Reflections on Trusting Trust

Ken Thompson

Motivation

- **It is very difficult to determine whether or not you can completely trust the software you use.**

How it Works

- Start with the unmodified C compiler and its source code
- Modify the source code of the compiler so that it will insert a backdoor into the program of your choice (e.g. ”login”) whenever the program is compiled.
- Also modify the compiler source code so that it will insert self-replicating code into the compiler; the self-replicating code inserts itself and the trojan horse above into the compiler whenever the compiler is compiled.

How it Works, cont’d

- Compile the original source for the C compiler, and install the resulting executable binary as the computer’s official C compiler.
- Replace the modified compiler source code with the original source code.
- Now, whenever someone recompiles the compiler, it will contain the trojan horses without any traces in the source code.

Lessons

- You can’t trust that programs you compile are free of trojan horses even if you examine the source code, because the compiler may be modifying them undetected.
- Looking at the compiler code doesn’t reveal this either, because the trojan horse code is only in the binary.

Questions

- **What if you write your own compiler in assembly language? Are you safe then?**
- **Can you really trust any of your software tools?**
- **What about your hardware? Can you trust that? Do you need to?**

How Cryptosystems Fail

Ross Anderson

Motivation

- Crypto systems are hard to build, and understanding how and why they fail will make it easier to build better ones.

Curtain of Silence

- **Information on crypto failures is hard to come by, because governments are the heaviest users and they keep it all secret.**
- **Even in other uses (e.g. banking), it may be to someone's advantage to suppress the fact that a failure has occurred.**
- **Consequently, there is a shortage of information on failures in crypto systems.**

Lessons from ATM industry

- **Cryptosystems fail in ways that are quite different from those that the designers originally considered**
  - Dishonest individuals (trusting the wrong people)
  - Management issues
  - Implementation errors

Lessons from ATM industry, cont'd

- Quality control is of utmost importance; a good design is useless if the implementation causes incorrect behavior
- Certifying that a particular system component (e.g. IBM ″security module″) is secure does not guarantee that the entire system is secure

How should we approach secure systems?

- Concentrate on what is LIKELY to go wrong, not just on what CAN go wrong.
- Design secure systems similar to safety-critical systems

Design paradigm

- Enumerate ALL failure modes, not just the “tricky” ones.
- List clearly what strategy is being adopted to prevent each failure mode.
- Explain how each strategy is implemented, including how failures of other system components are handled.
- Test whether all components, and the system as a whole, can be operated by the actual users (as opposed to the designers).

Questions

- How does the ”curtain of silence” benefit the people designing secure systems? How could it hurt them?
- How do the laws regarding liability in the U.S. vs. the U.K. help encourage or discourage good security practices by corporations?