Director

Wolfson Building, Parks Road, Oxford OX1 3QD, UK.
+44 (0) 1865 283605
Andrew.Martin@cs.ox.ac.uk
www.softeng.ox.ac.uk/andrew.martin