IPSEC CLUSTERING

Antti Nuopponen,[1] Sami Vaarala,[2] and Teemupekka Virtanen[3]

[1] *Emic Networks*
antti.nuopponen@iki.fi

[2] *Stinghorn*
sami.vaarala@iki.fi

[3] *Helsinki University of Technology*
tpv@tml.hut.fi

Abstract IPsec based VPNs are widely used to secure connections in the Internet. As the bandwidth in the Internet grows there is a need to create more powerful fault tolerant IPsec systems. We present an IPsec clustering model based on sharing the processing of each IPsec connection to all cluster nodes using a master node. This clustering model offers good scalability, fine graded load balancing, and fault tolerance while maintaining all IPsec security features. We also present a test implementation of the clustering model with test results.

Keywords: IPsec, high availability, clustering

INTRODUCTION

The Internet Protocol security architecture (IPsec)[2] is commonly used to secure connection in the Internet. In the Internet the bandwidth is growing all the time, and the users are taking advantage of it. This sets higher and higher performance requirements to all components of the Internet, IPsec systems included.

One approach to the improve performance of an IPsec implementation is to use high powered machines with dedicated cryptographic hardware. Another approach to increase the capacity of an IPsec implementation is to group multiple machines in a way that they act as one IPsec node. This technique is called clustering. In addition to better scalability the clustering also provides fault tolerance, and an ability to dynamically add and remove members from the system without affecting the cluster operation.

A straightforward way to cluster IPsec is to divide each IPsec connection to a dedicated node in the cluster. This approach however suffer form several

Y. Deswarte et al. (eds.), *Security and Protection in Information Processing Systems*

disadvantages. The load balancing can only be made in the connection level. If there is for example a cluster with three nodes and there are two connections into the cluster there is no way to take advantage of all cluster nodes.

The most serious disadvantage is the lack of ability to survive member failure situations transparently to the IPsec clients without compromising the security of the IPsec. The IPsec security associations (SA) in the cluster could be synchronized between cluster nodes, and in a failure situation a new member could start to serve connections of the member that failed. This however creates a security vulnerability because the replay protection information of the cluster member that failed is lost. Because the IPsec replay protection information has to be updated after every packet it can not be kept synchronized between cluster nodes with a reasonable overhead. After the cluster member has failed there is no way of telling which packets it already had received.

In order to create an IPsec clustering model that maintains all IPsec security features and is transparent to the IPsec clients a new approach is taken. The clustering presented in this paper is based on sharing the processing of each IPsec connection (SA) to all cluster nodes using a master node. The model assumes that the cluster members do not have an ability to keep IPsec security associations synchronized real-time and that every member of the cluster can potentially fail in any point of time.

The paper is organized as fallows. In section 1 existing clustering models are presented. Section 2 describes the clustering model presented in this paper. Section 4 describes a test implementation and the conclusions are given in section 5.

1. EXISTING CLUSTERING MODELS

There are some commercial IPsec clustering products available, but technical details of them are hard to obtain.

One IPsec clustering model is based on broadcasting all packets to all cluster members in layer 2 and then selecting the processing node based on some information on the packet. In this model there is no master node in the cluster and all cluster nodes make the choice whether to process a packet or not individually. Because each cluster node perform the IPsec processing independent of each other there is no way to keep the IPsec SAs synchronized, and thus the cluster can no guarantee the replay protection in a failure situation.

Other method to cluster IPsec is to dedicate a member of the cluster to each IPsec connection when the connection is initiated. In this model the same cluster member does all the processing of a given SA, and if the member fails then the SA has to be renegotiated. To avoid the renegotiation the cluster could synchronize each SA to more than one slave and then survive failure situation.

Unfortunately the SA replay protection information becomes the problem as in the previous model.

2. CLUSTERING ARCHITECTURE

The IPsec clustering architecture presented in this paper is based on sharing the same IP addresses between all cluster nodes. Each member of the cluster has at least two network interfaces, which are named public and private interface. Since the cluster is a security gateway between the IPsec client and the correspondent node only the IPsec tunnel mode is supported[2].

All packets sent to the cluster are first received by the cluster master, which is one of the cluster nodes and is automatically selected among the cluster members to handle the packet distribution. The master then forwards each packet to a cluster node for processing. Other members of the cluster are called slaves.

This kind of clustering technique makes the whole cluster to appear as a single IP network node, and no changes to the existing IPsec clients are required as long as the cluster does not lose the SA information.

It is assumed that an IPsec client can set up an IPsec SA with the cluster using Internet key exchange (IKE)[5]. Detailed description about how that is done is out of the scope of this paper, but for example IKE negotiations can be handled in the cluster master node.

Forwarding model

The forwarding in the master node is done packet-by-packet basis using a load sharing function that takes in the packet and produces an id that points to a member of the cluster to which to forward the packet for processing. The master itself does not take part to the IPsec processing because it would make it impossible to survive a master failure without compromising the replay protection.

The advantage of this kind of load sharing is that differences between IPsec connections do not affect the load differences of individual cluster members. If an IPsec connection creates a traffic peak the effect is a load peak in all cluster members not just in one member. Other big advantage is the ability to maintain IPsec security in failure situations.

With this kind of forwarding model two issues arise; first is the IPsec replay protection when receiving packets and the second is traffic based SA lifetime. It is unrealistic to keep the replay protection information synchronized real-time and therefore a cluster member A can not know what sequence number a member B has already received. If a packet with a sequence number n is forwarded to the cluster member A for processing and the packet is re-sent

to the cluster and forwarded to the cluster member B, IPsec replay protection does not work.

Next subsections describe how these problems can be solved.

Load sharing function

In order to maintain the replay protection feature of the IPsec when receiving IPsec packets the load sharing function must always map the same sequence number to the same authentication context, or to drop it if the authentication context no longer exists. To achieve this, the load sharing function uses the IPsec sequence number field as input and performs deterministic mapping from the sequence number to a cluster member id.

The sequence number field must always be present on an IPsec packet, and the sender must add incremental sequence number to every packet[4, 3]. If the receiver explicitly tells the sender not to use reply prevention the sender can stop adding the sequence number into each IPsec packet. This does not limit the use of the sequence number in the cluster solution, because the cluster is the receiver and the cluster does not send the notification to the IPsec client. This ensures that the client must add a sequence number to each packet.

The IPsec ESP[4] transform allows operation without packet authentication. In this mode the sequence number is not authenticated and the replay prevention is not used. The clustering solution can handle this mode in a same way than other modes by treating all packets as if they passed the replay protection check. As described in [6] the use of ESP without authentication opens security vulnerabilities, and thus it is better to use authentication with ESP.

If the ESP with authentication is used all IPsec packets that are secured using the same SA have a unique sequence number value that can not be modified. Because sequence number field is present in every IPsec packet, it can not be modified and it is unique in context of one SA, it is a good basis for multiplexing the connections between cluster nodes.

Because a cluster member will most likely lose its SA information in a failure situation the authentication context is linked into the cluster member in a way that every time a member leaves the cluster and joins it again a new authentication context is created.

Outbound packets do not have the sequence number when they are received by the master. In order to use the same load sharing model the cluster master first allocates a sequence number, places it to the packet and then uses the load sharing function to decide to which member to send the packet for the actual IPsec processing.

Inbound IP traffic processing

Inbound IP traffic that does not have an IPsec protection are handled by the cluster master as specified in [2].

When an IPsec protected packet is received by the master it first performs an SA lookup to find the correct SA for the packet. If no such SA is found the packet is discarded. After the master has located the correct SA it uses the load sharing function of that SA to get the cluster member id to which to forward the packet. The function may also produce a null result which indicates that the packet must be discarded. Ones the slave receives the packet it performs the same SA lookup to find the correct SA to use. If the slave can not locate the SA it discards the packet and send notification about missing SA to the master.

If the master receives notification about missing SA it send the SA to all cluster members.

The cluster master also maintains additional information about packets it forwards. This information includes the highest seen sequence number and highest authenticated sequence numbers. The master uses this information to survive from failure situations. The highest seen sequence number is the larges seen sequence number for each SA that the master has forwarded to a cluster node. The highest authenticated sequence number is the highest sequence number that the cluster nodes have reported correctly authenticated.

Outbound IP traffic processing

In outbound packet processing the cluster master first checks the correct action from the IPsec Security Policy Database (SPD)[2]. If the action is to apply IPsec and use replay protection the master then allocates next sequence number to the packet and uses a load sharing function to decide to which cluster member to send the packet to. The master attaches the sequence number into the packet so that the slave knows which sequence number to use.

Ones the slave receives the packet from the master it locates the correct SA from the SPD and then applies the IPsec protection specified in the SA using the sequence number assigned by the master. After packet is protected the slave sends it to the other endpoint of the IPsec connection.

Handling failure situations

There are two kind of failures that can occur; first is the master failure and the second is the slave failure. These are handled differently, and as long as they do not occur simultaneously the cluster can survive them transparently to the IPsec clients and without compromising security features of the IPsec.

In a slave failure situation the cluster master performs a load re-assignment that removes the failed slave from the cluster. This is done by defining a new

load sharing function starting from the next sequence number after the highest seen sequence number. Because the master forwards all packets to the cluster members for processing this guarantees that no replay attack is possible.

When the master fails a new master is automatically selected. The new master does not forward any packets until it has asked the highest authenticated sequence number for each SA from all cluster members. Ones the new master has this information it creates a new load sharing function for each SA. These functions only allow packets that have higher sequence number than already authenticated. Because the master does not perform IPsec processing the cluster slave members have all the replay protection information. If the master would participate into the IPsec processing there would be no way to survive from the master failure without compromising the replay protection.

In a situation were the master and at least on of the slaves fail simultaneously, the cluster has to drop all SAs, because it does not have enough information to recover without compromising the security.

Changing the mapping functions

When the state of the cluster changes in some way, for example a new member is added or a member fails, the cluster must update the load sharing function. The update must be made in a way that the mapping of an already seen sequence number does not change. This can be done by defining the new load sharing function in parts. When a new sharing scheme is taken into use the function is redefined starting from a sequence number larger than the largest seen by the master so far. For example the load sharing function could be defined as follows:

$$F(s) = \begin{cases} F1(s) & \text{,for } s \leq 100 \\ F2(s) & \text{,for } 100 < s < 500 \\ F3(s) & \text{,for } s \geq 500 \end{cases}$$

In this example packet with sequence number smaller or equal than 100 are mapped using function $F1$, packet with sequence numbers between 101 and 499 are mapped using function $F2$, and packet with sequence number larger or equal of 500 are mapped using function $F3$. The function $F1$ could for example map packet to slaves one, two, and three. The function $F2$ could represent a failure of a slave when packet are mapped to slaves one and two. Ones the slave three has rebooted it can rejoin the cluster and a new function $F3$ taken into use.

IPsec specification[2] defines the replay protection in a way that packets with sequence number smaller by the size of the replay window that the highest seen sequence number are automatically dropped. This allows the clustering implementation to forget the previous definitions of the load sharing functions after packet with larger sequence numbers are received. The required receive

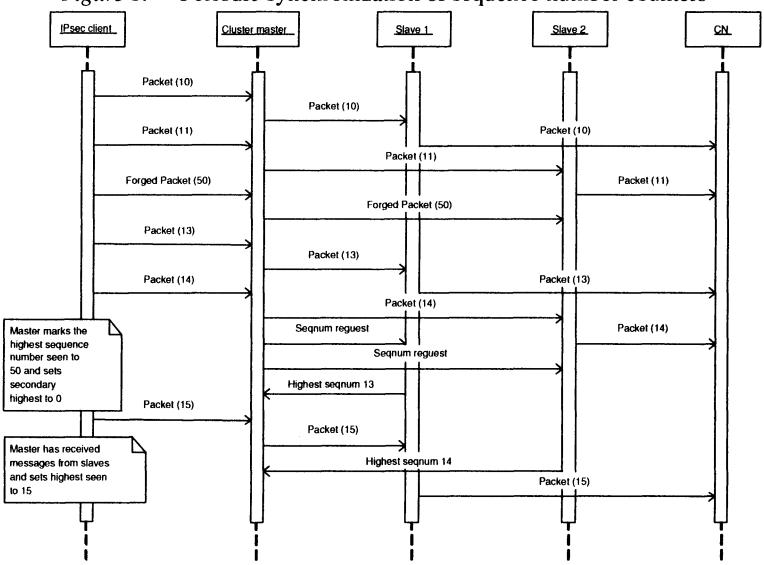
window size is 32, and recommended is 64. In the above example when the clustering has seen and authenticated a packet with sequence number larger than 164 it can forget the function definition $F1$. This feature allows the clustering to maintain only two function definition as long as it does not redefine the function before is has seen and authenticated packet with larger enough sequence number in order to forget the previous definition.

Replay protection information synchronization

In order to properly handle slave failure situation, the cluster needs to synchronize the replay protection information periodically. Synchronization has to be done in a way that it prevents potential denial of service attack that could be made using fake packets that have a very high sequence number. This attack is explained in more detail in section 3.

Figure 1 illustrates the sequence number synchronization. A message named "packet (xx)" is an IPsec protected packet with a sequence number of xx. At first the master receives two packets with sequence numbers 10 and 11. The master forwards these packets to slaves 1 and 2 and marks the highest sequence number seen to 11 ($s_h = 11$). Then the master receives a forged packet with sequence number 50 and forwards it to the slave 2. Now the highest seen sequence number is 50 ($s_h = 50$), even though the slave 2 dropped the packet because it was not authenticated correctly.

Figure 1. Periodic synchronization of sequence number counters



After the packet with sequence number 14, the master sends the periodic sequence number synchronization request to all slaves. When the master sends

the request it saves the highest seen sequence number ($s_s = 50$), and set the current value of the highest seen sequence number to zero ($s_h = 0$). Before all slaves have responded to the request the master has forwarded a packet with the sequence number 15 ($s_h = 15$) to the slave 1. After all slaves have replied to the request, the master checks the highest correctly authenticated sequence number from all slaves. Then the master sets the highest seen sequence number to the maximum of the current highest seen by the master ($s_h = 15$) and the highest authenticated by slaves ($s_a = 14$).

If the packet with sequence number 50 would have been correctly authenticated the slave 2 would have sent this in its sequence number synchronization reply, and the highest seen sequence number would have ended up to be 50.

The sequence number synchronization is done in this way to prevent a denial of service attack using forged sequence numbers. In this attack a malicious node sends fake packets to the cluster. These packets have a valid SPI value and a maximum sequence number. These packets cause the highest seen sequence number seen in master to be the same as in attack packets. If the cluster did not have a mechanism to roll back the highest seen sequence number a single packet would be enough to cause the value to remain high. This would cause a infinitely long delay when a slave fails, and the load sharing must be reassigned. Details how the slave failure situation is handled is explained in section 2.

3. ANALYSIS

Security association lifetimes

An IPsec security association has two different lifetimes: first is the time based lifetime defining how long an SA can be used, and the second is the traffic amount based lifetime defining how much data can be transferred using an SA.

Each cluster member can monitor the time based lifetime individually, and the only requirement is that cluster members keep their clocks synchronized. This can be done for example using the network time protocol[1].

The traffic amount based lifetime is more difficult to handle, because each cluster member that participates to the IPsec processing processes only a portion of the actual traffic. Fortunately traffic lifetime can be handled by being little over cautious. This is done in a way that each cluster member sends periodically the amount of traffic it has received to the cluster master. The master then send the total amount of traffic received back to each cluster member. In addition to this the cluster master also counts the traffic it forwards to the cluster members. If the traffic lifetime would be exceeded by adding the forwarded amount of traffic the master stops forwarding traffic of this SA and send request for data counters to all cluster members. Now the master knows

the actual amount of traffic received using this SA. If it does not exceed the traffic lifetime the master will continue forwarding traffic. Ones the lifetime would be again exceeded with the forwarded amount the master repeat the procedure. When the lifetime is actually exceeded the master will remove the SA from its database and optionally sends IKE notification to the IPsec client.

In a typical IPsec setup a new SA is negotiated when the old one is about to exceed its lifetimes, and that is why the cluster does not have to perform this kind of procedure in practice.

In a slave failure situation the master must relay on the amount it has forwarded to the clients and update the traffic counters according to it. This does not cause any security issues since the forwarded amount of traffic is always at least as large as the actual amount of traffic received using an SA.

Security

This section gives an analysis of the differences of the clustering method presented in this paper and a single node IPsec implementation. This analysis show that the security of the system presented is as good as single node IPsec implementation.

In a steady state the packet processing is logically done exactly as in a single node IPsec implementation as long as the load sharing function guarantees that all IPsec packets protected using the same SA with the same sequence number are forwarded to the same authentication context.

Because the changing of the load sharing function is done by defining a new function starting from a given sequence number there is no way that a packet could be forwarded using two different mapping functions.

The master failure situation does not generate any security weaknesses as long as only the master fails because other members of the cluster have the whole replay protection information available and the new master only forwards packets with larger sequence numbers that already received before the old master failed.

The slave failure situation is the only situation were there are a change to attack the cluster. In a situation were a slave fails a malicious node can perform a denial of service attack by sending fake IPsec packet that have very high sequence number. This attack is carried out by constantly sending these packets, and ones a slave fails the cluster master has forwarded a very high sequence number to that slave. Because the master does not know whether the packet was correctly authenticated it has to assume that it was, and only define a new load sharing function after this sequence number. This generates a situation were the cluster is unable to take a new load sharing function into use and the portion of the traffic previously handled by the failed node is discarded.

To make this attack less serious the cluster constantly synchronizes the replay protection information as specified in 2. Because of this synchronization the attacker must be actively sending packets when a slave fails. If it's not the highest seen sequence number is rolled back during the periodic synchronization.

There is a possibility to initiate a targeted denial of service attack against a single cluster node. This attack is performed by generating forged packets with valid SPI and high sequence number. When these packets are sent to the cluster they will end up to the same cluster node, since they have the same sequence number. When this kind of attack occurs the cluster will re-balance the load and the attack will automatically move to the next node. As long as the attack is ongoing the cluster will keep re-balancing the load over and over again. This will cause disruption to the cluster, but compared to single node implementation the situation is better since the whole cluster can not be taken down.

4. IMPLEMENTATION

We made an test implementation of the clustering model presented in this paper using an existing clustering implementation running on Linux. The existing clustering worked on a connection level in a way that each IPsec connection is assigned into a single cluster node. In a case of member failure the existing clustering requires a new IKE negotiation.

Load sharing function

The load sharing function $F(s) = id$ was implemented using a sharing table that contains 256 entries. Each of these entries contains a pointer to a member in the cluster. The table is filled in pseudo random way so that desired portion of the table entries point to a desired node. This way the packet forwarding code can just use last eight bits of the sequence number as an index into the table to get the id of the correct member to which to forward the packet.

The pseudo random function used to fill the sharing table needs not to be very good. Just a simple integer hash that divides sequence numbers in a pseudo random way works fine. The main point about the function is that there would not be long sequence of same ID in the share table to avoid bursts of traffic.

Figure 2 shows an example of a share table from a cluster that has three nodes that participate to the IPsec processing.

Figure 2. Example share table

1	2	1	3	1	2	1	3	1	2	1	3	1	2	1	3

Performance testing

The performance testing was done using four Pentium 4 1,8GHz machines as a cluster, two Pentium 4 2,4GHz as IPsec clients and own Pentium 3 machine as a server that runs a tool called iperf. The iperf is a tool that measures network throughput using UDP or TCP.

Throughput using the single node IPsec was measured to be 52Mbit/s in both directions. At this speed the processor load of the IPsec node handling the IPsec client was 100%, and the processor load of the IPsec client was about 40%. The machines used as an IPsec client and gateway were identical. The difference between processor loads is caused be the different software architecture of the Linux and Windows IPsec implementations.

The performance of the implementation made in this paper was found to highly depend on the amount of simultaneous TCP connections used to measure the performance. Analysis of the implementation showed that this effect was caused by reordering of packets when they were put through the cluster. Reordering caused TCP to drop window size and triggered the TCP fast retransmission that wasted the bandwidth. Overall the TCP behaves badly when reordering occurred. When the amount of concurrent TCP connections was increased the reordering effect became smaller for a single TCP stream and the performance improved significantly. More details how TCP reacts on reordering of packets is explained in [7].

The reordering effect sets an upper limit to throughput of a single TCP stream in this kind of cluster. When this limit is reached the TCP starts to suffer from the reordering. Estimation about this limit can be calculated using the network latency between cluster members. The amount of reordering can be calculated from the network throughput and extra delay in the route through a slave. The amount means that how many packets can be passed through the master before a packet sent through a slave arrives to the receiver.

In order to remove the re-ordering problem the cluster could be modified to route all packets through a ordering node that would use upper layer protocol information such as TCP sequence numbers to restore the right order. In principle this node could be the cluster master. However routing packets twice through the master would degrees the performance of the cluster.

Table 1 summarizes the results of the performance measurements using a single TCP stream. Upload means the traffic throughput from the IPsec client to the server, and the download means the traffic throughput from the server to the IPsec client. In a two member setup the traffic was divided in a way that the master handled 40% of the traffic and slave handled 60%. In a three member setup the master did not handle traffic at all and both slaves handled 50% each. In four member setup the master handled 1% of the traffic and each slave handled 33%.

Table 1. Results of the performance measurements

	Download	Upload
Single	53 Mbit/s	53 Mbit/s
1 member	47 Mbit/s	46 Mbit/s
2 members	41 Mbit/s	28 Mbit/s
3 members	68 Mbit/s	43 Mbit/s
4 members	30 Mbit/s	43 Mbit/s

As the table shows the three member setup provided the best performance of the new cluster. This was caused by the fact that the master did not handle the traffic at all and traffic routes through both slaves were almost equal in term of latency. In two and four member setups the traffic going through the master caused more reordering than in three member setup, because the route through the master was faster than the route through the slaves. Later measurements showed that even in a setup were the master does not handle traffic at all reordering seems to happen, because the throughput increase in this setup was also significant when traffic was divided to multiple TCP streams.

Fail-over testing

The fail-over testing was done with a cluster of two machines.

The first test measured how a failure of a slave affects the traffic throughput. The slave was removed from the cluster when the IPsec client was downloading through the cluster. The throughput was measured as an average in 5 second intervals using 5 simultaneous TCP streams. The traffic was divided between cluster members in a way that the master handled 40% of the traffic and the slave handled 60%.

Throughput in the first interval before the slave removal was 70Mbit/s. In the second interval were the slave was removed the throughput was 42Mbit/s, and in the last interval were the master handled the whole traffic the throughput was 47Mbit/s.

When the slave was removed it took 600ms from the master to notice that slave was removed. During this time slave's portion of the traffic was lost. After the master noticed that the slave was missing it re-assigned the load share, and the packet loss ended.

As can be seen from the throughput in interval were the slave was removed the impact was small. The throughput was only 5% smaller than it was using single node with no packet loss. Of course if the measurement interval would be smaller the affect would be larger. The 5 second interval gives impression about what kind of impact a slave failure would cause in a real situation. If the total traffic through the cluster is less than 47Mbit/s when a slave fails there is

no real affect to clients. If the total throughput is larger then all connections will be slowed down after the failure.

5. CONCLUSIONS

The IPsec clustering method presented in this paper provides high availability and scalability while maintaining all security features of the IPsec.

If the architecture presented in this paper would be taken into commercial use, more extensive test should be made to see how the cluster reacts to different kind of load peaks, and how the SA synchronization work in setup were the amount of SAs is in order of thousands. For use in very high speed connections such as site-to-site VPNs the packet re-ordering issues must be solved.

References

[1] David L. Mills: Network Time Protocol (Version 3). Request For Comments 1305, March 1992.

[2] Randall Atkinson, Stephen Kent: Security Architecture for IP. Request For Comments 2401, November 1998.

[3] Randall Atkinson, Stephen Kent: IP authentication header (AH). Request For Comments 2402, November 1998.

[4] Randall Atkinson, Stephen Kent: IP Encapsulating Security Payload (ESP). Request For Comments 2406, November 1998.

[5] Dave Carrel, Dan Harkins: The Internet Key Exchange (IKE). Request For Comments 2409, November 1998.

[6] Steven M. Bellovin: Problem Areas for the IP Security Protocols. Proceedings of the Sixth Usenix UNIX Security Symposium, July 1996.

[7] Ethan Blanton, Mark Allman: On making TCP more robust to packet reordering. ACM Computer Communication Review, 32(1), January 2002.