Part I

Introduction and General Concepts

Chapter 1

An Introduction to System Administration

system administrator, n.:
one who, as a primary job function, manages computer and network systems on behalf of another, such as an employer or client.

Figure 1.1: System Administrators: well caffeinated and always happy to serve you.

1.1 What exactly does a System Administrator do?

Every semester I start my class with this simple question: "What exactly does a System Administrator do?" My audience consists of undergradate and graduate students of Computer Science or Computer Engineering; some of them have experience working as a System Administrator or IT support (either part-time or full-time), while others may maintain their own home network or otherwise perform some of the tasks commonly associated with the job of a System Administrator. Some may have no practical experience in this area, but all do have an idea of what the job entails.

As a result, the responses are varied, ranging from very specific ("Use `tcpdump` to troubleshoot network problems.") to the more general ("Maintain the network."), quickly illustrating that there is no one concise description of the professional duties of System Administrators; rather, we identify a number of tasks that are expected to be performed by people holding this title. The broad range of these tasks – especially when put in context of the size and goals of different organizations and their deployment footprint – makes obvious the fact that a System Administrator in one company may not do the same thing as one in a different company. In fact, even within a single organization we usually find that two people may well both be called "SysAdmin", but perform rather different duties.

So what *does* a System Administrator do? It seems that most people agree that this job has to do with computers in some capacity, but that really is already where the definitions start to become fuzzy: System Administrators are in charge of "servers" as well as of personal computers, desktop machines, laptops and, increasingly, mobile devices. With the evolution of computing coming full circle, we find that the migration from the mainframe computer to the personal and independent computing device to the networked server back to the central *Infrastructure as a Service* model inherent in today's Cloud Computing concepts places the System Administrator in the middle of it all.

So perhaps the main focus of a System Administrator's job is then really the central connection between the independent systems, the network itself? Hmmm, System Administrators surely are involved in the deployment and installation of the computers in a datacenter (or is that the task of specific Datacenter Technicians?) and connecting all the different components

certainly involves a lot of cables. But then, don't we have *Network* Administrators, or is that a more specialized subcategory of System Administrators?

System Administrators seem to spend as much time typing cryptic commands into dark terminal windows as they do running cables and labelling hardware, and while they may no longer shuffle punch cards, they frequently do write programs to help them complete their tasks. System Administrators are known to get woken up in the middle of the night when things go "bump", and as a result they are also known to have a fondness for caffeinated drinks. They are able to kickstart a generator, assess the required cooling power for their server room, use duct tape in creative and unexpected ways, assemble servers out of mysterious looking parts, and may end up handling a circular saw or other heavy machinery when their Leatherman multi-tool cannot complete the task.

System Administrators plan, budget and design networks and backup or storage systems, add and delete users (well, user *accounts*, anyway[1]), install and update software packages, draft policy documents, fight spam with one hand while rebuilding a corrupted revision control system with the other. They have access to all systems in the organization, may undergo retina- and fingerprint scanners to access "Mission Impossible"-style protected datacenters and spend countless hours in front of a multitude of computer screens, typing away on oddly shaped keyboards consuming not entirely healthy amounts of coffee and energy drinks.

Well... in some places, a System Administrator might do all of this. In others, there might be different, more specialized people for the various tasks: there might be datacenter technicians and "SiteOps", Network Administrators, System Programmers, System Architects, Operators, Service Engineers, Storage Engineers, Site Reliability Engineers, Virtual Operations, Infrastructure Architects... the number of different job titles for things that might otherwise fall into the more general "SysAdmin" category seems endless.

The various areas of expertise included in the day to day routine such as system design, infrastructure architecture, system fault diagnosis, hardware benchmarking, and others eventually require experience in a number of related fields; a lot of them involve more than just a little bit of programming experience and in some cases complex infrastructure tools based on solid

[1]There is a strong love/hate relationship between System Administrators and their users. Much as the SA may joke that they wish they could make users disappear, without users the systems they are in charge of might well hum along uninterrupted, but would ultimately be entirely useless.

software engineering practices need to be built. This illustrates that just as it is hard to clearly define where System Administration begins, we often can't quite put our finger on where it ends and another discipline becomes the primary job function.

Even if job descriptions and duties differ significantly, there are a lot of people who simply identify as *System Administrator* regardless of job title. We all have something in common: *we manage and maintain computer and network systems.*

A typical bug

At one place of employment we had a tool that would add users to a given host by gathering the login information for all the users, `ssh(1)` to each host and update `/etc/passwd` using the host's native tools (such as `useradd(8)`, etc.). Many a system administrator has written a similar tool, and for the most part, this program worked reasonably well. (We will discuss how to better manage user accounts across large numbers of machines later in this book.)

But all software fails eventually, and for one reason or another I had to debug the tool. Groveling through a few hundred lines of perl that had clearly been written by different people in different stages of their career, I came across a code block that included the following call:

```
chmod(0666, "/dev/null");
```

Asking around, it became clear that *somehow* some of the systems ended up in a state where `/dev/null` was unreadable or -writable by normal users, which leads to a number of complications. But nobody had been able to track down just how exactly this happened until one day we were lucky enough to witness the change in permissions, but the only activity by a user with sufficient privileges to make such a change was a `sudo(8)` invocation of `less(1)`.

`less(1)` happens to make use of a history file to remember search commands between invocations, much like your shell may use a history file (such as `~/.bash_history`). `less(1)` is aware of the fact that the history file might contain sensitive information, and it actively changes the permissions on that file to only allow read/write access to the owner. But

for the root user, we explicitly symlinked all common such history files to `/dev/null` to avoid accidental leaking of secrets to disk.

And therein lies the bug: `less(1)` was invoked by the super user, it would check the permissions on the file `/root/.lesshst`, follow the redirection to `/dev/null` find that they're not `0600` and call `chmod(2)`, yielding and unreadable/unwritable `/dev/null`. We were able to confirm this behaviour and identified a code change in later versions of `less(1)` that fixed this problem.

Understanding such failures and identifying their root causes requires a deep understanding of the operating system and its tools, but tracking down bugs like this[1] is just one of the many aspects of a System Administrator's daily routine.

1.2 The Profession of System Administration

Most professions have a fairly clear and formal definition as well as an obvious career path and background requirements. In fact, for a person to take on a particular line of work, it frequently is a requirement to undergo a specific, regulated education, to perform a certain amount of practical training (such as an apprenticeship), to subscribe to specific professional ethics and guidelines, and eventually to obtain a license to practice the given profession. Doctors, scientists and lawyers all have well-defined professions, as do engineers of most disciplines.[2]

System Administration, on the other hand, does not (yet) have any of the criteria given above: there is no specific, regulated training, and as we have seen in Section 1.1 not even agreement on what precisely the job definition ought to be. The The USENIX Special Interest Group for Sysadmins (LISA) attempts to fill this void to some degree by providing for an informal but professional community. Still, due to the nature of the changing environment, any attempt to create a concise job description invariably fails by resorting to

[2]Software Engineering, another young discipline related to System Administration and technically a specialized branch of Computer Engineering, differs in that it does *not* have strict licensing regulations. Anybody can call themselves a "Software Engineer", just as anybody can call themselves a "System Administrator".

very general terms. Consider the following description given by the Buerau of Labor Statistics[2]:

> *Systems administrators* design, install, and support an organization's computer systems. They are responsible for LANs, WANs, network segments, and Internet and intranet systems. They work in a variety of environments, including large corporations, small businesses, and government organizations. They install and maintain network hardware and software, analyze problems, and monitor networks to ensure their availability to users. These workers gather data to evaluate a system's performance, identify user needs, and determine system and network requirements.

While this captures *in general* the common job duties of a System Administrator, it does not answer the question we asked initially: Just what *exactly* does a System Administrator do?

In order to better grasp the generic description, I've found it useful to break down the title into its components, *System* and *Administrator*. What, then, is a "System"? Any dictionary will give you a reasonable definition along the lines of:

> *system, n*: A group of interacting, interrelated, or interdependent elements that together form a complex whole.

In this context we will focus on *computer-human systems*[3] consisting of any number of computing devices, the network(s) connecting them, the users utilizing these resources, and of course the impact of the goals of the organization running such systems.

What about "Administrator"? Merriam-Webster's Collegiate Dictionary defines the term "to administer" as:

> *administer, v*: to manage or supervise the execution, use, or conduct of

The profession of System Administration should therefore be described as one in which practitioners "manage or supervise the execution, use, or conduct of a group of interacting, interrelated, or interdependent computer-human elements", leaving us, in terms of specificity, about where we began! We find slightly more formal definitions of actual job titles, desired skills

and a rather useful distinction between "small uniform", "complex", and "large and complex" sites in the LISA booklet "Job Descriptions for System Administrators"[5], but in searching for a suitably clear definition of the profession, we realize that the formal education as a System Administrator is almost completely lacking.

Job descriptions and postings normally list a degree in Computer Science or Computer Engineering as "desirable"; more frequently you will find a requirement expressed as "BS, MS or PhD in Computer Science or equivalent work experience". Computer Scientists will wonder just how, exactly, "work experience" can be mapped, treated as equivalent to their formal education – all too often, "Computer Science" in this context is equated with "knows how to program, understands TCP/IP".

As invaluable as actual hands-on experience is, it is no substitute for a formal degree in the area of Computer Science. (Nor is having such a degree a substitute for practical experience.) Rare is the candidate who can fully understand the average and worst-case runtime of his or her programs and scripts without having learned Big-O Notation; rare the system programmer who completely grasps the elegance of modularization provided by, say, function pointers without having learned the λ-calculus. Becoming a proficient System Administrator without this background is certainly possible, but it makes it significantly harder to advance beyond a certain level.

Be that as it may, the focus on many years of real-world experience as the primary qualification is both cause and effect. In the past, there really existed *no* formal training whatever: technically skilled people found themselves in the position of being the go-to guy to fix the computers, to make things work. Perhaps they ended up working with a more senior administrator in a sort of informal apprenticeship, but for the most part their skills were learned by experience, through trial by fire. The lack of any formal program in this area necessitates and then further encourages self-learning, and many things simply can only really be understood by experience. It's a cliché, a platitude that you learn the most from your worst mistakes – it is also entirely true.

As a result, many of the senior people in hiring positions do not have an academic background and are hesitant to require what was not available then to fill current openings. System Administrators are also still frequently treated as – and at times view themselves as fulfilling – a "janitorial" role. If no industry standard exists by which to measure a candidate's accomplishments in the field, how can we require formal education for a job that eludes definition in the first place?

Therein, however, lies a fallacy. Practical experience and formal training do not make up for one another; they complement each other. It *is* true that we learn the most from our worst mistakes (and thus need a chance to make them); it is *also* true that we frequently are only *able* to learn due to a deeper understanding of the problem.

The profession of System Administration is incredibly interesting precisely *because* it eschews a fixed definition, a formal governing body, a static career path, or a licensing exam. It attracts and invites interesting people from all sorts of backgrounds; it allows phenomenal potential for job growth and it is – must be, due to the advances in the industry – fast-paced and continually evolving. But it has become more mature: professional organizations like LISA have contributed significantly to the status of the profession by formalizing the requirements and duties as well as providing a Code of Ethics[3], and over the last few years more and more universities have started to offer classes and degree programs in this area.

System Administrators may not need to be licensed by a central board anytime soon – and people will continue to argue about whether or not that is desirable or even possible in this field – but what a few years ago still was best described as "a job" certainly has grown up to become a career, a craft, a *profession.*

1.3 System Administration Education

System Administration is rarely taught as an academic discipline in large part due to its perceived "blue-collar" status: from the top of the scholarly ivory tower, it must be hard to distinguish where an entry-level job like "Tech Support" or the "Help Desk" ends and where a profession demanding a significant background in Computer Science and Engineering begins.

System Administrators are highly skilled information technology specialists shouldering significant responsibility in any organization. As we discussed in the previous sections, the variety of job descriptions and differences in what a System Administrator actually may be doing make it difficult to provide simple step-by-step instructions on how to enter this field. As a

[3]For a more detailed discussion of both the (lack of a) definition of the profession and the ethical obligations we face, please refer to [15].

result, System Administration has long been a profession that is learned primarily by experience, where people grow into a position in order to fulfill the requirements of an organization rather than follow a career path well-defined by courses, degrees, and meaningful certifications.

The industry has responded by producing a large number of practical certification exams that hope to attest to the student's proficiency in the subject matter. Practically, most of these certifications appear to primarily test a person's ability to memorize specific commands, further reinforcing the notion that System Administration can be reduced to empirical knowledge, making practical experience indeed equivalent, if not superior to one of those scholastic degrees.

But certification that one remembers a specific vendor's unique configuration file syntax does not imply actual *learning* has taken place; holding a one-week "boot camp" – the name itself is revealing of it's educational intention – to drill enough information into the participants' heads such that they pass an exam at the end of the week does not guarantee long-term retention of that knowledge. Furthermore, there is no oversight of the topics taught in such classes, no review of suitability or correctness.

System Administrators who excel at their jobs do so not because they have accumulated arcane tribal knowledge about obscure pieces of hardware and odd software systems (useful as that is), but because they understand fundamental underlying principles and combine that knowledge with their concentrated experience.

System Administration – its principles, fundamentals and practice – should be *taught*. Since you are reading this book, chances are that you are currently taking a class in System Administration as a student or perhaps you are teaching such a class. But we cannot teach an entire profession in a single class or even using a few select courses. It is necessary to develop academic degree granting programs that comprehensively prepare students for the varied and wide requirements imposed on them when they enter the industry. Such programs should be combined with extensive real-world and hands-on experience wherever possible, perhaps by including internships, cooperative education and apprenticeships provided by industry leaders. As students you are entitled to practical and useful exercises; as educators we have an obligation to create these and make available the required resources.

We need classes that combine the academic mission of fostering independent research with the factual requirements posed by the positions offered in the marketplace. We need instructors who have years of experience in var-

ied environments, who understand the practical restrictions that may make the ideal theoretical solution utopian but who are consciously aware of these boundaries and able to teach their students the awareness of them. At the same time, we need scholars willing to further this profession through research and build a theoretical body of knowledge to become the foundation of future programs.

1.3.1 Challenges in System Administration Education

Formal classes in System Administration as part of a Computer Science or Engineering curriculum are still uncommon, but in the past few years more and more institutions have recognized the industry's need for academic courses that adequately prepare students for the multitude of responsibilities within this field and have started to offer such classes (and in some cases complete degree programs). But the history of this profession brings with it a number of unique challenges when it comes to formally teaching its principles and practices.

To really understand and appreciate some of the most general aspects of System Administration, you need to be exposed to actual running systems. Practical experience is so integral to this profession that it cannot be separated from the theoretical knowledge and postponed until the student enters his or her first job or apprenticeship. But therein lies a significant hurdle to traditional teaching methods: students need to administer a system, to have superuser access, to have a chance to configure a system for a specific service, and to make the kind of spectacular mistakes that experienced System Administrators value (if only in hindsight).

This normally conflicts with the requirements of the IT department at your university: students would require access to a number of different OS when the school's system administrators strive for a certain level of homogeneity. In order to understand OS installation concepts, file system tuning, and other low-level principles, students need to perform these tasks themselves. Learning to debug network connectivity issues or being able to actually see the payload of captured network traffic requires access to raw sockets, which the staff responsible for the security of the campus network would certainly rather not provide.

After years of struggling to give all students equal access to the resources required for the diverse assignments and frequently having to change the assignments to be able to make use of available resources, I finally decided to

do what many companies do: I outsourced the resource provision problem into "the Cloud" – specifically, to Amazon's EC2[4].[6] Similar results can be achieved by granting students access to a dedicated laboratory environment or by providing another on-demand infrastructure via virtualization of the local compute resources. If you are a student, it is worth your time to investigate all possible resources available to you; if you are an instructor, consult with your IT department as to the availability (or perhaps development) of such systems.

But teaching System Administration should also combine the practical elements noted above with topics from other academic fields. As an area of specialization, System Administration fits in well within the current Computer Science curricula, since the profession's requirements draw heavily from many of the same subjects. Shared requirements include Operating Systems, Networking, Database Systems, Distributed Systems, Cryptography and of course Software Engineering; at the same time, System Administrators benefit from deep insights in Computer Architecture and Systems Engineering as well as Project Management and of course "the 5 elements of administration" (Planning, Organizing, Command (directing people), Coordination, Control, also listed as Budgeting)[7], usually covered in Business Administration classes.

To advance the practice of System Administration, the two existing professional organizations LISA and the League of Professional System Administrators (LOPSA) are already promoting best practices and important communities. Still, we are missing strong proponents of System Administration as an academic discipline[5] But there are more and more of us who believe that it's time for the profession to be recognized as both requiring a theoretical background in addition to extensive practical experience and benefitting from the opportunities of dedicated research opportunities and the development of universally accepted degree granting requirements. Some even write books to further this goal...

[4]In this book, I will frequently make reference to or cite examples from this experience. At the same time, I also will suggest exercises that cannot be done in a cloud environment and that may require access to dedicated hardware.

[5]Mark Burgess, previously of the Oslo University College in Norway, one of the only universities offering a Master's degree in Network and System Administration, deserves explicit mention as one of the few exceptions.

1.4 The Three Pillars of Exceptional System Design

More than just maintain computers and infrastructure components, System Administrators control entire systems. With more experience and seniority, they are ultimately in charge of building these systems, of designing the infrastructure and its components so that the job title frequently morphs into that of a Systems or Network "Architect". It seems that people can't help but draw parallels to the so-called "real" world, where engineers and architects have well defined roles (and status). This does not come as a surprise; as Brooks[8] illustrates masterfully, the parallels between designing and building a house, a computer or a complex piece of software are plentiful.

Billy Hollis points out[9] that the title of the "Software Architect" is a misleading analogy, that the duties traditionally performed by people holding this title are much more akin to those of a structural engineer than those of a (traditional) architect. The role of the System Administrator is likewise frequently comparable to that of the structural engineer in the mechanical world, but the advanced senior System Administrator – who ultimately steps into the role of planning, designing and overseeing the construction of complex systems – might better known as "Systems Architect". Structural systems engineers are then needed to aide in the final design and implementation of the system in question.

Conceiving of and designing systems as complex as those System Administrators are in charge of, with their myriad pieces, the unpredictable human element and the rather unfortunate tendency of computers to do precisely what they were instructed to do rather than what we intended them to do requires expert in-depth knowledge in a broad range of topics. Exceptional System Administrators are generalists – they have been trained in a wide area of topics, deepened their understanding of several and perhaps have become subject matter experts in a few, but ultimately back into a role that allows them to apply their ability to connect the dots, to see the big picture and the connections between and effects of multiple systems on each other. They are well-equipped to model new components of an infrastructure or to redesign a system from scratch to meet the new requirements.

For a system to be suitable both for the immediate requirements today and for a few years down the road, when the current needs have changed, it must embody two basic principles: Scalability and Security. Neither of

these can be added to a system after it has been built: trying to apply "security" after the system interfaces have been defined yields restrictions, limitations; trying to make a system with inherent limitations perform under circumstances it was not designed for yields hacks and workarounds – the end result frequently resembles a fragile house of cards more than a solid reliable structure.

The third fundamental feature of expertly designed systems, Simplicity, is simultaneously obvious and counter-intuitive. Simplicity underlies both scalability and security, since reduced complexity implies better defined interfaces, minimized ambiguity in communications or data handling and increased flexibility. As with the other two core aspects, simplicity cannot be added after the fact; it must be inherent in the architecture. Simplicity is the enabler of both scalability and security.

An exceptional system exhibits inherent structural integrity (another wonderful analogy to the world of bricks and mortar), and this integrity is provided by these three pillars. Our focus on these components may initially seem arbitrary: meeting requirements across different teams with different priorities has long been the bane of many a program manager due to each team focusing on different qualities, of which we could have picked any number. However, upon more detailed analysis and with some years of experience we have time and again found Simplicity, Security and Scalability to be the core qualities enabling a harmonious development- and deployment process.

Throughout this book, we will analyze the systems we discuss with special focus on these crucial criteria. You will find that we will frequently talk in broad abstracts or surprising detail about the implications of our design decision.

1.4.1 Scalability

In recent years, the word "scalability" has become one of the defining requirements for virtually all technical solutions. Providers of infrastructure services and products throw it around in an attempt to impress their customers by how much load their systems can handle. It typically seems near synonymous with "high performance", the ability to handle large amounts of data, traffic, connections, and the like. This may be misleading, as people might be inclined to believe that such capabilities come at no additional (or incremental or proportional) cost. In fact, scalability does not relate to the costs of the running system, but to its architecture. A scalable system may

well require additional resources and throwing money at a problem actually *is* a perfectly viable solution in many cases. However, a scalable architecture is one that does not require a refactoring of the whole system, a restructuring of *how* data is handled based on changing circumstances. Instead, it abstracts data from process flow and simply adapts.

System Administrators frequently suffer from Goldilocks Syndrome, striving to provide a solution that is *just right*; wasting resources is just as anathema to their definition of a correct solution as not providing adaequate capabilities. Therefore, we tend to focus not only on the ability to scale *up* – to accommodate larger amounts of data or more requests per minute, for example – but also to scale *down*, to free up unused resources based on the demands. As such, our definition of "scalability" leans more towards the overall flexible nature of a scalable system, its ability to adapt to changing requirements at run time; a term frequently associated with Cloud Computing, "elasticity", perhaps more aptly describes this feature, yet it fails to quite capture as well the sense of meeting extreme requirements. We shall continue to use the term "scalability" for that reason.

While many systems are *flexible*, few are able to handle input or requirements an order of magnitude different from those initially conceived of when the system was designed. In order to accomodate such changes, systems are said to either scale *vertically* – that is, one system is able to handle the added load, possibly by addition of certain resources (network bandwidth, CPU power, memory, disk space, ...) – or to scale *horizontally* – that is, a single system is replicated and demands are spread across them evenly. Either approach has certain implications on the overall architecture. For horizontal scaling to be beneficial, for example, the problem space needs to be such that a distribution is both possible and algorithmically feasible. However, this adds communication overhead, as interfaces and data flow becomes more complex.

Whether by vertical or horizontal means, a scalable system is one that readily adapts to changing demand. The designer may not know what will be required of it in the future, but will make choices that permit the system to grow and shrink without bumping into arbitrary limits.

We will take a closer look at the implications of this idea on the software development practices and overall Unix Philosophy in Chapter 2; for the time being, let us consider the Unix tradition of simple tools operating on streams of text as a wonderful example of a clear interface definition. Anybody de-

veloping a tool that accepts input only from a file restricts the flexibility of the tool; this frequently goes hand in hand with an implicit limitation on the amount of data that can be handled (think maximum file size, buffers frequently allocated to read in the entire file, the ability to `seek(2)` on the file handle, ...). By choosing text rather than a binary format, any future use of the output is not limited by the original author's imagination of what future users might wish to accomplish. Given sufficiently large or, rather, diverse datasets, building more complex systems that perform equally well under heavy load using complicated, yet limited interfaces so frequently developed by major software companies easily becomes a frustrating exercise in determining these boundaries.

But scalability is not only of concern when tools are developed; as we design infrastructure components, we need to be acutely aware of the interfaces between them and what kind of data can flow back and forth. Frequently we have no control over the input (both type and amount), so our systems need to be fault tolerant of many unenvisioned circumstances. Even though somewhat counter-intuitive at first, I argue throughout this book that a robust system will remain resilient to overall failure by being comprised of infrastructure components that themselves may fail quickly (and explicitly) and that such well-defined behaviour underlies true scalability.

Hindsight being 20/20, scalability related issues often can be traced back to a lack of imagination or confidence in the system. What if our initial use case increases not by a factor of two or three, but hundredfold? Suppose our system is still in use in five years – will average input be likely to remain the same? In our discussions, exercises and problems we will encourage students to consider how a system performs if circumstances and inputs change by an order of magnitude. The ability to anticipate extraordinary change in requirements requires some practice and experience by the System Administrators in charge of the infrastructure; reacting in well-defined ways to such change is one of the core features of a reliable system, and the principles that make a system scalable – fault tolerance, data abstraction, clearly defined interfaces – must be applied at all stages. We will reinforce these axioms throughout all chapters in this book.

1.4.2 Security

All too frequently, the software or information technology industry treats system security as an afterthought, as something that can be added to the

final product once it has met all the other functional requirements, after the user interface has been determined and after all code has been written. It is then not surprising that the old adage that security and usability are directly and inversely related seems to hold true. Nevertheless, I would argue that the very problem statement – "the more secure you make something, the less usable it becomes"[6] – reflects a fundamental misunderstanding, as it implies that usability is present first and security "added" afterwards.

This approach suggests that the only way to reduce risks is to *take away* functionality; but any time you do that or otherwise restrict the users, they will either stop using the product/system altogether or come up with a creative solution that works around your newly imposed restrictions. To be effective, security needs to be built into the system from the design phase on. That is, rather than starting out with a solution that provides the desired functionality and then attempting to figure out how to get it to a secure state, we should instead begin with a secure albeit restricted state and then slowly add functionality – without compromising safety – until the desired capabilities are available. That is, we need to view security as an enabling factor present at the design's conception.

Much like software, a system infrastructure is usually developed with much looser an idea of what one wishes to accomplish than people are willing to admit. In order to achieve maximum functionality and the widest possible use, people tend to design interfaces with few restrictions. Often it is the role of the security-minded engineers to ask questions that require the developers or designers to revisit their requirements; proactive security imposes restrictions on what the system is able to do. That is a *good* thing! General-purpose systems are much harder to design than special-purpose sytems ("Constraints are friends"[8] whitelists provide significantly better and more reliable security than blacklists.

Any parent can tell you that it is nearly impossible to take away a toy from a child, even if the toy has not been used in months or years. The very idea of something being taken away seems to trigger in people (of all ages) a "but I might need it some day" response. Giving users access to a resource (network access, disk space, physical access, CPU power, ...) is trivial – restricting the use once access has been granted is near impossible; the genie cannot be put back into the bottle. It is therefore imperative to understand

[6]We will elaborate on the somewhat surprisingly accurate corrolary that "The more secure you make something, the less secure it becomes."[10] detail in a later chapter.

precisely what level of access to what resources you *actually* need and not build a system for the most widest possible use. Well defined – restricted – small components can still provide the flexibility to build scalable systems, but each component needs to be designed from the beginning with security in mind. Just as a chain is only as strong as its weakest link, an infrastructure is only as secure as its most open or exposed component.

Rare is the software developer who dismisses "security" as of minor importance, yet this mythical property everybody pays at least lip service to has an incredibly wide definition and may mean different things in different contexts. Generally speaking, we require *risk management* (we discuss this in more detail in a later chapter); more specifically we will look at a number of computer-human system specific aspects of security such as (but not limited to):

- Cryptography, and the three main features that allow it to help mitigate risks:
  - Secrecy or Confidentiality
  - Accuracy or Integrity
  - Authenticity
- Physical Security
- Service Availability
- Service Design
- Social Engineering and other aspects of human nature
- Trust

Throughout this book, we will allude to these components wherever appropriate, and we encourage students to come up with possible threat scenarios as a regular exercise. Establishing and fostering a proactively defensive, some may say "paranoid", mindset is an important element of a System Administrator's development. Knowing that we cannot provide an all-encompassing mantle of security after the system has been designed (much less deployed!), we strive to integrate it in all stages, leading it to bolster our infrastructure as an inherent feature.

1.4.3 Simplicity

As our discussion of scalability and security suggested, the practical application of these principles yields a reduction of interfaces, end-points, use cases and overall variance. In other words, scalable and secure systems are less *complex* and – it is worth drawing this distinction explicitly – much less *complicated*. A complex system may be well-organized and exhibit a clear, logical structure yet require subtle or intricate connections or components. Complicated systems, on the other hand, are irregular, unpredictable or difficult to follow.

"Complexity is the enemy." This quote has been repeated endlessly in so many contexts that it can almost be assumed common knowledge amongst software engineers, cryptographers and other security experts[11]. But while many people in the information technology industry may agree on this in principle, I have found that, like so many aphorisms, the translation into the real world – the actual application of its consequences – trails far behind the good intentions.

In the world of System Administration reducing complexity is particularly difficult. As we discussed in Sections 1.1 and 1.2, managing large computer networks has *inherent* complexity: multiple components must interface with each other in many ways to help people – simultaneously the origin of true entropy as far as computers are concerned and dangerously predictable – accomplish their individual and collective goals. But it is important to differentiate between *required* complexity and *accidental* complexity[12]. We strive to reduce overall complexity by building ultimately intricate systems out of smaller, simpler components.

To repeat an almost ubiquitous prime example of how simplicity enables astounding flexibility and may allow you to build complex systems, we will time and again draw the analogy to the concept of toy building blocks: themselves simple, nearly unbreakable and available in a reasonably limited number of shapes and sizes you can build just about anything with them.

Like these little blocks, the Unix operating system builds on the philosophy of simple tools that "do one thing and do it well", that work together, commonly by being connected via the ubiquitous pipe, operating on text streams[13]. We aim for solutions that exhibit comparable elegance. "Simple" does not, it should be noted, mean "trivial" or "easy": it is so much easier to *add* features, to increase the output, to justify additional input than it is to reduce a component to its bare necessities. As Antoine de Saint

Exupéry, observed[14]:

> *Perfection is reached not when there's nothing left to add, but when there's nothing left to remove.*

But how do we design a simple infrastructure? The simple answer is: we can't. The end result will always be complex. But the systems we create will be more fault tolerant, more performant, more flexible – in short: better – if we remember that our building blocks need to resemble Lego blocks more than swiss army knifes. In other words: we design *for* simplicity. We keep interfaces internally consistent and logical. We use simplicity as a scalpel to cut away the clutter.

The ability to resist the temptation to "enhance" your architecture or software tool to implement additional features that, if at all necessary, might better be provided by a separate component, usually only comes with strong will power and years of experience. At the same time, simplicity is found to underly all of the exemplary technologies you are likely to encounter and some of which we will present in the coming chapters. Sometimes it takes a second look to recognize and appreciate the genius of simplicity; hopefully this book will help sharpen your awareness of this trait.

1.5 The Future of System Administration

The System Adminstrator's job definition is diverse, and so is the rather significant change the profession has undergone over the years. Gone are the days of shuffling punch cards, of cycling magentic tapes, of connecting your dumb terminals using RS-232 serial connections, of stringing ribbon cables inside large computer cases... or are they?

Some things haven't really changed all that much. Technologies come and go, but certain principles have remained the same. Using fibre-optic cables rather than twisted-pair, using Infiniband or Fibre Channel instead of Ethernet or parallel SCSI, or using wireless networks instead of physically connected systems does not fundamentally change the day-to-day operations.

Virtualization and Cloud Computing may seem to limit the need of an organization to hire their own System Administrators, but I believe that rather than a threat to the profession these technologies are simply one of the ways in which we make progress as information technology specialists. A chapter towards the end of the book discusses the future direction of

System Administration and revisits these topics as well as some of the latest trends in how System Administration intertwines with other disciplines – the terms "DevOps" and "Agile Infrastructure" (or "Agile Operations") deserve particular attention in that context.

A few years ago, the Personal Computer signified a paradigm change away from the central Mainframe computer accessed remotely by users towards a distributed storage and compute model. System Administrators found themselves supporting more and more individually powerful connected devices, trying to centralize certain resources and services such as shared filespace or regular backups, for example. In the recent past, the move towards software and data storage services "in the cloud" has brought the evolution full circle: once again, we are challenged to provide reliable central services that are accessed by small, though increasingly mobile, devices.

The separation of *systems* from *networks*, if it ever really existed, is disappearing. Standalone systems separated from the Internet are rare; systems operating completely on their own without *any* network connectivity have become largely impractical. Does this change the job of the System Administrator? Does it make things more or less interesting, difficult, challenging, exciting, different? Will we see another shift back towards a model where data storage and processing happens again on the edge nodes of the network?

We don't know. But we do know this: whatever happens, System Administrators will be needed. Perhaps they will carry a different title and perform different practical tasks, but the concept of the profession will remain. Organizations small and large will need somebody with the knowledge and experience to analyze, troubleshoot and design new infrastructures; somebody who builds and maintains scalable and secure systems; somebody with an appreciation of the simple things in life.

> *Plus ça change, plus c'est la même chose.* – Jean-Baptiste Alphonse Karr

Figure 1.2: An IBM 704 mainframe. With cloud computing coming full circle, we already live in the future!

Problems and Exercises

Problems

1. Create a course notebook (electronic or otherwise). In it, write down your notes about each chapter, add any links to additional information, noteworthy insights, etc. After each chapter, write down lessons learned. Differentiate between those directly applicable to you and those you consider worthwhile reviewing or investigating in the future.

2. Create a folder for your course work as a central place for all your documentation. Whenever you go through practical exercises, make sure to write down how you solved a specific problem. Your documentation will later on help you review the lessons you learned and can be used as a practical how-to guide, so make sure to verify all steps before you write them down.

3. Ask a System Administrator (for example: in your university, of an open source project you participate in, at work) to describe their daily routine. Compare notes with others.

4. Consider the current most popular websites and Internet businesses or organizations. Try to find out details about their infrastructure and how it is maintained.

5. Research available classes and degree granting programs in the field of System Administration. Compare their contents to those of industry standard certification courses.

6. Research professional organizations related to System Administration (such as LISA and LOPSA).

   (a) Review their mission and content and consider joining their mailing lists. They provide a great way to keep up to date with real-world experiences and itemlems.

   (b) How do these organizations compare to the ACM, IEEE or ISOC?

7. Research common Internet standard bodies and practices. What is an RFC? What do the IETF, IANA and ICANN do?

8. Consider the systems you have access to: what are their primary functions, for what goals were they designed? Suppose they grew by an order of magnitude – what itemlems would you foresee?

9. Consider the systems you have access to: what kind and how is access granted? What kind of security itemlems can you imagine?

10. Research the terms "DevOps" and "SRE"SRE. In how far do the practices it represents change the typical job description of a System Administrator?

Exercises

1. Practice basic system tasks in your virtualized environment by creating different OS instances and run them. Once running, log in on the virtual host and run the required commands to:

   (a) display basic system information (`uname`, `ifconfig`, `netstat`, ...)

   (b) display the partition table

   (c) display the currently mounted file systems

   (d) display available disk space

2. Repeat Exercise 1 for a different Unix flavor than what you are used to. For example, if you used a Linux instance for Exercise 1, repeat it using OpenSolaris or FreeBSD.

3. Determine the Unix commands you execute most frequently (for example via analysis of your shell's history file). Analyze the top three commands for their complexity and interfaces.

4. Analyze your interactive shell usage.

   (a) Change your login shell for at least a week to one that you are not accustomed to. Make note of the changes in behaviour.

   (b) Disable tab completion and any aliases you have defined in your login shell for at least a week. How does this change your workflow?

   (c) Review the detailed documentation of your login shell's `builtin`s. Practice the use of those that you are not familiar with.

Bibliography

[1] "less bug, more /dev/null", Jan Schaumann, on the Internet at `http://is.gd/NpihsH` (visited November 29, 2015)

[2] Bureau of Labor Statistics, U.S. Department of Labor, *Occupational Outlook Handbook, 2010-11 Edition, Computer Network, Systems, and Database Administrators*, on the Internet at `http://www.bls.gov/oco/ocos305.htm` (visited December 10, 2011)

[3] Mark Burgess, *Analytical Network and System Administration: Managing Human-Computer Systems*, Wiley & Sons, 2004

[4] *Merriam-Webster's Collegiate Dictionary*, 10th Edition, Merriam-Webster

[5] Tina Darmohray, *Short Topics in System Administration: Job Descriptions for System Administrators*, 3d Edition, USENIX Association, Berkeley, CA, 2010

[6] Jan Schaumann, "Teaching System Administration in the Cloud" in *;login: The USENIX Magazine*, October 2010, Volume 35, Number 5

[7] *Fayol's Elements of Management* on the Internet at `https://en.wikipedia.org/wiki/Fayolism` (visited December 17, 2011)

[8] Frederick P. Brooks, Jr., *The Design of Design*, Addison-Wesley Professional, 2010

[9] *Billy Hollis Still Builds Apps*, transcript of an interview with Billy Hollis, ".NET Rocks", on the Internet at `http://is.gd/Suk2hr` (visited December 23, 2011)

[10] Donald A. Norman, "THE WAY I SEE IT: When security gets in the way", in *Interactions*, November 2009, Volume 16, Issue 6, ACM, New York, NY

[11] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno, *Cryptography Engineering*, John Wiley & Sons, March 2010

[12] Frederick P. Brooks, Kr., *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*, Addison-Wesley Professional, Anniversary edition (August 12, 1995)

[13] M. D. McIlroy, E. N. Pinson, and B. A. Tague *Unix Time-Sharing System Forward*, The Bell System Technical Journal. Bell Laboratories, 1978

[14] Antoine de Saint-Exupéry, *Wind, Sand and Stars*, Harcourt, 2002

[15] Jan Schaumann, *Primum non nocere - Ethical Obligations in Internet Operations*, VelocityConf NYC 2015; on the Internet at `https://www.netmeister.org/blog/primum-non-nocere.html` (visited January 16th, 2017)