Experiences with Host-to-Host IPsec

Tuomas Aura, Michael Roe, and Anish Mohammed

Microsoft Research

Abstract. This paper recounts some lessons that we learned from the deployment of host-to-host IPsec in a large corporate network. Several security issues arise from mismatches between the different identifier spaces used by applications, by the IPsec security policy database, and by the security infrastructure (X.509 certificates or Kerberos). Mobile hosts encounter additional problems because private IP addresses are not globally unique, and because they rely on an untrusted DNS server at the visited network. We also discuss a feature interaction in an enhanced IPsec firewall mechanism. The potential solutions are to relax the transparency of IPsec protection, to put applications directly in charge of their security and, in the long term, to redesign the security protocols not to use IP addresses as host identifiers.

1 Introduction

IPsec is a network-layer security protocol for the Internet that is intended to provide authentication and encryption of IP packets in a way that is transparent to applications. IPsec can be used between two hosts, between two security gateways, or between a host and a security gateway. IPsec was primarily specified with the security gateways and virtual private networks (VPN) in mind, but the expectation was that it could also be used end-to-end between two hosts.

This paper explains some of the difficulties that arise when IPsec is used in a host-to-host setting. The paper is based on security analysis and experiments that were done during the deployment of host-to-host IPsec on a large production network (tens of thousands of Windows hosts). We believe that the problems discovered are not unique to one network or one vendor's implementation, and that they explain why there are few examples of successful host-to-host IPsec deployments in large or medium-size networks.

The problems identified in this paper arise mainly from mismatches between the identifier spaces used in applications, in the IP layer, and in the security infrastructure. For example, IPsec security policies are typically defined in terms of IP addresses but the addresses mean little to the application and they do not appear in authentication credentials. Another reason for the problems is the fundamental design principle in IPsec that it should be a transparent layer that has no interaction with applications, apart from the configuration of a static security policy at the time of setting up the applications. We show that, in order for IPsec to meet the real application security requirements, the transparency needs to be relaxed and applications need to become security aware.

Most literature on security protocols concentrates on cryptographic algorithms and key-exchange protocols. We now know how to engineer a security protocol for authentication and encryption between abstract entities like Initiator and Respondent, or Alice and Bob. The latest IPsec specifications benefit from this work and represent the state of art in the field. The focus of this paper is on architectural issues, such as who defines the security policies and who has the authority over the various identifier spaces. We assume that the algorithms and protocols themselves are sound.

Arguments can be made that the vulnerabilities described in this paper are caused by flaws in the IPsec architecture. Equally well, it can be argued that we are using IPsec in the wrong way or that it has been implemented incorrectly. Either way, end-to-end encryption and authentication between hosts belonging to the same organization is clearly a reasonable security mechanism to ask for, and IPsec is a reasonable candidate to consider for the task. If IPsec in its currently implemented form fails, as we demonstrate it does, it then makes sense to ask what changes are needed to make the architecture meet our requirements.

The rest of the paper is structured as follows. We start with an overview of related work in section 2. Sections 3-4 provide an introduction to the IPsec architecture and to a well-known class of DNS-spoofing attacks. Section 5 shows how similar attacks are possible against host-to-host IPsec even if the name service is assumed to be secure. In section 6, we present a class of attacks that affects mobile hosts. Section 7 discusses an attack that was made possible by a non-standard extension to IPsec, and section 8 concludes the paper.

2 Related work

IPsec is defined by the Internet Engineering Task Force (IETF). The earlier IPsec architecture specification [12] was based on early implementation experiences. The latest revised version [13] has a well-defined security model. There are also two versions of the Internet key exchange protocol, IKEv1 [11] and IKEv2 [7]. Where it matters, we use the latest specifications. All observations and experiments, however, were conducted with implementations that follow the older specifications.

The research community has paid a lot of attention to the cryptography and the key-exchange protocol in IPsec [3, 9, 14, 20]. There is also some work on security-policy specification in large systems [10]. The architecture itself has received surprisingly little attention outside the IETF. The closest precedent to our work is by Ferguson and Schneier [9] who, in addition to evaluating the cryptography, make some radical recommendations for changes to the overall architecture, such as elimination of the transport mode. While the focus of this paper is on transport mode, all our observations apply equally well to tunnel mode when it is used host-to-host. Ferguson and Schneier also suggest that using IPsec for anything other than VPN will lead to problems but they do not elaborate on the topic. Meadows [17] pointed out that there might be problems with IKE if the authenticated identifiers are not based on IP addresses. A yet-

unpublished article by Trostle and Grossman [24] also discusses identifier-space issues in IPsec that are similar to the ones we raise in section 5.

Recently, much attention has been paid to "weak" security mechanisms, such as cookie exchanges, which reduce the number of potential attackers or make the attacks more expensive [2, 18, 23]. While these solutions are suitable for many applications and, in particular, for denial-of-service (DoS) prevention, they do not provide the kind of strong encryption and authentication that are the goal of IPsec. Thus, we have to assume a Dolev-Yao-type attacker [5] and cannot argue that a vulnerability in IPsec does not matter because it is unlikely that the attacker will be in the right place at the right time to exploit it. We do, however, compare the consequences of having a secure (e.g., DNSSec [6]) and an insecure name service. The conclusions are valid regardless of how the name-service security is implemented.

One high-level explanation for some of the problems discussed in this paper is that IP addresses are both host identifiers and location addresses [4]. There have been several attempts to separate these two functions, including the host identity protocol (HIP) [19], and Cryptographically Generated Addresses (CGA) [1]. In HIP, routable IP addresses are used only for routing and hosts are identified by the hash of a public key. Cryptographically generated addresses (CGA) [1], on the other hand, are routable IPv6 addresses that have the hash of a public key embedded in the address bits. This makes them work better as host identifiers. Mismatches between the identifiers used to specify the security policy and the identifiers provided by the authentication protocol have been studied in other protocol layers, for example middleware [15]. While we believe that many of these approaches have merit, we have chosen to work with standard DNS names and IP addresses in this paper.

3 How IPsec works

In this section, we give a simplified overview of the IPsec architecture with emphasis on the features that are needed in the rest of the paper. The architecture comprises protocols, security associations, and security policies.

IPsec, like most network security protocols, has a session protocol for the protection of data, and an authenticated key exchange protocol for establishing a shared session key. The session protocol is called the *Encapsulating Security Payload* (ESP). It takes care of the encryption and/or authentication of individual IP packets. There is another session protocol, the *Authentication Header* (AH), but its use is no longer recommended. The session keys are negotiated with the Internet Key Exchange (IKE), of which there are two versions (IKEv1 and IKEv2). The differences between the versions are largely unimportant to this paper.

The shared session state between two IPsec nodes is called a *security association* (SA). An SA determines the session protocol mode, the cryptographic algorithms, and the session keys used between the nodes. The SAs come in pairs,

one for each direction. Security associations are typically created by IKE, but they can also be configured manually by the system administrator.

In addition to the protocols and associations, an important part of the IPsec architecture is the security policy. Each host has a *security policy database* (SPD) that determines an *action* for each packet: whether the packet should be protected, discarded, or allowed to bypass IPsec processing. The SPD maps the protected packets to the right SAs, and triggers IKE to create an SA pair if no suitable one exists. The policy applies to both outbound and inbound packets. For outbound packets, an SA is used to add the encryption and message authentication code as required by the policy. For inbound packets, the policy determines what kind of protection the packet must have. Inbound packets that do not have the right protection, i.e., ones that were not received via the right SA, are discarded by IPsec.

The SPD is an ordered list of rules, each one of which consists of *selectors* and an action. The packet headers are compared against the selectors and the first rule with matching selectors determines the action to be taken on the packet. The exact packet-matching algorithm has changed over versions of the IPsec specification and varies from implementation to implementation; we stick to what is common between many implementations. The selectors are typically ranges of packet-header values, e.g., source and destination IP addresses and port numbers. For example, the SPD of figure 1 mandates ESP protection for communication with peers in the subnet 1.2.*.* and a BYPASS policy for other peers. In theory, the selectors are not limited to IP and transport-layer header fields but can take into account other context, such as the hostname, process and user at the source or destination. In practice, such selectors are rarely implemented because they can cause layer violations and are, therefore, hard to implement. (We will return to names as selectors in section 6.4.)

In this paper, we are interested in the difference between two types of IPsec applications. The first application is a VPN, in which encrypted and authenticated tunnels connect geographically separate parts of a private network. Originally, IPsec tunnels over the Internet were used to replace leased telephone lines and the goal was to provide security equivalent to a leased line. The tunnel in that case is an IPsec SA between two security gateways. Increasingly, VPN technology is used to connect individual remote hosts to private networks. In that case, an IPsec SA between a remote host and a security gateway replaces a dialup connection. In both situations, the SA is set up in tunnel mode. When establishing a tunnel-mode SA, authentication in IKE is typically based on a pre-shared long-term key or X.509 certificates [25] issued by an organizational certification authority.

The second IPsec application is host-to-host communication. In that case, the IPsec SAs are established between end hosts and encryption and authentication are performed by the end hosts themselves. This kind of SA is usually set up in transport mode, although tunnel mode can also be used. The modes have subtle differences in packet headers and in policy lookup, but we need not consider them here. The host-to-host communication differs from VPNs in that one uniform

policy cannot cover all traffic. Instead, hosts must support heterogeneous policies that allow them to talk simultaneously with peers that support IPsec and ones that do not. It may also be desirable to allow different authentication methods and certification authorities for different peers. The key exchange for transport-mode SAs is typically authenticated with certificates or Kerberos tickets [21], except for very small networks where pre-shared keys can be used. The lack of a global *public-key infrastructure* (PKI) for managing the certificates is often cited as the reason why host-to-host IPsec has not gained popularity in the Internet. However, we believe that there are other reasons (see section 5.1).

4 Classic DNS vulnerability

This section explains how IPsec interacts with name resolution, and how an attacker can exploit the insecurity of a name service to circumvent IPsec. While we assume that the name service is DNS, the same conclusions apply to other name resolution mechanism like NetBIOS.

4.1 IPsec and name resolution

When IPsec is used for host-to-host protection of application data, the application usually starts with a DNS name of the peer host to which it wants to connect. For example, the application may have obtained the host name from user input. The following actions then typically take place:

1. The application asks the operating system (OS) to resolve the host name (e.g., "server-b.example.org") into an IP address. The OS consults the name service and obtains an IP address (e.g., "1.2.3.4"), which it returns to the application.
2. The application asks the OS to open a TCP connection to the IP address. The TCP protocol layer initiates a TCP handshake by trying to send a SYN packet to the given peer IP address.
3. Before the packet is sent to the network, its header fields are compared against the SPD. If a matching policy is found and it requires the packet to be protected, IPsec checks whether a suitable SA already exists. If one does not exist, IKE is invoked to establish a pair of SA's with the peer. The sending of the TCP SYN packet is postponed until the key exchange completes.
4. Finally, when the SA exists, the TCP SYN packet is sent encrypted and/or authenticated with the session key. The following data packets are similarly protected.

4.2 DNS spoofing

In the well-known DNS spoofing attack [3], the attacker substitutes its own IP address (e.g., "3.4.5.6"), or someone else's address, for the DNS response. As

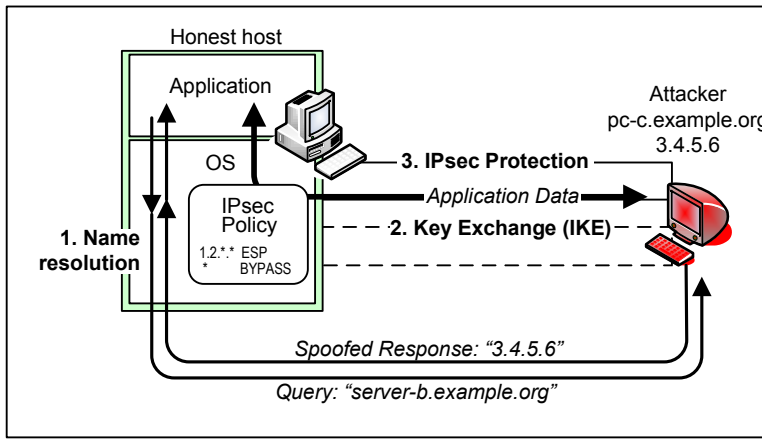

Fig. 1. DNS Spoofing Attack

a consequence, the application at the honest host connects to the wrong peer. This attack is shown in figure 1.

If the attacker is located at an IP address for which the honest host requires IPsec protection, then the attacker needs to authenticate itself to the honest host. This is usually called an *insider attack*. We will return to the insider attacks in section 5.1 where we show that they do not require DNS spoofing.

A more interesting DNS-spoofing scenario is one where the attacker is located at an IP address for which the honest host has a BYPASS policy (as in figure 1). In that case, the honest host will mistakenly skip authentication and send the packets unprotected. In practice, the honest host will always have a BYPASS policy for some peer addresses. First, there are hosts in almost every local network that do not support IPsec, such as network printers. The attacker may hijack the address of one such host. Second, most hosts will want to open unauthenticated connections to the public Internet. Thus, all IP addresses that are outside the honest host's own domain usually have a BYPASS policy.

Problem 1a: It is possible to bypass an IPsec policy with DNS spoofing.

Middle boxes, e.g., NATs, complicate the scenario but do not necessarily prevent the attacks. As long as the attacker manages to spoof the DNS response or is able to manipulate the DNS servers, the attack will succeed equally well from outside the NAT.

In the DNS-spoofing attack of figure 1, the wrong policy is applied to the packets because the attacker controls the binding of DNS names, which are the identifiers used by the application, to IP addresses, which are the identifiers used in the security policy. It is therefore tempting to blame the insecure name resolution mechanism for the attacks. This logically leads to the idea of introducing

a secure name service. In the following section, however, we will argue that the current IPsec architecture makes the security of the name service irrelevant.

5 Identifier mismatch

In this section, we assume that DNS spoofing is not possible. This allows us to focus on other design issues in IPsec. Indeed, we discover in section 5.1 that, regardless of whether the name service is secure or not, there still is a disconnection between host names and IP addresses in IPsec. In sections 5.2-5.5, we discuss potential solutions.

5.1 Attack without DNS spoofing

Recall that the security association between the honest host and its peer is created by the authenticated key exchange, IKE. A critical part of the key exchange is a step in which the peer host sends to the honest host a credential that proves its identity. The credential is usually an X.509 public-key certificate issued by a certification authority (CA) [25]. Equivalently, the credential could be a Kerberos ticket (in IKEv1 with GSS-API authentication [16, 21]), in which case it is issued by a Kerberos KDC. We will use the word certificate below but the discussion applies to the other credential types as well.

The purpose of the certificate in IKE is to bind together the peer host's identity and its public signature verification key, which is then used for cryptographic authentication. The identifier in the certificate is typically a host name. While certificates can, in theory, contain IP addresses, they rarely do. Thus, while the SPD is typically specified in terms of IP addresses, the certificate that is used to authenticate the peer host only contains the peer's host name.

In figure 2, we trace the data flow when an application at an honest host tries to connect to a peer. The application starts with a host name ("server-b.example.org"). It uses the name service to resolve the host name into an IP address ("1.2.3.4"). This time, the answer is correct because we assume no DNS spoofing. The application then asks the OS to connect to the IP address, which triggers IKE. Within the IKE protocol, the honest host receives the peer's certificate. After the necessary cryptographic computations, it would be crucial for the honest host to compare the identifier in the certificate against the identifier that the application started with. But the only identifier in the certificate is a host name ("pc-c.example.org") and the only identifier known to the TCP/IP stack at this point is the peer IP address. Thus, the stack is unable to authenticate the peer, or to detect the use of a wrong certificate.

The next question is, of course, what do real IPsec implementations do when they encounter this impossible decision. One answer is that IPsec is not commonly deployed in host-to-host scenarios. It is used between VPN gateways that have static IP addresses and pre-shared keys, so that certificates are unnecessary. IPsec is also used in host-to-gateway VPNs, where the necessary checks

can be made in a non-standard way by special VPN management software. (A mechanism for these checks is being standardized. See section 5.3.)

But there is an increasing demand to secure private networks from intruders with host-to-host IPsec. The only solution that we have seen in actual IPsec deployments is to skip the impossible check between the host name in the certificate and the IP address in the SPD and to continue as if everything were ok! (Appendix A contains instructions for a simple experiment to test this.)

The consequence of the failure to check the name in the certificate is to enable insider attacks. An attacker only needs to have a certificate issued by a CA that the honest host trusts. This means that, in an intranet, such as the large network we observed, any authorized host can impersonate all other authorized hosts. On the other hand, an intruder that plugs into the network or a host that is on a revocation or quarantine list cannot. Spoofed packets that manage to enter the intranet, e.g., via an unauthorized ingress route are also detected and discarded. In the network that we observed, these were the main security goals and spoofing attacks by authorized hosts were a secondary concern.

Another way to explain the situation is to say that host-to-host IPsec provides group authentication but not authentication of individual hosts. This is sometimes acceptable but not always. It is of paramount importance that those who rely on the authentication understand the limitations of the technology. Unfortunately, one is easily misled by IPsec literature on this point.

Problem 1b: Host-to-host IPsec currently provides only group authentication. Hosts certified by the same CA can impersonate each other.

At present, host-to-host IPsec is not commonly used between Internet hosts that belong to different organizations. The reason for this is usually thought to be the lack of a global PKI. But even if a global PKI existed, host-to-host IPsec with certificate authentication on the Internet would be insecure because anyone with a certificate from the global PKI would be able to impersonate everyone else. The above facts are important to keep in mind because, with IKEv2, there will be the possibility of using TLS certificates for server authentication and passwords for client authentication (IKEv2 with EAP-PEAP authentication). While this could partially solve the PKI availability issue, it would not prevent the certified servers from mounting impersonation attacks against each other.

5.2 Secure name service

The honest host in figure 2 is faced with comparing a name ("PC-C") and an IP address ("1.2.3.4"). One's first thought, especially if one happens to be staring at that part of the code and not thinking of the high-level architecture, is to use secure DNS for this purpose. There are practical problems, however, in addition to the fact that secure DNS is not widely deployed.

If secure DNS is used to prevent the attacks, then we need both a secure forward lookup (find the address based on a name) and a secure reverse lookup (find the name based on an address). The forward lookup is required in the first

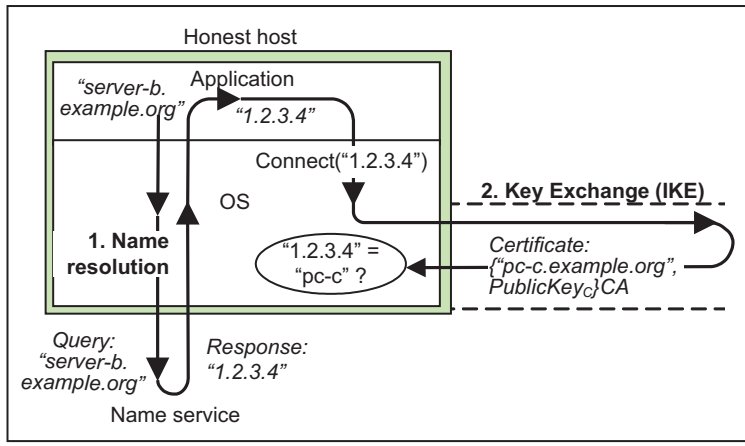

Fig. 2. Interaction of DNS and IKE

phase where the application resolves the original name into an IP address. The reverse lookup is needed during the key exchange for comparing the IP address and the name in the certificate. Note that the OS was asked to connect to a specific IP address and, therefore, has to map from the address to the name and not the other way. Otherwise, the attacker could easily set its own DNS records to map its host name ("pc-c.example.org") to any address, including the authentic peer address ("1.2.3.4"). In other words, the secure forward lookup verifies that the owner of the name wants to associate it with a specific IP address, and the secure reverse lookup verifies that the owner of the IP address wants to associate it with the DNS name. For this reason, the reverse DNS records form a hierarchy that is isomorphic with the way IP addresses are allocated.

There are several practical issues with the secure reverse DNS lookup. Firstly, secure DNS is not any more widely available for reverse lookups than for forward lookups. Secondly, many hosts have dynamic addresses (DHCP or IPv6 autoconfiguration) and reverse DNS records are generally not updated for them. While reverse DNS is supposed to work for all Internet hosts, the response is often a generic name assigned by an ISP rather than the name actually used by the host. Thirdly, if an IP address has multiple names associated with it, the reverse lookup usually returns only one of them, and not necessarily the one used in the certificate. Hosts behind a NAT have the same problem because they share a single IP address, which means that there is no one unique owner for the address. Yet, they may still use IPsec with NAT traversal support. Fourthly, there are many ways in which the application may learn the IP address corresponding to the peer host name. IPsec does not require the use of one specific technology, such as DNS. In many cases, the IP address is obtained by an application-specific mechanism (e.g., in a SIP INVITE message) or from a proprietary name resolu-

tion mechanism (e.g., NetBIOS). It is therefore difficult to ensure comprehensive protection for name resolution.

These obstacles can sometimes be overcome within an intranet. For example, some DHCP servers automatically update the reverse DNS records when address allocation changes and IPsec can be used to protect the DNS queries and responses with an authoritative server. Nevertheless, it is clear that reverse DNS is not a generic solution to the identifier mismatch problem.

5.3 Peer authorization database

The latest version of the IPsec architecture specification [13] proposes the addition of a *peer authorization database* (PAD) to each IPsec host. The purpose of this database is to indicate the range of SPD identities (IP addresses) that each authenticated identity (name in the certificate) is authorized to use. The semantics of such a database is essentially the same as what we wanted to achieve with reverse DNS above: it determines which authenticated names may be associated with an IP address. The difference is that the PAD is implemented as a local database rather than as a lookup from a server.

For a VPN, the PAD solves the reverse mapping problem elegantly. For gateway-to-gateway VPN, it removes the need to distribute pre-shared keys. Instead, the gateways can be configured with a static table of peer gateway names and their IP addresses, and certificates can be used for authentication in IKE. In host-to-gateway VPN, the gateway typically allocates an intranet IP address for the remote host. IKEv2 even provides a standard mechanism to do so. In that case, the ownership of the IP address is immediately known to the gateway and the mapping can be added into the PAD. In both VPN scenarios, the PAD is a standard way to implement the checks that well-designed VPN software previously had to do in some ad-hoc manner. For host-to-host IPsec, however, the PAD is not so much a solution as a rephrasing of the problem. The question becomes how to implement the PAD, which is an equivalent task to that of implementing the secure reverse DNS.

5.4 Giving up statelessness

It is possible that all the attention given to secure DNS technology has lead us to look for the solution in the wrong place. The security property really required by the application is not a correspondence between a host name and an IP address. What the application cares about is that the host name in the certificate ("pc-c.example.org") is the same as the host name that it started with ("server-b.example.org"). If this can be ensured, the application does not care what the IP address is.

The problem is twofold. First, by the time the application calls *connect*, it simply passes to the OS an IP address without any indication of how the address was obtained. Second, the IPsec policy is stated in terms of IP addresses.

The first problem stems from the fact that the IP layer has been designed to be as stateless as possible. That is, the IP layer is not aware of higher-layer

connections or sessions and does not maintain a state for them. This design principle has usually served protocol designers well. In this case, however, it would be beneficial for the OS to remember that it has previously performed the name resolution and to check that the name in the certificate is the one that it previously resolved to the same IP address. Care needs to be taken, however, when implementing this check. For example, it is not sufficient to look up the certified name in the local DNS cache. This is because the DNS cache stores the forward lookup result and, as we explained in section 5.2, a reverse lookup is needed.

In order to create a strong link between the resolved name that the application passed to the OS and the IP address that is compared against the name in the certificate, we can use the following trick: Instead of returning the IP address to the application, the name resolution returns a 32-bit or 128-bit handle, which is an index to a table that stores the actual IP address. When the application calls connect, the OS looks up the name and IP address from the table. During peer authentication, the name from the table is compared against the name in the certificate. This ensures that the peer is authenticated with the exact same name that the application started with. The idea comes from the HIP local-scope identifiers [19]. The disadvantage of this solution is that applications that do not treat IP addresses as opaque numbers will break. The same is true for applications that connect to an IP address obtained by some other means than a DNS lookup.

The second problem listed above is that the IPsec security policy is typically stated in terms of IP addresses. This means that authentication requirements can be different for different peer IP addresses. An attacker can exploit this by hijacking an address that corresponds to a BYPASS policy. As we explained in section 4.2, host-to-host IPsec policies always allow some peers, such as printers or non-intranet hosts, to bypass the protection. Thus, even if the authentication is always done with the right identifier, it does not prevent the attacker from turning authentication off, and IP spoofing allows them to do exactly that.

5.5 Giving up transparency

IPsec is designed to be transparent to the higher protocol layers. The security policy is specified by an administrator and enforced by the IP layer without any involvement by the applications. Yet, the purpose of IPsec is to protect application data. This conflict may be the best explanation for the identifier mismatch problem. The application cannot pass to the OS its security requirements such as the CA and peer name to be used in the authentication. Neither can it check these parameters after the handshake. In many IPsec implementations, an application can call *setsockopt* and *getsockopt* to switch on encryption or authentication and to query their state but it has no access to the certificate.

The solution we suggest is to compromise the transparency principle and to provide the application with the means to check what kind of protection is provided for its data. In the specific problem that we have discussed here, it is sufficient to provide an API for checking whether packets sent and received via

a given socket are authenticated and/or encrypted, and to check the identifier that was authenticated in the key exchange. A more radical change to the IPsec architecture would be to replace the SPD with a per-socket policy. The system administrator may, however, still want to enforce some policies independent of the applications. This leads to problems of policy reconciliation, which is beyond the scope of this paper.

Solution 1: Allow applications to specify the IPsec policy for their data and to check that the right kind of security is in place, including the authenticated peer identifier and the issuer of the peer credentials.

6 Attacks against Mobile Hosts

Next, we will consider what effect host mobility, such as a laptop being moved to a different local network, has on IPsec. We first highlight, in section 6.1, reasons why DNS spoofing is much more serious a threat than is usually thought. In section 6.2, we show that confusion between different private address spaces can lead to attacks even without DNS spoofing.

6.1 Trusting Local Name Servers

It is quite clear from section 4.2 that a secure name service is required before IPsec can provide strong host-to-host security. Nevertheless, it is tempting to ignore the problems and to assume that DNS spoofing is sufficiently cumbersome to deter most attackers. For example, it is difficult for an attacker on the Internet to interfere with name resolution in the intranet. The goal of this section is to present a common scenario where the spoofing is trivial to implement. We assume, for a moment, that secure DNS is not in use.

When a mobile host, such as a laptop computer, connects to the Internet at a new access network, it usually configures its network connection based on information provided by a local DHCP server at that network. One of the configuration parameters provided by DHCP is the IP address of a local DNS server. Thus, a mobile host depends on the local DNS server to define the binding between host names and IP addresses. As a consequence, malicious administrators at the access network can provide false DNS responses and, therefore, circumvent the mobile's IPsec policy. Furthermore, all they need to do this is standard DNS server software.

Figure 3 shows an example of this attack. The mobile host unwisely trusts the DNS server at the access network to map a peer name ("server-b.example.org") into an IP address. By selecting an address for which the host has a BYPASS policy, the DNS server can both turn off IPsec protection and redirect the connection to the wrong destination. Since IPsec is usually bypassed for the public Internet, the attacker may be able to use its real IP address.

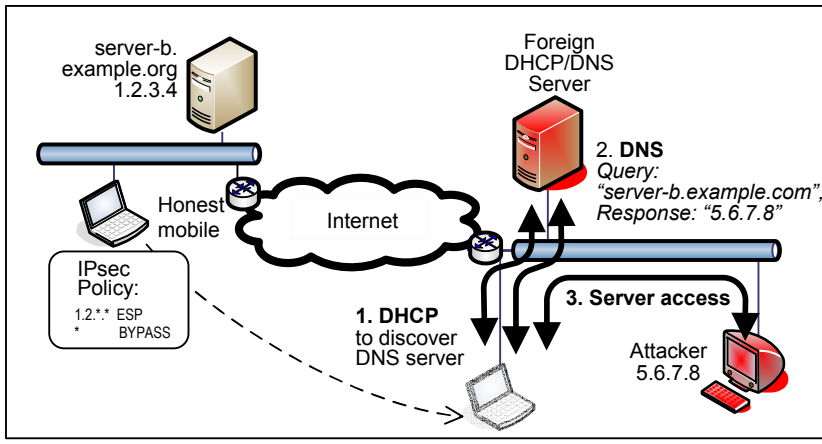

Fig. 3. Mobile host using a foreign DNS server

Problem 2: The mobile host relies on a name server at an untrusted access network to define the mapping from names to IP addresses. Therefore, the access network can circumvent the mobile host's IPsec policy.

Another way to express the problem is to say that the physical location of the mobile host determines the authority in whose name space the DNS name resolution happens. However, the problem is not completely solved by the mobile connecting to an authoritative DNS server at its home network. This is because there are commonly available tools for intercepting the requests transparently and for redirecting them to a local server at the access network.

Secure DNS with signed resource records would prevent the attack as it is based on signature chains and provides the same level of security regardless of which DNS server a host uses. A less secure solution is to protect access to an authoritative, or just friendly, DNS server with a DNS transaction security mechanism or with IPsec.

Solution 2: Never rely on insecure DNS at an untrusted access network. If secure DNS is not available, use IPsec to protect access to a more trustworthy name server.

The following section will, however, show that some attacks against the mobile host do not depend on DNS spoofing.

6.2 Ambiguity of Private Address Spaces

Some IP address ranges have been allocated for private use [22]. Anyone can use these addresses in their private network but packets with a private source or destination address must never be routed to the Internet. Examples of this

kind of addresses are the ranges 10.*.*.* and 192.168.*.* in IPv4 and site-local addresses in IPv6. The most common use for private addresses is behind a NAT.

It is usually thought that private addresses provide a level of security because packets containing these addresses cannot cross network boundaries. If a host has only an address allocated from a private address space, then it will be impossible to route any packets from the Internet to the host, or from the host to the Internet, except when the communication is specifically enabled by a NAT. For this reason, it is common to place critical intranet resources, such as important file servers, into subnets with private addresses. It is also tempting to give this kind of protection to other hosts that should not be accessible from the Internet, e.g., printers.

The problem with the private addresses is that mobile hosts can move between two networks that both use the same private address range. The traditional wisdom that packets with private addresses cannot cross network boundaries is no longer true because the sender (or receiver) itself crosses the boundary.

For example, the mobile host in figure 4 uses an authoritative name server at its home network in order to avoid the attack of section 6.1. The name server maps a name ("printer-b.example.org") to a private address ("192.168.0.10"). But when the mobile host connects to this address, the packets are routed to an untrusted host at the access network. Even worse, the mobile host has a BYPASS policy for the address. Connecting to the wrong printer or server can cause the mobile host to reveal confidential information to the attacker. Sometimes, it can compromise the mobile host's integrity if the server, for example, distributes configuration information.

Problem 3: When a mobile host moves between physical networks, it can become confused about which private address space it is currently at. As a consequence, it may apply the wrong IPsec policy to packets destined to a private IP address.

This kind of situation can arise, for example, when most of the mobile host's communication, including DNS queries, is routed to its home network via a VPN connection but some addresses are excluded from the tunneling to allow the mobile to access resources (e.g., printers) local to the access network. Another situation that occurs commonly is one where the private address is statically configured (in a *hosts* file) or it is in the mobile's DNS cache at the time it attaches to the untrusted access network.

The problem is similar to multihomed hosts and scoped IPv6 addresses. Multihomed hosts, however, have the benefit of being able to differentiate addresses by the interface on which they appear. Given that the scoping of IPv4 addresses has never been completely solved for multihomed hosts, we do not expect there to be any magic bullet for the mobile-host scenario either.

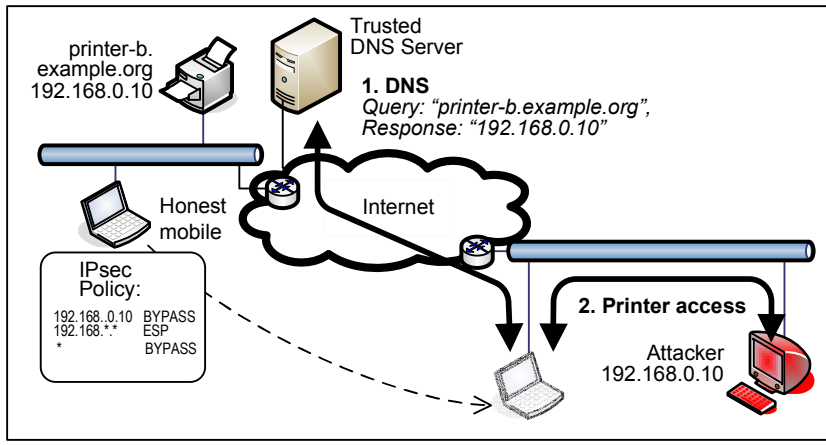

Fig. 4. Mobile host moving between private address spaces

Solution 3: When configuring the IPsec policy in a mobile host, one should not assume any degree of protection from physical or logical network boundaries. In particular, BYPASS policies for private and site-local addresses are dangerous.

6.3 Communication failures in private address spaces

There is another consequence of the confusion between private address spaces. If a host requires protection for a private peer address, communication with that address will fail unless the peer is able to authenticate itself. For example, the policy in figure 4 prevents the mobile host from communicating with any hosts in the access network that it is visiting (except with the attacker).

Problem 4: A mobile host that requires protection for private or site-local peer addresses cannot communicate with hosts that use the same addresses in other administrative domains.

Stationary hosts do not experience the problem because they always reside in one administrative domain. Mobile hosts, on the other hand, move between networks. Configuring IPsec protection for a private address range in a mobile host effectively monopolizes that address range for use in only one domain.

We will not offer any immediate solution to this problem. In the light of the previous section, we feel that the communication failure is preferable to a security failure. It seems that private address spaces are fundamentally irreconcilable with an authentication mechanism that requires unique identifiers. System administrators have tried to solve the problem by guessing which sub-ranges of the private address space are not used by anyone else, a solution that is hardly sustainable if everyone adopts it.

6.4 Names in security policies

It is clear from the above that IP addresses are not the ideal identifiers for specifying a security policy for host-to-host communication, and we might want to replace them with host names or some other unique identifiers. This approach works in principle but is has a major limitation that we will explain below.

The IPsec specification allows policy selectors in the SPD to be specified in terms of host and user names but the semantics is not quite what one might, naively, expect it to be. The problem is that it is not possible to use the peer host name or user name as a selector in a BYPASS policy. While other types of selectors are matched directly against header fields of an inbound or outbound packet, the name selectors must be matched against the authenticated peer name from the peer's credential. Thus, any policy specified in terms of such selectors requires authentication of the peer. This means that name selectors can prevent insider attacks where there is confusion between two authenticated identities, but they cannot prevent the attacker from exploiting a BYPASS policy to get around IPsec. In this respect, the name selectors are similar to the peer authorization database. The following solution comes, therefore, with a major caveat:

> **Solution 4:** Redesign IPsec so that IP addresses are not used as selectors. However, this only helps if all peers are authenticated.

It is generally the case for all network security mechanism that, if some peers are allowed to bypass authentication, then the attacker can pretend to be one of them and, thus, bypass the security. The damage can be limited, as network administrators know, by making BYPASS policies as specific as possible, for example, by restricting them to specific port numbers.

Again, it appears that the ultimate solution is the one we already stated in section 5.5: putting the applications in charge of their own security. When authentication is provided by a transparent layer, the applications cannot know whether a specific packet has been authenticated or bypassed protection. On the other hand, if the applications are allowed to specify and enforce their own security policies, they can decide which data needs authentication and which does not.

7 IPsec as a firewall

In this section, we will discuss an unexpected feature interaction in IPsec when it is used as a firewall. Although the attacks depend on a non-standard IPsec feature, they are interesting because they are an example of two security mechanisms, each of which is sensible and useful on its own, that together result in security failure.

The DISCARD policies in IPsec can be used to implement an elementary firewall policy. That is, IPsec can filter packets by any criteria that are expressible with the selectors in the SPD. Typically this means blocking port numbers or ranges of IP addresses. With standard IPsec, the firewall is necessarily stateless:

each packet is processed individually and without regard to any packets sent or received previously. Nobody would expect to be able to implement complex application-layer policies with IPsec but it seems a pity that one cannot express the kind of simple stateful policies that even end users can configure in personal firewalls and home NATs. In particular, one would want to differentiate between packets that belong to a connection initiated by the local host and packets that arrive unsolicited.

Some IPsec implementations provide this kind of functionality, even though it is not a part of the standard IPsec architecture. One way to implement this is a policy that, upon sending an outbound packet to an IP address, creates a state ("soft SA") to allow further packets from that peer to bypass IPsec. This kind of policy is used, for example, to enable access to printers and embedded devices that do not support IPsec.

Another, seemingly unrelated, issue with the current IPsec specifications and implementations is that they lack full support for broadcast and multicast communication. There are two ways a system administrator can deal with the unsupported traffic: it can be discarded or allowed to bypass IPsec. In any system that depends on broadcast or multicast protocols for its operation, the choice is obviously to configure a BYPASS policy for these types of traffic. Encryption is usually unnecessary because these types of communication are accessible to large groups of hosts anyway. The lack of authentication is a bigger issue but it can be addressed with application-specific security mechanisms where necessary.

Unfortunately, the two features described above combine to create a vulnerability, as shown in figure 5. The honest host in the figure has been configured with a stateful policy because it needs to access printers that do not support IPsec and, in a large network, managing the BYPASS policies individually would be too tedious. In order to circumvent the stateful policy, the attacker needs to get the honest host to send an outbound unicast packet. If the honest host required authentication for all inbound communication, the attacker would not be able to manipulate it into sending such a packet. But when the policy at the honest host allows inbound broadcast and multicast packets to bypass IPsec, the attacker can use these to trigger the outbound packet. Indeed, there are several broadcast-based protocols that can be exploited for this. For example, the BOOTP, DHCP, and NetBIOS protocols send unicast responses to broadcast requests. In general, it is extremely difficult to determine at the IP layer which end host is the real initiator of the communication. Therefore, security policies that differentiate between the initiator and respondent tend to be fragile.

Problem 5a: Stateful firewall extensions to IPsec can be exploited by triggering outbound packets with inbound traffic for which there is a BYPASS policy, such as broadcast and multicast packets.

The difference between IPsec and dedicated firewall software is that the typical configuration for IPsec allows inbound packets to bypass protection if they belong to a traffic type for which there is no suitable protection mechanism. Otherwise, one would have to delay host-to-host IPsec deployment until the

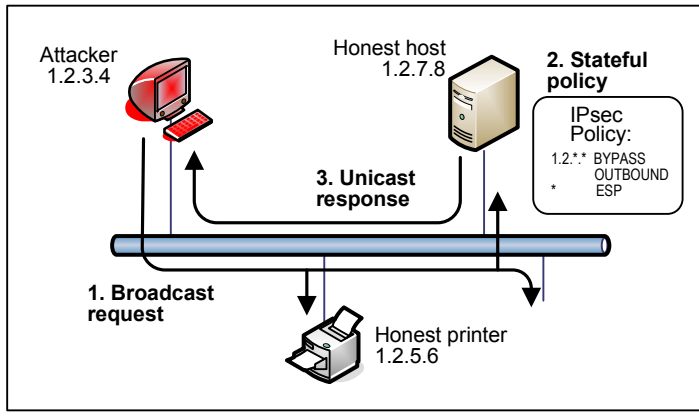

Fig. 5. Circumventing a stateful policy with broadcast

technology is in place to protect all possible types of communication. Firewalls, on the other hand, tend to discard packets by default. Thus, the above problem is, in part, explained by the different motivation for the administrator in configuring IPsec and firewall policies.

It seems that there will always be some IP packets for which no protection is available and which the honest host nevertheless wants to receive. For example, IPv6 is increasingly enabled on Internet hosts for experimentation, yet many IPv6 implementations still lack IPsec functionality or administrative support. This means that IPsec policies for IPv6 are often more relaxed than for IPv4. There is clearly a trade-off between security and availability of new services.

Problem 5b: New protocols often lack full IPsec support. The attacker can often use them to circumvent IPsec protection of existing protocols.

Solution 5: Care must be taken in reconciling strong security policies and availability of network communications during the deployment of new protocols.

8 Conclusion

This paper documents several vulnerabilities in host-to-host IPsec. There is no doubt that, over the years, administrators and developers have encountered all of these issues and either found solutions similar to the ones proposed here or moved to using other security protocols like TLS. Yet, it is a widely held belief that, if one implements and configures IPsec correctly, it can meet all of one's network security requirements. We aim to show that this is only possible with major changes to the philosophy that underlies IPsec. In particular, security

should not be implemented as a transparent layer. Instead, applications should be allowed to manage their own security policies. Also, IP addresses are not meaningful identifiers for specifying security policies for application data.

We have observed all of the vulnerabilities described in this paper in actual IPsec implementations or configurations. One could claim that little is achieved by deploying host-to-host IPsec in its current form. There are, however, some mitigating factors. Firstly, some of the weaknesses result from compromises made for deployment reasons. For example, the need to have exemptions for multicast traffic and for non-IPsec-capable hosts may slowly disappear, perhaps with the transition to IPv6. Secondly, we found that attacks tested in laboratory conditions rarely work on production networks because modern applications and middleware (e.g., RPC mechanisms) implement their own security. Indeed, they have been designed with the assumption that the network layer is insecure. This leads us to the question whether there remains any role for host-to-host IPsec, but that discussion is beyond the scope this paper.

A A Simple Experiment

The following simple experiment demonstrates that the OS is unable to check the correctness of the name in the certificate (as explained in section 5.1): First, set up the IPsec policy to use a transport mode SA and certificate authentication. Then, act yourself as the application of figure 2. In a shell window, use *nslookup* or a similar utility to resolve the peer host's IP address. Finally, invoke *telnet* to connect to the IP address. By the time you enter the *telnet* command, the only place where the host name is remembered is your head. Surely, the OS cannot have compared the host name in your mind with the one in the certificate.

References

1. Tuomas Aura. Cryptographically generated addresses. In *Proc. 6th Information Security Conference (ISC 2003)*, volume 2851 of *LNCS*, pages 29–43. Springer, 2003.
2. Tuomas Aura, Michael Roe, and Jari Arkko. Security of Internet location management. In *Proc. 18th Annual Computer Security Applications Conference*, Las Vegas, December 2002. IEEE Computer Society.
3. Steven M. Bellovin. Problem areas for the IP security protocols. In *Proc. 6th Usenix Unix Security Symposium*, pages 205–214, San Jose, CA USA, July 1996. USENIX Association.
4. B. Carpenter, J. Crowcroft, and Y. Rekhter. IPv4 address behaviour today. RFC 2101, IETF Network Working Group, February 1997.
5. D. Dolev and A. Yao. On the security of public-key protocols. *Communications of the ACM*, 29(8):198–208, August 1983.
6. Donald Eastlake. Domain name system security extensions. RFC 2535, IETF Network Working Group, March 1999.
7. Charlie Kaufman (Ed.). Internet key exchange (IKEv2) protocol. Internet-Draft draft-ietf-ipsec-ikev2-17.txt, IETF IPsec Working Group, September 2004. Work in progress.

8. Carl Ellison, Bill Franz, Butler Lampson, Ron Rivest, Brian M. Thomas, and Tatu Ylönen. SPKI certificate theory. RFC 2693, IETF Network Working Group, September 1999.
9. Niels Ferguson and Bruce Schneier. A cryptographic evaluation of IPsec. Technical report, Counterpane Labs, 1999.
10. Joshua D. Guttman, Amy L. Herzog, and F. Javier Thayer. Authentication and confidentiality via IPsec. In *Computer Security - ESORICS 2000*, volume 1895 of *LNCS*, pages 255–272, Toulouse, France, October 2000. Springer.
11. Dan Harkins and Dave Carrel. The Internet key exchange (IKE). RFC 2409, IETF Network Working Group, November 1998.
12. Stephen Kent and Randall Atkinson. Security architecture for the Internet Protocol. RFC 2401, IETF Network Working Group, November 1998.
13. Stephen Kent and Karen Seo. Security architecture for the Internet protocol. Internet-Draft draft-ietf-ipsec-rfc2401bis-03, IETF IPsec Working Group, September 2004. Work in progress.
14. Hugo Krawczyk. SIGMA: The 'SIGn-and-MAc' approach to authenticated Diffie-Hellman and its use in the IKE-protocols. In *Advances in Cryptology: Proc. CRYPTO 2003*, volume 2729 of *LNCS*, pages 400–425, Santa Barbara, CA USA, August 2003. Springer.
15. Ulrich Lang, Dieter Gollmann, and Rudolf Schreiner. Verifiable identifiers in middleware security. In *Proc. 17th Annual Computer Security Applications Conference*, pages 450–459, New Orleans, LA USA, December 2001. IEEE Computer Society.
16. John Linn. Generic security service application program interface version 2, update 1. RFC 2743, IETF, January 2000.
17. Catherine Meadows. Analysis of the Internet Key Exchange protocol using the NRL protocol analyzer. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 1999.
18. Pekka Nikander. Denial-of-service, address ownership, and early authentication in the IPv6 world. In *Proc. 9th International Workshop on Security Protocols*, volume 2467 of *LNCS*, pages 12–21, Cambridge, UK, April 2001. Springer 2002.
19. Pekka Nikander, Jukka Ylitalo, and Jorma Wall. Integrating security, mobility, and multi-homing in a HIP way. In *Proc. Network and Distributed Systems Security Symposium (NDSS'03)*, pages 87–99, San Diego, CA USA, February 2003.
20. Radia Perlman and Charlie Kaufman. Key exchange in IPSec: Analysis of IKE. *IEEE Internet Computing*, 4(6):50–56, November/December 2000.
21. Derrell Piper and Brian Swander. A GSS-API authentication method for IKE. Internet-Draft draft-ietf-ipsec-isakmp-gss-auth-07, IETF, July 2001. Expired.
22. Y. Rekhter, B. Moskowitz, D. Karrenberg, G J. De Groot, and E. Lear. Address allocation for private internets. RFC 1918, IETF, February 1996.
23. Christoph L. Schuba, Ivan V. Krsul, Markus G. Kuhn, Eugene H. Spaffold, Aurobindo Sundaram, and Diego Zamboni. Analysis of a denial of service attack on TCP. In *Proc. 1997 IEEE Symposium on Security and Privacy*, pages 208–223, Oakland, CA USA, May 1997. IEEE Computer Society Press.
24. Jonathon Trostle and Bill Gossman. Techniques for improving the security and manageability of IPsec policy. *International Journal of Information Security*, 4(3):209–226, 2005.
25. International Telecommunication Union. ITU-T recommendation X.509 (11/93) - The Directory: Authentication Framework, November 1993.