# Protection

Thierry Sans

# Protecting Programs

**How to lower the risk of a program security flaw resulting from a bug?**

1. Build better programs
2. Build better operating systems

# Build Better Programs

# Why are we still vulnerable to buffer overflows?

**Why code written in assembly code or C are subject to buffer overflow attacks?**

- ➡ Because C has primitives to manipulate the memory directly (pointers ect ...)

# Choosing a better programming language

**Some languages are type-safe (i.e memory safe)**

- Pure Lisp, pure Java, ADA, ML …

**Some languages isolate potentially unsafe code**

- Modula-3, Java with native methods, C# …

**Some languages are hopeless**

- Assembly languages, C …

# Type-Safe Programs

➡ Cannot access arbitrary memory addresses

➡ Cannot corrupt their own memory

✓ Do not crash

# So why are we still using unsafe programming languages?

**If other programming languages are “memory safe”, why are we not using them instead?**

➡ Because C and assembly code are used when a program requires high performances (audio, vide, calculus …)
or when dealing with hardware directly (OS, drivers ….)

# Can we write better programs with unsafe programming languages?

**Defensive Programming Approach**

1. Adopting good programming practices
2. Being security aware programmer

**Proactive Approach**

3. Using security libraries
4. Performing penetration testing

**Formal Approach**

5. Using formal methods to verify a program
6. Using formal methods to generate a program

# Defensive Programming Approach

# 2. Adopting good programming practices

**Modularity**

➡ Have separate modules for separate functionalities

✓ Easier to find security flaws when components are independent

**Encapsulation**

➡ Limit the interaction between the components

✓ Avoid wrong usage of the components

**Information hiding**

➡ Hide the implementation

◉ Black box model does not improve security

# 2. Being security aware programmer

- ✓ Check the inputs, even between components that belongs to the same application (mutual suspicion)
- ✓ Be "fault tolerant" by having a consistent policy to handle failure (managing exceptions)
- ✓ Reuse known and widely used code by using design patterns and existing libraries

# Proactive Approach

# 3. Using security libraries - stack smashing protection

**Check that the stack has not been altered when a function returns**

➡ If it has been altered, the program exits with a segmentation fault

✓ Verification embedded into the program by the compiler

# 3. Using security libraries - examples

- ✓ Libsafe
- ✓ StackGuard
- ✓ ProPolice (gcc patches)
- ✓ Microsoft's Data Execution Prevention

# 4. Testing and Penetration Testing

**Testing the functionalities**

✓ Unit test, Integration test, Performance test and so on …

**Testing the security**

✓ Penetration tests

➡ Try to make the software fail by pushing the limits of a "normal" usage i.e test what the program is not supposed to do

# Formal Approach

# 5. Using formal methods to verify a program

**Static analysis** - analyzing the code to detect security flaws

- Control flow - analyzing the sequence of instructions
- Data flow - analyzing how the date are accessed
- Data structure - analyzing how data are organized

➡ Abstract interpretation [Cousot]

✓ Verification of critical embedded software in Airbus aircrafts

# 6. Using formal methods to generate a program

Mathematical description of the problem

*Refinement steps*

Proof of correctness

Executable code
or hardware design

# 6. Using formal methods to generate a program

**Hardware design** (VHDL, Verilog)

- ✓ Used by semi-conductor companies such as Intel

**Critical embedded software** (B/Z, Lustre/Esterel)

- ✓ Urban Transportation
  (METEOR Metro Line 14 in Paris by Alstom)
- ✓ Rail transportation (Eurostar)
- ✓ Aeronautic (Airbus, Eurocopter, Dassault)
- ✓ Nuclear plants (Schneider Electric)

# Pros and cons of using formal methods

✓ Nothing better than a mathematical proof

➡ A code "proven safe" is safe

◉ Development is time and effort (and so money) consuming

➡ Should be motivated by the risk analysis

◉ Do not prevent from specification bugs

➡ Example of network protocols

# Build Better Operating Systems

# Confined execution environment - Sandbox

**A sandbox** is tightly-controlled set of resources for untrusted programs to run in

- ➡ Sandboxing servers - virtual machines
- ➡ Sandboxing programs
  - Chroot and AppArmor in Linux
  - Sandbox in MacOS Lion
  - Metro App Sandboxing in Windows 8
- ➡ Sandboxing applets - Java and Flash in web browsers

# Intrusion Detection/Prevention Systems

- Host-based Intrusion Detection Systems (IDS)
- Host-based Intrusion Prevention systems (IPS)

✓ Based on signatures (well known programs)

✓ Based on behaviors (unknown programs)

➡ Example : Syslog and Systrace on Linux

◉ Vulnerable to malicious programs residing in the kernel called "rootkits"

# Os security features

- **Ubuntu Linux**
  https://wiki.ubuntu.com/Security/Features

- **Windows 7**
  http://resources.infosecinstitute.com/windows-7-security-features/

- **OS X**
  https://www.apple.com/osx/what-is/security.html

# Security Assurance

Thierry Sans

# Why this lecture about "assurance"?

In my experience, Security Assurance is a boring part …

... but you may get a job just because you know what it is!

# What is security assurance?

I ask your company to create a piece of software for me …

… as a non-security expert, what kind of assurance can you give me about the security of your product?

# Why and how to evaluate security?

**Why do we care about security assurance?**

- If you think we should not care about security you have been sitting in the wrong class for half a semester

**How to evaluate the security of a product/system?**

- evaluate the person making the product
- or evaluate the product itself

# Standards and certifications

**How do we agree on evaluation criteria?**

➡ Certifications based on Standards

**Who should run the evaluation?**
**Who delivers the certification?**

➡ A certification authority (trusted third-party) called a registrar

# Outline

**Evaluate and certify an organization**

- ISO/IEC 27000 series

**Evaluate and certify a product or a system**

- TCSEC ("The Orange Book") (1983-1999)
  was the American standard
- ITSEC (1991-2001)
  was the European standard
- ISO/IEC 15408 ("The Common Criteria") (since 1998)
  is the actual international standard

# Certification of an organization

# ISO 27000-series

# ISO/IEC 27k - Security Assurance

**Objective** - provide the best practice recommendations on information security management, risks and controls

➡ equivalent to the ISO/IEC-9000series (quality assurance)

# ISO/IEC 27k in action

## How to get certified?

1. The organization submit an evaluation plan to the registrar
2. The registrar runs the first audit and grant the certification
3. The registrar keeps on auditing the organization to guarantee the certification

# What is inside the ISO/IEC 27k

## The code of practice (ISO/IEC 27002)

- List of 133 candidate control objectives and controls
- Each control must be addressed one by one in the evaluation plan (extras can be added)

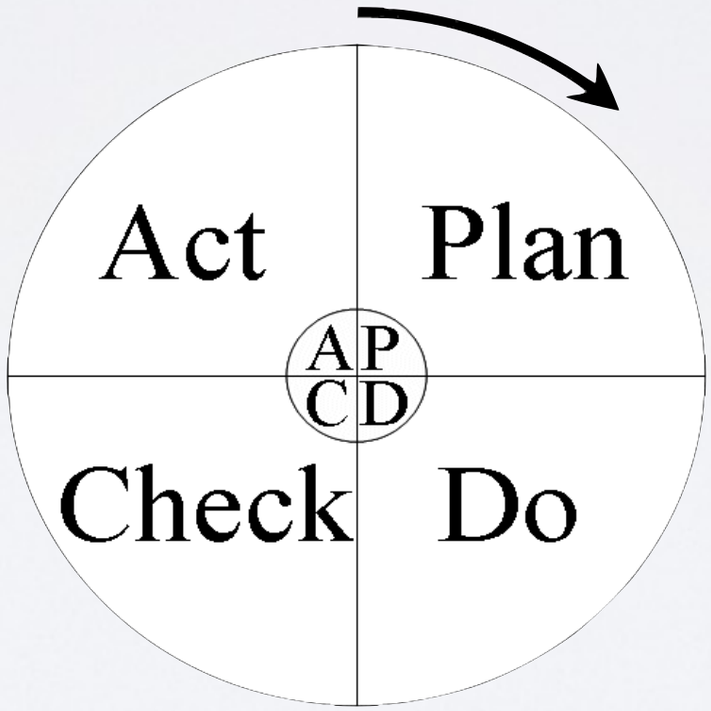
# Governing principles

Based on an iterative problem-solving process

➡ Deming's Wheel (PDCA)

improve
the security assurance

run a risk analysis
and define the security policy




measure
the security solutions

design and build
security solutions
(called controls)

# These controls about ...

- Risk assessment (how to drive the risk analysis)
- Security policy
- Organization of information security (governance)
- Asset management (inventory and classification of information assets)
- Human resources protection (security aspects for employees joining, moving and leaving an organization)
- Physical and environmental security (protection of computer facilities)

# ... and more

- Communications and operations management (infrastructure supporting the activity)
- Access Control (restrictions of access rights)
- Information systems acquisition, development and maintenance (result of the activity)
- Information security incident management (CERT)
- Compliance (ensuring conformance with security policies, standards, laws and regulations)

# Certification of a product/system Common Criteria

# Objective and methods

**Objective** - provide an evaluation methodology

- Defining the set of security functionalities
- Defining a set of assurance requirements
- Determining whether the product or system meet the assurance requirements
- Determining a measure of the evaluation results Evaluation Assurance Level (EAL)

➡ Technical Evaluation based on security assurance methods

- Testing and penetration testing
- Formal development and/or formal verification

# TCSEC - "The Orange Book" (1983-1999)

**Objective** - evaluate and classify computer systems (i.e. OS) regarding the storage, retrieving and processing of sensitive data

➡ Initiated by the US DOD in the 70's

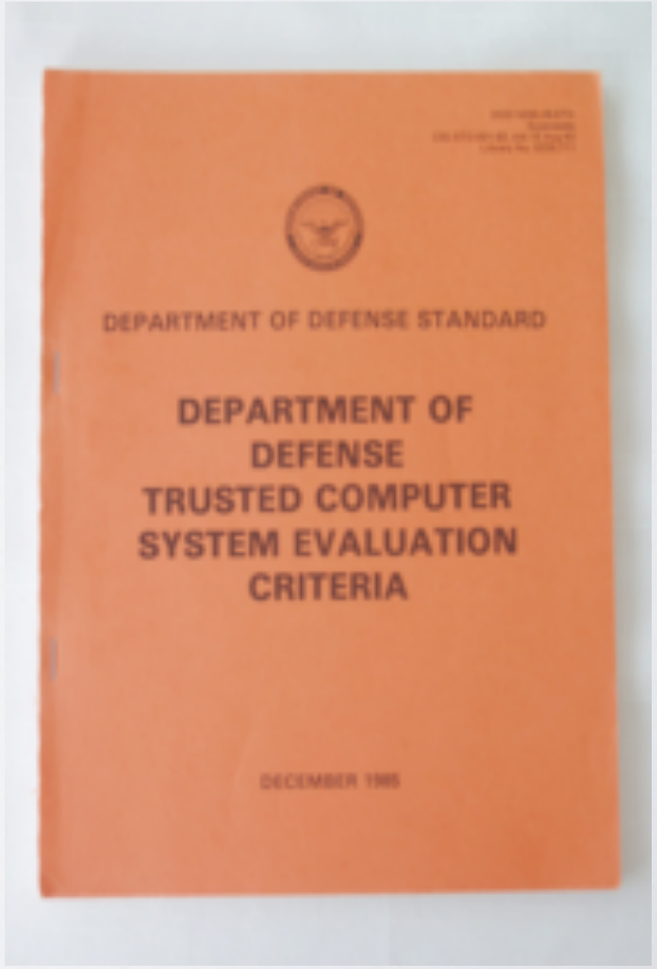



photo from Wikipedia

# Governing principles

Introduce the concept of **policy**

- It must be explicit and enforceable by a computer system
- Two kind of policies are considered DAC and MAC

Introduce the concept of **accountability**

- Users must be identified and authenticated
- Each access must be logged

# TCSEC - security assurance classes (1991-2001)

**Class D** - minimal protection

➡ No security requirements

**Class C** - discretionary security protection

➡ Multi-user environment and data with different sensitivity levels

**Class B** - mandatory security protection

➡ Object labels, user clearance levels and multilevel security policy

**Class A** - verified protection

➡ Formal design and verification

# The Common Criteria (since 1998)

**Protection Profile** - the functionalities and the security requirements of the product/system

- ➡ Written by the system consumer

**Security Target** - identifies the security properties

- ➡ Written by the software designer in response to the protection profile

Let's look at some protection profiles on http://www.commoncriteriaportal.org/pps/

# Evaluation Assurance Levels (EAL)

**EAL 1** - Functionally Tested

➡ Requires a documentation of the security functions vouching for a minimum confidence regarding their correction but threats are not considered as serious

**EAL 2** - Structurally Tested

➡ Requires the delivery of test procedures and test results

**EAL 3** - Methodically Tested and Checked

➡ Requires the developers to be aware of good software engineering practices

# Evaluation Assurance Levels (EAL)

**EAL 4** - Methodically Designed, Tested and Reviewed

- ➡ Requires good commercial developments methods to ensure good software engineering practices

**EAL 5** - Semi-formally Designed and Tested

- ➡ Requires rigorous commercial development practices supported by a security expert

# Evaluation Assurance Levels (EAL)

**EAL 6** - Semi-formally Verified Design and Tested

➡ Requires a rigorous development environment

**EAL 7** - Formally Verified and Tested

➡ Requires a rigorous security-oriented development environment

Let's see some certified products on http://www.commoncriteriaportal.org/products/

# Drawbacks of product evaluation

1. Preparing the documentation for evaluation is a long time effort

➡ The product is obsolete once certified

2. Going through such an evaluation is a costly process

➡ The return on investment is not necessarily a more secure product

3. The evaluation is performed on the documentation and not on the product itself

➡ A good EAL does not prevent from security flaws

# What are the related jobs

**You can become**

- an auditor evaluating an organization or a product and delivering the certification
- a consultant helping to write the documents needed to pass the certification

**What do you need to know/do?**

- CS and IT systems in general
- the standards
- get certified as an auditor or a consultant and get hired by a registrar