Michael Salib 6.033 4/9/02

One can never be perfectly cetain that a given C program is free of Thompson's trojan. Perfect certainty is an illusion that does not exist in the real world. At best, we can work to reduce the risk, to lower the window of vunerability.

One way to reduce the risk of catching Thompson's trojan is to exploit the trojan's own weakness. For example, the trojan cannot be too large or complex before optimizers become suspicious enough to start poking into the structure of the compiler binary, looking for the source of bloat. For the same reason, it cannot impose too great a run time performance penalty during compilation (i.e., it cannot perform exhaustive pattern matching on every single input string). Consequently, we can assume that the mechanism used to detect if the login program or compiler itself is being compiled is relatively simple. Such a mechanism may consist of pattern matching on input file name.

Our strategy for detecting the presence of the trojan in our C compiler is simple: first, we obfuscate the source code of our compiler and then compile both the obfuscated compiler source code and the normal compiler source code. The two compiler binaries that result should be bit-for-bit identical. The obfuscation technique used should be specific to each site and should be based on general purpose tools. Otherwise, the trojan could specially compile any special purpose obfuscation or comparison tools used. The trojan cannot, however, afford to specially compile truly general purpose programs like tr or diff without risking inadvertant exposure. Of course, we should analyze not just the compiler itself, but the entire tool chain it relys on as well. This includes lexical analyzers (lex), automatic parser generators (yacc), assemblers, static and dynamic linkers, dependancy tracking tools (make), editors (emacs), and source code control systems (like CVS or bitkeeper).

A second strategy would be to use several different compilers to compile our target compiler. We would then use the resulting compiler binaries recompile our target compiler's source code. The resulting compiler binaries should be bit-for-bit identical. In practice, this is difficult because most compilers are not available as source code and because compiler bootstrapping is complex and idiosyncratic enought that it's unlikely one compiler could successfully build a completely different compiler.