```
**********************************************************
   27490 Mon Jul 14 15:55:10 2014
new/usr/src/uts/common/inet/ip/ip_tunables.c
5000 Set ipsec_policy_log_interval to 0 by default
**********************************************************
______unchanged_portion_omitted_

 490 /*
 491  * All of these are alterable, within the min/max values given, at run time.
 492  *
 493  * Note: All those tunables which do not start with "_" are Committed and
 494  * therefore are public. See PSARC 2010/080.
 495  */
 496 mod_prop_info_t ip_propinfo_tbl[] = {
 497         /* tunable - 0 */
 498         { "_respond_to_address_mask_broadcast", MOD_PROTO_IP,
 499             mod_set_boolean, mod_get_boolean,
 500             {B_FALSE}, {B_FALSE} },

 502         { "_respond_to_echo_broadcast", MOD_PROTO_IP,
 503             mod_set_boolean, mod_get_boolean,
 504             {B_TRUE},  {B_TRUE} },

 506         { "_respond_to_echo_multicast", MOD_PROTO_IPV4,
 507             mod_set_boolean, mod_get_boolean,
 508             {B_TRUE}, {B_TRUE} },

 510         { "_respond_to_timestamp", MOD_PROTO_IP,
 511             mod_set_boolean, mod_get_boolean,
 512             {B_FALSE}, {B_FALSE} },

 514         { "_respond_to_timestamp_broadcast", MOD_PROTO_IP,
 515             mod_set_boolean, mod_get_boolean,
 516             {B_FALSE}, {B_FALSE} },

 518         { "_send_redirects", MOD_PROTO_IPV4,
 519             mod_set_boolean, mod_get_boolean,
 520             {B_TRUE}, {B_TRUE} },

 522         { "_forward_directed_broadcasts", MOD_PROTO_IP,
 523             mod_set_boolean, mod_get_boolean,
 524             {B_FALSE}, {B_FALSE} },

 526         { "_mrtdebug", MOD_PROTO_IP,
 527             mod_set_uint32, mod_get_uint32,
 528             {0, 10, 0}, {0} },

 530         { "_ire_reclaim_fraction", MOD_PROTO_IP,
 531             mod_set_uint32, mod_get_uint32,
 532             {1, 8, 3}, {3} },

 534         { "_nce_reclaim_fraction", MOD_PROTO_IP,
 535             mod_set_uint32, mod_get_uint32,
 536             {1, 8, 3}, {3} },

 538         /* tunable - 10 */
 539         { "_dce_reclaim_fraction", MOD_PROTO_IP,
 540             mod_set_uint32, mod_get_uint32,
 541             {1, 8, 3}, {3} },

 543         { "ttl", MOD_PROTO_IPV4,
 544             mod_set_uint32, mod_get_uint32,
 545             {1, 255, 255}, {255} },

 547         { "_forward_src_routed", MOD_PROTO_IPV4,
 548             mod_set_boolean, mod_get_boolean,
```



```
 549             {B_FALSE}, {B_FALSE} },

 551         { "_wroff_extra", MOD_PROTO_IP,
 552             mod_set_uint32, mod_get_uint32,
 553             {0, 256, 32}, {32} },

 555         /* following tunable is in seconds - a deviant! */
 556         { "_pathmtu_interval", MOD_PROTO_IP,
 557             mod_set_uint32, mod_get_uint32,
 558             {2, 999999999, 60*20}, {60*20} },

 560         { "_icmp_return_data_bytes", MOD_PROTO_IPV4,
 561             mod_set_uint32, mod_get_uint32,
 562             {8, 65536, 64}, {64} },

 564         { "_path_mtu_discovery", MOD_PROTO_IP,
 565             mod_set_boolean, mod_get_boolean,
 566             {B_TRUE}, {B_TRUE} },

 568         { "_pmtu_min", MOD_PROTO_IP,
 569             mod_set_uint32, mod_get_uint32,
 570             {68, 65535, 576}, {576} },

 572         { "_ignore_redirect", MOD_PROTO_IPV4,
 573             mod_set_boolean, mod_get_boolean,
 574             {B_FALSE}, {B_FALSE} },

 576         { "_arp_icmp_error", MOD_PROTO_IP,
 577             mod_set_boolean, mod_get_boolean,
 578             {B_FALSE}, {B_FALSE} },

 580         /* tunable - 20 */
 581         { "_broadcast_ttl", MOD_PROTO_IP,
 582             mod_set_uint32, mod_get_uint32,
 583             {1, 254, 1}, {1} },

 585         { "_icmp_err_interval", MOD_PROTO_IP,
 586             mod_set_uint32, mod_get_uint32,
 587             {0, 99999, 100}, {100} },

 589         { "_icmp_err_burst", MOD_PROTO_IP,
 590             mod_set_uint32, mod_get_uint32,
 591             {1, 99999, 10}, {10} },

 593         { "_reass_queue_bytes", MOD_PROTO_IP,
 594             mod_set_uint32, mod_get_uint32,
 595             {0, 999999999, 1000000}, {1000000} },

 597         /*
 598          * See comments for ip_strict_src_multihoming for an explanation
 599          * of the semantics of ip_strict_dst_multihoming
 600          */
 601         { "_strict_dst_multihoming", MOD_PROTO_IPV4,
 602             mod_set_uint32, mod_get_uint32,
 603             {0, 1, 0}, {0} },

 605         { "_addrs_per_if", MOD_PROTO_IP,
 606             mod_set_uint32, mod_get_uint32,
 607             {1, MAX_ADDRS_PER_IF, 256}, {256} },

 609         { "_ipsec_override_persocket_policy", MOD_PROTO_IP,
 610             mod_set_boolean, mod_get_boolean,
 611             {B_FALSE}, {B_FALSE} },

 613         { "_icmp_accept_clear_messages", MOD_PROTO_IP,
 614             mod_set_boolean, mod_get_boolean,
```

```
615             {B_TRUE}, {B_TRUE} },

617         { "_igmp_accept_clear_messages", MOD_PROTO_IP,
618             mod_set_boolean, mod_get_boolean,
619             {B_TRUE}, {B_TRUE} },

621         { "_ndp_delay_first_probe_time", MOD_PROTO_IP,
622             mod_set_uint32, mod_get_uint32,
623             {2, 999999999, ND_DELAY_FIRST_PROBE_TIME},
624             {ND_DELAY_FIRST_PROBE_TIME} },

626         /* tunable - 30 */
627         { "_ndp_max_unicast_solicit", MOD_PROTO_IP,
628             mod_set_uint32, mod_get_uint32,
629             {1, 999999999, ND_MAX_UNICAST_SOLICIT}, {ND_MAX_UNICAST_SOLICIT} },

631         { "hoplimit", MOD_PROTO_IPV6,
632             mod_set_uint32, mod_get_uint32,
633             {1, 255, IPV6_MAX_HOPS}, {IPV6_MAX_HOPS} },

635         { "_icmp_return_data_bytes", MOD_PROTO_IPV6,
636             mod_set_uint32, mod_get_uint32,
637             {8, IPV6_MIN_MTU, IPV6_MIN_MTU}, {IPV6_MIN_MTU} },

639         { "_forward_src_routed", MOD_PROTO_IPV6,
640             mod_set_boolean, mod_get_boolean,
641             {B_FALSE}, {B_FALSE} },

643         { "_respond_to_echo_multicast", MOD_PROTO_IPV6,
644             mod_set_boolean, mod_get_boolean,
645             {B_TRUE}, {B_TRUE} },

647         { "_send_redirects", MOD_PROTO_IPV6,
648             mod_set_boolean, mod_get_boolean,
649             {B_TRUE}, {B_TRUE} },

651         { "_ignore_redirect", MOD_PROTO_IPV6,
652             mod_set_boolean, mod_get_boolean,
653             {B_FALSE}, {B_FALSE} },

655         /*
656          * See comments for ip6_strict_src_multihoming for an explanation
657          * of the semantics of ip6_strict_dst_multihoming
658          */
659         { "_strict_dst_multihoming", MOD_PROTO_IPV6,
660             mod_set_uint32, mod_get_uint32,
661             {0, 1, 0}, {0} },

663         { "_src_check", MOD_PROTO_IP,
664             mod_set_uint32, mod_get_uint32,
665             {0, 2, 2}, {2} },

667         { "_ipsec_policy_log_interval", MOD_PROTO_IP,
668             mod_set_uint32, mod_get_uint32,
669             {0, 999999, 0}, {0} },
669             {0, 999999, 1000}, {1000} },

671         /* tunable - 40 */
672         { "_pim_accept_clear_messages", MOD_PROTO_IP,
673             mod_set_boolean, mod_get_boolean,
674             {B_TRUE}, {B_TRUE} },

676         { "_ndp_unsolicit_interval", MOD_PROTO_IP,
677             mod_set_uint32, mod_get_uint32,
678             {1000, 20000, 2000}, {2000} },
```



```
680         { "_ndp_unsolicit_count", MOD_PROTO_IP,
681             mod_set_uint32, mod_get_uint32,
682             {1, 20, 3}, {3} },

684         { "_ignore_home_address_opt", MOD_PROTO_IPV6,
685             mod_set_boolean, mod_get_boolean,
686             {B_TRUE}, {B_TRUE} },

688         { "_policy_mask", MOD_PROTO_IP,
689             mod_set_uint32, mod_get_uint32,
690             {0, 15, 0}, {0} },

692         { "_ecmp_behavior", MOD_PROTO_IP,
693             mod_set_uint32, mod_get_uint32,
694             {0, 2, 2}, {2} },

696         { "_multirt_ttl", MOD_PROTO_IP,
697             mod_set_uint32, mod_get_uint32,
698             {0, 255, 1}, {1} },

700         /* following tunable is in seconds - a deviant */
701         { "_ire_badcnt_lifetime", MOD_PROTO_IP,
702             mod_set_uint32, mod_get_uint32,
703             {0, 3600, 60}, {60} },

705         { "_max_temp_idle", MOD_PROTO_IP,
706             mod_set_uint32, mod_get_uint32,
707             {0, 999999, 60*60*24}, {60*60*24} },

709         { "_max_temp_defend", MOD_PROTO_IP,
710             mod_set_uint32, mod_get_uint32,
711             {0, 1000, 1}, {1} },

713         /* tunable - 50 */
714         /*
715          * when a conflict of an active address is detected,
716          * defend up to ip_max_defend times, within any
717          * ip_defend_interval span.
718          */
719         { "_max_defend", MOD_PROTO_IP,
720             mod_set_uint32, mod_get_uint32,
721             {0, 1000, 3}, {3} },

723         { "_defend_interval", MOD_PROTO_IP,
724             mod_set_uint32, mod_get_uint32,
725             {0, 999999, 30}, {30} },

727         { "_dup_recovery", MOD_PROTO_IP,
728             mod_set_uint32, mod_get_uint32,
729             {0, 3600000, 300000}, {300000} },

731         { "_restrict_interzone_loopback", MOD_PROTO_IP,
732             mod_set_boolean, mod_get_boolean,
733             {B_TRUE}, {B_TRUE} },

735         { "_lso_outbound", MOD_PROTO_IP,
736             mod_set_boolean, mod_get_boolean,
737             {B_TRUE}, {B_TRUE} },

739         { "_igmp_max_version", MOD_PROTO_IP,
740             mod_set_uint32, mod_get_uint32,
741             {IGMP_V1_ROUTER, IGMP_V3_ROUTER, IGMP_V3_ROUTER},
742             {IGMP_V3_ROUTER} },

744         { "_mld_max_version", MOD_PROTO_IP,
745             mod_set_uint32, mod_get_uint32,
```

```
746             {MLD_V1_ROUTER, MLD_V2_ROUTER, MLD_V2_ROUTER}, {MLD_V2_ROUTER} },

748         { "forwarding", MOD_PROTO_IPV4,
749             ip_set_forwarding, ip_get_forwarding,
750             {IP_FORWARD_NEVER}, {IP_FORWARD_NEVER} },

752         { "forwarding", MOD_PROTO_IPV6,
753             ip_set_forwarding, ip_get_forwarding,
754             {IP_FORWARD_NEVER}, {IP_FORWARD_NEVER} },

756         { "_reasm_timeout", MOD_PROTO_IPV4,
757             mod_set_uint32, mod_get_uint32,
758             {5, 255, IP_REASM_TIMEOUT},
759             {IP_REASM_TIMEOUT} },

761         /* tunable - 60 */
762         { "_reasm_timeout", MOD_PROTO_IPV6,
763             mod_set_uint32, mod_get_uint32,
764             {5, 255, IPV6_REASM_TIMEOUT},
765             {IPV6_REASM_TIMEOUT} },

767         { "_cgtp_filter", MOD_PROTO_IP,
768             ip_set_cgtp_filter, mod_get_boolean,
769             {B_FALSE}, {B_FALSE} },

771         /* delay before sending first probe: */
772         { "_arp_probe_delay", MOD_PROTO_IP,
773             mod_set_uint32, mod_get_uint32,
774             {0, 20000, 1000}, {1000} },

776         { "_arp_fastprobe_delay", MOD_PROTO_IP,
777             mod_set_uint32, mod_get_uint32,
778             {0, 20000, 100}, {100} },

780         /* interval at which DAD probes are sent: */
781         { "_arp_probe_interval", MOD_PROTO_IP,
782             mod_set_uint32, mod_get_uint32,
783             {10, 20000, 1500}, {1500} },

785         { "_arp_fastprobe_interval", MOD_PROTO_IP,
786             mod_set_uint32, mod_get_uint32,
787             {10, 20000, 150}, {150} },

789         { "_arp_probe_count", MOD_PROTO_IP,
790             mod_set_uint32, mod_get_uint32,
791             {0, 20, 3}, {3} },

793         { "_arp_fastprobe_count", MOD_PROTO_IP,
794             mod_set_uint32, mod_get_uint32,
795             {0, 20, 3}, {3} },

797         { "_dad_announce_interval", MOD_PROTO_IPV4,
798             mod_set_uint32, mod_get_uint32,
799             {0, 3600000, 15000}, {15000} },

801         { "_dad_announce_interval", MOD_PROTO_IPV6,
802             mod_set_uint32, mod_get_uint32,
803             {0, 3600000, 15000}, {15000} },

805         /* tunable - 70 */
806         /*
807          * Rate limiting parameters for DAD defense used in
808          * ill_defend_rate_limit():
809          * defend_rate : pkts/hour permitted
810          * defend_interval : time that can elapse before we send out a
811          *                       DAD defense.
```



```
812          * defend_period: denominator for defend_rate (in seconds).
813          */
814         { "_arp_defend_interval", MOD_PROTO_IP,
815             mod_set_uint32, mod_get_uint32,
816             {0, 3600000, 300000}, {300000} },

818         { "_arp_defend_rate", MOD_PROTO_IP,
819             mod_set_uint32, mod_get_uint32,
820             {0, 20000, 100}, {100} },

822         { "_ndp_defend_interval", MOD_PROTO_IP,
823             mod_set_uint32, mod_get_uint32,
824             {0, 3600000, 300000}, {300000} },

826         { "_ndp_defend_rate", MOD_PROTO_IP,
827             mod_set_uint32, mod_get_uint32,
828             {0, 20000, 100}, {100} },

830         { "_arp_defend_period", MOD_PROTO_IP,
831             mod_set_uint32, mod_get_uint32,
832             {5, 86400, 3600}, {3600} },

834         { "_ndp_defend_period", MOD_PROTO_IP,
835             mod_set_uint32, mod_get_uint32,
836             {5, 86400, 3600}, {3600} },

838         { "_icmp_return_pmtu", MOD_PROTO_IPV4,
839             mod_set_boolean, mod_get_boolean,
840             {B_TRUE}, {B_TRUE} },

842         { "_icmp_return_pmtu", MOD_PROTO_IPV6,
843             mod_set_boolean, mod_get_boolean,
844             {B_TRUE}, {B_TRUE} },

846         /*
847          * publish count/interval values used to announce local addresses
848          * for IPv4, IPv6.
849          */
850         { "_arp_publish_count", MOD_PROTO_IP,
851             mod_set_uint32, mod_get_uint32,
852             {1, 20, 5}, {5} },

854         { "_arp_publish_interval", MOD_PROTO_IP,
855             mod_set_uint32, mod_get_uint32,
856             {1000, 20000, 2000}, {2000} },

858         /* tunable - 80 */
859         /*
860          * The ip*strict_src_multihoming and ip*strict_dst_multihoming provide
861          * a range of choices for setting strong/weak/preferred end-system
862          * behavior. The semantics for setting these are:
863          *
864          * ip*_strict_dst_multihoming = 0
865          *    weak end system model for managing ip destination addresses.
866          *    A packet with IP dst D1 that's received on interface I1 will be
867          *    accepted as long as D1 is one of the local addresses on
868          *    the machine, even if D1 is not configured on I1.
869          * ip*strict_dst_multihioming = 1
870          *    strong end system model for managing ip destination addresses.
871          *    A packet with IP dst D1 that's received on interface I1 will be
872          *    accepted if, and only if, D1 is configured on I1.
873          *
874          * ip*strict_src_multihoming = 0
875          *    Source agnostic route selection for outgoing packets: the
876          *    outgoing interface for a packet will be computed using
877          *    default algorithms for route selection, where the route
```

```
 878          *    with the longest matching prefix is chosen for the output
 879          *    unless other route selection constraints are explicitly
 880          *    specified during routing table lookup.  This may result
 881          *    in packet being sent out on interface I2 with source
 882          *    address S1, even though S1 is not a configured address on I2.
 883          * ip*strict_src_multihoming = 1
 884          *    Preferred source aware route selection for outgoing packets: for
 885          *    a packet with source S2, destination D2, the route selection
 886          *    algorithm will first attempt to find a route for the destination
 887          *    that goes out through an interface where S2 is
 888          *    configured. If such a route cannot be found, then the
 889          *    best-matching route for D2 will be selected.
 890          * ip*strict_src_multihoming = 2
 891          *    Source aware route selection for outgoing packets: a packet will
 892          *    be sent out on an interface I2 only if the src address S2 of the
 893          *    packet is a configured address on I2. In conjunction with
 894          *    the setting 'ip_strict_dst_multihoming == 1', this will result in
 895          *    the implementation of Strong ES as defined in Section 3.3.4.2 of
 896          *    RFC 1122
 897          */
 898         { "_strict_src_multihoming", MOD_PROTO_IPV4,
 899             ip_set_src_multihoming, mod_get_uint32,
 900             {0, 2, 0}, {0} },

 902         { "_strict_src_multihoming", MOD_PROTO_IPV6,
 903             ip_set_src_multihoming, mod_get_uint32,
 904             {0, 2, 0}, {0} },

 906 #ifdef DEBUG
 907         { "_drop_inbound_icmpv6", MOD_PROTO_IPV6,
 908             mod_set_boolean, mod_get_boolean,
 909             {B_FALSE}, {B_FALSE} },
 910 #else
 911         { "", 0, NULL, NULL, {0}, {0} },
 912 #endif

 914         { "_dce_reclaim_threshold", MOD_PROTO_IP,
 915             mod_set_uint32, mod_get_uint32,
 916             {1, 100000, 32}, {32} },

 918         { "mtu", MOD_PROTO_IPV4, NULL, ip_get_mtu, {0}, {0} },

 920         { "mtu", MOD_PROTO_IPV6, NULL, ip_get_mtu, {0}, {0} },

 922         /*
 923          * The following entry is a placeholder for 'ip_debug' global
 924          * variable. Within these callback functions, we will be
 925          * setting/getting the global variable
 926          */
 927         { "_debug", MOD_PROTO_IP,
 928             ip_set_debug, ip_get_debug,
 929             {0, 20, 0}, {0} },

 931         { "hostmodel", MOD_PROTO_IPV4, ip_set_hostmodel, ip_get_hostmodel,
 932             {IP_WEAK_ES, IP_STRONG_ES, IP_WEAK_ES}, {IP_WEAK_ES} },

 934         { "hostmodel", MOD_PROTO_IPV6, ip_set_hostmodel, ip_get_hostmodel,
 935             {IP_WEAK_ES, IP_STRONG_ES, IP_WEAK_ES}, {IP_WEAK_ES} },

 937         { "?", MOD_PROTO_IP, NULL, mod_get_allprop, {0}, {0} },

 939         { NULL, 0, NULL, NULL, {0}, {0} }
 940 };
```

unchanged_portion_omitted_