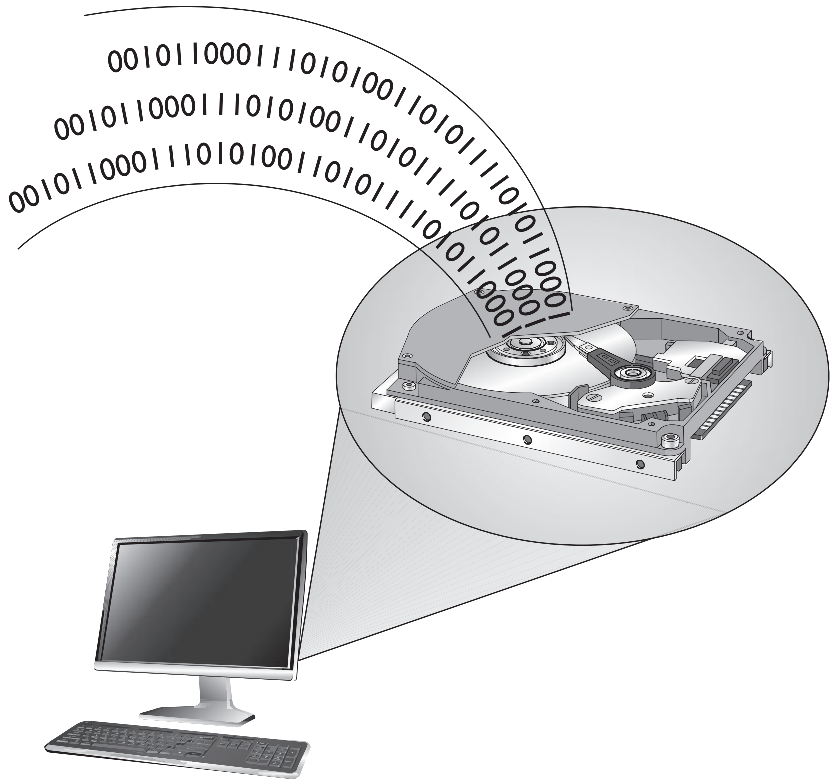


Figure 6-2 Overwriting storage media to protect sensitive data

dealing with solid-state devices. The degree to which information may be recoverable by a sufficiently motivated and capable adversary must not be underestimated or guessed at in ignorance. For the highest-value digital assets, and for all data regulated by government or military classification rules, read and follow the rules and standards.

The guiding principle for deciding what is the necessary method (and cost) of data erasure is to ensure that the enemies' cost of recovering the data exceeds the value of the data. "Sink the company" (or "sink the country") information has value that is so high that the destruction of the storage devices, which involves both the cost of the destruction and the total loss of any potential reusable value of the storage media, is justified. For most other categories of information, multiple or simple overwriting is sufficient. Each organization must evaluate the value of its digital assets and then choose the appropriate erasure/disposal method.

Chapter 5 discussed methods for secure clearing, purging, and destruction of electronic media. Other forms of information, such as paper, microfilm, and microfiche, also require secure disposal. "Dumpster diving" is the practice of searching through trash at homes and businesses to find valuable information that was simply thrown away without being first securely destroyed through shredding or burning.

Atoms and Data

A device that performs degaussing generates a coercive magnetic force that reduces the magnetic flux density of the storage media to zero. This magnetic force is what properly erases data from media. Data is stored on magnetic media by the representation of the polarization of the atoms. Degaussing changes this polarization (magnetic alignment) by using a type of large magnet to bring it back to its original flux (magnetic alignment).

Digital Asset Management

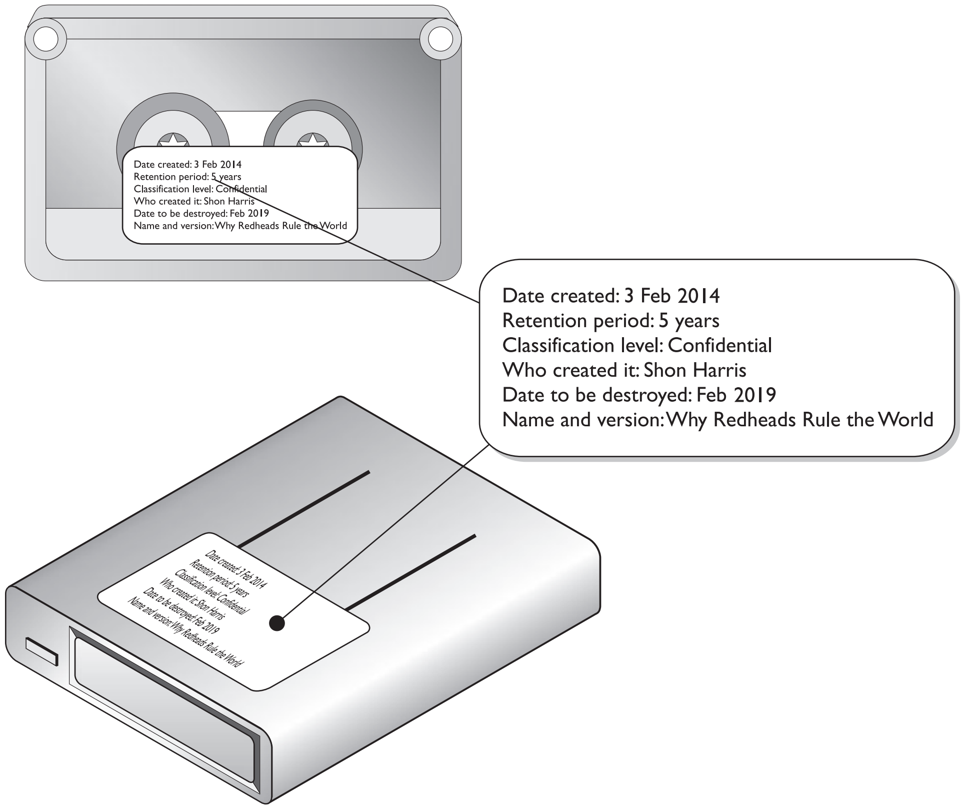
Digital asset management is the process by which organizations ensure their digital assets are properly stored, well protected, and easily available to authorized users. While specific implementations vary, they typically involve the following tasks:

- **Tracking** (audit logging) who has custody of each digital asset at any given moment. This creates the same kind of audit trail as any audit logging activity—to allow an investigation to determine where information was at any given time, who had it, and, for particularly sensitive information, why they accessed it. This enables an investigator to focus efforts on particular people, places, and times if a breach is suspected or known to have happened.
- **Effectively implementing access controls** to restrict who can access each asset to only those people defined by its owner and to enforce the appropriate security measures based on the classification of the digital asset. Certain types of media, due to their sensitivity and storage media, may require special handling. As an example, classified government information may require that the asset may only be removed from the library or its usual storage place under physical guard, and even then may not be removed from the building. Access controls will include *physical* (locked doors, drawers, cabinets, or safes), *technical* (access and authorization control of any automated system for retrieving contents of information in the library), and *administrative* (the actual rules for who is supposed to do what to each piece of information). Finally, the digital media may need to change format, as in printing electronic data to paper, and still needs to be protected at the necessary level, no matter what format it is in. Procedures must include how to continue to provide the appropriate protection. For example, sensitive material that is to be mailed should be sent in a sealable inner envelope and only via a courier service.
- **Tracking the number and location of backup versions** (both onsite and offsite). This is necessary to ensure proper disposal of information when the information reaches the end of its lifespan, to account for the location

and accessibility of information during audits, and to find a backup copy of information if the primary source of the information is lost or damaged.

- **Documenting the history of changes.** For example, when a particular version of a software application kept in the library has been deemed obsolete, this fact must be recorded so the obsolete version of the application is not used unless that particular obsolete version is required. Even once no possible need for the actual asset remains, retaining a log of the former existence and the time and method of its deletion may be useful to demonstrate due diligence.
- **Ensuring environmental conditions do not endanger storage media.** If you store digital assets on local storage media, each media type may be susceptible to damage from one or more environmental influences. For example, all types are susceptible to fire, and most are susceptible to liquids, smoke, and dust. Magnetic storage media are susceptible to strong magnetic fields. Magnetic and optical media are susceptible to variations in temperature and humidity. A media library and any other space where reference copies of information are stored must be physically built so all types of media will be kept within their environmental parameters, and the environment must be monitored to ensure conditions do not range outside of those parameters. Media libraries are particularly useful when large amounts of information must be stored and physically/environmentally protected so that the high cost of environmental control and media management may be centralized in a small number of physical locations and so that cost is spread out over the large number of items stored in the library.
- **Inventorying digital assets** to detect if any asset has been lost or improperly changed. This can reduce the amount of damage a violation of the other protection responsibilities could cause by detecting such violations sooner rather than later, and is a necessary part of the digital asset management life cycle by which the controls in place are verified as being sufficient.
- **Carrying out secure disposal activities.** Disposal activities usually begin at the point at which the information is no longer valuable and becomes a potential liability. Secure disposal of media/information can add significant cost to media management. Knowing that only a certain percentage of the information must be securely erased at the end of its life may significantly reduce the long-term operating costs of the company. Similarly, knowing that certain information must be disposed of securely can reduce the possibility of a storage device being simply thrown in a dumpster and then found by someone who publicly embarrasses or blackmails the company over the data security breach represented by that inappropriate disposal of the information. The business must take into account the useful lifetime of the information to the business, legal, and regulatory restrictions and, conversely, the requirements for retention and archiving when making these decisions. If a law or regulation requires the information to be kept beyond its normally useful lifetime for the business, then disposition may involve *archiving*—moving the information from the ready (and possibly more expensive) accessibility of a library to a long-term stable and (with some effort) retrievable format that has lower storage costs.

- **Internal and external labeling** of each piece of asset in the library should include
  - Date created
  - Retention period
  - Classification level
  - Who created it
  - Date to be destroyed
  - Name and version


Date created: 3 Feb 2014
Retention period: 5 years
Classification level: Confidential
Who created it: Shon Harris
Date to be destroyed: Feb 2019
Name and version: Why Redheads Rule the World

Digital Rights Management

So, how can we protect our digital assets when they leave our organizations? For example, if you share a sensitive file or software system with a customer, how can you ensure that only authorized users gain access to it? *Digital Rights Management (DRM)* refers to a set of technologies that is applied to controlling access to copyrighted data. The technologies themselves don't need to be developed exclusively for this purpose. It is the use of a technology that makes it DRM, not its design. In fact, many of the DRM technologies in use today are standard cryptographic ones. For example, when you buy a Software as a Service

(SaaS) license for, say, Office 365, Microsoft uses standard user authentication and authorization technologies to ensure that you only install and run the allowed number of copies of the software. Without these checks during the installation (and periodically thereafter), most of the features will stop working after a period of time. A potential problem with this approach is that the end-user device may not have Internet connectivity.

An approach to DRM that does not require Internet connectivity is the use of product keys. When you install your application, the key you enter is checked against a proprietary algorithm and, if it matches, the installation is activated. It might be tempting to equate this approach to symmetric key encryption, but in reality, the algorithms employed are not always up to cryptographic standards. Since the user has access to both the key and the executable code of the algorithm, the latter can be reverse-engineered with a bit of effort. This could allow a malicious user to develop a product-key generator with which to effectively bypass DRM. A common way around this threat is to require a one-time online activation of the key.

DRM technologies are also used to protect documents. Adobe, Amazon, and Apple all have their own approaches to limiting the number of copies of an electronic book (e-book) that you can download and read. Another approach to DRM is the use of digital watermarks, which are embedded into the file and can document details such as the owner of the file, the licensee (user), and date of purchase. While watermarks will not stop someone from illegally copying and distributing files, they could help the owner track, identify, and prosecute the perpetrator. An example technique for implementing watermarks is called steganography.

Steganography

Steganography is a method of hiding data in another media type so the very existence of the data is concealed. Common steps are illustrated in Figure 6-3. Only the sender and receiver are supposed to be able to see the message because it is secretly hidden

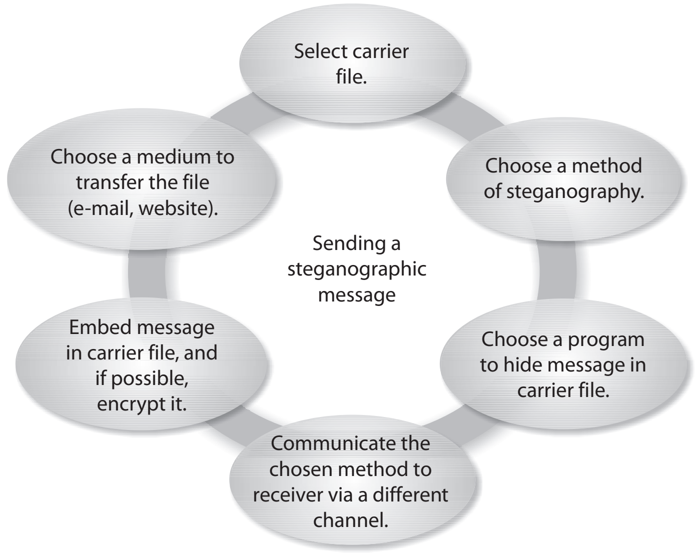

Select carrier file.
Choose a method of steganography.
Choose a program to hide message in carrier file.
Communicate the chosen method to receiver via a different channel.
Embed message in carrier file, and if possible, encrypt it.
Choose a medium to transfer the file (e-mail, website).
Sending a steganographic message

Figure 6-3 Main components of steganography

in a graphic, audio file, document, or other type of media. The message is often just hidden, and not necessarily encrypted. Encrypted messages can draw attention because the encryption tells the bad guy, "This is something sensitive." A message hidden in a picture of your grandmother would not attract this type of attention, even though the same secret message can be embedded into this image. Steganography is a type of security through obscurity.

Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a document file, image file, program, or protocol. Media files are ideal for steganographic transmission because of their large size. As a simple example, a sender might start with an innocuous image file and adjust the color of every 100th pixel to correspond to a letter in the alphabet, a change so subtle that someone not specifically looking for it is unlikely to notice it.

Let's look at the components that are involved with steganography:

- **Carrier** A signal, data stream, or file that has hidden information (payload) inside of it
- **Stegomedium** The medium in which the information is hidden
- **Payload** The information that is to be concealed and transmitted

A method of embedding the message into some types of media is to use the *least significant bit (LSB)*. Many types of files have some bits that can be modified and not affect the file they are in, which is where secret data can be hidden without altering the file in a visible manner. In the LSB approach, graphics with a high resolution or an audio file that has many different types of sounds (high bit rate) are the most successful for hiding information within. There is commonly no noticeable distortion, and the file is usually not increased to a size that can be detected. A 24-bit bitmap file will have 8 bits representing each of the three color values, which are red, green, and blue. These 8 bits are within each pixel. If we consider just the blue, there will be 2^8 different values of blue. The difference between 11111111 and 11111110 in the value for blue intensity is likely to be undetectable by the human eye. Therefore, the least significant bit can be used for something other than color information.

A digital graphic is just a file that shows different colors and intensities of light. The larger the file, the more bits that can be modified without much notice or distortion.

Data Loss Prevention

Unless we diligently apply the right controls to our data wherever it may be, we should expect that some of it will eventually end up in the wrong hands. In fact, even if we do everything right, the risk of this happening will never be eliminated. *Data loss* is the flow of sensitive information, such as PII, to unauthorized external parties. Leaks of personal information by an organization can cause large financial losses. The costs commonly include

- Investigating the incident and remediating the problem
- Contacting affected individuals to inform them about the incident

- Penalties and fines to regulatory agencies
- Contractual liabilities
- Mitigating expenses (such as free credit monitoring services for affected individuals)
- Direct damages to affected individuals

In addition to financial losses, a company's reputation may be damaged and individuals' identities may be stolen.

The most common cause of data breach for a business is a lack of awareness and discipline among employees—an overwhelming majority of all leaks are the result of negligence. The most common forms of negligent data breaches occur due to the inappropriate removal of information—for instance, from a secure company system to an insecure home computer so that the employee can work from home—or due to simple theft of an insecure laptop or tape from a taxi cab, airport security checkpoint, or shipping box. However, breaches also occur due to negligent uses of technologies that are inappropriate for a particular use—for example, reassigning some type of medium (say, a page frame, disk sector, or magnetic tape) that contained one or more objects to an unrelated purpose without securely ensuring that the media contained no residual data.

It would be too easy to simply blame employees for any inappropriate use of information that results in the information being put at risk, followed by breaches. Employees have a job to do, and their understanding of that job is almost entirely based on what their employer tells them. What an employer tells an employee about the job is not limited to, and may not even primarily be in, the "job description." Instead, it will be in the feedback the employee receives on a day-to-day and year-to-year basis regarding their work. If the company in its routine communications to employees and its recurring training, performance reviews, and salary/bonus processes does not include security awareness, then employees will not understand security to be a part of their job.

The more complex the environment and types of media used, the more communication and training that are required to ensure that the environment is well protected. Further, except in government and military environments, company policies and even awareness training will not stop the most dedicated employees from making the best use of up-to-date consumer technologies, including those technologies not yet integrated into the corporate environment, and even those technologies not yet reasonably secured for the corporate environment or corporate information. Companies must stay aware of new consumer technologies and how employees (wish to) use them in the corporate environment. Just saying "no" will not stop an employee from using, say, a personal smartphone, a USB thumb drive, or webmail to forward corporate data to their home e-mail address in order to work on the data when out of the office. Companies must include in their technical security controls the ability to detect and/or prevent such actions through, for example, computer lockdowns, which prevent writing sensitive data to non-company-owned storage devices, such as USB thumb drives, and e-mailing sensitive information to nonapproved e-mail destinations.

Data loss prevention (DLP) comprises the actions that organizations take to prevent unauthorized external parties from gaining access to sensitive data. That definition has some key terms. First, the data has to be considered *sensitive*, the meaning of which we spent a good chunk of the beginning of this chapter discussing. We can't keep every single datum safely locked away inside our systems, so we focus our attention, efforts, and funds on the truly important data. Second, DLP is concerned with *external parties*. If somebody in the accounting department gains access to internal R&D data, that is a problem, but technically it is not considered a data leak. Finally, the external party gaining access to our sensitive data must be *unauthorized* to do so. If former business partners have some of our sensitive data that they were authorized to get at the time they were employed, then that is not considered a leak either. While this emphasis on semantics may seem excessive, it is necessary to properly approach this tremendous threat to our organizations.

EXAM TIP The terms data loss and data leak are used interchangeably by most security professionals. Technically, however, data loss means we do not know where the data is (e.g., after the theft of a laptop), while data leak means that the confidentiality of the data has been compromised (e.g., when the laptop thief posts the files on the Internet).

The real challenge to DLP is in taking a holistic view of our organization. This perspective must incorporate our people, our processes, and then our information. A common mistake when it comes to DLP is to treat the problem as a technological one. If all we do is buy or develop the latest technology aimed at stopping leaks, we are very likely to leak data. If, on the other hand, we consider DLP a program and not a project, and we pay due attention to our business processes, policies, culture, and people, then we have a good fighting chance at mitigating many or even most of the potential leaks. Ultimately, like everything else concerning information system security, we have to acknowledge that despite our best efforts, we will have bad days. The best we can do is stick to the program and make our bad days less frequent and less bad.

General Approaches to DLP

There is no one-size-fits-all approach to DLP, but there are tried-and-true principles that can be helpful. One important principle is the integration of DLP with our risk management processes. This allows us to balance out the totality of risks we face and favor controls that mitigate those risks in multiple areas simultaneously. Not only is this helpful in making the most of our resources, but it also keeps us from making decisions in one silo with little or no regard to their impacts on other silos. In the sections that follow, we will look at key elements of any approach to DLP.

Data Inventories It is difficult to defend an unknown target. Similarly, it is difficult to prevent the leaking of data of which we are unaware or whose sensitivity is unknown. Some organizations try to protect all their data from leakage, but this is not a good approach. For starters, acquiring the resources required to protect everything is likely cost prohibitive to most organizations. Even if an organization is able to afford this level of protection, it runs a very high risk of violating the privacy of its employees and/or customers by examining every single piece of data in its systems.

A good approach is to find and characterize all the data in your organization before you even look at DLP solutions. The task can seem overwhelming at first, but it helps to prioritize things a bit. You can start off by determining what is the most important kind of data for your organization. A compromise of these assets could lead to direct financial losses or give your competitors an advantage in your sector. Are these healthcare records? Financial records? Product designs? Military plans? Once you figure this out, you can start looking for that data across your servers, workstations, mobile devices, cloud computing platforms, and anywhere else it may live. Keep in mind that this data can live in a variety of formats (e.g., database management system records or files) and media (e.g., hard drives or backup tapes). If your experience doing this for the first time is typical, you will probably be amazed at the places in which you find sensitive data.

Once you get a handle on what is your high-value data and where it resides, you can gradually expand the scope of your search to include less valuable, but still sensitive, data. For instance, if your critical data involves designs for next-generation radios, you would want to look for information that could allow someone to get insights into those designs even if they can't directly obtain them. So, for example, if you have patent filings, FCC license applications, and contracts with suppliers of electronic components, then an adversary may be able to use all this data to figure out what you're designing even without direct access to your new radio's plans. This is why it is so difficult for Apple to keep secret all the features of a new iPhone ahead of its launch. Often there is very little you can do to mitigate this risk, but some organizations have gone as far as to file patents and applications they don't intend to use in an effort to deceive adversaries as to their true plans. Obviously, and just as in any other security decision, the costs of these countermeasures must be weighed against the value of the information you're trying to protect. As you keep expanding the scope of your search, you will reach a point of diminishing returns in which the data you are inventorying is not worth the time you spend looking for it.

NOTE We cover the threats posed by adversaries compiling public information (aggregation) and using it to derive otherwise private information (inference) in Chapter 7.

Once you are satisfied that you have inventoried your sensitive data, the next step is to characterize it. We already covered the classification of information earlier in this chapter, so you should know all about data labels. Another element of this characterization is ownership. Who owns a particular set of data? Beyond that, who should be authorized to read or modify it? Depending on your organization, your data may have other characteristics of importance to the DLP effort, such as which data is regulated and how long it must be retained.

Data Flows Data that stays put is usually of little use to anyone. Most data will move according to specific business processes through specific network pathways. Understanding data flows at this intersection between business and IT is critical to implementing DLP. Many organizations put their DLP sensors at the perimeter of their networks, thinking that is where the leakages would occur. But if that's the only location these sensors are placed, a large number of leaks may not be detected or stopped. Additionally, as we will discuss in detail when we cover network-based DLP, perimeter sensors can often be bypassed by sophisticated attackers.

A better approach is to use a variety of sensors tuned to specific data flows. Suppose you have a software development team that routinely passes finished code to a quality assurance (QA) team for testing. The code is sensitive, but the QA team is authorized to read (and perhaps modify) it. However, the QA team is not authorized to access code under development or code from projects past. If an adversary compromises the computer used by a member of the QA team and attempts to access the source code for different projects, a DLP solution that is not tuned to that business process will not detect the compromise. The adversary could then repackage the data to avoid your perimeter monitors and successfully extract the data.

Data Protection Strategy The example just described highlights the need for a comprehensive, risk-based data protection strategy. The extent to which we attempt to mitigate these exfiltration routes depends on our assessment of the risk of their use. Obviously, as we increase our scrutiny of a growing set of data items, our costs will grow disproportionately. We usually can't watch everything all the time, so what do we do?

Once we have our data inventories and understand our data flows, we have enough information to do a risk assessment. Recall that we described this process in detail in Chapter 2. The trick is to incorporate data loss into that process. Since we can't guarantee that we will successfully defend against all attacks, we have to assume that sometimes our adversaries will gain access to our networks. Not only does our data protection strategy have to cover our approach to keeping attackers out, but it also must describe how we protect our data against a threat agent that is already inside. The following are some key areas to consider when developing data protection strategies:

- **Backup and recovery** Though we have been focusing our attention on data leaks, it is also important to consider the steps to prevent the loss of this data due to electromechanical or human failures. As we take care of this, we need to also consider the risk that, while we focus our attention on preventing leaks of our primary data stores, our adversaries may be focusing their attention on stealing the backups.
- **Data life cycle** Most of us can intuitively grasp the security issues at each of the stages of the data life cycle. However, we tend to disregard securing the data as it transitions from one stage to another. For instance, if we are archiving data at an offsite location, are we ensuring that it is protected as it travels there?
- **Physical security** While IT provides a wealth of tools and resources to help us protect our data, we must also consider what happens when an adversary just steals a hard drive left in an unsecured area, as happened to Sentara Heart Hospital in Norfolk, Virginia, in August 2015.
- **Security culture** Our information systems users can be a tremendous control if properly educated and incentivized. By developing a culture of security within our organizations, we not only reduce the incidence of users clicking on malicious links and opening attachments, but we also turn each of them into a security sensor, able to detect attacks that we may not otherwise be able to.

- **Privacy** Every data protection policy should carefully balance the need to monitor data with the need to protect our users' privacy. If we allow our users to check personal e-mail or visit social media sites during their breaks, would our systems be quietly monitoring their private communications?
- **Organizational change** Many large organizations grow because of mergers and acquisitions. When these changes happen, we must ensure that the data protection approaches of all entities involved are consistent and sufficient. To do otherwise is to ensure that the overall security posture of the new organization is the lesser of its constituents' security postures.

Implementation, Testing, and Tuning All the elements of a DLP process that we have discussed so far (i.e., data inventories, data flows, and data protection strategies) are administrative in nature. We finally get to discuss the part of DLP with which most of us are familiar: deploying and running a toolset. The sequence of our discussion so far has been deliberate in that the technological part needs to be informed by the other elements we've covered. Many organizations have wasted large sums of money on so-called solutions that, though well-known and highly regarded, are just not suitable for their particular environment.

Assuming we've done our administrative homework and have a good understanding of our true DLP requirements, we can evaluate products according to our own criteria, not someone else's. The following are some aspects of a possible solution that most organizations will want to consider when comparing competing products:

- **Sensitive data awareness** Different tools will use different approaches to analyzing the sensitivity of documents' contents and the context in which they are being used. In general terms, the more depth of analysis and breadth of techniques that a product offers, the better. Typical approaches to finding and tracking sensitive data include keywords, regular expressions, tags, and statistical methods.
- **Policy engine** Policies are at the heart of any DLP solution. Unfortunately, not all policy engines are created equal. Some allow extremely granular control but require obscure methods for defining these policies. Other solutions are less expressive but are simple to understand. There is no right answer here, so each organization will weigh this aspect of a set of solutions differently.
- **Interoperability** DLP tools must play nicely with existing infrastructure, which is why most vendors will assure you that their product is interoperable. The trick becomes to determine precisely how this integration takes place. Some products are technically interoperable but, in practice, require so much effort to integrate that they become infeasible.
- **Accuracy** At the end of the day, DLP solutions keep your data out of the hands of unauthorized entities. Therefore, the right solution is one that is accurate in its identification and prevention of incidents that result in the leakage of sensitive data. The best way to assess this criterion is by testing a candidate solution in an environment that mimics the actual conditions in the organization.

Once we select a DLP solution, the next interrelated tasks are integration, testing, and tuning. Obviously, we want to ensure that bringing the new toolset online won't disrupt any of our existing systems or processes, but testing needs to cover a lot more than that. The most critical elements when testing any DLP solution are to verify that it allows authorized data processing and to ensure that it prevents unauthorized data processing.

Verifying that authorized processes are not hampered by the DLP solution is fairly straightforward if we have already inventoried our data and the authorized flows. The data flows, in particular, will tell us exactly what our tests should look like. For instance, if we have a data flow for source code from the software development team to the QA team, then we should test that it is in fact allowed to occur by the new DLP tool. We probably won't have the resources to exhaustively test all flows, which means we should prioritize them based on their criticality to the organization. As time permits, we can always come back and test the remaining, and arguably less common or critical, processes (before our users do).

Testing the second critical element, that the DLP solution prevents unauthorized flows, requires a bit more work and creativity. Essentially, we are trying to imagine the ways in which threat agents might cause our data to leak. A useful tool in documenting these types of activities is called the misuse case. *Misuse cases* describe threat actors and the tasks they want to perform on the system. They are related to *use cases*, which are used by system analysts to document the tasks that authorized actors want to perform on a system. By compiling a list of misuse cases, we can keep a record of which data leak scenarios are most likely, most dangerous, or both. Just like we did when testing authorized flows, we can then prioritize which misuse cases we test first if we are resource constrained. As we test these potential misuses, it is important to ensure that the DLP system behaves in the manner we expect—that is to say, that it *prevents* a leak and doesn't just alert to it. Some organizations have been shocked to learn that their DLP solution has been alerting them about data leaks but doing nothing to stop them, letting their data leak right into the hands of their adversaries.

NOTE We cover misuse cases in detail in Chapter 18.

Finally, we must remember that everything changes. The solution that is exquisitely implemented, finely tuned, and effective immediately is probably going to be ineffective in the near future if we don't continuously monitor, maintain, and improve it. Apart from the efficacy of the tool itself, our organizations change as people, products, and services come and go. The ensuing cultural and environmental changes will also change the effectiveness of our DLP solutions. And, obviously, if we fail to realize that users are installing rogue access points, using thumb drives without restriction, or clicking malicious links, then it is just a matter of time before our expensive DLP solution will be circumvented.

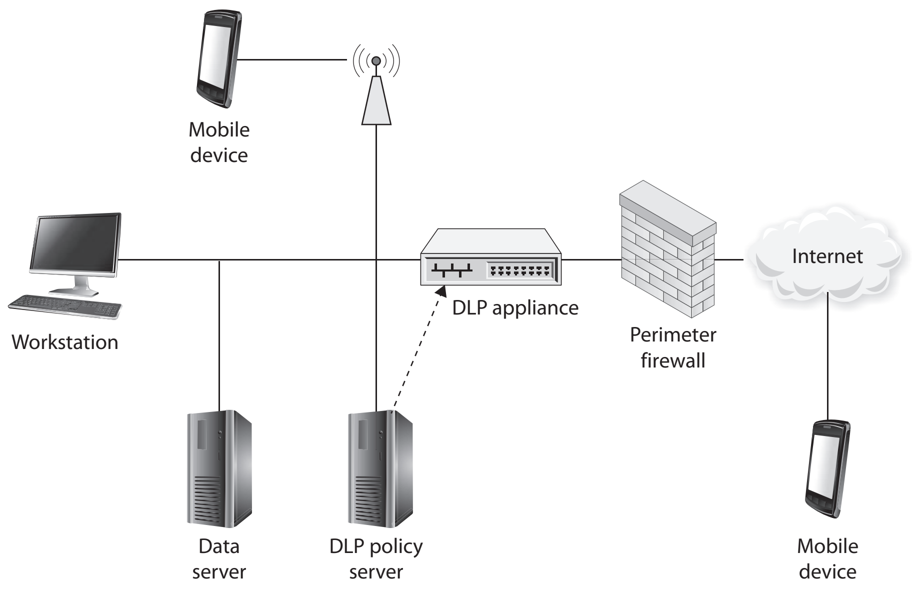


Figure 6-4 Network DLP

Network DLP

Network DLP (NDLP) applies data protection policies to data in motion. NDLP products are normally implemented as appliances that are deployed at the perimeter of an organization's networks. They can also be deployed at the boundaries of internal subnetworks and could be deployed as modules within a modular security appliance. Figure 6-4 shows how an NDLP solution might be deployed with a single appliance at the edge of the network and communicating with a DLP policy server.

DLP Resiliency

Resiliency is the ability to deal with challenges, damage, and crises and bounce back to normal or near-normal condition in short order. It is an important element of security in general and of DLP in particular.

Assume your organization's information systems have been compromised (and it wasn't detected): What does the adversary do next, and how can you detect and deal with *that*? It is a sad reality that virtually all organizations have been attacked and that most have been breached. A key differentiator between those who withstand attacks relatively unscathed and those who suffer tremendous damage is their attitude toward operating in contested environments. If an organization's entire security strategy hinges on keeping adversaries off its networks, then it will likely fail catastrophically when an adversary manages to break in. If, on the other hand, the strategy builds on the concept of resiliency and accounts for the continuation of critical processes even with adversaries operating inside the perimeter, then the failures will likely be less destructive and restoration may be much quicker.

From a practical perspective, the high cost of NDLP devices leads most organizations to deploy them at traffic choke points rather than throughout the network. Consequently, NDLP devices likely will not detect leaks that don't traverse the network segment on which the devices are installed. For example, suppose that an attacker is able to connect to a wireless access point and gain unauthorized access to a subnet that is not protected by an NDLP tool. This can be visualized in Figure 6-4 by supposing that the attacker is using the device connected to the WAP. Though this might seem like an obvious mistake, many organizations fail to consider their wireless subnets when planning for DLP. Alternatively, malicious insiders could connect their workstations directly to a mobile or external storage device, copy sensitive data, and remove it from the premises completely undetected.

The principal drawback of an NDLP solution is that it will not protect data on devices that are not on the organizational network. Mobile device users will be most at risk, since they will be vulnerable whenever they leave the premises. Since we expect the ranks of our mobile users to continue to increase into the future, this will be an enduring challenge for NDLP.

Endpoint DLP

Endpoint DLP (EDLP) applies protection policies to data at rest and data in use. EDLP is implemented in software running on each protected endpoint. This software, usually called a *DLP agent*, communicates with the DLP policy server to update policies and report events. Figure 6-5 illustrates an EDLP implementation.

EDLP allows a degree of protection that is normally not possible with NDLP. The reason is that the data is observable at the point of creation. When a user enters PII on

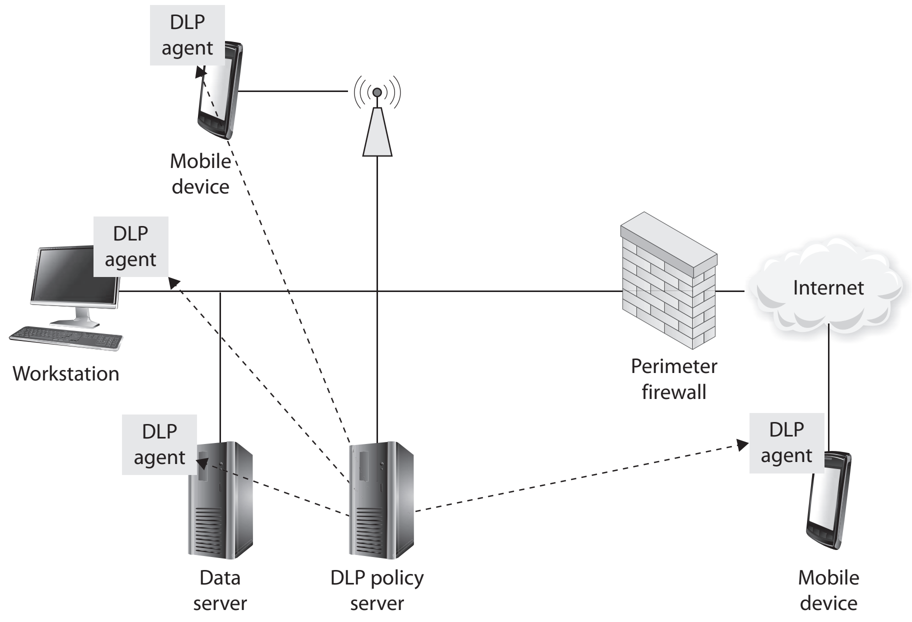


Figure 6-5 Endpoint DLP

the device during an interview with a client, the EDLP agent detects the new sensitive data and immediately applies the pertinent protection policies to it. Even if the data is encrypted on the device when it is at rest, it will have to be decrypted whenever it is in use, which allows for EDLP inspection and monitoring. Finally, if the user attempts to copy the data to a non-networked device such as a thumb drive, or if it is improperly deleted, EDLP will pick up on these possible policy violations. None of these examples would be possible using NDLP.

The main drawback of EDLP is complexity. Compared to NDLP, these solutions require a lot more presence points in the organization, and each of these points may have unique configuration, execution, or authentication challenges. Additionally, since the agents must be deployed to every device that could possibly handle sensitive data, the cost could be much higher than that of an NDLP solution. Another challenge is ensuring that all the agents are updated regularly, both for software patches and policy changes. Finally, since a pure EDLP solution is unaware of data-in-motion protection violations, it would be possible for attackers to circumvent the protections (e.g., by disabling the agent through malware) and leave the organization blind to the ongoing leakages. It is typically harder to disable NDLP, because it is normally implemented in an appliance that is difficult for attackers to exploit.

Hybrid DLP

Another approach to DLP is to deploy both NDLP and EDLP across the enterprise. Obviously, this approach is the costliest and most complex. For organizations that can afford it, however, it offers the best coverage. Figure 6-6 shows how a hybrid NDLP/EDLP deployment might look.

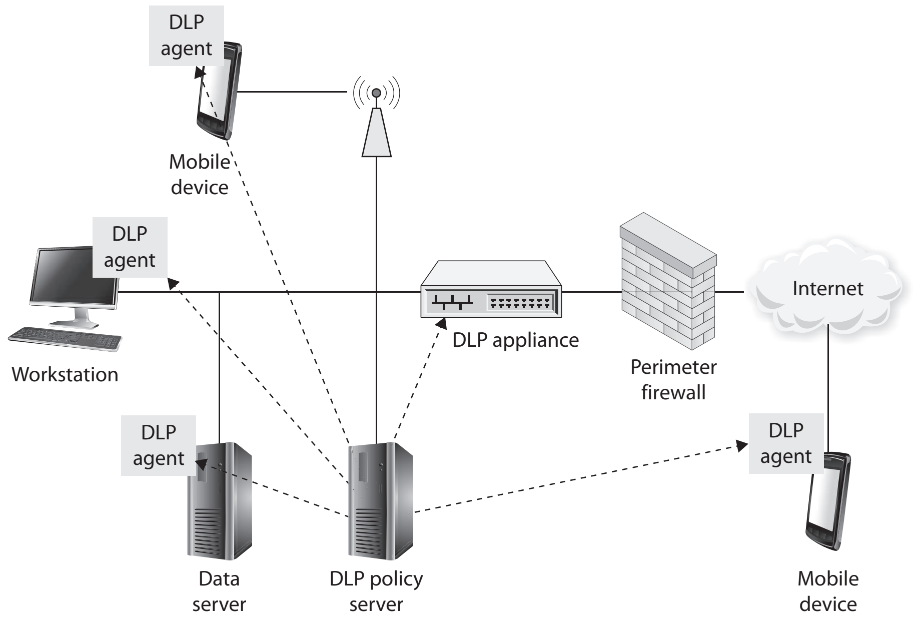


Figure 6-6 Hybrid NDLP/EDLP

Cloud Access Security Broker

The DLP approaches described so far work best (or perhaps only) in traditional network environments that have a clearly defined perimeter. But what about organizations that use cloud services, especially services that employees can access from their own devices? Whatever happens in the cloud is usually not visible (or controllable) by the organization. A *cloud access security broker (CASB)* is a system that provides visibility and security controls for cloud services. A CASB monitors what users do in the cloud and applies whatever policies and controls are applicable to that activity.

For example, suppose a nurse at a healthcare organization uses Microsoft 365 to take notes when interviewing a new patient. That document is created and exists only in the cloud and clearly contains sensitive healthcare information that must be protected under HIPAA. Without a CASB solution, the organization would depend solely on the nurse doing the right things, including ensuring the data is encrypted and not shared with any unauthorized parties. A CASB could automatically update the inventory of sensitive data, apply any labels in the document's metadata for tracking it, encrypt it, and ensure it is only shared with specific authorized entities.

Most CASBs do their work by leveraging one of two techniques: proxies or application programming interfaces (APIs). The proxy technique places the CASB in the data path between the endpoint and the cloud service provider, as shown on the left in Figure 6-7. For example, you could have an appliance in your network that automatically detects user connection requests to a cloud service, intercepts that user connection, and creates a tunnel to the service provider. In this way, all traffic to the cloud is routed through the CASB so that it can inspect it and apply the appropriate controls.

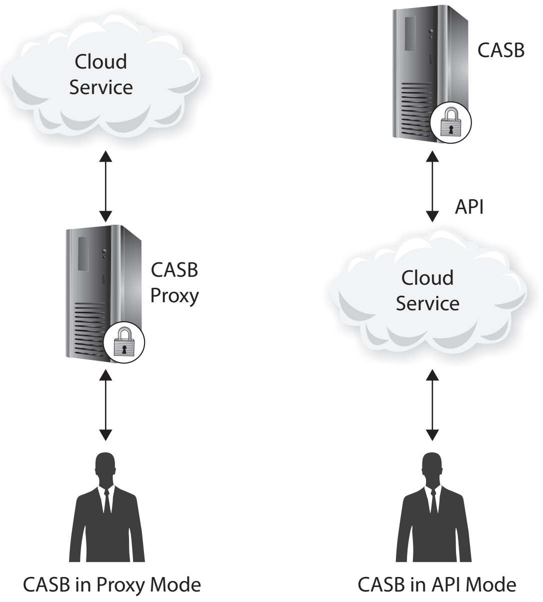


Figure 6-7 Two common approaches to implementing CASBs: proxy and API

But what if you have remote users who are not connected to your organization through a VPN? What about staff members trying to access the cloud services through a personal device (assuming that is allowed)? In those situations, you can set up a reverse proxy. The way this works is that the users log into the cloud service, which is configured to immediately route them back to the CASB, which then completes the connection back to the cloud.

There are a number of challenges with using proxies for CASBs. For starters, they need to intercept the users' encrypted traffic, which will generate browser alerts unless the browsers are configured to trust the proxy. While this works on organizational computers, it is a bit trickier to do on personally owned devices. Another challenge is that, depending on how much traffic goes to cloud service providers, the CASB can become a choke point that slows down the user experience. It also represents a single point of failure unless you deploy redundant systems. Perhaps the biggest challenge, however, has to do with the fast pace of innovation and updates to cloud services. As new features are added and others changed or removed, the CASB needs to be updated accordingly. The problem is not only that the CASB will miss something important but that it may actually break a feature by not knowing how to deal with it properly. For this reason, some vendors such as Google and Microsoft advise against using CASBs in proxy mode.

The other way to implement CASBs is by leveraging the APIs exposed by the service providers themselves, as you can see on the right side of Figure 6-7. An API is a way to have one software system directly access functionality in another one. For example, a properly authenticated CASB could ask Exchange Online (a cloud e-mail solution) for all the activities in the last 24 hours. Most cloud services include APIs to support CASB and, better yet, these APIs are updated by the vendors themselves. This ensures the CASB won't break anything as new features come up.

Chapter Review

Protecting data assets is a much more dynamic and difficult prospect than is protecting most other asset types. The main reason for this is that data is so fluid. It can be stored in unanticipated places, flow in multiple directions (and to multiple recipients) simultaneously, and end up being used in unexpected ways. Our data protection strategies must account for the various states in which our data may be found. For each state, there are multiple unique threats that our security controls must mitigate.

Still, regardless of our best efforts, data may end up in the wrong hands. We want to implement protection methods that minimize the risk of this happening, alert us as quickly as possible if it does, and allow us to track and, if possible, recover the data effectively. We devoted particular attention to three methods of protecting data that you should remember for the exam and for your job: Digital Rights Management (DRM), data loss/leak prevention (DLP), and cloud access security brokers (CASBs).

Quick Review

- Data at rest refers to data that resides in external or auxiliary storage devices, such as hard drives or optical discs.
- Every major operating system supports whole-disk encryption, which is a good way to protect data at rest.

- Data in motion is data that is moving between computing nodes over a data network such as the Internet.
- TLS, IPSec, and VPNs are typical ways to use cryptography to protect data in motion.
- Data in use is the term for data residing in primary storage devices, such as volatile memory (e.g., RAM), memory caches, or CPU registers.
- Scoping is taking a broader standard and trimming out the irrelevant or otherwise unwanted parts.
- Tailoring is making changes to specific provisions in a standard so they better address your requirements.
- A digital asset is anything that exists in digital form, has intrinsic value to the organization, and to which access should be restricted in some way.
- Digital asset management is the process by which organizations ensure their digital assets are properly stored, protected, and easily available to authorized users.
- Steganography is a method of hiding data in another media type so the very existence of the data is concealed.
- Digital Rights Management (DRM) refers to a set of technologies that is applied to controlling access to copyrighted data.
- Data leakage is the flow of sensitive information to unauthorized external parties.
- Data loss prevention (DLP) comprises the actions that organizations take to prevent unauthorized external parties from gaining access to sensitive data.
- Network DLP (NDLP) applies data protection policies to data in motion.
- Endpoint DLP (EDLP) applies data protection policies to data at rest and data in use.
- Cloud access security brokers (CASBs) provide visibility and control over user activities on cloud services.

Questions

Please remember that these questions are formatted and asked in a certain way for a reason. Keep in mind that the CISSP exam is asking questions at a conceptual level. Questions may not always have the perfect answer, and the candidate is advised against always looking for the perfect answer. Instead, the candidate should look for the best answer in the list.

1. Data at rest is commonly

A. Using a RESTful protocol for transmission

B. Stored in registers

C. Being transmitted across the network

D. Stored in external storage devices

2. Data in motion is commonly
   - **A.** Using a RESTful protocol for transmission
   - **B.** Stored in registers
   - **C.** Being transmitted across the network
   - **D.** Stored in external storage devices
3. Data in use is commonly
   - **A.** Using a RESTful protocol for transmission
   - **B.** Stored in registers
   - **C.** Being transmitted across the network
   - **D.** Stored in external storage devices
4. Which of the following best describes an application of cryptography to protect data at rest?
   - **A.** VPN
   - **B.** Degaussing
   - **C.** Whole-disk encryption
   - **D.** Up-to-date antivirus software
5. Which of the following best describes an application of cryptography to protect data in motion?
   - **A.** Testing software against side-channel attacks
   - **B.** TLS
   - **C.** Whole-disk encryption
   - **D.** EDLP
6. Which of the following is not a digital asset management task?
   - **A.** Tracking the number and location of backup versions
   - **B.** Deciding the classification of data assets
   - **C.** Documenting the history of changes
   - **D.** Carrying out secure disposal activities
7. Which data protection method would best allow you to detect a malicious insider trying to access a data asset within your corporate infrastructure?
   - **A.** Digital Rights Management (DRM)
   - **B.** Steganography
   - **C.** Cloud access security broker (CASB)
   - **D.** Data loss prevention (DLP)

8. What term best describes the flow of data assets to an unauthorized external party?
   - **A.** Data leakage
   - **B.** Data in motion
   - **C.** Data flow
   - **D.** Steganography

Answers

1. **D.** Data at rest is characterized by residing in secondary storage devices such as disk drives, DVDs, or magnetic tapes. Registers are temporary storage within the CPU and are used for data storage only when the data is being used.
2. **C.** Data in motion is characterized by network or off-host transmission. The RESTful protocol, while pertaining to a subset of data on a network, is not as good an answer as option C.
3. **B.** Registers are used only while data is being used by the CPU, so when data is resident in registers, it is, by definition, in use.
4. **C.** Data at rest is best protected using whole-disk encryption on the user workstations or mobile computers. None of the other options apply to data at rest.
5. **B.** Data in motion is best protected by network encryption solutions such as TLS, VPN, or IPSec. None of the other options apply to data in motion.
6. **B.** The classification of a data asset is determined by the asset owner before it starts being managed. Otherwise, how would the manager know how to handle it? All other answers are typically part of digital asset management.
7. **C.** Cloud access security brokers (CASBs) provide visibility and control over user activities on cloud services. Provided the asset in question is in the cloud, this would be your best option. Data loss prevention (DLP) systems are primarily concerned with preventing unauthorized external parties from gaining access to sensitive data.
8. **A.** Data leakage is the flow of sensitive information to unauthorized external parties.

This page intentionally left blank

PART III

Security Architecture and Engineering

■ **Chapter 7**	System Architectures
■ **Chapter 8**	Cryptology
■ **Chapter 9**	Security Architectures
■ **Chapter 10**	Site and Facility Security

This page intentionally left blank

CHAPTER 7

System Architectures

This chapter presents the following:

- General system architectures
- Industrial control systems
- Virtualized systems
- Cloud-based systems
- Pervasive systems
- Distributed systems

Computer system analysis is like child-rearing; you can do grievous damage, but you cannot ensure success.

—Tom DeMarco

As we have seen in previous chapters, most systems leverage other systems in some way, whether by sharing data with each other or by sharing services with each other. While each system has its own set of vulnerabilities, the interdependencies between them create a new class of vulnerabilities that we must address. In this chapter, we look at ways to assess and mitigate the vulnerabilities of security architectures, designs, and solution elements. We'll do this by looking at some of the most common system architectures. For each, we classify components based on their roles and the manner in which they interact with others. Along the way, we'll look at potential vulnerabilities in each architecture and also at the manner in which these vulnerabilities might affect other connected components.

General System Architectures

A *system* is a set of things working together in order to do something. An *architecture* describes the designed structure of something. A *system architecture*, then, is a description of how specific components are deliberately put together to perform some actions. Recall from the Chapter 4 discussion of TOGAF and the Zachman Framework that there are different perspectives or levels of abstraction at which a system architecture can be presented depending on the audience. In this chapter, we present what TOGAF would call application architectures. In other words, we describe how applications running in one or more computing devices interact with each other and with users.

Client-Based Systems

Let's start with the simplest computing system architecture, the one that ruled the early days of personal computing. *Client-based systems* are embodied in applications that execute entirely on one user device (such as a workstation or smartphone). The software is installed on a specific computer, and we can use it with no network connectivity. To be clear, the application may still reach out for software patches and updates or to save and retrieve files, but none of its core features require any processing on a remote device. Examples of these are the text and graphic applications that ship with almost every operating system. You could save documents on remote servers, but even with no networking the app is fully functional.

One of the main vulnerabilities of client-based systems is that they tend to have weak authentication mechanisms (if they have them at all). This means an adversary who gains access to the application would be able to also access its data on local or even remote data stores. Furthermore, this data is usually stored in plaintext (unless the underlying operating system encrypts it), which means that even without using the application, the adversary could read its data with ease.

Server-Based Systems

Unlike client-based systems, *server-based systems* (also called *client/server systems*) require that two (or more) separate applications interact with each other across a network connection in order for users to benefit from them. One application (the client) requests services over a network connection that the other application (the server) fulfills. Perhaps the most common example of a server-based application is your web browser, which is designed to connect to a web server. Sure, you could just use your browser to read local documents, but that's not really the way it's meant to be used. Most of us use our browsers to connect two tiers, a client and a server, which is why we call it a two-tier architecture.

Generally, server-based systems are known as *n*-tier architectures, where *n* is a numerical variable that can assume any value. The reason for this is that most of the time only the development team would know the number of tiers in the architecture (which could change over time) even if to the user it looks like just two. Consider the example of browsing the Web, which is probably a two-tier architecture if you are reading a static web page on a small web server. If, on the other hand, you are browsing a typical commercial site, you will probably be going through many more tiers. For example, your client (tier 1) could be connecting to a web server (tier 2) that provides the static HTML, CSS, and some images. The dynamic content, however, is pulled by the web server from an application server (tier 3) that in turn gets the necessary data from a back-end database (tier 4). Figure 7-1 shows what this four-tier architecture would look like.

As you can imagine by looking at Figure 7-1, there are multiple potential security issues to address in a server-based architecture. For starters, access to each tier needs to be deliberately and strictly controlled. Having users authenticate from their clients makes perfect sense, but we must not forget that each of the tiers needs to establish and maintain trust with the others. A common way to ensure this is by developing access control lists (ACLs) that determine which connections are allowed. For example, the database management system in Figure 7-1 might be listening on port 5432 (the default

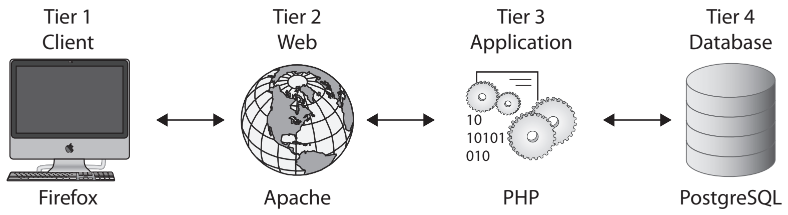


Figure 7-1 A typical four-tier server-based system

port for PostgreSQL, a popular open-source database server), so it makes perfect sense for the application server on tier 3 to connect to that port on the database server. However, it probably shouldn't be allowed to connect on port 3389 and establish a Remote Desktop Protocol (RDP) session because servers don't normally communicate this way.

The following are some other guidelines in securing server-based systems. Keep in mind, however, that this list is by no means comprehensive; it's just meant to give you food for thought.

- Block traffic by default between any components and allow only the specific set of connections that are absolutely necessary.
- Ensure all software is patched and updated as soon as possible.
- Maintain backups (ideally offline) of all servers.
- Use strong authentication for both clients and servers.
- Encrypt all network communications, even between the various servers.
- Encrypt all sensitive data stored anywhere in the system
- Enable logging of all relevant system events, ideally to a remote server.

Database Systems

Most interactive (as opposed to static) web content, such as that in the example four-tier architecture we just looked at, requires a web application to interact with some sort of data source. You may be looking at a catalog of products on an e-commerce site, updating customer data on a customer relationship management (CRM) system, or just reading a blog online. In any case, you need a system to manage your product, or customer, or blog data. This is where database systems come in.

A database management system (DBMS) is a software system that allows you to efficiently create, read, update, and delete (CRUD) any given set of data. Of course, you can always keep all the data in a text file, but that makes it *really* hard to organize, search, maintain, and share among multiple users. A DBMS makes this all easy. It is optimized for efficient storage of data, which means that, unlike flat files, it gives you ways to optimize the storage of all your information. A DBMS also provides the capability to speed up searches, for example, through the use of indexes. Another key feature of a DBMS is that it can provide mechanisms to prevent the accidental corruption of data while it is being manipulated. We typically call changes to a database *transactions*, which is a term to describe the sequence of actions required to change the state of the database.

A foundational principle in database transactions is referred to as their ACID properties, which stands for atomicity, consistency, isolation, and durability. *Atomicity* means that either the entire transactions succeeds or the DBMS rolls it back to its previous state (in other words, clicks the "undo" button). Suppose you are transferring funds between two bank accounts. This transaction consists of two distinct operations: first, you withdraw the funds from the first account, and then you deposit the same amount of funds into the second account. What would happen if there's a massive power outage right after the withdrawal is complete but before the deposit happens? In that case, the money could just disappear. If this was an atomic transaction, the system would detect the failure and put the funds back into the source account.

Consistency means that the transaction strictly follows all applicable rules (e.g., you can't withdraw funds that don't exist) on any and all data affected. *Isolation* means that if transactions are allowed to happen in parallel (which most of them are), then they will be isolated from each other so that the effects of one don't corrupt another. In other words, isolated transactions have the same effect whether they happen in parallel or one after the other. Finally, *durability* is the property that ensures that a completed transaction is permanently stored (for instance, in nonvolatile memory) so that it cannot be wiped by a power outage or other such failure.

Securing database systems mainly requires the same steps we listed for securing server-based systems. However, databases introduce two unique security issues you need to consider: aggregation and inference. *Aggregation* happens when a user does not have the clearance or permission to access specific information but she does have the permission to access components of this information. She can then figure out the rest and obtain restricted information. She can learn of information from different sources and combine it to learn something she does not have the clearance to know.

The following is a silly conceptual example. Let's say a database administrator does not want anyone in the Users group to be able to figure out a specific sentence, so he segregates the sentence into components and restricts the Users group from accessing it, as represented in Figure 7-2. However, Emily can access components A, C, and F. Because she is particularly bright, she figures out the sentence and now knows the restricted secret.

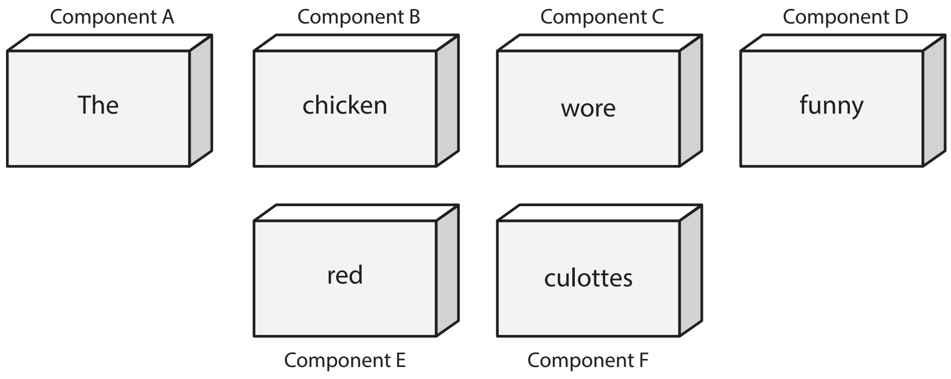


Figure 7-2 Because Emily has access to components A, C, and F, she can figure out the secret sentence through aggregation.

To prevent aggregation, the subject, and any application or process acting on the subject's behalf, needs to be prevented from gaining access to the whole collection, including the independent components. The objects can be placed into containers, which are classified at a higher level to prevent access from subjects with lower-level permissions or clearances. A subject's queries can also be tracked, and context-dependent access control can be enforced. This would keep a history of the objects that a subject has accessed and restrict an access attempt if there is an indication that an aggregation attack is underway.

EXAM TIP *Aggregation* is the act of combining information from separate sources. The combination of the data forms new information, which the subject does not have the necessary rights to access. The combined information has a sensitivity that is greater than that of the individual parts.

The other security issue is *inference*, which is the intended result of aggregation. The inference problem happens when a subject deduces the full story from the pieces he learned of through aggregation. This is seen when data at a lower security level indirectly portrays data at a higher level.

EXAM TIP *Inference* is the ability to derive information not explicitly available.

For example, if a clerk were restricted from knowing the planned movements of troops based in a specific country but did have access to food shipment requirement forms and tent allocation documents, he could figure out that the troops were moving to a specific place because that is where the food and tents are being shipped. The food shipment and tent allocation documents were classified as confidential, and the troop movement was classified as top secret. Because of the varying classifications, the clerk could access and ascertain top-secret information he was not supposed to know.

The trick is to prevent the subject, or any application or process acting on behalf of that subject, from indirectly gaining access to the inferable information. This problem is usually dealt with in the development of the database by implementing content- and context-dependent access control rules. *Content-dependent access control* is based on the sensitivity of the data. The more sensitive the data, the smaller the subset of individuals who can gain access to the data.

Context-dependent access control means that the software "understands" what actions should be allowed based upon the state and sequence of the request. So what does that mean? It means the software must keep track of previous access attempts by the user and understand what sequences of access steps are allowed. Content-dependent access control can go like this: "Does Julio have access to File A?" The system reviews the ACL on File A and returns with a response of "Yes, Julio can access the file, but can only read it." In a context-dependent access control situation, it would be more like this: "Does Julio have access to File A?" The system then reviews several pieces of data: What other access attempts has Julio made? Is this request out of sequence of how a safe series of requests takes place? Does this request fall within the allowed time period of system access (8 A.M. to 5 P.M.)? If the answers to all of these questions are within a set of preconfigured parameters, Julio can access the file. If not, he can't.

If context-dependent access control is being used to protect against inference attacks, the database software would need to keep track of what the user is requesting. So Julio makes a request to see field 1, then field 5, then field 20, which the system allows, but once he asks to see field 15, the database does not allow this access attempt. The software must be preprogrammed (usually through a rule-based engine) as to what sequence and how much data Julio is allowed to view. If he is allowed to view more information, he may have enough data to infer something we don't want him to know.

Obviously, content-dependent access control is not as complex as context-dependent access control because of the number of items that need to be processed by the system.

Some other common attempts to prevent inference attacks are cell suppression, partitioning the database, and noise and perturbation. *Cell suppression* is a technique used to hide specific cells that contain information that could be used in inference attacks. *Partitioning* the database involves dividing the database into different parts, which makes it much harder for an unauthorized individual to find connecting pieces of data that can be brought together and other information that can be deduced or uncovered. *Noise and perturbation* is a technique of inserting bogus information in the hopes of misdirecting an attacker or confusing the matter enough that the actual attack will not be fruitful.

Often, security is not integrated into the planning and development of a database. Security is an afterthought, and a trusted front end is developed to be used with the database instead. This approach is limited in the granularity of security and in the types of security functions that can take place.

A common theme in security is a balance between effective security and functionality. In many cases, the more you secure something, the less functionality you have. Although this could be the desired result, it is important not to excessively impede user productivity when security is being introduced.

High-Performance Computing Systems

All the architectures we've discussed so far in this chapter support significant amounts of computing. From high-end workstations used for high-resolution video processing to massive worldwide e-commerce sites supporting hundreds of millions of transactions per day, the power available to these systems today is very impressive indeed. As we will see shortly, the use of highly scalable cloud services can help turbo-charge these architectures, too. But what happens when even that is not enough? That's when we have to abandon these architectures and go for something altogether different.

High-performance computing (HPC) is the aggregation of computing power in ways that exceed the capabilities of general-purpose computers for the specific purpose of solving large problems. You may have already encountered this architecture if you've read about supercomputers. These are devices whose performance is so optimized that, even with electrons traveling at close to the speed of light down their wires, engineers spend significant design effort to make those wires even a few inches shorter. This is partially achieved by dividing the thousands (or tens of thousands) of processors in a typical system into tightly packed clusters, each with its own high-speed storage devices. Large problems can be broken down into individual jobs and assigned to the different clusters by a central scheduler. Once these smaller jobs are completed, they are progressively put together with other jobs (which, in turn, would be a job) until the final answer is computed.

While it may seem that most of us will seldom (if ever) work with HPC, the move toward big data analytics will probably drive us there sooner rather than later. For this reason, we need to be at least aware of some of the biggest security challenges with HPC. The first one is, quite simply, the very purpose of HPC's existence: efficiency. Large organizations spend millions of dollars building these custom systems for the purpose of crunching numbers really fast. Security tends to slow down (at least a little) just about everything, so we're already fighting an uphill battle. Fortunately, the very fact that HPC systems are so expensive and esoteric can help us justify the first rule for securing them, which is to put them in their own isolated enclave. Complete isolation is probably infeasible in many cases because raw data must flow in and solutions must flow out at some point. The goal would be to identify exactly how those flows should happen and then force them through a few gateways that can restrict who can communicate with the HPC system and under what conditions.

Another way in which HPC systems actually help us secure them is by following some very specific patterns of behavior during normal operations: jobs come in to the schedulers, which then assign them to specific clusters, which then return results in a specific format. Apart from some housekeeping functions, that's pretty much all that happens in an HPC system. It just happens *a lot*! These predictable patterns mean that anomaly detection is much easier than in a typical IT environment with thousands of users each doing their own thing.

Finally, since performance is so critical to HPC, most attacks are likely to affect it in noticeable ways. For this reason, simply monitoring the performance of the system will probably reveal nefarious activities. This noticeable impact on performance, as we will see shortly, affects other, less-esoteric systems, like those that control our factories, refineries, and electric grids.

Industrial Control Systems

Industrial control systems (ICS) consist of information technology that is specifically designed to control physical devices in industrial processes. ICS exist on factory floors to control conveyor belts and industrial robots. They exist in the power and water infrastructures to control the flows of these utilities. Because, unlike the majority of other IT systems, ICS control things that can directly cause physical harm to humans, safety must be paramount in operating and securing them. Another important consideration is that, due to the roles these systems typically fulfill in manufacturing and infrastructure, maintaining their "uptime" or availability is critical. For these two reasons (safety and availability), securing ICS requires a slightly different approach than that used to secure traditional IT systems.

EXAM TIP Safety is the paramount concern in operating and securing industrial control systems.

The term industrial control system actually is an umbrella term covering a number of somewhat different technologies that were developed independently to solve different problems. The term encompasses programmable logic controllers (PLCs) that open or close valves, remote terminal units (RTUs) that relay readings and execute commands, and specialized databases called data historians that capture all process data for analysis. ICS, with all its technologies, protocols, and devices, can generally be divided into two solution spaces:

- Controlling physical processes that take place in a (more or less) local area. This involves what are called distributed control systems (DCS).
- Controlling processes that take place at multiple sites separated by significant distances. This is addressed through supervisory control and data acquisition (SCADA).

We'll delve into both of these solution spaces shortly.

NOTE A good resource for ensuring ICS safety, security, and availability is NIST Special Publication 800-82, Revision 2, *Guide to Industrial Control Systems (ICS) Security*, discussed further later in this section.

Another umbrella term you may see is *operational technology (OT)*, which includes both ICS and some traditional IT systems that are needed to make sure all the ICS devices can talk to each other. Figure 7-3 shows the relationship between these terms. Note that there is overlap between DCS and SCADA, in this case shown by the PLC, which supports both types of systems. Before we discuss each of the two major categories of ICS, let's take a quick look at some of the devices, like PLCs, that are needed to make these systems work.

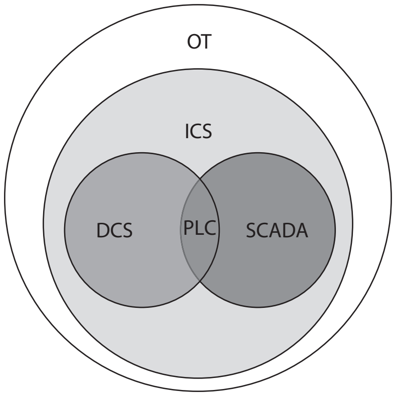


Figure 7-3 Relationship between OT terms

Devices

There are a lot of different types of devices in use in OT systems. Increasingly, the lines between these types are blurred as different features converge in newer devices. However, most OT environments will have PLCs, a human-machine interface (HMI), and a data historian, which we describe in the following sections. Please note that you don't need to memorize what any of the following devices do in order to pass the CISSP exam. However, being familiar with them will help you understand the security implications of ICS and how OT and IT systems intertwine in the real world.

Programmable Logic Controller

When automation (the physical kind, not the computing kind to which we're accustomed) first showed up on factory floors, it was bulky, brittle, and difficult to maintain. If, for instance, you wanted an automatic hammer to drive nails into boxes moving through a conveyor belt, you would arrange a series of electrical relays such that they would sequentially actuate the hammer, retrieve it, and then wait for the next box. Whenever you wanted to change your process or repurpose the hammer, you would have to suffer through a complex and error-prone reconfiguration process.

Programmable logic controllers (PLCs) are computers designed to control electromechanical processes such as assembly lines, elevators, roller coasters, and nuclear centrifuges. The idea is that a PLC can be used in one application today and then easily reprogrammed to control something else tomorrow. PLCs normally connect to the devices they control over a standard serial interface such as RS-232, and to the devices that control them over Ethernet cables. The communications protocols themselves, however, are not always standard. The dominant protocols are Modbus and EtherNet/IP, but this is not universal. While this lack of universality in communications protocols creates additional challenges to securing PLCs, we are seeing a trend toward standardization of these serial connection protocols. This is particularly important because, while early PLCs had limited or no network connectivity, it is now rare to see a PLC that is not network-enabled.

PLCs can present some tough security challenges. Unlike the IT devices with which many of us are more familiar, these OT devices tend to have very long lifetimes. It's not unusual for production systems to include PLCs that are ten years old or older. Depending on how the ICS was architected, it may be difficult to update or patch the PLCs. When you couple this difficulty with the risk of causing downtime to a critical industrial process, you may understand why some PLCs can go years without getting patched. To make things worse, we've seen plenty of PLCs using factory default passwords that are well documented. While modern PLCs come with better security features, odds are that an OT environment will have some legacy controllers hiding somewhere. The best thing to do is to ensure that all PLC network segments are strictly isolated from all nonessential devices and are monitored closely for anomalous traffic.

Human-Machine Interface

A *human-machine interface (HMI)* is usually a regular workstation running a proprietary supervisory system that allows operators to monitor and control an ICS. An HMI normally has a dashboard that shows a diagram or schematic of the system being controlled,

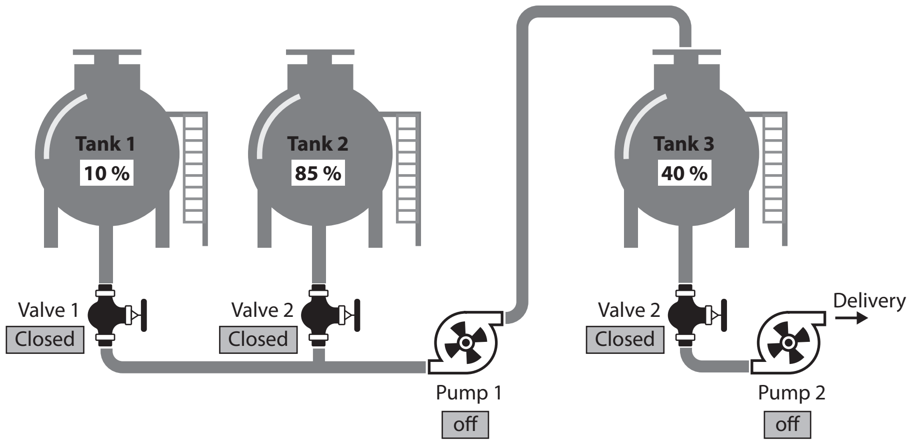


Figure 7-4 A simplified HMI screen

the readings from whatever sensors the system has in place, and buttons with which to control your actuators. Figure 7-4 shows a simplified HMI screen for a small fuel distribution system. Each of the three tanks shows how much fuel it contains. Three valves control the flow of fuel between the tanks, and all three are closed. If the operator wanted to move fuel around, she would simply click the CLOSED button, it would change to OPEN, and the fuel would be free to move. Similarly, clicking the OFF button on the pumps would turn them on to actually move the fuel around.

Another feature of HMIs is alarm monitoring. Each sensor (like those monitoring tank levels in our example) can be configured to alarm if certain values are reached. This is particularly important when it comes to the pressure in a pipeline, the temperature in a tank, or the load on a power line. HMIs usually include automation features that can automatically instruct PLCs to take certain actions when alarm conditions are met, such as tripping breakers when loads are too high.

HMIs simplify the myriad of details that make the ICS work so that the operators are not overwhelmed. In the simple example in Figure 7-4, Pump 1 would typically have a safety feature that would prevent it from being open unless Valve 1 and/or Valve 2 were open and the capacity in Tank 3 was not 100 percent. These features are manually programmed by the plant staff when the system is installed and are periodically audited for safety. Keep in mind that safety is of even more importance than security in OT environments.

Technically, securing an HMI is mostly the same as securing any IT system. Keep in mind that this is normally just a regular workstation that just happens to be running this proprietary piece of software. The challenge is that, because HMIs are part of mission-critical industrial systems where safety and efficiency are paramount, there can be significant resistance from OT staff to making any changes or taking any actions that can compromise either of these imperatives. These actions, of course, could include the typical security measures such as installing endpoint detection and response (EDR) systems, scanning them for vulnerabilities, conducting penetration tests, or even mandating unique credentials for each user with strong authentication. (Imagine what could happen if the HMI is locked, there is an emergency, and the logged-in user is on a break.)

Data Historian

As the name suggests, a *data historian* is a data repository that keeps a history of everything seen in the ICS. This includes all sensor values, alarms, and commands issued, all of which are timestamped. A data historian can communicate directly with other ICS devices, such as PLCs and HMIs. Sometimes, a data historian is embedded with (or at least running on the same workstation as) the HMI. Most OT environments, however, have a dedicated data historian (apart from the HMI) in a different network segment. The main reason for this is that this device usually communicates with enterprise IT systems for planning and accounting purposes. For example, the data historian in our fuel system example would provide data on how much fuel was delivered out of Tank 3.

One of the key challenges in securing the data historian stems from the fact that it frequently has to talk to both PLCs (and similar devices) and enterprise IT systems (e.g., for accounting purposes). A best practice when this is required is to put the data historian in a specially hardened network segment like a demilitarized zone (DMZ) and implement restrictive ACLs to ensure unidirectional traffic from the PLCs to the historian and from the historian to the enterprise IT systems. This can be done using a traditional firewall (or even a router), but some organizations instead use specialized devices called *data diodes*, which are security hardened and permit traffic to flow only in one direction.

Distributed Control System

A *distributed control system (DCS)* is a network of control devices within fairly close proximity that are part of one or more industrial processes. DCS usage is very common in manufacturing plants, oil refineries, and power plants, and is characterized by decisions being made in a concerted manner, but by different nodes within the system.

You can think of a DCS as a hierarchy of devices. At the bottom level, you will find the physical devices that are being controlled or that provide inputs to the system. One level up, you will find the microcontrollers and PLCs that directly interact with the physical devices but also communicate with higher-level controllers. Above the PLCs are the supervisory computers that control, for example, a given production line. You can also have a higher level that deals with plant-wide controls, which would require some coordination among different production lines.

As you can see, the concept of a DCS was born from the need to control fairly localized physical processes. Because of this, the communications protocols in use are not optimized for wide-area communications or for security. Another byproduct of this localized approach is that DCS users felt for many years that all they needed to do to secure their systems was to provide physical security. If the bad guys can't get into the plant, it was thought, then they can't break our systems. This is because, typically, a DCS consists of devices within the same plant. However, technological advances and converging technologies are blurring the line between a DCS and a SCADA system.

Supervisory Control and Data Acquisition

While DCS technology is well suited for local processes such as those in a manufacturing plant, it was never intended to operate across great distances. The *supervisory control and data acquisition (SCADA)* systems were developed to control large-scale physical processes involving nodes separated by significant distances. The main conceptual differences between DCS and SCADA are size and distances. So, while the control of a power plant is perfectly suited for a traditional DCS, the distribution of the generated power across a power grid would require a SCADA system.

SCADA systems typically involve three kinds of devices: endpoints, backends, and user stations. A *remote terminal unit (RTU)* is an endpoint that connects directly to sensors and/or actuators. Though there are still plenty of RTUs in use, many RTUs have been replaced with PLCs. The *data acquisition servers (DAS)* are backends that receive all data from the endpoints through a telemetry system and perform whatever correlation or analysis may be necessary. Finally, the users in charge of controlling the system interact with it through the use of the previously introduced *human-machine interface (HMI)*, the user station that displays the data from the endpoints and allows the users to issue commands to the actuators (e.g., to close a valve or open a switch).

One of the main challenges with operating at great distances is effective communications, particularly when parts of the process occur in areas with limited, spotty, or nonexistent telecommunications infrastructures. SCADA systems commonly use dedicated cables and radio links to cover these large expanses. Many legacy SCADA implementations rely on older proprietary communications protocols and devices. For many years, this led this community to feel secure because only someone with detailed knowledge of an obscure protocol and access to specialized communications gear could compromise the system. In part, this assumption is one of the causes of the lack of effective security controls on legacy SCADA communications. While this thinking may have been arguable in the past, today's convergence on IP-based protocols makes it clear that this is not a secure way of doing business.

ICS Security

The single greatest vulnerability in ICS is their increasing connectivity to traditional IT networks. This has two notable side effects: it accelerates convergence toward standard protocols, and it exposes once-private systems to anyone with an Internet connection. NIST SP 800-82 Rev. 2 has a variety of recommendations for ICS security, but we highlight some of the most important ones here:

- Apply a risk management process to ICS.
- Segment the network to place IDS/IPS at the subnet boundaries.
- Disable unneeded ports and services on all ICS devices.
- Implement least privilege through the ICS.
- Use encryption wherever feasible.
- Ensure there is a process for patch management.
- Monitor audit trails regularly.

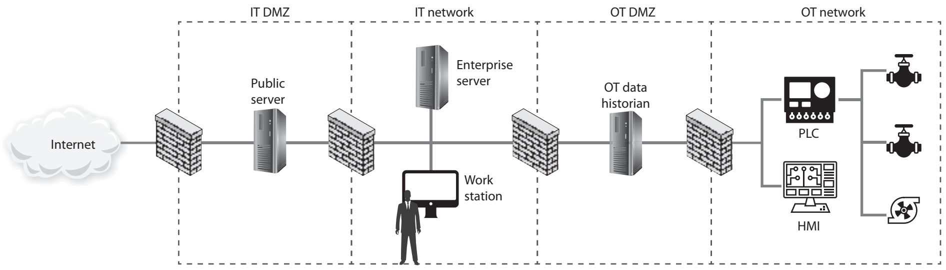


Figure 7-5 A simplified IT/OT environment

Let's look at a concrete (if seriously simplified) example in Figure 7-5. We're only showing a handful of IT and OT devices, but the zones are representative of a real environment. Starting from the right, you see the valves and pumps that are controlled by the PLC in the OT network. The PLC is directly connected to the HMI so that the PLC can be monitored and controlled by the operator. Both the PLC and the HMI are also connected (through a firewall) to the OT data historian in the OT DMZ. This is so that everything that happens in the OT network can be logged and analyzed. The OT data historian can also communicate with the enterprise server in the IT network to pass whatever data is required for planning, accounting, auditing, and reporting. If a user, say, in the accounting department, wants any of this data, he would get it from the enterprise server and would not be able to connect directly to the OT data historian. If a customer wanted to check via the Internet how much fuel they've been dispensed, they would log into their portal on the public server and that device would query the enterprise server for the relevant data.

Note that each segment is protected by a firewall (or data diode) that allows only specific devices in the next zone to connect in very restrictive ways to get only specific data. No device should ever be able to connect any further than one segment to the left or right.

Network segmentation also helps mitigate one of the common risks in many OT environments: unpatched devices. It is not rare to find devices that have been operating unpatched for several years. There are many reasons for this. First, ICS devices have very long shelf lives. They can remain in use for a decade or longer and may no longer receive updates from the manufacturer. They can also be very expensive, which means organizations may be unwilling or unable to set up a separate laboratory in which to test patches to ensure they don't cause unanticipated effects on the production systems. While this is a pretty standard practice in IT environments, it is pretty rare in the OT world. Without prior testing, patches could cause outages or safety issues and, as we know, maintaining availability and ensuring safety are the two imperatives of the OT world.

So, it is not all that strange for us to have to live with unpatched devices. The solution is to isolate them as best as we can. At a very minimum, it should be impossible for ICS devices to be reachable from the Internet. Better yet, we control access strictly from one zone to the next, as discussed previously. But for unpatched control devices, we have to be extremely paranoid and surround them with protective barriers that are monitored continuously.

EXAM TIP The most important principle in defending OT systems is to isolate them from the public Internet, either logically or physically.

Virtualized Systems

If you have been into computers for a while, you might remember computer games that did not have the complex, lifelike graphics of today's games. *Pong* and *Asteroids* were what we had to play with when we were younger. In those simpler times, the games were 16-bit and were written to work in a 16-bit MS-DOS environment. When our Windows operating systems moved from 16-bit to 32-bit, the 32-bit operating systems were written to be backward compatible, so someone could still load and play a 16-bit game in an environment that the game did not understand. The continuation of this little life pleasure was available to users because the OSs created virtual environments for the games to run in. Backward compatibility was also introduced with 64-bit OSs.

When a 32-bit application needs to interact with a 64-bit OS, it has been developed to make system calls and interact with the computer's memory in a way that would only work within a 32-bit OS—not a 64-bit system. So, the virtual environment simulates a 32-bit OS, and when the application makes a request, the OS converts the 32-bit request into a 64-bit request (this is called *thunking*) and reacts to the request appropriately. When the system sends a reply to this request, it changes the 64-bit reply into a 32-bit reply so the application understands it.

Today, virtual environments are much more advanced. *Virtualized systems* are those that exist in software-simulated environments. In our previous example of *Pong*, the 16-bit game "thinks" it is running on a 16-bit computer when in fact this is an illusion created by a layer of virtualizing software. In this case, the virtualized system was developed to provide backward compatibility. In many other cases, virtualization allows us to run multiple services or even full computers simultaneously on the same hardware, greatly enhancing resource (e.g., memory, processor) utilization, reducing operating costs, and even providing improved security, among other benefits.

Virtual Machines

Virtual machines (VMs) are entire computer systems that reside inside a virtualized environment. This means that you could have a legitimate Windows workstation running within a Linux server, complete with automatic updates from Microsoft, licensed apps from any vendor, and performance that is virtually indistinguishable (pun intended) from a similar Windows system running on "bare metal." This VM is commonly referred to as a *guest* that is executed in the *host* environment, which, in our example, would be the Linux server.

Virtualization allows a single host environment to execute multiple guests at once, with multiple VMs dynamically pooling resources from a common physical system. Computer resources such as RAM, processors, and storage are emulated through the host environment. The VMs do not directly access these resources; instead, they communicate with a *hypervisor* within the host environment, which is responsible for managing system resources. The hypervisor is the central program that controls the execution of the various guest operating

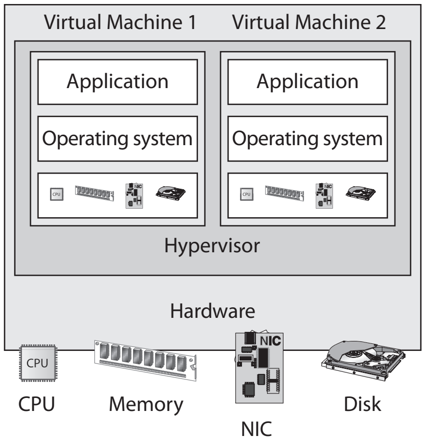


Figure 7-6 The hypervisor controls virtual machine instances.

systems and provides the abstraction level between the guest and host environments, as shown in Figure 7-6.

There are two types of hypervisors. A *type 1 hypervisor* runs directly on hardware or "bare metal" and manages access to it by its VMs. This is the sort of setup we use in server rooms and cloud environments. Examples of type 1 hypervisors are Citrix/Xen Server and VMware ESXi. A *type 2 hypervisor*, on the other hand, runs as an application on an OS. This allows users, for example, to host a Windows VM in their macOS computer. Type 2 hypervisors are commonly used by developers and security researchers to test their work in a controlled environment or use applications that are not available for the host OS. Examples of type 2 hypervisors are Oracle VM VirtualBox and VMware Workstation.

Hypervisors allow you to have one computer running several different operating systems at one time. For example, you can run a system with Windows 10, Linux, and Windows 2016 on one computer. Think of a house that has different rooms. Each OS gets its own room, but each shares the same resources that the house provides—a foundation, electricity, water, roof, and so on. An OS that is "living" in a specific room does not need to know about or interact with another OS in another room to take advantage of the resources provided by the house. The same concept happens in a computer: Each OS shares the resources provided by the physical system (memory, processor, buses, and so on). The OSs "live" and work in their own "rooms," which are the guest VMs. The physical computer itself is the host.

Why would we want to virtualize our machines? One reason is that it is cheaper than having a full physical system for each and every operating system. If they can all live on one system and share the same physical resources, your costs are reduced immensely. This is the same reason people get roommates. The rent can be split among different people, and all can share the same house and resources. Another reason to use virtualization is security. Providing to each OS its own "clean" environment to work within reduces the possibility of the various OSs negatively interacting with each other.

Furthermore, since every aspect of the virtual machine, including the contents of its disk drives and even its memory, is stored as files within the host, restoring a backup is a snap.

All you have to do is drop the set of backed-up files onto a new hypervisor and you will instantly restore a VM to whatever state it was in when the backup was made. Contrast this with having to rebuild a physical computer from backups, which can take a lot longer.

On the flip side of security, any vulnerability in the hypervisor would give an attacker unparalleled and virtually undetectable (pun not intended) power to compromise the confidentiality, integrity, or availability of VMs running on it. This is not a hypothetical scenario, as both VirtualBox and VMware have reported (and patched) such vulnerabilities in recent years. The takeaway from these discoveries is that we should assume that any component of an information system could be compromised and ask ourselves the questions "how would I detect it?" and "how can I mitigate it?"

Containerization

As virtualization matured, a new branch called *containerization* emerged. A container is an application that runs in its own isolated user space. Whereas virtual machines have their own complete operating systems running on top of hypervisors and share the resources provided by the bare metal, containers sit on top of OSs and share the resources provided by the host OS. Instead of abstracting the hardware for guest OSs, container software abstracts the kernel of the OS for the applications running above it. This allows for low overhead in running many applications and improved speed in deploying instances, because a whole VM doesn't have to be started for every application. Rather, the application, services, processes, libraries, and any other dependencies can be wrapped up into one unit.

Additionally, each container operates in a sandbox, with the only means to interact being through the user interface or application programming interface (API) calls. The big names to know in this space are Docker on the commercial side and Kubernetes as the open-source alternative. Containers have enabled rapid development operations because developers can test their code more quickly, changing only the components necessary in the container and then redeploying.

Securing containers requires a different approach than we'd take with full-sized VMs. Obviously, we want to harden the host OS. But we also need to pay attention to each container and the manner in which it interacts with clients and other containers. Keep in mind that containers are frequently used in rapid development. This means that, unless you build secure development right into the development team, you will likely end up with insecure code. We'll address the integration of development, security, and operations staff when we discuss DevSecOps in Chapters 24 and 25, but for now remember that it's really difficult to secure containers that have been developed insecurely.

NIST offers some excellent specific guidance on securing containers in NIST SP 800-190, *Application Container Security Guide*. Among the most important recommendations in that publication are the following:

- Use container-specific host OSs instead of general-purpose ones to reduce attack surfaces.
- Only group containers with the same purpose, sensitivity, and threat posture on a single host OS kernel to allow for additional defense in depth.

- Adopt container-specific vulnerability management tools and processes for images to prevent compromises.
- Use container-aware runtime defense tools such as intrusion prevention systems.

Microservices

A common use of containers is to host *microservices*, which is a way of developing software where, rather than building one large enterprise application, the functionality is divided into multiple smaller components that, working together in a distributed manner, implement all the needed features. Think of it as a software development version of the old "divide and conquer" approach. Microservices are considered an architectural style rather than a standard, but there is broad consensus that they consist of small, decentralized, individually deployable services built around business capabilities. They also tend to be *loosely coupled*, which means there aren't a lot of dependencies between the individual services. As a result, microservices are quick to develop, test, and deploy and can be exchanged without breaking the larger system. For many business applications, microservices are also more efficient and scalable than monolithic server-based architectures.

NOTE Containers and microservices don't have to be used together. It's just very common to do so.

The decentralization of microservices can present a security challenge. How can you track adversarial behaviors through a system of microservices, where each service does one discrete task? The answer is *log aggregation*. Whereas microservices are decentralized, we want to log them in a centralized fashion so we can look for patterns that span multiple services and can point to malicious intent. Admittedly, you will need automation and perhaps data analytics or artificial intelligence to detect these malicious events, but you won't have a chance at spotting them unless you aggregate the logs.

Serverless

If we gain efficiency and scalability by breaking up a big service into a bunch of microservices, can we gain even more by breaking up the microservices further? The answer, in many cases, is yes, because hosting a service (even a micro one) means that you have to provision, manage, update, and run the thing. So, if we're going to go further down this road of dividing and conquering, the next level of granularity is individual functions.

Hosting a service usually means setting up hardware, provisioning and managing servers, defining load management mechanisms, setting up requirements, and running the service. In a *serverless* architecture, the services offered to end users, such as compute, storage, or messaging, along with their required configuration and management, can be performed without a requirement from the user to set up any server infrastructure. The focus is strictly at the individual function level. These serverless models are designed primarily for massive scaling and high availability. Additionally, from a cost perspective, they are attractive, because billing occurs based on what cycles are actually used versus what is provisioned in advance.

Integrating security mechanisms into serverless models is not as simple as ensuring that the underlying technologies are hardened. Because visibility into host infrastructure operations is limited, implementing countermeasures for remote code execution or modifying access control lists isn't as straightforward as it would be with traditional server design. In the serverless model, security analysts are usually restricted to applying controls at the application or function level and then keeping a close eye on network traffic.

As you probably know by now, serverless architectures rely on the capability to automatically and securely provision, run, and then deprovision computing resources on demand. This capability undergirds their economic promise: you only pay for exactly the computing you need to perform just the functions that are required, and not a penny more. It is also essential to meet the arbitrary scalability of serverless systems. This capability is characteristic of cloud computing.

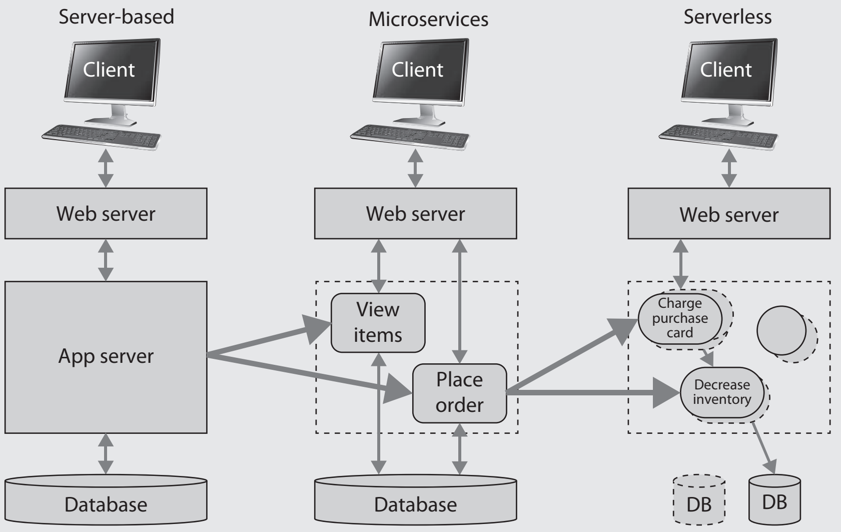
Comparing Server-Based, Microservice, and Serverless Architectures

A typical service houses a bunch of functions within it. Think of a very simple e-commerce web application server. It allows customers to log in, view the items that are for sale, and place orders. When placing an order, the server invokes a multitude of functions. For instance, it may have to charge the payment card, decrease inventory, schedule a shipment, and send a confirmation message. Here's how each of these three architectures handle this.

Server-based implementations provide all services (and their component functions) in the same physical or virtual server that houses the monolithic web application. The server must always be available (meaning powered on and connected to the Internet). If there's a sudden spike in orders, you better hope you have enough bandwidth, memory, and processing power to handle it. If you don't, you get to build a new server from scratch and either replace the original server with a beefier one or load-balance between the two. Either way, you now have more infrastructure to keep up and running.

Microservices can be created for each of the major features in the web application: view items and place orders. Each microservice lives in its own container and gets called as needed. If you see that spike in orders, you deploy a new container (in seconds), perhaps in a different host, and can destroy it when you no longer need it. Sure, you'll need some supervisory process to figure out when and how to spin up new containers, but at least you can dynamically respond to increased demands.

Serverless approaches would decompose each service into its fundamental functions and then dynamically provision those functions as needed. In other words, there is never a big web application server (like in the server-based approach) or even a microservice for order processing that is up and running. Instead, the charge_payment_card function is invoked in whatever infrastructure is available whenever a card needs to be processed. If that function is successful, it invokes the decrease_inventory function, again, in whatever infrastructure is available, and so on. After each function terminates, it simply evaporates so that no more resources are consumed than are absolutely needed. If there's a sudden spike in demand, the orchestrator spins up whatever additional resources are needed to run as many functions as are required.



Cloud-Based Systems

If you were asked to install a brand-new server room for your organization, you would probably have to clear your calendar for weeks (or longer) to address the many tasks that would be involved. From power and environmental controls to hardware acquisition, installation, and configuration to software builds, the list is long and full of headaches. Now, imagine that you can provision all the needed servers in minutes using a simple graphical interface or a short script and that you can get rid of them just as quickly when you no longer need them. This is one of the benefits of cloud computing.

Cloud computing is the use of shared, remote computing devices for the purpose of providing improved efficiencies, performance, reliability, scalability, and security. These devices are usually based on virtual machines running on shared infrastructure and can be outsourced to a third-party cloud service provider (CSP) on a *public cloud* or provided in-house on a *private cloud*. If you don't feel comfortable sharing infrastructure with random strangers (though this is done securely), there is also a *virtual private cloud (VPC)* model in which you get your own walled garden inside an otherwise public cloud.

Generally speaking, there are three models for cloud computing services:

- **Software as a Service (SaaS)** The user of SaaS is allowed to use a specific application that executes on the CSP's environment. Examples of SaaS are Microsoft 365 and Google Apps, which you use via a web interface but someone else provisions and maintains everything for you.
- **Platform as a Service (PaaS)** In this model, the user gets access to a computing platform that is typically built on a server operating system. An example of this would be spawning an instance of Windows Server 2019 to provide a web server. The CSP is normally responsible for configuring and securing the platform, however, so the user normally doesn't get administrative privileges over the entire platform.
- **Infrastructure as a Service (IaaS)** If you want full, unfettered access to (and responsibility for securing) a cloud-based VM, you would want to use the IaaS model. Following up on the previous example, this would allow you to manage the patching of the Windows Server 2019 instance. The catch is that the CSP has no responsibility for security; it's all on you.

If you are a user of IaaS, you probably won't do things too differently than you already do to secure your systems. The only exception is that you wouldn't have physical access to the computers if a CSP hosts them. If, on the other hand, you use SaaS or PaaS, the security of your systems will almost always rely on the policies and contracts that you put into place. The policies will dictate how your users interact with the cloud services. This would include the information classification levels that would be allowed on those services, terms of use, and other policies. The contracts will specify the quality of service and what the CSP will do with or for you in responding to security events.

CAUTION It is imperative that you carefully review the terms of service when evaluating a potential contract for cloud services and consider them in the context of your organization's security. Though the industry is getting better all the time, security provisions are oftentimes lacking in these contracts at this time.

Software as a Service

SaaS is pervasively used by most enterprises. According to some estimates, the average company uses nearly 2,000 unique cloud services for everything from writing memos to managing their sales pipeline. The whole idea is that, apart from a fairly small amount of allowed customization, you just pay for the licenses and the vendor takes care of making sure all your users have access to the software, regardless of where they are.

Given the popularity of SaaS solutions, cloud service providers such as Microsoft, Amazon, Cisco, and Google often dedicate large teams to securing all aspects of their service infrastructure. Increasingly, however, most security incidents involving SaaS occur at the data-handling level, where these infrastructure companies do not have the responsibility or visibility required to take action. For example, how could the CSP be held liable when one of your employees shares a confidential file with an unauthorized third party?

So, visibility is one of our main concerns as security professionals when it comes to SaaS. Do you know what assets you have and how they are being used? The "McAfee 2019 Cloud Adoption and Risk Report" describes the disconnect between the number of cloud services that organizations believe are being accessed by their users and the number of cloud services that are actually being accessed. The discrepancy, according to the report, can be several orders of magnitude. As we have mentioned before, you can't protect what you don't know you have. This is where solutions like cloud access security brokers (CASBs) and data loss prevention (DLP) systems can come in very handy.

NOTE We already covered CASBs and DLP systems in Chapter 6.

Platform as a Service

What if, instead of licensing someone else's application, you have developed your own and need a place to host it for your users? You'd want to have a fair amount of flexibility in terms of configuring the hosting environment, but you probably could use some help in terms of provisioning and securing it. You can secure the app, for sure, but would like someone else to take care of things like hardening the host, patching the underlying OS, and maybe even monitoring access to the VM. This is where PaaS comes in.

PaaS has a similar set of functionalities as SaaS and provides many of the same benefits in that the CSP manages the foundational technologies of the stack in a manner transparent to the end user. You simply tell your provider, "I'd like a Windows Server 2019 with 64 gigabytes of RAM and eight cores," and, voilà, there it is. You get direct access to a development or deployment environment that enables you to build and host your own solutions on a cloud-based infrastructure without having to build your own infrastructure. PaaS solutions, therefore, are optimized to provide value focused on software development. PaaS, by its very nature, is designed to provide an organization with tools that interact directly with what may be its most important asset: its source code.

At the physical infrastructure, in PaaS, service providers assume the responsibility of maintenance and protection and employ a number of methods to deter successful exploits at this level. This often means PaaS providers require trusted sources for hardware, use strong physical security for its data centers, and monitor access to the physical servers and connections to and from them. Additionally, PaaS providers often enhance their protection against distributed denial-of-service (DDoS) attacks using network-based technologies that require no additional configuration from the user.

While the PaaS model makes a lot of provisioning, maintenance, and security problems go away for you, it is worth noting that it does nothing to protect the software systems you host there. If you build and deploy insecure code, there is very little your CSP will be able to do to keep it protected. PaaS providers focus on the infrastructure on which the service runs, but you still have to ensure that the software is secure and the appropriate controls are in place. We'll dive into how to build secure code in Chapters 24 and 25.

Infrastructure as a Service

Sometimes, you just have to roll up your sleeves, get your hands dirty, and build your own servers from the ground up. Maybe the applications and services you have developed require your IT and security teams to install and configure components at the OS level that would not be accessible to you in the PaaS model. You don't need someone to make platforms that they manage available to you; you need to build platforms from the ground up yourself. IaaS gives you just that. You upload an image to the CSP's environment and build your own hosts however you need them.

As a method of efficiently assigning hardware through a process of constant assignment and reclamation, IaaS offers an effective and affordable way for organizations to get all of the benefits of managing their own hardware without incurring the massive overhead costs associated with acquisition, physical storage, and disposal of the hardware. In this service model, the vendor provides the hardware, network, and storage resources necessary for the user to install and maintain any operating system, dependencies, and applications they want. The vendor deals with all hardware issues for you, leaving you to focus on the virtual hosts.

In the IaaS model, the majority of the security controls (apart from physical ones) are your responsibility. Obviously, you want to have a robust security team to manage these. Still, there are some risks that are beyond your control and for which you rely on your vendor, such as any vulnerabilities that could allow an attacker to exploit flaws in hard disks, RAM, CPU caches, and GPUs. One attack scenario affecting IaaS cloud providers could enable a malicious actor to implant persistent back doors for data theft into bare-metal cloud servers. A vulnerability either in the hypervisor supporting the visualization of various tenant systems or in the firmware of the hardware in use could introduce a vector for this attack. This attack would be difficult for the customer to detect because it would be possible for all services to appear unaffected at a higher level of the technology stack.

Though the likelihood of a successful exploit of this kind of vulnerability is quite low, defects and errors at this level may still incur significant costs unrelated to an actual exploit. Take, for example, the 2014 hypervisor update performed by Amazon Web Services (AWS), which essentially forced a complete restart of a major cloud offering, the Elastic Compute Cloud (EC2). In response to the discovery of a critical security flaw in the open-source hypervisor Xen, Amazon forced EC2 instances globally to restart to ensure the patch would take correctly and that customers remained unaffected. In most cases, though, as with many other cloud services, attacks against IaaS environments are possible because of misconfiguration on the customer side.

Everything as a Service

It's worth reviewing the basic premise of cloud service offerings: you save money by only paying for exactly the resources you actually use, while having the capacity to scale those up as much as you need to at a moment's notice. If you think about it, this model can apply to things other than applications and computers. *Everything as a Service (XaaS)* captures the trend to apply the cloud model to a large range of offerings, from entertainment (e.g., television shows and feature-length movies), to cybersecurity (e.g., Security as a Service), to serverless computing environments (e.g., Function as a Service). Get ready for the inevitable barrage of <fill-in-the-blank> as a Service offerings coming your way.

Cloud Deployment Models

By now you may be a big believer in the promise of cloud computing but may be wondering, "Where, exactly, *is* the cloud?" The answer, as in so many questions in our field, is "It depends." There are four common models for deploying cloud computing resources, each with its own features and limitations:

- A *public cloud* is the most prevalent model, in which a vendor like AWS owns all the resources and provides them as a service to all its customers. Importantly, the resources are shared among all customers, albeit in a transparent and secure manner. Public cloud vendors typically also offer a virtual private cloud (VPC) as an option, in which increased isolation between users provides added security.
- A *private cloud* is owned and operated by the organization that uses its services. Here, you own, operate, and maintain the servers, storage, and networking needed to provide the services, which means you don't share resources with anyone. This approach can provide the best security, but the tradeoff might be higher costs and a cap on scalability.
- A *community cloud* is a private cloud that is co-owned (or at least shared) by a specific set of partner organizations. This approach is commonly implemented in large conglomerates where multiple firms report to the same higher-tier headquarters.
- A *hybrid cloud* combines on-premises infrastructure with a public cloud, with a significant effort placed in the management of how data and applications leverage each solution to achieve organizational goals. Organizations that use a hybrid model often derive benefits offered by both public and private models.

Pervasive Systems

Cloud computing is all about the concentration of computing power so that it may be dynamically reallocated among customers. Going in the opposite conceptual direction, *pervasive computing* (also called *ubiquitous computing* or *ubicomp*) is the concept that small (even tiny) amounts of computing power are spread out everywhere and computing is embedded into everyday objects that communicate with each other, often with little or no user interaction, to do very specific things for particular customers. In this model, computers are everywhere and communicate on their own with each other, bringing really cool new features but also really thorny new security challenges.

Embedded Systems

An *embedded system* is a self-contained computer system (that is, it has its own processor, memory, and input/output devices) designed for a very specific purpose. An embedded device is part of (or embedded into) some other mechanical or electrical device or system. Embedded systems typically are cheap, rugged, and small, and they use very little power. They are usually built around *microcontrollers*, which are specialized devices that consist of a CPU, memory, and peripheral control interfaces. Microcontrollers have a very basic operating system, if they have one at all. A digital thermometer is an example of a very simple embedded system; other examples of embedded systems include traffic lights and factory assembly line controllers. As you can see from these examples, embedded systems are frequently used to sense and/or act on a physical environment. For this reason, they are sometimes called *cyber-physical systems*.

The main challenge in securing embedded systems is that of ensuring the security of the software that drives them. Many vendors build their embedded systems around commercially available microprocessors, but they use their own proprietary code that is difficult, if not impossible, for a customer to audit. Depending on the risk tolerance of your organization, this may be acceptable as long as the embedded systems are standalone. The problem, however, is that these systems are increasingly shipping with some sort of network connectivity. For example, some organizations have discovered that some of their embedded devices have "phone home" features that are not documented. In some cases, this has resulted in potentially sensitive information being transmitted to the manufacturer. If a full audit of the embedded device security is not possible, at a very minimum, you should ensure that you see what data flows in and out of it across any network.

Another security issue presented by many embedded systems concerns the ability to update and patch them securely. Many embedded devices are deployed in environments where they have no Internet connectivity. Even if this is not the case and the devices can check for updates, establishing secure communications or verifying digitally signed code, both of which require processor-intensive cryptography, may not be possible on a cheap device.

Internet of Things

The *Internet of Things (IoT)* is the global network of connected embedded systems. What distinguishes the IoT is that each node is connected to the Internet and is uniquely addressable. By some accounts, this network is expected to reach 31 billion devices by 2025, which makes this a booming sector of the global economy. Perhaps the most visible aspect of this explosion is in the area of smart homes in which lights, furnaces, and even refrigerators collaborate to create the best environment for the residents.

With this level of connectivity and access to physical devices, the IoT poses many security challenges. Among the issues to address by anyone considering adoption of IoT devices are the following:

- **Authentication** Embedded devices are not known for incorporating strong authentication support, which is the reason why most IoT devices have very poor (if any) authentication.

- **Encryption** Cryptography is typically expensive in terms of processing power and memory requirements, both of which are very limited in IoT devices. The fallout of this is that data at rest and data in transit can be vulnerable in many parts of the IoT.
- **Updates** Though IoT devices are networked, many vendors in this fast-moving sector do not provide functionality to automatically update their software and firmware when patches are available.

Perhaps the most dramatic illustration to date of what can happen when millions of insecure IoT devices are exploited by an attacker is the Mirai botnet. Mirai is a malware strain that infects IoT devices and was behind one of the largest and most effective botnets in recent history. The Mirai botnet took down major websites via massive DDoS attacks against several sites and service providers using hundreds of thousands of compromised IoT devices. In October 2016, a Mirai attack targeted the popular DNS provider Dyn, which provided name resolution to many popular websites such as Airbnb, Amazon, GitHub, HBO, Netflix, PayPal, Reddit, and Twitter. After taking down Dyn, Mirai left millions of users unable to access these sites for hours.

Distributed Systems

A distributed system is one in which multiple computers work together to do something. The earlier section "Server-Based Systems" already covered a specific example of a four-tier distributed system. It is this collaboration that more generally defines a distributed system. A server-based system is a specific kind of distributed system in which devices in one group (or tier) act as clients for devices in an adjacent group. A tier-1 client cannot work directly with the tier-4 database, as shown earlier in Figure 7-1. We could then say that a *distributed system* is any system in which multiple computing nodes, interconnected by a network, exchange information for the accomplishment of collective tasks.

Not all distributed systems are hierarchical like the example in Figure 7-1. Another approach to distributed computing is found in *peer-to-peer systems*, which are systems in which each node is considered an equal (as opposed to a client or a server) to all others. There is no overarching structure, and nodes are free to request services from any other node. The result is an extremely resilient structure that fares well even when large numbers of nodes become disconnected or otherwise unavailable. If you had a typical client/server model and you lost your server, you'd be down for the count. In a peer-to-peer system, you could lose multiple nodes and still be able to accomplish whatever task you needed to. Clearly, not every application lends itself to this model, because some tasks are inherently hierarchical or centralized. Popular examples of peer-to-peer systems are file sharing systems like BitTorrent, anonymizing networks like The Onion Router (TOR), and cryptocurrencies like bitcoin.

One of the most important issues in securing distributed systems is network communications, which are essential to these systems. While the obvious approach would be to encrypt all traffic, it can be challenging to ensure all nodes are using cryptography that is robust enough to mitigate attacks. This is particularly true when the system includes IoT or OT components that may not have the same crypto capabilities as traditional computers.

Even if you encrypt all traffic (and you really should) in a distributed system, there's still the issue of trust. How do we ensure that every user and every node is trustworthy? How could you tell if part of the system was compromised? Identity and access management is another key area to address, as is the ability to isolate users or nodes from the system should they become compromised.

NOTE We will discuss identity and access management (IAM) in Chapter 16.

Edge Computing Systems

An interesting challenge brought about by the proliferation of IoT devices is how to service them in a responsive, scalable, and cost-effective manner. To understand the problem, let's first consider a server-based example. Suppose you enjoy playing a massively multiplayer online game (MMOG) on your web browser. The game company would probably host the backend servers in the cloud to allow massive scalability, so the processing power is not an issue. Now suppose all these servers were provisioned in the eastern United States. Gamers in New York would have no problem enjoying the game, but those in Japan would probably have noticeable network latency issues because every one of their commands would have to be sent literally around the world to be processed by the U.S. servers, and then the resulting graphics sent back around the world to the player in Japan. That player would probably lose interest in the game really quickly. Now, suppose that the company kept its main servers in the United States but provisioned regional servers, with one of them in, say, Singapore. Most of the commands are processed in the regional server, which means that the user experience of players in Japan is a lot better, while the global leaderboard is maintained centrally in the United States. This is an example of edge computing.

Edge computing is an evolution of content distribution networks (CDNs), which were designed to bring web content closer to its clients. CDNs helped with internationalization of websites but were also very good for mitigating the effects of DDoS attacks. *Edge computing* is a distributed system in which some computational and data storage assets are deployed close to where they are needed in order to reduce latency and network traffic. As shown in Figure 7-7, an edge computing architecture typically has three layers: end devices, edge devices, and cloud infrastructure. The end devices can be anything from smart thermometers to self-driving cars. They have a requirement for processing data in real time, which means there are fairly precise time constraints. Think of a thermal sensor in one of your data centers and how you would need to have an alarm within minutes (at most) of it detecting rising or excessive heat.

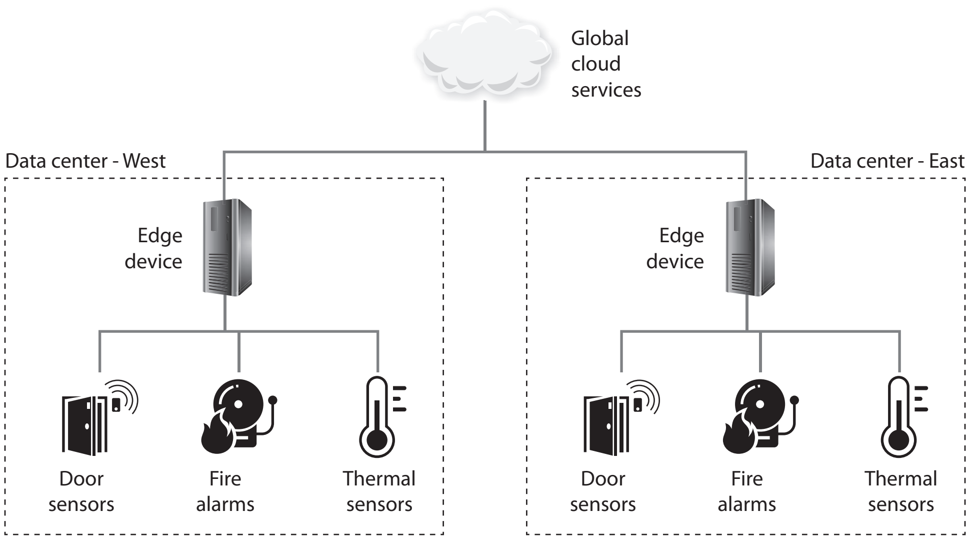


Figure 7-7 A sample edge computing architecture for facility management

To reduce the turnaround time for these computing requirements, we deploy edge devices that are closer to, and in some cases embedded within, the end devices. Returning to the thermometer example, suppose you have several of these devices in each of your two data centers. You also have a multitude of other sensors such as fire alarms and door sensors. Rather than configuring an alarm to sound whenever the data center gets too hot, you integrate all these sensors to develop an understanding of what is going in the facility. For example, maybe the temperature is rising because someone left the back door open on a hot summer day. If it keeps going up, you want to sound a door alarm, not necessarily a temperature alarm, and do it while there is still time for the cooling system to keep the ambient temperature within tolerance. The sensors (including the thermometer) would send their data to the edge device, which is located near or in the same facility. This reduces the time needed to compute solutions and also provides a degree of protection against network outages. The determination to sound the door alarm (and when) is made there, locally, at the edge device. All (or maybe some of) the data from all the sensors at both data centers is also sent to the global cloud services infrastructure. There, we can take our time and run data analytics to discover useful patterns that could tell us how to be more efficient in how we use our resources around the world.

NOTE As increased computing power finds its way into IoT devices, these too are becoming edge devices in some cases.