**Placement of security devices in an enterprise network: Analysis and Implementation**

**Introduction**

Network security consists of the provisions and policies adopted by a network administrator to prevent and monitor unauthorized access, misuse, modification, or denial of a computer network and network-accessible resources. Now a days the firms are more concerned with their security along with their products. These are becoming more fine-grained due to extensive use of various network services and newly evolving security threats. In addition to this, most of the organizations not only emphasizing on enforcement of the security constraints but also requiring satisfaction of different business constraints like, cost, usability demand, etc. The implementation of strong defence in a network exploring different security design alternatives as well as resolving the contention between the security and business constraints is an important but challenging problem.

Usually, the organizational security requirements cover: (i) connectivity requirement that defines the service flows between various network devices; and (ii) isolation requirement that defines various isolation patterns as combination of different security devices (firewall, IPSec, IDS, and NAT etc.) and their relative order based on the device capability. An isolation pattern signifies the level of security resistance, for example, traffic filtering (firewall), IPSec based encryption, payload traffic inspection (IDS), hiding traffic source identity (NAT) and/or combination of these constraints.

On the other hand, the firms’ business constraints include usability and cost. Installation of different security devices significantly affect these constraints. For example, implementing both IPSec and IDS instead of firewall might cause some usable application inaccessible from a node, thereby reducing the usability. The major requirement is to find usable security configuration exploring different security design alternatives (isolation levels) that increases the usability without significant degradation of the overall network security. At the same time, it is required to find best security isolation in affordable cost. Therefore, a careful balancing between the security and business constraints is required to determine correct and cost-effective security configurations.

Here, only the placement of security devices between nodes is defined using four isolation patterns - Firewall, IPSec, IDS and NAT – to meet the security needs in cost effective manner. In other words, optimized placement of security devices with minimum cost requirements is described here. Now, let us understand the isolation patterns and working of them (firewall, IPSec, IDS, and NAT).

**Isolation Patterns**

Firewall

Firewall is a barrier to keep destructive forces away from your property. In fact, that’s why it’s called a firewall. Its job is similar to a physical firewall that keeps a fire from spreading from one area to the next. A firewall is simply a program or hardware device that filters the information coming through the Internet connection into your private network or computer system. If an incoming packet of information is flagged by the filters, it is not allowed through. A company can set up rules like this for FTP servers, Web servers, Telnet servers and so on. In addition, the company can control how employees connect to Web sites, whether files are allowed to leave the company over the network and so on. A firewall gives a company tremendous control over how people use the network.

Firewalls use one or more of three methods to control traffic flowing in and out of the network:

* Packet filtering - Packets (small chunks of data) are analysed against a set of *filters*. Packets that make it through the filters are sent to the requesting system and all others are discarded.
* Proxy service - Information from the Internet is retrieved by the firewall and then sent to the requesting system and vice versa.
* Stateful inspection - A newer method that doesn't examine the contents of each packet but instead compares certain key parts of the packet to a database of trusted information. Information traveling from inside the firewall to the outside is monitored for specific defining characteristics, then incoming information is compared to these characteristics. If the comparison yields a reasonable match, the information is allowed through. Otherwise it is discarded.

Firewalls are customizable. This means that one can add or remove filters based on several conditions. Some of these are:

**IP addresses** - Each machine on the Internet is assigned a unique address called an [IP address](http://computer.howstuffworks.com/internet/basics/question549.htm). IP addresses are 32-bit numbers, normally expressed as four "octets" in a "dotted decimal number." For example, if a certain IP address outside the company is reading too many files from a server, the firewall can block all traffic to or from that IP address.

**Domain names** - Because it is hard to remember the string of numbers that make up an IP address, and because IP addresses sometimes need to change, all servers on the Internet also have human-readable names, called [domain names](http://computer.howstuffworks.com/dns.htm). For example, it is easier for most of us to remember www.google.com than it is to remember 74.125.236.83. A company might block all access to certain domain names, or allow access only to specific domain names.

**Ports** - Any server machine makes its services available to the Internet using numbered ports, one for each service that is available on the server. For example, if a server machine is running a Web (HTTP) server and an FTP server, the Web server would typically be available on port 80, and the FTP server would be available on port 21. A company might block port 21 access on all machines but one inside the company.

**Specific words and phrases** - This can be anything. The firewall will search thoroughly each packet of information for an exact match of the text listed in the filter. For example, one could instruct the firewall to block any packet with the word "X-rated" in it. The key here is that it has to be an exact match. The "X-rated" filter would not catch "X rated" (no hyphen). But one can include as many words, phrases and variations of them as per need.

With a hardware firewall, the firewall unit itself is normally the gateway. Hardware firewalls are incredibly secure and not very expensive.

IPSec

Internet Protocol Security (IPSec) is a protocol suite for securing Internet Protocol (IP) communications by authenticating and encrypting each IP packet of a communication session. IPSec includes protocols for establishing mutual authentication between agents at the beginning of the session and negotiation of cryptographic keys to be used during the session. IPSec can be used in protecting data flows between a pair of hosts (host-to-host), between a pair of security gateways (network-to-network), or between a security gateway and a host (network-to-host).

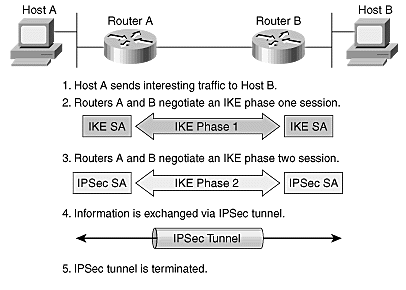
IPSec consists of two sub-protocols which provide the instructions a VPN needs to secure its packets:

Encapsulated Security Payload (ESP) encrypts the packet's payload (the data it's transporting) with a symmetric key.

Authentication Header (AH) uses a hashing operation on the packet header to help hide certain packet information (like the sender's identity) until it gets to its destination.

Networked devices can use IPSec in one of two encryption modes. In transport mode, devices encrypt the data traveling between them. In tunnel mode, the devices build a virtual tunnel between two networks. As you might guess, VPNs use IPSec in tunnel mode with IPSec ESP and IPSec AH working together.

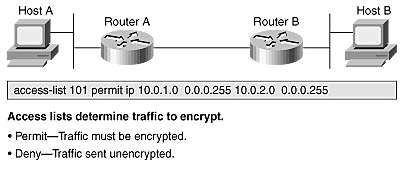
IPSec involves many component technologies and encryption methods. Yet IPSec's operation can be broken down into five main steps. The steps can be visualized by the following diagram:-



The five steps are further described in detail as follows:

Step 1: Defining Interesting Traffic

What type of traffic is to be considered in a specific way is part of formulating a security policy for use of a VPN. The policy is then implemented in the configuration interface for each particular IPSec peer. Access lists are used to determine the traffic to encrypt. The access lists are assigned to a crypto policy such that permit statements indicate that the selected traffic must be encrypted, and deny statements can be used to indicate that the selected traffic must be sent unencrypted. When the particular traffic is generated or transits the IPSec client, the client initiates the next step in the process, negotiating an IKE (Internet Key exchange) phase one exchange.



Step 2: IKE Phase One

The basic purpose of IKE phase one is to authenticate the IPSec peers and to set up a secure channel between the peers to enable IKE exchanges. IKE phase one performs the following functions:

* Authenticates and protects the identities of the IPSec peers.
* Negotiates a matching IKE SA (Security Association) policy between peers to protect the IKE exchange.
* Performs an authenticated Diffie-Hellman exchange with the end result of having matching shared secret keys.
* Sets up a secure tunnel to negotiate IKE phase two parameters.

IKE phase one occurs in two modes: Main mode and Aggressive mode

Main Mode

Main mode has three two-way exchanges between the initiator and receiver.

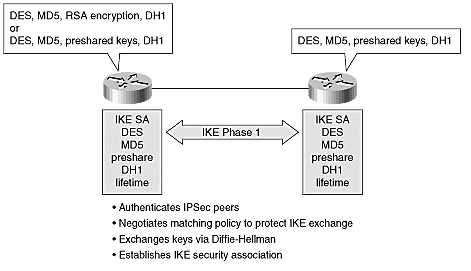
First exchange—the algorithms and hashes used to secure the IKE communications are agreed upon in matching IKE SAs in each peer.

Second exchange—this exchange uses a Diffie-Hellman exchange to generate shared secret keying material used to generate shared secret keys and to pass nonce, which are random numbers sent to the other party, signed, and returned to prove their identity.

Third exchange—this exchange verifies the other side's identity. The identity value is the IPSec peer's IP address in encrypted form. The main outcome of main mode is matching IKE SAs between peers to provide a protected pipe for subsequent protected ISAKMP exchanges between the IKE peers.

Aggressive Mode

In the aggressive mode, fewer exchanges are done and with fewer packets. In the first exchange, almost everything is squeezed into the proposed IKE SA values, the Diffie-Hellman public key, a nonce that the other party signs, and an identity packet, which can be used to verify the initiator's identity through a third party. The receiver sends everything back that is needed to complete the exchange. The only thing left is for the initiator to confirm the exchange. The weakness of using the aggressive mode is that both sides have exchanged information before there is a secure channel. Therefore, it is possible to sniff the wire and discover who formed the new SA. However, aggressive mode is faster than main mode.



Step 3: IKE Phase Two

The purpose of IKE phase two is to negotiate IPSec SAs to set up the IPSec tunnel. IKE phase two performs the following functions:

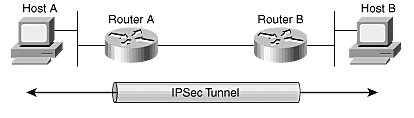
* Negotiates IPSec SA parameters protected by an existing IKE SA
* Establishes IPSec security associations
* Periodically renegotiates IPSec SAs to ensure security
* Optionally performs an additional Diffie-Hellman exchange

IKE phase 2 has one mode, called quick mode. Quick mode occurs after IKE has established the secure tunnel in phase one. It negotiates a shared IPSec policy, derives shared secret keying material used for the IPSec security algorithms, and establishes IPSec SAs. Quick mode exchanges nonce (a number or bit string used only once) that provide replay protection. The nonce are used to generate new shared secret key material and prevent replay attacks from generating bogus SAs.

Quick mode is also used to renegotiate a new IPSec SA when the IPSec SA lifetime expires. Base quick mode is used to refresh the keying material used to create the shared secret key based on the keying material derived from the Diffie-Hellman exchange in phase one.

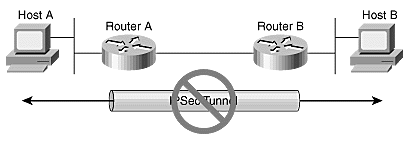
Step 4: IPSec Encrypted Tunnel

After IKE phase two is complete and quick mode has established IPSec SAs, information is exchanged by an IPSec tunnel. Packets are encrypted and decrypted using the encryption specified in the IPSec SA.



Step 5: Tunnel Termination

IPSec SAs terminate through deletion or by timing out. An SA can time out when a specified number of seconds have elapsed or when a specified number of bytes have passed through the tunnel. When the SAs terminate, the keys are also discarded. When subsequent IPSec SAs are needed for a flow, IKE performs a new phase two and, if necessary, a new phase one negotiation. A successful negotiation results in new SAs and new keys. New SAs can be established before the existing SAs expire so that a given flow can continue uninterrupted.



IDS

An intrusion detection system (IDS) is a device or software application that monitors network or system activities for malicious activities or policy violations and produces reports to a management station.

In this case, computer or network is the one using an alarm system. So when a computer hacker is trying to find ways into network or computer, it can provide user with an alert (via e-mail or some means through its own management software) that something or someone is attempting to break in or has already broken in.

The intruder from this perspective can be a live malicious hacker, or it could be an Internet worm or malware intended to exploit your computer's vulnerabilities in order to spread itself.

There are two basic types of IDS: host-based and network-based. Host-based IDS are designed to provide IDS functions for the protection of the host where it resides. In most cases, a host-based IDS also runs as a service and/or and application within the host it is protecting. Network-based IDS are designed to watch network traffic and alert of any malicious activities it sees on the network port it is monitoring. Network-based IDS typically run on a separate host or dedicated appliance designed to perform this function.

Combining these two solutions to protect user’s computers can be part of a layered network and computer protection plan.

Computer intrusion detection systems all have these basic components:

Sensor

* + Activity or packet capture engine
  + Behavioural or signature detection engine

Backend

* + Event recording database
  + Alerting engine

Frontend

* + User interface
  + Command & control

These components could be physically separate or could all reside on the same host. There could be one or more sensors, and in larger setups, one or more backends.

**Sensor**

The sensor is the primary component for detecting hacking activities on the computer or on the network. It has a packet capture and activity capture engine to help it get access to activities efficiently and quickly. Most IDS have a signature database which it uses to determine the presence of an event, and more advanced IDS have behavioural activity detection to determine malicious behaviour. The good thing about the latter is that it allows the sensor to detect what is called a "[zero day attack](http://en.wikipedia.org/wiki/Zero_day_attack)"- compared to signature-based detection which can only detect activities that have already been seen in the wild before.

**Backend**

The backend is where actual alerting and recording happens. This allows the sensor to focus on the function of detection for efficiency and speed. The backend collects all events detected by the sensors, and is the component that does the function of alerting. Alerting can come in the form of:

* Log. Log to the database.
* E-mail. The alert can be sent to one or more recipients.
* [SNMP](http://en.wikipedia.org/wiki/Simple_Network_Management_Protocol) trap. There are applications out there that can collect SNMP traps of various kinds. The backend can send an SNMP trap to a SNMP trap collection and viewer (e.g. [HP Open view](http://en.wikipedia.org/wiki/HP_Openview)).
* Block. Some advance IDS have the ability to cause a connection block (i.e. cause a connection reset--[TCP reset](http://en.wikipedia.org/wiki/TCP_reset_attack#TCP_resets)--between the hacker's computer and the target)
* Display. The alert can be sent to a console that shows the various events that the sensor is detecting.

Aside from providing the repository and the alerting mechanism, the backend provides the IDS setup and configuration storage.

**Frontend**

The frontend is the IDS’s direct user interface. From the frontend, the user can do the following:

* View events that the sensor has detected
* Setup IDS configuration
* Update signature database and behavioural detection engine
* Update sensor and other parts of the IDS

The components of an IDS work together as a whole to provide an early warning or post intrusion alerting system. In explaining how things work, we will use a network-based IDS as a point of reference.

On network-based IDS, the sensor will typically run as a separate host, and the backend and frontend will generally run on another host. In a bigger environment, there will be more than one sensor, and the backend and frontend are on a separate host. Summarizing the whole:-

* **Sensors Detect and Report**
* Backend Collect and Alert
* Frontend- Command and Control

NAT

Network address translation (NAT) is a function by which IP addresses within a packet are replaced with different IP addresses. This function is most commonly performed by either routers or firewalls.

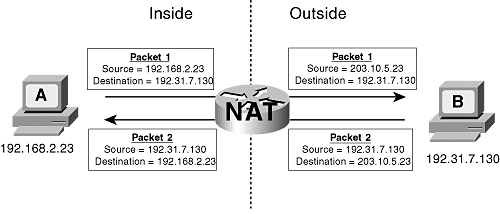


Figure depicts a simple NAT function. Device A has an IP address that belongs to the private range specified by RFC 1918, whereas device B has a public IP address. When device A sends a packet to device B, the packet passes through a router that is running NAT. The NAT replaces device A's private address (192.168.2.23) in the source address field with a public address (203.10.5.23) that can be routed across the Internet, and forwards the packet. When device B sends a reply to device A, the destination address of the packet is 203.10.5.23. This packet again passes through the NAT router, and the destination address is replaced with device A's private address.

NAT is transparent to the end systems involved in the translation. In Figure, device A knows only that its IP address is 192.168.2.23; it is unaware of the 203.10.5.23 address. Device B, on the other hand, thinks the address of device A is 203.10.5.23; it knows nothing about the 192.168.2.23 address. That address is "hidden" from device B.

NAT can hide addresses in both directions. In Figure, NAT is performed on the addresses of both device A and device B. Device A thinks device B's address is 172.16.80.91, when in fact device B's real address is 192.31.7.130.

When an inside device sends a packet to the Internet, the NAT dynamically selects a public address from the inside global address pool and maps it to the device's inside local address. This mapping is entered into the NAT table. For instance, figure below shows that three inside devices from the enterprise in diagram —10.1.1.1.20, 10.1.197.64, and 10.1.63.148— have sent packets through the NAT. Three addresses from the IG pool—205.110.96.2, 205.110.96.3, and 205.110.96.1, respectively—have been mapped to the IL addresses.

NATrouter#show ip nat translations

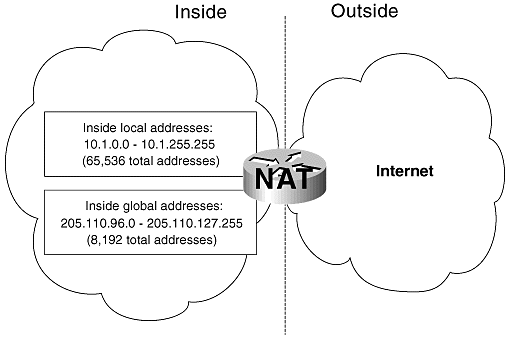
Pro Inside global Inside local Outside local Outside global

--- 205.110.96.2 10.1.1.20 --- ---

--- 205.110.96.3 10.1.197.64 --- ---

--- 205.110.96.1 10.1.63.148 --- ---

NAT router#



Network administrators can define isolation patterns considering different combination of security devices (primitive isolations). They can objectively exclude some of the device combinations according to requirements or domain knowledge. Administrators can also define the capability of different isolation patterns by providing relative order of these patterns, *total* or *partial*. Now, we formalize the network isolation as a set of rules, *{ir*1,…,*irn}*, where, each isolation rule *iri* is formally represented as follows:

*irk* : (*src* = *i*) *∧* (*dst* = *j*) *∧* (*service* = *g*) *⇒*

The decision variable, = 1 indicates that the corresponding *k*th isolation pattern is required to be deployed between node pair (*i; j*) under service *g*. Here, *k* represents the relative order of the isolation pattern. A possible set of primitive isolation patterns with their relative order is presented in Table 1. It shows that *k* = 1 for “firewall Deny” and *k* = 3 for “IPSec based authentication”, etc. So, if = 1, then the service *g* must be denied between the node pair (*i, j*) through firewall.

Table 1: Isolation Patterns

|  |  |  |  |
| --- | --- | --- | --- |
| **Isolation Order** | **Isolation Pattern** | **Decision Variable** | **Symbol Used** |
| **1** | Firewall Deny |  | C:\Users\!..Swapnil..Aryan..!\Desktop\Finalize\Isolation Patterns Images\Firewall.jpeg |
| **2** | IPSec Encryption |  | C:\Users\!..Swapnil..Aryan..!\Desktop\Finalize\Isolation Patterns Images\IPSec.jpeg |
| **3** | Payload Inspection (IDS) |  | C:\Users\!..Swapnil..Aryan..!\Desktop\Finalize\Isolation Patterns Images\IDS.jpg |
| **4** | Source Identity Hiding (NAT) |  | C:\Users\!..Swapnil..Aryan..!\Desktop\Finalize\Isolation Patterns Images\NAT.jpg |

C:\Users\!..Swapnil..Aryan..!\Desktop\Finalize\Isolation Patterns Images\router.jpgC:\Users\!..Swapnil..Aryan..!\Desktop\Finalize\Isolation Patterns Images\Node.jpegOther symbols used for client is and router is

**Implementation Setup**

Here for our algorithm the platform can be either Windows or Linux, but we have done it in Windows environment. The maximum number of nodes that the algorithm can evaluate efficiently is 100. The algorithm is implemented in ‘C’ language and the final visualized output is given by the Graphviz application. Considering the combinations of 4 isolation patterns as discussed above we are generating an optimised placement of security devices and their respective graphs drawn by the Graphviz application. The details of the IDE (Integrated Development Environment) used for C, i.e. Code::Blocks and the visualization done by Graphviz is explained below.

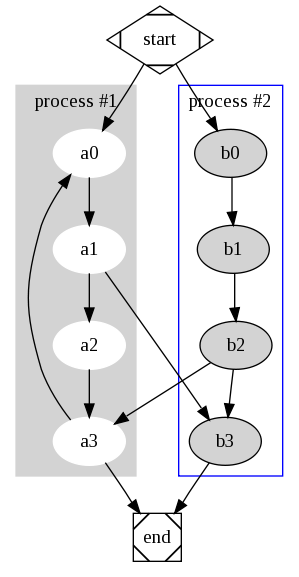
**Software Used**

Before we move on with the procedure of placement of security devices, let us look at the various tools used for the designing output of the required network.

We need a C compiler, gcc 4.3.8 version. **Code::Blocks** is a free C, C++ and Fortran IDE built to meet the most demanding needs of its users. It is designed to be very extensible and fully configurable. Here we have used this tool for writing, editing and compiling our required C program. The result produced is passed to Graphviz application for the visualization of the network along with their security devices placed between them.

GVEdit Graph File Editor for Graphvizversion: 1.02 Graphviz: version 2.38.0.

**Graphviz** is open source graph visualization software. Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. It has important applications in networking, bioinformatics, software engineering, database and web design, machine learning, and in visual interfaces for other technical domains. Below is the sample example and the features of Graphviz.

Features

The Graphviz layout programs take descriptions of graphs in a simple text language, and make diagrams in useful formats, such as images and SVG for web pages; PDF or Postscript for inclusion in other documents; or display in an interactive graph browser.  Graphviz has many useful features for concrete diagrams, such as options for colors, fonts, tabular node layouts, line styles, hyperlinks, and custom shapes.

Roadmap

**dot -**"hierarchical" or layered drawings of directed graphs. This is the default tool to use if edges have directionality.

**neato -**"spring model'' layouts.  This is the default tool to use if the graph is not too large (about 100 nodes) and you don't know anything else about it. Neato attempts to minimize a global energy function, which is equivalent to statistical multi-dimensional scaling.

**fdp -**"spring model'' layouts similar to those of neato, but does this by reducing forces rather than working with energy.

**sfdp -**multiscale version of fdp for the layout of large graphs.

**twopi -**radial layouts, after Graham Wills 97. Nodes are placed on concentric circles depending their distance from a given root node.

**circo -** circular layout, after Six and Tollis 99, Kauffman and Wiese 02. This is suitable for certain diagrams of multiple cyclic structures, such as certain telecommunications networks.

Here we, have only used the dot model and implemented the required diagram.

**Placement of Security Devices**

Here, we consider a dummy network topology, organizational security (connectivity and isolation) requirements and business constraints (usability and cost) as input; and then synthesizes the correct and optimal security configurations that maximizes isolation while satisfying the requirements.

Here the algorithm generates a correct and cost-effective network security configuration (isolation patterns between each pair of nodes) using constraint satisfaction checking. However, the optimal placement of security devices in the network may not be determined based on the isolation results. This is because of the fact that there may have similar types of isolation (say, firewall) between a source *i* and multiple destinations *j*, which signifies to deploy multiple firewalls between all such node pairs. This placement might not be optimal as in that case, a single firewall might be sufficient to place at the source *i*. Therefore, given isolation patterns for each pair of nodes in the network, it is required to find minimum number of devices for enforcing the security constraints. We need to present a procedural approach for determining optimal placement of the security devices.

Isolation Equivalence Zone Creation: This process logically combines all nodes, *j* with similar isolation configuration with respect to a specific node *i* into a single network zone.

Node Isolation Equivalence Zone: An *Isolation Equivalence Zone*, is defined as a logical collection of nodes which have same isolation value with respect to a node *i* under isolation pattern *k*.

An isolation zone with respect to a node *i* under isolation pattern *k* is represented as follows:

= UjєN {j| = 1}

We procedurally create the isolation equivalence zone with respect to each node *i* in the network.

Now, multiple isolation zones may contain overlapping member nodes. Thus, these zones should be further combined to group the nodes optimally in terms of isolations. This is done by creating different isolation partite classes based on the similarity in zone members.

Isolation Partite Class: An *Isolation Partite Class,*  is defined as logical collection of isolation zones w.r.t a zone such that + ⊆ + and the node *x* does not belong to. Here, in +, the plus sign indicates that the zones cannot be empty if they are going to take part in the formation of the isolation partite class.

The partite class is formally represented as follows:

= Ux{( |) ⊆ ⋀ ¬(x ∈ )}

This process signifies reducing the redundancies between the isolation zones. Each *isolation class* has two parties; the suffix sequence (here, *i* ∈ *N* and *x* ∈ *N*) of the class, the left partite(nodes that will belong in the left group of the isolation device); and the participating zones of class represent right partite.

It is to be noted that, if the node, *x* belongs to the corresponding *determine* isolation zone, , then x is not included in the class, .

This is because of the fact that there may exist an isolation (of same pattern *k*) between the node *x* and the other member nodes of the zone, . After creating different partite classes, we procedurally determine the security device placement between different zones under the partite classes. The pseudo code of this process is presented in **Algorithm 2.**

Before the implementation of algorithm we need to know the input and output patterns. Input pattern is of .csv extension files, i.e. comma separated values and the output file is of .txt extension, i.e. text file.  
The output file is in the format which is read by graphviz application and a corresponding graph according to the nodes, zones, classes and isolation patterns is drawn for better visualization.

In input file, i.e. .csv file we enter the zone number and the number of nodes in it with varying isolation patterns. For example, suppose we have two zones Z1 and Z2 having 3,4,5 and 4,5 as their corresponding nodes between which isolation patterns 2,3 and 1,2 are required respectively. So, in .csv file we write them as 1,2,3,0,3,4,5,-1 and 2,4,5,0,4,5,-1. Here, 0 indicates that the following digits are meant to be read as nodes and -1 is the termination of a single line (i.e. details of a zone is complete). At the end of file we put a $ sign to indicate the end of file.

**Algorithm 1: Class creation**

1. Select initial zone and count its number of nodes in it.
2. Check the other zones, except the initial one, which have less or equal ‘number of nodes’ as the initial one.
3. Once step 2 is satisfied, check if the ‘nodes’ in initial one matches exactly with the next corresponding zones.
4. If step 3 is satisfied check for their similar isolation patterns and those which can be overlapped or discarded and then combine the zones together in the same class.
5. Continue this till all zones are taken into consideration.
6. Create next class and goto step 1.

**Algorithm 2: Deriving Security Device Placement**

1. Begin
2. Create a sorted list (*,* *,…,* ) based on descending order of || such that ≡ .
3. Select first element from.
4. Place a level security device between the left partite and the right partite, zones in. In this the level security can be anything ranging from single isolation pattern to multiple isolation pattern. To simplify , x can be anything like 1,2,3,4 or 1234 or combinations of these four isolation patterns which is described in detail in the upcoming examples.
5. Remove all the zones {, ,..} from (*,* *,…,* )
6. Remove all the *empty sets* from.
7. GotoStep 2 if is *non-empty.*
8. End.

We first create a sorted list S′ of partite classes based on decreasing order of class size, ||, the total number of distinct nodes in all participating zones in a class. Then, we procedurally select each element class, si (each element represents a partite class, Sx1x2..) in sequence from S′ and place a security device between the two partites of that class. After the corresponding device placement, we remove from all

the classes, the zones that are already considered. This process is applied continually until the list S′ becomes an empty set. In this way, for each isolation pattern k, our framework derives the optimal device placement in the network. Let us see few examples for the above procedure.

**Example 1**

This is the .csv file for the following example.

Let us implement an example for single isolation pattern where *k=1*. Consider, the following isolation equivalent zones under *kth* isolation pattern (determined based on the isolation variables) for a network of 6 nodes, (1,2*,* 3*,* 4*,* 5*,* and 6):

= {2,3,4}; = {3,4}; = {4,5}; = {2,3}; = {2,3}; = {3}

The associated isolation classes for these zones are as follows:

= {Z1,Z5,Z6}; = {Z2,Z6}; = {Z3}; ={Z4,Z5,Z6}; = {Z5,Z4,Z6}; = {Z6}.

Now, the different iterations of the device placement procedure are shown as follows:

Iteration1: S′(S156, S456 S546, S26, S3, S6); (1, 5, 6) △ {Z1 Z5 Z6} ∼ (2, 3, 4)

Iteration2: S′(S2 = {Z2}; S3 = {Z3}; S4 = {Z4}; S5 ={Z4}); (2) △ Z2 ∼ (3, 4)

Iteration3: S3 = {Z3}; S4 = {Z4}; S5 = {Z4}); (3) △ Z3 ∼ (4, 5)

Iteration4: S4 = {Z4}; S5 = {Z4}); (4) △ Z4 ∼ (2, 3)

Here, x △ y denotes that the security device need to be placed between the nodes/node groups x and y.

Following is the output of the above example from the graphviz application. It shows all the connectivity and security device placement of the required pattern. The Graphviz file is: 

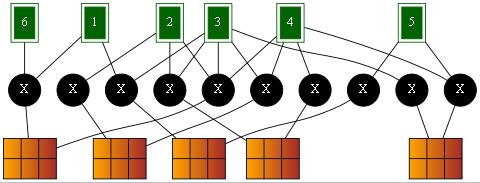


Figure 1: showing the detailed connectivity and isolation pattern of example 1.

**Example 2**

**** This is the .csv file for the following example.

Let us implement another example for single isolation pattern where *k=2 i.e*. IPSec. Consider, the following isolation equivalent zones under *kth* isolation pattern (determined based on the isolation variables) for a network of 4 nodes, (1,2*,* 3 *and* 4):

= {3, 4}; = {4}; = {1}; = {2, 3};

The associated isolation classes for these zones are as follows:

= {Z1, Z2}; = {Z2}; = {Z3}; = {Z4};

Now, the different iterations of the device placement procedure are shown as follows:

Iteration1: S′(S12, S4, S3, S2); (1, 2) △ {Z1,Z2 } ∼ (3, 4)

Iteration2: S′(S4, S3); (4) △ {Z4 } ∼ (2,3)

Iteration3: S′( S3); (3) △ {Z3 } ∼ (3, 4)

Here, x △ y denotes that the security device need to be placed between the nodes/node groups x and y.

Following is the output of the above example from the graphviz application. It shows all the connectivity and security device placement of the required pattern. IPSec tunnel is required to be placed between the nodes to carry out the encryption and decryption process.

The corresponding Graphviz text file is :

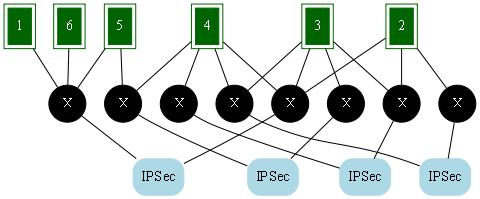


Figure 2: showing the detailed connectivity and isolation pattern of example 2.

**Example 3**

 This is the .csv file for the following example.

Let us implement an example for multiple isolation pattern where k=1, 2, 3 or their combinations i.e. firewall, IPSec and IDS. Consider, the following isolation equivalent zones under the given isolation pattern (determined based on the isolation variables) for a network of 4 nodes, (1, 2, 3 and 4):

= {3, 4}; i.e. firewall, IPSec and IDS is placed between node 1 and node 3 and 4.

= {4}; i.e. IPSec and IDS is placed between node 2 and node 4.

= {1}; i.e. only firewall is placed between node 3 and node 1.

= {2, 3}; i.e. firewall as well as IPSec is required to place between node 4 and node 2 and 3.

Here, the zones which need a firewall as well as IPSec to be placed between the nodes are only allowed to have IPSec between them as IPSec does the work of firewall so there is no need of putting both the security devices. The associated isolation classes for these zones after the modification of isolation patterns of zones are as follows:

= {Z1, Z2}; = {Z2}; = {Z3}; = {Z4};

Now, the different iterations of the device placement procedure are shown as

Iteration1: S′(S12, S4, S3, S2); (1, 2) △ {Z1,Z2 } ∼ (3, 4)

Iteration2: S′(S4, S3); (4) △ {Z4 } ∼ (2,3)

Iteration3: S′( S3); (3) △ {Z3 } ∼ (3, 4)

Here, x △ y are in their usual form. Following is the output of the above example from the graphviz application. The corresponding Graphviz text file is : 

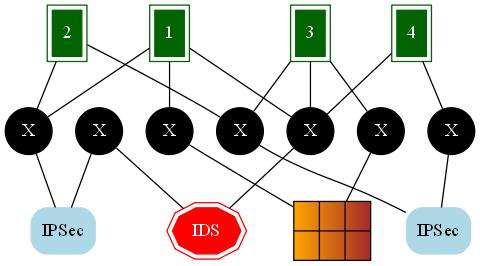
****

Figure 3: showing the detailed connectivity and isolation pattern of example 3.

**Example 4**

This is the .csv file for the following example.

Let us implement another example of multiple isolation patterns between various nodes. Keeping the data same as was in Example 3, here we just modify the isolation patterns and look the results.

So, we have the following isolation equivalent zones under the given isolation pattern (determined based on the isolation variables) for a network of 4 nodes, (1, 2, 3 and 4):

= {3, 4}; = {4}; = {1}; = {2, 3};

The explanation for discarding of security devices according to their functions remains same as Example 3. The associated isolation classes for these zones after the modification of isolation patterns of zones are as follows:

= {Z1, Z2}; = {Z2}; = {Z3}; = {Z4};

Now, the different iterations of the device placement procedure are shown as

Iteration1: S′(S12, S4, S3, S2); (1, 2) △ {Z1,Z2 } ∼ (3, 4)

Iteration2: S′(S4, S3); (4) △ {Z4 } ∼ (2,3)

Iteration3: S′( S3); (3) △ {Z3 } ∼ (3, 4)

Here, x △ y are in their usual form. Following is the output of the above example from the graphviz application and its corresponding Graphviz text file is : 

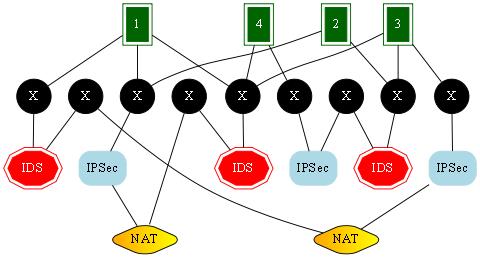


Figure 4: showing the detailed connectivity and isolation pattern of example 4.

Thus, Graphviz application has proved helpful in visualizing the nodes and their respective connectivity with the other nodes in a given network. The time required to execute each of the above diagrams is represented as follows in a table. The time in the 3rd column is the time taken to generate the output as well as the time required to generate the text file for the Graphviz application to generate the graphs.

**Execution Time**

The following table (table 2) describes the execution time taken by the algorithm for each of the four examples that we have considered above:

Table 2: Execution Time

|  |  |  |  |
| --- | --- | --- | --- |
| **Example** | **Isolation Devices** | **No. Of Nodes** | **Execution Time (in secs)** |
| 1 | Firewall | 6 | 1.741 |
| 2 | IPSec | 6 | 1.748 |
| 3 | Firewall + IPSec + IDS | 4 | 1.587 |
| 4 | Firewall + IPsec + IDS + NAT | 4 | 1.507 |

**Program**

The ‘C’ program that is required to implement the algorithm is here: 

The .exe file for direct testing the program is provided here: 

**Conclusion**

Although security architecture design usually follows well-known principles such as, isolation, defense-in- depth, fail safety etc., but, it is still performed in an ad-hoc manner. Recently, many issues has been raised about the validity and optimality of the security architecture when the design requires balancing of different competing factors such as, isolation, cost and usability. Therefore, generating a usable and optimal security configuration resolving the contention between the security requirements and business constraints is an important but challenging problem.

**References**

1. A Formal Approach for Network Security Design - Padmalochan Bera, Ehab Al-Shaer and Mohammad Ashiqur Rahaman.
2. <http://computer.howstuffworks.com/firewall3.htm>
3. Book: Cisco Secure Virtual Private Networks and Chapter: VPNs and VPN Technologies by Andrew Mason.
4. Intrusion Detection Systems: How They Work, written by: J. Forlanda, edited by: Lamar Stonecypher.
5. Book: [Routing TCP/IP, Volume II (CCIE Professional Development)](http://www.ciscopress.com/store/routing-tcp-ip-volume-ii-ccie-professional-development-9781578700899?w_ptgrevartcl=Network+Address+Translation_25273), chapter Network Address Translation, By Jeff Doyle and Jennifer Carroll.
6. http://www.graphviz.org/