Implementation Issues

Ramki Thurimella
Motivation

- Most attacks don’t happen because of broken cryptography
- Attackers find other easier routes
- Be aware of the environment in which crypto operates
- Many OSs exist, but none in widespread use that was designed with security as a primary goal
- It is impossible to implement a secure system
Motivation (cont.)

- Attacks on cryptography are especially damaging (similar to the burglar having a key) because they are invisible
- Make sure that cryptography is NEVER the weakest link
- Crypto systems last for a long time (40 to 50 years)
Creating Correct Programs

- A program is *correct* if it behaves according to its specs

**Specs of a program**
- Requirements: informal description of what it achieves
- Functional Specs: Detailed & exhaustive definition of the behavior (can be measured outside). Most important
- Implementation Design: Document detailing internals
Creating Correct Programs (cont.)

- Test & Fix
  
  “Testing can only show the presence of bugs, never the absence of bugs”
  
  -- Edsger Dijkstra

- Current Program proving techniques cannot handle day-to-day programming tasks

- Rules about bugs
  
  - If you find a bug, implement a test that detects it. Keep running this test on the future versions
  - Think what caused it and WHERE else it could be
  - Do a simple statistical analysis. Shows which part is buggy and what type of errors are most frequent
Creating Correct Programs (cont.)

- Lax attitude
  - Programs crash, people lose work, do nothing about it
  - “should have saved it”
  - Vendors make users sign waiver from product liability
  - In the auto industry, bugs are fixed by recalls
Creating Correct Programs (cont.)

- Authors’ advice
  - Take lessons from the airline industry
  - Every part is examined
  - When a part is changed, it is recorded and signed by the supervisor

- One of the *safest* industries

- Expensive

- Advocate same approach, though VERY expensive
Creating Secure Software

- Correct software
  - Hit button A, B will happen

- Secure software
  - Lack of functionality. For e.g. No matter what the attack is, never give a root shell without proper credentials

- Can test for functionality not for lack of it

- Standard implementation techniques are inadequate to create secure code

- Cryptography is a small part of a larger system
Keeping Secrets

- For secure channel, 2 secrets: key & data
- Consider only secure communication (not long term storage), i.e. send message from A to B
- Discussion limited to transient secrets
- Wiping State
  - Wipe as soon as no longer need it
  - Certainly wipe before losing control of the physical media storing the secrets
Keeping Secrets (cont.)

- Wiping state
  - Surprising number of problems
  - Program written in C/C++
    - Need to depend on the Main program to call the wipe function
    - Could put wiping code in destructors
  - C/C++ has too many other problems
  - Compilers have aggressive optimizations that might undo explicit wipe code
Keeping Secrets (cont.)

- Wiping state
  - Surprising number of problems
  - Program written in C/C++
    - Need to depend on the Main program to call the wipe function
    - Could put wiping code in destructors
  - C/C++ has too many other problems
  - Compilers have aggressive optimizations that might turn wipe code into noops (no operations)
Keeping Secrets (cont.)

- Wiping State
  - Java has its own set of problems
  - Most objects live in heap
  - Don’t have control on garbage control
  - Exception handling “unwinds” the stack and might not delete heap objects
  - Could add `finally` clause and put wipe code there
- Desirable if variables can be declared `sensitive`—system will make sure they don’t linger
Keeping Secrets (cont.)

- **Swap file**
  - Most OSs use virtual memory
  - Can run many programs in parallel
  - Only a portion of the program is kept in memory
  - If a reference is made to outside the portion
    - You have a *page miss*
    - Write one of the pages of the current portion to disk
    - Load the new page from the disk
  - The place on disk where pages are temporarily stored—*the swap file*
  - Encrypt while writing to disk (big performance hit)
  - Or tell OS never to swap out some locations (complicated)
Caches

- Hierarchy of memories (closer to the CPU, faster to access)
- Keeps copies
- Like a page miss, there can be cache miss
- “Invalid” (aka dirty) bit, if the contents are not current, but system sometimes does not clear the contents
- Good news: Only OS can directly access cache
Keeping Secrets (cont.)

- Data Retention in Memory
  - Simply overwriting does NOT delete data
  - SRAM static RAM used for caches, hard drive buffers etc.
  - DRAM used for main memory
    - stores a small charge on a very small capacitor
    - Causes migration of impurities
    - An attacker with physical control can recover
    - cold boot attacks: recover keys after reboot
    - Also switch memory between machines and recover keys
Keeping Secrets (cont.)

- Possible solution
  - Instead of keeping the same value K for a long time
  - Store \( h(R) \oplus K \) where \( R \) is a random value
  - Store \( R \) separately
  - Change \( R \) frequently
  - To retrieve \( K \)
    - Retrieve \( R \) first
    - Computer \( h(R) \)
    - XOR that with \( h(R) \oplus K \)
  - Works only for few bits
Keeping Secrets (cont.)

- Access by others
  - UNIX
    - superusers can read arbitrary locations
    - Can force a core dump of program
  - Windows
    - Attach debuggers to running programs
      - Read memory locations
  - In general, no defense against these attacks
Keeping Secrets (cont.)

- Data Integrity
  - Underlying hardware can corrupt data
  - Use error correcting codes
- What to do in general
  - Not adequate support from programming languages and operating systems to prevent leaks
  - Security considerations affect every part, even outside crypto libraries
Quality of Code

- Simplicity
  - Complexity worst enemy of security
  - Simplicity is king in security
- Secure Channel from Chapter 7 good example
  - Don’t give choices & Options
  - Choice: secure or insecure
- Modularization
  - Design, analyze & implement separately
  - Module—crypto primitive
Quality of Code (cont.)

- Modularization (cont.)
  - Examine interface—simple
  - Each module must solve ONLY its problems
    - Create no side effects
  - Many optimizations are insignificant—don’t do them
    - Do work in large chunks
    - Only optimize if can deliver measurable improvement on performance
Quality of Code (cont.)

- Assertions
  - Distrust other modules
  - Used during development to check logical consistency of states
  - If failure, give detailed reasons why
  - Error caught by an assertion does not lead to security breach
  - Leave them in production code
    - Ignoring might lead to incorrect results
Quality of Code (cont.)

- **Buffer Overflows**
  - One of the most common ways of breaching security

- **Fixes**
  - Always do array bound checking
  - Stay away from languages like C/C++ that don’t automatically do it
Quality of Code (cont.)

- Testing
  - Do extensive testing
  - Testing & security analysis are complementary, but different
  - Generic set of tests to test functional specs
    - One programmer implements a module
    - Another tests it
  - Second to be developed by the programmer of the module
    - Test implementation limits (boundary conditions)
  - Quick test code to test every time program starts with known answers
    - If further development destabilized it would be known
Side Channel Attacks

- Magnetic fields
- Timing attacks
- RF emissions (common)
- Power consumption (especially on smart cards)
- Interference on other data channels
- Impossible to defend against all side channel attacks
  - Being aware, can design with this knowledge