Problem Areas for the IP Security Protocols

Steven M. Bellovin
smb@research.att.com
AT&T Research

Abstract

The Internet Engineering Task Force (IETF) is in the process of adopting standards for IP-layer encryption and authentication (IPSEC). We describe a number of attacks against various versions of these protocols, including confidentiality failures and authentication failures. The implications of these attacks are troubling for the utility of this entire effort.

1 Introduction

The Internet Engineering Task Force (IETF) is in the process of adopting standards for IP-layer encryption and authentication (IPSEC) [Atk95c, Atk95a, Atk95b, MS95, MKS95a]. While these protocols should provide a marked increase in Internet security, they themselves have had a checkered history. It is very much worth recounting the design history, not just to avoid the "oral history" problem in the IPSEC working group, but also because we as a profession learn more from knowing what doesn't work. As a wise sage[1] once said, "Learn from the mistakes of others; you'll never live long enough to make them all yourself."

The failures we discuss here include confidentiality failures—attackers can read encrypted data–and spoofing failures—attackers can transmit phony data. In short, these attacks can render IPSEC useless.

Many (but not all) of the problems stem from the intrinsic properties of the encryption modes used, coupled with the lack of integrity checking in some security transforms and the use of host-pair keying. It has become painfully clear that these combinations are deeply flawed. People assume that since decrypting with the wrong key will yield garbage, additional integrity checking is not needed. Regrettably, this is not the case.

[1] Alfred E. Neuman of MAD Magazine

Proceedings of the Sixth Usenix UNIX Security Symposium, July 22-25, 1996, San Jose, CA. ftp://ftp.research.att.com/dist/smb/badesp.ps

Some of the attacks discussed here were presented informally at the 32nd IETF meeting in Danvers, MA, in March of 1995. Others have been been discussed elsewhere, such as on the IPSEC mailing list or in [WB96].

2 Properties of Encryption Modes

The ciphers of interest here fall into two broad categories. First, we have block ciphers such as DES [NBS77] used in *cipher block chaining* mode (CBC). Second, the use of stream ciphers has been suggested, in early versions of the SKIP protocol [Azi94] and with the standard ESP header [CW96]; these are byte-at-a-time ciphers. For our purposes, both modes have some significant limitations.

The discussion below focuses on aspects of interest to us. More detailed information on these and other cipher modes can be found in [Sch96].

2.1 Notation

We use $C_i = K[P_i]$ to mean "ciphertext C_i results from the encryption of plaintext P_i using key K. The corresponding decryption is written $P_i = K^{-1}[C_i]$. The symbol $\oplus$ denotes bitwise exclusive-OR.

In showing transforms, the subscript K in ESP_K denotes "ESP encryption using key K".

2.2 Cipher Block Chaining

CBC encryption [NBS80] operates by encrypting the exclusive-OR of each plaintext block and the previous ciphertext block:

$$C_i = K[P_i \oplus C_{i-1}].$$

To encrypt the first plaintext block, C_0 is set to the *initializaton vector* (IV). IVs may be agreed upon in advance, transmitted encrypted, or transmitted in the clear. Using non-constant IVs is often recommended, in order to disguise common prefixes. For our purposes, that does not matter much, as the first encrypted block will almost always be a TCP [Pos81c], UDP [Pos80], or IP [Pos81b] header, and thus will almost always vary.

Decryption is the inverse operation:

$$P_i = C_{i-1} \oplus K^{-1}[C_i].$$

To encrypt data that is not a multiple of the underlying cipher's block size, some sort of padding and length information must be added. There are a number of different techniques that may be used; none add much to the security of the encryption.

CBC mode encryption has several properties of interest to us. First, the prefix of a CBC encryption is the encryption of the prefix. That is, given a stream of ciphertext $\langle C_1, \ldots, C_i, \ldots, C_n \rangle$, the sequence $\langle C_1, \ldots, C_i \rangle$ is the encryption of $\langle P_1, \ldots, P_i \rangle$. (This may be complicated somewhat by a trailing padding and length indicator scheme.) An attacker can thus truncate a block of encrypted text.

A second property is a generalization of the first. A substring $\langle C_i, \ldots, C_j \rangle$ is a valid CBC encryption of $\langle P_i, \ldots, P_j \rangle$, so long as the IV can be set to C_{i-1}. This allows the attacker to extract any portion of the encrypted message.

The third interesting property is limited error propagation. If a ciphertext block is corrupted in transit, either deliberately or accidentally, only it and the following plaintext block are damaged. If a ciphertext block is dropped, only the following plaintext block will be damaged. The following example illustrates this:

$$\begin{aligned} P_i &= C_{i-1} \oplus K^{-1}[C_i] \\ P_{i+1} &= C_i \oplus K^{-1}[C_{i+1}] \\ P_{i+2} &= C_{i+1} \oplus K^{-1}[C_{i+2}]. \end{aligned}$$

Suppose block C_i is damaged. P_i will be garbled unpredictably, since it depends on the decryption of C_i. P_{i+1} will be damaged in a predictable fashion, since its value is derived from an exclusive-OR with C_i. But P_{i+2} will remain intact, since it depends on C_{i+2} and C_{i+1}, and does not derive from either C_i or P_{i+1}.

Taken collectively, these properties permit cut-and-paste operations. Ciphertext blocks from different messages encrypted with the same key can be combined; only the block immediately following the splice point will be garbled upon decryption.

2.3 Stream Ciphers

There are many ways to build stream ciphers; the ones we are interested in operate by generating a stream of key bytes k_i which are exclusive-ORed with the plaintext, one byte at a time:

$$C_i = k_i \oplus P_i.$$

Clearly, any substring of ciphertext bytes can be decrypted independently, so long as the right starting point is used. There is no error propagation; if a ciphertext byte is damaged, the corresponding plaintext byte is changed in a predictable way, and no other bytes are affected. Stream ciphers of this type cannot cope with byte deletions or insertions.

If the key byte stream is generated by encrypting a counter using a block cipher

$$k_i = K[i]$$

encryption and decryption can start at any point. A standard DES mode, *Output Feedback Mode* (OFB), works by feeding back the block cipher output into itsef:

$$k_i = K[k_{i-1}].$$

Keystreams generated by this mechanism can be cranked forward or backward from a known value of i and k_i, but cannot be started at an arbitrary point.

A third way to generate the key byte stream is by a specialized stream cipher. Whether or not you can restart at an arbitrary point or crank backwards depends on the details of the cipher design.

3 The Attacks

For most of these attacks, we will assume that ESP encryption [Atk95b] is used, but that AH authentication [Atk95a] is not used. Host-pair keying is used. That is, a single key exists between each pair of communicating hosts. This is in distinction to user-oriented keying, or connection-oriented keying, where many keys can be used between two given machines. As needed, we will assume that the attacker X has a legitimate login on one or both of the machines in question, but does not have privileged access to either. Finally, we assume that the attacker has the usual powers over transmitted data: the ability to read, modify, delete, or inject new packets.

3.1 Reading Encrypted Data

The primary purpose of encryption is privacy. An attacker who can read other people's messages has

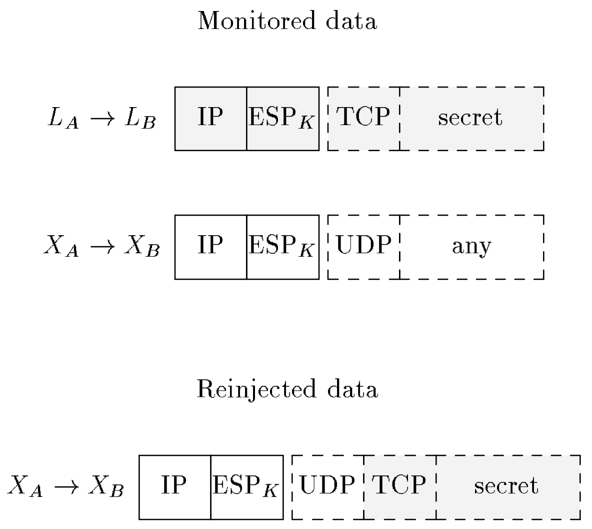

Figure 1: Cutting and pasting legitimate messages to decrypt someone else's traffic. The dashed lines denote encrypted data; the shaded boxes represent data belonging to the legitimate user.

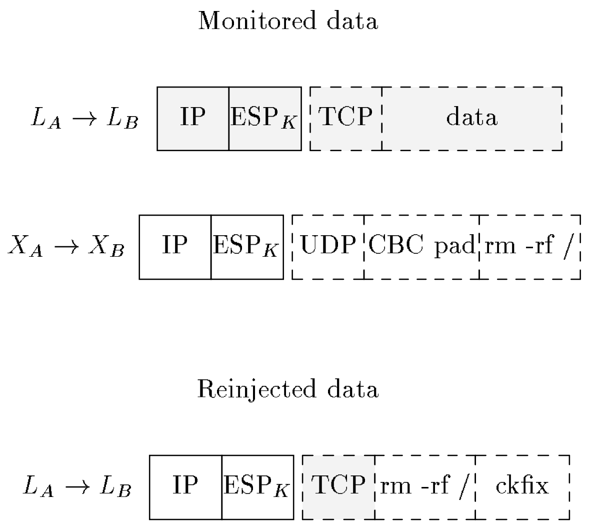

Figure 2: Cutting and pasting legitimate messages to hijack someone else's session. The dashed lines denote encrypted data; the shaded boxes represent data belonging to the legitimate user.

completely defeated the security system. There are a variety of ways in which this can be done.

Assume that a legitimate message is sent from user L_A on machine A to L_B on machine B. The attacker picks this up, and sends a UDP message from X_A to X_B. The encrypted portion of the first message is then inserted into the body of the second message, along with any necessary padding to make the lengths match; this forged message is reinjected into the network for receipt by X_B (Figure 1).

Because of the CBC self-healing properties, the first block of L_A's TCP header may be lost (or may not, if we copy the IV as well). Nothing else will be lost; the body will be readable. If we are using IPv4, the attacker may be able to request that no UDP checksum validation be done; on IPv6, where that isn't possible, on average only about 2^{16} tries are necessary to fool the checksum.

If L_A and L_B are using UDP to communicate, the attack may be even easier. X can wait until process L_B is finished, allocate the same UDP port number on host B as L_B used, and reinject the packets onto the wire where they will be received again by B.. The kernel will decrypt them, and pass them along. The same attack is probably possible with TCP, though it's a bit harder, as TCP's connection-oriented port numbers and sequence numbers will get in the way. Still, on modern high-speed networks it isn't that hard to make the sequence numbers wrap, and the attacker may be able to learn the necessary port numbers by polling via `netstat` on either machine.

3.2 Session Hijacking

The same sorts of techniques can be used to insert bogus data into someone else's encrypted session. Again, the attacker monitors both a legitimate packet and one sent from X_A to X_B. This second packet contains the data that is to be inserted into someone else's stream. If the legitimate packet can be deleted, the new data can simply be substituted into the payload; if not, a new packet can be constructed with the nasty stuff appended to the legitimate data (Figure 2). After all, TCP has no length field, and it is perfectly content to read a packet where part of the data has already been received but part is new.

Nominally, a padding block must be inserted at the splice point, to prevent the CBC decryption process from damaging valuable spoofed data. The confusion caused by these extra bytes, which will decrypt to garbage, are not significant; the attacker probably needs to send a few extra bytes anyway, to restore to a known state such as the shell prompt.

A related attack involves inducing a machine to send text of the attacker's choice, and cutting it at the proper CBC boundary. The resulting encrypted blocks can be reinjected onto the net, with a new IP header.

Using the chosen plaintext mechanisms discussed below (Section 3.9), it is not even necessary for the attacker to have a login on either machine. The message from X_A to X_B then becomes ciphertext emitted by host A, in response to X's prompting.

3.3 Fragmentation Attacks

Suppose that an IPSEC implementation does its security processing after fragmentation. Although prohibited by the RFCs, it is comparatively difficult for a "bump-in-the-cord" encryptor to avoid this practice, as the unit may be handed fragments by the host's IP implementation. In that case, the attacker can induce the machine to send a large packet (see Section 3.9), with nasty headers and data just after the fragmentation boundary. The attacker intercepts the packet, changes the IP header to zero the fragment offset field, and reinjects the packet. It will be treated as a complete packet when received. Note that both ESP and AH may be present; the packet will appear to be perfectly valid.

The root cause here is that the fragmentation fields in the IP header are subject to change during transmission, and hence are not included in the AH calculations. This may change for IPv6, where intermediate routers are not allowed to fragment packets. But in that case, the AH header would have to cover the fragmentation header as well. A more general solution is to use tunnel mode for all fragmented packets [WB96]; the inclusion of the extra IP header will protect the fragmentation indicators on the inside.

3.4 Weaknesses of Stream Ciphers

Early versions of the SKIP protocol [Azi94] suggested the use of RC4 [Sch96, pp. 397–398], a stream cipher. Packets included a 64-bit byte sequence number field; the decryptor's stream cipher engine would be cranked until it reached that point. This field also serves as a replay detection mechanism. Authentication was possible but not mandatory.

There are several problems with this scheme. The most obvious is a denial of service attack: an enemy could send a packet with a much larger sequence number, forcing the recipient to spend a lot of cycles turning the crank. In addition, legitimate packets would be rejected, as they would fall between the old sequence number value and the new one—it's difficult or impossible to unwind the cipher state.

The obvious counters to this are to limit the maximum change δ between the sequence number fields in two packets, and to cache recent previous states of the stream cipher engine. But the former runs afoul of the need to span network or host downtime, and the latter can probably be defeated by a burst of forged packets.

Other attacks are more serious. Stream ciphers such as RC4 suffer from a very serious disadvantage: changes to the ciphertext show up as predictable changes to the decrypted plaintext. Suppose that an attacker can trick a machine into sending a known message to a target. This packet can be intercepted, modified, and reinjected onto the wire.

A final attack combines these two threats. Suppose that δ is large enough that an attacker can cause the sequence number field to wrap around. This is difficult but by no means out of the question; if δ is 2^{32}, an enemy who can send at 60% of the bandwidth of an FDDI or 100BaseT net can accomplish this in less than 10 minutes. First cause one machine to send a moderately large amount of known plaintext to the target. This can be done in a variety of ways, such as having it forward a mail message. Next, wrap the sequence number counter. Use the known plaintext to recover the key stream used to protect your text, and use it to encrypt a new message.

A more recent proposal for use of stream ciphers [CW96] addresses some of these issues. For example, a maximum forward change δ is defined, and sequence number wrapping is explicitly prohibited. But this proposal explicitly rejects the inclusion of an integrity check, suggesting that in many cases its use is not necessary, and that an AH header can be added if desired.

3.5 Abusing IVs

Because IVs are sent in the clear, and because after decryption P_1 is produced by an exclusive-OR with the IV, an attacker can introduce predictable changes into P_1. For UDP packets, the entire header is in the first block, thus making it possible to divert packets to new connections. Because of the checksum, changing the TCP headers is somewhat harder, though probably not impossible. IP has a checksum as well, the packet identification field—an arbitrary number—is in P_1; it seems possible to change it to compensate for changes to the the fragmentation control fields; that in turn might enable some of the attacks described in [ZRT95].

Some other attacks on IVs are described in [VK83]; while not all of the scenarios described there are applicable to IPSEC, some are worrisome. But their suggestion that each security association use a separate, secret constant IV does not work well for us; it is too easy to recover most of the bits of the IV. Use a cut-and-paste attack to force decryption of the first block, which will in general be part of either a TCP or an IP header. For the latter, most of the fields are effectively constant, save for the fragment-id which we don't care about in any event; for the latter, the port number fields can be recovered by seeing what ports are in use and the sequence number can be deduced by a modified sequence number guessing attack [Mor85, Bel89]. Any remaining uncertainty can be dealt with by brute force; at most, $2^{1}6$ trials will be needed.

3.6 Encypted Hash Functions

It has been suggested that integrity checking can be accomplished by calculating a cryptographic hash of the input packet before encrypting, and encrypting both the packet and the hash. This scheme falls to a cut-and-paste attack using chosen plaintext.

Prepare a new packet, including the hash function output, and cause it to be transmitted. Snip it out of the intercepted packet, using the ciphertext block preceeding the chosen text as the new IV. If the hash function is at the beginning of the packet and a secret IV is used, the mechanisms described above can be employed in conjunction with this scheme.

Other attacks based on integrity-checking failures may be found in [JMM85, SG92a, SG92b, SG93].

3.7 Proxy Encryption

Hosts that will forward received packets that are not addressed to them may be victimized by a proxy encryption attack [WB96]. In this attack, the enemy builds a packet with the IP source address of the forwarder, and a target of some destination machine. If the IPSEC implemenation isn't careful, it will encrypt and authenticate the packet using its own secret keys, thereby convincing the target of the provenance of the message.

Routers are most vulnerable to this attack, since they are in the business of forwarding packets. However, ordinary hosts may be targeted as well. Mechanisms for launching the attack include IP source routing, IP tunneling, and direct injection onto the local network.

3.8 Reading Short Blocks

David Wagner has devised an attack that uses known plaintext and simple active measures to read encrypted data. While the attack is not universally applicable, it does work for the user-to-host data—including any typed passwords—in `telnet` sessions

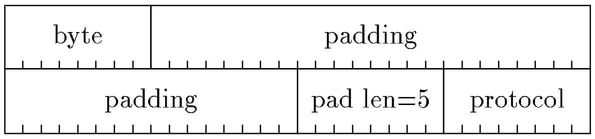

Figure 3: Format of the last block of an ESP packet, when only one data byte is present.

if either tunnel mode ESP or the TCP timestamp options [BBJ92] are used.

Because DES is a block cipher, data shorter than the block length—64 bits—must be padded to the block length before encryption. The format used for the standard DES-CBC transform is shown in Figure 3. Suppose a single keystroke is being sent. If the preceeding data exactly fills a multiple of this size, this keystroke will occupy the first byte of the last block. The next-to-last byte will contain 5, and the last byte will contain either 6 for TCP or 4 for IP-in-IP. The contents of the intermediate bytes are ignored by the receiving host.

A standard TCP header is 20 bytes long, which is not a multiple of the DES block length. If tunnel mode is used, though, the 20-byte IP header is included, raising the total length to 40 bytes, or eight DES blocks. Similarly, the recommended format for the timestamp options [BBJ92, Appendix A], occupies 12 bytes, bringing the total TCP header length to 32 bytes, which is also an integral number of DES blocks. In either of these cases, single keystrokes will be sent alone in a pad block.

The trick, then, is to send ciphertext blocks whose seventh and eighth bytes of plaintext are the appropriate constants, and whose first byte ranges over the possible character set. If the guessed byte value is incorrect, the TCP checksum will be wrong, and the segment will be silently dropped. A correct value, on the other hand, will elicit an ACK packet, the existence of which (though not, of course, the contents) will be detectable by the attacker. This is true even if the packet represents an old duplicate; the replays will be ignored but will still generate an ACK, according to the TCP specification [Pos81c].

We now need ciphertext blocks with known values for these three bytes; this totals 2^{24} blocks. These could either be generated by the chosen plaintext techniques discussed in Section 3.9, or it could be gleaned by observation. That isn't that hard; if IP tunnel mode is used, the first encrypted block is part of the IP header, and the fields of interest are almost always constant: the IP version, the header length, the type of service, and the fragment offset.

We cannot just use the observed ciphertext for a known plaintext message; the CBC formatting interferes. However, we can recover the appropriate information. Suppose that C'_i corresponds to plaintext block P'_i encrypted under key K. Then by the rules of CBC decryption,

$$K^{-1}[C'_i] = P'_i \oplus C_{i-1}.$$

Call this value N_i. If the C'_{i-1} values differ, so will the N_i, even if the P'_i values are all the same. This allows us to collect all 2^{24} necessary values. We do not in fact need all possible values of the last two bytes; at this point, however, we do not know which we will need.

Now intercept a encrypted packet $\langle C_1, \ldots, C_n \rangle$ from the target stream. Generate a group of new messages $\langle C_1, \ldots, C_{n-1}, T_t \rangle$ where $T_t = C'_j$ such that $C_{n-1} \oplus N_j$ has t as the first byte and the proper last two bytes. When the receiver decrypts T_k, it sees N_j; stripping off the chaining yields $C_{n-1} \oplus N_j$, which has the proper value. Reinject each of these messages and watch for a 40-byte response. If you see one, you know that t was the encrypted byte.

There are several interesting corollaries to this attack. First, unlike most of our cut-and-paste attacks, the use of AH may not help. If the header sequence is IP-ESP-AH-TCP, the authentication header simply ensures that the TCP portion is correct. But if our guess at t is right, it *will* be correct; we will simply know that twice, once from AH and once from TCP. To be sure, AH failures can trigger alarms, but it isn't clear that this is useful; tearing down a session that received too many AH failures is an invitation to denial-of-service attacks.

Second, the attack is aided because the DES-CBC specification [MKS95a] requires that received padding bytes be ignored. If they had some known fixed value, checked by the receiver, it would be much harder to generate the T_k messages, since far too much known plaintext would be needed. On the other hand, using fixed values will generate plenty of known plaintext for cryptanalytic attacks, every time the user hits ENTER.

A third observation we can make is that using more padding (Figure 4), as is permitted, does not help; in fact, it hurts. We still know that most upstream messages contain a single data byte; if they are the longer, the excess is probably random padding, which means that the block following the header contains only the data byte we are interested in. We don't even have to worry about the padding length and protocol fields, which reduces the known plaintext requirement to 2^8 blocks. In fact, we can use this trick to reduce the total known plaintext requirement to 2^8 blocks!

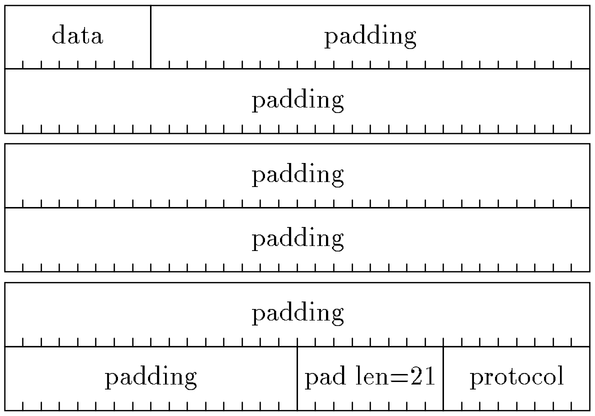

Figure 4: Format of the end of an ESP packet with 21 bytes of padding.

As before, we build T_t; this time, though, we only care about the first byte t. We now build two more trailing ciphertext blocks, T' and T''. The last block, T'', is a random selection from our table of known plaintexts. But we know what it will decrypt to. The final value comes from the exclusive-OR with T'; we select it so that the final two bytes come out right. We have no idea what the plaintext corresponding to T' will be, but we don't care; it's random padding that the receiver will ignore.

Can we generalize this attack to recover longer sequences than one byte? We may be able to extend it to two byte blocks; beyond that seems unlikely. Collecting the 2^{16} known plaintexts is not too hard; however, we would have to transmit 2^{16} packets containing $T_{t_1 t_2}$.

Going to three bytes creates even more problems; apart from needing to send 2^{24} trial packets, the TCP checksum is only 16 bits; accordingly, there is ambiguity in the decryption. Without good knowledge of the distribution of the t_i bytes, an accurate answer is impossible. Alternatively, one could raise 2^{24} alarms by relying on AH to do the detection.

3.9 Chosen Plaintext

Many of these attacks are aided by how easy it is to make a machine encrypt chosen plaintext. For example, one can often connect to the mail port, and send a long message destined for some user on a neighboring machine. The first machine will happily forward it to the second, permitting the ciphertext to be monitored. To increase the fun, SMTP-level source routing can be done, forcing the first machine to send it to the second, and the second to a nonexistent user on a third machine; this last will cause the message to bounce and not be transmitted, but the damage will be done.

Other useful techniques involve exploiting weaknesses in the IPSEC implemenation. If, for example, a machine will respond to a plaintext ICMP [Pos81a] or UDP ECHO packet with an encrypted reply, an attacker can forge the initial message and monitor the response. An alternative technique that is often useful is to send an encrypted ECHO message and look for a plaintext response; depending on the local configuration, that may happen as well. We know of several different implementations that will fall victim to one or both of these scenarios.

Beyond their utility for the high-level attacks we have described here, chosen plaintext has cryptographic significance as well. Many cryptanalytic attacks, such as differential cryptanalysis [BS93b, BS91, BS93a], depend on the attacker being able to choose the plaintext to be encrypted. While the quantities needed to attack DES are still out of reach, it appears to be quite feasible to trick local machines into encrypting tens of gigabytes of data. For some ciphers, such as members of the FEAL family [Sch96, pp. 308–311], this may be quite sufficient to mount an attack.

4 Key Changes versus SKIP

A central principle behind SKIP [AMP95] is that a long-lived master key exists implicitly between any two hosts. But this automatic keying makes it difficult for a host to delete a key unilaterally. Recent drafts have addressed this by creating a counter n. If the value of n in a received packet differs from the host's n by more than 1, the packet is rejected. A host can thus delete keys by bumping its counter by 2. Unfortunately, the sender's n is bound to a coarsely synchronized clock, implying that it would not know how to use the new key for up to one hour. Addressing this issue would require long-lived state about the current offset of n on a per-host basis, or easy certificate revocation.

5 Defenses

Against many of these attacks, the proper defense is use of integrity-checking. If a message is properly checked, it cannot be cut apart. More precisely, all received messages should be checked for integrity, using acceptably strong cryptographic techniques. We note that the current protocols do not have a separate mechanism for integrity, as opposed to authentication; however, the authentication transforms do protect the integrity of the message.

A second generic defense technique is to avoid reuse of keying material for more than one "connection". An attacker cannot cut and paste between connections if they use different keys; the inserted material will not decrypt properly.

If this is not feasible, keys should be changed reasonably frequently. For stream ciphers especially, it is necessary to do this based on time, data received, and too large a difference in the indicated sequence number. Recent versions of SKIP [AMP95] have the proper facilities for doing this; it is imperative that older ones not be used.

Replay defenses are also a good idea. If per-connection keying is not used, they are mandatory in certain contexts; packet authentication will not help reject a replay of a perfectly valid packet.

6 Conclusions

The attacks described here are troubling. We have outlined a fair number of very different mechanisms; we strongly suspect there are others as well. Proper cryptographic practice will certainly help; however, there are some very subtle design issues as well, and these are probably harder to find and fix.

In general, hosts should aim for per-connection or per-user keying. The former is probably preferable; the Bad Guys can send evil things to a terminal by way of email, and then replay them to the X server, which would have the same userid. Nor is it always clear who the proper "user" is. Consider the `rsh` protocol, where a separate connection is set up for `stderr`. To what userid should this second connection be keyed? Note that while the client program could in principle be running as either the user or `root` at this point, the server has not been informed of the user's identity yet.

To avoid some of the chosen plaintext attacks, we suggest a simple security policy: never reply to a plaintext message with an encrypted one, and vice versa. If necessary, an ICMP error message can be sent, or key negotiation commenced. Furthermore, SPI pairs should be established by the key negotiation process; messages received via one SPI should always be replied to using its peer. This puts certain constraints on the key management process, especially during rekeying.

It is quite clear that encryption without integrity checking is all but useless. We strongly recommend that all systems mandate joint use of the two options. It is in some sense irrelevant if AH and ESP are two separate protocols, or if integrity checking is made an integral part of each ESP transform; however, the bookkeeping issues may be considerably simplified if the latter path is chosen. Consider, for example, the problem of an authentication key expiring independently of the associated encryption key, or inconsistent pairwise relationships for AH and ESP. Furthermore, not all combinations are secure; given the way authentication failures can be used to compromise secrecy, the authentication transform must be at least as strong as the secrecy mechanism. It would seem to make little sense to combine MD5 [Riv92], with its $O(2^{64})$ strength against birthday attacks, with triple-DES [MKS95b].

These attacks and recommendations, taken *in toto*, leave us feeling very nervous about network layer encryption. It may be that its promise of transparent, ubiquitous security cannot be kept, at least in general. Use of it when outsiders have access to the endpoint machines, via either logins or network services, seems particularly inadvisable. We suggest that its use be restricted to the following situations:

1. Router-to-router encryption to provide virtual private networks, in conjunction with a firewall. Insiders have other means of attack; the firewall should keep outsiders from mounting chosen plaintext attacks on inside machines. There are still some risks—mail destined for a machine inside one private cloud could be routed by the enemy to the mail gateway inside another cloud; the traffic, when relayed, will be encrypted. There are implications here for the proper integration of encryption and firewalls; we will not pursue the matter further here.

2. As a special case of the above, a "call home" tunnel from a mobile machine to its firewall. A great deal of care must be taken, though, to ensure that the mobile machine does not respond at all to packets sent to its outside address.

3. Possibly between two single-user hosts, though the potential for attack here is quite high.

For more general use, we recommend moving towards the cryptographic processing towards the transport layer. The semantics are quite clear for TCP: for each new socket, create a new pair of SPIs. For UDP, the binding must be between a socket and every host it has ever communicated with. In either case, when the socket is destroyed all of its associated SPIs must be destroyed as well. Looked at another way, the incoming SPI *is* a pointer to the socket. (A useful side-effect of this policy is that ICMP messages will work again: the returned portion of the packet will contain the SPI, which points to the socket.)

A scheme like this could put a heavy load on the key management protocol. Even a simple set of rekey messages would add several round trips; for short exchanges, such as HTTP transfers, this is probably unacceptable. We suggest that the key exchange protocol allocate n SPIs, n probably a power of 2, where the key for SPI_i is some one-way function of i and the negotiated master key. A host could start using a new SPI in the allocated range without further negotiation; to avoid race conditions, the initiator should start allocating from one end while the responder allocates from the other. A new group of SPIs would be allocated when too few were left in the old group.

The situation is rather more problematic for things like DNS servers, which would have to maintain very many active keys. SKIP would simplify the situation, as each message could have its own traffic key K_p; however, it suffers from the flaw that a receiver has no way to force the sender to use a different key. Probably, the right answer is to omit IPSEC entirely for DNS messages, and instead rely on authenticated DNS records [EK96].

7 Acknowledgments

Some of these specific attacks were first pointed out by others, including Ashar Aziz of Sun Microsystems and David A. Wagner of the University of California at Berkeley. Wagner, Ran Atkinson, Perry Metzger, and Hilarie Orman made many useful suggestions about draft versions of this paper.

References

[AMP95] Ashar Aziz, Tom Markson, and Hemma Prafullchandra. Simple key-management for Internet protocols (SKIP). Internet draft; work in progress, December 21, 1995. (draft-ietf-ipsec-skip-06.txt).

[Atk95a] R. Atkinson. IP authentication header. Request for Comments (Proposed Standard) RFC 1826, Internet Engineering Task Force, August 1995.

[Atk95b] R. Atkinson. IP encapsulating security payload (ESP). Request for Comments (Proposed Standard) RFC 1827, Internet Engineering Task Force, August 1995.

[Atk95c] R. Atkinson. Security architecture for the internet protocol. Request for Comments (Proposed Standard) RFC 1825, Internet Engineering Task Force, August 1995.

[Azi94] Ashar Aziz. Simple key-management for Internet protocols (SKIP). Obsolete Internet draft, October 25, 1994. (draft-ietf-ipsec-aziz-skip-00.txt).

[BBJ92] D. Borman, R. Braden, and V. Jacobson. TCP extensions for high performance. Request for Comments (Proposed Standard) RFC 1323, Internet Engineering Task Force, May 1992. (Obsoletes RFC1185).

[Bel89] Steven M. Bellovin. Security problems in the TCP/IP protocol suite. *Computer Communications Review*, 19(2):32–48, April 1989.

[BS91] Eli Biham and Adi Shamir. Differential cryptanalysis of DES-like cryptosystems. *Journal of Cryptology*, 4(1):3–72, 1991.

[BS93a] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard.* Springer-Verlag, Berlin, 1993.

[BS93b] Eli Biham and Adi Shamir. Differential cryptanalysis of the full 16-round DES. In *Advances in Cryptology: Proceedings of CRYPTO '92*, pages 487–496. Springer-Verlag, 1993.

[CW96] Germano Caronni and Marcel Waldvogel. The ESP stream transform. Internet draft; work in progress, April 1996. (draft-caronni-esp-stream-00.txt).

[EK96] Donald E. Eastlake, 3rd and Charles W. Kaufman. Domain name system protocol security extensions. Internet draft; work in progress, January 30, 1996. (draft-ietf-dnssec-secext-09.txt).

[JMM85] Robert R. Jueneman, Stephan M. Matyas, and Carl H. Meyer. Message authentication. *IEEE Communications*, 23(9):29–40, September 1985.

[MKS95a] P. Metzger, P. Karn, and W. Simpson. The ESP DES-CBC transform. Request for Comments (Proposed Standard) RFC 1829, Internet Engineering Task Force, August 1995.

[MKS95b] P. Metzger, P. Karn, and W. Simpson. The ESP triple DES-CBC transform. Request for Comments (Experimental) RFC 1851, Internet Engineering Task Force, October 1995.

[Mor85] Robert T. Morris. A weakness in the 4.2BSD UNIX TCP/IP software. Computing Science Technical Report 117, AT&T Bell Laboratories, Murray Hill, NJ, February 1985.

[MS95] P. Metzger and W. Simpson. IP authentication using keyed MD5. Request for Comments (Proposed Standard) RFC 1828, Internet Engineering Task Force, August 1995.

[NBS77] NBS. Data encryption standard, January 1977. Federal Information Processing Standards Publication 46.

[NBS80] NBS. DES modes of operation, December 1980. Federal Information Processing Standards Publication 81.

[Pos80] J. Postel. User datagram protocol. Request for Comments (Standard) STD 6, RFC 768, Internet Engineering Task Force, August 1980.

[Pos81a] J. Postel. Internet control message protocol. Request for Comments (Standard) STD 5, RFC 792, Internet Engineering Task Force, September 1981. (Obsoletes RFC0777).

[Pos81b] J. Postel. Internet protocol. Request for Comments (Standard) RFC 791, Internet Engineering Task Force, September 1981. (Obsoletes RFC0760).

[Pos81c] J. Postel. Transmission control protocol. Request for Comments (Standard) STD 7, RFC 793, Internet Engineering Task Force, September 1981.

[Riv92] R. Rivest. The MD5 message-digest algorithm. Request for Comments (Informational) RFC 1321, Internet Engineering Task Force, April 1992.

[Sch96] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, New York, second edition, 1996.

[SG92a] Stuart G. Stubblebine and Virgil D. Gligor. On message integrity in cryptographic protocols. In *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, pages 85–104, Oakland, CA, May 1992.

[SG92b] Stuart G. Stubblebine and Virgil D. Gligor. On message integrity in cryptographic protocols. Computer Science Technical Report 2843, University of Maryland, College Park, MD, February 1992.

[SG93] Stuart G. Stubblebine and Virgil D. Gligor. Protocol design for integrity protection. In *Proc. IEEE Computer Society Symposium on Research in Security and Privacy*, pages 41–53, Oakland, CA, May 1993.

[VK83] V. L. Voydock and S. T. Kent. Security mechanisms in high-level network protocols. *ACM Computing Surveys*, 15(2):135–171, June 1983.

[WB96] David A. Wagner and Steven M. Bellovin. A "bump in the stack" encryptor for MS-DOS systems. In *Proceedings of the Symposium on Network and Distributed System Security*, pages 155–160, San Diego, February 1996.

[ZRT95] P. Ziemba, D. Reed, and P. Traina. Security considerations for IP fragment filtering. Request for Comments (Informational) RFC 1858, Internet Engineering Task Force, October 1995.