Reflections on Trusting Trust
Ken Thompson

Communications of the ACM v 27 no 8 (1984) pp 761–763.

Not a sample R209 presentation.

Probably the shortest paper we will read this year.

Article version of Ken Thompson’s acceptance speech of the 1984 Turing Award for the UNIX operating system, on behalf of himself Dennis Ritchie.

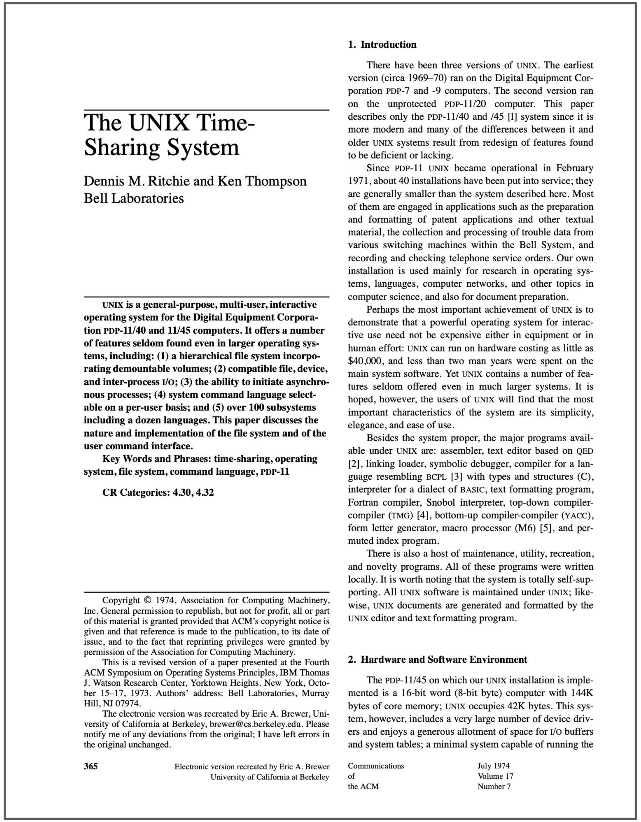
The Turing Award is something like a Nobel Prize for computer science.

Short intro with shout outs and thanks, followed by a thought-provoking technical contribution.

Now seen as a seminal piece of work in computer security.

Not an academic, peer-reviewed paper – but published in highly respected Communications of the ACM (CACM).

The UNIX Time-Sharing System (Ritchie and Thompson)



The UNIX Time-Sharing System

Dennis M. Ritchie and Ken Thompson
Bell Laboratories

UNIX is a general-purpose, multi-user, interactive operating system for the Digital Equipment Corporation PDP-11/40 and 11/45 computers. It offers a number of features seldom found even in larger operating systems, including: (1) a hierarchical file system incorporating demountable volumes; (2) compatible file, device, and inter-process I/O; (3) the ability to initiate asynchronous processes; (4) system command language selectable on a per-user basis; and (5) over 100 subsystems including a dozen languages. This paper discusses the nature and implementation of the file system and of the user command interface.

Key Words and Phrases: time-sharing, operating system, file system, command language, PDP-11

CR Categories: 4.30, 4.32

This is a revised version of a paper presented at the Fourth ACM Symposium on Operating Systems Principles, IBM Thomas J. Watson Research Center, Yorktown Heights. New York, October 15–17, 1973. Authors' address: Bell Laboratories, Murray Hill, NJ 07974.

The electronic version was recreated by Eric A. Brewer, University of California at Berkeley, brewer@cs.berkeley.edu. Please notify me of any deviations from the original; I have left errors in the original unchanged.

1. Introduction

There have been three versions of UNIX. The earliest version (circa 1969–70) ran on the Digital Equipment Corporation PDP-7 and -9 computers. The second version ran on the unprotected PDP-11/20 computer. This paper describes only the PDP-11/40 and /45 [l] system since it is more modern and many of the differences between it and older UNIX systems result from redesign of features found to be deficient or lacking.

Since PDP-11 UNIX became operational in February 1971, about 40 installations have been put into service; they are generally smaller than the system described here. Most of them are engaged in applications such as the preparation and formatting of patent applications and other textual material, the collection and processing of trouble data from various switching machines within the Bell System, and recording and checking telephone service orders. Our own installation is used mainly for research in operating systems, languages, computer networks, and other topics in computer science, and also for document preparation.

Perhaps the most important achievement of UNIX is to demonstrate that a powerful operating system for interactive use need not be expensive either in equipment or in human effort: UNIX can run on hardware costing as little as $40,000, and less than two man years were spent on the main system software. Yet UNIX contains a number of features seldom offered even in much larger systems. It is hoped, however, the users of UNIX will find that the most important characteristics of the system are its simplicity, elegance, and ease of use.

Besides the system proper, the major programs available under UNIX are: assembler, text editor based on QED [2], linking loader, symbolic debugger, compiler for a language resembling BCPL [3] with types and structures (C), interpreter for a dialect of BASIC, text formatting program, Fortran compiler, Snobol interpreter, top-down compiler-compiler (TMG) [4], bottom-up compiler-compiler (YACC), form letter generator, macro processor (M6) [5], and permuted index program.

There is also a host of maintenance, utility, recreation, and novelty programs. All of these programs were written locally. It is worth noting that the system is totally self-supporting. All UNIX software is maintained under UNIX; likewise, UNIX documents are generated and formatted by the UNIX editor and text formatting program.

2. Hardware and Software Environment

The PDP-11/45 on which our UNIX installation is implemented is a 16-bit word (8-bit byte) computer with 144K bytes of core memory; UNIX occupies 42K bytes. This system, however, includes a very large number of device drivers and enjoys a generous allotment of space for I/O buffers and system tables; a minimal system capable of running the

Paper:

- Based on research at Bell Labs - describes the UNIX operating system.
- File system, process model, shell, traps, and statistics.
- Influences including Multics.

But more importantly:

- Incredibly influential design and implementation: UNIX itself, BSD, Linux, macOS/iOS, …
- Literally billions of systems implementing UNIX design principles around the world today

The author, Thompson, was also a coauthor of the C programming language, an has since been a lead author of the Go language.

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

Stage 1: The program that prints itself (good undergraduate programming exercise).

Stage 2: A nifty and instructive observation about the source code of a compiler and the compiler binary: Ideas about program execution may exist only in the compiler binary and other generated binaries, not in the source code at all.

Stage 3: Uses this technique to inject two “bugs” into the compiler: one that perpetuates the compiler changes, and the second that trojans the login program.

The result: A source-code invisible, self-perpetuating trojan of the full operating system. There is the suggestion of generality to linkers, microcode, etc.

Acknowledgement

I first read of the possibility of such a Trojan horse in an Air Force critique [4] of the security of an early implementation of Multics. I cannot find a more specific reference to this document. I would appreciate it if anyone who can supply this reference would let me know.

…

4. Unknown Air Force Document.

Paul Karger and Roger Schell. **Multics Security Evaluation, Volume II: Vulnerability Analysis**. Technical Report ESD-TR-74-193, v II, Electronic Systems Division, Air Force Systems Command, Hanscom Field, Bedford, MA 01731 (June 1974)

It was noted above that while object code trap doors are invisible, they are vulnerable to recompilations. The compiler (or assembler) trap door is inserted to permit object code trap doors to survive even a complete recompilation of the entire system. In Multics, most of the ring 0 supervisor is written in PL/I. A penetrator could insert a trap door in the PL/I compiler to note when it is compiling a ring 0 module. Then the compiler would insert an object code trap door in the ring 0 module without listing the code in the listing. Since the PL/I compiler is itself written in PL/I, the trap door can maintain itself, even when the compiler is recompiled. (38) Compiler trap doors are significantly more complex than the other trap doors described here, because they require a detailed knowledge of the compiler design. However, they are quite practical to implement at a cost of perhaps five times the level shown in Section 3.5. It should be noted that even costs several hundred times larger than those shown here would be considered nominal to a foreign agent.

This US air force report is one of the earliest and most important pieces of work on adversarial reasoning. We haven't assigned it this year, but there's also a retrospective paper worth reading and thinking about.

Discussion topics

What was the contribution being recognised by this Turing award?

What standards of evidence, acknowledgement, and so on, do we hold an invited talk to – vs an academic paper?

The attack as described: What current systems could this idea apply to? How practical is it?

What are the values of source-code vs. binary analysis to find trojans?

It it turtles all the way down?

What are the broader implications?

What about the original Air Force work? How do we feel about the sort-of citation?