addresses or host names that are trusted to assert that user's identity without a password. This allows any user to open up access to his own account without administrative supervision. It also gives this right to any program the user runs. If one machine is compromised inside your network, the firewall cannot help, since so many other machines, otherwise well-managed, will trust anything coming from the compromised machine. A lot of the above utilities are dangerous. For instance, ICMP ping can be exploited to find machines to break into. Sending an ICMP message to Alice saying that some range of addresses is 528 FIREWALLS 20.1 unreachable will cause Alice to hang up its connections to machines in the range specified by that ICMP message. Redirects can be used to cause a host to send traffic in a different direction, perhaps towards a compromised machine, making man-in-the-middle attacks easier. Finger can be used to find accounts to start attacking (searching the password space), or more ominously, to find personal information about users. 20.1 PACKET FILTERS The simplest form of firewall selectively discards packets based on configurable criteria, such as addresses in the IP header. For example, it might be configured to only allow some systems on your network to communicate outside, or some addresses outside your network to communicate into your network. For each direction, the firewall might be configured with a set of legal source and destination addresses, and it drops any packets that don't conform. This is known as address filtering. Address filtering can protect internal nodes that have no business communicating with the outside world. Perhaps they are willing to let anyone log in with the administrator password that came preconfigured by the manufacturer, but if the firewall prevents any connections to them, the exposure is closed. Packet filters usually look at more than the addresses. A typical security policy is that for certain types of traffic (e.g., email, web surfing), the rewards outweigh the risks, so those types of traffic should be allowed through the firewall, whereas other types of traffic (say telnet), should not be allowed through. To allow certain types of traffic between host A and B while disallowing others, a firewall can look at the protocol type in the IP header and the ports in the layer 4 (TCP or UDP) header, as well as anything at any fixed offset in the packet. For web traffic, either the source or destination port will be 80, because http (see §25.3 HTTP) has been assigned port 80. For email, either the source or destination port will be 25. Firewalls can be even fancier. Perhaps the policy is to allow connections initiated by machines inside the firewall, but disallow connections initiated by machines outside the firewall. Suppose machine A inside the firewall initiates a connection to machine B outside the firewall. During the conversation, the firewall will see packets from both directions (A to B as well as B to A), so it can't simply disallow packets from B to A. The way it manages to enforce only connections initiated by A is to look at the TCP header. TCP has a flag (called ACK) that is set on all but the first packet, the one that establishes the connection. So if the firewall disallows packets from B without ACK set in the TCP header, then it will have the desired effect, in general. Unfortunately, there are protocols such as ftp and X Windows that require B to make a TCP connection to A, even though it was A that initiated the ftp or X Windows session. With intimate 20.2 APPLICATION LEVEL GATEWAY 529 knowledge of the applications and implementations, the firewall can be configured to more or less do what you want. For instance, TCP has the client open a TCP connection for control traffic to the server, whereas the server then opens a TCP connection for data to the port from which the client initiated its TCP connection. The firewall can allow ftp sessions initiated from inside by allowing TCP connections initiated from outside as long as they are initiated to high numbered ports. (Dynamically assigned ports have high numbers, and a TCP client process will be assigned a dynamically assigned port.) Another approach is a stateful packet filter, i.e., a packet filter that remembers what has happened in the recent past and changes its filtering rules dynamically as a result. A stateful packet filter could, for instance, note that a connection was initiated from inside using IP address s, to IP address d, and then allow (for some period of time) connections from IP address d to IP address s. In practice, the stateful packet filter would be much more cautious than that. For example, in the case of ftp, the stateful packet filter can notice when a connection is opened from s to the ftp control port (21) at d, and remember the incoming ftp data port (say p) requested by s. (The stateful packet filter has to parse ftp control packets to do this.) As long as the ftp control connection stays opened, the stateful packet filter will allow connections from the ftp data port (20) at d to port p at s. Packet filters are frequently added as a feature to routers, allowing them to have minimal cost and impact on performance. 20.2 APPLICATION LEVEL GATEWAY Another way to protect your vulnerable network is by use of an application level gateway. The gateway could have two network adaptors and act as a router, but more often it is placed between two packet filtering firewalls, using three boxes (see Figure 20-2). The two firewalls are routers that refuse to forward anything unless it's to or from the gateway. Firewall F2 refuses to forward anything from the global net unless the destination address is the gateway, and refuses to forward anyyour net firewall F1 firewall F2 global net gateway Figure 20-2. Application Level Gateway 530 FIREWALLS 20.2 thing to the global net unless the source is the gateway. Firewall F1 refuses to forward anything from your network unless the destination address is the gateway, and refuses to forward anything to your network unless the source address is the gateway. To transfer a file from your network to the global network, you need to have someone from inside transfer the file to the gateway machine, and then the file is accessible to be read by the outside world. Similarly, to read a file into your network a user has to arrange for it to first get copied to the gateway machine. To log into a machine in the global network you first log into the gateway machine, and from there you can access machines in the remote network. An application level gateway is sometimes known as a bastion host. It must be implemented and configured to be very secure. The portion of the network between the two firewalls (a single LAN in Figure 20-2), is known as the DMZ (demilitarized zone). The gateway need not support every possible application. An example strategy is to allow only electronic mail to pass between your corporate network and the outside world. The intention is to specifically disallow file transfer and remote login. But electronic mail can certainly be used to transfer files. Sometimes a firewall might specifically disallow very large electronic mail messages, on the theory that this will limit the ability to transfer files. But often large electronic mail messages are perfectly legitimate, and any file can be broken down into small pieces. Because of such firewalls, many applications that post data for public retrieval on the Internet have automatic mail responders to which you can send an email message requesting the file, and the file will be sent back as an email message—several email messages, in fact, since such applications also realize that email messages have to be kept small in order to make it through the firewall. It is slow and painful for legitimate users to get files that way. Typically, the file arrives in several pieces. Each piece has to be extracted from the received email (to get rid of email headers). Then the pieces must be rearranged into proper order (since email tends to get out of order) and assembled into a file. A common application to support on a gateway is http, the protocol used for web surfing. Web browsers were designed with the concept of a proxy. If my machine can't connect to the outside world, I can configure it to forward all http requests to a gateway that will make the requests for me and send me back the results. This has some additional advantages: if many people in my company are requesting the same information, the gateway can cache the responses and not go back to the remote web site if the cache contains the requested information; and if web sites are trying to track my browsing using my IP address, it will be harder because they will get the same IP address (that of the gateway) for all requests coming from inside my company. 20.3 ENCRYPTED TUNNELS 531 20.3 ENCRYPTED TUNNELS A tunnel is a point-to-point connection in which the actual communication occurs across a network. Figure 20-3. Connecting a Private Network over a Public Internet Suppose the only reason you've hooked into the Internet is to connect disconnected pieces of your own network to each other. Instead of the example in Figure 20-3, you might have bought dedicated links between G1, G2, and G3, and trusted those links as part of your corporate network because you owned them. But it's likely to be cheaper to have the Gs pass data across the Internet. How can you trust your corporate data crossing over the Internet? You do this by configuring G1, G2, and G3 with security information about each other (such as cryptographic keys). All information between them is cryptographically protected. The most common protocol for encrypted tunnels is IPsec (see Chapter 15 IPsec: AH and ESP). The mechanics of the tunnel, from the IP point of view, is that when A sends a packet to C, A will launch it with an IP header that has source=A, destination=C. When G1 sends it across the tunnel, it puts it into another envelope, i.e., it adds an additional IP header, treating the inner header as data. The outer IP header will contain source=G1 and destination=G2. And all the contents (including the inner IP header) will be encrypted and integrity protected, so it is safe to traverse the Internet. G1, G2, and G3 use the Internet like some sort of insecure wire. You might want your users to be able to access the corporate network from across the Internet as well. Suppose B is some sort of workstation that can attach to the Internet in any location. To do this, B would create a tunnel with one of the Gs. This configuration is often referred to as a VPN (virtual private network). Another way to think about this is to consider your communication path to be a sequence of links, some of which are secure and some of which aren't. The right way to ensure secure communication is end-to-end, i.e., where the two communicating parties do all the security work and assume that the medium over which they are communicating is insecure. But the parties may not be capable of providing security, which is okay if all the links and routers in the path between them are Internet G1 G2 G3 A C B 532 FIREWALLS 20.4 secure. When some of the links are subject to eavesdropping, you can encrypt over those links. Only the endpoints of the tunnel need to do encryption and decryption. 20.4 COMPARISONS A packet filter can do its job without requiring any changes to the software in any of the communicating nodes. It prevents some conversations, but the allowed conversations proceed normally (assuming the packet filter isn't using some bizarre rules such as disallowing any packet that has the ASCII pattern for pornography or some other suspicious word anywhere inside). An application level gateway also can be deployed without changing any software, but it is visible to the users. They have to log into the firewall, and from there connect to whatever they are allowed to connect to. Application level gateways can be much more powerful than packet filters. For instance, because they are application-aware, they might be able to look at the data inside email messages to scan for viruses. A VPN can also be deployed without changing any of the nodes inside your corporate network, but a node such as B in Figure 20-3 will need special software in order to create an encrypted tunnel to one of the Gs. Packet filters and application level gateways are intended to allow limited communication between nodes inside your network and "outsiders". VPNs are intended to maintain an isolated private network that allows full access to "insiders", and no access to "outsiders". 20.5 WHY FIREWALLS DON'T WORK Firewalls alone (without also doing end-to-end security) assume that all the bad guys are on the outside, and everyone inside can be completely trusted. This is, of course, an unwarranted assumption. Should employees have access (read and write) to the salary database, for instance? Even if the company is so careful about hiring that no employee would ever do anything bad intentionally, firewalls can be defeated by somehow injecting malicious code into the corporate network. This can be done by tricking someone into downloading something from the Internet or launching an executable from an email message. It is quite common for an attacker to break into one system inside your firewall, and then use that system as a platform for attacking other systems. Someone once described firewall protected networks as “hard and crunchy on the outside; soft and chewy on the inside”. 20.6 DENIAL-OF-SERVICE ATTACKS 533 Firewalls often make it difficult for legitimate users to get their work done. The firewall might be configured incorrectly, or not recognize a new legitimate application. And if the firewall allows anything through (say email or http), people figure out how to do what they need to do by disguising it as traffic that the firewall is configured to allow. File transfer is common by sending the file in email. To prevent this, some firewalls are configured to not allow very large email messages, in which case the users circumvent the firewall by breaking the file into small enough chunks, each chunk sent as a different email message. It’s clumsy and annoying, but it works. The ironic term for disguising traffic in order to fool a firewall is to carry your traffic in a firewall friendly protocol. Since firewalls commonly allow http traffic (since it’s the protocol used for browsing the web), there are many proposals for doing things over http, from network management to carrying IP over http, which would allow any traffic through! Firewall-friendly? The whole point is to defeat the best efforts of the firewall administrator to disallow what you are doing! It isn’t somehow “easier” for the firewall to carry http traffic than any other. The easiest thing for the firewall would be to allow everything or nothing through! Just as breaking a large file into lots of pieces to be individually carried in separate emails is inefficient, having protocols run on top of http rather than simply on top of IP is also inefficient in terms of bandwidth and computation. 20.6 DENIAL-OF-SERVICE ATTACKS A denial-of-service attack is one in which an attacker prevents good guys from accessing a service, but does not enable unauthorized access to any services. In the naive old days, security people dismissed the prospect of denial-of-service attacks as unlikely, since the attacker had nothing to gain. Of course that turned out to be faulty reasoning. There are terrorists, disgruntled employees, and people who delight in causing mischief for no good reason. In the earliest types of denial-of-service attacks, the attacker repeatedly sent messages to the victim machine. Most machines at the time were vulnerable to this sort of attack since they had resources that could easily be depleted. For instance, the storage area for keeping track of pending TCP connections tended to be very limited, on the order of, say, ten connections. The probability of ten legitimate users connecting during a single network round trip time was sufficiently small that ten was a reasonable number. But it was easy for the attacking machine to fill up this table, even if the attacking machine was attached to the Internet with a low-speed link. To avoid being caught at this mischief, it was common for the attacker to send these malicious packets from forged source addresses. This made it difficult to find (and prosecute) the attacker, and it made it difficult to recognize packets from the malicious machine and filter them at a firewall. 534 FIREWALLS 20.7 In defense, people started deploying routers with the capability of doing sanity checks on the source address. These routers could be configured to drop packets with a source address that could not have legitimately come from the direction from which the packet was received. And end node storage areas were increased so that a single attacker, at the speeds at which such attackers were typically connected to the Internet, could not fill the pending TCP connection table. Another level of escalation was to send a single packet that caused a lot of legitimate machines to send messages to the victim machine. An example of such a packet is a packet transmitted to the broadcast address on some LAN, with the packet’s source address forged to the address of the victim’s machine, asking for all receivers to respond. All the machines on that LAN will send a response to the victim’s machine. Such a mechanism magnifies the effect the attacker can have from his single machine, since each packet he creates turns into n packets directed at the victim machine. The next level of escalation is known as a distributed denial-of-service attack. In this form of attack the attacker breaks into a lot of innocent machines, and installs software on them to have them all attack the victim machine. These innocent machines are called zombies or drones. With enough zombies, any machine can be made inaccessible, since even if the machine itself can process packets as fast as they can possibly arrive (i.e., the speed of the wire attaching it to the Internet), the links or routers in front of that machine can be overwhelmed. Since the packets are coming from hundreds or thousands of innocent machines, it is hard to distinguish these packets from packets coming from legitimate users. 20.7 SHOULD FIREWALLS GO AWAY? Even if the world deployed end-to-end security everywhere, firewalls still serve an important purpose. The can keep intruders from doing the equivalent of jiggling all the doors inside your building to see if someone accidentally left one unlocked. And they can help protect against denial-of-service attacks by keeping nuisance traffic off your net. 535 21 MORE SECURITY SYSTEMS With every release, software gets more complex and less secure until the only security left is job security. —Al Eldridge A lot of systems have been deployed with security features, and it is rather surprising how different they can be from one another. In this chapter we give an overview of some of the systems we have found interesting. Some of them are quite old and have died out, but we describe them anyway because they have some interesting features, either positive or negative. Since these are not standards, we are not attempting to describe them in sufficient detail to implement them, but rather only enough to describe their interesting concepts. Why haven't these systems been fully documented in the public literature? One reason is that lack of documentation makes it harder for a relatively unsophisticated cracker to exploit security flaws that might exist. Sometimes it is because a company considers the technology a trade secret. Often it is just because careful documentation of the security features is a lot of work and has little commercial benefit. 21.1 NETWARE V3 Novell's networking package is known as NetWare. This section describes the cryptographic security in Version 3. Version 4 security was enhanced to use public key cryptography. We describe Version 4 security in §21.2 NetWare V4. Each server has a database consisting of information about each authorized client. The information includes the user's name, the salt, and the hash of the user's password with the salt. The salt is not actually a random number, but instead is a 32-bit user ID assigned by the server when the user is installed in the server database. The salt serves two purposes. It makes the hash of Alice's 536 MORE SECURITY SYSTEMS 21.1 password server-dependent, so that if Alice uses the same password on multiple servers, the database at each server will, with high probability, store a different hash of her password. It also makes the hash of Ted's password different from the hash of Alice's password even if they have the same password. We'll use X to designate the hash of the password stored at the server. Alice tells her workstation her name, password, and the name of the server to which she'd like to log in. The workstation sends Alice's name to the server. The server sends the salt value and a random challenge R to the workstation. The workstation performs the hash of the password Alice typed with the salt, and now both the workstation and the server know the same secret value X (assuming Alice typed her password correctly). Now the workstation and the server each computes hash(X,R) to get a value we'll call Y. The workstation transmits Y to the server, and if it matches hash(X,R), then the server considers Alice authenticated. At this point the workstation and the server each compute Z=hash(X,R,constant string). Z is what they'll use as a secret session key. After the authentication exchange each packet is integrity-protected by performing a hash of the username1 userID1=salt1 hash(salt1,password1)=X1 username2 userID2=salt2 hash(salt2,password2 ...)=X2 Figure 21-1. Server Password Database Alice Alice,pwd,Bob Workstation Alice Server Bob knows X = hash(password,salt) R,salt picks random R computes X' = hash(pwd, salt) Y' = hash(X',R) computes Y = hash(X,R) Y' computes Z' = hash(X',R,k) to use as the session key verifies Y = Y' computes Z = hash(X,R,k) to use as the session key Figure 21-2.

NetWare V3 Authentication 21.2 NETWARE V4 537 packet and Z. Once the workstation receives an integrity-protected packet from the server, mutual authentication has occurred, because the server has proven it knows Z. Actually, for performance reasons, only the first 52 bytes of the packet are cryptographically checksummed. Checksumming the first 52 bytes of the packet is sufficient to answer the threats of eavesdroppers gaining information from the authentication exchange and intruders hijacking a session after the initial authentication exchange. It does not prevent an attacker from removing a packet in transit and substituting one with modified data after the first 52 bytes, but this is a far less realistic threat, particularly if the two communicating parties are on the same LAN. Why did Novell choose 52 bytes rather than some nice power of 2? Because 52 bytes is long enough to protect all the information necessary to prevent hijacking, and there's additional information appended to those 52 bytes to form a single 64-byte message digest block. The extra information is the 4-byte packet length and the 8-byte secret. Why does the server store the secret value X rather than simply the user's password? A workstation that knew the value X would be able to impersonate Alice without knowing Alice's password. But in terms of real-world threats, it is a lot more secure to store X than Alice's password. If Trudy were to steal the server's database and acquire X, she'd only be able to exploit X if she were able to acquire the client authentication code and modify it to use X rather than compute it from what the user types. In contrast, if the server stored the user's password directly, then if Trudy stole the server's database, she'd be able to impersonate Alice directly by typing her password at unmodified client code. Storing X rather than the password will not prevent Trudy from doing an off-line password-guessing attack, but if Alice chose her password wisely, then an off-line password-guessing attack is unlikely to succeed. Another reason it is more secure to store X than the user's password is because of the salt. Let's assume Trudy can modify the client code, and can therefore use X to impersonate Alice directly, and therefore X is as security-sensitive as Alice's actual password. Since X is very likely to be different at each server, even if Alice's password is the same, then Trudy will only be able to impersonate Alice at the server from which Trudy acquired the database. 21.2 NETWARE V4 There were two major reasons for modifying security in Version 4 of NetWare. The first was to utilize public key cryptography, which has the potential to make theft of the server database less of a security issue. The other was to simplify management. Instead of having a database entry for Alice separately managed at each server Alice is authorized to use, NetWare Directory Services (NDS) stores Alice's security information, and servers retrieve the information from NDS. 538 MORE SECURITY SYSTEMS 21.2 Each user has an RSA key, but of course the user does not remember that. The user remembers a password. NDS stores Alice's private key encrypted with her password. To prevent off-line password guessing, Alice's workstation must prove knowledge of her password before NDS will transmit her encrypted private key. The pre-authentication phase, where her workstation proves it knows her password, is similar to NetWare V3. NDS stores a hash of Alice's password and salt. In V3 we said the salt was useful because it made the hash of Alice's password different on different servers. That case does not apply for V4 since Alice's entry is only stored in one place—NetWare Directory Services. But the salt is still useful, since it means that an intruder who reads the directory service database has to conduct separate dictionary attacks against each account. First Alice logs into the workstation by typing her name and password. The workstation then authenticates on Alice's behalf to NDS in order to obtain Alice's private key. For security as well as performance, once the workstation acquires Alice's private key from NDS, it converts it into a temporary Gillou-Quisquater (GQ) key (see §21.2.1 NetWare's Guillou-Quisquater Authentication Scheme) and then forgets her password and private RSA key. Authentication with GQ is public key authentication, but there are two important differences from authenticating with RSA. • A private RSA key lasts forever. A GQ key, as NetWare uses it, expires—the workstation determines when the GQ key should expire when it creates the GQ key from the RSA key. If some untrustworthy software steals Alice's password or private RSA key, it can impersonate her forever (or until she notices what has happened and changes her long-term private key). Conversion to a GQ key limits the amount of time in which damage can be done. Note that this is the same reasoning that led the Kerberos designers to introduce the concept of a TGT and session key for the user rather than using the user's master key throughout a session. • GQ authentication is faster than RSA for the client (Alice, the thing proving her identity) but slower for the server (Bob, the thing authenticating Alice). The NetWare designers envisioned slow PCs for clients and fast PCs for servers. NetWare could have implemented the conversion of the long-term secret into a short-term secret differently. Rather than generate a GQ key pair, the workstation could choose a new RSA key pair, generate a certificate for it by signing it with the user's long-term private RSA key, and then forget the user's long-term secret. That is what DASS does, as we'll see in §21.4 DASS/SPX. But generating a temporary GQ key is much faster than generating an RSA key, so the NetWare scheme has higher performance during login. Once the workstation has acquired a GQ key for Alice, it stores it for the duration of her session and uses it to authenticate on her behalf to any servers she accesses during her session. If Alice asks to log into server Bob, then Bob must retrieve Alice's public RSA key from NDS. Alice's temporary GQ key works with her permanent public RSA key, in the sense that a signature generated with her temporary GQ key is verified using her permanent public RSA key. 21.2 NETWARE V4 539 To summarize, the information stored for Alice in NDS includes Here is the protocol in which Alice's workstation obtains her encrypted private RSA key from NDS: First Alice logs in and her workstation communicates with NDS in order to obtain her private RSA key, which her workstation will use to generate a finite-lifetime GQ key for her: 1. Alice types her name and password to her workstation. 2. The workstation uses that information to authenticate Alice on her behalf to NDS, in an authentication exchange nearly identical to NetWare V3, except for the final message of the pre-authentication. The final message in V3 consists simply of Y, which is hash(X,R), where R is the challenge transmitted by the server. In V4, in addition to transmitting Y, the workstaAlice's name hash(salt,password)=X used to verify that Alice's workstation knows her password before NDS transmits her encrypted private RSA key public RSA key encrypted private RSA key encrypted using a hash of Alice's password Figure 21-3. NetWare V4 NDS Database Entry for Alice Alice Alice,pwd Workstation Alice NDS knows Alice's X = hash(password,salt) encrypted private key public key R,salt picks random R computes X' = hash(pwd, salt) Y' = hash(X',R) chooses large random R2 retrieves public key of NDS (if not cached) computes Y = hash(X,R) {Y ',R2}NDS verifies Y = Y' Y{encrypted private key ⊕ R2} Figure 21-4. NetWare V4 Pre-authentication 540 MORE SECURITY SYSTEMS 21.2.1 tion transmits a large random number R2 (chosen by the workstation), which will be used by NDS for concealing NDS's reply to the workstation from eavesdroppers. The entire message (Y and the random number R2) is encrypted with the NDS's public key. Encrypting the message with NDS's public key means that an eavesdropper cannot use Y and R to verify password guesses off-line (as it could in V3). 3. Once NDS is assured that the workstation knows Alice's password, NDS transmits Alice's encrypted private RSA key to the workstation. To prevent off-line password guessing by eavesdroppers, the encrypted private RSA key is ⊕'d with the random number R2 sent in the previous step, and then encrypted with Y. 4. The workstation decrypts Alice's private RSA key. It immediately turns it into a GQ key with an expiration time, and then forgets the password and private RSA key. Here is what occurs when Alice accesses a network resource, say a server named Bob. 1. Alice asks to log into Bob. 2. Bob obtains Alice's public key from NDS. 3. Alice is authenticated in a public key authentication handshake in which the workstation uses Alice's GQ key and Bob uses her public RSA key. 4. A session key is also established as part of the authentication handshake between the workstation and Bob, and that session key is used to integrity-protect the initial portion of each of the packets of the Alice-Bob exchange (just like in V3). 21.2.1 NetWare's Guillou-Quisquater Authentication Scheme NetWare V4 authentication is not based on RSA, but rather on a variant of an algorithm by Guillou and Quisquater. We'll call NetWare's variant GQ. GQ requires that each user have an RSA key pair. The user's private RSA key is used to generate a private GQ key with a temporary lifetime. Then the workstation forgets the user's private RSA key, and future authentication is done using only the GQ key and the user's public RSA key. The GQ key can only do signatures (it can't encrypt or decrypt). Signature verification uses the public RSA key. This is how the workstation acquires a private GQ key for the user: 1. The workstation acquires the user's private RSA key. 2. The workstation generates a message, m0, which includes the validity interval for the GQ key it is about to create. 21.2.1 NETWARE V4 541 3. The workstation signs m0 with the user's private RSA key. This signature of m0 is the user's private GQ key, Q. The workstation must not divulge this quantity. 4. The workstation forgets the user's private RSA key, but remembers m0 (which is used, together with the user's public RSA key, to verify a GQ signature) and Q (the RSA signature on m0). This is how the GQ key is used to sign a message: 1. To sign a message, say m, the workstation needs to know Q, m, and the user's public RSA key. These quantities are all fed into a magic function that outputs a signature. As used by NetWare, m is the challenge supplied by the server. 2. The signature verification function requires as input m, the signature on m, m0, and the user's public RSA key. With these inputs, the signature verification function decides whether or not the signature is valid. The mathematics involved are ugly and unintuitive, but we'll document them here for completeness. We'll describe the mechanics, but to understand why it's secure, you should read [GUIL88]. • The user's RSA key pair is ⟨e,n⟩ for the public key, and ⟨d,n⟩ for the private key. • Q = m0 d mod n is the RSA signature on m0. To sign a message m, do the following: 1. Generate eight random numbers r1,...r8 in the range [2, n−2]. 2. Compute eight quantities x1,...x8, where xi = ri e mod n. 3. Compute MD4(m|x1|x2|...|x8). 4. Break the 128-bit message digest computed in the previous step into 8 16-bit quantities and call the pieces c1,...c8. 5. Compute eight quantities y1,...y8, where yi = riQci mod n. 6. The GQ signature of m consists of x1,...x8, and y1,...y8. To verify the GQ signature of m, do the following: 1. Receive m0, m, x1,...x8, and y1,...y8. 2. Reliably find the user's public key ⟨e,n⟩. 3. Compute MD4(m|x1|x2|...|x8). 542 MORE SECURITY SYSTEMS 21.3 4. Break the 128-bit message digest computed in the previous step into 8 16-bit quantities and call the pieces c1,...c8. 5. Accept the signature if for each i, yi e = xim0 ci mod n. Why does this work? yi = riQci mod n (this is how yi got computed in the signing algorithm) yi e = (riQci) e = ri e (Qe) ci mod n ri e = xi mod n (this is how xi got computed in the signing algorithm) Q = m0 d mod n, so Qe = m0 mod n yi e = (riQci) e = ri e (Qe) ci = xim0 ci mod n, which is what is being tested 21.3 KRYPTOKNIGHT KryptoKnight was developed at IBM. Its product name is NetSP (Network Security Program). We really like the name KryptoKnight, so we'll use that rather than the probably more correct term NetSP. It is a secret key based authentication and key establishment system similar to Kerberos. There are several papers describing the concepts behind it [MOLV92, BIRD93, HAUS94, BIRD95, JANS95]. Ways in which KryptoKnight is similar to Kerberos: • They both use KDCs, though KryptoKnight calls them Authentication Servers. • There can be multiple realms, which KryptoKnight refers to as domains. • There can be replicated KDCs within a single realm. • Upon first login, the user's password is converted into a long-term key, which is used to obtain a TGT from the KDC, and then both the password and long-term key are forgotten by the workstation. • Like Kerberos V5, to prevent an intruder from simply asking the KDC for a TGT for Alice and then using that TGT to mount an off-line password-guessing attack, Alice's workstation needs to prove to the KDC that it knows her password before the KDC will transmit a TGT. KryptoKnight differs from Kerberos in the following ways: 21.3.1 KRYPTOKNIGHT 543 • KryptoKnight uses message digest functions rather than DES for authentication and even for encryption of tickets. Using message digest functions rather than DES in authentication offers performance advantages and may make it easier to comply with export controls. • With Kerberos, Alice must obtain a ticket to Bob before initiating a conversation. In KryptoKnight, there is the option for Alice to alert Bob directly that she would like to converse, and have Bob do the handshaking with the KDC(s). One example where this functionality is vital is where Bob is a firewall and prevents Alice from accessing anything inside the network, including the KDC, until Alice has been authenticated by Bob. Another example is where it is less expensive for Bob than Alice to communicate with the KDC, for instance if Alice has a very-low-bandwidth link into the network. • Like Kerberos V4 but unlike Kerberos V5, interrealm authentication between Alice and Bob is only allowed when Alice's KDC and Bob's KDC have been administratively set up with a shared key; i.e., transit realms are not allowed. • KryptoKnight does not rely on synchronized clocks, and can instead use random number challenges. • The KryptoKnight designers were very careful to minimize the number of messages, lengths of messages, and amount of encryption (for example, Kerberos V4 unnecessarily encrypts an already-encrypted ticket). Ironically, in some cases KryptoKnight uses more messages than Kerberos, but this is because of not relying on synchronized clocks and therefore requiring an extra message to transmit a challenge. • Other than pre-authentication, KryptoKnight does not bother with any of the features that got added to Kerberos V5, such as delegation, authorization data, postdated and renewable tickets, and hierarchical realms. 21.3.1 KryptoKnight Tickets In Kerberos, when Alice asks for a ticket to Bob, she is given a quantity, encrypted with Bob's key, containing the session key Alice and Bob will share and other information such as Alice's name and the expiration time of the ticket. In a ticket, the only information that requires encryption is the session key, but the rest of the information has to be protected from modification. KryptoKnight tickets are rather elegant. Indeed in a KryptoKnight ticket, the only thing that is encrypted is the session key. And it is encrypted using a message digest function, rather than by secret key encryption. The most important fields in a ticket are Alice's name, the expiration time, and the encrypted session key. The session key is encrypted by ⊕ing it with a quantity that Bob and the KDC, but no 544 MORE SECURITY SYSTEMS 21.3.2 one else, can compute. That quantity is the message digest of the concatenation of the secret Bob shares with the KDC, Alice's name, and the expiration time of the ticket. If someone were to modify the name or expiration time in the ticket, then when Bob computes the message digest of his secret|Alice|expiration time, he'll get a totally different value, and the ticket will decrypt to something totally unpredictable, and therefore useless to an intruder. So this encryption method for encrypting the session key also serves as integrity protection for the unencrypted fields in the ticket. 21.3.2 Authenticators During the authentication handshake, Alice and Bob send each other something functionally equivalent to Kerberos authenticators. The Kerberos authenticator is a timestamp DES-encrypted with the shared secret. In KryptoKnight, it is instead a nonce, generated by the other side as a challenge. Alice "signs" Bob's challenge by prepending the secret she and Bob share and returning the message digest of the result. Similarly, Bob signs Alice's challenge. KryptoKnight provides all Kerberos V4 functionality (except encryption of user data) using message digest functions rather than secret key cryptography. 21.3.3 Nonces vs. Timestamps We said before that KryptoKnight does not require synchronized time. Actually, the authentication protocols can use either timestamps or nonces. There is a trade-off between nonces and timestamps. Timestamps require somewhat-synchronized clocks. If time is only used to indicate expiration time, it doesn't need to be very accurate. But if an encrypted timestamp is sent as part of the authentication handshake, then time synchronization is more critical. Relying on timestamps also raises another security vulnerability, since the timekeeping mechanism has to be kept secure (or else an intruder can accomplish bad things by setting the clocks back a month or so). The disadvantages of using nonces instead of timestamps are that nonces require additional messages, and they do not allow Alice to hold onto a ticket to Bob and use it multiple times (see below). Because sometimes one mechanism is preferable to the other, KryptoKnight tickets contain both a nonce and an expiration time, though usually only one of the fields will be security-relevant in any given exchange. Authenticators can also be based on time or nonces. How does KryptoKnight work when it uses nonces instead of timestamps in the authentication handshake? The timestamp in a ticket to Bob assures Bob that Alice has obtained the ticket recently, because the expiration time has not yet occurred. In order to use a nonce instead of an expiration time, Alice needs to ask Bob for a nonce before she can request a ticket to Bob. She sends Bob's nonce to the KDC and it puts it into the ticket. Bob has to remember the nonce he gave to Alice, and make sure that the ticket is based on his nonce. 21.3.4 DASS/SPX 545 Actually the nonce does not need to be transmitted as part of the ticket. The nonce, instead of being explicitly in the ticket, is just one of the fields (along with Alice's name and the secret that Bob and the KDC share) which get input into the message digest that the KDC $\oplus$s with the session key. With the nonce mechanism, once Alice is removed from the KDC database, she is unable to obtain more tickets. With the expiration time mechanism, once Alice obtains a ticket there is no way to revoke it until the expiration time occurs. 21.3.4 Data Encryption Once a session is set up, if encryption is desired, the data is encrypted with a secret key cryptographic scheme. The prototype used DES. The product they shipped used a weakened secret key algorithm with a 40-bit key, which was the price IBM paid to make the product freely exportable from the U.S. 21.4 DASS/SPX DASS stands for Distributed Authentication Security Service. It was deployed as SPX, pronounced Sphinx, and nobody has come up with an acronym expansion for SPX. It was developed at Digital Equipment Corporation and documented in RFC 1507 and [TARD91]. SPX is technically the product name, whereas DASS is the architecture, rather like NetSP and KryptoKnight. We'll use the term DASS because we think it is the more commonly used term for it in the security community, though we haven't done any official polls. 21.4.1 DASS Certification Hierarchy DASS has a certificate hierarchy similar to what we describe in §13.3.8 Bottom-Up with Name Constraints. Conceptually there's a CA responsible for each node in the naming hierarchy. Each CA signs a certificate for its parent and for each of its children. These are known as up certificates and down certificates, respectively. DASS also allows cross certificates, where a CA can sign a certificate for any other CA, so that authentication can short-circuit the hierarchy for performance or security reasons. Also, there does not need to be a distinct CA for every node in the tree. One CA could be responsible for many parts of the naming tree. 546 MORE SECURITY SYSTEMS 21.4.2 DASS uses X.509 syntax for certificates and originally envisioned storing certificates in an X.500-style directory service, but since one was not deployed, the DASS designers invented their own certificate distribution service, which they called a CDC, for Certificate Distribution Center. It not only stores certificates, but also stores encrypted private keys. Certificates are publicly readable. To obtain the encrypted private key, the client machine must prove knowledge of the user's password, and must know the public key of the CDC in order for this exchange to be secure. DASS would benefit from one of the protocols described in §10.4 Strong Password Credentials Download Protocols, but they did not exist at the time of its design. 21.4.2 Login Key Once the workstation retrieves the user's private key P, it immediately chooses what DASS refers to as a login RSA key pair and then generates a certificate, which we'll call the login certificate, signing it with P, stating the user's login public key and an expiration time. Then it forgets P and remembers only the login private key and the login certificate. This is similar to obtaining a TGT in Kerberos, or obtaining a GQ key in NetWare V4. The DASS method is lower-performance during login than the NetWare method because it takes more computation to generate an RSA key pair than a GQ key. During authentication the DASS method is lower-performance on the client side because RSA signature generation is slower than GQ signature generation. But the DASS method is higher-performance on the server side, because with small public exponents, RSA signature verification is faster than GQ signature verification. The DASS designers envisioned a world where workstations had cycles to burn while servers were overburdened. The NetWare designers envisioned the opposite. 21.4.3 DASS Authentication Handshake Let's say user Alice accesses resource Bob. We won't distinguish between Alice's workstation and Alice. Obviously it's Alice's workstation that is performing the cryptographic operations, but we'll refer to the two ends of the conversation as Alice and Bob. The initial authentication handshake is a mutual authentication handshake based on public keys. Alice knows Bob's public key by looking up and verifying his certificate in the CDC. Bob knows Alice's long-term public key by looking up her certificate in the CDC (and verifying the certificate signature). But the key in Alice's certificate is not the key Alice will be using. Alice has to transmit her login certificate to Bob, and Bob, after following the certificate chain, now knows Alice's public key for this login session. In the process of doing an authentication handshake, Alice and Bob establish a shared secret key. Future cryptographic operations, such as encryption of the conversation, are done using that shared secret key. For performance reasons, DASS is designed so that subsequent authentication 21.4.3 DASS/SPX 547 exchanges between the same two parties can also be done without any public key operations, using only the shared secret key. It is interesting how the secret key is established between Alice and Bob. Alice (her workstation of course) chooses a DES key SAlice-Bob at random, encrypts SAlice-Bob with Bob's public key, and signs the result using her login private key (for integrity protection). We'll use X to designate the encrypted signed SAlice-Bob. Alice sends her login certificate and X to Bob, along with an authenticator proving she knows the key SAlice-Bob. Bob then does the following: • gets Alice's long-term public key, by retrieving and verifying her certificate from the CDC • verifies Alice's login certificate by using her long-term public key • extracts Alice's login public key from her login certificate • reverses Alice's signature on X by using her login public key • reverses the encryption of X using his own private key, getting SAlice-Bob • verifies the authenticator by decrypting it using SAlice-Bob and checking whether the time is valid • encrypts the timestamp using SAlice-Bob and returns it to provide mutual authentication For performance reasons, both Bob and Alice cache both SAlice-Bob and X. If Alice accesses Bob again, Alice will transmit X again, with a new authenticator. So why does Bob need to remember X? The reason he does is to save himself the trouble of cryptographically unwrapping X again in order to obtain SAlice-Bob. Alice might have forgotten SAlice-Bob and chosen a new secret. So Bob has to check if the X he receives is the same as the one he has cached, but if they match he can assume he's using the same key as before with Alice. If they don't match, or if Bob has forgotten the cached information, the authentication handshake works just fine. It just involves the extra computation of Bob cryptographically unwrapping X. Why does Alice need to cache X? Since Alice does not know whether Bob has cached SAlice-Bob, she has to send X again so that the authentication handshake can proceed whether or not Alice login certificate, X, authenticator SAlice-Bob{timestamp} Bob (if mutual authentication) Figure 21-5. DASS Authentication 548 MORE SECURITY SYSTEMS 21.4.4 Bob has cached SAlice-Bob. Since she has to perform cryptographic operations in order to regenerate X, it saves her time if she caches X. An interesting feature of DASS is that the authentication handshake is designed to work in a single message in the case of one-way authentication, and two messages in the case of mutual authentication. The price it pays to reach this theoretical minimum number of messages is that it requires roughly synchronized clocks, like Kerberos. 21.4.4 DASS Authenticators When Alice initiates a connection to Bob, she sends an authenticator. When mutual authentication is required, Bob sends an authenticator back to Alice. The authenticator is very different in the two directions. In the Alice→Bob direction, Alice sends Bob an unencrypted timestamp and a MAC. The MAC is computed by doing a DES CBC residue using the secret key SAlice-Bob and an IV of 0, computed over the timestamp and the network layer source and destination addresses extracted from the network layer header. The authenticator Bob sends back to Alice is the timestamp encrypted with SAlice-Bob. 21.4.5 DASS Delegation DASS is designed so that if Alice wants to delegate to Bob in addition to performing an authentication handshake with him, the delegation can be piggybacked on the authentication handshake. Recall that during the authentication handshake (without delegation), Alice sends Bob her login certificate, the magic quantity X, and an authenticator. Remember that X is the session key SAlice-Bob encrypted with Bob's public key and signed with Alice's login key. If delegation is being done as well as authentication, there's one less public key operation, because instead of sending X (which is the session key encrypted with Bob's public key and then signed with Alice's private key), Alice just sends the session key encrypted with Bob's public key. In order to delegate to Bob, she sends Bob her login private key encrypted with SAlice-Bob. As in the non-delegation case, she also sends an authenticator proving she knows SAlice-Bob. It takes some thought as to why, in the delegation case, it isn't necessary to sign the encrypted session key, whereas it is necessary in the non-delegation case. The DASS designers really enjoyed standing on their heads to minimize public key operations (see Homework Problem 1). 21.4.6 LOTUS NOTES SECURITY 549 21.4.6 Saving Bits While DASS was intended to work with a variety of protocols, and was only actually deployed with TCP/IP, the DASS designers wanted to integrate their protocols with DECnet Phase IV, which introduced some interesting constraints. They could only piggyback the security information on existing transport layer connection messages if the additional information did not make the transport layer connection messages longer than an Ethernet packet (approximately 1500 bytes). Alice initiates contact with Bob with a connect request message. Bob replies with a connect confirm message. There were only 16 spare bytes in a connect confirm, and DASS managed to only use 8 of them. DASS needed more space in a connect request, because Alice sends her login certificate, X, and an authenticator. Luckily, there was enough room. But there would not have been enough room if the DASS designers didn't spend a lot of time doing clever compression of the data they needed to send. The exact packet formats are not important, but the most dramatic encoding trick they played was the encoding of Alice's login private key when she transmits it to Bob for delegation. Recall that an RSA public key consists of ⟨e,n⟩, where n is the modulus and e is the public exponent. An RSA private key consists of ⟨d,n⟩. Alice's login certificate contains ⟨e,n⟩. You'd expect Alice to send d in order to give Bob her login private key. But instead, she sends p, which is the smaller of the factors of n, and will be about half as big as d (so it will be about 256 bits instead of 512). Bob has to divide n by p to get q, and then use Euclid's algorithm to calculate d (given that he knows e and n from her login certificate). Actually, when doing delegation it's friendly to pass more than just d, since if Bob knows n's factorization he can do private key operations more efficiently. Given p, Bob can compute all the information that would have been good to send him. If instead, Alice were to pass all the information so that Bob didn't need to do any computation, it would take about 2½ times the size of the modulus (so about 1300 bits). The DASS method makes Bob do some work up front, but then he can sign efficiently on Alice's behalf. DASS does use ASN.1 encoding, which might seem surprising since its designers were so worried about encoding efficiency. But they were very careful to avoid sending redundant information, and they were clever in their use of ASN.1 syntax. By using IMPLICIT and other tricks they avoided the size explosion found in Kerberos V5 and X.509. 21.5 LOTUS NOTES SECURITY Lotus Notes has a security system without a cute name, or really any name at all. Lotus Notes was developed by a company called Iris Associates, which was acquired by Lotus (the name was 550 MORE SECURITY SYSTEMS 21.5.1 changed to Lotus Notes after the acquisition, not due to some amazing coincidence). Later Lotus was assimilated by IBM, who changed the name of the server end of the product to Lotus Domino. The first three versions had the clever names version 1, version 2, and version 3. In the first edition of this book, we boldly speculated that the version they were working on at the time would be called version 4. But we were wrong. The next two versions were called R4 and R5 (where R stands for "release"). Apparently there is some subtle marketing distinction between a version and a release. The next version about to be released as of the writing of this book is called "release 6", and the marketeers have sternly said that it is NOT to be called R6. While the security has evolved over the years—in particular the adding of S/MIME and SSL support for interoperation with other products—the Notes proprietary public key infrastructure (they prefer to call it pre-standard) has remained substantially the same. Like the other systems we've discussed, Lotus Notes security provides for mutual authentication and establishment of a shared session key. It also provides for electronic mail security and for digitally signing active content. It is public key based, though just like all security systems, it uses public keys in conjunction with secret keys for encryption. The algorithms it uses are RSA, MD2, RC2, and RC4 (see glossary). 21.5.1 ID Files An ID file contains the sort of user-specific security information that most other deployed schemes store in a directory service. Lotus Notes security assumes that a user Alice will carry her ID file on a floppy or smart card, or have it stored in nonvolatile memory on her workstation. Because an ID file might be stolen or lost, the ID file is encrypted with Alice's password. There are interesting trade-offs between the ID file scheme and a directory service based scheme. The advantages of storing the user's security information in a directory service are: • It is more convenient. If ID files are kept on floppies or smart cards, users have to carry something around, and if they lose it or forget it they can't log into the network. If ID files are kept on a workstation, then the user is restricted to using that workstation. It is possible to store a given user's ID file on more than one workstation, but it becomes infeasible in some environments to maintain a user's ID file on every possible workstation that that user might use. And operations like password changes become very awkward. • It might be more secure. A directory service based scheme, if properly designed, can make it difficult for an intruder to capture a quantity with which to do password guessing. If an ID file is on a floppy, someone who steals the floppy can do an off-line password-guessing attack. If the ID file is stored on a workstation, anyone who can physically access the workstation can read out the information and do off-line guessing. In the case of a smart card, however, it is possible to design a smart card that will refuse to divulge its contents, and 21.5.2 LOTUS NOTES SECURITY 551 which will limit the number of password guesses by disabling itself after too many incorrect guesses. In release 6, Lotus Notes offers the option of storing the ID file on a server and accessing it with a protocol similar to the ones in §10.4 Strong Password Credentials Download Protocols. The advantages of the ID file scheme are: • It might be more secure, because it provides "two-factor authentication", based on both what you have and what you know. • It requires less of the network to be operational. Sometimes the environment is quite primitive, such as when the network itself is mostly broken and you need security for network management in order to fix the network, or when directory service replicas need to authenticate one another, or when a bunch of people with laptops show up at a meeting and want to form a private little network. • It might be more secure because someone who had physical access to a directory service replica could read the directory service database and do off-line password guessing. With ID files, very careful users can keep their information out of the enemy's hands, whereas with a directory service solution, the users have to trust someone else to protect the directory service replicas. Lotus Notes security does depend on the directory service for encrypting mail, finding cross certificates, and revoking certificates, but basic authentication and signature verification works even if the directory service is unavailable. 21.5.2 Coping with Export Controls It is amusing to see what lengths Lotus had to go to in order to satisfy export criteria, and yet provide reasonable security where legal. Each user has two RSA key pairs, a long one and a short one. Bulk encryption, as you'd expect, is done using secret key cryptography. Again, to satisfy the export rules at the time Lotus asked for a license, there were two key lengths for secret keys, a short one (on the order of 40 bits) and a long one (on the order of 64 bits). The long keys were used within the U.S. and Canada. The short ones were used for encryption when (at least) one of the participants was outside the U.S. and Canada. As export controls changed, successively larger keys were permitted, but the need for backward compatibility forced negotiation among a large number of different sizes. In R4, they introduced a particularly baroque mechanism for coping with export. At the time, the U.S. government was pushing Key Escrow (see §21.9 Clipper) as the solution that would allow people to be secure against anyone but them. But this idea was extremely unpopular. Lotus was 552 MORE SECURITY SYSTEMS 21.5.3 faced with the difficult decision of continuing with 40 bit keys (which people had publicly broken and were not considered trustworthy) or developing a Key Escrow system (which everyone hated). They came up with a novel compromise. They used long secret keys both inside and outside the U.S., but when one of the parties of a communication was outside the U.S., they encrypted all but 40 bits of the key using a public key provided to them by the NSA, and included that encrypted blob in the message header. That way, customers outside the U.S. got 40 bits of protection from the NSA (the maximum allowed at the time) and good protection from everyone else (including the people who were demonstrating the weakness of 40 bit keys). They announced this compromise with great fanfare and press releases, but the feature went largely unnoticed. Ironically, years later the feature led to a scandal when a Swedish newspaper "discovered" (in the documentation) that the NSA had a secret back door in the product, and criticized Lotus for cooperating with Big Brother. In R5, export controls were relaxed sufficiently that they could use the same encryption worldwide. But to this day, they have long and short RSA keys for interoperation with old versions. In Lotus Notes electronic mail, as in PGP, PEM, and S/MIME, public keys are used for authentication and for encryption of per-session secret keys that are used for message encryption. To satisfy export criteria, if either participant is outside the U.S., a short secret per-message key is used. And it is encrypted with the short public key. But the signature on the message is computed using the sender's long public key, whether or not the participants are within the U.S. This means that the signature on the message is secure even against an adversary capable of breaking the shorter RSA keys and secret keys used for encrypting the message. It isn't clear why the export control people insisted on requiring use of a short public key to encrypt a short secret key. If they can break short RSA keys, then they can extract the secret key, no matter how long, providing it is encrypted with a short RSA key. And if they can break short (40-bit) secret keys, they can decrypt any message encrypted with a short key even if that key is encrypted with a long RSA key. 21.5.3 Certificates for Hierarchical Names The Lotus Notes public key infrastructure is fairly straightforward and resembles the model described in §15.3.8 Bottom-Up with Name Constraints. A user is given a certificate by the CA responsible for that portion of the naming hierarchy. (Actually, the user has two certificates, one for a long key and one for a short key, but we'll ignore that in our discussion.) There's a chain of certificates from the root of the organization down to the user, and every user and server stores that chain in its ID file. The user presents that chain when authenticating (see §21.5.5 Lotus Notes Authentication). In order to authenticate Alice, Bob needs the chain of certificates from a common ancestor of Alice and Bob down to Alice. 21.5.4 LOTUS NOTES SECURITY 553 But there may not be a common ancestor. The name tree for Alice may be disjoint from the name tree for Bob. For instance, they might be in different companies, and there may be no common root level with a CA jointly managed by the two companies. In this case, the only way for Bob to authenticate Alice is through a cross certificate. Some ancestor of Bob has to have created a certificate for some ancestor of Alice. Cross certificates are stored in the directory service. When Bob wants to authenticate Alice, he searches the directory service for a cross certificate from one of his ancestors to one of Alice's ancestors. If no such cross certificate exists, then Bob and Alice cannot authenticate. In any case in which Bob needs a cross certificate for Alice, Alice will likewise need to find a cross certificate for Bob in her directory service. If Alice is a human at a workstation when Lotus Notes discovers the need for a cross certificate, it will prompt with the name and public key of the organization and invite Alice to create a personal cross certificate to that organization. In practice, most users create the cross certificate without checking the public key or even understanding what the message means. The resultant security is still better than nothing, because the cross certificate can only be used to authenticate entities in the particular remote organization and Alice will be warned if she ever connects to a server from that organization that presents a different public key. In other systems, like DASS, each CA in the naming hierarchy issues an up certificate for its parent CA, in addition to issuing down certificates for each of its child CAs. These certificates are stored in the corresponding directory. The DASS scheme is actually more convenient in the case where the key of a CA changes. With the DASS scheme, all that is necessary is for each child CA of the changed CA to reissue its up certificate, and the parent CA of the changed CA to issue a new down certificate. With the Lotus Notes scheme, if the key of a CA high up in the name space changes, a change has to be made to the ID file of each user. There is currently no automated method of doing this. On the other hand, the Lotus scheme has a lesser dependence on the directory service and can therefore be used when more network components are not functioning. 21.5.4 Certificates for Flat Names In versions 1 and 2 of Lotus Notes, and maintained since for backward compatibility, they had a system of Flat Names (people's names were just their names, and could not be assumed unique between organizations and perhaps not even within organizations). We describe them here because they represent another interesting way to construct a PKI. With hierarchical names, it's fairly easy to choose a chain of certificates that authenticates Alice to Bob, assuming the CA hierarchy follows the naming hierarchy. Flat names, by definition, don't have any structure. With flat names we could assume (like PGP) that applications will somehow find an appropriate chain of certificates. The flat-name version of Lotus Notes does not use chains of certificates. Instead, Alice is responsible for obtaining certificates for herself from different CAs. If she has enough certificates from enough different CAs, then when she attempts to communicate with Bob, it is likely she'll have a certificate 554 MORE SECURITY SYSTEMS 21.5.5 from a CA that Bob knows and trusts. Lotus certificates aren't X.509-encoded, but they contain basically the same information. In the flat name version, Alice starts out having one certificate, created by the administrator that created her account and ID file, and then she obtains other certificates as needed. In a small organization, there might only be a single CA. The non-hierarchical version of Lotus Notes assumes there may be numerous CAs, and each has been configured to know some of the other CAs' public keys. If Alice has a certificate from CA1, she can obtain a certificate from CA2 if CA2 has been configured with a public key for CA1 and if Bob, the human who is managing CA2, decides he wants to give Alice a certificate. The semi-automated method for Alice to get a certificate is to send a signed message to Bob asking him to give her a certificate. The signed message contains all of Alice's certificates. The mail program verifies Alice's signature and informs Bob of the list of CAs that have vouched for Alice's public key. Bob makes a policy decision based on Alice's name and the CA names. In particular, he must make sure that he has not already created a certificate for a different person whose name also happens to be Alice, and that no one is likely to confuse this Alice with some other one. If he thinks Alice's name is reasonable, and he trusts at least one of the CAs from which Alice has a certificate, he creates and mails back a certificate with Alice's name and public key signed by CA2. In this way Alice collects a bunch of certificates from some set of CAs. Lotus allows Alice to specify, for the CA corresponding to each of her certificates, whether Alice trusts that CA or not. If Alice marks a CA as being untrustworthy, then she will not necessarily believe a certificate signed by that CA. But even if she personally has nothing but contempt for some CA, say CAd, she might need a certificate from CAd in order to communicate with someone who does respect CAd. When two things, say Alice and Bob, authenticate, they each send the other a list of CAs they'll trust, and then each of them (hopefully) finds a certificate the other will believe and sends it to the other. For example, say Alice has certificates from CA1, CA2, and CA3, and marks CA2 and CA3 as trusted. Suppose Bob has certificates from CA1, CA2, and CA4, and marks CA1 and CA4 as trusted. Early in the authentication handshake, Alice sends the list of CAs she trusts, namely {CA2, CA3} to Bob, and Bob sends {CA1, CA4} to Alice. Alice has a certificate Bob will believe, namely the one signed by CA1, so she sends that one to Bob. Bob also happens to have a certificate Alice will believe, the one from CA2, and he sends that certificate to Alice. It might have been nice to allow Alice to maintain a list of CAs she trusts rather than having the trust information be a flag on certificates, since then Alice could accept certificates from a CA that hasn't granted Alice a certificate. 21.5.5 Lotus Notes Authentication The authentication handshake in Lotus Notes is similar to DASS, in that it allows caching of state from one connection to another. If both parties have cached the state, they can eliminate the step where they exchange and verify certificates. 21.5.5 LOTUS NOTES SECURITY 555 In the first two messages, Alice and Bob tell each other their public keys and give each other a challenge. In the next two messages, Alice and Bob send each other their certificate chains, starting from their organizational root (or, in the flat name case, there are an extra two messages where Alice and Bob exchange the names of CAs they trust). In messages 5 and 6, Bob sends Alice their long-term shared secret, encrypted with Alice's public key (to foil eavesdroppers) and signed with Bob's private key, to prove it's Bob sending the message (Alice takes Bob's word for it on their shared secret key). There's a clever performance reason why messages 5 and 6 are sent as two messages rather than having Bob send the signed encrypted long-term secret as one quantity. The reason is to allow the computation-intensive private key operations, the one where Alice decrypts the long-term secret and the one in which Bob signs the encrypted long-term secret, to proceed in parallel. Remember that RSA public key operations can be made faster than private key operations by using a small public exponent (§6.3.4.3 Having a Small Constant e). In message 7, Alice proves her identity. She needed to know her private key in order to extract the long-term secret, which she then uses to encrypt Bob's challenge. In message 8, Bob proves his identity and securely sends Alice a session key they will share for this one conversation. The reason Alice and Bob send the unsigned public keys in the beginning is for performance reasons. If the parties have, on a previous connection, exchanged and verified each other's certificates, then the step of exchanging and verifying the certificates can be skipped. In the case where they've maintained state, messages 3 through 6 are skipped. 1 Alice "Alice", Alice's public key PA, challenge RA Bob 2 "Bob", Bob's public key

PB, challenge RB 3 Alice’s certificate chain 4 Bob’s certificate chain 5 {long-term secret}Alice 6 Bob’s signature on previous message 7 long-term secret{RB} 8 long-term secret{RA, session key} 556 MORE SECURITY SYSTEMS 21.5.6 Note that some of these messages might be longer than a single packet. The authentication handshake is actually implemented on top of a reliable transport layer protocol, so it is possible for any of the messages in the handshake to consist of multiple physical packets. 21.5.6 The Authentication Long-Term Secret In a DASS authentication, both Alice and Bob may retain a cache, and if they remember information from a previous Alice-Bob exchange, they can save themselves some processing. The Lotus scheme as we described it in the previous section would seem to have the same property. However, the Lotus scheme is even more clever, since Bob does not need to keep any state! Note that in message 5, Bob chooses the long-term secret and sends it to Alice encrypted with Alice’s public key. The clever idea (devised by Al Eldridge, who designed Lotus Notes security) is that Bob does not choose a random number for the long-term secret, but rather computes it as a cryptographic hash of Alice’s name and a secret known only to Bob. If Alice authenticates a second time, Bob will choose the same long-term secret. If Alice has cached the long-term secret from a previous authentication, she can skip messages 3 through 6 and send message 7, which is Bob’s challenge (from message 2) encrypted with the cached long-term secret. Bob computes the long-term secret based only on Alice’s name and his own secret. Why is this secure? Bob doesn’t know what CA, if any, is vouching for Alice. But Bob does know that on some previous exchange he was impressed enough with Alice’s certificates that he was willing to tell her a long-term secret. Bob is allowed to change his own hashing secret and in fact does so periodically for security reasons. If Bob has changed his hashing secret since Alice’s previous authentication, then Alice will have the wrong long-term secret. In that case her authentication will fail and she will revert to the unabridged authentication handshake, and this time (assuming Bob still likes her certificates) she’ll get the new long-term secret. Bob only changes his secret about once a month, which means that a server only needs to do one private key operation per user per month to support authentication. This gives it a substantial performance advantage over SSL, which does RSA operations much more frequently. A minor industry has grown up around building RSA accelerators to support SSL servers, but none is necessary with the Lotus Notes design. 21.5.7 Mail The authentication handshake described in the previous section requires Alice and Bob to be actively communicating with each other. With electronic mail, however, Alice will compose a mes- 21.5.8 DCE SECURITY 557 sage to be transmitted to Bob, and Bob will eventually receive the message. He might be completely disconnected from the network at the time Alice composes and transmits the message, so Alice has to be able to accomplish message signing and message encryption without shaking hands with Bob. Lotus Notes mail allows you to encrypt, sign, both encrypt and sign, or do no cryptographic protection of a message. Alice includes her certificates in any message she signs. 21.5.8 Certification Revocation Since Alice sends Bob her certificates as part of authentication, certificate revocation is awkward in Lotus Notes security. There is no such thing as a certificate revocation list. Instead, a user’s certificate is revoked by removing it from the directory service. The theory is, if Bob is paranoid he can check the directory service to ensure Alice’s certificate is there when Alice logs in, or he can remember when he last looked up Alice’s certificate so that he doesn’t need to do it if he’s done it recently. If Bob is not paranoid, he does not need to check the directory service, increasing availability and performance at the expense of security. For Lotus servers, this is a configuration option. 21.6 DCE SECURITY I declare this thing open—whatever it is. —Prince Philip DCE stands for Distributed Computing Environment. It is an OSF (Open Software Foundation) product. OSF is a multivendor consortium. DCE security has been incorporated into a number of products from a variety of vendors. DCE security is conceptually similar to Kerberos, and indeed uses Kerberos V5 as one of its components. DCE has a modular design. Kerberos V5 is used for authentication although alternative mechanisms can in principle be substituted. Authentication, authorization, and encryption mechanisms are architecturally separate. Recall that Kerberos uses KDCs and Ticket Granting Servers, and in the Kerberos chapters we called both things the KDC, since they have to share the same database and same keys and therefore really are the same thing. DCE adds Privilege Servers and Registration Servers. In practice a Privilege Server and a Registration Server come packaged with a KDC. Privilege Servers and Registration Servers are on-line trusted entities, like KDCs. The purpose of a Privilege Server is to get the principal's UUID (universal unique ID), and the groups (see §15.8.3 Groups) to which that principal belongs, to be included in a Kerberos ticket in a secure way. The purpose of a Registration Server is to provide a combined database for a KDC and corresponding Privilege Server. 558 MORE SECURITY SYSTEMS 21.6 The KDC uses a master key for each principal, along with other stuff that's less interesting. The Privilege Server uses, for each principal, a UUID and the set of groups to which that principal belongs. ACLs (access control lists) exist outside the context of DCE and in many of the operating systems that support DCE. ACLs could list principals by name, but names are long and subject to change. So in most implementations, ACLs list UIDs (user IDs) and GIDs (group IDs) instead. In Kerberos the principal's name is in a ticket, and Kerberos gives no help in translating from name to UID or in looking up the set of GIDs associated with a principal. Furthermore, Kerberos does not standardize UIDs and GIDs. Different platforms have different forms of UID and GID. For example, most UNIX systems have 32-bit UIDs and GIDs. Some really old UNIX systems have 16-bit UIDs and GIDs. Kerberos assumes systems will translate names to UIDs in some platform-specific way. DCE standardizes group and user UUIDs at 128 bits long and standardizes DCE ACLs containing those UUIDs. Additionally, DCE does the translation from name to UUID, which is convenient for the applications but makes it difficult for them to use the native ACL format with DCE. The DCE designers, knowing that a UID field of 32 bits was really insufficient, hoped that they'd inspire the operating systems to migrate to their standard of 128 bits. In the meantime most DCE implementations use 32-bit UIDs and GIDs padded out with a constant to look like 128-bit UUIDs. A UUID in an ACL can belong to either a principal or a group. A user Alice will in general not need to know her own UUID, but usually is capable of remembering her own name, so the login exchange is based on her name. A user should be authorized to access something if the user's UUID is in the ACL or if a group to which the user belongs is listed in the ACL. DCE wanted an efficient method of knowing to which groups a user belongs, and this was done by including group information in the ticket. It is interesting how the DCE designers managed to get this information to be included in Kerberos tickets without modifying Kerberos. It would have been a small change to Kerberos to have the KDC maintain UUID and group membership with each principal, and put that information into the ticket. But instead, they designed elaborate mechanisms so they could work with an architecturally pure Kerberos. They still can't use unmodified KDCs, however, because they access the KDC using DCE RPC instead of UDP datagrams, and because management depends on being able to modify data through the Registration Server. Kerberos V5 has a field known as AUTHORIZATION DATA. Kerberos does not interpret the contents of this field. Rather it allows a user (say Alice), when requesting a TGT, to specify a value for that field. Kerberos just copies the requested value into the AUTHORIZATION DATA field of the TGT, and then copies that field into the AUTHORIZATION DATA field of any tickets issued based on that TGT. Alice's workstation requests TGTs and tickets on Alice's behalf, but does not maintain the database of Alice's UUID or group memberships. Even if it has that information, it can't be trusted to supply it. So Alice's Privilege Server supplies that information. The Privilege Server reads the database of user name/UUID/group membership from the Registration Server. 21.6 DCE SECURITY 559 With DCE, Alice logs in just like in Kerberos (see Figure 21-6). Alice's workstation gets an ordinary Kerberos TGT with Alice's name inside. The workstation then requests a ticket to the Privilege Server, which to the KDC is just another principal in the realm. The workstation then contacts the Privilege Server. The Privilege Server extracts Alice's name from the ticket, looks up Alice's UUID and group membership, and requests a TGT from the KDC with the Privilege Server's name in the CLIENT NAME field, and Alice's UUID and group membership in the AUTHORIZATION DATA. This TGT is referred to in DCE as a PTGT, for Privilege Ticket Granting Ticket. The Privilege Server then, in an encrypted message, gives Alice's workstation the PTGT and the corresponding session key (the one encrypted inside the PTGT). Alice's workstation uses the PTGT and corresponding key instead of the original TGT. Note that the above description is architectural—since the Privilege Server and KDC are packaged together, the messages between them might not actually occur. Now let's say that Alice asks to talk to server Bob. The workstation sends the PTGT to the KDC, along with the request for a ticket to Bob. The KDC will return to Alice a ticket to Bob with the AUTHORIZATION DATA field copied from the PTGT (so that it will contain Alice's UUID and group membership), and with the CLIENT NAME field equal to the Privilege Server's name. The ticket does not contain Alice's name, but it is not needed, since ACLs list Alice's UUID or use group UUIDs. Alice Alice needs a TGT KDC Privilege Server (PS) TGT need ticket to PS ticket to PS Alice needs a PTGT looks up Alice's need TGT, AUTHORIZATION DATA = UID and groups Alice's UID and groups TGT and session key forwards TGT and session key PTGT and session key Figure 21-6. Obtaining a PTGT in DCE 560 MORE SECURITY SYSTEMS 21.6 A DCE application, Bob, refuses to honor Kerberos tickets unless the Privilege Server's name is in the CLIENT NAME field. Once Bob checks to make sure the Privilege Server's name is in the ticket, Bob uses the AUTHORIZATION DATA field to discover Alice's UUID and group membership. Since the only way for Alice's workstation to obtain such a ticket (and corresponding session key) is through following the proper procedure, Bob can be assured that the UUID and group information in the ticket is correct. Bob checks this against the ACL of the resource Alice is asking to use. Multirealm DCE is similar to multirealm Kerberos. Suppose Alice is in realm A and Bob (the resource Alice would like to access) is in realm B. Furthermore, assume B is not a principal in A, so that it is necessary to go through transit realm C. (See Figure 21-7.) Alice PTGT; need ticket to Bob KDC A ticket to KDC C ticket to KDC C; need ticket to Privilege Server B KDC C ticket to KDC B ticket to KDC B; need ticket to Privilege Server B ticket to Privilege Server B ticket to Privilege Server B; need PTGT Privilege Server B KDC B need TGT TGT PTGT PTGT from Privilege Server B; need ticket to Bob ticket to Bob ticket to Bob (with CLIENT NAME = Privilege Server B; AUTHENTICATION DATA = Alice's UID, realm UID, groups) Bob Figure 21-7. Multirealm DCE 21.6 DCE SECURITY 561 Alice (her workstation) uses her PTGT to request from her KDC a ticket to Bob. Her KDC knows that Bob is not local, so instead gives her a ticket to the KDC in realm C. The AUTHORIZATION DATA and CLIENT NAME will get copied from the PTGT into the ticket to C, so the ticket to the KDC in realm C will still contain A's Privilege Server as CLIENT NAME, and Alice's UUID and groups in AUTHORIZATION DATA. Alice uses the ticket to the KDC in C to ask for a ticket to the Privilege Server in Bob's realm (B). C's KDC will return a ticket to the KDC in B. CLIENT NAME will still be A's Privilege Server. AUTHORIZATION DATA will still be Alice's UUID and groups. The TRANSITED field will contain C. Alice then uses the ticket to B's KDC to ask the KDC for a ticket to B's Privilege Server. The reason she needs to do this is to get a ticket with B's Privilege Server's name in the CLIENT NAME field, since Bob will not honor a ticket unless the CLIENT NAME is the name of the Privilege Server in Bob's realm. Alice then contacts B's Privilege Server. B's Privilege Server checks the ticket for legality. Since the ticket is coming from outside the realm, B's Privilege

Server will reject it if the CLIENT NAME is not the standard text string that denotes Privilege Server. B's Privilege Server also checks the TRANSITED field for acceptability. The recommended policy for an acceptable transit path is traversal of the tree implied by the hierarchical names, going up to the least common ancestor and then down, though with a single cross-link allowed on the path. With DCE security it is only Privilege Servers that check the TRANSITED field. In fact, once a Privilege Server returns a PTGT, the transit information is gone. DCE made the decision that it was more practical for the Privilege Server to check the transit information. Kerberos, remember, leaves it up to each application. Assuming B's Privilege Server accepts the transit information, it requests a ticket from B's KDC that contains Privilege Server B as CLIENT NAME, and Alice's UUID, group information, and realm UUID as AUTHORIZATION DATA. That's a PTGT just like the original PTGT, except now it has B's Privilege Server's name instead of A's. Alice uses that PTGT to request, from B's KDC, a ticket to Bob. Finally she gets a ticket to Bob, with CLIENT NAME Privilege Server B and AUTHORIZATION DATA equal to her UUID, group membership, and realm UUID. Because of the way that DCE did groups, a user can only be in groups maintained by the Privilege Server of the user's realm. It is possible to put groups and individuals from other realms into a DCE ACL, since each ACL entry effectively consists of a pair of UUIDs: the UUID of either the individual or the group, and the UUID of the realm in which the group or individual resides. Once Alice and Bob have mutually authenticated, DCE does not use the Kerberos integrity-protected or encrypted data exchanges. Instead, DCE has its own mechanisms for integrity protection, or integrity plus privacy protection within its RPC protocol, using the session key Kerberos does provide. As with Kerberos, the protocols are designed to allow multiple different cryptographic algorithms, but today DCE uses MD5 and DES. 562 MORE SECURITY SYSTEMS 21.7 21.7 MICROSOFT WINDOWS SECURITY 21.7.1 LAN Manager and NTLM LAN Manager comes equipped with a simple, straightforward cryptographic authentication protocol between client and server based on shared secrets. Each server that a user is entitled to access is configured with security information about that user, including a hash of the user's password. When a user Alice wishes to access a server, she types her name and password at her workstation, which contacts the server, sending Alice's name. The server sends a challenge, which the workstation encrypts using the hash of the user's password. Some applications (like RPC) continue cryptographic protection beyond the initial handshake using the user's hashed password to establish a session key. This protocol is very similar to NetWare V3. In both protocols, the fact that the server stores a hash of the user's password rather than the actual password does not theoretically make the scheme more secure. A modified version of the client software could impersonate the user if it directly used the hash of the password rather than hashing the string the user types. But it would be a lot more convenient and practical for an intruder Trudy if she could capture the actual password rather than the hash of the password, since then Trudy could type the password at unmodified client code. One difference between the protocols is that the LAN Manager scheme does not use salt. That means that if the user uses the same password on multiple servers, and an intruder captures one server's database, the intruder can impersonate the user at other servers. For NT, the scheme was extended in a way that was transparent to client machines. Transparent means that the new scheme works with the old client code and in fact a client machine cannot tell based on the protocol whether the server is using the old protocol or the new protocol. In the NT protocol, instead of keeping the security information at each server, security information is stored in a trusted on-line entity called a domain controller. The new scheme is called NTLM. It has the following advantages: • Management is simplified, since user security information only needs to be configured into a single location, the domain controller. • It is more user-friendly, since a user will have a single