

Chapter 3

Adaptive Services Overview

The Adaptive Services PIC (AS PIC) allows you to provide multiple services on a single PIC by configuring a set of services and applications. The AS PIC offers a special range of services you configure in one or more service sets.

You can install the AS PIC in any M-series router. The M7i router includes an integrated version of the AS PIC as an optional component, which offers all the features of the standalone version at a reduced bandwidth.



NOTE: To take advantage of all the features available on the AS PIC, you must install it in an Enhanced FPC in an M-series router equipped with an Internet Processor II ASIC, or a similarly equipped T-series routing platform. To find out whether your router hardware is suitably equipped, you can use the `show chassis` command. For more information, see the *JUNOS Internet Software Protocols, Class of Service, Chassis, and Management Command Reference*.

The following services are configured within a service set and are available only on the Adaptive Services PIC:

Stateful firewall—A type of firewall filter that considers state information derived from previous communications and other applications when evaluating traffic.

Network Address Translation (NAT)—A security procedure for concealing host addresses on a private network behind a pool of public addresses.

Intrusion detection services (IDS)—A set of tools for detecting, redirecting, and preventing certain kinds of network attack and intrusion.

Internet Protocol Security (IPSec)—A set of tools for configuring manual or dynamic security associations (SAs) for encryption of data traffic.

The configuration for these services comprises a series of rules that you can arrange in order of precedence as a *rule set*. Each rule follows the structure of a firewall filter, with a `from` statement containing input or match conditions and a `then` statement containing actions to be taken if the match conditions are met.

The following services are also configured on the AS PIC, but do not use the rule set definition:

Layer 2 Tunneling Protocol (L2TP)—A tool for setting up secure tunnels using Point-to-Point Protocol (PPP) encapsulation across Layer 2 networks.

Voice services—A feature that uses the compressed real-time transport protocol (RTP) to enable voice over IP traffic to use low-speed links more effectively.

In addition, JUNOS software includes the following tools for configuring services:

Application protocols definition—Allows you to configure properties of application protocols that are subject to processing by router services, and group the application definitions into application sets.

Service-set definition—Allows you to configure combinations of directional rules and default settings that control the behavior of each service in the service set.

This chapter includes the following topics:

Services Configuration Flow on page 20

Stateful Firewall Overview on page 21

Network Address Translation Overview on page 24

IPSec Overview on page 24

Layer 2 Tunneling Protocol Overview on page 27

Voice Services Overview on page 27

Examples: Services Interfaces Configuration on page 28

Services Configuration Flow

You follow these general steps to configure services:

1. Define application objects by configuring statements at the [edit applications] hierarchy level.
2. Define service rules by configuring statements at the [edit services (ids | ipsec-vpn | nat | stateful-firewall) rule] hierarchy level.
3. Group the service rules by configuring the rule-set statement at the [edit services (ids | ipsec-vpn | nat | stateful-firewall)] hierarchy level.

4. Group service rule sets under a service-set definition by configuring the service-set statement at the [edit services] hierarchy level.
5. Apply the service set on an interface by including the service-set statement at the [edit interfaces *interface-name* unit *logical-unit-number* family inet service (input | output)] hierarchy level. Alternatively, you can configure AS PIC logical interfaces as a next-hop destination by including the next-hop-service statement at the [edit services service-set *service-set-name*] hierarchy level.



NOTE: You can configure IDS, NAT, and stateful firewall service rules within the same service set. You must configure IPSec services in a separate service set, although you can apply both service sets to the same AS PIC.

Stateful Firewall Overview

Routers use firewalls to track and control the flow of traffic. The Adaptive Services PIC employs a type of firewall called a *stateful firewall*. Contrasted with a *stateless* firewall that inspects packets in isolation, a stateful firewall provides an extra layer of security by using state information derived from past communications and other applications to make dynamic control decisions for new communication attempts.

Stateful firewalls group relevant *flows* into *conversations*. A flow is identified by the following five properties:

- Source address
- Source port
- Destination address
- Destination port
- Protocol

A typical TCP or UDP conversation consists of two flows: the initiation flow and the responder flow. However, some conversations, such as an FTP conversation, might consist of two control flows and many data flows.

Firewall rules govern whether the conversation is allowed to be established. If a conversation is allowed, all flows within the conversation are permitted, including flows that are created during the life cycle of the conversation.

You configure stateful firewalls using a powerful rule-driven conversation handling path. A *rule* consists of direction, source address, source port, destination address, destination port, IP protocol value, and application protocol or service. In addition to the specific values you configure, you can assign the value any to rule objects, addresses, or ports, which allows them to match any input value. Finally, you can optionally negate the rule objects, which negates the result of the type-specific match.

Firewall rules are directional. For each new conversation, the router software checks the initiation flow matching the direction specified by the rule.

Firewall rules are ordered. The software checks the rules in the order in which you include them in the configuration. The first time the firewall discovers a match, the router implements the action specified by that rule. Rules still unchecked are ignored.

Firewall Application Protocols Support

By inspecting the application protocol data, the AS PIC firewall can intelligently enforce security policies and allow only the minimal required packet traffic to flow through the firewall.

The firewall rules are configured in relation to an interface. By default, the stateful firewall allows all sessions initiated from the hosts behind the interface to pass through the router.

Stateful Firewall Anomaly Checking

The stateful firewall recognizes the following events as anomalies and sends them to the intrusion detection services (IDS) software for processing:

IP anomalies:

- IP version is not correct.

- IP header length field is too small.

- IP header length is set larger than the entire packet.

- Bad header checksum.

- IP total length field is shorter than header length.

- Packet has incorrect IP options.

- ICMP packet length error.

- Time-to-live (TTL) equals 0.

IP address anomalies:

- IP packet source is a broadcast or multicast.

- Land attack (source IP equals destination IP).

IP fragmentation anomalies:

- IP fragment overlap.

- IP fragment missed.

- IP fragment length error.

- IP packet length is more than 64 KB.

- Tiny fragment attack.

TCP anomalies:

TCP port 0.

TCP sequence number 0 and flags 0.

TCP sequence number 0 and FIN/PSH/RST flags set.

TCP flags with wrong combination (TCP FIN/RST or SYN/(URG|FIN|RST)).

Bad TCP checksum.

UDP anomalies:

UDP source or destination port 0.

UDP header length check failed.

Bad UDP checksum.

Anomalies found through stateful TCP or UDP checks:

SYN followed by SYN-ACK packets without ACK from initiator.

SYN followed by RST packets.

SYN without SYN-ACK.

Non-SYN first flow packet.

ICMP unreachable errors for SYN packets.

ICMP unreachable errors for UDP packets.

Packets dropped according to stateful firewall rules.

If you employ stateful anomaly detection in conjunction with stateless detection, IDS can provide early warning for a wide range of attacks, including these:

TCP or UDP network probes and port scanning

SYN flood attacks

IP fragmentation-based attacks such as teardrop, bonk, and boink

Network Address Translation Overview

Network Address Translation (NAT) is a mechanism for concealing a set of host addresses on a private network behind a pool of public addresses. It can be used as a security measure to protect the host addresses from direct targeting in network attacks.

Traditional NAT, specified in RFC 3022, *Traditional IP Network Address Translator*, is fully supported by JUNOS software. In addition, network address port translation (NAPT) is supported for source addresses.

The AS PIC interfaces support three types of NAT processing: static-source, dynamic-source, and static-destination.

Static-source NAT hides a private network without using NAPT.

Dynamic-source NAT hides a private network using NAPT.

Static-destination NAT makes selected private servers accessible.

You can implement NAT to hide one or many hosts on a private network behind a pool of public IP addresses. The pool can be as small as one IP address, or it can be a set of contiguous IP addresses. You can specify a port range to restrict port translation when NAT is configured in dynamic-source mode.

Private address to public address binding can be either static or dynamic. In the basic NAT mode, a NAT rule can force a private IP address to be always bound to a public address; in the NAPT mode, a NAT rule can force a paired private address and private TU port to be mapped to a public IP and public TU port. However, when the address binding is not statically forced by the NAT rules, NAT can dynamically pick an available address or address and TU port pairing when a new session starts.

Like most traditional NAT implementations, the JUNOS implementation of NAT supports sessions initiated from the private side only. Sessions initiated from the public side are supported only when you configure static address binding.

NAT becomes activated only when a packet passes the AS PIC stateful firewall inspection. For this reason, you must configure stateful firewall rules in combination with NAT. A stateful firewall discard packet does not reach NAT even if the packet otherwise matches the NAT rule.

IPSec Overview

The JUNOS software supports Internet Protocol Security (IPSec). This section discusses the following topics, which provide background information about configuring IPSec:

IPSec on page 25

Security Associations on page 25

IKE on page 25

Comparison of IPSec Services and ES Interface Configuration on page 26

For a list of IPSec- and IKE-supported standards, see the *JUNOS Internet Software System Basics Configuration Guide*.

IPSec

The IPSec architecture provides a security suite for the IPv4 and IPv6 network layers. The suite provides such functionality as authentication of origin, data integrity, confidentiality, replay protection, and nonrepudiation of source. In addition to IPSec, the JUNOS software also supports the Internet Key Exchange (IKE), which defines mechanisms for key generation and exchange, and manages security associations (SAs).

IPSec also defines a security association and key management framework that can be used with any network layer protocol. The SA specifies what protection policy to apply to traffic between two IP-layer entities. IPSec provides secure tunnels between two peers.

Security Associations

To use IPSec security services, you create SAs between hosts. An SA is a simplex connection that allows two hosts to communicate with each other securely by means of IPSec. There are two types of SAs, manual and dynamic:

Manual SAs require no negotiation; all values, including the keys, are static and specified in the configuration. Manual SAs statically define the Security Parameter Index (SPI) values, algorithms, and keys to be used, and require matching configurations on both ends of the tunnel. Each peer must have the same configured options for communication to take place.

Dynamic SAs require additional configuration. With dynamic SAs, you configure IKE first and then the SA. IKE creates dynamic security associations; it negotiates SAs for IPSec. The IKE configuration defines the algorithms and keys used to establish the secure IKE connection with the peer security gateway. This connection is then used to dynamically agree upon keys and other data used by the dynamic IPSec SA. The IKE SA is negotiated first and then used to protect the negotiations that determine the dynamic IPSec SAs.

IKE

IKE is a key management protocol that creates dynamic SAs; it negotiates SAs for IPSec. An IKE configuration defines the algorithms and keys used to establish a secure connection with a peer security gateway.

IKE performs the following tasks:

- Negotiates and manages IKE and IPSec parameters.

- Authenticates secure key exchange.

- Provides mutual peer authentication by means of shared secrets (not passwords) and public keys.

- Provides identity protection (in main mode).

IKE consists of two phases. In the first phase, it negotiates security attributes and establishes shared secrets to form the bidirectional IKE SA. In the second phase, inbound and outbound IPSec SAs are established and the IKE SA secures the exchanges. IKE also generates keying material, provides Perfect Forward Secrecy, and exchanges identities.

Comparison of IPSec Services and ES Interface Configuration

Table 2 compares the top-level configuration of IPSec features on the ES PIC interfaces and on the Adaptive Services PIC interfaces.

Table 2: Statement Equivalents for ES and Adaptive Services Interfaces

ES PIC Configuration	AS PIC IPSec Services Configuration
[edit security ipsec] proposal {...}	[edit services ipsec-vpn ipsec] proposal {...}
[edit security ipsec] policy {...}	[edit services ipsec-vpn ipsec] policy {...}
[edit security ipsec] security-association sa-dynamic {...}	[edit services ipsec-vpn rule <i>rule-name</i>] term <i>term-name</i> match-conditions {...} then dynamic {...}
[edit security ipsec] security-association sa-manual {...}	[edit services ipsec-vpn rule <i>rule-name</i>] term <i>term-name</i> match-conditions {...} then manual {...}
[edit security ike] proposal {...}	[edit services ipsec-vpn ike] proposal {...}
[edit security ike] policy {...}	[edit services ipsec-vpn ike] policy {...}
Not available	[edit services ipsec-vpn] rule-set {...}
Not available	[edit services ipsec-vpn] service-set {...}
[edit interfaces <i>es-fpc/pic/port</i>] tunnel source <i>address</i>	[edit services ipsec-vpn service-set <i>set-name</i> ipsec-vpn local-gateway <i>address</i>]
[edit interfaces <i>es-fpc/pic/port</i>] tunnel destination <i>address</i>	[edit services ipsec-vpn rule <i>rule-name</i>] remote-gateway <i>address</i>



NOTE: Although many of the same statements and properties are valid on both platforms, the configurations are not interchangeable. You must commit a complete configuration for the PIC type that is installed in your router.

Layer 2 Tunneling Protocol Overview

The Layer 2 Tunneling Protocol (L2TP) is defined in the following standard:

RFC 2661, *Layer Two Tunneling Protocol (L2TP)*

L2TP facilitates the tunneling of Point-to-Point Protocol (PPP) packets across an intervening network in a way that is as transparent as possible to both end users and applications. It employs access profiles for group and individual user access, and uses authentication to establish secure connections between the two ends of each tunnel.

The L2TP services available on the AS PIC are supported only on M7i routing platforms.

Voice Services Overview

The Adaptive Services PIC includes a voice services feature that allows you to specify a particular type of logical interface to accommodate voice over IP traffic. This interface uses the compressed real-time transport protocol (RTP), which is defined in the following standard:

RFC 2508, *Compressing IP/UDP/RTP Headers for Low-Speed Serial Links*

Compressed RTP enables voice traffic to use low-speed links more effectively by compressing the 40-byte IP/UDP/RTP header down to 2 to 4 bytes in most cases.

Voice services on the AS PIC support single-link Multilink PPP (MLPPP)-encapsulated IPv4 traffic over T1, E1, and T3 physical interfaces.

Examples: Services Interfaces Configuration

The following configuration includes all the items necessary to configure services on an interface. For examples showing individual service configurations, see the chapter that describes that service in detail.

```
[edit]
interfaces {
  fe-0/1/0 {
    unit 0 {
      family inet {
        service {
          input {
            service-set Firewall-Set;
          }
          output {
            service-set Firewall-Set;
          }
        }
        address 10.1.3.2/24;
      }
    }
  }
  fe-0/1/1 {
    unit 0 {
      family inet {
        filter {
          input Sample;
        }
        address 112.148.1.2/24;
      }
    }
  }
  sp-1/0/0 {
    unit 0 {
      family inet {
        address 10.1.3.2/32 {
          destination 10.1.3.50;
        }
      }
    }
  }
}
```

```

forwarding-options {
  sampling {
    input {
      family inet {
        rate 1;
      }
    }
    output {
      cflowd 10.1.3.1 {
        port 2055;
        version 5;
      }
      flow-inactive-timeout 15;
      flow-active-timeout 60;
      interface sp-1/0/0 {
        engine-id 1;
        engine-type 136;
        source-address 10.1.3.2;
      }
    }
  }
}
firewall {
  filter Sample {
    term Sample {
      then {
        count Sample;
        sample;
        accept;
      }
    }
  }
}
services {
  stateful-firewall {
    rule Rule1 {
      match-direction input;
      term 1 {
        from {
          application-sets Applications;
        }
        then {
          accept;
        }
      }
      term accept {
        then {
          accept;
        }
      }
    }
  }
}

```

```

rule Rule2 {
  match-direction output;
  term Local {
    from {
      source-address {
        10.1.3.2/32;
      }
    }
    then {
      accept;
    }
  }
}
}
ids {
  rule Attacks {
    match-direction output;
    term Match {
      from {
        application-sets Applications;
      }
      then {
        logging {
          syslog;
        }
      }
    }
  }
}
nat {
  pool public {
    address-range low 112.148.2.1 high 112.148.2.32;
    port automatic;
  }
  rule Private-Public {
    match-direction input;
    term Translate {
      then {
        translated {
          source-pool public;
          translation-type source dynamic;
        }
      }
    }
  }
}
service-set Firewall-Set {
  stateful-firewall-rules Rule1;
  stateful-firewall-rules Rule2;
  nat-rules Private-Public;
  ids-rules Attacks;
  interface-service {
    service-interface sp-1/0/0;
  }
}
}

```

```

applications {
  application ICMP {
    application-protocol icmp;
  }
  application FTP {
    application-protocol ftp;
    destination-port ftp;
  }
  application-set Applications {
    application ICMP;
    application FTP;
  }
}

```

The following example combines VRF and services configuration:

```

[edit policy-options]
policy-statement test-policy {
  term t1 {
    then reject;
  }
}

[edit routing-instances]
test {
  interface ge-0/2/0.0;
  interface sp-1/3/0.20;
  instance-type vrf;
  route-distinguisher 10.58.255.1:37;
  vrf-import test-policy;
  vrf-export test-policy;
  routing-options {
    static {
      route 0.0.0.0/0 next-table inet.0;
    }
  }
}

[edit interfaces]
ge-0/2/0 {
  unit 0 {
    family inet {
      service {
        input service-set nat-me;
        output service-set nat-me;
      }
    }
  }
}

```

```

sp-1/3/0 {
  unit 0 {
    family inet;
  }
  unit 20 {
    family inet;
    service-domain inside;
  }
}

[edit services]
stateful-firewall {
  rule allow-any-input {
    match-direction input;
    term t1 {
      then accept;
    }
  }
}
nat {
  pool hide-pool {
    address 10.58.16.100;
    port automatic;
  }
  rule hide-all-input {
    match-direction input;
    term t1 {
      then {
        translated {
          source-pool hide-pool;
          translation-type source dynamic;
        }
      }
    }
  }
}
service-set nat-me {
  stateful-firewall-rules allow-any-input;
  nat-rules hide-all-input;
  interface-service {
    service-interface sp-1/3/0.20;
  }
}

```

The following example shows dynamic-source NAT applied as a next-hop service:

```
[edit interfaces]
ge-0/2/0{
  unit 0 {
    family mpls;
  }
}
sp-1/3/0 {
  unit 0 {
    family inet;
  }
  unit 20 {
    family inet;
  }
  unit 32 {
    family inet;
  }
}

[edit routing-instances]
protected-domain {
  interface ge-0/2/0.0;
  interface sp-1/3/0.20;
  instance-type vrf;
  route-distinguisher 10.58.255.17:37;
  vrf-import protected-domain-policy;
  vrf-export protected-domain-policy;
  routing-options {
    static {
      route 0.0.0.0/0 next-hop sp-1/3/0.20;
    }
  }
}

[edit policy-options]
policy-statement protected-domain-policy {
  term t1 {
    then reject;
  }
}

[edit services]
stateful-firewall {
  rule allow-all {
    match-direction input;
    term t1 {
      then {
        accept;
      }
    }
  }
}
}
```

```

nat {
  pool my-pool {
    address 10.58.16.100;
    port automatic;
  }
  rule hide-all {
    match-direction input;
    term t1 {
      then {
        translated {
          source-pool my-pool;
          translation-type source dynamic;
        }
      }
    }
  }
}
service-set null-sfw-with-nat {
  stateful-firewall-rules allow-all;
  nat-rules hide-all;
  next-hop-service {
    inside-service-interface sp-1/3/0.20;
    outside-service-interface sp-1/3/0.32;
  }
}

```

The following example configuration enables NAT between VRFs with overlapping private addresses, using distinct public addresses for the source and destination NAT in this scenario:

A host in vrf-a traverses 10.58.16.201 to reach 10.58.0.2 in vrf-b.

A host in vrf-b traverses 10.58.16.101 to reach 10.58.0.2 in vrf-a.

```

[edit interfaces]
ge-0/2/0{
  unit 0 {
    family inet {
      address 10.58.0.1/24;
      service {
        input service-set vrf-a-svc-set;
        output service-set vrf-a-svc-set;
      }
    }
  }
}
ge-0/3/0{
  unit 0 {
    family inet {
      address 10.58.0.1/24;
      service {
        input service-set vrf-b-svc-set;
        output service-set vrf-b-svc-set;
      }
    }
  }
}

```

```

sp-1/3/0 {
  unit 0 {
    family inet;
  }
  unit 10 {
    family inet;
    service-domain inside;
  }
  unit 20 {
    family inet;
    service-domain inside;
  }
}

[edit policy-options]
policy-statement test-policy {
  term t1 {
    then reject;
  }
}

[edit routing-instances]
vrf-a {
  interface ge-0/2/0.0;
  interface sp-1/3/0.10;
  instance-type vrf;
  route-distinguisher 1.1.1.1:1;
  vrf-import test-policy;
  vrf-export test-policy;
  routing-options {
    static {
      route 0.0.0.0/0 next-table inet.0;
    }
  }
}
vrf-b {
  interface ge-0/3/0.0;
  interface sp-1/3/0.20;
  instance-type vrf;
  route-distinguisher 2.2.2.2:2;
  vrf-import test-policy;
  vrf-export test-policy;
  routing-options {
    static {
      route 0.0.0.0/0 next-table inet.0;
    }
  }
}
}

```

```
[edit services]
stateful-firewall {
  rule allow-all {
    match-direction input-output;
    term t1 {
      then {
        accept;
      }
    }
  }
}
nat {
  pool vrf-a-src-pool {
    address 10.58.16.100;
    port automatic;
  }
  pool vrf-a-dst-pool {
    address 10.58.0.2;
  }
  rule vrf-a-input {
    match-direction input;
    term t1 {
      then {
        translated {
          source-pool vrf-a-src-pool;
          translation-type source dynamic;
        }
      }
    }
  }
  rule vrf-a-output {
    match-direction output;
    term t1 {
      from {
        destination-address 10.58.16.101;
      }
      then {
        translated {
          destination-pool vrf-a-dst-pool;
          translation-type destination static;
        }
      }
    }
  }
}
```

```

pool vrf-b-src-pool {
    address 10.58.16.200;
    port automatic;
}
pool vrf-b-dst-pool {
    address 10.58.0.2;
}
rule vrf-b-input {
    match-direction input;
    term t1 {
        then {
            translated {
                source-pool vrf-b-src-pool;
                translation-type source dynamic;
            }
        }
    }
}
rule vrf-b-output {
    match-direction output;
    term t1 {
        from {
            destination-address 10.58.16.201;
        }
        then {
            translated {
                destination-pool vrf-b-dst-pool;
                translation-type destination static;
            }
        }
    }
}
}
service-set vrf-a-svc-set {
    stateful-firewall-rules allow-all;
    nat-rules vrf-a-input;
    nat-rules vrf-a-output;
    interface-service {
        service-interface sp-1/3/0.10;
    }
}
service-set vrf-b-svc-set {
    stateful-firewall-rules allow-all;
    nat-rules vrf-b-input;
    nat-rules vrf-b-output;
    interface-service {
        service-interface sp-1/3/0.20;
    }
}
}

```

The following example supports BOOTP and broadcast addresses:

```
[edit applications]
application bootp {
  application-protocol bootp;
  protocol udp;
  destination-port 67;
}

[edit services]
stateful-firewall bootp-support {
  rule bootp-allow {
    direction input;
    term bootp-allow {
      from {
        destination-address [ any-unicast, 255.255.255.255 ];
        application bootp;
      }
      then {
        accept;
      }
    }
  }
}
```