Independent Submission
Request for Comments: 6479
Category: Informational
ISSN: 2070-1721
January 2012

X. Zhang
T. Tsou
Futurewei Technologies

IPsec Anti-Replay Algorithm without Bit Shifting

Abstract

This document presents an alternate method to do the anti-replay checks and updates for IP Authentication Header (AH) and Encapsulating Security Protocol (ESP). The method defined in this document obviates the need for bit shifting and it reduces the number of times an anti-replay window is adjusted.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

"IP Authentication Header" [RFC4302] and "IP Encapsulating Security Payload (ESP)" [RFC4303] define an anti-replay service that employs a sliding window mechanism. The mechanism, when enabled by a receiver, uses an anti-replay window of size W. This window limits how far out of order a packet can be, relative to the packet with the highest sequence number that has been authenticated so far. The window can be represented by a range [WB, WT], where WB=WT-W+1. The whole anti-replay window can be thought of as a string of bits. The value of each bit indicates whether or not a packet with that sequence number has been received and authenticated, so that the replay packet can be detected and rejected. If the packet is received, the receiver gets the sequence number S in the packet. If S is inside window (S<=WT and S>=WB), then the receiver checks the corresponding bit (location is S-WB) in the window to see if this S has already been seen. If S<WB, the packet is dropped. If S>WT and is validated, the window is advanced by (S-WT) bits. The new window becomes [WB+S-WT, S]. The new bits in this new window are set to indicate that no packets with those sequence numbers have been received. The typical implementation (for example, the integrity algorithm [RFC4302]) is done by shifting (S-WT) bits. In normal cases, the packets arrive in order, which results in continuous updates and bit-shifting operations.

[RFC4302] and [RFC4303] define minimum window sizes of 32 and 64. But no requirement is established for minimum or recommended window sizes beyond 64 packets. The window size needs to be based on reasonable expectations for packet re-ordering. For a high-end, multi-core network processor with multiple crypto cores, a window size bigger than 64 or 128 is needed due to the varied IPsec processing latency caused by different cores. In such a case, the window sliding is tremendously costly even with hardware acceleration to do the bit shifting. This document describes an alternate method to avoid bit shifting. It only discusses the anti-replay processing at the receiving side. The processing is always safe and has no interoperability effects. Even with a window size bigger than the usual 32- or 64-bit window, no interoperability issues are caused.

Any node employing practices that potentially cause reordering beyond the usual 32- or 64-bit window may lead to interoperability or performance problems, however. For instance, if either the sending node or routers along the path cause significant re-ordering, this can lead to inability of the receiving IPsec endpoint to process the packets, as many current implementations do not support the extensions defined in this memo. Similarly, such reordering can cause significant problems for transport and upper-layer protocols, and is generally best avoided.

2. Description of the New Anti-Replay Algorithm

Here we present an easy way to update the window index only, while also reducing the number window updates. The basic idea is illustrated in the following figures. Suppose that we configure the window size W, which consists of M-1 blocks, where M is a power of two (2). Each block contains N bits, where N is also a power of two (2). It can be a byte (8 bit) or word (32 bit), or multiple words. The supported sliding window size is (M-1)*N. However, it covers up M blocks (four blocks as shown in Figure 1). All these M blocks are circulated and become a ring of blocks, each with N bits. In this way, the supported sliding window (M-1 blocks) is always a subset window of the actual window when the window slides.

Initially, the actual window is defined by a low- and high-end index [WB, WT], as illustrated in Figure 1.

```
+--------+--------+--------+--------+
|xxxxxxcc|cccccccc|cccccccc|ccccc100|
+--------+--------+--------+--------+
       ^                          ^
       |                          |
       WB                         WT
```

Figure 1: The sliding window [WB, WT] in which WT is the last validated sequence number, and the supported window size W is WT-WB+1. (x=don’t care bit, c=check bit)

If we receive a packet with the sequence number (S) greater than WT, we slide the window. But we only change the window index by adding the difference (S-WT) to both WT and WB (WB is automatically changed as the window size is fixed). So, S becomes the largest sequence number of the received packets. Figure 2 shows the case that the packet with sequence number S=WT+1 is received.

```
   +--------+--------+--------+--------+
   |xxxxxxcc|cccccccc|cccccccc|ccccc110|
   +--------+--------+--------+--------+
           ^                          ^
           |                          |
           WB                         WT
```

Figure 2: The sliding window [WB, WT] after S=WT+1

If S is in a different block from where WT is, we have to initialize all bit values in the blocks to 0 without bit shifting. If S passes several blocks, we have to initialize several blocks instead of only one block. Figure 3 shows that the sequence number already passed the block boundary. Immediately after the update, all the check bits should be 0 in the block where WT resides.

```
   +--------+--------+--------+--------+
   |ccc10000|xxxxcccc|cccccccc|cccccccc|
   +--------+--------+--------+--------+
       ^         ^
       |         |
       WT        WB
```

Figure 3: The sliding window [WB, WT] after S passes the boundary

After the update, the new window still covers the configured window. This means the configured sub-window also slides, conforming to the sliding window protocol. The actual effect is somewhat like shifting the block. In this way, the bit shifting is deemed unnecessary.

It is also easier and much faster to check the window with the sequence number because the sequence number check does not depend on the lowest index WB. Instead, it only depends on the sequence number of the received packet. If we receive a sequence number S, the bit location is the lowest several bits of the sequence number, which only depends on the block size (N). The block index is several bits before the location bits, which only depends on the window size (M).

We do not specify how many redundancy bits are needed, except that it should be a power of two (2) for computation efficiency. If the microprocessor is 32 bits, 32 might be a better choice while 64 might be better for 64-bit microprocessor. For a microprocessor with cache support, one cache line is also a good choice. It also depends on the size of the sliding window. If we have N redundancy bits (for example, 32 bits in the above description), we only need 1/N times update of blocks, comparing to the bit-shifting algorithm in [RFC4302].

The cost of this method is extra byte(s) being used as a redundant window. The cost will be minimal if the window size is big enough. Actually, the extra redundant bits are not completely wasted. We could reuse the unused bits in the block where index WB resides, i.e., the supported window size could be (M-1)*N, plus the unused bits in the last block.

3. Example of the New Anti-Replay Algorithm

Here is the example code to implement the algorithm of anti-replay checks and updates, which is described in the previous sections.

```
<CODE BEGINS>

/**
 * Copyright (c) 2012 IETF Trust and the persons identified as
 * authors of the code. All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, is permitted pursuant to, and subject to the license
 * terms contained in, the Simplified BSD License set forth in Section
 * 4.c of the IETF Trust's Legal Provisions Relating to IETF Documents
 * (http://trustee.ietf.org/license-info).
 *
 */

/**
 * In this algorithm, the hidden window size must be a power of two,
 * for example, 1024 bits.  The redundant bits must also be a power of
 * two, for example 32 bits.  Thus, the supported anti-replay window
 * size is the hidden window size minus the redundant bits.  It is 992
 * in this example.  The size of the integer depends on microprocessor
 * architecture.  In this example, we assume that the software runs on
 * a 32-bit microprocessor.  So the size of the integer is 32.  In order
 * to convert the bitmap into an array of integers, the total number of
 * integers is the hidden window size divided by the size of the
 * integer.
 *
 * struct ipsec_sa contains the window and window related parameters,
 * such as the window size and the last acknowledged sequence number.
 *
 * all the value of macro can be changed, but must follow the rule
 * defined in the algorithm.
 */

#define SIZE_OF_INTEGER       32 /** 32-bit microprocessor */
#define BITMAP_LEN            (1024/ SIZE_OF_INTEGER)
                                /** in terms of the 32-bit integer */
#define BITMAP_INDEX_MASK     (IPSEC_BITMAP_LEN-1)
#define REDUNDANT_BIT_SHIFTS  5
#define REDUNDANT_BITS        (1<<REDUNDANT_BIT_SHIFTS)
#define BITMAP_LOC_MASK       (IPSEC_REDUNDANT_BITS-1)

int
ipsec_check_replay_window (struct ipsec_sa *ipsa,
                           uint32_t sequence_number)
{
    int bit_location;
    int index;

    /**
     * replay shut off
     */
    if (ipsa->replaywin_size == 0) {
        return 1;
    }

    /**
     * first == 0 or wrapped
     */
    if (sequence_number == 0) {
        return 0;
    }

    /**
     * first check if the sequence number is in the range
     */
    if (sequence_number>ipsa->replaywin_lastseq) {
        return 1;  /** larger is always good */
    }
    /**
     * The packet is too old and out of the window
     */
    if ((sequence_number + ipsa->replaywin_size) <
        ipsa->replaywin_lastseq) {
          return 0;
    }

    /**
     * The sequence is inside the sliding window
     * now check the bit in the bitmap
     * bit location only depends on the sequence number
     */
    bit_location = sequence_number&BITMAP_LOC_MASK;
    index = (sequence_number>>REDUNDANT_BIT_SHIFTS)&BITMAP_INDEX_MASK;

    /*
     * this packet has already been received
     */
    if (ipsa->replaywin_bitmap[index]&(1<<bit_location)) {
        return 0;
    }

    return 1;
}

int
ipsec_update_replay_window (struct ipsec_sa *ipsa,
                            uint32_t sequence_number)
{
    int bit_location;
    int index, index_cur, id;
    int diff;

    if (ipsa->replaywin_size == 0) {  /** replay shut off */
        return 1;
    }

    if (sequence_number == 0) {
        return 0;     /** first == 0 or wrapped */
    }
    /**
     * the packet is too old, no need to update
     */
    if ((ipsa->replaywin_size + sequence_number) <
        ipsa->replaywin_lastseq) {
           return 0;
    }

    /**
     * now update the bit
     */
    index = (sequence_number>>REDUNDANT_BIT_SHIFTS);

/**
 * first check if the sequence number is in the range
 */
if (sequence_number>ipsa->replaywin_lastseq) {
    index_cur = ipsa->replaywin_lastseq>>REDUNDANT_BIT_SHIFTS;
    diff = index - index_cur;
    if (diff > BITMAP_LEN) {  /* something unusual in this case */
        diff = BITMAP_LEN;
    }

    for (id = 0; id < diff; ++id) {
        ipsa->replaywin_bitmap[(id+index_cur+1)&BITMAP_INDEX_MASK]
            = 0;
    }

    ipsa->replaywin_lastseq = sequence_number;
}

    index &= BITMAP_INDEX_MASK;
    bit_location = sequence_number&BITMAP_LOC_MASK;

    /* this packet has already been received */
    if (ipsa->replaywin_bitmap[index]&(1<<bit_location)) {
    return 0;
}

    ipsa->replaywin_bitmap[index] |= (1<<bit_location);

    return 1;
}

<CODE ENDS>
```

4. Security Considerations

This document does not change [RFC4302] or [RFC4303]. It provides an alternate method for anti-replay.

5. Acknowledgements

The idea in this document came from the software design on one high-performance multi-core network processor. The new network processor core integrates a dozen of crypto core in a distributed way, which makes hardware anti-replay service impossible.

6. Normative References

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, December 2005.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, December 2005.

Authors' Addresses

Xiangyang Zhang
Futurewei Technologies
2330 Central Expressway
Santa Clara, California 95051
USA

Phone: +1-408-330-4545
EMail: xiangyang.zhang@huawei.com

Tina Tsou (Ting Zou)
Futurewei Technologies
2330 Central Expressway
Santa Clara, California 95051
USA

Phone: +1-408-859-4996
EMail: tena@huawei.com