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. This paper studies 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. Our focus is on IPsec, an important and widely-used suite of protocols providing security at the IP layer of network communications. Despite well-known results in theoretical cryptography highlighting the vulnerabilities of unauthenticated encryption, the IPsec standards currently mandate its support. We present evidence that such "encryption-only" configurations are in fact still often selected by users 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. Finally in this paper, we reflect on the reasons why this unsatisfactory situation persists, and make some recommendations for the future development of IPsec and cryptographic software in general.

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]. That community is concerned with well-equipped attackers who have access to decryption oracles and who merely need to distinguish the encryption of one ciphertext from another to be considered successful. This conservative approach is laudable and is now finding its way into standards and specifications. 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 an inherent resistance to change within user communities, unless there is clear and easily-absorbed evidence that such change is imperative. This situation is worsened by the gulf in understanding between the communities of theoreticians and end-users: results that are folklore in the former community are often only vaguely understood by the latter. This is in spite of several high-profile examples where the lack of strong integrity checks is known to lead to attacks [30, 6, 8, 33], or where inappropriate use of integrity mechanisms still leaves systems vulnerable [3, 9]. There are also many examples outside the area of authenticated encryption where attacks are made possible by poor implementation or the failure to use cryptography appropriately – see [7, 22, 25] for examples.

* The work described in this paper was partly supported by the European Commission under contract IST-2002-507932 (ECRYPT).

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

But the lack of understanding can go both ways: theoreticians sometimes do not understand the everyday problems faced by implementors and users, and the pragmatic approach to security that this produces. 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 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. Users may sometimes disregard attacks if they are attacks on-paper, rather than being fully demonstrated attacks that work in practice against deployed systems. So one must be careful not to over-estimate the impact that such "attacks in theory" have on the long-term behaviour of developers and users.

In this paper, our focus is on the use of integrity protection and encryption in IPsec, an important and widely-used suite of protocols providing security at the IP layer of network communications. We provide a short introduction to IPsec in Section 2. Bellovin [6] was the first to point out how 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 [23]) 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 of ESP used in current deployments [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 we are not saying that the theoreticians are correct and the practitioners are wrong here – we are merely observing that the gap exists. Indeed, the developers of [16] had good reasons involving backward-compatibility and performance for mandating support for an encryption-only mode.[1] Perhaps if Bellovin's attacks had been more damaging, the IPsec community would have taken a different path and mandated the use of authenticated encryption in ESP.

It is our belief that the availability of the encryption-only option in IPsec has led many users into actually using it. After all, users do not generally 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. Since this is precisely what the encryption-only version of ESP appears to provide, it would be a natural choice when selecting from amongst the myriad of IPsec options. (This point is also made in [11].) As evidence for our belief, we note that we have found several on-line tutorials showing, in a command-by-command fashion, how to build IPsec VPNs that use ESP for encryption, but that appear to provide no additional integrity protection.[2] Further evidence comes from the interactions that we had with the IPsec vendor and user communities after the release of the vulnerability announcements [26, 31] describing our attacks: while some vendors were aware of Bellovin's work and had taken steps

[1] Particularly informative are IPsec mailing list discussions on this issue: `http://www.vpnc.org/ietf-ipsec/97.ipsec/msg00633.html`

[2] See for example: `http://www.netbsd.org/Documentation/network/ipsec` and `http://lartc.org/howto/lartc.ipsec.tunnel.html`. As another example, the Cisco IPsec documentation that was available at `http://www.cisco.com/univercd/cc/td/doc/product/ismg/policy/ver21/ipsec/ch01.htm` conspicuously fails to mention the dangers of encryption-only ESP, stating that "If you require data confidentiality only in your IPSec tunnel implementation, you should use ESP without authentication. By leaving off the authentication service, you gain some performance speed but lose the authentication service." This documentation has now been removed from the Cisco website.

to prevent the selection of encryption-only configurations (despite this meaning that, technically, they were no longer in compliance with [16]), others were initially much less well-informed, and at least one vendor has since issued a patch.[3]

1.1 Our Contribution

Given the above background, it should be evident that the encryption-only configuration of IPsec is still likely to be in common use, in spite of Bellovin's work [6]. The main contribution of this paper is to present new attacks against this configuration of IPsec that are as realistic and devastating as possible, with the aim of persuading users to migrate away from encryption-only IPsec configurations. 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 new version of ESP [18], which 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 here that relying on higher layers for the provision of integrity in IPsec is inherently insecure in practice as well: as we shall see, our attacks are completed before these higher layers even have an opportunity to examine the data. Some of our attacks also apply to some configurations using the IPsec protocol AH (Authentication Header) for integrity protection – for example when AH is applied end-to-end and tunnelled inside gateway-to-gateway encryption-only ESP. This configuration appears to be secure, since datagrams are now integrity protected. But integrity checking and decryption take place at different points, and this opens the door to our attacks.

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 gulf in understanding between theoreticians and users and how this leads 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.

[3] See `http://www.securityfocus.com/archive/1/407774`

1.2 Overview of Paper

In the next section, we provide sufficient background on the IPsec and IP protocols to make the paper self-contained. In Section 3, we outline a possible attack strategy based on rewriting of destination IP addresses in inner IP datagrams. This discussion is included as a prelude to our later attacks. Section 4 presents a class of attacks based on how IP handles options processing. We also report on an implementation against ESP using DES in that section. Then in Section 5, we present a third group of attacks, which are based on manipulating the IP header protocol field, and report on an implementation against ESP using AES. In Section 6, we consider the practical impact of our attacks as well as obvious countermeasures that prevent them. Our reflections on the gaps between cryptographic theory and practice in the context of IPsec can be found in Section 7.

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 [10, 13] for accessible introductions to IPsec. Implementations of IPsec exist in Microsoft Windows 2000 and XP, and in the Linux kernel from release 2.6 onwards.[4] Various other open source projects are also developing IPsec implementations. IPsec is also 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. The use of gateways means that hosts need not be IPsec-aware, but that security is provided from gateway-to-gateway rather than in an end-to-end fashion.

IPsec provides authentication and integrity protection and/or confidentiality services for network layer data 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. Block ciphers in counter mode and dedicated stream ciphers may also be used, but these are less commonly seen in practice. 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 [28] and AES [12]. We describe the use of CBC mode in ESP in more detail in the next section. ESP may also be configured to provide integrity protection through the application of a MAC algorithm.

IPsec can be configured in a wide variety of ways. One common use is in building Virtual Private Networks (VPNs), where IPsec is usually configured to use the ESP protocol in tunnel mode to provide confidentiality. ESP can also be used to provide an integrity protection service for the VPN traffic, or this service may be provided by AH.

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

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

authentication field after the encrypted portion. This contains the MAC value if ESP's optional integrity protection features are in use. The way in which ESP modifies datagrams in tunnel mode is illustrated in Figure 1.

The AH protocol also provides integrity protection (and prevention of replays), though its scope of application is different from that of the MAC in ESP. 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 IKE. We will simply assume that key establishment has taken place, either manually, or using one of the many possible methods supported by IKE, and that no rekeying takes place during the course of our attacks.

2.2 CBC Mode Encryption in ESP

The variant of CBC mode that is used by ESP in tunnel mode is described below. For more details, see [16, 21, 12, 28].

First of all, the original (inner) datagram that is to be protected is treated as a sequence of bytes. This sequence is padded with a particular pattern of bytes and then a single next header byte is appended. The padding is constructed in such a way that the total number of bytes in the processed sequence is a multiple of the number of bytes in a block of the encryption algorithm (for example, 8 for DES and 16 for AES), and so that the padding can be removed unambiguously. It is permissible for the padding to be of variable length and to extend over multiple blocks. This might aid in preventing traffic analysis. We assume throughout that the minimum amount of padding is used, though our attacks are easily modified to handle variable length padding. This padding occupies the ESP trailer field in Figure 1.

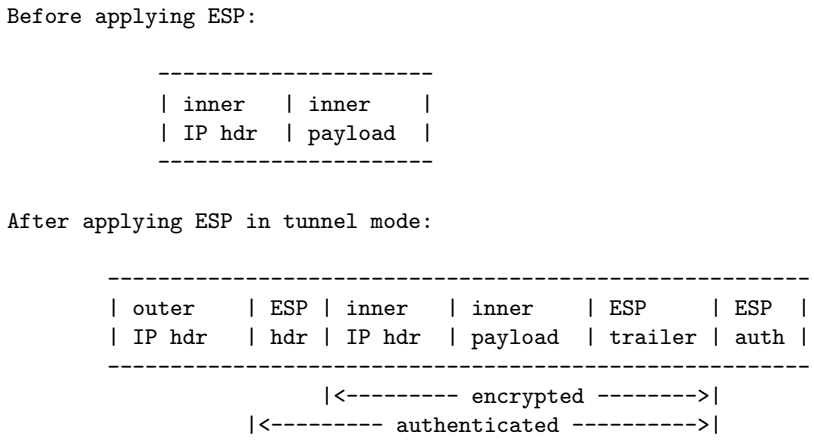

Fig. 1. ESP datagram structure according to RFC 2406, [16].

Let us assume that the byte sequence after padding consists of q blocks, each of n bits (where $n = 64$ for DES and $n = 128$ for AES, for example). 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. Then the CBC mode encryption in ESP proceeds as follows. First of all, an n-bit initialization vector, denoted IV, is selected at random. Then ciphertext blocks are generated according to the equations:

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

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 (who is also in possession of the key K), the payload of the outer datagram can be recovered using the equations:

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

Any padding and the next header byte can then be stripped off, revealing the original inner datagram. At this point, Section 5.2 of the IPsec architectural RFC [15] mandates that implementations should check that the cryptographic processing performed to recover inner datagram in fact does match that specified in local IPsec policies. Presumably, if the check fails, the packet should be dropped, though this is not made explicit in [15].[5] 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.

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 – this can be seen from the relation $P_{i-1} = C_{i-2} \oplus d_K(C_{i-1})$. On the other hand, if the modification is made in C_0 (equal to IV), then no damage to plaintext blocks will result. Note that in carrying out this attack, the attacker does not directly gain information about the contents of plaintext blocks; rather he can (to an extent) control the plaintexts seen by the legitimate decrypting party. This kind of attack is easily prevented by the appropriate use of an integrity protection mechanism, such as a MAC algorithm.

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 2.

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, indicating that the header length is 20 bytes and no additional options bytes are present. If the IHL value is greater than 5, then additional 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 and can be used to allow the header to carry additional instructions or information. 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. For example, our 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. Slightly more than half of the 256 possible values are already allocated to specific upper layer protocols. The set of values that a host might place here when generating

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

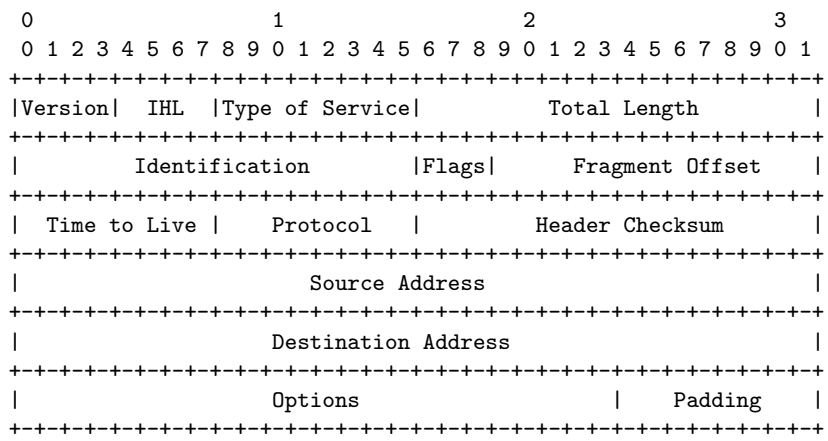

Fig. 2. Structure of IP header according to RFC 791, [20].

a datagram depends on that host's configuration and the protocols it supports. A minimal set of supported protocols would include ICMP, TCP and UDP. A typical host may support two or three more. When an IP datagram reaches its intended destination host (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 protocol field is not usually inspected by intermediate routers.

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. The Header Checksum provides a verification that the information used in processing a datagram has been transmitted correctly. If the Header Checksum fails, the datagram is discarded silently by the entity which detects the error. This checksum is not, and was not designed to be, cryptographically robust.

In order to understand our attacks, it is important to reflect on the sequence of steps taken by IP when processing a datagram. In Linux, the sequence is as follows (where we omit any discussion of fragmentation for simplicity). 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 (if this host is configured for routing). 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 packet dropped if the TTL has reached zero.

Given the above discussion, it is evident that any attack based on the processing of a particular header field must ensure that, with reasonable probability, any header processing taking place before that field is inspected does not result in the datagram being dropped. This has implications for the ways in which we can manipulate ciphertext blocks. In particular the attacker needs to ensure that the header checksum is still correct after making any changes to the header.

2.5 ICMP

Finally in this background section, we provide a brief overview of 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 [29], 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 [29], 8 bytes of payload should be included; this is, for example, how ICMP seems to be implemented in Microsoft Windows 2000. 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. Reference [2] also specifies that ICMP implementations should be able to limit the rate at which ICMP messages will be generated, as an aid to preventing to Denial-of-Service attacks.

We have discussed above two circumstances under which ICMP messages will be generated: due to incorrectly formatted options bytes, and due to unsupported upper-layer protocols.

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, and sketch the 128-bit version later. 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: we assume 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

The first phase of the attack is based on manipulating the Destination Address field in the inner datagram. Recall that this 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:

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 $M =$ `DestAddr` $\oplus$ `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. Figure 3 illustrates how the attack modifies the various inner header fields.

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. The further datagrams need not even be destined for or originate from the same pair of hosts. 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.[6] Padding with 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 (illustrated in Figure 4) ensures that, upon decryption by the 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).

It is easy to see how this second phase could be automated to give an efficient method for recovering the complete contents of multiple captured datagrams, with many plaintext bits being obtained for each datagram injected into the network. Thus the cost of the main attack can be amortized over many datagrams. It illustrates an idea that we re-use several times in the attacks to follow.

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. This might be more realistic. For example, it may be that the host is on a network whose

[6] 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. Moreover, even though [16] indicates that padding should be checked, it does not explicitly specify what action should be taken in the event of a padding error. The Linux kernel implementation simply ignores such errors. In any event, our attacks preserve proper padding and so the manner in which padding errors are handled is irrelevant here.

network prefix is known to the attacker, perhaps because it is the same as the prefix of the security gateway, or because it is a widely-used private address prefix.

The main idea is as follows. Instead of using a mask equal to `DestAddr` $\oplus$ `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.

As an illustrative example, suppose the attacker knows that the inner datagram has as its destination a host on the class C private network `192.168.0.0`. Then the attacker will be successful after on average 2^7 choices of the 2^8 possible counter values, and hence after the injection of roughly 2^{24} datagrams. After this effort, a more efficient second phase can once again be used.

A further optimization is possible if the attacker can eavesdrop all traffic on a network that is equal in class to that portion of the address `DestAddr` that he knows: now there is no need to use a counter, and the attack works exactly as in Section 3.1. This attack might be realized if, for example, the attacker controls a router for a network of the appropriate size. An obvious modification of the attack also works if the attacker knows nothing about the address `DestAddr` but is able to eavesdrop on all traffic emanating from the security gateway. In this situation, it might be useful for the attacker to arrange for a particular bit pattern to appear in the IP headers so that the datagrams can subsequently be recognized. This can be done, for example, by modifying the identification and fragmentation fields of the inner header through manipulation of the IV.

3.4 The 128-Bit Case

We briefly consider the destination address rewriting attack in the case where the block cipher used by ESP has 128-bit blocks. With notation as before, now the first four 32-bit words of the IP header are contained in P_1 and the destination address is contained in the first 32 bits of P_2. Thus the destination address can only be altered either at the cost of randomizing the entire contents of P_1 (in which case the header of the modified inner datagram is far less likely to be accepted as valid by the security gateway) or by selecting a random C_2', in which case the attacker has no control over this destination address. In the latter situation, the attack is successful if the attacker can intercept all traffic from the security gateway: now the attacker can create a valid header for the inner datagram by flipping the bits of C_0 (i.e. the IV) in positions corresponding to the location of the header checksum in P_1. This can be done in a systematic fashion so that the average effort required to produce a valid inner datagram is 2^{15} iterations. After this success, a faster second phase can be employed, using similar ideas to those that worked in the 64-bit case.

3.5 Attack Implementation

As a proof of concept and as a precursor to our main attacks, we implemented the 128-bit version of the first phase of this attack against IP and IPsec as implemented in the Linux kernel. We indeed found that roughly 2^{15} iterations were sufficient to produce the desired plaintext-bearing datagram. 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

We sketch the ideas behind 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, then 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. Since P_2 contains the TTL, protocol, header checksum and source address fields, 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 the inner datagram. 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. However, by iterating the attack sufficiently often and using new random bytes on each iteration, the attacker achieve a reasonable overall success probability. We will quantify the success rate for the Linux implementation of IP in Section 4.4 below.

The attack may fail for other reasons too: the security gateways may be programmed to drop ICMP traffic, or policy checks may mean that all traffic not directed to/from the ESP tunnel between the gateways is dropped.

This attack is illustrated in Figure 5 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.

4.2 The Second Phase

Recall that the first phase above captures an ICMP message containing the header and a segment of the payload of the modified inner datagram. The amount of payload recovered depends on the variant of ICMP implemented on the gateway, and is possibly as small as 64 bits. Tricks similar to those introduced in Section 3.2 can be used to speed up the recovery of the remaining payload bytes from the remainder of the initial target datagram and further target datagrams in a second phase. Once again, a successful header can be re-used and is guaranteed to always generate an ICMP message. The attacker can insert a number of target blocks into the ESP-encrypted payload after the header each time, this number depending on the amount of payload returned by ICMP. The attacker must again take care to complete the encrypted portion so that the payload is correctly padded. 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.

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. This would make the attacks far easier to mount in practice.

A complication arises when the amount of inner datagram payload returned by the ICMP message is smaller than the block size of the ESP encryption algorithm. This means that less than a full block of payload is returned in each ICMP message. This situation occurs when, for example, ESP uses AES and ICMP returns only 64 bits of payload. In this instance, it is necessary to align the end of the inner datagram header with a block boundary by setting the IHL field appropriately, and only a fraction of the bytes of each block can be recovered in the second, fast phase. It is possible to recover the remaining bytes of each block in a variant of the first phase, by placing the target block so as to ensure that it is interpreted as being part of the options field in the inner datagram. We omit the details of this variant.

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 6 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). The tunnel was initiated using scripting to set-up the required IPsec policies and manual keying. 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.

We used a value of 6 for the modified IHL field, so as to maximise the amount of plaintext bytes returned for each injected datagram in the second phase. The corresponding mask value for the 4-bit field is `0011`.

We have 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 per iteration of the first phase of the attack in the 64-bit case is roughly 0.85×2^{-16}, meaning that the success probability after t iterations should be:

$$1 - (1 - 0.85 \times 2^{-16})^t \tag{1}$$

From this, a theoretical curve can be drawn, showing success probability against the number of iterations made; it can also be calculated that 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. Over these 100 runs, the minimum number of iterations needed was found to be 1884 and the maximum 292000 (the latter taking about 10 minutes). Figure 7 shows the percentage of runs needing a particular number of iterations for success, both as predicted by (1) and as observed in our experiment. It can be seen that theory and experiment are in excellent agreement.

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 large amounts of plaintext. For example, a typical inner datagram carrying 1500 bytes of payload can be recovered using just 3 injected datagrams in the second phase.

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. The speed at which our software can break this configuration of IPsec is restricted only by ICMP rate limiting at the security gateway. 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: the attacker is able to intercept ESP-protected datagrams, to inject modified datagrams, and to monitor one of the gateways for ICMP messages not sent through the IPsec tunnel. This last requirement can again be relaxed, at the cost of a less efficient attack.

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. Usually, it is sufficient to flip only the most significant bit of the protocol field to achieve this. 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, otherwise the ICMP response will be routed through the IPsec tunnel back to the original source of the inner datagram. Secondly, the attacker must fix the header checksum so that it contains the correct value for the modified inner header – otherwise, the inner datagram

will simply be dropped by the security gateway and never reach the end host. Fortunately, in the 128-bit case, both of these requirements can be met by further manipulating IV.

In fact, a careful attacker who manipulates only a single bit in the protocol field and a single bit in the source address field can do much better than simply trying all possible header checksum values in an attempt to correct that checksum (an attack that would require 2^{15} iterations on average). We sketch next how this can be achieved (see also Figure 8).

Analysis of the IP header checksum algorithm reveals that flipping a single bit in the header has the effect of XORing the 16-bit checksum value with one of only 17 possible masks, these masks having a geometric probability distribution. For example, Table 1 shows the table T_{15} of masks and their probabilities for the case where the bit flip in the header is in the most significant (rightmost) position in one of the 16-bit words input to the checksum algorithm[7]. Similar tables of masks with identical probabilities can be computed for all other positions: the table of masks T_j for position j ($0 \leq j < 16$) can be obtained by performing a 16-bit left-rotate through $15 - j$ positions on the masks in Table 1.

Mask	Probability
0000000000000001	1/2
0000000000000011	1/4
0000000000000111	1/8
⋮	⋮
1111111111111111	2^{-16}
1111111111111110	2^{-16}

Table 1. Table T_{15} of masks and their probabilities for bit position 15.

Consider then an attacker who modifies the protocol field by effecting a flip in bit i (where $0 \leq i < 8$) and who alters the inner source address by forcing a flip in bit j (where $0 \leq j < 32$). To correct the inner header checksum, the attacker effectively needs to XOR it with two masks in sequence, one from the table T_{i+8} (because the protocol field is the most significant byte position in the 16-bit checksum calculation) and one from the table $T_{j \bmod 16}$. Of course, the attacker does not know exactly which pair of masks will work. So the attacker's strategy is to inject a sequence of datagrams into the network, changing IV in the positions corresponding to the header checksum at each iteration, using as the mask at each iteration a distinct XOR of masks from the tables T_{i+8} and $T_{j \bmod 16}$. Clearly, the attacker should use these masks in decreasing order of probability. The probability of any mask $M_1 \oplus M_2$ can be obtained by multiplying the individual probabilities of the masks M_1 and M_2 (taking care that when $i + 8 = j \bmod 16$, the same mask can arise in more than one way). A maximum of $17^2 = 289$ iterations will be needed, with an expected number much smaller than this because of the geometrical nature of the probability distribution of the individual masks. In fact, a simple analysis shows that when $i+8 \neq j \bmod 16$, the expected number 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 contents of the remainder of the initial target datagram and further target datagrams

[7] Note that the checksum algorithm itself regards the rightmost bit, bit 15, as being the least significant bit for the purposes of performing the checksum arithmetic.

can be invoked. The main idea is to replace blocks $C_3, C_4, \ldots$ of the ESP-encrypted portion of the successful datagram from the first phase with target ciphertext blocks.

Because of the relationship between header fields and plaintext block boundaries, this second phase does require ICMP to carry at least 28 bytes of payload of the inner datagram in order to be completely successful. This condition is certainly met if the end host is running Linux. If ICMP carries p bytes of payload, where p is between 13 and 28, then $p - 12$ bytes of *every* plaintext block can be recovered efficiently; if p is less than 13, then no further bytes can be recovered in this second phase. In this case, a variant of the first phase (in which the target ciphertext block replaces C_2) can be used to recover $p+4$ bytes of plaintext data from each block, using on average 2^{15} iterations per block.

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. This block then decrypts to give P_2 containing a TTL field, protocol field, header checksum and source address that are effectively random, while the destination address, occupying the first 32 bits of P_3, is unscathed. The success probability of the attack is now limited by the need for the random checksum to have the correct value, and for the random protocol field to represent an unsupported protocol at the end host. Thus the success probability depends on the number of unsupported protocols at the end host. In practice, the success probability per iteration 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. We omit the details.

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 6. 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. Over these 1000 runs, the minimum number of iterations needed was found to be 1 and the maximum 80. Our attack client leaves a short interval between iterations in order to accurately detect ICMP messages. The variants requiring 2^{15} iterations can attempt many more iterations per second, as with the attack in Section 4.4. Figure 9 shows the percentage of runs needing a particular number of iterations for success, both as predicted by the analysis sketched in Section 5.1 and as observed in our experiment. Theory and experiment again agree well.

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 and Countermeasures

6.1 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. We have also explained how the attacks can be made even

more effective when information concerning the source address of inner datagrams is available to the attacker. 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 have focussed on Linux in our implementation of the attacks. The open-source nature of Linux has enabled us to examine in detail its implementation of the IP, ICMP and IPsec protocols and assess the feasibility of various attack ideas before committing to the laborious process of code development. The ease with which simple IPsec policies could be specified (and modified) and network diagnostics performed also simplified our task. We note that, as with [25], 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. [32]). Of course, similar issues may arise in closed-source implementations and the open-source community does have a good track record in quick release of patches.

We have performed only limited experiments against other IP/IPsec implementations, and more work is needed to assess their vulnerability to our attacks. As we've seen, key factors in determining this include: the size of payloads in ICMP messages generated by the IP stack; the way in which the IP stack handles errors during IP options processing; the number of higher layer protocols supported by the OS; and whether or not IPsec performs policy checks after cryptographic processing. We have been informed that, like Linux, the Solaris implementation of IPsec does not perform these checks [24]. With our existing attack client in place, the testing of further implementations should be straightforward. Whether or not a given deployment of IPsec is vulnerable to our attacks depends on the aforementioned key factors, as well as further issues such as IPsec policy, firewall rulesets and so on.

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 this is not an easy area in which to gather accurate figures. We do know that several vendors attempt to disable encryption-only. For example, this is the case with Openswan, whose developers present a code fragment showing that encryption must be combined with integrity protection in their implementation [27]. 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. Thus the source code fragment presented at [27], by itself, may not be sufficient to prevent our attacks, as it appears only to force users to select *some* combination of encryption and integrity protection, whereas only *particular* combinations will definitely prevent our attacks.

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.

6.2 Countermeasures

We report here on the immediate countermeasures that can be taken to ensure systems are not vulnerable to our attacks.

Perhaps the simplest way to guarantee immunity from these attacks is to configure ESP so as to use both confidentiality *and* integrity protection (provided of course that the IPsec implementation supports this). The additional use of integrity protection within ESP foils our attacks, since the modified datagrams will be immediately rejected with overwhelming probability.

Another option is use the AH protocol alongside ESP to provide integrity protection. However, as we've highlighted above, this must be done carefully: the configuration where AH in transport mode is applied end-to-end and tunnelled inside ESP is still vulnerable.

Adding some form of integrity protection may not always be desirable or possible in already fielded systems. For example, it may have an unacceptable impact on throughput, or require extensive modification to deployed code. In this situation, using an RFC-compliant implementation that properly implements post-processing checks would be a reasonable approach. An alternative is to remove the error reporting, by restricting the generation of ICMP messages by IP software or by filtering these messages at a firewall or security gateway. However, we regard both of these approaches as being less preferable than using some form of integrity protection.

7 Conclusions

IPsec is a complicated set of protocols, and gaining true security from IPsec lies in the details of configuration, policy, algorithm selection and key management. Despite many years of development, IPsec standards and deployments still allow insecure configurations to be selected. Our work shows just how weak one such configuration can be.

Our attacks yield several lessons for the IPsec community (including theoreticians, authors of standards, implementors and users).

Our view is that configurations known to be weak, either in theory or in practice, should be outlawed in the IPsec standards as much as possible. Our view is that the gap between standards and users that is created by allowing such configurations is too large to bridge, no matter how many warnings are issued in the RFCs. Unfortunately, ESPv3 [18] still permits the use of encryption-only ESP. Naturally, the need for backward-compatibility and the potential for impact on performance may mean that eliminating this mode is unattractive. It can also be argued that users should be permitted to make their own choices about how integrity protection is supplied, in which case it might help if implementors passed on RFC warnings to users. 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 standard itself: our experience is that implementors and users can't be expected to get it right often enough. We note that ESPv3 has support for "combined mode algorithms". These combine encryption and integrity protection into a single authenticated encryption transform. We believe that their inclusion represents a progressive step in the development of IPsec.

The complexity of the IPsec standards has been commented on extensively before [11]. It certainly does not help an implementation team if processing checks important to the security of one module (ESP) are actually contained in another document altogether (RFC 2401, [15]), though it is understandable why, in the case of IPsec, these checks were placed in an architectural document. It is worrying that the security of the encryption-only mode depends completely on these simple 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 note that the quality of the IPsec RFCs is improving in this area: the relevant checks are given in much more explicit detail in the new IPsec architecture document [17]. This should help close the gap between the standards and their implementations.

More generally, we hope that our work can help to bridge the gap between the theory and practice of cryptography. We have given what we believe to be a compelling demonstration of the weaknesses of encryption-only ESP. We hope that it will be of interest to everybody in the cryptographic community, interpreted in its broadest sense.

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 [26] 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 Transactions on Information and System Security (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. D.Bleichenbacher, "Chosen Ciphertext Attacks Against Protocols Based on the RSA Encryption Standard PKCS #1", in *H. Krawczyk (ed.), Advances in Cryptology – CRYPTO 1998*, LNCS Vol. 1462, Springer-Verlag, 1998, pp. 1–12.
8. 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.
9. 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
10. N. Doraswamy and D. Harkins. *IPsec: the new security standard for the Internet, Intranets and Virtual Private Networks (second edition)*, Prentice Hall PTR, 2003.
11. N. Ferguson and B. Schneier, "A cryptographic evaluation of IPsec." Unpublished manuscript, Feb. 1999. Available from `http://www.schneier.com/paper-ipsec.html`.
12. S. Frankel, R. Glenn and S. Kelly, "The AES-CBC Cipher Algorithm and Its Use with IPsec", RFC 3602, Sept. 2003.
13. 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.
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. J. Manger, "A chosen ciphertext attack on RSA Optimal Asymmetric Encryption Padding (OAEP) as standardized in PKCS #1 v2.0," in *J. Kilian (ed.), Advances in Cryptology – CRYPTO 2001*, LNCS Vol. 2139, Springer-Verlag 2001, pp. 230-238.
23. C.B. McCubbin, A.A. Selcuk and D. Sidhu, "Initialization vector attacks on the IPsec protocol suite." In *9th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2000)*, IEEE Computer Society, 2000, pp. 171–175.

24. D. McDonald, personal communication, 6th March 2006.
25. 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.
26. NISCC Vulnerability Advisory IPSEC - 004033, 9th May 2005. Available from `http://www.niscc.gov.uk/niscc/docs/al-20050509-00386.html?lang=en`.
27. OpenSwan annoucement "No version of Openswan is vulnerable to NISCC Vulnerability Advisory IPSEC 004033", May 13 2005. Available from `http://www.openswan.org/niscc`.
28. R. Pereira and R. Adams, "The ESP CBC-Mode Cipher Algorithms", RFC 2451, Nov. 1998.
29. J. Postel, "Internet Control Message Protocol", RFC 792, Sept. 1981.
30. S. Stubblebine and V. Gligor, "On Message Integrity in Cryptographic Protocols", in *Proc. IEEE Symposium on Research in Security and Privacy*, May 1992, pp. 85–104.
31. US-CERT vulnerability Note VU#302220, 9th May 2005. Available from `http://www.kb.cert.org/vuls/id/302220`.
32. 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.
33. T. Yu, S. Hartman and K. Raeburn, "The perils of unauthenticated encryption: Kerberos version 4", in *Proc. NDSS 2004*, The Internet Society, 2004.

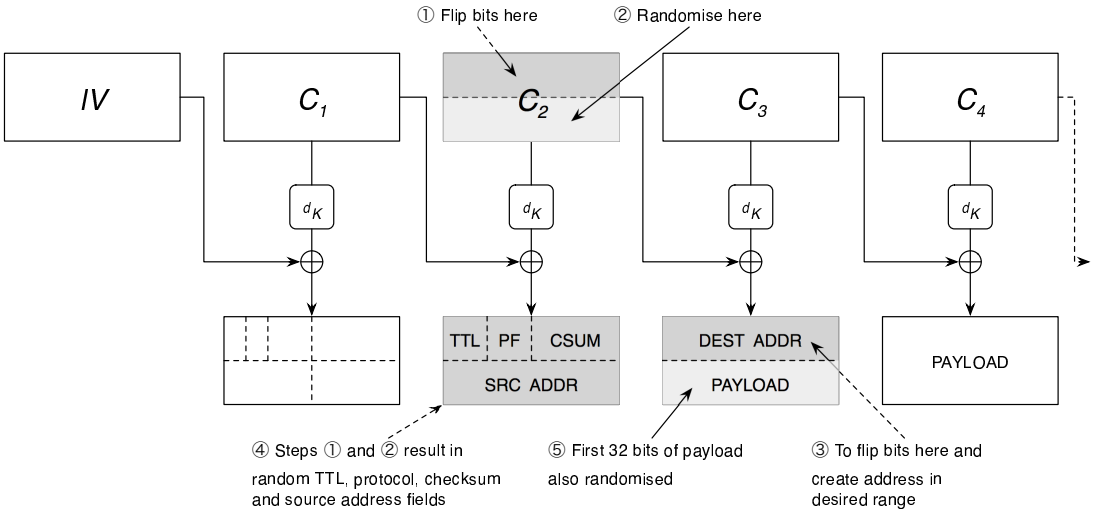

Fig. 3. Modifications to inner header fields in destination address rewriting attack, 64-bit case.

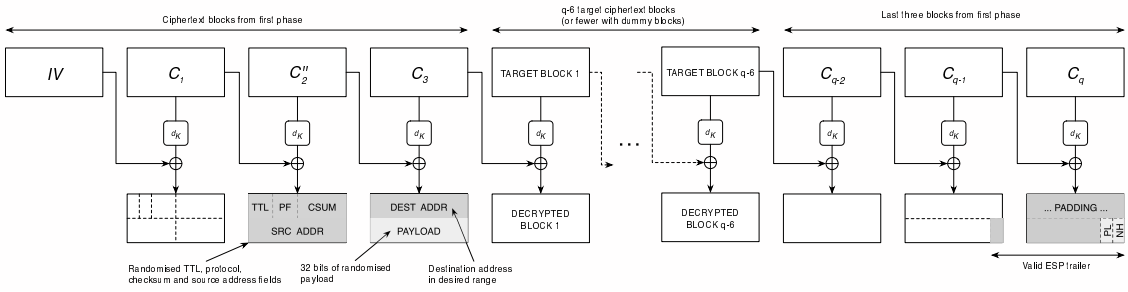

Fig. 4. Re-using a successful header from the first phase.

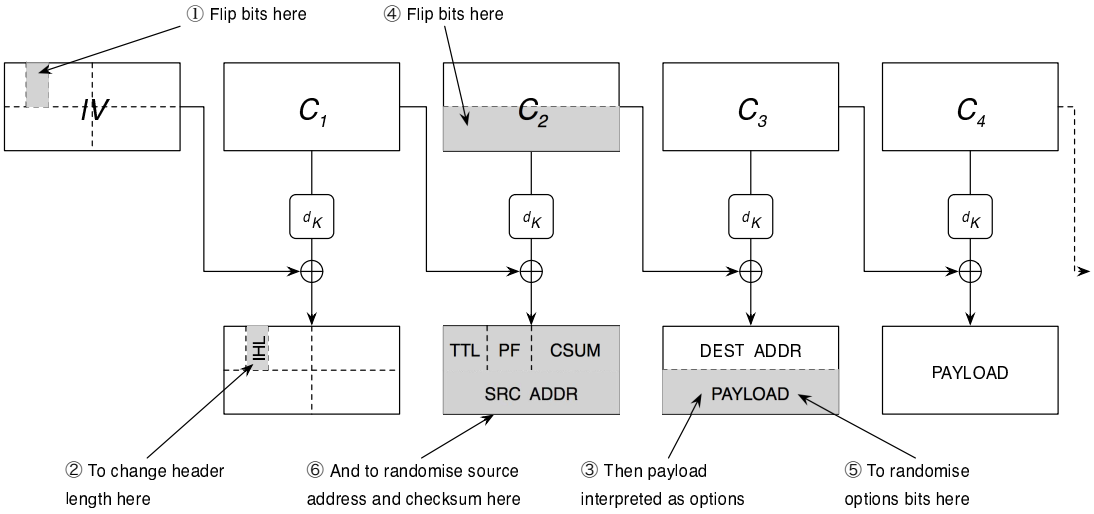

Fig. 5. Modifications to inner header fields in options processing attack, 64-bit case.

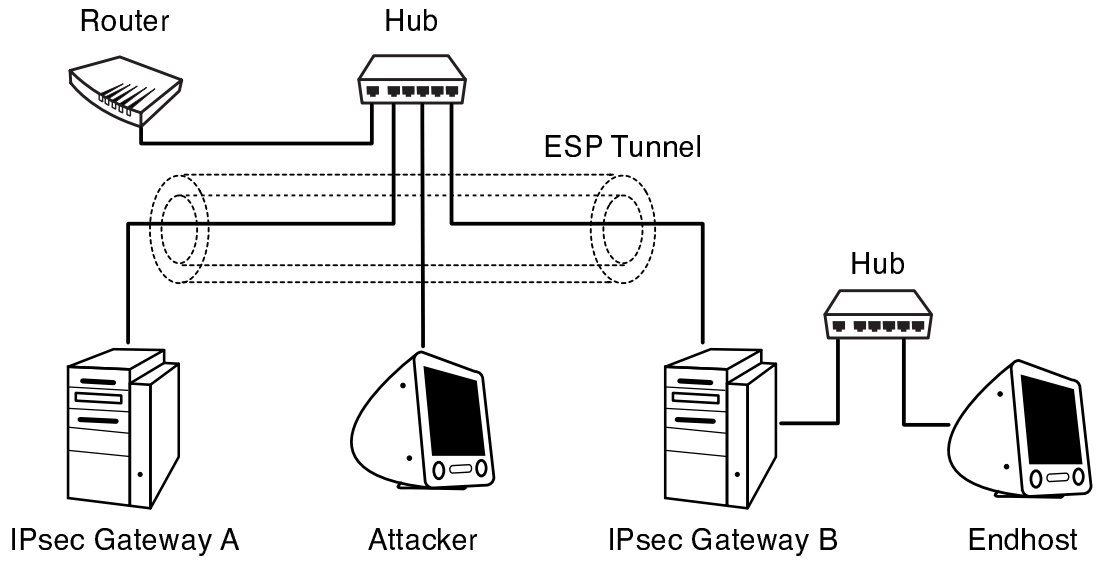

Fig. 6. Experimental set-up for attacks based on options processing and protocol field manipulation.

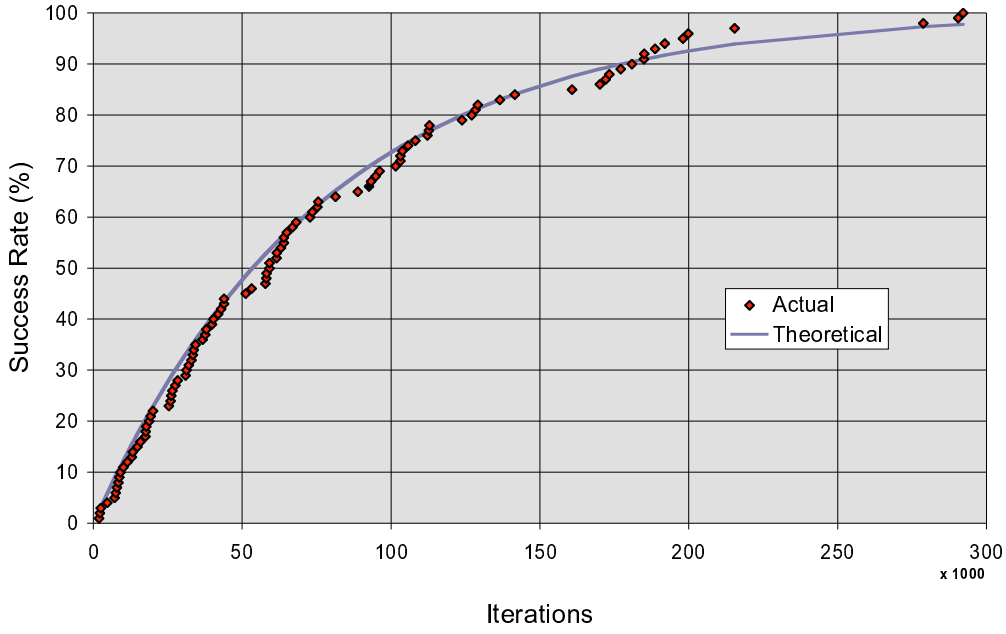

Fig. 7. Performance of 64-bit attack based on options processing.

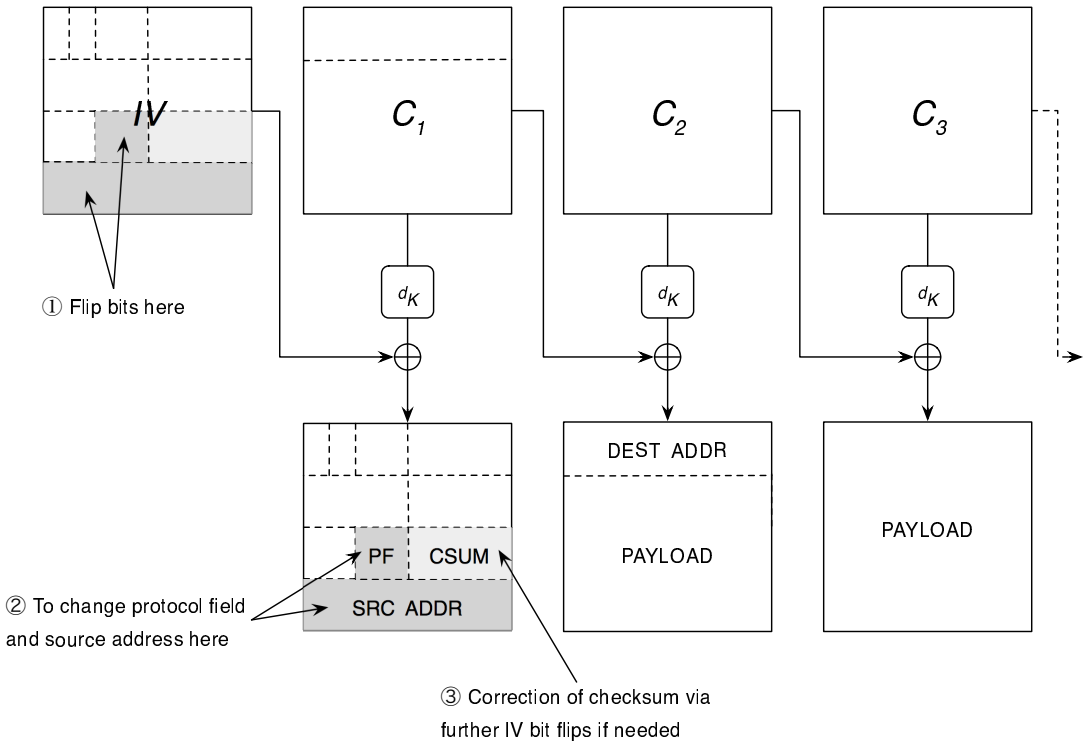

Fig. 8. Modifications to inner header fields in protocol field attack, 128-bit case.

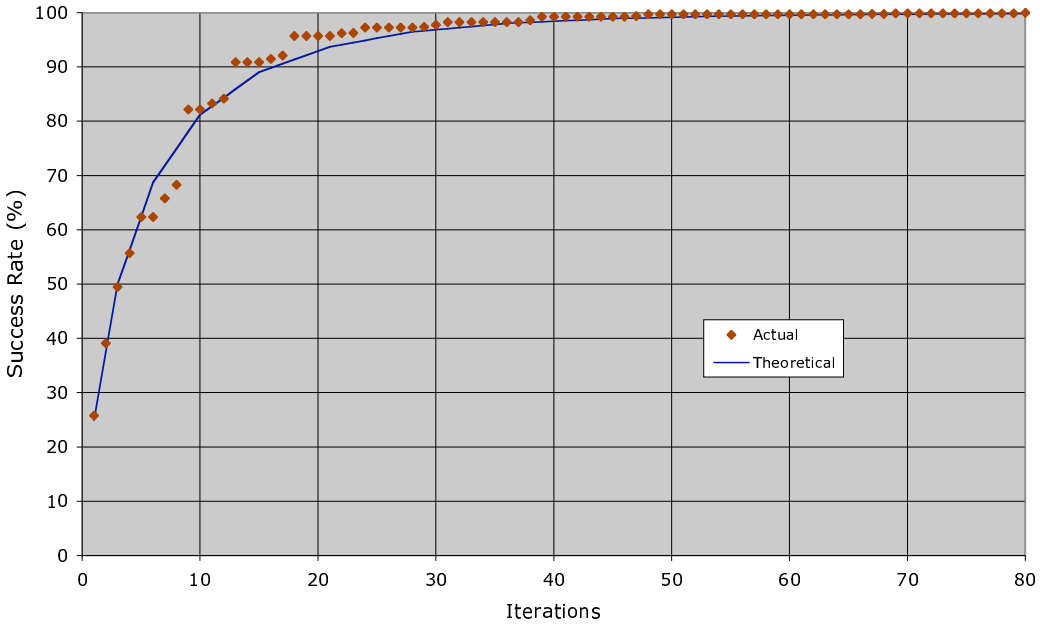

Fig. 9. Performance of 128-bit attack based on manipulation of protocol field.