IPSec Lab

1 Overview

The learning objective of this lab is for students to integrate a number of essential security principles in the implementation of IPSec. IPSec is a set of protocols developed by the IETF to support secure exchange of packets at the IP layer. IPSec has been deployed widely to implement Virtual Private Networks (VPNs). The design and implementation of IPSec exemplify a number of security principles, including encryption, one-way hashing, integrity, authentication, key management, and key exchange. Furthermore, IPSec demonstrates how cryptography algorithms are integrated into the TCP/IP protocols in a transparent way, such that the existing programs and systems do not need to be aware of the addition of IPSec. In this lab, students will implement a simplified version of IPSec for `Minix`.

2 Lab Tasks

The entire IPSec protocol is too complicated for a lab that is targeted at four to six weeks. To make it feasible, we only implement a part of the IPSec protocol; in addition, we have made a number of assumptions to simplify the implementation.

(1) ESP Tunneling Mode. IPSec has two different types of headers: Authentication Header (AH) and Encapsulating Security Payload (ESP); moreover, there are two modes of applying IPSec protection to a packet: the *Transport* mode and the *Tunnel* mode. In this lab, you only need to implement the *ESP tunneling mode*. In ESP, the authentication is optional; however, in this lab, we make it mandatory. Namely, the ESP authentication part should be included in every ESP packet.

```
         BEFORE APPLYING ESP
      ----------------------------
IPv4  |orig IP hdr  |     |      |
      |(any options)| TCP | Data |
      ----------------------------

         AFTER APPLYING ESP
      ---------------------------------------------------------------
IPv4  |New IP hdr   | ESP |orig IP hdr  |     |      |   ESP   | ESP|
      |(any options)| Hdr |(any options)| TCP | Data | Padding |Auth|
      ---------------------------------------------------------------
                          |<------------ encrypted ----------->|
                    |<------------ authenticated ------------->|
```

(2) Security Association (SA) To enable IPSec between two hosts, the hosts must be configured. Configuration of IPSec is achieved by defining Security Associations (SAs). A Security Association is a simplex "connection" that affords security services to the traffic carried by it. To secure typical, bi-directional communication between two hosts, or between two security gateways, two Security Associations (one in each direction) are required.

A security association is uniquely identified by a triple consisting of a Security Parameter Index (SPI), an IP Destination Address, and a security protocol (AH or ESP) identifier. There are two types of SAs: transport mode and tunnel mode. Since in this lab, we only implement the tunnel mode, so we only have the tunnel mode SA. We use an example to illustrate the use of SAs:

```
On Host: 192.168.10.100:
------------------------
   Direction     Dest IP      Protocol  Mode      SPI
   OUTBOUND   192.168.10.200    ESP     Tunnel    5598
   INBOUND    192.168.10.100    ESP     Tunnel    6380

On Host: 192.168.10.200:
------------------------
   Direction     Dest IP      Protocol  Model     SPI
   OUTBOUND   192.168.10.100    ESP     Tunnel    6380
   INBOUND    192.168.10.200    ESP     Tunnel    5598
```

The first SA on host `192.168.10.100` indicates that for any outbound packet to `192.168.10.200`, we would use the ESP tunnel mode to process the packet. The SPI value we put in the ESP header is 5598. It should be noted that the SPI value will be attached to ESP packet, and it allows the receiving side lookup the security parameters (e.g. keys) using this index. The number needs to be unique for a node. The second SA on `192.168.10.100` indicates that for any inbound IPSec packet, if the target is `192.168.10.100`[1], and the SPI in the packet is 6380, then use this entry to process the packet. To make this setting works on `192.168.10.100`, the SAs on the other end of the tunnel (`192.168.10.200`) should be set up accordingly. It should be noted that a SA is set for each direction. That is why we have two SAs on each host to setup a bi-directional tunnel between `192.168.10.100` and `192.168.10.200`.

An important part of SAs is Security Parameter Index (SPI). SPI is an 32-bit identifier that helps the recipient select which security parameters to use on the received packet. SPI can be thought of as an index into a table of security parameter settings. In the above example, SPI 5598 refers to the security parameters used by the communication from `192.168.10.100` to `192.168.10.200`, and SPI 6380 refers to the security parameters used by the other direction. On both machines, the security parameters indexed by the same SPI should be the same. For example, the following SPIs and security parameters should be set on both machines.

```
On Host: 192.168.10.100  and 192.168.10.200
--------------------------------------------
SPI     Encryption    Key       MAC
5598    AES-CBC      "aaaaa"    HMAC-SHA-256
6380    AES-CFB      "bbbbb"    HMAC-MD5
```

[1]Note that gateways can have multiple IP addresses, each having different IPSec tunnels.

(3) Setting the Outer IP Header in ESP Tunnel Mode. In ESP tunnel mode, an outer IP header needs to be constructed. Please read the RFC 2401 (Section 5.1.2) for details on how the outer header is constructed. We would like to mention how the src and dest IP addresses are constructed in the outer IP header. The way how they are constructed depends the type of the IPSec tunnel:

- *Host-to-Host Tunnel:* If we only use IPSec to establish an ESP tunnel between two hosts, then the src and dest IP addresses will be copied from the inner IP header. However, in addition to this host-to-host tunnel,

- *Host-to-Gateway Tunnel:* In this type of tunnel, the src IP is still copied from the inner IP header, but the dest IP becomes an gateway's IP address. For example, an original packet with dest IP A can be wrapped in a IPSec packet with dest IP G (G is a gateway). When the packet arrives at G through the host-to-gateway ESP tunnel, G unwraps the IPSec packet, retrieves the original packet, and routes it to the intended target A.

- *Gateway-to-Gateway Tunnel:* in this type of tunnel, both src and dest IP addresses are different from the inner IP header. Settings of src and dest IP addresses should also be defined in SAs, so you should add corresponding fields to the SAs entries used in the previous example.

The host-to-gateway and gateway-to-gateway tunnels are widely used to create Virtual Private Network (VPN), which brings geographically distributed computers together to form a secure virtual network. For example, you can have a host X in London, which creates a host-to-gateway ESP tunnel with a headquarter's gateway G located in New York. From the security perspective, G can consider that X is directly connected to itself, and no one can compromise the communication between X and G, even though the actual communication goes through the untrusted Internet. Therefore, the headquarter can treat X as a member of its own private network, rather than as an outsider.

In this lab, your IPSec implementation should be able to support the host-to-host, host-to-gateway, and gateway-to-gateway tunnels. Moreover, you need to to demonstrate how your implementation can be used to construct VPNs. In the guideline, we will describe how to set up your network environment to demonstrate your VPNs.

(4) SA and Key Management. IPSec mandates support for both manual and automated SA and cryptographic key management. The IPSec protocols are largely independent of the associated SA management techniques, although the techniques involved do affect some of the security services offered by the protocols.

The simplest form of management is manual management, in which a person manually configures each system with keying material and security association management data relevant to secure communication with other systems. Manual techniques are practical in small, static environments but they do not scale well. For example, a company could create a Virtual Private Network (VPN) using IPSec in security gateways at several sites. If the number of sites is small, and since all the sites come under the purview of a single administrative domain, this is likely to be a feasible context for manual management techniques.

Widespread deployment and use of IPSec requires an Internet-standard, scalable, automated, SA management protocol. Such support is required to accommodate on-demand creation of SAs, e.g., for user- and session-oriented keying. (Note that the notion of "rekeying" an SA actually implies creation of a new SA with a new SPI, a process that generally implies use of an automated SA/key management protocol.) The default automated key management protocol selected for use with IPSec is IKE (Internet Key Exchange) under the IPSec domain of interpretation. Other automated SA management protocols may be employed.

In this lab, you only need to implement the manual method; namely, system administrators at both ends of a communication manually setup and manage the SAs and secret keys. Your implementation should provide system administrators with an interface to conduct such manual management.

(5) Encryption Algorithm. We assume that AES algorithm (a 128-bit block cipher) is used for encryption and decryption. AES's key size can be 128 bits, 192 bits, or 256 bits. Your IPSec implementation should be able to support all these three options. The code given in `aes.c` is for encrypting/decrypting one block (i.e. 128 bits); if we need to encrypt/decrypt data that are more than one block, we need to use a specific AES mode, such as ECB (Electronic Code Book), CBC (Cipher Block Chaining), CFB (Cipher Feedback), etc. In this lab, we only support the AES-CBC and AES-CFB modes. You need to implement AES-CBC and AES-CFB using the given AES code.

Both modes require an Initial Vector (IV), which should be carried in each packet. According to RFC 3602 (http://www.faqs.org/rfcs/rfc3602.html), the ESP payload is made up of the IV followed by raw ciphertext. Thus the payload field, as defined in ESP, is broken down according to the following diagram:

```
+---------------+---------------+---------------+---------------+
|                                                               |
+               Initialization Vector (16 octets)               +
|                                                               |
+---------------+---------------+---------------+---------------+
|                                                               |
~ Encrypted Payload (variable length, a multiple of 16 octets)  ~
|                                                               |
+---------------------------------------------------------------+
```

AES-CBC requires that data must be encrypted as data chunk with 16 bytes unit. If the data is not multiple of 16, we need to pad the data, and save how many octets we have padded. receivers need this length to restore the original data after decryption.

(6) MAC Algorithm. To compute the authentication data in the ESP tail, we need to generate a MAC (Message Authentication Code). A family of MAC algorithms is called HMAC (Hashed MAC), which is built on one-way hash functions. A specific HMAC algorithm is called HMAC-XYZ if the underlying hash function is XYZ. IPSec can support various HMAC instances, such as HMAC-MD5, HMAC-SHA-256, etc. In this lab, we only support HMAC-SHA-256. The implementation of hash algorithm SHA-256 is given to you; you need to use it to implement HMAC-SHA-256. To help you, we provide an implementation of HMAC-MD5, which is quite similar to HMAC-SHA-256.

3 Design and Implementation Issues

In this lab, you need to make a number of design and implementation choices. Your choices should be justified, and the justification should be included in your lab report.

1. *IPSec Configuration.* By default, machines communicate with each other without using IPSec. To let two machines A and B communicate using IPSec, system administrators need to configure A and B accordingly. Your system should be able to support such configuration. The configuration should not require a system reboot. You might need to implement some commands to achieve this goal.

   When we setup IPSec between A and B, but not between A and C, A should still be able to communicate with both B and C, where IPSec is used between A and B, while regular IP is used between A and C. Moreover, your implementation should be *backward compatible*; namely, your IPSec-enabled `Minix` should still be able to communicate with other machines that do not support IPSec.

2. *Transparency.* Your implementation should be transparent to the upper TCP, UDP, and application layers, especially the application layer. Namely, applications such as `telnet`, `ftp`, etc. should not be affected at all. You can use these applications to test your IPSec implementation, while turning on sniffers to monitor whether the traffic is encrypted or not.

3. *Fragmentation.* You need to think about when to start IPSec within the IP protocol. Should it be done before fragmentation or after? In your demo, you should demonstrate that IP fragmentation still works. You need to think about how to demonstrate this. You may have to write a program or find a suitable tool to achieve this goal. For example, you can write a program that constructs a large UDP packet; sending this UDP packet will cause fragmentation.

4. *Impact on existing TCP connection.* It is possible that in the middle of an existing TCP connection (over an IPSec tunnel), the key used for the tunnel is modified, but not at the same time for the both ends. Namely, there is a short period of time when the two ends of the IPSec tunnel do not have the same key. What will happen to the existing TCP connection? Will it be broken? If you implement the IPSec correctly, it should not. You need to demonstrate this.

5. *Key Management* You need to think about the following key management issues regarding the keys used by IPSec: what data structure do you use to store keys? where do you store keys? how to secure keys? how to update keys. Regarding key updates, system administrators should be able to add/delete/modify/print the keys dynamically (i.e., there is no need for system rebooting).

4 Suggestions

Based on our past experience with this lab, we have compiled a list of suggestions in the following. It should be noted that this list only serves for suggestion purposes; if your designs or experience are different, feel free to ignore them, but we appreciate it if you can sent us your suggestions.

1. *Modularization.* Modularize your implementation into three major parts: (1) Process outgoing packets in `ip_write.c`. (2) Process incoming packets in `ip_read.c`. (3) SA and key management. The third module are loosely connected with the other two modules, and can be independently implemented. However, many students feel that the third module is the easiest to implement among the three modules, because, unlike the previous two modules, it does not require understanding and modification of the IP stack.

2. *Code Reading.* You need to read a lot of `Minix` code in this lab. It is quite inconvenient to read code in the `Minix` environment because of the lack of tool support in `Minix`. We suggest that you copy the entire source code to your host machine (`Windows` or `Linux`), and use code-reading tools that are available on those platforms. All the source code of `Minix` can be found under the `/usr` directory. We also put a copy of the entire source code on the web page of this lab.

   Browsing source code of `Minix` is not easy, because source code is in a number of directories. Sometimes, it is quite difficult to find where a function or data structure is defined. Without right tools, you can always use the generic search tools, such as `find` and `grep`. However, many of our past students have suggested a very useful tool called *Source Insight*, which makes it much easier to navigate source code of a complicated system. It provides an easy way to trace function and data structure definitions, as well as other useful features. This software can be found at http://www.sourceinsight.com; it is not free, but it does have a 30-day free trial period, which should be enough for this lab. Another choice for browsing source code is to use the online `Minix` source code at http://chiota.tamacom.com/tour/kernel/minix/.

3. *How* `Minix` *Networking Works I.* Understanding how networking works in `Minix` is essential for this project. Several helpful documentations are available. In particular, we highly recommend the documentation at `http://www.os-forum.com/minix/net/`, which provides a line-by-line analysis of Philip Homburg's network service for Minix, version 2.0.4 (the version that we use in this lab). Our past students found the documentation very useful. Please focus on three files: `buf.c`, `ip_read.c` and `ip_write.c`. All outgoing IP packets are processed in `ip_write.c`, and all incoming IP packets sent to up layers (TCP/UDP) are processed in `ip_read.c`. You need to use functions defined in `buf.c` and add IPSec functions in `ip_read.c` and `ip_writes.c`.

4. *How* `Minix` *Networking Works II.* We have developed a document to further help you understand how the `Minix` networking works. The document can be found at the lab web site. It guides you through several source code to show you a big picture on how a packet is forwarded from application to ICMP/TCP/UDP to IP, and then to Ethernet. It also describes how `add_route.c` and `pr_routes.c` works. These last two files (in `/usr/src/commands/simple`) can serve as a good example on how to store and maintain (routing) information in the kernel. If your need to do the similar thing (i.e., storing information in the kernel), you can use the system calls in `inet`, such as `ioctl()` in `ip_ioctl.c`, which need to be changed to add more functionalities. The files `pr_routes.c` and `add_routes.c` give you a good example on how to use the system calls.

5. *Network Setup for VPN Demonstration.* Please refer to our document "IPSec Gateway-to-Gateway Network Configuration". This document is listed in the lab web page.

5 Software Engineering

It should be noted that building software for security purpose is quite different from traditional software engineering. Although the common professional software engineering practice still applies to this project to ensure that the developed software system works correctly, extra engineering principles should be followed to ensure the system works *securely*.

- *Threat evaluation:* Before designing a system, developers should evaluate the potential attacks that the system might face. The design of the system should address how the system can defeat these attacks. In your final project report, you need to include such threat evaluation.

- *Using cryptographic algorithms correctly:* Although the cryptographic algorithms that you use might be strong, using them incorrectly will still make your system vulnerable. There are many real-world stories regarding the misuse of encryption and one-way hash algorithms. In this project, you should make sure that you follow good practice:

  - *Choice of algorithms:* Although in this lab, we have chosen the encryption and MAC algorithms for you. In real world, when you need to make your own choice, you need to understand the strength and weakness of the algorithms. For example, you should never choose DES because of its proven weakness in key length.
  - *Choice of modes:* You should understand the strength of each encryption mode, and avoid using the modes that are weak in security, such as the Electronic Codebook (ECB) mode.
  - *Randomizing initialization vector (IV):* It has been shown that for some encryption algorithm (such as DES), repeating using the same IV is not safe. Therefore, it is a good practice to always use a randomly-generated IV at each time. DO NOT hard-code the IV value in the program.

  - *Pseudo-random number generators:* make sure that your pseudo-random number generators are good, i.e., the number that it generates are random and unpredictable.
  - *Key management:* One of the challenges in cryptography is key management, i.e., how/where to store keys, how to update keys, how to protect keys, etc. In your project report, you need to describe how you handle the key management problem. In particular, you should describe your key management for the following scenario (you are not required to implement this scenario, but you must describe your design): as we said earlier, in this IPSec project, we allow administrators to manually type in the keys at both ends of an IPSec tunnel. If a computer (e.g. a gateway) needs to establish many IPSec tunnels with other machines, administrators might want the machine to automatically load the keys from a configuration file. Please describe how you plan to implement your system to support this.

- *Security testing:* In addition to testing the functionalities of your system, you should also test the security of your system. The test cases that you use for testing should cover those potential attacks identified in threat evaluation. In your report, you need to include these test cases, and justify how they are related to the threat evaluation.

6 Submission and Demonstration

You should submit a detailed lab report to describe your design and implementation. You should also describe how you test the functionalities and security of your system. You also need to demonstrate your system to us. Please sign up a demonstration time slot with the TA. Please take the following into consideration when you prepare for demonstraiton:

- The total time of the demo will be 15 minutes, no more additional time would be given. So prepare your demonstration so you can cover the important features.
- You are entirely responsible for showing the demo. We will NOT even touch the keyboard during the demonstration; so you should not depend on us to test your system. If you fail to demo some important features of your system, we will assume that your system does not have those features.
- You need to practice before you come to the demonstration. If the system crashes or anything goes wrong, it is your own fault. We will not debug your problems, nor give you extra time for it.
- During the demo, you should consider yourself as salesmen, and you want to sell your system to us. You are given 15 minutes to show us how good your system is. So think about your sales strategies. If you have implemented a great system, but fail to show us how good it is, you are not likely to get a good grade.
- Do turn off the messages your system prints out for debugging purposes. Those messages should not appear in a demonstration.

7 Grading Criteria

The grading criteria are described in the following. To gain those points, you need to demonstrate the corresponding features:

1. Crypto library: 10 points.

2. IPSec configuration: 20 points.
   - User-level utilities to allow administrator to configure IPSec tunnels, such as add/delete tunnels, set/update keys, set/update security parameters, etc.
   - Utilities for administrators to list IPSec configuration.
3. IP and ICMP Protocols: 15 points.
   - IP fragmentation should still work. You need to demonstrate how to test this.
   - ICMP-based applications such as `ping` should still work.
4. TCP and UDP Protocols: 25 points (it should be noted that you are not supposed to modify the TCP and UDP parts, and your IPSec should not affect the these parts).
   - TCP-based applications, such as `telnet` and `ftp`, should still work.
   - Updating keys used in a IPSec tunnel should not break the existing TCP connections. You can update the key on one end of an IPSec tunnel; you should be able to see that the existing TCP connections using this tunnel will freeze, but not broken. After you update the keys on the other end of the tunnel, the connections will resume working. This is a good way to test whether your IPSec implementation breaks TCP.
   - UDP-based applications should still work. You can use the provided UDP client/server program to do the testing.
5. Virtual Private Network (VPN): 10 points.
6. Software Engineering and overall impression (20 points): we will evaluate how well you apply the software engineering principles in ensuring the security of your system. It is your responsibility to show us the evidence during your demonstration. If you don't show us anything regarding this, we will assume that you have not given this a serious thought, and will hence deduct points from you.

8 Reference

1. RFC 2401 – Security Architecture for IPSec.
2. RFC 2406 – IP Encapsulating Security Payload (ESP).