Security as an Artificial Science, System Administration, and Tools

Paul C. van Oorschot Version: 29 June 2022[1]

Abstract. *In practice, the security of a system is often strongly impacted by the tools used to create, configure and manage each of its constituent components. Here we consider the role of such tools themselves in the body of knowledge comprising computer and Internet security.*

Modern society runs on networked computer and communication systems. They are relied on directly or indirectly for most of our daily activities—from workplace communications and financial services to entertainment and travel. Many of the systems that we rely on take the form of an operating system platform supporting network communications and hosting application software, all on some underlying hardware.

Operating systems (OSs) and applications are themselves built and managed using a wide variety of tools. Prominent among these are programming languages and their own support systems, e.g., compilers or interpreters, and run-time libraries. By this path, the security properties of our computer systems are strongly related to the software tools used to build, run and maintain the core operating systems, and the tools used in developing and maintaining them and their applications.

In this article we focus on operating systems, and in particular the Unix family, due to its dominant influence. A main question we consider is: What place does the Unix family of operating systems occupy in the *body of knowledge* of computer and Internet security?

The answer depends on who you ask, and what they believe such a body should include as its core elements. (An overview of efforts to determine a body of knowledge for security is given in our Sept-Oct 2021 article in this department [6].) A broader question is: How much of the body of knowledge in security should focus on general or abstract concepts and theory, versus detail-heavy aspects such as OS-specific tools and low level commands that are nonetheless critically important for security? This is related to the roles and relative importance of principles and policies, versus specific mechanisms and practices and arcane shell commands.

My goal here is to encourage thought about such questions, rather than deliver clear answers—which I do not claim to have. The motivation is that collectively reconsidering such questions on a regular basis is beneficial, as we continually refine how and what we teach as practical security—what topics to prioritize. So we will consider: To what extent are the fine-grained details of operating system configuration, and system administration in general, core parts of security knowledge?

Knowledge and Artificial Sciences

Unix is a human-created artifact (as are, for example, the C language and software programs in general). Most traditional science studies phenomena that occur naturally in our physical world. This raises the issue of whether our main question, on the place of Unix in our security body of knowledge, is about science at all. Hardware-software systems are human creations, and their practical security is largely about the behavior of human-created artifacts. This puts computer security squarely in the realm of what 1975 Turing Award winner (`https://amturing.acm.org/award_winners/simon_1031467.cfm`) and 1978 Nobel Prize in Economics winner Herbert Simon called the *artificial sciences* [7]. These differ from the *natural sciences* in the nature of their subject matter, and in the implications of their results.

[1] (c)IEEE *Security & Privacy* 20(6): 74–78, 2022, at `https://ieeexplore.ieee.org/document/9935622`

Note for example that, in the subset of natural sciences that comprise the physical sciences (e.g., chemistry, physics, astronomy), the subjects of experimentation and observation are natural phenomena provided by the physical world and largely beyond our own making. The periodic table of elements in chemistry is essentially fixed, gravity in physics is a function of our universe rather than a human creation, and the solar system is beyond our control—notably, making astronomy an observational science. Research in the natural sciences is strongly associated with experimentation in the empirical world, with learning based on observation and inductive reasoning.

In contrast, we may note that distinct from the natural sciences, mathematics deals with the abstract world, dominated by axiomatic reasoning using deductive logic. Theoretical areas of computer science share this property, but more applied fields, including most of practical security, are different yet again, in that the focus of study is human-created artifacts as noted above. (The full story is a bit more clouded, as computer security is a cross-cutting field whose subfields straddle computer science, engineering, mathematics and other diverse disciplines [3].)

In security, while it has become fashionable to use the term *empirical studies* to include measurement of security-related properties of Internet ecosystems, this largely means studying human-created artifacts, i.e., software, rather than empirical studies in the sense of the natural sciences. Likewise, when security researchers use the term *in the wild*, this means software running on human-made machines beyond a research laboratory, e.g., on devices of common end-users, data in packets on public networks. The analogy to biologists studying wildlife in the jungle is thus rather coarse—the jungle in our security world is comprised of human artifacts, ignoring the extra complication of human users and their behavior.

Let us return now to operating systems and security. As a dominant OS, Unix and its configuration on individual machines play a major role in many security incidents involving large systems. (A trend worth noting here: ubiquitous cloud hosting for public facing services has reduced the proportion of servers administered by non-professionals.) Related to human artifacts, a main observation is that Unix could have turned out to be quite different—indeed, each of the main Unix branches and their own many versions (see sidebar: The Unix Menagerie) are evidence of this. Differences range from relatively major design alterations to minor design choices, functional extensions and commands, command syntax and options (often configuration-related), interfaces and new design features. This is demonstrated also by alternatives with comparable functionality, e.g., versions of the Windows operating system. As is well known, even single minor configuration changes can introduce new security vulnerabilities, or eliminate them. A main point here is that many pieces of operational security knowledge are specific to Unix variants and versions.

General vs Detailed Knowledge

To pursue our main question, now we also ask: If we aim to capture the (or a) body of knowledge in security, beyond generalized knowledge, to what extent should it include details of specific human-created artifacts—for example, versions of a Linux OS, or turning attention to configuration and operational details of major applications, security-relevant details specific to Apache or NGINX web servers? Given that the number of variations of such software artifacts is unbounded, a reasonable approach to learn and organize knowledge—in security and other fields—is to focus on the dominant class representatives (e.g., operating systems) plus others that uniquely illustrate important characteristics. In computer security, and systems areas of computer science in general, learning by concrete examples is also both common and well accepted.

Another common educational approach is to introduce general concepts, and bring them into focus through detailed examples on specific instances of systems. This leads to a mix of general concepts, approaches and principles, and well-chosen examples using representative architectures

The Unix Menagerie (sidebar)

The history of Unix operating systems has many twists. For context we give a simplified overview. For more details, see Tanenbaum [10], Garfinkel et al. [2], and online sources [11].

The story begins with the Multics operating system design dating to 1964, involving MIT, AT&T Bell Labs, and GE (later Honeywell). When AT&T left the project in 1969, its Multics team member Ken Thompson embarked on a much simpler cousin that would be called Unix. After the first version rapidly emerged, he was joined by colleague Dennis Ritchie, whose new C language was adopted to reimplement Unix circa 1973, aiding portability. A popular feature was a rich set of simple composable utility programs (*tools*) that performed single tasks well.

Despite AT&T being a regulated monopoly barred from marketing or supporting software products, interest in Unix swelled among university researchers and educators, aided by a 1973 SOSP presentation, and enhancements made by UC Berkeley researchers (Thompson's 1975 sabbatical there helped). They, like others, had obtained the source code from AT&T at modest cost, and grad student Bill Joy led efforts that improved and extended functionality.

UC Berkeley was allowed to distribute its so-called *Berkeley Software Distribution* (BSD) to users who purchased an AT&T Unix license. The upgrades were available at distribution cost as the BSD, sometimes called Berkeley Unix. DARPA provided funding, aiming to reduce costs in its own projects. BSD Unix had many popular features such as longer filenames, symbolic links, new utilities, and especially TCP/IP networking in BSD 4.2 (released August 1983).

In 1984 Bell Labs, upon being divested from AT&T, was allowed to market and support software, and refined AT&T System V Unix as a commercial product. System V gained market prominence, promoted as "standard" Unix while itself adopting many of the BSD enhancements. As a side effect of AT&T's trademark on UNIX, commercial versions of System V with a variety of BSD-inspired vendor modifications were sold under many disparate names, such as HP-UX (HP), AIX (IBM), IRIX (SGI), Ultrix (DEC). SunOS was a largely BSD-based commercial product released in 1983 by Sun Microsystems. Later, aiming to merge BSD and System V, Sun and AT&T jointly built Solaris, marketed as System V Release 4.

NetBSD, released in 1983, emerged from a Berkeley group aiming to escape AT&T's licensing terms (recall AT&T Unix was not open-source) by rewriting all AT&T-originated source code in BSD software. Differing priorities led to forks FreeBSD (from which DragonFly BSD forked in 2003) and OpenBSD (aiming to prioritize security and reliability).

Open-source Linux from Linus Torvalds appeared in 1991, with strong ties to Richard Stallman's GNU project and C compiler (gcc). Despite a monolithic kernel, Linux borrowed many ideas from Andrew Tanenbaum's Minix. Minix, released in 1987 and available for educational use (including source code), was a microkernel-based variant of Unix designed to be "small enough for students to understand" [10].

Apple's widely used macOS (a rebranding of the 2001-released Mac OS X), emerged as a combination of Apple's Mac OS 9, BSD elements, and the NeXTSTEP operating system (whose kernel is based on the CMU-origin Mach kernel).

and mechanisms in given instances of, e.g., operating systems. Typically, the volume of low-level details needed for expertise in particular areas is beyond that possible to cover in even a modest set of courses in a degree program. As in other fields, expert-level details are picked up by those who proceed into given subfields as specialists.

As an aside, the formal-sounding term *body of knowledge* (BoK) for security remains imprecise at this point. We use this term to mean what academics might view as consensus core topics—but these remain subject to opinion, and it remains unclear whether there is a consensus on an expected level of detail; both are subject to evolution [6].

The issue of level of detail, and general knowledge versus instance-specific details, raises the usual tensions and tradeoffs between applied and academic interests [9], practice versus theory, and training for immediate jobs versus education to prepare for future challenges across a career [8]. A more useful question to consider is perhaps: What subset of security knowledge should be prioritized for teaching in, say, university security programs? Here by the less formal term *security knowledge* we mean knowledge considered useful by a particular security community—again noting that this may vary substantially across communities.

Regarding what should comprise the core topics of a security BoK, this depends on the backgrounds and goals of those determining the list—they might favor, e.g., theory and formal methods more, or low-level technical details and real-world operations. Naturally, academics will draw up a different list, or weighting, than security practitioners, or business managers. One might begin with the view that the BoK is, de facto, what appears in published literature—research papers, books targeting a variety of audiences, trade articles—what has been captured in writing, and might be called *explicit knowledge*. However, aside from redundancy and volume issues, written knowledge often favors academic topics, and does not capture *tacit knowledge*; to use an analogy, one cannot successfully convey in writing how to excel in dance, or how to perfect a golf swing.

Unix Security Handbook

To go into more detail on the role played by Unix in practical systems security, consider the widely used 2003 book *Practical Unix & Internet Security* by Garfinkel, Spafford and Schwartz [2]. As suggested by its title, the book delivers any and all details that would be found in the OS chapter of an introductory security textbook, e.g., including filesystem security and access control, but also much more. The primary target audience is not end-users, nor system developers—for example, source code and system calls are not a focus.

Rather, this is a security handbook for Unix/Linux system administrators (sysadmins), with considerable focus on system configuration including for network protocols, software installation, operations, logging and audit tools, and handling security incidents. It goes into fine-grained advice for executing and automating these tasks, at command level detail (including shell commands and scripts). Command and configuration differences for major flavors of Unix are covered, including the main split between AT&T's System V and BSD Unix (see sidebar).

So, this handbook contains a wealth of valuable advice and expert knowledge—myriads of configuration and operational details that help system administrators secure real-world systems. Where should this content be placed on a "security knowledge" priority list? All of it is arguably important in practice, most is specific to particular OS versions, and some is generic and generalizable—but to be used in practice, the generic knowledge must be translated back into often obscure OS-specific or application-specific commands.

My technical heart leads me to rate the contents of this book as high priority, based on my subjective sense that the individual items are important, e.g., how to efficiently check that a system

has no world-writable temporary files, that an end-user's home directory is not in the search path for invoking an executable file, and that startup files are not easily modified by unauthorized parties.

On the other hand, the specific details for how to prevent "known" attack patterns differ across major operating systems and versions thereof, and all details obviously cannot be taught for all OS versions; details at this level can be difficult to cover even in highly specialized courses. Nonetheless much of this knowledge is critical for sysadmins managing large numbers of user accounts and machines hosting critical public services. Large subsets of this knowledge are acquired by those working daily at the system level in Unix environments—albeit often over many years. Thus, what security topics are taught in college or university programs boils down to a matter of prioritization and available time, subjective instructor choices and decisions by authors of textbooks or other course materials, ideally taking into account the target audience and what they are being prepared for. An entire body of knowledge cannot be taught to anyone, let alone everyone.

These issues may lead us to also ask: To what extent do typical undergraduate security courses cover (secure) system configuration? My sense, from a review of popular computer and network security textbooks, is that system configuration is a little-covered topic. More generally, system operations, administration and maintenance (including redundancy, backup and recovery, and information management) is perhaps more popular in information technology and management programs than in security or computer science courses. Whether or not this is a reasonable state of affairs is harder to say.

Sysadmin Handbook for Unix and Linux

As a second book example, and the leading book on Unix system administration, consider the 2017 fifth edition of *UNIX and Linux System Administration Handbook* by Nemeth at al. [5]. While only one of its 31 chapters has the word *security* in its title ("security" not being in the title of chapters on, e.g., access control or user management), numerous chapters address topics in which security is critically important. The book's main content is divided into four parts: basic administration, networking, storage, and operations. It naturally covers many of the same topics as *Practical Unix & Internet Security* above.

As expected, the Nemeth handbook (being both newer and longer) has extra or deeper coverage on topics such as virtualization, containers, cloud computing, and data center issues; and covers traditional sysadmin topics such as scripting and shell programming (compare to Kochan and Wood [4]), configuring mail, DNS and web servers, monitoring, and logging/auditing. Due to the popularity of Linux (vs BSD and other variants, with System V having fallen out of favor long ago), its focus on Linux, including in examples, is an excellent choice today for a wide audience.

We may now ask: Which of these Nemeth topics are "core" knowledge for security? A colleague has suggested that we have no strong sense of what comprises "core sysadmin skills" as they relate to security. I am inclined to agree. While the Nemeth book is not explicitly security-centered (in contrast to Garfinkel et al.), security is critically important in the domain of system administration. This reminds us of a general truth [1]: before one can contribute security expertise to a particular domain, one often must first become well-versed (if not an expert) in that domain also; first learn how something is supposed to work, then how it can be broken, then how to prevent or detect that.

If we proceed to ask: Where in a university computer science or security curriculum do we teach system administration, I believe the answer is often "Nowhere". Again, system configuration may be mentioned as important in courses, but is taught less often—arguably, despite weak configuration being a frequent root cause for security issues in computing systems (perhaps more than, e.g., inherent operating system weaknesses). Apparently for many, (secure) system administration is learned through apprenticeship, by trial and error in practice, or not at all. Sysadmin courses

are more common at technical colleges specializing in information technology than at universities, which traditionally focus more on *why* (and design) than *how* (and training). As security in practice requires both, the trick is to find the right mix, and where to teach which skillsets and abilities.

Back to human artifacts and security

We are now back to our main theme, asking what security-relevant lessons we should be teaching, and learning, about operating systems and their use. Across the menagerie of Unix systems, many low-level details—details that must be entered or set precisely for safely configured systems—differ depending on the Unix variant being used. And this is not to mention, e.g., Windows or mobile systems. Unlike chemists who may continue to use the same periodic table throughout their lives, our main frames of reference change regularly. On the other hand, professionals in clinical medicine might recognize some analogies with security practitioners.

This leads to revisiting the definition of security knowledge, our view of what its important elements are, asking to what extent we should teach details versus concepts and principles, and at the details level, what to prioritize. By consensus, concrete examples are invaluable in teaching and learning, to cement understanding, and as instances of general concepts. Also, as a colleague reminds me, we need not only car mechanics, but engine designers and safety testers. I expect there are many approaches, and that the diversity of approaches is a strength across our global community.

Unix is a dominant technology resulting from countless specific, often subjective design choices, many that profoundly impact system security. Most of these design details, large and small, could have turned out to be quite different in other versions of history—and remain subject to change, as software is a human artifact. The study of human-made artifacts is what much of practical security is about—understanding artifact-specific attacks, as well as generic methods underlying particular instances. This leaves open-ended questions as to what comprises security knowledge. Our goal is to motivate continually revisiting these questions, rather than pretending to have clear answers.

References

[1] T Aura. Why you shouldn't study security. *IEEE Secur. Priv.* 4(3): 74-76 (2006).

[2] S Garfinkel, G Spafford, A Schwartz. *Practical Unix and Internet Security (Third edition)*. O'Reilly, 2003.

[3] C Herley, PC van Oorschot. SoK: Science, Security, and the Elusive Goal of Security as a Scientific Pursuit. 2017 IEEE Symposium on Security and Privacy, San Jose, CA, pages 99-120.

[4] SG Kochan, P Wood. *Unix Shell Programming (Third edition)*. SAMS Publishing, 2003.

[5] E Nemeth, G Snyder, TR Hein, B Whaley, D Mackin. *UNIX and Linux System Administration Handbook (Fifth edition)*. Addison-Wesley, 2017.

[6] PC van Oorschot. Coevolution of Security's Body of Knowledge and Curricula. *IEEE Security & Privacy* 19(5):83-89, Sept-Oct 2021.

[7] HA Simon. *The Sciences of the Artificial (Third edition)*. MIT Press, 1996.

[8] EH Spafford and DL Burley. An interview with Gene Spafford on balancing breadth and depth in cybersecurity education. *ACM Inroads* 5(1):42–46, March 2014.

[9] DE Stokes. *Pasteur's quadrant: Basic science and technological innovation.* Brookings Institution Press, 1997.

[10] AS Tanenbaum. *Modern Operating Systems (Third edition)*. Pearson Prentice Hall, 2008.

[11] Wikipedia. List of Unix Systems. `https://en.wikipedia.org/wiki/List_of_Unix_systems`, accessed 28-Aug-2022.