Software Security

Application-level sandboxing

Erik Poll

Radboud Universiteit Nijmegen

Overview

1. Compartmentalisation
2. Classic OS access control
   - compartmentalisation *between* processes
   - Chapter 2 of lecture notes
3. Language-level access control
   - compartmentalisation *within* a process
   - by sandboxing support in safe programming languages
     - notably Java and .NET
   - Chapter 4 of lecture notes
4. Hardware-based sandboxing
   - compartmentalisation *within* a process,

     also for unsafe languages

1. Compartmentalisation

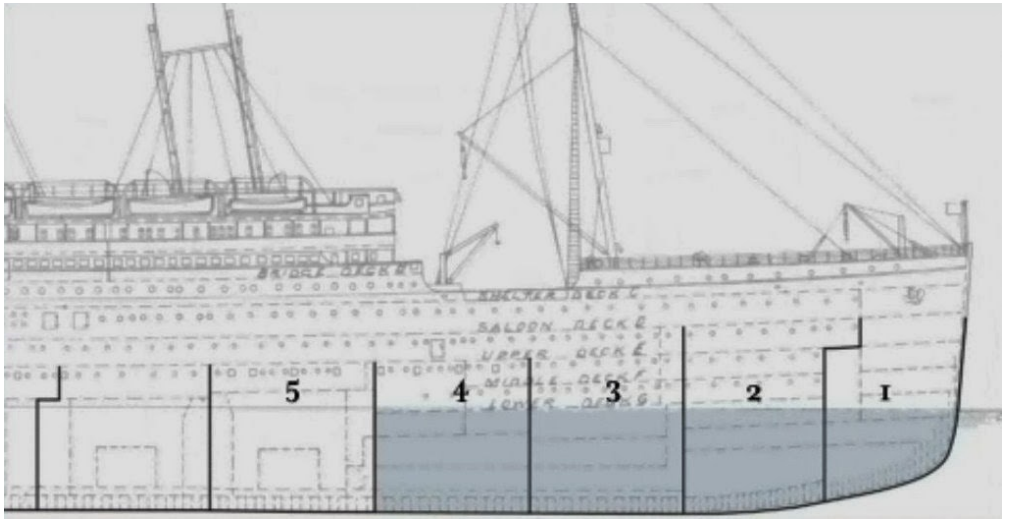
aka

isolation

aka

sandboxing

Compartmentalisation in ships



Titanic

Does this mean compartmentalising is a bad idea?

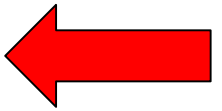
No, but the attacker model was wrong.

- **Making vessel double-hulled would have been a better form of compartmentalising.**

Compartmentalisation examples

Compartmentalisation can be applied on many levels

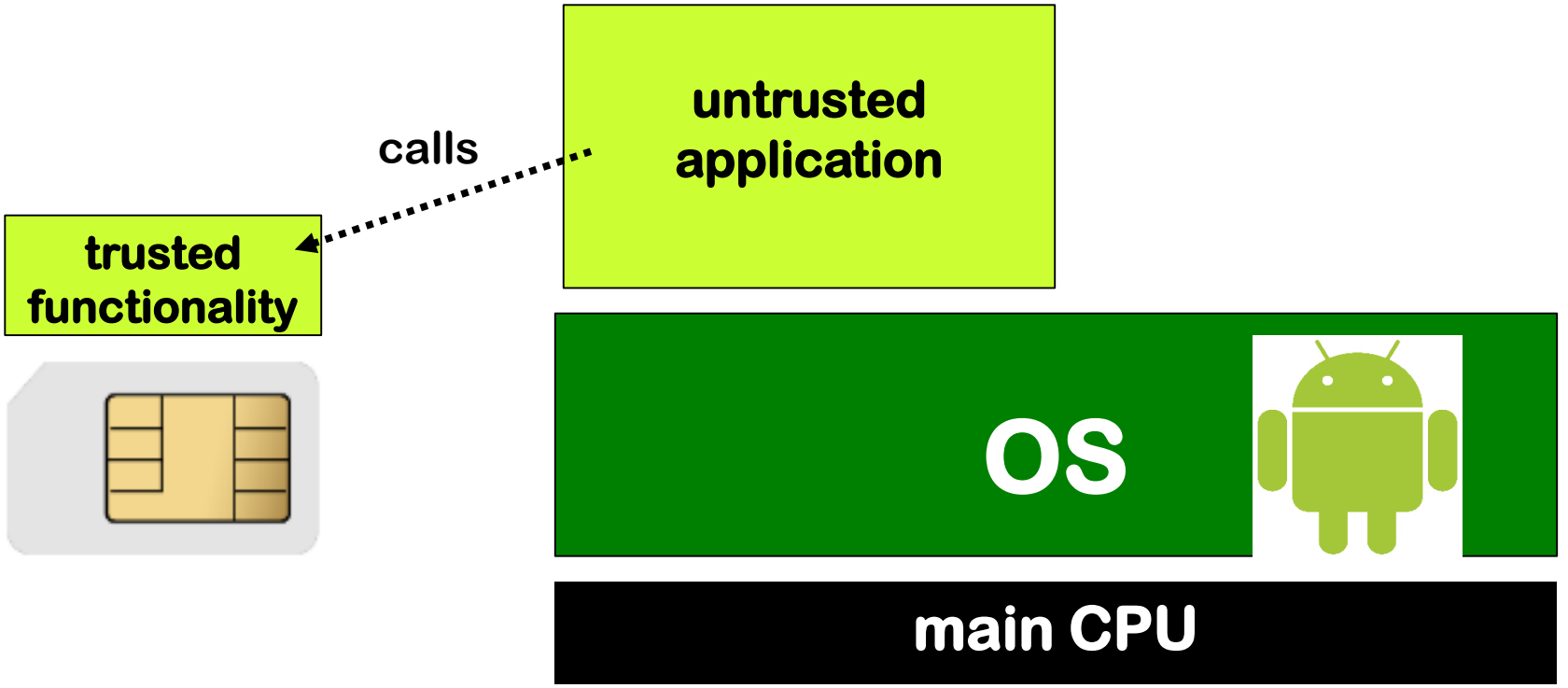
- In an organisation
  - eg terrorist cells in Al Qaida or extreme animal rights group
- In an IT system
  - eg different machines for different tasks
- On a single computer, eg
  - different processes for different tasks
  - different user accounts for different task
  - use virtual machines to isolate tasks
  - partition your hard disk & install two OSs
- Inside a program / application / app / process
  - different 'modules' with different tasks



Focus of today

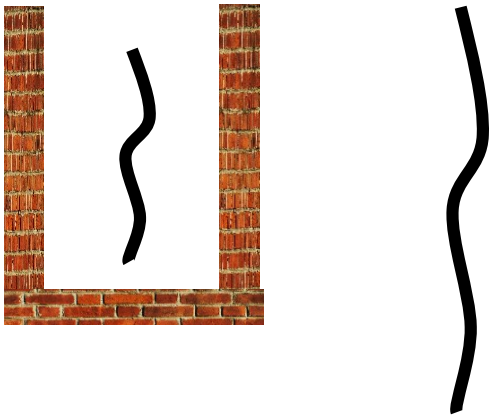
Compartmentalisation example: SIM card in phone

A SIM provides some trusted functionality (with a small TCB) to a larger untrusted application (with a larger TCB)



Isolation vs CIA (Confidentiality, Integrity & Availability)

Isolation is a very useful security property for programs and processes (i.e. program in execution)



'isolation' can be understood in **CIA** terms, as

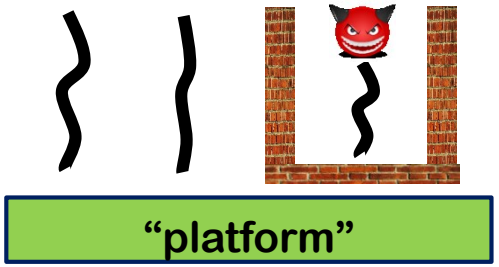
confidentiality and integrity of both data and code,

but conceptually less clear

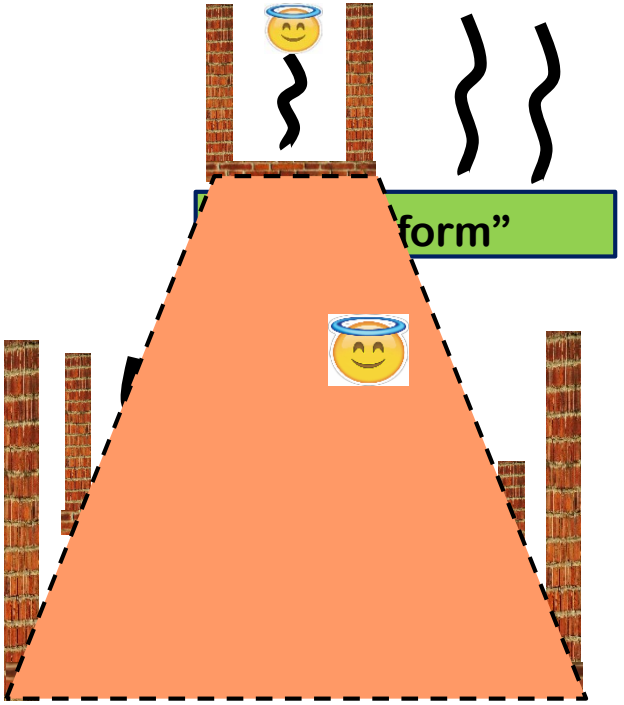
Two use cases for compartments

Compartmentalisation is good to isolate different trust levels

1. to contain a untrusted process from attacking others
   - aka sandboxing



2. to protect a trusted process from outside attacks
   - Here, it makes sense to apply it recursively



Compartmentalisation

Important questions to ask about any form of compartmentalisation

- **What is the Trusted Computing Base (TCB) ?**
  - **Compartmentalising critical functionality inside a trusted process reduces the TCB for that functionality inside that process, but increases the TCB with the TCB of the enforcement mechanism**
- **Can the compartmentalisation be controlled by policies?**
  - **How expressive & complex are these policies?**
  - **Expressivity can be good, but resulting complexity can be bad…**
- **What are input & output channels?**
  - **We want exposed interfaces to be as simple, small, and just powerful enough**
- **Are there any hidden channels?** **Eg timing behaviour**
  - **These can be used deliberately, as covert channels, or exist by accident, as side channels**

Access control

Some compartments offer access control that can be configured

It involves

1. Rights/permissions
2. Parties (eg. users, processes, components)
3. Policies that give rights to parties

   – specifying who is allowed to do what
4. Runtime monitoring to enforce policies,
   which becomes part of the TCB

Compartmentalisation for security design

1. Divide systems into chunks – aka compartments, components,…

   Different compartments for different tasks

2. Give minimal access rights to each compartment

   aka principle of least privilege

3. Have strong encapsulation between compartments

   so flaw in one compartment cannot corrupt others

4. Have clear and simple interfaces between compartments

   exposing minimal functionality

Benefits:

a. Reduces TCB for certain security-sensitive functionality

b. Reduces the impact of any security flaws.