Chapter 5. Firewalls and Intrusion Detection Systems

> Summoning grids—pentacles with attitude—have a number of uses. Unsurprisingly, summoning spirits from the vasty deeps of Hilbert space is one of them. They can also be used, by the foolhardy or terminally reckless, to open gateways to other spaces (most of which are utterly inhospitable to human-like life). Finally, they can be used to create a firewall, like a science fictional force-field only buggier and prone to hacking attacks by extra-dimensional script kiddies with pseudopods. Which is why nobody with any sense uses them casually.
>
> *The Apocalypse Codex*
> —CHARLES STROSS

5.1 What Firewalls Don't Do

Since the dawn of the commercial Internet, firewalls have been a mainstay of the defense. Many books have been written about them, including two I co-authored [Cheswick and Bellovin 1994; Cheswick, Bellovin, and Rubin 2003]. That said, their utility, and in particular the protection they provide, has diminished markedly over the years. The time has come to ask whether the general-purpose firewall—the one protecting an enterprise—is still worth its capital, operational, and productivity cost.

When the world was young and Bill Cheswick and I wrote the first edition of *Firewalls and Internet Security*, laptops were rare, Wi-Fi and hotel broadband were non-existent, and smart phones weren't even dreamed of. External users logged in to time-sharing machines via the firewall to read their email; companies had very few Internet links to other companies. Even the web was new; the section on it was one of the last things we added to the book before it went to press, and we declined the suggestion that something called a "URL" be employed to state the location of useful resources.

None of that is true today. There is a massive amount of connectivity through and around a typical large firewall, hundreds or even thousands of links. We noted quite some years ago that AT&T had at least 200 links to business partners [Cheswick, Bellovin, and Rubin 2003, p. xiii]; anecdotally, that sort of interconnection has grown greatly in the intervening time. Employees telecommute and travel, staying in touch all the while from a variety of devices including personally owned ones. Attempts to restrict what employees do from their own machines are generally futile (see Chapter 14). Furthermore, much of the important employee traffic to the company, especially email retrieval, is easily encrypted; adding a customs stop at the firewall can *weaken* security, since the encryption is no longer end to end. Whence, then, the traditional firewall? Does it actually do any good? Note carefully that I'm not saying that firewalls *were* wrong; I do not believe that at all. Rather, I'm saying that the world has changed and that the decision to rely on them should be reexamined and perhaps abandoned.

It helps to go back to what we wrote in *Firewalls*. The real problem, we noted, was

buggy code; the purpose of the firewall was to keep the bad guys away from the bugs. Today's firewalls demonstrably cannot do that. Web browsers on $\aleph_0$ different devices are exposed to malware daily, and you can't even start to use a hotel network until you turn off all proxying and VPNs. Similarly, all sorts of nastiness is emailed to people every day, often on their unofficial, unapproved, personally owned, external email accounts, accounts that they check from their employee laptops. (Yes, I know that many security policies prohibit such behavior. They also prohibit employees from copying data to flash drives so that they can get work done at home or while they're on the road. Again, see Chapter 14.)

Beyond that, modern computers—though not (yet?) most tablets or smart phones—all have built-in firewalls; if those are properly configured (see Section 15.3), you may get more security at less cost by scrapping your customs booth. If we enhanced these devices still further to use cryptographically based distributed firewall technology [Bellovin 1999], we'd be in better shape still.

5.2 A Theory of Firewalls

Fundamentally, a traditional firewall is a security policy enforcement device that takes advantage of a topological chokepoint. Let's look at it analytically. There are three properties necessary for a firewall to be effective:

1. There must exist a topological chokepoint at which to place a firewall. Formally, we can regard the network as a graph and the firewall as a cutpoint that partitions the graph into two or more disjoint components.
2. The nodes on the "inside" of the firewall share the same security policy. (See Section 16.2 for more discussion about creation of security policies.)
3. All nodes on the "inside" must be "good"; all nodes on the outside are, if not actually "bad," untrusted.

When one or more of these conditions does not hold, a firewall cannot succeed. Today, none are true for the typical enterprise.

Property 1 fails because of the number of links a typical company has, links that do not go through "the" firewall. These links may be to suppliers, customers, joint venture partners, outsourcees, what have you.

Property 2 fails because of the number of computers used today. With so many nodes, the policies have to differ drastically. When firewalls first became popular, only a small subset of employees needed Internet connectivity. For all practical purposes, there was no web. Email was not the way business was done. Documents were faxed rather than scanned and attached. If, by some chance, your company needed connectivity to another company, you leased a circuit from the phone company, but that wasn't a gross exposure because most companies didn't have very much connectivity even internally. (Bear in mind that *Firewalls* came out $1\frac{1}{2}$ years before the release of Windows 95, the first Microsoft operating system with TCP/IP support. You could get TCP/IP, but it was an add-on product from some outside vendor. Most machines were not upgraded in that fashion.)

Property 3 fails, too, partly because of the large population on the inside, and partly because of mobile nodes: if they get infected when they're on the outside, they're effectively bad guys when inside the firewall.

There's a corollary to this: firewalls can work in environments where these conditions still hold. To take a neglected but important case, most residences are protectable by firewalls. (Admittedly, parents and teenagers often disagree on the proper security policy or even on what constitutes "good" behavior. Besides, consumers have smart phones and laptops, too. Computer-savvy parents will sometimes set up a separate "teen-net," isolated from "parent-net.")

It is worth noting that these three properties, with the possible exception of Property 3, are not absolute; tolerating minor deviations is feasible. You can have two or possibly three firewalls—with more than that, coordinating policy is hard, and there's too much chance of return traffic going through a different firewall—you can have a small number of different policies for machines like the mail server, etc.

Given this model, we can construct scenarios where firewalls are effective. A degenerate case is a single machine. If we regard the firewall as residing between the network interface and the TCP/IP stack (and this is, in fact, how it is often implemented), all three conditions are obviously satisfied. What makes this interesting is where policy comes from. All current operating systems permit the machine's administrator to set the policy for that machine. Some packages permit a central administrator to ship policy to many endpoints. Finally, if we use cryptographically verified identity to accomplish our partition (and thus use a virtual network topology rather than a physical one), we achieve the distributed firewall described in [Bellovin 1999].

A more interesting case is a departmental print server. The policy is simple: anyone within the department is allowed to print; no one outside is allowed to print at all. Property 2 is therefore satisfied. Departments typically don't have rich connectivity, thus satisfying Property 1. Finally, since by policy anyone on the inside has permission to use the printers, everyone is by definition good, satisfying the final property.

There's one wrinkle. If the department has an external link, perhaps to a supplier, systems on the far side of that link should be barred from reaching the printer. That could be accomplished by a packet filter on the router handling that connection; alternately (and per the discussion below on threat models), it can be ignored.

It is fair to ask what other choices there might be. Any network connected device needs some sort of access control; many do not provide it, or do not do a good job of it. Your typical printer, for example, does not support TLS or logins and passwords; if it did, many computers would have trouble talking to it. However, some sort of access control is necessary; I do *not* want some bored teenager launching a denial of service attack on my paper budget or on my printers' hardware [Cui and Stolfo 2011]. (I recently bought a new home printer with IPv6 support—which I promptly turned off because there was no access control option; anyone who knew its name or IP address could have reached it.)

There are other services that are commonly used within a department or other small group; file servers are the obvious example. These often do have their own authentication; nevertheless, the service provided is sufficiently sensitive (and the underlying code has been sufficiently buggy, at least in the past) that an extra layer of protection is useful.

The solution is a *point firewall*, a simple firewall in front of a limited set of resources. Point firewalls work because of their scope: they are not trying to protect arbitrarily many devices, they are not enforcing complex rules, they are not dealing with thousands of exceptions. There is also a more subtle philosophical difference: their primary function is not just bug deflection; rather, they are add-on access control mechanisms. Nevertheless, since they are enforcing a policy they qualify as firewalls.

If we add the threat environment to our model, we can generalize still further. In particular, we will assess the properties separately for different threat levels. When we do, we see that enterprise firewalls do provide a modicum of protection against low-skill hackers. Specifically, let's consider joy hackers.

Property 2 is effectively true; typically, everyone has the same policy against the sorts of attacks launched by joy hackers. A simple "no inbound calls" policy, plus rules to force all mail and perhaps web traffic through specific virus scanner-equipped gateways will likely handle all of the usual attacks from that grade of bad guy.

Property 1 is more subtle. If we temporarily ignore links to other companies, we're left with the usual handful of connections to ISPs; these are the traditional locations for firewalls. This property is thus satisfied.

It is Property 3 that is the most interesting for this scenario. I will assert that at the joy hacker level, to a first approximation all employees are honest good guys. Yes, there is embezzlement, insider trading, and the usual rate of petty thefts from the stockroom or the paper cabinet (Bell Labs used to use four ring loose leaf binders, apparently to discourage people from bringing these home as school supplies). However, what little low-grade technical malfeasance there is tends to be locally targeted; employees will attack what they know, and by definition unskilled attackers do not have the tools or knowledge to learn the extended network topology, especially to other companies. We thus satisfy Property 3 and Property 1.

The risk from mobile devices remains, but today's dangerous viruses are not the work of script kiddies. We can thus conclude that traditional enterprise firewalls do provide some protection.

Our protection breaks down if we consider opportunistic hackers, and of course all bets are off when dealing with MI-31. An opportunistic hacker is capable of launching sophisticated viruses and worms, working out (or stumbling on) a path through interconnected companies (or using a worm that does the same thing), and so on. We thus lose Property 1. Property 2 also fails, because a sophisticated attacker can find and exploit weaker policies. Still, the biggest risks can be deflected, at least partially, if we can protect the external links (Chapter 11).

Targetiers don't have much technical skill, but they are targeting you. If nothing else, it means that we lose Property 3 because such people will resort to physical presence in their

attacks. Depending on just how much skill they have, they may also be able to exploit links between companies, which violates Property 1. In other words, enterprise firewalls do not protect against higher grades of attackers.

It is important to realize that the three properties were applicable all along; however, in the mid-1990s the threat profile was very different. There were few targetiers or Andromedans, with the arguable exception of Cliff Stoll's Stasi-controlled East Germans [1988; 1989]. There were opportunistic attackers, including some very good ones, but with no interconnected companies and few mobile devices their scope of operation was limited.

* * *

Another way to think about firewalls is to realize that no firewall can provide protection at any layer other than the one at which it operates. A typical packet filter, for example, works at layer 3 and a bit of layer 4 (the port numbers); as such, it can filter by IP address and service. It can't look at MAC addresses (especially from more than one hop away because it never sees them!), nor can it look inside email messages. The trouble is that a good firewall needs to operate at multiple layers. It may need to do TCP normalization, to deal with attackers who are playing games with the subtle semantics of TCP [Handley, Kreibich, and Paxson 2001]; it definitely needs to scan email for viruses, block nasty web sites, and more. Furthermore, the trend in recent years has been to layer more and more non-web protocols on top of HTTP or HTTPS, rather than directly on TCP or TCP+TLS; simple port number filtering or circuit relaying will no longer do the job. (HTTP is sometimes referred to as the "universal solvent for firewalls.") Indeed, even in *Firewalls* we described the need for FTP and X11 application proxies; the need has gotten much more urgent since then.

One result of the increased attention to higher-level protocols has been the rise of *Deep Packet Inspection (DPI)* [N. Anderson 2007]. A DPI firewall has rules and policies that look at more or less arbitrary parts of packets. This is difficult, and not just because of performance issues; in general TCP implementations will split up messages as they see fit, forcing a DPI system to reassemble packets and keep extra state. Furthermore, the ability to express policies in terms of the contents of packets has led to evermore complex policies; these themselves are a significant source of trouble.

The result of this shift is that firewalls have gotten far more complex. There are many more different application that are of interest; each of them requires custom code on the firewall to enforce policies and to delete or otherwise defang sketchy stuff. This is bad for security. It pays to look back at what Cheswick and I wrote in *Firewalls*:

> **Axiom 1 (Murphy)** *All programs are buggy.*
>
> **Theorem 1 (Law of Large Programs)** *Large programs are even buggier than their size would indicate.*

Are Network Address Translators Firewalls?

Network address translators (NAT boxes) [Srisuresh and Egevang 2001] are a source of controversy in the networking community. Some people denounce them as the spawn of Satan, excrescences on the body technic that interfere with end to end communication. Others point to their necessity—we've long since run out of IPv4 addresses—and tout the interference as a security virtue: NATs are firewalls, they say. Are they right? If so, do their benefits outweigh their disadvantages?

By definition, NATs operate at the network layer, with a slight excursion up to the transport layer to inspect and modify port numbers. As a consequence, they provide no protection against things like emailed malware and nasty web sites. The answer to the first question is therefore obvious: NATs are not firewalls per se; however, they do provide protection more or less equivalent to that of a packet filter. Combined with protections common to many consumer ISPs or large enterprises—a central mail server with virus detection, and web filtering done by a proxy or by the features built in to many browsers—there is a tolerably complete level of protection, more than one would have without the NAT. Alternate schemes based on host resident filtering generally require configuration, ruling them out for most home use.

The issue of balance, though, is rather more complicated and subjective. Some of the cost is already borne by everyone, in the form of excess protocol complexity or in the need for auxiliary helper servers. There are other features that don't work very well through NATs. As end systems become more and more hardened against direct attacks, the benefits of NATs decrease and the costs become higher. On balance, I'd say they're not worth it—and I've enjoyed having direct IPv6 access to my house without interference from a NAT. Of course, until IPv6 becomes ubiquitous, most home users have no real choice.

Proof: By inspection. ■

Corollary 1.1 *A security-relevant program has security bugs.*

Theorem 2 *If you do not run a program, it does not matter whether or not it is buggy.*

Proof: As in all logical systems, (**false** ⇒ **true**) = **true**. ■

Corollary 2.1 *If you do not run a program, it does not matter if it has security holes.*

Theorem 3 *Exposed machines should run as few programs as possible; the ones that are run should be as small as possible.*

Proof: Follows directly from Corollary 1.1 and Corollary 2.1. ■

Corollary 3.1 (Fundamental Theorem of Firewalls) *Most hosts cannot meet our requirements: they run too many programs that are too large. Therefore, the only solution is to isolate them behind a firewall if you wish to run any programs at all.*

In other words, the reason that firewalls were secure is that they ran many fewer programs, and hence didn't have as much vulnerable code. Given the hundreds of applications that a modern firewall has to support, and given the complexity of some of those applications (e.g., SIP), it is far from clear that less code is involved. In fact, an enterprise firewall today, supporting very many users, endpoints, and policies, is arguably running *more* Internet-facing code than a typical host. Perhaps the code is higher quality, and perhaps the firewalls are better administered than end user machines—but perhaps not.

The code complexity issue is another driver for smaller, more specialized firewalls. The XML-scanning firewall protecting some database machine may be just as buggy as the same code on an apatosaurus-sized firewall for the enterprise, but if it fails it exposes one database machine, not an entire company.

* * *

We can therefore draw some conclusions about the role of firewalls in today's net.

- Small-scale firewalls, protecting a network about the size run by a single system administrator, still serve a useful function. Generally speaking, these will be packet filters and hence not require extra hardware.
- Complex server applications are rarely amenable to firewall protection, unless the firewall has some very, very good (and very, very well-written) sanitizing technology.
- An enterprise firewall retains value against low-skill attackers but is actually a point of risk, not protection, when trying to filter complex protocols against sophisticated adversaries. If you have such services that must be accessible from the outside, use packet filtering on the enterprise firewall and a separate protection layer near the server itself. This is discussed in more detail in Section 11.3.
- Arguably, mobile devices—laptops, tablets, smart phones—should never be fully trusted, not because they use wireless connections, but because they're much more likely to carry malware (see Property 3). This suggests that your wireless LAN should be outside the firewall, with a VPN+filter for access by iToys and the like. The suggestion is analyzed in greater detail in Chapter 9.

Planning for Failure

Given the complexity of some application firewall modules, it is not unreasonable to suspect that they might fail. What is the proper course of action? The proper design *assumes* that a failure can happen and tries to mitigate the consequences. Here are two approaches.

The first approach is to abandon the notion of an application-specific firewall for that system. Unless the firewall can do filtering or blocking that the host itself can't do, it doesn't add any value over a high-quality host application. Web servers are a good example; the danger comes from the HTTP itself and the scripts that are run in response; what can the firewall add? What a simple firewall, such as a packet filter, *can* do is block access to other ports on the server and prevent illicit outgoing calls. So—the server is hacked, but the attacker can't go anywhere else in the company and may not even be able to steal the data. This is a cheap design, in that all it needs is a router port with the appropriate access control rules; only the dodgy server is behind that port.

The second approach uses a properly designed application firewall; again, all that's behind it is the server you want to protect. By "properly designed" I mean one that implements the same type of dual protection: a packet filter followed by an application-specific module. There are two utterly crucial internal architectural details. For one thing, the application proxy must be "behind" the packet filter, that is, between the packet filter and the port facing the server, so that any outward-bound traffic from the proxy to the rest of the company must pass through the packet filter.

Furthermore, the internal structure of the firewall must be such that if the proxy module itself is penetrated, the attacker cannot reprogram the packet filter. Unfortunately, it's very hard to learn that sort of internal detail about the design.

5.3 Intrusion Detection Systems

"Do you mean to admit that *you* may have been invaded and searched —tracelessly?" Alcon fairly shrieked the thought.

"Certainly," the psychologist replied, coldly. "While I do not believe that it has been done, the possibility must be conceded. What we could do, we have done; but what science can do, science can circumvent."

Second Stage Lensman
—E. E. "DOC" SMITH

An *intrusion detection system (IDS)* is a backup security mechanism. It assumes that your other defenses—firewalls, hardened hosts, goat entrails (tofu entrails for vegetarian security professionals) offered up in the dark of the moon—have failed. The task then is to notice the successful attack as soon as possible, which permits minimization of the

damage, either via automated systems or their backup humans.

Most of what I've said about antivirus technology is true of IDSs as well. An IDS can be signature or anomaly based; the same advantages and disadvantages apply. The key difference is deployment scenarios and hence inputs; antivirus programs operate on files, whereas IDSs are more multifarious.

IDSs are generally classified as network or host intrusion detection systems; for the latter, they can operate on network or host behavior or content. Each of these approaches has benefits and limitations.

The big attraction of anything network based is the same as the big attraction of a firewall: it's scalable, in that there are typically many fewer networks to instrument than hosts. In fact, the firewall is one very common location to install a network IDS, since by definition all traffic from the outside is supposed to pass through that chokepoint.

Doing intrusion detection in the network, by grabbing packets in flight, is difficult. The obvious problem is dealing with encrypted traffic; more seriously, it's all too easy to miss packets. There are also theoretical issues with enemies who try to exploit odd corner cases in the network protocol specs [Handley, Kreibich, and Paxson 2001], though such behavior seems to be rare or unknown in the wild. (If the Andromedans are doing this, perhaps they haven't yet been caught at it?)

The simplest form of network IDS relies on IP addresses and port numbers: if the packets are going to destinations that some parties shouldn't try to reach, you know there's a problem. The technique is analogous to the "network telescope" concept [Cheswick 2010; C. Shannon and Moore 2004]: if some IP addresses are deliberately left empty, packets sent to them (or from them!) are a priori suspicious. The same can be true of certain ports on sensitive hosts, especially if you have good information on just who can legitimately send to them.

If that's all you want to do, though, don't bother trying to look at packets. Your routers are already doing that; some places have built IDSs based on routers' NetFlow data.[1]

1. "PaIRS: Point of contact and Incident Response System," http://goo.gl/xhroc.

More sophisticated network monitoring can be done as well. There are comparatively simple systems that look for simple patterns of data, such as Bro [Paxson 1998; Paxson 1999] or Snort [Roesch 1999]. DPI systems [N. Anderson 2007] are more sophisticated; they look at higher layers of the stack and are often used for various sorts of governmental monitoring [Poe 2006].

The fundamental problem with any form of network IDS is that it lacks context. Yes, DPI and other forms of network monitoring can detect suspicious packets, but it's difficult for even the best network scanners to reassemble every file in transit and then scan it for malware. That sort of thing is much easier to do on hosts. Hosts can also look at log files; more importantly—and all but impossible to do on the wire—they can scan their own file systems for unexpected changes [G. Kim and Spafford 1994a; G. Kim and Spafford 1994b; G. Kim and Spafford 1994c]. Finally, host based IDSs are network independent;

they can detect problems no matter how they arrive, whether via the Internet or carried in on an infected USB flash disk.

Host-based IDSs can do one more thing more easily than their network partners: they can emulate network protocols, above the level of any encryption. Depending on their purpose, they can be part of or intermediaries for the real network daemons; alternatively, they can be pure fakes, doing nothing but detecting things that you hope will never happen. This is an ancient technique [Bellovin 1992], but it is useful nevertheless.

5.4 Intrusion Prevention Systems

> We have a VPN, and firewalls, and you do not want to mess with them because the design spec for the Laundry's firewall software is not to keep intruders out, but to make them undergo spontaneous combustion when they get in: as Bob puts it, it's the only way to be sure.
>
> *The Annihilation Score*
>
> —CHARLES STROSS

Suppose a network IDS does detect something unpleasant. Then what? An *intrusion prevention system (IPS)* can best be described as an IDS with an attitude. Rather than simply detecting something bad, they try to do something about it. The trick is avoiding collateral damage, or at any rate collateral damage that's worse than what the attack would have caused if left unmolested. The worst situation is a successful attack whose goal was to induce you to perform harmful actions.

Consider, for example, the Slammer worm [Moore et al. 2003]. Slammer spread via a single UDP packet to port 1434, used by a Microsoft SQL server. Because UDP does not require a 3-way handshake the way TCP does, the worm spread extremely quickly; its growth rate was limited by the outbound bandwidth of infected hosts. One can postulate an IPS that noticed links being clogged, saw a tremendous spike in traffic to a rarely seen port, and automatically set up a filter rule blocking such packets. It makes perfect sense, and that is in fact what was done by many ISPs. Now imagine a variant of Slammer that emitted three packets to UDP port 53 for every one it sent to 1434. The packets to 53 —DNS—would, for this example, be harmless, but would a network based IPS know that? All that it can see are three facts: links are being clogged by an unprecedented flood of traffic; many of the unusual packets are to port 1434; even more of them are to port 53. Would it try to shut down both ports? If so, the IPS would effectively turn off the Internet. (N.B. I've slightly simplified the details of this enhanced attack; correcting it is left as an exercise for the reader.)

An IPS can do many things. As with an IDS it can be host or network resident; both sitings have advantages and disadvantages. Depending on where it is located, it can block connections, quarantine files, modify packets, and more [Scarfone and Mell 2007]. Forrest and Somayaji described one that slowed down suspect processes, rather than killing them [2000]; this scheme doesn't do irrevocable harm if it's guessed incorrectly.

Ultimately, the IPS problem rests on three pillars: very good detection, selection of

countermeasures, and matching the countermeasures to confidence in identification of the root cause of the problem. This last issue is much less studied than the second, which in turn is much less studied than the first.

5.5 Extrusion Detection

Extrusion detection is a specialized form of IDS. It's aimed at one particular form of harm: someone trying to steal your data and export it. The trick is picking up the outbound data transfer. There are two challenges: picking out the right data, amidst all of the legitimate (or at least normal) traffic, and distinguishing authorized from unauthorized transfers. This latter isn't trivial; uploading a chip design to a foundry can be the normal way of doing business, while sending it to the Andromedans' web server most likely is not. Extrusion detection has one principal advantage over many other types of security systems: it can cope with rogue insiders.

There are a number of ways to perform extrusion detection. One of the simplest is the *honeypot*: create fake files that will attract the attention of a spy, commercial or governmental, and wait for someone to grab one. This has been done a number of times, most famously by Stoll [1988; 1989] in the "Wily Hacker" incident. (Bill Cheswick and I subtitled *Firewalls* "Repelling the Wily Hacker" in homage to Cliff, and we used that phrase with his permission.) Briefly, he discovered intruders in a University of California computer system and traced them to Germany. To allow enough time for technicians to trace the attackers' phone calls—this was in the days of dial-up modems—he created fake documents about the Strategic Defense Initiative, a missile defense system, and waited for someone to look at them. His trap was successful; the attackers' response included sufficient indicia of espionage that he notified the FBI.

The WikiLeaks Cables

The case of the classified US diplomatic cables given to WikiLeaks and then published provides an interesting case study in extrusion detection. The factual basis for this analysis is mostly taken from [Capehart 2012a; Capehart 2012b; Capehart 2012c; Capehart 2012d; Capehart 2012e], a series worth reading for the discussion of procedural issues (even though I feel that some of Capehart's conclusions are debatable); also see [Zetter 2011] and [BBC 2014].

What happened is that Chelsea Manning (at the time of the actions and the trial, male and known as "Bradley"), an apparently untrustworthy individual who nevertheless had access to systems holding classified documents, developed scripts to do bulk downloads. The downloaded documents were burned onto a CD labeled "Lady Gaga"; the contents of the CD were then shared with WikiLeaks.

There has been a lot of criticism of the US State Department for having such sensitive data available with very weak controls. While the issue is indeed debatable, it was the result of a deliberate decision to increase availability of data, even highly classified data, to individuals with suitable clearances; lack of information sharing had been seen as one of the problems leading up to the 9/11

terrorist attacks.

However, what could and should have been done was to log accesses, and *look for unusual patterns*. Apparently, more than 250,000 cables were downloaded. Are there any legitimate uses for that sort of bulk download by a single individual? Proper log files, and proper analysis of them, would have shown that something unusual was happening. At the least, security personnel could have investigated.

Manning herself apparently understood the problem. She wrote, "Weak servers, weak logging, weak physical security, weak counter-intelligence, inattentive signal analysis… a perfect storm" [Poulsen and Zetter 2010].

Marcus Ranum has summed it up nicely [Field 2010]:

> Then the other piece of the puzzle that I find is really interesting is the apparent inability of the people who lost the data, the original data holders, to tell what data was stolen and while it was being stolen *[sic]*. And this is an important message for anyone who is a CISO [Chief Information Security Officer] because it shows what can happen when your data leaks if you don't have auditing and logging in place so that you can go back and say, "Well, OK if we do believe this guy leaked a bunch of information, what information did he actually access and when?" Of course, ideally you would get in front of that process and maybe detect the fact that somebody who really didn't have a need to access this particular information was downloading [this information] in one fell swoop. That is kind of a red flag, I would think.

Precisely.

In the more usual scenario, you don't know whether you've been penetrated. Accordingly, the proper honeypot strategy requires a wide range of believable-seeming decoys. Exactly what decoys you should create depends on your system; you want something that resembles normal items on that system. Bowen at al. [Bowen 2011; Bowen et al. 2009] describe a number of types of decoys—"honey documents"—including fake logins for banking web sites; they also define criteria for decoy generation. One notable aspect to their work was the use of *web bugs*, normally used by advertisers to track consumers on the web. With a web bug, opening the document causes an attempt to fetch a file (typically a 1 × 1–pixel transparent GIF) from a monitored HTTP server.

Naturally, a sufficiently knowledgeable attacker can dodge many decoys. Someone who suspects web bugs, for example, would simply read exfiltrated documents on an offline computer. Indeed, intelligence agencies' classified networks are generally disconnected from the outside world [R. A. Clarke and Knake 2010], so no strategy that relies on active documents can succeed. A different approach is needed, one based on IDS technology.

The big advantage of honey documents is that they're transport independent. That is, no matter how the files are exfiltrated, the trap can be sprung on any network connected

machine used to view them. Even printing them out first doesn't help; the monitors will detect the documents being opened inside the enterprise.

You may be able to detect network based exfiltration while the documents are being transported if your network is configured in a firewall-friendly manner, that is, if Property 1 holds. An extrusion detection module can be installed at the firewall; it can then attempt to detect misbehavior from amidst the noise of routine Internet traffic. As with IDSs, one can approach this from a signature or anomaly detection perspective. Signature detection can look for certain documents or perhaps markings—should the strings "Company Confidential" or "Top Secret UMBRA" appear in outbound mail?—or anomaly detection. Anomaly detection might be as simple as volume—does this person or IP address normally send so much data? Does someone in that organization normally send so much?—or it may be based on the statistical characteristics of the outbound data.

There are some interesting wrinkles here that makes extrusion detection harder in some ways than firewalls or intrusion detection. For one thing, someone exporting information is freer to use encryption because he or she can control both ends of the channel. By contrast, an attacker breaking in to an organization can only encrypt traffic if the vulnerability being exploited uses encryption. Of course, once the penetration is complete, the attacker can install any sort of back door desired, including encrypted ones. Even if encryption is used, the defenders aren't helpless. While crypto does hide the precise content being sent, it can't hide the volume; more importantly, encrypted data has a very unique flat byte distribution; this, too, is anomalous if from a source or to a destination that does not normally receive such. An example is shown in Figure 5.1, which compares the byte frequency distribution of a JPG file and an encrypted version of the same file.