Chapter 10. Clouds and Virtualization

> She had never driven through the clouds. It was an adventure that always she had longed to experience. The wind was strong and it was with difficulty that she maneuvered the craft from the hangar without accident, but once away it raced swiftly out above the twin cities. The buffeting winds caught and tossed it, and the girl laughed aloud in sheer joy of the resultant thrills. She handled the little ship like a veteran, though few veterans would have faced the menace of such a storm in so light a craft. Swiftly she rose toward the clouds, racing with the scudding streamers of the storm-swept fragments, and a moment later she was swallowed by the dense masses billowing above.
>
> *The Chessmen of Mars*
>
> —EDGAR RICE BURROUGHS

10.1 Distribution and Isolation

Two seemingly disparate technologies have gained increasing attention in recent years, cloud computing and *virtual machines (VMs)*. The only way to improve the buzzword level of a virtualized, cloud-enabled system is to add that other au courant word, "ecosystem," to it. Though the mechanisms are not precisely new—VM technology dates back to the 1960s [Meyer and Seawright 1970], and those of us of a certain technical age have to squint to differentiate "the cloud" from the time-sharing service bureaus of the same era—their importance has increased dramatically in recent years. Most of the attention has been on the economic and functionality impact of cloud services, albeit with much handwringing about its supposed insecurity.

Virtualization, on the other hand, has been hailed as the solution to security problems, ranging from sandboxing desktop apps to protection in the cloud. Even virus scanning, it is said, can be improved by VM architectures.

From a broader perspective, the two can be seen as complementary. The cloud seemingly creates new security problems, while virtualization solves them. Is this an accurate summary? Not quite. Each has a sphere of action that the other doesn't impinge on; in those places, the two are completely independent. Furthermore, even where they do interact the risks and benefits are not one-sided; cloud services have security advantages even without virtualization, and VMs can have their own set of risks.

In most situations, cloud computing is based on VM technology; I'll therefore discuss virtualization first.

10.2 Virtual Machines

The basic concept of virtualization is simple: Take an ordinary, off-the-shelf operating system, and, instead of running it in privileged mode on real hardware, run it as a user program in virtual address space provided by the *hypervisor*. When the *guest OS* attempts to execute a privileged instruction, it instead traps to the hypervisor, which examines the intended instruction and emulates it. Thus, if the virtualized OS is trying to write a block of memory to a disk drive, the hypervisor will take that block and schedule it to be written somewhere appropriate: a physical drive dedicated to that guest, a file taking the place of a disk drive, or even a network connection to a storage server. When the operation is complete, the hypervisor simulates an interrupt to the guest machine. Modern machines have special architectures and instructions to make this more efficient, but the basic concept is still the same.

As well as emulating privileged instructions, the hypervisor creates and destroys VMs, allocates and manages resources of the underlying hardware (e.g., disk, RAM, CPU time), separates different VMs from each other, and so on. It has an interface (sometimes command line, sometimes graphical) to let users control such actions, and to do things like press the virtual reset button. There are also *hypervisor calls* (*hypercalls*) that allow VMs to make explicit requests; these might be for things like copy and paste between the guest OS and the underlying system.

Consider this in light of the virtual instruction set and effective target environment concepts introduced in Chapter 4. The virtual instruction set presented to the guest OS includes all of the real hardware instructions, emulated if necessary, as well as the facilities made available by hypercalls. The effective target environment is the set of physical or virtual resources allocated to it by the hypervisor. If this sounds familiar, it should; it is quite clear that a hypervisor *is* an operating system, and can even have its own privilege escalation vulnerabilities (see, for example, [CERT 2012]). (Although today's VMs generally consist of a few extra kernel and user-level modules on a familiar OS such as Windows or Linux, it doesn't have to be done that way. In fact, the very first VM hypervisor, CP-67 [Meyer and Seawright 1970], was extremely specialized and provided none of the facilities commonly associated with an OS. It didn't even have a file system; through at least the early 1980s, system administrators creating user VMs had to manually specify absolute ranges of disk cylinders to be assigned to each virtual disk.)

In most typical configurations, individual VMs are strongly separated from each other; this is the property that has led some people to propose that VM technology is the solution to our security woes. This is at best partly true, and even the part that is true will hold only under certain circumstances.

A guest OS is, of course, an OS; guest Windows is Windows, guest Linux is Linux, and so on. These do not magically become less vulnerable because of the existence of a hypervisor between them and the silicon; an SQL injection attack against a virtualized web server will succeed or fail in the same way as it would on the same configuration and server without a hypervisor. The same care is necessary, the same feeding is necessary, and the same patches are necessary. In fact, if nothing else in the environment changes, VMs can be *less* secure because the sudden increase in the number of operating systems

can create a very large bump in the system administration load.

On the other hand, the isolation between guest VMs on a single box is much stronger than the isolation between two different users on a conventional OS; as such, the effective target environment for any malware is less. A VM can thus help contain an intrusion. This breaks down, however, when VM technology is used as a *sandbox* technology (see Section 10.3) for user applications. The trouble is that there needs to be too many interactions and connections. A virtualized web browser still needs to be able to save downloaded files and pass mailto: URLs to the mailer; a virtualized mailer needs to talk to the web browser to handle embedded URLs, etc. The degree of protection provided by virtualization in that kind of environment can be quite limited [Bellovin 2006b].

At best, a VM is about the equivalent of letting your enemy have a machine in a rack in your data center. Even without the added risks of cache timing attacks and bugs allowing escape from the virtualized environment, this is probably not the sort of thing you'd be enthusiastic about. On the other hand, if the network connectivity from that machine were limited, say by VLANs and/or internal firewalls, the incremental risk is low.

If a hypervisor is just an OS, why should it be more secure than a conventional OS? After all, an ordinary operating system is supposed to separate different users' programs from each other; why, if these fail, should one think that a VM architecture will succeed?

The answer is the nature of the virtual instruction set. From the perspective of the hypervisor, there are very, very few (and perhaps no) privileged operations, which in turn means there are very, very few (and perhaps no) opportunities for mistakes. Yes, there are resource access requests, for example, attaching a virtual disk drive, but these are generally done by the VM's operator using its command interface; the VM itself may not have the ability to make such requests, valid or not.

A related reason is the absence of privileged VMs. Historically, very few security holes have been in the kernel (though this is changing); rather, they've been due to buggy applications that are either tricked into abusing their privileges or are themselves exploited. Exploitation of a VM, though, is a guest OS issue and with no VMs having any special abilities there is no way to trick one into misusing its powers.

The strong separation provides for another advantage: the hypervisor can do intrusion detection and virus scanning of the guest OSs without fear of interference from a successful attack. Such interference is quite common, ranging from simple measures—adding entries to /etc/hosts to direct virus signature update requests to the wrong place to sophisticated rootkits, collections of programs and tools that hide intrusions. My favorite example involved a sabotaged version of /sbin/init, traditionally process 1 on Unix-like systems. This program is invoked from the kernel at boot time; all other processes on the system are direct or indirect descendants of it. Naturally, a simple scan for changed files (e.g., by Tripwire [G. Kim and Spafford 1994a; G. Kim and Spafford 1994b; G. Kim and Spafford 1994c]) would detect the modified version of /sbin/init, so it took precautions: it modified the kernel's file system code to include a module that looked for requests to open /sbin/init. If the processID was 1—that is, if it was the OS itself trying to boot and hence invoke init—the hacked version was loaded. All other requests,

including Tripwire's, would get the original, unmodified version.

Putting the scanner in the host OS solves this problem if we assume that the infection cannot spread. If the IDS knows a lot about the setup of the VM and knows something about the guest OS, it can do the scans. There are difficulties, but they can be surmounted; see, for one example, [T. Garfinkel and Rosenblum 2003].

10.3 Sandboxes

Sandboxing, sometimes known as *jailing*, is a class of techniques designed to limit the access rights—the effective target environment—of a program. The intent, of course, is to limit the damage that that program can do if it is evil, infected, or subverted. The concept has been around for many years. Bill Cheswick used a jail (and apparently coined the usage) when we were monitoring the attacker he dubbed "Berferd" [Cheswick 1992; Cheswick 2010; Cheswick and Bellovin 1994]. Cheswick's fundamental isolation primitive was chroot(), a system call that has been in Unix since the late 1970s. Sandboxing has become of much more interest in the last few years, since both Microsoft and Apple have made it a central part of their desktop architectures, most notably for browsers but also for other high-risk applications.

Quite a variety of mechanisms have been used to implement sandboxes. For our purposes here, though, those details aren't important. From an architectural perspective, there are three interesting questions:

1. What restrictions are imposed on the sandboxed program? Alternatively, what sensitive resources are still accessible?
2. What is the granularity of exceptions to the general restrictions?
3. How easy is it to use correctly, that is, how hard is it to configure and maintain?

Naturally, the questions interact. VMs, for example, restrict more or less every form of access, but they're hard to maintain (you have an entire extra OS to patch). Similarly, fine-grained security policies require a lot of detailed work to understand exactly what resources are needed; this provides a lot of flexibility, but it's easy to get it wrong. If you restrict too much, you may find that under unusual circumstances the application doesn't run; restrict too little and you create security holes. Leaving some of the decisions to system administrators or (worse yet) end users is generally an invitation to trouble.

In practice, strong isolation is relatively easy: put the suspect code on a VM. Apart from the maintenance issues, the isolation is often too strong; it's too hard to share information, even aside from the problem of creating the policy specification. This strong isolation alone is why VMs are not good general-purpose sandboxes.

There are two issues with sharing items across the boundary of the sandbox. The first, as noted, is the policy specification issue. The second is more subtle: there is now a communications channel between the untrusted application and the outside world. You have to be extremely suspicious of anything coming out of the sandbox; conceptually, that program is under the control of your enemy (and of course, the more skillful your enemy

is the more you have to worry). Consider just a simple bidirectional stream. This is *exactly* equivalent to a network connection, but we already know that nasty things can happen via Internet connections; streams from the sandbox are no better. The situation almost feels like the Schrödinger's cat experiment [Trimmer 1980]: until you look at the sandboxed process, in some sense nothing has happened—but here, it's the observer who may end up dead instead of the cat.

For a case study, let's look back at my suggestion at the end of Chapter 9, that the signon to a public wireless network be done via a sandboxed browser. Why would a sandbox help? More importantly, what sort of sandbox is appropriate? What sort of communication to and from the sandbox is necessary?

The risk we were worried about was unpleasant things happening to a roaming laptop on a public wireless network. A strict VPN, where all outbound traffic was encrypted and no inbound traffic was accepted if it were not cryptographically protected, was the solution I suggested. However, that left no way to sign on to the public network, so I suggested a sandboxed browser.

Assuming that the VPN does its job, the specific issue is what can happen to the sandboxed browser. That suggests one characteristic of the sandbox: it should permit no other network I/O than by the browser and its utterly necessary adjuncts, such as DNS queries. It would be nice if the browser itself were minimally configured, with no Java, no JavaScript, no Flash, and so on, but that's not realistic; too many login screens need one or both of the latter two. Still, the network capabilities of the sandbox should be strictly limited.

Another characteristic is that to a first approximation, we don't need any output from the sandbox; that tremendously improves security. Unfortunately, that is just a first approximation; there are many strange and wondrous varieties of login systems. As Ted Lemon has noted, "The ingenuity that is applied to breaking the Internet by hotel Internet providers is genuinely inspiring. If only they would use their powers for good… ."[1] I've seen screens where the first login attempt gives you a password you need for subsequent uses. I've seen pop-up windows that give live countdown timers for your session; if you close that window, you're assumed to have logged off. I've seen screens that show your credit card receipt, which you really need to save to be reimbursed for the expenditure. The details vary, but it seems clear that some way to save the output from such a session, if only as an image file, is necessary. Though not ideal, it is still reasonably safe.

1. "Ted Lemon—Google+—The network at World IPv6 forum has a captive portal which intercepts port 80," https://plus.google.com/108428495411541457022/posts/2782SLV9Fe2.

The third characteristic, though, is more problematic: the browser may need access to your password manager (Chapter 7). Often, you're logging on to a network run by a service provider for which you have an account; that account needs a password, and of course you want your password manager to keep track of it for you. How should you implement this?

The wrong answer is to run your regular password manager in the sandbox. Doing so

would imply that an untrusted environment—the browser that's running in the sandbox precisely because it's doing dangerous things—should have access to all of your credentials. It may be acceptable to have a separate instance of the password manager there, one that only knows about network access credentials, but that implies the need for persistent storage across instantiations of the sandbox. That's undesirable, because it means that infections can persist, too; you want to be able to discard the sandbox when you're done logging in. Still, this is an option that may be useful for some people.

The other extreme is to do it manually: run the password manager in your regular environment, look up the password you need, and type it into the sandbox. Alternately, you can probably use copy and paste; that's possible with most hypervisors. You lose something, though: automatic selection of the proper password based on the URL you've visited, which is a useful guard against phishing attacks.

It's tempting at this point to try to devise a custom solution, a specialized channel that would allow for precisely the proper queries. Is there some way to export the actual URL from the sandbox and import the proper login details? It can be done, but it's probably not a good idea.

Assume some sort of channel back from the browser—the compromised browser, running in a thoroughly p0wned sandbox—to the password manager. The first risk is the obvious one, as discussed above: is the code that is listening to this channel trustworthy? Let's assume that it is. It's still a dangerous situation. For obvious reasons, the browser can't be allowed to make direct queries—"here's a URL, give me the login credentials"—as opposed to popping up some sort of dialog box asking the user to permit the action. Visualize that request, in the context of this somewhat tongue-in-cheek definition of a dialog box: "A window in which resides a button labeled 'OK' and a variety of text and other content that users ignore."[2] In other words, too often (and probably far too often) users will ignore the URL in the dialog box and just give their assent to their password being collected. This sort of habituation to security warnings is very well known (see, e.g., [Egelman, Cranor, and Hong 2008]). If a user is expecting a password dialog box and is not expecting an attack, all too frequently the user will simply assent to the request and not bother reading it.

2. "Glossary—W3C Web Security Context Wiki," http://www.w3.org/2006/WSC/wiki/Glossary.

We are faced with an unpleasant dilemma here: do we trust the user to notice which URL is being requested, and copy and paste the proper password, or do we trust the user to notice a bogus hostname in a dialog box? We can resolve it by looking at the comparative harm from the two failure modes. In the first case, the malware will get the password to the network the user is trying to access; in the second, it will receive some arbitrary password of the user's. The former is of comparatively low value; the latter could be high value, but the success of the attack depends in part on the attacker's ability to guess for which sites the user has passwords. There are some obvious guesses that will often be successful—a Google or Facebook login, or something like American Express in

an airport, to name just a few—but a targetier or MI-31 is likely to *know* what sites you connect to. Looked at this way, the choice is quite clear: a custom interface to the password manager from the sandbox is far riskier, especially against sophisticated adversaries.

It's also worth realizing the limitations of sandboxing. In this scenario, it does nothing to prevent you from connecting to a phony access point or being asked to enter your credit card details to a bogus access server. A sandbox is designed to protect your host against software you wish to run; these threats represent evil on external boxes.

10.4 The Cloud

Is the cloud "secure"?

The question as I have just phrased it is unanswerable because neither "secure" nor (especially) "cloud" have rigorous definitions. We also have to ask the first question in any security dialog: "what are you trying to protect, and against whom?" And there's one more question that is asked all too infrequently: secure compared with which alternatives? This last question is often the most interesting.

Intuitively, the cloud can provide computing cycles (e.g., Amazon's EC2) and/or remote storage (see Section 10.7); this latter can be just for the owner, or it can permit sharing. For "security," we can use the usual trio of confidentiality, integrity, and availability. Our questions, then, are these: for remote storage and computing, does the cloud provide more or less confidentiality, integrity, and availability, across a wide spectrum of attackers, compared with providing the same functionality yourself?

Availability is probably the easiest to answer. I assert that despite occasional well-publicized failures, a professionally run cloud service is *more* available than a typical in-house solution. There are more redundant resources that can be used to resolve outages, whether malicious or accidental in origin. A good cloud service will use RAID disks and back them up. (How recent are your backups? When did you last test your ability to recover from a disk crash?) Put it this way: do you have more servers than Amazon? Do you have more bandwidth than Google? Yes, a failure at a large provider will affect more users [Brodkin 2012]; conversely, we hear about such failures more than we hear about the routine (and frequent) outages at typical corporations. The difficult issue is whether your enterprise can function if all of its computer capability is cloud resident; diversity is always a good thing. But should you seek diversity in your own environment or via different cloud providers? If you need very high availability, you can't just accept a cloud provider's word that all will be well.

Integrity and confidentiality are somewhat harder to assess. Most (though of course not all) penetrations result from exploitation of holes for which patches are already available. Is your own in-house staff conscientious about installing all available fixes? Are your systems properly configured, especially for sharing data? Would a service provider do better? The questions aren't easy to answer. If the reason for a delay in patching is lack of resources, the cloud provider is likely to be better. On the other hand, many enterprises

delay until they can assess the compatibility of their own in-house applications with the new system—and cloud providers have very many applications to worry about. Sharing resources with outsiders is almost certainly better done via the cloud; their access control mechanisms are tuned for that sort of scenario, and they've dealt with the complexity of the underlying platforms.

In-house computing probably has the edge when considering possible attackers. Apart from a provider's own employees turning to the dark side, you run the risk of being collateral damage when some other customer is being targeted. There are also legal issues to consider: under the laws of many different countries, you arguably have less protection against "subpoena attacks" when your data isn't stored in house [Maxwell and Wolf 2012].

I do not claim that the answer to cloud computing is simple. I do assert that running your own systems is not inherently better, even from a security perspective. You need to think about the problem without preconceptions and do a detailed assessment for your own particular situation.

10.5 Security Architecture of Cloud Providers

Although there are many different ways to build a cloud provider, from a security perspective there are three interlinked pieces: the service platform, the administration and provisioning system, and the corporation. A failure in any of these pieces or in the people associated with them can lead to trouble for customers.

The service platform is what gets all of the security attention, but there are questions that have to be answered. Is it Windows or Linux? How is virtualization done? Where are files stored? How are network segments partitioned? What security mechanisms are used for the switches and routers? How much bandwidth is available? What flavor of hypervisor is used? What auxiliary services, such as credit card services, are provided? Who has physical access to the data centers? How are they screened? All of these are important; the answers should be addressed in reasonable detail in provider documentation, for example, [Amazon 2011].

The service platform isn't the only risk, though. The administration and provisioning system—the mechanisms, both web-based and programmatic, by which customers request and configure services—are quite important, too. The issue is not so much abuse of the direct mechanisms (can Customer A request access to B's files? What if C's password is stolen?) as what can happen if the provisioning system is penetrated. This is a very plausible threat. If nothing else, a provisioning system obviously requires databases, and many, many web sites have fallen to SQL injection attacks. Beyond that, of course, there's the human element: are the people who program, maintain, administer, and operate these systems screened to the same extent as those who deal with the service platform?

The third piece of the security puzzle is the corporation itself. Cloud providers are like any other corporation, complete with marketing departments, human resources departments, clerks, lawyers, system administrators, and vice presidents in charge of dispensing administratium [DeBuvitz 1989]. The risk here is indirect but nevertheless very

real: the people who run the service and provisioning platforms are employees and thus have to deal with internal web servers and computers run by the IT group (as well, of course, as shoveling the administratium out of their offices). If an attacker can penetrate an internal system, it can be used to infect a system administrator's computer. If the same computer is used to talk to the service platform, *vae victis*!

The obvious defense against this scenario is an airgap: give such employees two computers, one for general corporate computing and one for work that gets near the service platform, with no connection between them. This is obviously not foolproof—again, remember Stuxnet [Falliere, Murchu, and Chien 2011; Zetter 2014]—but from a customer's perspective, the questions are whether it's even attempted, and how well employees actually follow good practices.

We've seen similar penetrations in the real world. Consider how Twitter was hacked a few years ago [FTC 2010]. In one incident, the bad guys launched a password-guessing attack against an administrative login. In another case, an attacker penetrated the personal email account of an employee. Two similar passwords were stored in plaintext in this account. With this data, the hacker was able to guess the employee's Twitter administrative password and used it to commit mischief.

10.6 Cloud Computing

Computation, especially on small amounts of data, is one of the simpler and more straightforward uses for the cloud. You lease a machine (typically a VM) with a particular operating system, upload the input data, run your programs, and download the results. Depending on the cloud provider, you may find it easy to lease many VMs. Typically, you only pay for the CPU time and data transfers you actually use. What are the security risks?

The first thing to realize is that a VM is, as noted, a computer with an operating system, with all of the risks appertaining thereto. If a given OS is insecure if run on bare silicon, sprinkling it with cloud dust does not magically secure it. The surrounding environment may provide some protection (see below), but an OS is an OS.

There has yet to be a perfect, bug-free, secure operating system; given that, it is obvious that it will be necessary to apply patches on occasion. Who will maintain the OS for your VMs? There are different models possible. Sometimes, you, the user will be responsible; in others, the cloud provider will do it. Neither is intrinsically better; while it's nice to be rid of the responsibility, OS patches can break your applications (Chapter 13).

Beyond the simple OS, some cloud providers supply interfaces to common applications and services: databases, credit card billing systems, content management systems, even shopping carts for small online stores. These programs can also have security bugs, which will also require patches. The provider will generally handle that, though of course with some risk of breaking code of yours that talks to these services.

Suppose, though, that you're on your own, and the cloud provider does not patch your OS and applications for you. What then? The cloud provides three things: a large amount of computing capacity, surge capacity for peak loads, and (in theory) higher availability

than most stand-alone data centers. Against that, you take the risk of a penetration from another customer, from the provider's staff, or from some hacker who has penetrated the cloud provider, and in particular their support systems. It is difficult to measure any of these risks, especially that of insider attacks; such crimes are rarely detected, let alone reported. There may also be some extra advantages if you don't provide enough physical security for your own data centers; there may also be better tools for managing large numbers of VMs.

Cloud computing is an especially good place to treat security as an economic issue. There is certainly some incremental risk; on the other hand, running a data center sized for peak loads can be quite expensive, particularly when you include the costs of electricity and cooling. Furthermore, you have your own insiders to worry about; do you vet your personnel as well as a cloud provider does for its people?

10.7 Cloud Storage

Storage is a particularly important special case of cloud computing. Computation is generally private; users want as much isolation from other customers as is feasible. VMs are thus a good approach, since the strong isolation they provide is all to the good.

Storage is another matter. Cloud storage is desirable precisely because it makes data sharing easy. Services like Dropbox, Box.net, Google Drive, and more make it easy to share files, either broadly or selectively. Is this a safe thing to do?

One question is why organizations might prefer to use these external services for external file sharing, rather than doing it in house. After all, all major operating systems (and most minor ones) have some form of file-sharing ability. There seem to be at least three reasons, apart from the lovely automagic synchronization some of these services provide. First, someone has to manage the authorization name space: if you want to let some external parties view your files, you have to have a way to name them; this in turn means that they have to have some sort of login and credentials on some part of your system. Cloud providers handle that for you; all you have to know is the login name your partners use. Second, file sharing is obviously quite delicate from a security perspective, and several of the popular networked file systems have had a long, sad history of security problems. It's better not to rely on such systems in a high-threat environment. Finally, and partly for this reason, corporate file servers tend to live behind the firewall, where they're unavailable to legitimate outside users. Indeed, many corporate firewalls will even block outbound requests for such protocols; by contrast, most cloud storage services use HTTP or HTTPS, which are allowed through.

The first reason is valid. Managing such logins is a difficult matter, and more so procedurally than technically; it's not at all unreasonable to want to let someone else have the headaches. The other two, though, are troubling. Why should cloud storage vendors be able to produce more secure storage access code than Microsoft, Apple, Oracle, et al.? Perhaps more to the point, can they? Do we have any strong evidence that they've actually succeeded? There is some reason for optimism; the cloud protocols were specifically built for one purpose, secure Internet-wide sharing; at least two of the local solutions, Oracle's

NFS and Microsoft's CIFS, were built on top of very general and problematic *Remote Procedure Call (RPC)* protocols and had authorization mechanisms aimed at local users, making them harder to secure.

On the other hand, at least one cloud storage vendor, Dropbox, has experienced a serious security failure [Singel 2011]. Somehow, a bug in a program change let it accept any password and gain access to any file at all. The flaw was in the authentication mechanisms, not the file access protocol itself, but as I've noted over and over again, security is a systems property; you cannot restrict your attention to just one component.

The third reason for the popularity of cloud storage is the most worrisome. When our security mechanisms exclude valid uses and force use of external solutions that evade—there is no milder word—the firewall, there is something seriously wrong with our protocols, our firewalls, our policies, or (most likely) all of the above. It is possible to bring such services in house—see the discussion in Section 11.3—but it takes a fair amount of effort and care.

Flaws in the design or implementation of the authentication and file access protocols are the two obvious failure modes. There are several other possibilities as well, the most serious of which are the cloud provider being hacked or suffering an insider attack. They're tempting targets for skilled attackers because a single penetration exposes the resources of many other organizations. (Per the threat classification matrix, the provider would be the victim of a targeted or APT attack; its customers would probably be opportunistic victims.) The usual approach to the latter has been via process: the provider uses stringent procedures to limit the number of employees who can get at user files. A better solution to both, from a technical perspective, is to encrypt and decrypt files on the client machines. This, however, makes sharing much more difficult. It takes a moderately complex cryptographic protocol, especially if you want to be able to revoke access—and you almost certainly do.

Another theoretical issue is transport security. Most providers have used the obvious solution, HTTPS rather than HTTP, but it's something to watch for. Note, though, that unless the developers were quite cautious, cloud storage over HTTPS is susceptible to the same PKI flaws as a web browser. A cloud client could easily do certificate pinning, but it's unclear which, if any, products actually implement that. Many mobile apps have trouble doing even basic TLS [Fahl et al. 2012; Georgiev et al. 2012].

The biggest risk, though, has nothing to do with clouds, cumulo-silicon or otherwise. Rather, it is intrinsic to the sharing process: getting the *access control list (ACL)* correct. I'm not even talking about the risks of client-side insider misbehavior. Every shared object is potentially accessible by anyone with a valid login on the server, in that they can construct requests to read or write the file. This is limited by ACLs, but there is a large body of research showing that ACLs are extremely hard to manage, especially in complex situations [Madejski, M. Johnson, and Bellovin 2012; Reeder, Kelley, et al. 2008; Reeder and Maxion 2005; Smetters and Good 2009]. In a cloud scenario, where many different people may set controls on many shared resources, this is a recipe for disaster. Let me give just one scenario. Suppose your company shares many files with an outside vendor; you

then switch vendors. The initial set of ACLs were created and maintained by someone who has since left your company, and because of changes in your system design in the years since the original contract was signed there was a lot of churn in exactly what files needed to be shared. How will you revoke all of the access rights held by the first company?

It is worth repeating that this last issue would arise even if you used your own, internal file servers. The only saving grace, were you to do it in house, is the increased ability to write custom scripts to search your file system—more likely, *file systems*—for all such files; you may not have that ability with a cloud provider. Their ability to help you solve problems like this may be the biggest single security difference between otherwise similar offerings.

10.8 Analysis

By now, it is clear that there is not and cannot be any simple answer to the question of cloud!security. There are too many types of cloud services and usage patterns; each has its own risks and benefits. Do not believe any blanket assertions about the security or insecurity of "the" cloud; there is simply no such thing. The same is true of sandboxing and virtualization; not only are there different uses (and hence different security properties), there are different implementations.

The risk/benefit trade-offs for cloud storage seem to favor using it, especially for sharing data with external parties. Even if there were similar protocols for in-house cloud storage, the problem of managing the external username space is daunting. The gain from outsourcing that responsibility, though, comes with a potentially serious cost: you have little control over the authentication mechanisms used for your own employees, let alone for your partners'. As noted in Chapter 7, there are many ways to do it poorly. The issue, though, is not the cloud per se, but how a particular service is provided; the risk would be the same if you made the same poor choices yourself.

A very serious concern is unmanaged use of cloud storage by your own employees. If people want to take work home—and many people do—cloud storage can be much more convenient than flash drives. Either may be against corporate policy; both are hard to stop. Fundamentally, though, this is a people problem and a cultural problem more than a technical one. Yes, you can get add-on products that limit use of USB drives, at which point you've made life difficult internally. (Have you ever handed a memory stick with a presentation to a colleague? I sure have, and before that CDs and floppies; slide shows to go are de rigeur in large organizations [Bumiller 2010].) Similarly, you can try to add firewall rules that block unauthorized connections to popular cloud services, but it's easy to set up relays on external private servers and work around the blockages. The real issue is the mismatch between employee work habits or desires (and, perhaps, how people are evaluated) and corporate security policies. Unless you're being targeted by the Andromedans, you're probably better off finding a secure way to let people use cloud storage than trying to ban it.

There's an interesting trade-off here. Which is better, a private file server available

internally and externally, or a commercial cloud storage provider? As noted, common network file system protocols do not have a great reputation for security; however, commercial providers have a relatively large attack surface, per Section 10.5. You would have the same attack surface but you may be less tempting to targetiers; besides, you *know* what precautions you've taken.

A possible solution to some of the security issues posed by a private store is to put it on an external network, but make it accessible only via a VPN, even from inside your organization. The trick will be finding VPN software flexible enough (and comprehensible enough) to allow for two or more secure networks—one for general "phone home" access and one for the file store—and automatic enough that users don't have to think about how to use it. (That's why you want encrypted access even from the inside: you want the user's behavior to be as similar as possible, inside or outside.) The fundamental precept here is to make it easy for people to do the right thing; forcing them to fight menus of soul-wrenching complexity will not endear your system to your colleagues.

If your security policies do permit some sharing of material, you're probably better off trusting your users [M. Johnson et al. 2009]. When official mechanisms are too complicated, people will ignore them and resort to the aforementioned work-arounds.

If one looks only at security, cloud computation is a tougher problem than cloud storage because there's no issue of external logins that need to be maintained. Cloud storage can give you new functionality—shared access—while cloud computation merely saves you money. That, then, is how the question should be answered: is the expected difference in loss from relying on a cloud computation provider more or less than what you save by employing such facilities? Unfortunately, as posed that question is unanswerable, since measuring security isn't possible even without trying to measure the probability of an attack in different situations [Bellovin 2006a]. A different approach is needed.

Start by comparing, as best you can, the security of each of the provider's three components (Section 10.5) with your own equivalents. Pay no attention to mechanisms designed to separate different customers from each other; you don't have the equivalent. You'll probably find that your provisioning component is minimal; if you can't find your own instance, consider the provider's as part of their service platform.

Most of the time, a cloud provider will win on availability and physical security. In many cases, they'll also do better on personnel security, especially for the delivery platform; that is, after all, what they're selling, and many businesses do not do much in the way of background checking. For technical security issues, while you can't measure your security proactively, you can get a good handle by looking retrospectively. Select some pieces of software you run, including the operating system, and do a deep dive into its vulnerability list over the last year. Don't restrict yourself to the vendors' bug lists; much of the time, vendors don't post security holes until patches are available. Check official compendia like the *Common Vulnerabilities and Exposures (CVE)* list (http://cve.mitre.org/cve/index.html) and especially the *National Vulnerability Database*

(NVD) (http://nvd.nist.gov/), as well as sites like bugtraq and Full Disclosure. When were the vulnerabilities known? When were patches available? When did you install them? For how long were you vulnerable? You won't find it easy to get similar data from providers, but now you know what questions to ask. Try this one: what do they do when a bug report shows up on one of those lists? If they don't monitor them—well, that's an answer, too. (On the other hand, do your security people monitor those lists?) Don't forget to ask about auxiliary services offered by the provider; you may not use them, but if they're hacked do they provide an entrée into your systems?

Life gets more interesting when you factor in threat models. Ordinary care should shield both you and cloud providers against joy hackers. Against targetiers, there are advantages in both directions. On the one hand, you may prefer an external service, precisely because it is external and hence less known to insiders; on the other hand, a popular cloud provider may attract outsiders who think there are fruitful pickings once they break in. The really interesting questions, though, are posed by the more skilled attackers, the opportunistic hackers and the Andromedans.

A cloud provider is a *very* tempting target because if someone penetrates it they gain a lot. They have not only the immense power of the provider's platforms, they have access to the systems of many different customers. This can turn some opportunistic hackers into targetiers—and a skilled targetier is, of course, an APT. Perhaps they're at the lower left of that quadrant (given the nomenclature in this book, shall I say that they're from the Lesser Magellanic Cloud?), but they are in the quadrant, with all that implies. Your systems may be collateral damage to such an incident, but that's small comfort. The conclusion is that a cloud provider has to be held to higher security standards than most companies.

Finally, what if your own threat model includes attacks from Andromeda? Does reliance on a cloud provider make things worse? If the cloud provider is good, they're already trying to build defenses at that level. Your problem is that your attack surface has increased, and from a direction against which you have few defenses. MI-31 is the sort of organization that can find hypervisor bugs, bribe or extort employees, and so on. They're also quite capable of setting up a fake company that purchases services from your provider, giving them closer connections to you. This is a serious threat model, which means that you need a really good cloud provider. The best rule of thumb here is to select one whose security—technical, people, and process—is better than your own.

* * *

It is harder to do a blanket analysis on sandboxing per se because there are many kinds of sandboxes and many applications that can benefit from it. The trick is to match the scenario to the available protective technology. The other big issue is the system administration overhead necessary to set up the sandbox; VMs, in general, need as much care and feeding as do regular ones, so the expense will depend on how efficient your organization is at system administration.

Some characteristics that are a good fit for a VM sandbox include few channels to the outside world, simple policies controlling those channels, and infrequent creation and

destruction of the sandbox. A standing service—say, an inbound mail handler—is a good example. By contrast, it does not make sense to spin up a VM for each message; the rate is far too high.

A departmental file server passes the creation rate test, if it instantiates a sandbox per client, but it fails the policy test: complex access control rules that govern which users can perform which operations on given files. Web browsers fail both tests, if you want a separate sandbox for each site visited.

Another issue is what resources are available within the sandbox. VMs provide many; they're great at keeping misbehaving code away from other machines' files, but they don't restrict network access, they have plenty of privileged code lying around, etc. If you need fine-grained restrictions, some sort of jail technology may be better. On the other hand, if all you need is a restriction on its network reach, you could use an inward-facing firewall to block that and get the other advantages of virtualization.

Note the implication here: often, the right choice is to use more than one sandbox mechanism. Thus, a browser may run with reduced privileges overall, but still use separate userids for different sites [S. Ioannidis and Bellovin 2001; H. J. Wang et al. 2009]. That provides intrabrowser protection.

All of the major operating systems have their own sandbox scheme, though the design principles vary tremendously. In general, the choice is between simplicity—of design and implementation, and hence of the implementation and configuration of sandboxed applications—and flexibility. This means that it's relatively hard to port a sandboxed application between OSs unless it assumes minimum granularity and is coded accordingly. (That's one reason that Firefox is sometimes called the least secure browser [Anthony 2014]: it doesn't use a sandbox, because its many different platforms have very different sandboxing paradigms.) A consequence of that is that the same program may be less safe on one OS than on another, which in turn means that flexibility in choice of OS might be appropriate in some deployments.

I sometimes ask my students which operating system is the most secure. It's the wrong question; the proper one is "which OS makes it easiest to write secure applications?" Sandboxing is part of the answer.