USENIX Association

Proceedings of the LISA 2001 15th Systems Administration Conference

San Diego, California, USA
December 2–7, 2001

IPSECvalidate – A Tool to Validate IPSEC Configurations

Reiner Sailer, Arup Acharya, Mandis Beigi, Raymond Jennings, and Dinesh Verma
– IBM T. J. Watson Research Center NY

ABSTRACT

This paper describes a tool for validating the proper configuration of the IPSEC protocol suite including IKE. The tool validates that two hosts are able to communicate (normal ping functionality) and that this communication is occurring using the proper authentication/encryption transformations as required by IPSEC. IPSEC configuration is very complex, and administrators are often unable to determine if a machine configuration is offering the desired protection. IPSEC and IKE operate in a manner transparent to IP applications; an administrator is therefore unable to check the proper operation of an IPSEC "security association" using traditional IP tools.

Introduction

Security for IP-based networks has become increasingly important: with many companies relying on Virtual Private Networks (VPNs) for distributed intranet, extranet, and remote access services, security requirements have become essential. The IETF has developed security protocols and mechanisms that extend conventional IP services by security services [5, 10, 3].

The IP security protocols (IPSEC) are used to encapsulate IP data packets (tunnel mode) or their payload (transport mode). Two protocols are standardized: the IP Encapsulating Security Payload (ESP [7]) and the IP Authentication Header (AH [6]), which offer confidentiality and authentication services. The Internet Key Exchange (IKE [4]) comprises protocols and mechanisms to automatically configure these IPSEC protocols and to maintain so-called "security associations." Once configured, IKE will set up and maintain IPSEC protected links autonomously as needed.

The downside is that configuring IKE and IPSEC is quite complex due to the flexibility needed to ensure inter-operability of different IKE and IPSEC implementations and to different security needs. Different encapsulation techniques, operation modes, and algorithms, which may vary for different network interfaces and destinations, making it difficult for system administrators to determine the specific security configuration used when communicating with a particular host.

Furthermore, emerging IPSEC management tools automatically configure VPNs by configuring IKE or – in case of manual keying – IPSEC to set up protected links between the gateways of the interconnected networks according to a company-wide security policy. Management tools are also needed to translate changes in the security policy into proper configuration changes of IKE and IPSEC. Our own work on the central management of IPSEC VPNs and the resolution of inter-operability problems of IPSEC implementations have underscored the need for additional tools that validate IPSEC configurations.

We tried several other ways to solve our problems: comparing round-trip times reported by the ping program to decide whether IPSEC encryption or authentication is actually applied and looking up IPSEC management commands of various IPSEC implementations to get unintelligible information about active "security associations." Eventually, we decided to develop our own tool with ease-of-use as a primary design goal. Our validation tool, called IPSECvalidate,

- validates what kind of encapsulation (ESP, AH, AH/ESP) and what mode of operation (transport, tunnel) is applied to IP packets from the local host to a particular remote host
- offers a command-line interface that can be used by other programs to validate VPNs consisting of multiple IPSEC links
- is independent of specific IPSEC implementations (as our scenarios span AIX machines, Windows2000 machines, Linux machines, and Cisco routers).

We will begin by describing the validation goals in more detail. Next, we will present the approach that we chose to validating IPSEC configurations. We will discuss alternative approaches and justify the chosen approach. Finally, we evaluate the potential impact of different kinds of attacks on the reliability of the validation tool's report. Specifically, we show that our tool is resistant against attacks from the network; i.e., attacks from the network cannot make our validation tool report that IPSEC protection is present although there is actually no protection.

Validation Goals

In this paper, we propose a methodology to test the validity of communication tunnels in an IP network that have been set up using IP security protocols. IPSEC tunnels are used to construct a VPN over a public network such as the Internet. VPNs are typically used by multi-site organizations to interconnect their different locations. Until recently, this was accomplished by setting up dedicated tunnels such as Frame Relay circuits between two sites. However, a much less expensive solution is to use a common public network such as the Internet instead of dedicated interconnect links. This is accomplished by setting up IP based tunnels over the public IP network. At the same time, this raises a security issue since unlike dedicated Frame Relay circuits that carry an organization's traffic in an exclusive manner, the public IP network is a shared network. Thus, IPSEC mechanisms are used to set up secure tunnels that comprise a virtual private network amongst the different sites of a multi-site organization over a shared public IP network. This paper does not focus on the mechanism to set up such IPSEC tunnels but instead describes mechanisms to ensure that once such tunnels are set up, they operate as expected.

Our goal is to test whether the configuration of IKE and IPSEC on the various nodes of a VPN has been applied successfully. We consider a VPN to be comprised of a set of sites ("trusted domains") interconnected by an insecure Internet ("distrusted domain"), and want to ensure that the VPNs connecting these sites are correctly configured, thus providing protection. Furthermore, this needs to be verified before any of the client sites send data. Figure 1 depicts the basic scenario in which we did our work.

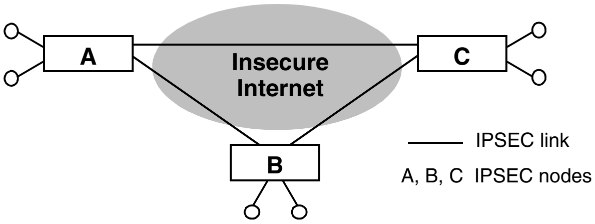


Figure 1: Basic VPN scenario using IPSEC.

This paper will present methodologies to test whether IPSEC associations have been successfully established between the three IPSEC nodes A, B and C of the VPN. These test methodologies are applied to each pair-wise connection, e.g., between A and B.

We assume that the IKE/IPSEC implementations on nodes A and B conform to the standards and that the connectivity configuration (e.g., VPN routing tables, policy tables) has been set up correctly. After nodes A and B have been configured with IKE/IPSEC, the problem is to check whether packets sent between A and B travel on the wire with the proper IPSEC parameters applied to the packets, e.g., whether they are sent as clear text or protected. IPSEC can be applied either through an Authentication Header (AH) or through an Encapsulating Security Payload (ESP), thus requiring two different sets of tests.

There are three possible methods we might chose to accomplish our goals: (a) snoop the packet on the sending node's network interface, (b) force an intermediate router on the path from A to B to send a portion of the packet back to the sending node, and (c) specially configure a router guaranteed to be on the path from A to B to return all packets to the sender if their header fits a specific pattern.

We implemented option (a) in our validation tool. We will discuss the alternative approaches and assumptions about attackers after presenting our implementation of this approach.

Note that the need for inspecting packets on the wire arises because the packets need to be checked for validity after IPSEC has been applied at the sending node and before IPSEC is applied at the receiving node. Otherwise, once a packet has traversed the IPSEC layers on both the sending and receiving nodes, there is no way to distinguish between packets that traversed the intervening network in clear text or with IPSEC applied.

Tool Description

The validation application is started by specifying a destination IP address and a protocol number:

```
$ipsecvalidate -d 192.168.19.48 -p 50
report:ICMP Packet Loss 0%
Transformations [IPSEC ESP tunnel mode]
are occurring as expected
$
```

IPSECvalidate tests whether all packets being sent to or received from destination address 192.168.19.48 are using the protocol specified by the -p option (e.g., protocol number 50, IPSEC ESP) for communication. To accomplish this, the application listens to all link layer frames both sent and received on all physical interfaces of the local host and examines all of these frames. It then compares the protocol number given as a command-line argument with the protocol field in the IP header of packets transported in these frames. The validation process consists of two parts:

- Any packet sent to or received from the remote host are inspected at the data link level regarding their encapsulation (protocol).
- ICMP Echo Request packets are sent to the destination host and ICMP Echo Reply messages are used to verify connectivity and to generate traffic, which is then inspected to verify the respective encapsulation.

The first step is to determine how many and what types of interfaces are present. This is typically accomplished by sending a query to the kernel. It is important to verify that each interface is operational and capable of hearing its own transmissions; flags in the data structure returned by the kernel will verify this. If any interface is not capable of hearing its own transmissions then it is not possible to verify the outbound communication.

The second step is to determine the source IP address to be used in the Echo Request packets. If a machine has a single interface then this interface's address is used. However, if a machine is multi-homed (which is the usual case for gateways) then the source address for the Echo Request packets has to be determined in order to recognize the corresponding Echo Reply packets while listening on the network interfaces. The source address to be used is determined by repetitive calls using the routing socket API. A routing socket is used to look-up the gateway for the remote host (destination). Once the gateway is determined the next step is to find the IP address of the interface that will be used to send the packets to the remote host (via the gateway). This also accomplished using the routing socket.

Once the list of valid interfaces has been created and the outgoing interface for the Echo Request packets has been determined, we create an appropriate number of slave threads, each listening to a single interface. Each slave thread will look for frames that match the source and destination IP addresses.

Several different methods can be used to listen to an interface. One method is to make use of a network tap; this is accomplished by creating a network tap socket and binding to a particular interface. The network tap receives copies of all packets that an interface sees. One problem encountered with the implementation based on AIX Unix was that only one application on a machine could have access to the network tap at one time. Therefore, if another application were currently using the network tap, the IPSECvalidate application would fail to run. The benefit of using the network tap is ease of programming.

We chose another method – snooping packets. On AIX we did this using the Berkeley Packet Filter API [9]; on Linux, we used the packet capture library (libpcap [2]).

Once the slave threads have been created and are listening to their respective interfaces, the main thread will send out a series of Echo Request (ping) packets to the destination host. The main thread also listens for Echo Reply packets to verify that the destination was reached and to check for packet loss. We include the process ID (PID) of the main thread in the Echo Request packets and inspect the PID payload of the received Echo Reply messages to make sure we count the replies to only our own requests.

We guard each local host interface by a listening thread because we might receive IP packets from the remote host via different interfaces. Each slave thread counts those filtered packets whose protocol field matches or doesn't match the protocol specified in the command line.

Once the main thread has received the ping responses or a time-out occurred, it will terminate the slave threads and release any resources associated with them. At this point the slave threads should have seen at least as many frames as Echo packets have been sent and received. If any slave thread received a frame where the source and destination IP addresses matched but the protocol field was not as specified then the transformations are considered to be not occurring as expected and the tunnel is not working correctly. Otherwise the transformations are occurring as expected.

If called with the "quiet" command line option, IPSECvalidate does not produce any output but communicates the validation results in the return value. This option can be used by other applications to dynamically determine the protection applied to IP packets exchanged with a particular remote network node, e.g., to support access control decisions.

The tool was implemented for AIX and Linux. On AIX, we used a modified packet capture library to capture and filter layer 2 frames and the standard routing socket to determine interfaces and routes. On Linux we used the standard packet capture library [2] and the library of IPRoute2 [8] to determine interfaces and routing information. The packet capture library supports numerous operating systems including FreeBSD, BSD, Linux, HP-UX, and Solaris. IPSECvalidate can be ported to other Unix-based operating systems by adapting the packet capture and routing socket calls as needed to the interfaces offered by the respective system.

Alternative Approaches

We considered several other approaches to the problem, only to find they were not feasible.

One approach we tried was to develop a protocol that runs between both of the machines in order to validate a secure tunnel. Using such a protocol, an initiating machine would send an initial probe message to a remote machine with which the IPSEC tunnel has been established. The other machine would then respond with a reply message which will only be generated if the corresponding packet was encrypted properly. If such a protocol can be developed, it will provide functionality which is similar to the one implemented by our tool.

However, because of the method in which IPSEC has been designed, the existence of a tunnel is transparent to applications running above the IP-layer. Therefore, an application layer implementation of this protocol would not be feasible. Such a protocol would need to be incorporated as a part of the IPSEC implementation. Since no such standard protocol is currently considered by the IPSEC working groups within the IETF, we would have to implement a non-standard extension to the protocol – something we did not find acceptable.

A second approach we considered was to use the TTL-expiration scheme used by programs such as traceroute. In this case, we would inspect packet headers after the relevant IPSEC transformations have been applied. In order to capture packets after the IPSEC transformation, packets are created using a TTL field which is bound to expire, e.g., a TTL of 1. This would cause the next-hop router to send back the IP-header of the transformed packet and an additional eight bytes of the ESP/AH header. However, the difficulty lies in creating an IP packet with the TTL set to 1, which will then be sent through the IPSEC encryption routines. When IPSEC is used in the tunnel mode, the TTL of the outer header is often set to the IP default TTL, thus the packet will only be returned back by the network at the other end of the IPSEC tunnel, rather than from the next hop router. This scheme would not be able to validate the proper operation of IPSEC in the tunnel mode because the returned packets cross the insecure Internet unprotected. Even if the remote router uses IPSEC to protect such packets, they cannot be used to validate the IPSEC configuration as we would have to assume that IPSEC is working correctly for the validation – something we did not find acceptable

Security Evaluation

This section illustrates the benefits of the IPSEC validation tool. First, we interpret the output of IPSECvalidate in the regular case. Then, we discuss the interpretation of the output under different assumptions: We assume in turn insecure hosts, wrongly configured IKE and IPSEC policies and "security association" databases, and attackers having access to the network.

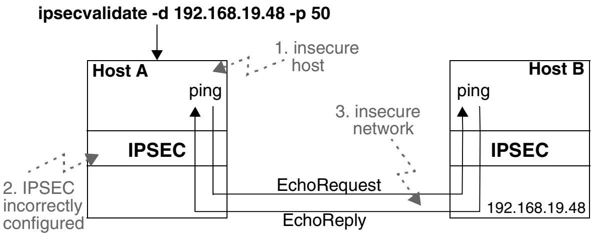
If both the local host (A) and the remote host (B) work correctly, then IPSECvalidate reports whether transformations (e.g., AH, ESP, AHESP) are occurring to data packets sent to and received from host B. If the reported ICMP packet loss is less than 100% then the IPSEC configurations of host A and host B are interoperable. Since the validation tool inspects all packets between A and B on all interfaces (on host A), route settings do not affect the validation result.

When examining the robustness of our tool against attackers, we restrict our discussion to the following scenarios:

- Local host A is insecure, i.e., its runtime environment does not work as expected.
- IKE/IPSEC configuration files do not reflect the users' security expectations.
- Attackers have access to the network and can read, replay, insert, and delete messages.

The first two scenarios involve compromised software and hardware (Trojan Horses [1]) or incorrect configuration, whereas the third scenario assumes external attackers. Figure 2 illustrates the validation tool's environment and points vulnerable to attacks.

This figure also shows how connectivity of hosts A and B is validated by the Echo protocol (which operates on top of the IPSEC sublayer). If IPSEC transforms ICMP packets the same way it transforms other IP packets, the IPSEC sublayers of hosts A and B are inter-operable for IP traffic if connectivity is reported. Other packet types, e.g., ARP packets, have mostly local significance and are usually not protected by IPSEC.



```
ipsecvalidate -d 192.168.19.48 -p 50
```

Figure 2: Points vulnerable to attacks.

If host A is corrupted due to internal attacks then users cannot securely interact with it; a validation tool running on this host is useless. If host B is corrupted, then using properly configured IPSEC to communicate with this host does not offer benefits; host B could distribute the actual session keys to the insecure Internet or leak any information through unprotected channels. The output of the validation tool is not useful in either case. This underscores the importance of secure runtime environments to reliably control and protect network access.

If the IPSEC configuration is corrupted on only one of the hosts, host A cannot communicate with host B. IPSECvalidate will report 100% ICMP loss in this case. Basically, 100% ICMP loss occurs if the network connection between hosts A and B is interrupted, ping is blocked by packet filters in between A or B, or hosts A and B have incompatible IKE or IPSEC configurations. If connectivity between hosts A and B over IPSEC is given then the IPSEC encapsulation is validated and the user can decide whether the configuration is as expected.

We will now examine how active attacks from inside the network can affect the output of IPSECvalidate. Such attacks – aimed at deceiving IPSEC protection – must generally be assumed when connected to the Internet.

Attackers cannot deceive that transformations are occurring, (i.e., they cannot provoke false positive transformation reports) because outbound packets are inspected before they enter the network; attackers cannot forge these packets from within the network. Because IPSECvalidate actively sends Echo Request packets, at least these outbound packets will be seen by one of the listening threads; these packets will reveal to the listening thread at the respective outbound interface whether expected transformations occur.

It is not possible for an external attackers to fool the tool into believing that there is connectivity, i.e., provoke false positive connectivity reports, if IPSEC uses authentication or encryption because the Echo protocol is applied on top of IPSEC. IPSEC will discard inserted or replayed packets. If no replay detection is used, replay attacks are still difficult as we include the ID of the sending process in the ping packets and check it when receiving Echo Replies. This PID is likely to change for different invocations; hence even without authentication and replay protection, attackers cannot successfully replay Echo Reply messages of former connectivity checks (even if the IPSEC session key does not change). A random number can be added to the PID to achieve stronger protection against replay attacks.

Nevertheless, active attacks originating from the network (e.g., through replay, insertion, or deletion of messages) can provoke false negative reports:

- The tool can incorrectly report that transformations are not occurring as expected; this happens because IPSECvalidate does not interact with the IPSEC implementation. Therefore, it cannot validate the authenticity of incoming packets. Accordingly, attackers can insert forged packets or replay old packets that are not IPSEC-encapsulated; those are inspected by IPSECvalidate as incoming packets from host B.
- The tool can incorrectly report that connectivity is not given; this happens because attackers can selectively filter Echo Request or Reply packets exchanged between hosts A and B if no encryption is used.

In summary, IPSECvalidate can be used to verify that all IP packets between a pair of hosts are transformed using the expected security protocols and modes, regardless of the interfaces used to transmit and receive the packets.

Finally, IPSECvalidate cannot determine whether ESP actually uses strong encryption. Nevertheless, the validation tool does look for the IP header within the ESP body. If it finds the IP header at the expected offset then chances are that ESP does not use encryption (NULL encryption). However, if the IP header is not found we cannot conclude that strong encryption is used. Consequently, the tool is useful to determine the IPSEC encapsulation but it is at this time not thought to validate the theoretical strength of transformations (determined by algorithms and key lengths).

Summary & Outlook

In summary, IPSECvalidate can be used to verify that all IP packets between a pair of hosts are transformed using the expected security protocols and modes, regardless of the interfaces used to transmit and receive the packets. It also validates whether both hosts have compatible IKE and IPSEC configurations (connectivity over IPSEC). The validation tool is independent of the respective IKE and IPSEC implementations because it is exclusively based on standardized IPSEC protocol information and because it does not need any access to IPSEC databases. The tool assumes that the hosts are working properly and that the algorithms used within the transformations are applied correctly.

IPSECvalidate has been developed to support the validation of VPNS based on IPSEC tunnels. This can be achieved by running IPSECvalidate on all participants of a VPN and analyzing the results either on a central management node or locally on the VPN clients. The tool has proven to be very useful during the development of configuration tools for IPSEC-based VPNs. Local administrators and users will benefit from the tool because it makes normally transparent security mechanisms visible on demand.

IPSECvalidate can be used to verify any IP-based encapsulation protocol, e.g., GRE, IP-IP, or IPComp [11], simply by specifying the appropriate protocol number via the command-line options.

Future Work & Availability

Possible future extensions include heuristics that determine based on compression gain or code distribution with higher reliability whether ESP actually encrypts data or not.

We are going to release IPSECvalidate binaries for AIX and Linux to the community. Afterwards, we intend to go through the process to make the Linux source code available under GPL.

Acknowledgment

The authors would like to thank Adam S. Moskowitz for his very useful comments and suggestions, which considerably improved the presentation of this contribution.

Author Information

Arup Acharya received his B.Tech (Hons.) from IIT, Kharagpur and Ph.D. in Computer Science from Rutgers University in January, 1995. He was briefly associated with WINLAB (Wireless Information Networks Lab), Rutgers University as a post-doc working on the interaction of IP multicast with Mobile IP. He joined NEC Computers and Communications Research Labs, Princeton in May, 1995 where he worked on protocol architectures for high speed wired and mobile wireless networks. Arup joined IBM TJ Watson Research Center in Jan 2000 where his current work involves leveraging MPLS core networks to design a fast and scalable web switching infrastructure and architectures for pay-per-use Internet access through public WLAN and Bluetooth access points. Arup has been awarded three patents in high speed wired/wireless networking. He has published more than thirty papers in conferences and journals and has been a technical program committee member for MobiCom for the past three years. His homepage is http://www.research.ibm.com/people/a/arup/ .

Mandis Beigi received her Bachelors of Engineering in Electrical Engineering from the State University of New York at Stony Brook in 1993. She received her Masters of Science in the field of Electrical Engineering from Columbia University in 1995. She has been working at IBM since 1994. She currently works at the IBM T. J. Watson Research Center in the enterprise networking department and is also a Ph.D. student at Columbia University. She has worked on quality of service, service differentiation and network monitoring and management.

Raymond Jennings III is a research engineer at the IBM Watson Research Center. He obtained his MS in Computer Engineering from Manhattan College in 1996, and is currently pursuing his Ph.D. at Polytechnic University. His interests include operating systems and network performance.

Reiner Sailer is a Research Staff Member at the IBM T J Watson Research Center at Hawthorne, NY. He is an expert in network security, general purpose secure runtime environments, and security architectures for distributed applications. He holds a masters in Computer Science from the University of Karlsruhe and a doctorate of Electronic Engineering from the University of Stuttgart, Germany.

Dinesh Verma is a research manager at the IBM Watson Research Center. He obtained his Ph.D in Computer Networks from U. of California, Berkely in 1991, and has since worked at Philips Research and IBM Research in the topic of TCP/IP and ATM communication networks. He has authored three books and over thirty publications related to networking. His current interests include network performance and QoS, security, policy based management, content distribution networks and peer to peer networks.

References

[1] J. Anderson, "Computer Security Technology Planning Study," *ESD-TR-73-51, Vol I+II*, HQ Electronic Systems Division, Hanscom AFB, Ma., 1972.

[2] Network Research Group at the Lawrence Berkeley National Laboratory. *LIBPCAP 0.4: Packet Capture Library*, ftp.ee.lbl.gov, 1997.

[3] N. Doraswamy and D. Harkins, *IPSEC – The New Security Standard for the Internet, Intranets, and Virtual Private Networks*, Prentice Hall, 1999.

[4] D. Harkins and D. Carrel, *RFC2409 – The Internet Key Exchange (IKE)*. Proposed Standard, 1998.

[5] S. Kent and R. Atkinson, *RFC2401 – Security Architecture for the Internet Protocol,* Proposed Standard, 1998.

[6] S. Kent and R. Atkinson, *RFC2402 – IP Authentication Header*, Proposed Standard, 1998.

[7] S. Kent and R. Atkinson, *RFC2406 – IP Encapsulating Security Payload*, Proposed Standard, 1998.

[8] A. Kuznetsov, Institute of Nuclear Research, Moscow. "IPROUTE2," ftp.inr.ac.ru/ip-routing or ftp.sunset.se/pub/Linux/ip-routing, 1999.

[9] S. McCanne and Van Jacobson, "The BSD Packet Filter: A New Architecture for User-level Packet Capture," *USENIX Conference*, 1993.

[10] D. Piper, *RFC2407 – the Internet IP Security Domain of Interpretation for ISAKMP,* Proposed Standard, 1998.

[11] A. Shacham, R. Monsour, R. Pereira, and M. Thomas, *RFC2393 – IP Payload Compression Protocol (IPComp),* Proposed Standard, 1998.