Cryptography in Theory and Practice: The Case of Encryption in IPsec*

Kenneth G. Paterson and Arnold K.L. Yau**

Information Security Group,
Royal Holloway, University of London,
Egham, Surrey, TW20 0EX,
United Kingdom.
{kenny.paterson,a.yau}@rhul.ac.uk

Abstract. Despite well-known results in theoretical cryptography highlighting the vulnerabilities of unauthenticated encryption, the IPsec standards mandate its support. We present evidence that such "encryption-only" configurations are in fact still often selected by users of IPsec in practice, even with strong warnings advising against this in the IPsec standards. We then describe a variety of attacks against such configurations and report on their successful implementation in the case of the Linux kernel implementation of IPsec. Our attacks are realistic in their requirements, highly efficient, and recover the complete contents of IPsec-protected datagrams. Our attacks still apply when integrity protection is provided by a higher layer protocol, and in some cases even when it is supplied by IPsec itself.

Keywords: IPsec, integrity, encryption, ESP.

1 Introduction

The need for authenticated encryption is well understood in the cryptographic research community – see for example [4, 5, 14]. High-profile examples where the lack of strong integrity checks is known to lead to attacks or where inappropriate use of integrity mechanisms still leaves systems vulnerable are plentiful [3, 6–8, 28, 30]. However the process of adopting authenticated encryption in fielded systems is slower. Naturally, it takes time to translate theory into standards, standards into products and finally, for users to take up the latest versions of products. There is also resistance to change without clear and easily-absorbed evidence that such change is imperative. Attacks in the cryptographic literature can be rather technical and difficult for non-experts to understand. In some cases, it may also be that the attacks are not perceived by users as having a high

* The work described in this paper was partly supported by the European Commission under contract IST-2002-507932 (ECRYPT). An extended version is available [25].

** This author supported by EPSRC and Hewlett-Packard Laboratories Bristol through CASE award 01301027.

impact. Theoreticians are rightly concerned about attacks on indistinguishability of ciphertexts, but users are perhaps less so. Attacks requiring huge numbers of chosen plaintexts are interesting to theoreticians, but may not unduly concern practitioners. Attacks on paper are easier to dismiss than fully demonstrated attacks that work in practice against deployed systems.

In this paper, our focus is on the use of integrity protection and encryption in IPsec, a widely-used suite of protocols providing security for IP. We provide a short introduction to IPsec in Section 2. Bellovin [6] was the first to point out that the lack of integrity protection in the first version of IPsec's encryption protocol ESP (Encapsulating Security Payload) [1] leads to security weaknesses. However, the attacks in [6] are actually quite limited in their practical impact. A close examination of [6] shows that the attacks presented in [6, Sections 3.1 and 3.2] only work in the rather unrealistic scenario where the attacker has access to accounts on the two network hosts performing the IPsec processing. The other concrete attack in [6] is contained in Section 3.8 and is attributed to Wagner. It recovers just a single byte of plaintext, from datagrams having special formats, and then only if 2^{24} ciphertexts matching chosen plaintexts are available to the attacker. Moreover, the attacks in [6] (and the related paper [22]) are really only sketches of what might be possible rather than fully implemented, working attacks: they are examples of "attacks in theory". Nevertheless, Bellovin's attacks are well-known in the cryptographic and IPsec standards communities, and are cited in subsequent versions of the ESP standards [16, 18]. The version in current use, [16], refers to [6] when warning of the dangers of using encryption without additional integrity protection, and requires support for integrity protection. However it also mandates that any implementation of ESP *must* include support for encryption-only processing. This surely illustrates the chasm that exists between the theory and practice of cryptography. Note that the developers of [16] did have good practical reasons (backward compatibility and performance) for mandating support for an encryption-only mode.

It is our belief that the availability of the encryption-only option in IPsec has led users into actually using it, in spite of Bellovin's work. After all, users do not typically read RFCs or research papers, and an inexperienced network administrator might reasonably believe that it is sufficient to use an encryption algorithm on its own to provide confidentiality for data, especially when selecting from amongst the myriad of IPsec options. (This point is also made in [10].) We have found several on-line tutorials showing how to configure IPsec VPNs using ESP for encryption with no additional integrity protection.[1] After the release of the vulnerability announcement [24] describing our attacks, we became aware that some vendors were aware of Bellovin's work and had taken steps to prevent the selection of encryption-only configurations, but others were much less well-informed, or less concerned.

[1] See for example: `http://www.netbsd.org/Documentation/network/ipsec` and `http://lartc.org/howto/lartc.ipsec.tunnel.html`.

1.1 Our Contribution

We present new attacks against the encryption-only configuration of IPsec that are as realistic and devastating as possible, with the aim of finally convincing users not to select it. In this respect, our attacks have several attractive features. Firstly, they are ciphertext-only attacks. Thus they do not require any special operating conditions under which, for example, the ciphertexts matching chosen plaintexts are generated. Nor do they require large amounts of ciphertext to be successful: the attacks can be mounted given only a single encrypted datagram. Secondly, the attacks merely require the attacker to be able to inject IP datagrams into the network and intercept certain responses. Some variants of our attacks even enable these responses to be sent directly to the attacker's machine. Thirdly, the attacks are very efficient. For example, one variant that we have implemented requires the injection of only a handful of datagrams to recover the complete contents of a datagram encrypted using AES. Fourthly, the attacks are flexible, with a range of variants being applicable in different circumstances. And finally, we have written an attack client which shows that the attacks work in practice against the native implementation of IPsec in Linux. For example, our client effectively allows a real-time cryptanalysis of encryption-only IPsec when AES is used as the encryption algorithm. In all these senses, our attacks improve on the pioneering work of Bellovin [6].

Our work also has consequences for the newly published version of ESP [18]. This RFC no longer requires mandatory support for encryption-only, and repeats the advice of [16] concerning the need for integrity protection, but then goes on to say: *"ESP allows encryption-only [...] because this may offer considerably better performance and still provide adequate security, e.g., when higher layer authentication/integrity protection is offered independently."* It is already known in theory that applying authentication followed by encryption to build an authenticated encryption scheme does not result in a generically secure construction [19]. We demonstrate that relying on higher layers for the provision of integrity in IPsec is inherently insecure in practice as well. Some of our attacks even apply to configurations using the IPsec protocol AH (Authentication Header) for integrity protection.

More generally, our attacks provide a stark illustration, should one still be required, of the general need to make appropriate use of authenticated encryption in fielded systems. We hope that this paper will also be of use to theoreticians in the field of authenticated encryption searching for convincing real-world examples to motivate their work.

A further theme of this paper is to illustrate the gaps that exist between cryptography as studied in theory, as defined in standards, as implemented by software engineers, and as actually consumed by users. For example, we have already commented on the differences in viewpoints of theoreticians and users, and how this can lead to the use of encryption-only ESP in practice. As another example, our attacks should in fact be prevented by any RFC-compliant implementation of IPsec, because of some seemingly innocuous post-processing checks specified in the architectural standard for IPsec [15]. Yet the native Linux version of IPsec fails to implement these checks. Drawing on our experiences with IPsec, we make some recommendations which we hope will help to bridge these gaps.

2 Background

2.1 IPsec

IPsec, as defined in RFCs 2401–2412, provides security at the IP layer. The interested reader is invited to consult [9, 12] for accessible introductions to IPsec. Implementations of IPsec exist in Microsoft Windows XP, in the Linux kernel from release 2.6 onwards.[2] Various other open source projects are also developing IPsec implementations and IPsec is widely supported in commercial networking hardware. The IPsec protocols provide data confidentiality, integrity protection, data origin authentication and anti-replay services as well as supporting automated key management.

The IPsec protocols can be deployed in two basic modes: transport and tunnel. In tunnel mode, on which we focus here, cryptographic protection is provided for entire IP datagrams. In essence, a whole datagram plus security fields is treated as the new payload of an outer IP datagram, with its own header, called the outer header. The original, or inner, IP datagram is said to be *encapsulated* within the outer IP datagram. In tunnel mode, IPsec processing is typically performed at security gateways on behalf of endpoint hosts. The gateways could be perimeter firewalls or routers.

IPsec provides authentication and integrity protection and/or confidentiality services through the AH and ESP protocols. Our focus here is on the ESP protocol, as defined in [16, 18]. ESP is normally invoked to provide confidentiality, and usually makes use of a block cipher algorithm operating in CBC mode. In tunnel mode, the entire inner IP datagram is encrypted and forms part of the payload of the outer IP datagram. The use in ESP of a variety of block ciphers has been specified, including DES [21], triple-DES [26] and AES [11]. ESP in tunnel mode inserts security information in the form of a header between the outer IP header and the encrypted version of the inner datagram. This ESP header indicates which algorithms and keys were used to protect the payload in a 32-bit field called the Security Parameters Index (SPI). The ESP header also contains a 32-bit sequence number to prevent packet replays; when ESP is used with encryption-only, this sequence number is simply ignored by IPsec implementations (as it is not protected in any way). ESP in tunnel mode may also append an authentication field after the encrypted portion. This contains a MAC value if ESP's optional integrity protection features are in use.

Further discussion of IPsec configuration and the combined usage of AH and ESP in tunnel and transport modes is beyond the scope of this paper. IPsec provides an automated key management service through the Internet Key

[2] All further references to Linux in this paper refer to official release 2.6.8.1 of the Linux kernel from `http://kernel.org`.

Exchange (IKE) [13]. We will simply assume that key establishment for ESP has taken place, either manually or using IKE.

2.2 CBC Mode Encryption in ESP

We outline how CBC mode is used by ESP in tunnel mode. For more details, see [16, 21, 11, 26]. First of all, the original (inner) datagram that is to be protected is treated as a sequence of bytes. This sequence is padded and then a single Next Header byte is appended. It is permissible for the padding to be of variable length and to extend over multiple blocks. We assume throughout that the minimum amount of padding is used, though our attacks are easily modified to handle variable length padding. Let us assume that the byte sequence after padding consists of q blocks, each of n bits. We denote these blocks by $P_1, P_2, \ldots, P_q$. We use K to denote the key used for the block cipher algorithm and $e_K(\cdot)$ ($d_K(\cdot)$) to denote encryption (decryption) of blocks using key K. An n-bit initialization vector, denoted IV, is selected at random. Then ciphertext blocks are generated according to the equations:

$$C_0 = IV, \quad C_i = e_K(C_{i-1} \oplus P_i), \quad (1 \leq i \leq q).$$

The encrypted portion of the outer datagram is then defined to be the sequence of $q+1$ blocks $C_0, C_1, \ldots, C_q$.

At the receiving security gateway, the payload of the outer datagram can be recovered using the equations: $P_i = C_{i-1} \oplus d_K(C_i), 1 \leq i \leq q$. Any padding and the Next Header byte can then be stripped off. At this point, Section 5.2 of the IPsec architectural RFC [15] mandates that implementations should check that the cryptographic processing performed to recover the inner datagram does in fact match that specified in local IPsec policies. Presumably, if the check fails, the datagram should be dropped, though this is not made explicit in [15].[3] In the Linux kernel implementation of IPsec, the inner datagram is passed directly to the IP software on the receiving gateway, without any policy checks being performed. This IP software usually just routes the inner datagram to the intended destination specified in the destination address of the inner datagram.

2.3 Bit Flipping Attacks

CBC mode has a well-known weakness, commonly known as the bit flipping vulnerability. Suppose an attacker captures a CBC mode ciphertext $C_0, C_1, \ldots, C_q$, then flips (inverts) a specific bit j in C_{i-1} and injects the modified ciphertext into the network. Upon receipt and decryption, this bit flip is transformed into a bit flip in position j in the plaintext block P_i. This can be seen by examining the decryption equation $P_i = C_{i-1} \oplus d_K(C_i)$. Thus an attacker can introduce controlled changes into the value of block P_i seen by the decrypting party, simply by flipping bits in C_{i-1} and injecting modified ciphertexts.

[3] Note that these checks are not specified in the ESP RFCs [16, 18]. The requirement to drop datagrams has now been made explicit in [17].

Of course, a problem for the attacker is that any modification to C_{i-1} typically results in a value of P_{i-1} that is effectively randomized. On the other hand, if the modification is made in C_0 (equal to IV), then no damage to plaintext blocks will result.

2.4 IP Datagram Headers

The execution of our attacks on ESP in tunnel mode depends in a detailed way on the structure of the headers of IP datagrams and on the order in which the fields of these headers are processed. We focus here only on IPv4 headers, as specified in detail in [20], and on describing those fields that are key to our attacks. The lay-out of the IP header is shown schematically in Figure 1.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version|  IHL  |Type of Service|          Total Length         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Identification        |Flags|      Fragment Offset    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Time to Live |    Protocol   |         Header Checksum       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                       Source Address                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Destination Address                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                    Options                    |    Padding    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Fig. 1. Structure of IP header according to RFC 791, [20].

The IHL (Internet Header Length) field is 4 bits long and has a value between 5 and 15. This field indicates the length of the header in 32-bit words. The typical value is 5; larger values indicate that options bytes are present after the main header, in the Options field. This field can be up to ten 32-bit words (40 bytes) in length. It has a strict format; if the format is not followed, then IP implementations typically generate an ICMP (Internet Control Message Protocol) "parameter problem" message which is routed to the host indicated in the Source Address field. Experiments confirm that, upon receipt of a datagram with random bytes in the Options field, the implementation of IP in Linux generates an ICMP message with probability roughly 98.5%. We discuss ICMP in more detail below.

The Protocol field is 8 bits (1 byte) long and indicates which upper layer protocol is carried in the IP datagram payload. A minimal set of supported protocols include ICMP, TCP and UDP. When an IP datagram reaches its intended destination (as specified in the 32-bit Destination Address field), the protocol field is inspected. This value determines to which upper layer protocol the payload is passed. If the field contains a value corresponding to a protocol that is not supported at that host, then the local IP implementation should generate an ICMP "protocol unreachable" message.

The Header Checksum field is a 16-bit (2-byte) value that is formed by interpreting the header (including the Options field if present) as a sequence of 16-bit words, summing them using 1's complement arithmetic, and then taking the 1's complement of the result. If the Header Checksum fails, the datagram is discarded silently.

In Linux, the sequence of steps taken by IP when processing a datagram is as follows. First of all, basic checks are performed on the Version field and IHL field. The next action is to check the Header Checksum field. After this, a datagram length check is carried out using the Total Length field. The datagram is dropped if any of these checks fails. Next, options processing is carried out if the IHL field indicates that options are present. Assuming this is completed successfully, a routing decision is made: either the datagram is delivered locally or is forwarded to another host. In the former case the Protocol field is used to determine the upper layer protocol to which the datagram payload should be passed. In the latter case, the TTL field is checked and the datagram dropped if the TTL has reached zero.

2.5 ICMP

ICMP is a vital part of IP implementations, allowing network problems to be reported to Internet hosts, routes to be tested, and diagnostics to be gathered. ICMP was originally specified in [27], and revised for IPv4 routers in [2]. In the event of a "problem datagram" being received by a host, that host generates an ICMP message. This message includes the entire IP header of the offending datagram (including any options), together with a variable number of bytes of the datagram's payload. According to [27], 8 bytes of payload should be included. On the other hand, according to [2], the ICMP datagram should contain as much of the original datagram as possible without the length of the ICMP datagram exceeding 576 bytes. This is intended to aid fault diagnosis, and is how ICMP is implemented in the Linux kernel.

3 Attacks Based on Destination Address Rewriting

We are now ready to discuss our first group of attacks on encryption-only ESP in tunnel mode. We focus on the case where the block cipher used by ESP has 64-bit blocks. The two-phase attack we describe here serves as an introduction to the more sophisticated attacks to follow. We describe the attack in the context of a pair of security gateways communicating using encryption-only ESP in tunnel mode to protect the traffic between them. The attack also works in more general applications of this configuration of ESP.

We need to make one major assumption for the attack to work: that the attacker, controlling the host located at IP address `AttAddr`, knows the destination IP address `DestAddr` of the target inner datagrams. This assumption will be relaxed shortly.

3.1 The First Phase

Recall that the Destination Address field lies in the fifth 32-bit word of the IP header, and therefore forms the first 32 bits of plaintext block P_3 in the sequence of blocks to be encrypted in CBC mode by ESP. The second 32 bits of this block is the first 32 bits of the payload of the inner datagram. This phase proceeds as follows, with the attacker at `AttAddr` listening for IP datagrams during the attack (see also Figure 2):

1. Capture a target ESP-protected outer datagram from the network. Let $C_0, C_1, \ldots, C_q$ denote the encrypted portion of this datagram's payload.
2. Modify block C_2 in the first 32 bits by XORing it with the 32-bit mask $M = \texttt{DestAddr} \oplus \texttt{AttAddr}$ to obtain a block C_2'.
3. **Repeat**:
   – a. Modify block C_2', now in the last 32 bits, by setting these bits to a random 32-bit value R. Let C_2'' denote the modified block.
   – b. Prepare a modified datagram that is identical to the one captured in step 1, except that block C_2 of the encrypted portion is replaced with C_2''. Inject this modified datagram into the network.
   Until a datagram is received by the attacker at `AttAddr`.

To see why this phase might work, notice that each injected datagram now has `AttAddr` as the destination address of the inner datagram. So when the security gateway receives the modified outer datagram and decrypts the encrypted portion, it recovers an inner datagram that will then be routed directly to the attacker's machine (we are assuming here that datagrams are not checked after IPsec processing to see if the correct IPsec policies were applied; this is the case in the Linux kernel implementation, in contradiction to [15]). The inner datagram is in unencrypted form, and its payload will be identical to that of the original inner datagram except possibly in the first 32 bits (corresponding to the randomization of the second half of C_2). These payload bits can be recovered easily using the relation $P_3 = P_3' \oplus (M||R)$ where P_3' is the third block in the received datagram, M is the address mask used in step 2 and R the random bits introduced in step 3.

Of course, because of the modifications made to block C_2 during the attack, block P_2 of the inner datagram is essentially randomized, so the header of the modified inner datagram is likely to be invalid. Block P_2 contains the time to live (TTL), protocol, header checksum and source address fields. Thus the success rate of each iteration of the attack depends on the combined probability that the TTL is sufficiently large so that the inner datagram reaches the attacker's machine, that the checksum is valid for the new header, and that the new inner source address is routable. All other fields in the header will be correct, since they lie in plaintext block P_1 which is not modified in the attack.

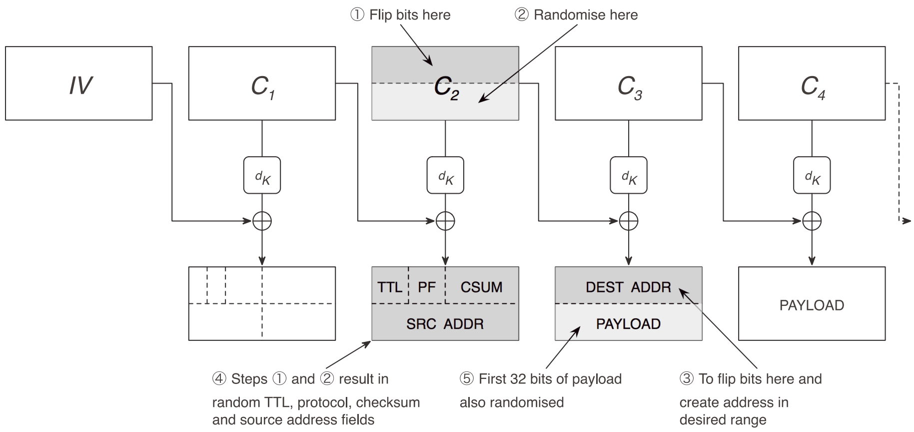

Fig. 2. Modifications to inner header fields in destination address rewriting attack, 64-bit case.

Based on our experience in implementing our other attacks, we estimate that this success probability should be roughly 2^{-17} per iteration, with the largest factor of 2^{-16} coming from the requirement for the random checksum to be a valid one. From this, it can be calculated that 2^{17} iterations of steps 3a and 3b of the attack will give a success probability of about 60%.

3.2 The Second Phase – Recovering Further Plaintext

An attacker who has conducted the first phase against an encrypted inner datagram of the form $C_0, C_1, \ldots, C_q$ does not need to repeat it in order to obtain decrypted versions of further inner datagrams. Instead, the contents of new datagrams can be recovered much more efficiently, as follows.

The attacker reuses the payload portion C_0, C_1, C_2'', C_3 of the outer datagram that was successful in the first phase, splicing onto it any $q - 6$ consecutive ciphertext blocks from the encrypted payload of the new target datagram, and finishing with the last three blocks C_{q-2}, C_{q-1}, C_q of the original target.[4] Dummy blocks can be used if necessary to ensure that a total of q blocks are present.

The attacker then uses this modified byte sequence as the encrypted payload of an outer datagram. This construction ensures that, upon decryption by the

[4] In fact, often only the last two blocks need to be preserved because the padding rarely extends over more than one block. Variable length padding of up to 255 bytes is allowed in [16]; our attacks are easily modified to handle this.

security gateway, the payload is correctly padded and is interpreted as an inner datagram with a valid header and a destination address equal to `AttAddr`. This datagram will be routed to the attacker's machine (for the same reasons that the successful datagram from the main attack was). From this datagram, a total of $64(q-6)$ bits of plaintext from the new target datagram can be recovered (the first 64 bits are obtained using a similar to trick to that used to recover P_3 in the main attack; the remaining bits appear in clear in blocks 5 up to $q-3$ of the datagram payload).

3.3 Relaxing the Address Assumption

Our main assumption that the attacker know the complete destination IP address of the inner datagram can be relaxed. It is enough that the attacker knows a significant portion of this IP address. The main idea is as follows. Instead of using a mask equal to $\texttt{DestAddr} \oplus \texttt{AttAddr}$ in step 2 of the attack, the attacker instead uses a mask which modifies that portion of the destination address known to the attacker so that it equals the corresponding portion of the address of his target machine. He then modifies the remaining bits of the destination address using a counter, and repeats the main attack for each counter value. One counter value will produce a destination address exactly matching that of the attacker; for this counter value, the attacker has the same probability as before (roughly 2^{-17}) of receiving a datagram from the gateway. After this effort, a more efficient second phase can again be used. Other variants are also possible [25].

3.4 Attack Implementation

As a proof of concept and as a precursor to our main attacks, we implemented a 128-bit version of the first phase of this attack against IP and IPsec as implemented in the Linux kernel. We found that roughly 2^{15} iterations were sufficient to produce the desired plaintext-bearing datagram, in line with a theoretical analysis of our 128-bit attack than can be found in [25]. This experiment confirmed the fact that the Linux implementation of IPsec does *not* carry out the policy checks described in Section 2.2 (otherwise the modified inner datagrams would be dropped because they would fail to match the IPsec policies used in their recovery).

4 Attacks Based on IP Options Processing

Our next set of attacks exploits the way in which IP implementations generate ICMP messages when processing incorrectly formatted options fields in IP headers. We focus on the case where the block cipher used by ESP has 64-bit blocks. We again describe the attack in the context of a pair of security gateways communicating using encryption-only ESP in tunnel mode.

We need to make some assumptions for the attack to work. As usual, we assume that the attacker is able to intercept ESP-protected datagrams and to inject modified datagrams into the network. We additionally assume that the attacker is able to monitor one of the gateways for ICMP messages not sent through the IPsec tunnel. A third-party network service provider is in a perfect position to mount this attack, for example. This would also be easily achievable if the IPsec traffic was being broadcast on a wireless network in which WEP (or an equivalent) was not in use. We will see later how this requirement can be relaxed in the 128-bit case, provided the attacker has (partial) information about inner source addresses.

4.1 The First Phase

As before, the attacker has captured an outer datagram and wishes to recover the plaintext version of the encrypted portion of its payload. Recall that the IHL field is located in the first byte of the IP header, and therefore lies in plaintext block P_1 in the sequence of blocks to be encrypted in CBC mode by ESP. The attacker modifies the contents of the IHL field of the inner datagram by flipping appropriate bits in IV, making the IHL equal a value greater than 5. When the inner datagram is subsequently processed by the IP software on the security gateway, the first word(s) of the payload (forming the contents of the second half of P_3 onwards) will be interpreted as options bytes. We randomize the values of these bytes (as seen by the security gateway) by placing a random value in the last 32 bits of C_2. Then with high probability, these bytes will be incorrectly formatted, resulting in the generation of an ICMP "parameter problem" message. The payload of this ICMP message will contain the header and a segment of the payload of the inner datagram. Thus, if it can be captured by the attacker, he can learn plaintext information from the inner datagram. However, randomizing bytes in C_2 has the additional effect of randomizing the contents of P_2 after decryption by the security gateway. So the inner datagram is likely to be dropped silently by the security gateway before any IP options processing takes place, because of an incorrect checksum value. Thus, in fact, the ICMP message will not often be generated. Moreover, the ICMP message, if generated, will be sent to the random source address now specified in P_2. This helps to ensure that the ICMP message is not sent through the IPsec tunnel between the security gateways, thus making it visible to the attacker, but also means that this address may not be routable. These problems can be overcome by iterating the attack sufficiently often and using new random bytes on each iteration. We will quantify the success rate for the Linux implementation of IP in Section 4.4 below.

This attack is illustrated in Figure 3 and formalized below.

1. Capture a target ESP-protected outer datagram from the network. Let $C_0, C_1, \ldots, C_q$ denote the encrypted portion of this datagram's payload.
2. Modify block $C_0 = IV$ in the first byte, XORing it with a mask which increases the IHL to a value greater than 5, obtaining a block C_0'.
3. **Repeat**:
   – a. Modify block C_2 in the last 32 bits, by setting these bits to a random 32-bit value R. Let C_2' denote the modified block.

– b. Prepare a modified datagram that is identical to the one captured in step 1, except that blocks C_0 and C_2 of the encrypted portion are replaced with C_0' and C_2'. Inject this modified datagram into the network.
Until an ICMP message is intercepted.

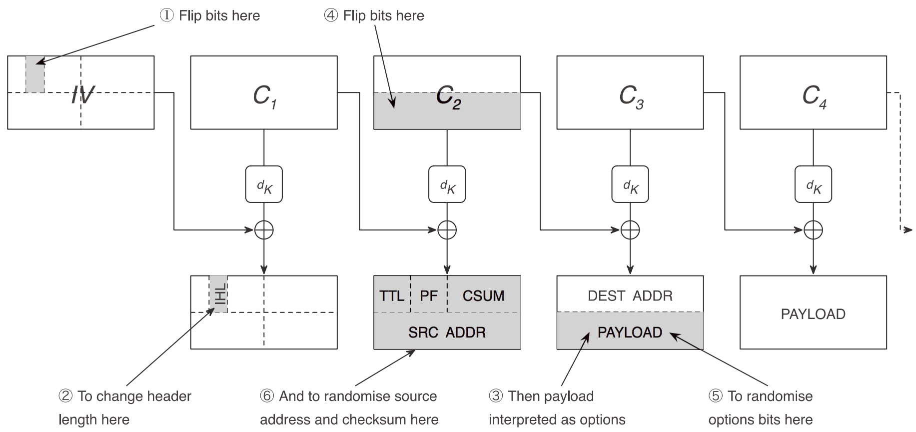

Fig. 3. Modifications to inner header fields in options processing attack, 64-bit case.

4.2 The Second Phase

Tricks similar to those introduced in Section 3.2 can be used in a second phase to speed up the recovery of all payload bytes from the remainder of the initial target datagram and further target datagrams. Once again, a successful header can be re-used and is guaranteed to always generate an ICMP message. The speed of recovery of plaintext in this second phase is limited only by the rate at which the security gateway is permitted to generate ICMP messages and by the number of payload bytes returned by ICMP.

4.3 The 128-Bit Case

A similar attack is possible when the block cipher used by ESP has 128-bit blocks. Now, however, the IHL field, Header Checksum field and Source Address field can all be manipulated by bit flipping in $C_0 = IV$. This allows the possible checksums to be tested systematically, which improves the success probability. The payload bytes which get interpreted as options bytes by the security gateway can be randomized by selecting a random value for C_2. Again, further plaintext can be recovered faster in a second phase which re-uses the successful header from the first phase. Moreover, if the attacker has some (or full) knowledge of the source address of the inner datagrams, then he can use similar ideas to those explored in Section 3.3 to direct the ICMP response to his own machine, this time by changing the source address in the inner header by manipulating the IV. This is an important variant, since it removes the most stringent requirement for our attack, namely that the attacker be able to monitor the security gateway for ICMP messages.

4.4 Attack Implementation

We have successfully carried out the two phases of our attack against IP and IPsec as implemented in the Linux kernel. We describe the main features and results of this implementation here.

Figure 4 shows the experimental set-up, with two Linux machines acting as security gateways for an ESP tunnel using either DES or AES as the encryption algorithm (the end host shown in this figure is not active during this attack). These machines are connected to a hub, as is the attack platform – this is simply to ease packet sniffing in the network. Also connected to this hub is a router, configured to act as the default router for the security gateways, thus ensuring that any ICMP messages can take at least a first hop towards their destinations.

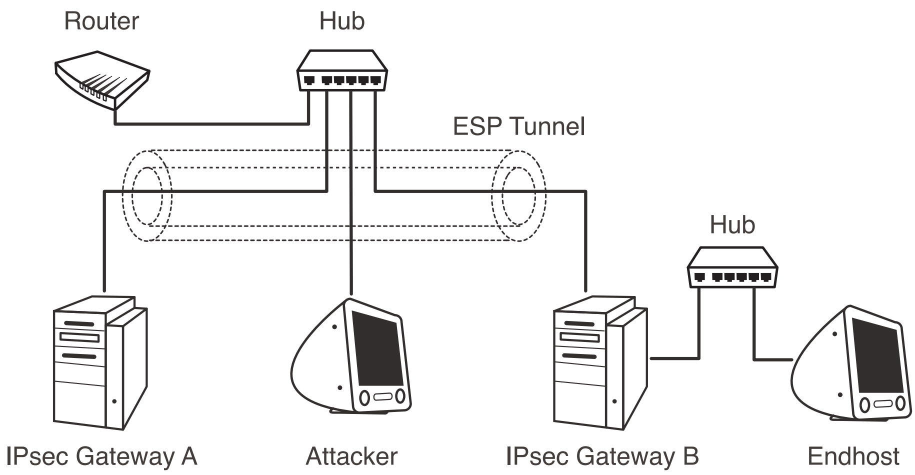

Fig. 4. Experimental set-up for attacks based on options processing and protocol field manipulation.

We used a value of 6 for the modified IHL field, so as to maximise the number of plaintext bytes returned for each injected datagram in the second phase. We observed experimentally that presenting a datagram with a random source address and random options bytes to the IP implementation in Linux results in an ICMP "parameter problem" message with probability about 0.85. Moreover, the probability that a random 16-bit value represents the correct header checksum for the modified inner datagram is roughly 2^{-16}. Thus the expected success probability of the first phase of the attack in the 64-bit case is roughly 0.85×2^{-16} per iteration. For example, then, 2^{16} iterations should give a success rate of 57%.

We performed 100 runs of the first phase of the attack. An average of 77600 iterations (taking on average 2.64 minutes with our attack client) were needed to successfully generate an ICMP message. Linux is generous in providing 524 bytes of inner datagram payload in ICMP messages. As a consequence, the first phase and each injected datagram in the second phase yields 512 bytes of plaintext data (provided the encrypted payload in the target selected for the first phase is longer than 568 bytes, including the IV and encrypted inner header). Thus the second phase can rapidly recover the complete contents of inner datagrams. Our attack client, written in C, captures multiple ESP-protected datagrams, selects the one of optimum length for the first phase, conducts the first phase, and then runs the second, faster phase on remaining datagrams. Our attack client is also written to carry out the 128-bit variant of this attack.

5 Attacks Based on Protocol Field Manipulation

Our third class of attacks exploits the way in which IP implementations generate ICMP messages when faced with unsupported upper layer protocols. We focus on the case where the block cipher used by ESP has 128-bit blocks, as this is the more efficient case. We need to make the same assumptions as in Section 4 for the attack to work.

5.1 The First Phase

Recall that the protocol field is located in the second byte of the third 32-bit word of the IP header, and therefore lies in plaintext block P_1 in the sequence of blocks to be encrypted in CBC mode by ESP. The attacker modifies the contents of the protocol field of the inner datagram by flipping appropriate bits in IV, making the field equal a value corresponding to an upper layer protocol that is not supported by the end host receiving the inner datagram. Now, when the inner datagram arrives at the end host that is its final destination, an ICMP "protocol unreachable" message will be generated. The payload of this ICMP message will contain the header and a segment of the payload of the inner datagram. Thus, if it can be captured by the attacker, then he can learn plaintext information from the inner datagram. Note that, in contrast to the attack based on options processing, the end host, not the security gateway, generates the ICMP message.

An attacker must solve two problems here. Firstly, the attacker must alter the source address of the inner datagram, so that the ICMP response will not be routed through the IPsec tunnel and so that the attacker can intercept it. Secondly, the attacker must fix the header checksum so that it contains the correct value for the modified inner header. Fortunately, in the 128-bit case, both of these requirements can be met by further manipulating only IV, and in a systematic way that leads to a very efficient attack.

Consider an attacker who modifies the protocol field by forcing a flip in bit i of the field (where $0 \le i < 8$) and who alters the inner source address by forcing a flip in bit j of the address (where $0 \le j < 32$). These bit flips can both be induced by manipulating IV. To correct the inner header checksum, the attacker XORs it with two masks in sequence (one mask for each bit flip), again by flipping bits in IV. A detailed analysis of the checksum algorithm (see [25]) shows that one of only 17 possible masks will correct each bit flip. The attacker tries these pairs of masks in decreasing order of probability. A maximum of $17^2 = 289$ iterations will be needed, with an expected number much smaller than this because of the way mask probabilities are distributed. In fact, a simple analysis shows that when $i + 8 \ne j \bmod 16$, the expected number of iterations is slightly less than 7, and smaller still when $i + 8 = j \bmod 16$. This attack can be formalized just as with the earlier attacks.

In an important variant of this attack, now requiring on average 2^{15} iterations, the attacker can additionally exploit knowledge of the inner source address to rewrite this address, thus ensuring that any ICMP response is directed to a host he controls. This removes the requirement that the attacker be able to monitor the security gateway for ICMP messages.

5.2 The Second Phase

Just as with the attack in Section 4, once the first phase is complete, a second phase which recovers the complete contents of the remainder of the initial target datagram and further target datagrams can be invoked.

5.3 The 64-Bit Case

A similar, but less efficient, attack is possible when the block cipher used by ESP has 64-bit blocks, but now the protocol field is manipulated by randomizing the last 32 bits of block C_2. The success probability is now limited by the need for a random checksum to have the correct value, and for a random protocol field to represent an unsupported protocol. In practice, it is close to 2^{-16}, because, typically, only a handful of protocols are supported. Again, further plaintext can be recovered faster in a second phase which re-uses the successful header from the first phase.

5.4 Attack Implementation

We have successfully implemented the two phases of the 128-bit attack against the Linux kernel implementation of IP and IPsec in our attack client. The experimental set-up is shown in Figure 4. In our attack, we used values $i = 0$ and $j = 6$ (many other pairs worked equally well).

According to the probability analysis sketched in Section 5.1, the expected number of iterations of the first phase with these parameters is slightly less than 7. We performed 1000 runs of the first phase of the attack. An average of 6.53

iterations (taking 1.34 seconds with our attack client) was needed to successfully generate an ICMP "protocol unreachable" message containing plaintext information. Because of the way in which Linux implements ICMP, the first phase and each injected datagram in the second phase yields about 500 bytes of plaintext data. This means that our attack client is able to recover large amounts of plaintext easily in the second phase of the attack. Overall, because of the small number of trials needed, the attack effectively takes place in real time.

6 Impact

We have presented a number of attacks and variants on encryption-only ESP in tunnel mode, as implemented in the Linux kernel. The attacks are efficient and have been demonstrated to work under realistic network conditions. Perhaps surprisingly, ESP using a 128-bit block cipher such as AES may be more vulnerable to our attacks than one using a 64-bit block cipher. The underlying reason for this is that in the 128-bit case, more fields of the inner header can be manipulated by modifying IV, without any impact on the contents of plaintext blocks. A related point is that the complexity of the attacks does not depend on the key size of the block cipher employed by ESP: triple-DES is just as vulnerable as DES.

We note that, as with [23], our work demonstrates that the open source approach does not necessarily result in secure software: an encryption-only configuration was all too easy to select, the IPsec implementation did not carry out the post-processing checks mandated in the RFCs, and we found other flaws in the implementation, particularly in the handling of padding (c.f. [29]).

Concerning the real-world impact of our attacks, we have presented evidence in the introduction that encryption-only IPsec may still be in common use. But we have performed only limited experiments against other IP/IPsec implementations. We do know that several vendors attempt to disable encryption-only. However, disabling encryption-only configurations is not enough to prevent our attacks, as they still apply to some configurations where integrity-protection is supplied by IPsec itself. As just one instance, the attacks in Sections 3 and 4 still work if AH is applied in transport mode end-to-end and is tunnelled inside ESP from gateway-to-gateway. This is because the redirection or ICMP generation take place at the gateway, before any integrity checking occurs. We note too that our attacks are not prevented if integrity protection is offered independently of IPsec by a higher-layer protocol. This contradicts the statement made in [18] that we quoted in Section 1.

7 Conclusions

We believe that the dangers of encryption-only ESP that we have highlighted here, coupled with the difficulty of ensuring that security-unaware users pick strong configurations from amongst the myriad possibilities, means that a conservative approach is called for in the IPsec standards themselves. Unfortunately, ESPv3 [18] still permits the use of encryption-only ESP, though it is no longer mandatory to support it.

The complexity of the IPsec standards has been commented on before [10]. It certainly does not help an implementation team if processing checks important to the security of one module (ESP) are contained in another document altogether (RFC 2401, [15]). It is worrying that the security of the encryption-only mode depends completely on these checks being carried out: the security dangles from a very thin thread indeed, as our attacks on the native Linux implementation make clear. It would help, then, if the reasons why those checks need to be performed were spelled out in the standard: this would give an implementor a stronger motivation for getting things right.

We hope that this work will help in persuading users to migrate away from encryption-only IPsec configurations. We also hope that it serves as an instructive example to the theoretical community of the gaps that exist between theory and practice in cryptography, and that it helps to bridge these gaps.

Acknowledgements

We would like to thank Steve Kent and David Wagner for providing important information and context. We would also like to thank the members of the NISCC Vulnerability Team for their assistance in evaluating the impact of our attacks and for helping us in working with the IPsec vendor and user communities ahead of their vulnerability advisory [24] concerning this work. Nessim Kisserli's assistance with lab-space and hardware issues was also invaluable.

References

1. R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 1827, August 1995.
2. F. Baker, "Requirements for IPv4 Routers", RFC 1812, June 1995.
3. M. Bellare, T. Kohno and C. Namprempre, "Breaking and provably repairing the SSH authenticated encryption scheme: A case study of the Encode-then-Encrypt-and-MAC paradigm." *ACM TISSEC*, Vol. 7, No. 2, May 2004, pp. 206–241.
4. M. Bellare and C. Namprempre, "Authenticated Encryption: Relations among notions and analysis of the generic composition paradigm." In *T. Okamoto (ed.), Advances in Cryptology – ASIACRYPT 2000*, LNCS Vol. 1976, Springer-Verlag, 2000, pp. 531–545.
5. M. Bellare and P. Rogaway, "Encode-then-encipher encryption: How to exploit nonces or redundancy in plaintexts for efficient cryptography." In *T. Okamoto (ed.), Advances in Cryptology – ASIACRYPT 2000*, LNCS Vol. 1976, Springer-Verlag, 2000, pp.317–330.
6. S. Bellovin, "Problem Areas for the IP Security Protocols", in *Proceedings of the Sixth Usenix Unix Security Symposium*, pp. 1–16, San Jose, CA, July 1996.
7. N. Borisov, I. Goldberg and D. Wagner, "Intercepting Mobile Communications: The Insecurity of 802.11", in *Proc. MOBICOM 2001*, ACM Press, 2001, pp. 180–189.

8. B.Canvel, A.P. Hiltgen, S. Vaudenay and M. Vuagnoux, "Password Interception in a SSL/TLS Channel," in *D. Boneh (ed.), Advances in Cryptology – CRYPTO 2003*, LNCS Vol. 2729, Springer-Verlag, 2003, pp. 583–599
9. N. Doraswamy and D. Harkins. *IPsec: the new security standard for the Internet, Intranets and Virtual Private Networks (second edition)*, Prentice Hall PTR, 2003.
10. N. Ferguson and B. Schneier, "A cryptographic evaluation of IPsec." Unpublished manuscrip available from `http://www.schneier.com/paper-ipsec.html`.
11. S. Frankel, R. Glenn and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec", RFC 3602, Sept. 2003.
12. S. Frankel, K. Kent, R. Lewkowski, A.D. Orebaugh, R.W. Ritchey and S.R. Sharma, "Guide to IPsec VPNs", NIST Special Publication 800-77 (Draft), January 2005.
13. D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, Nov. 1998.
14. J. Katz and M. Yung, "Unforgeable encryption and chosen ciphertext secure modes of operation." In *B. Schneier (ed.), FSE 2000*, LNCS Vol. 1978, Springer-Verlag 2001, pp. 284–299.
15. S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, Nov. 1998.
16. S. Kent and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, Nov. 1998.
17. S. Kent and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301 (obsoletes RFC 2401), Dec. 2005.
18. S. Kent, "IP Encapsulating Security Payload (ESP)", RFC 4303 (obsoletes RFC 2406), Dec. 2005.
19. H. Krawczyk, "The Order of Encryption and Authentication for Protecting Communications (Or: How Secure Is SSL?)", in *J. Kilian (ed.), Advances in Cryptology – CRYPTO 2001*, LNCS Vol. 2139, Springer-Verlag 2001, pp. 310–331.
20. Internet Protocol, RFC 791, Sept. 1981.
21. C. Madson and N. Doraswamy, "The ESP DES-CBC Cipher Algorithm With Explicit IV", RFC 2405, Nov. 1998.
22. C.B. McCubbin, A.A. Selcuk and D. Sidhu, "Initialization vector attacks on the IPsec protocol suite." In *WETICE 2000*, IEEE Computer Society, pp. 171–175.
23. P.Q. Nguyen, "Can we trust cryptographic software? Cryptographic flaws in GNU Privacy Guard v1.2.3", in *C. Cachin (ed.), Advances in Cryptology – EUROCRYPT 2004*, LNCS Vol. 3027, Springer-Verlag 2004, pp. 555–570.
24. NISCC Vulnerability Advisory IPSEC - 004033, 9th May 2005. Available from `http://www.niscc.gov.uk/niscc/docs/al-20050509-00386.html?lang=en`.
25. K.G. Paterson and A.K.L. Yau, "Cryptography in Theory and Practice: The Case of Encryption in IPsec." Extended version of this paper available from `http://eprint.iacr.org/2005/416`.
26. R. Pereira and R. Adams, "The ESP CBC-Mode Cipher Algorithms", RFC 2451, Nov. 1998.
27. J. Postel, "Internet Control Message Protocol", RFC 792, Sept. 1981.
28. S. Stubblebine and V. Gligor, "On Message Integrity in Cryptographic Protocols", in *IEEE Security and Privacy*, May 1992, pp. 85–104.
29. S. Vaudenay, "Security flaws induced by CBC padding – applications to SSL, IPSEC, WTLS...", in *L.R. Knudsen (ed.), Advances in Cryptology – EUROCRYPT 2002*, LNCS Vol. 2332, Springer-Verlag 2002, pp. 534–545.
30. T. Yu, S. Hartman and K. Raeburn, "The perils of unauthenticated encryption: Kerberos version 4", in *Proc. NDSS 2004*, The Internet Society, 2004.