35. Firewalls

35. Firewalls

35.1. Introduction to Controlling Network Access

Suppose you are given a machine and asked to harden it against external attacks. How would you go about doing it?

A possible starting point might be to look at the functionality and the network services that this machine is providing to the outside world. If any of the network services have bugs or security flaws, an attacker could exploit that part of the service and might be able to penetrate your machine. As we know, bugs are inevitable, and bugs in security-critical applications often lead to security holes. Therefore, the more network services that your machine runs, the greater the risk of attacks.

The general principle here is that bugs present in code that you do not run cannot hurt you. Therefore, the less functionality you try to provide, the less of an opportunity exists for security vulnerabilities in that functionality. This suggests one simple way to reduce the risk of external attack: *Turn off every unnecessary network service.* By disabling every network-accessible application that isn't absolutely needed, you are essentially building a stripped-down box that runs the least amount of code necessary. After all, any code that you don't run, can't hurt you. And for any network service that you do run, double-check that it has been implemented and configured securely, and take every precaution possible to ensure that it is safe.

While this is an intuitive and fairly effective approach when you only have one or two machines to secure, the problem becomes slightly more complicated when we scale things up. Suppose you are now in charge of security for all of Caltopia, and your job is to protect the computer systems, the network and its infrastructure from external attacks. If Caltopia has thousands of computers, it will be extremely difficult to harden every single machine individually as each computer could have different operating systems and different hardware platforms. Furthermore, different users could have very different requirements, where a service that could be disabled for one user might be essential to another user's job. At this scale, it is often hard just to get an accurate list of all machines inside the company, and if you miss even one machine, it could then become a vulnerable point that could be broken into and serve as a launching point for attackers to attack the rest of the Caltopia network. So, managing each computer individually is probably infeasible at this scale.

However, it is still true that one risk factor is the number of network services that are accessible to outsiders. This suggests a possible defense; if we could block, *in the network*, outsiders from being able to interact with many of the network services from running on internal machines, we could potentially reduce the risk. This is exactly the concept behind *firewalls*, which are devices designed to block access to network services that are running on internal machines. Essentially, firewalls reduce risk by blocking network outsiders from having unwanted access to all the network services by acting as a choke point between the internet (outsiders) and your internal network. Now, all we need to know to implement a firewall is:

1. What is our *security policy*? Namely, which network services should be made visible to the outside world and which should be blocked? How do we discern insiders from outsiders?
2. How will we *enforce the security policy*? How do we build a firewall that does what we want it to do and what are the implementation issues?

35.2. Security Policy

If you wanted to visualize the topology of the internal network, you could think about having an internal network, which hosts all of the company's machines, the external world, which is the rest of the internet, and a communications link between the two.

How do we decide which computers have to be affected by the firewall and which don't? A very simple threat model could have us decide that we trust all company employees, but we don't trust anyone else. Thus, we define the internal network to contain machines owned by trusted employees and the external world to include everyone (and everything) else. The link to our Internet Service Provider (ISP) could be the link between the two networks.

Perhaps the simplest security policy that we could easily implement would be an outbound-only policy. Before we delve into how it works, let's define and distinguish between inbound and outbound connections. An *inbound connection* is one that is initiated by external users and attempts to connect to services that are running on internal machines. On the other hand, an *outbound connection* is one which is initiated by an internal user, and attempts to initiate contact with external services. An outbound-only connection would permit all outbound connections, but inbound connections would be denied outright. The reasoning behind such a connection is that internal users are trusted, so if they wish to open a connection, we will let them. The effect of the resulting connection is that none of our network services are visible to the outside world, but they can still be accessed by internal users. The issue is that such a security policy is likely too restrictive for any large organization since it means that the company cannot run any public web server, a mail server, an FTP server, etc. Therefore, we need a little more flexibility in defining our security policy.

More generally, our security policy is going to be some form of *access control policy*, wherein we have two subjects: a generic internal user and an anonymous external user. We then define an *object* to be the set of network services that are run on all inside machines; if there are 100 machines, and each machine runs 7 network services, then we have 700 objects. An access control policy should specify, for each subject and each object, whether that subject has permission to access that object.

A firewall is used to enforce a specific kind of access control policy, one where insider users are permitted to connect to any network service desired, whereas external users are restricted; there are some services that are intended to be externally visible (and thus, external users are permitted to connect to those services), but there are also other services that are not intended to be accessible from the outside world (and these services are blocked by the access policy).

As a security administrator, your first job would be to identify a security policy, namely which services should external users have access to and which services should external users not have access to. Generally, there are two main philosophies that we might use to determine which services we allow external users to connect to:

- *Default-allow or blacklist*: By default, every network service is permitted unless it has been specifically listed as denied. Under this approach, one might start by allowing external users to access all internal services, and then mark a couple of services that are known to be unsafe and therefore should be blocked. For example, if you learn today that there is a new threat that targets Extensible Messaging and Presence Protocol (XMPP) servers, you might be inclined to revise your security policy by denying outsiders access to your XMPP servers.
- *Default-deny or whitelist*: By default, every network service is denied to external users, unless it has been specifically listed as allowed. Here, we might start off with a list of a few known servers that need to be visible to the outside world (which have been judged to be reasonably safe). External users will be denied access to any service that is not on this list of allowed services. For example, if Caltopia users complain that their department's File Transfer Protocol (FTP) server is inaccessible to the outside world (since it is not on the allowed list), we can check to see if they are running a safe and properly configured implementation of the FTP service, and then add the FTP service to the "allowed" list.

The default-allow policy is more convenient since, from a functionality point of view, everything stays working. However, from a security point of view, the default-allow policy is dangerous since it *fails-open*, meaning that if any mistake is made (if there is some service that is vulnerable but you forgot to add it to the "deny" list), the result is likely to be some form of an expensive security failure.

On the other hand, the default-deny policy *fails-closed*, meaning that if any mistake is made (if some service that is safe has been mistakenly omitted from the "allow" list), then the result is the loss of functionality or availability, but not a security breach. When operating at large scales, such errors of omission are likely to be common. Since errors of omission are a lot more dangerous in a default-allow policy than in a default-deny policy, and because the cost of a security failure is often a lot more than the cost of a loss of functionality, default-deny is usually a much safer bet.

Another advantage of the default-deny policy is that when a system fails open (like in default-allow), you might never notice the failure, and since attackers who penetrate the system are unlikely to tell you that they have done so, security breaches may go undetected for long periods of time. This gets you into an arms race, wherein you have to keep up with all of the attacks that adversaries discover, and even stay ahead of them. This arms race is usually a losing proposition since there are a lot more of the attackers than there are defenders and the attacker only has to win once to make you extremely vulnerable. In contrast, when a system fails closed (like in default-deny), someone will probably notice the flaw and will likely complain that some service is not working; since the omission is immediately evident and easily correctable, failures in default-allow systems are much more costly than failures in default-deny systems.

As such, a majority of well-implemented firewalls implement a default-deny system, wherein the security policy specifies a list of "allowed" services that external users are permitted to connect to, and all other services are forbidden. To determine whether a service should be removed from the allowed list, some kind of risk assessment and cost-benefit analysis is applied; if some service is too risky compared to its benefits, it is removed from the allowed list.

To identify network services, recall that TCP and UDP connections can be uniquely identified by the machine's IP address and port number. Therefore, we can identify each network service with a triplet (m, r, p), where m is the IP address of a machine, r is the protocol identifier (i.e. TCP or UDP), and p is the port number. For instance, the company might have its official web server hosted on machine 1.2.3.4, and then (1.2.3.4, TCP, 80) would be added to the allowed list. In a default-deny policy, the list of network services that should be externally visible would be represented as a set of these triplets.

35.3. Enforcement: Stateful Packet Filters

The main idea behind enforcing security policies is to do so at a choke point in the network. The existence of a central choke point gives us a single place to monitor, where we can easily enforce a security policy on thousands of machines with minimal effort. This idea of a choke point is similar to that of physical security; at an airport, for example, all passengers are funneled through a security checkpoint where access can be controlled. It is easier to perform such checks at one, or a few, checkpoints rather than hundreds or even thousands of entrances.

A stateful packet filter is a router that checks each packet against the provided access control policy. If the policy allows the packet, it is forwarded on towards its destination; if the policy denies the packet, then the packet is dropped and is not forwarded. The access control policy is usually specified as a list of rules; as the firewall processes each packet, it examines the rules one-by-one, and the first matching rule determines how the packet will be handled.

Typically, rules specify which connections are allowed. The rule can list the protocol (tcp or udp), the initiator's IP address (the machine that initiated the connection), the initiator's port number, the recipient's IP address (the machine that the connection is directed to), and the recipient's port number. A rule can use wildcards, denoted by the symbol $*$, for any of these. Each rule also specifies what action to take for matching connections; typical values might be ALLOW or DROP.

For example, take the following ruleset:
allow tcp ∗ : ∗ → \$ 1.2.3.4:25\$
drop ∗ ∗ : ∗ → ∗ : ∗

This ruleset allows anyone to open a TCP connection to port 25 on machine 1.2.3.4, but blocks all other connections.

A stateful packet filter maintains state, meaning that it keeps track of all open connections that have been established. When a packet is processed, the filter allows the firewall to check whether the packet is part of a connection that is already open. If it is, then the packet can be forwarded. Without state, it is harder to know how to handle the packet; for example, if we see a packet that is going from X to Y, we don't know if the packet was on a connection that was initiated by X or by Y, the answer to which might determine whether or not the packet is allowed to be forwarded. By keeping state, stateful packet filters allow policies that inspect the data, like for example, a policy that blocks any attempt to log into an FTP server with the username "root". However, stateful packet filters must be written extremely carefully to ensure that it only keeps a small amount of information per connection to ensure that the firewall does not run out of memory.

35.4. Enforcement: Other Firewalls

Stateless packet filters tend to operate on the network level and generally only look at TCP, UDP, and IP headers. In contrast to stateful packet filters, stateless packet filters do not keep any state, meaning that each packet is handled as it arrives, with no memory or history retained by the firewall.

Application-layer firewalls restrict traffic according to the content of the data fields. These types of firewalls have certain security advantages since they can enforce more restrictive security policies and can transform data on the fly.

Rather than simply inspecting traffic, we can also build firewalls that participate in application layer exchanges. For example, we can introduce a web proxy in a network and configure all local systems to use it for their web access. The local web browsers would then connect to the proxy rather than directly to remote web servers, and the proxy would in turn make the actual remote request. A major benefit of this design is that we can include monitoring in the proxy that has available for its decision-making all of the application-layer information associated with a given request and reply, so we can make fine-grained allow/deny decisions based on a wealth of information. This sort of design isn't specific to web proxies but can be done for many different types of applications. The general term is an application proxy or gateway proxy. One difficulty with using this approach, however, is implementation complexity. The application proxy needs to understand all of the details of the application protocol that it mediates. Another potential issue concerns performance. If we bottleneck all of the site's outbound traffic through just a few proxy systems, they may be overwhelmed by the load.

35.5. Firewall Principles

In general, the mechanism that enforces an access control policy often takes the form of a reference monitor. The purpose of a reference monitor is to examine every request to access any controlled resource (an "object") and determine whether that request should be allowed.

There are three security properties that any reference monitor should have:

- Unbypassable (also known as Always invoked): The reference monitor should be invoked on every operation that is controlled by the access control policy. There must be no way to bypass the reference monitor (i.e., the complete mediation property): all security-relevant operations must be mediated by the reference monitor.
- Tamper-resistant: The reference monitor should be protected from tampering by other agents. For instance, other parties should not be able to modify its code or state. The integrity of the reference monitor must be maintained.
- Verifiable: It should be possible to verify the correctness of the reference monitor, including that it actually does enforce the desired access control policy correctly. This usually requires that the reference monitor be extremely simple, as generally it is beyond the state of the art to verify the correctness of subsystems with any significant degree of complexity.

We can recognize a firewall as an instance of a reference monitor. How are these three properties achieved?

- Always invoked: We assumed that the packet filter is placed on a chokepoint link, with the property that all communications between the internal and external networks must traverse this link. Thus, the packet filter has an opportunity to inspect all such packets. Moreover, packets are not forwarded across this link unless the packet filter inspects them and forwards them (there needs to be no other mechanism by which packets might flow across this link). Of course, in some cases we discover that it doesn't work out like we hoped. For instance, maybe a user hooks up an unsecured wireless access point to their internal machine. Then anyone who drives by with a wireless-enabled laptop effectively gains access to the internal network, bypassing the packet filter. This illustrates that, to use a firewall safely, we'd better be sure that our firewalls cover all of the links between the internal network and the external world. We term this set of links as the security perimeter.
- Tamper-resistant: We haven't really discussed how to make packet filters resistant to attack. However, they obviously should be hardened as much as possible, because they are a single point of failure. Fortunately, their desired functionality is relatively simple, so we should have a reasonable chance at protecting them from outside attack. For instance, they might not need to run a standard operating system, any user-level programs, or network services, eliminating many avenues of outside attack. More generally, we can use firewall protection for the firewall itself, and not allow any management access to the firewall device except from specific trusted machines. Of course, we must also ensure the physical security of the packet filter device.
- Verifiable: In current practice, unfortunately the correctness of a firewall's operation is generally not verified in any systematic fashion. The software is usually too complex for this to be feasible. And we do suffer as a result of our failure to verify packet filters: over time, there have been bugs that allowed attackers to defeat the intended security policy by sending unexpected packets that the packet filter doesn't handle quite the way it should. In addition, experience has shown that firewall policies rapidly become complex. Thus, even if a firewall's internal workings are entirely correct, the rules it enforces may not in fact accurately reflect the access controls that the operator believes they provide.

Finally, firewalls also embody *orthogonal security* meaning that it can be deployed to protect pre-existing legacy systems much more easily than other security mechanisms that have to be integrated with the rest of the system. A reference monitor that filters the set of requests, dropping unallowed requests but allowing allowed requests to pass through unchanged, is essentially transparent to the rest of the system: other components do not need to be aware of the presence of the reference monitor.

35.6. Firewall Advantages

- Central control: A firewall provides a single point of control. When security policies change, only the firewall has to be updated; we do not have to touch individual machines. For instance, when a new threat to an Internet service is discovered, it is often possible to very quickly block it by modifying the firewall's security policy slightly, and all internal machines benefit from this protection. This makes it easier to administer, control, and update the security policy for an entire organization.
- Easy to deploy: Because firewalls are essentially transparent to internal hosts, there is an easy migration path, and they are easy to deploy (incrementally, or all at once). Because one firewall can protect thousands of machines, they provide a huge amount of leverage.
- Solve an important problem: Firewalls address a burning problem. Security vulnerabilities in network services are rampant. In principle, a better response might be to clean up the quality of the code in our network services; but that is an enormous challenge, and firewalls are much cheaper.

35.7. Firewall Disadvantages

- Loss of functionality: The very essence of the firewalls concept involves turning off functionality, and often users miss the disabled functionality. Some applications don't work with firewalls. For instance, peer-to-peer networks have big problems: if both users are behind firewalls, then when one user tries to connect to another user, the second user's firewall will see this as an inbound connection and will usually block it. The observation underlying firewalls is that connectivity begets risk, and firewalls are all about managing risk by reducing connectivity from the outside world to internal machines. It should be no surprise that reducing network connectivity can reduce the usefulness of the network.

- The malicious insider problem: Firewalls make the assumption that insiders are trusted. This gives internal users the power to violate your security policy. Firewalls are usually used to establish a security perimeter between the inside and outside world. However, if a malicious party breaches that security perimeter in any way, or otherwise gains control of an inside machine, then the malicious party becomes trusted and can wreak havoc, because inside machines have unlimited power to attack other inside machines. For this reason, Bill Cheswick called firewalled networks a "crunchy outer coating, with a soft, chewy center." There is nothing that the firewall can do once a bad guy gets inside the security perimeter. We see this in practice. For example, laptops have become a serious problem. People take their laptop on a trip with them, connect to the Internet from their hotel room (without any firewall), get infected with malware, then bring their laptop home and connect it to their company's internal network, and the malware proceeds to infect other internal machines.
- Adversarial applications: The previous two properties can combine in a particularly problematic way. Suppose that an application developer realizes their protocol is going to be blocked by their users' firewalls. What do you think they are going to do? Often, what happens is that the application tunnels its traffic over HTTP (web, port 80) or SMTP (email, port 25). Many firewalls allow port 80 traffic, because the web is the "killer app" of the Internet, but now the firewall cannot distinguish between this application's traffic and real web traffic.

The fact that insiders are trusted has as a consequence that all applications that insiders execute will be trusted, too, and when such applications act in a way that subverts the security policy, the effectiveness of the firewall can be limited (even though the application developers probably do not think of themselves as malicious). The end result is that, over time, more and more traffic goes over ports nominally associated with other application protocols (particularly port 80, intended for web access), with firewalls gaining less and less visibility into the traffic that traverses them. As a result firewalls are becoming increasingly less effective.