Anexo I: Snort

```
/*
 *
 * Function: main(int, char *)
 *
 * Purpose:  Handle program entry and exit, call main prog sections
 *           This can handle both regular (command-line) style
 *           startup, as well as Win32 Service style startup.
 *
 * Arguments: See command line args in README file
 *
 * Returns: 0 => normal exit, 1 => exit on error
 *
 */
int main(int argc, char *argv[])
{

    [...]

    snort_argc = argc;
    snort_argv = argv;

    return SnortMain(argc, argv);
}
```

Código 1: Función main de Snort en snort.c

```
/*
 *
 * Function: SnortMain(int, char *)
 *
 * Purpose:  The real place that the program handles entry and exit.  Called
 *           called by main(), or by SnortServiceMain().
 *
 * Arguments: See command line args in README file
 *
 * Returns: 0 => normal exit, 1 => exit on error
 *
 */
int SnortMain(int argc, char *argv[])
{
    [...]

    SnortInit(argc, argv);

    [...]
}
```

Código 2: Función SnortMain situada en snort.c (fase offline)

```
static void SnortInit(int argc, char **argv)
{
    InitSignals();

    [...]

    InitGlobals();

    /* chew up the command line */
    ParseCmdLine(argc, argv);

    [...]

    LogMessage("\n");
    LogMessage("        --== Initializing Snort ==--\n");

    if (!ScVersionMode())
    {
        /* Every run mode except version will potentially need output
         * If output plugins should become dynamic, this needs to move */
        RegisterOutputPlugins();
        [...]
    }

    /* if we're using the rules system, it gets initialized here */
    if (snort_conf_file != NULL)
    {
        SnortConfig *sc;

        /* initialize all the plugin modules */
        RegisterPreprocessors();
        RegisterRuleOptions();

        [...]

        LogMessage("Parsing Rules file \"%s\"\n", snort_conf_file);
        sc = ParseSnortConf();

        /* Merge the command line and config file confs to take care of
         * command line overriding config file.
         * Set the global snort_conf that will be used during run time */
        snort_conf = MergeSnortConfs(snort_cmd_line_conf, sc);

        [...]

    }
    else if (ScPacketLogMode() || ScPacketDumpMode())
    {
        /* Make sure there is a log directory */
        /* This will return the cmd line conf and resolve the output
         * configuration */
        SnortConfig* sc = ParseSnortConf();
        snort_conf = MergeSnortConfs(snort_cmd_line_conf, sc);

        [...]

    }

    [...]

    ConfigureOutputPlugins(snort_conf);

    /* Have to split up configuring preprocessors between internal and dynamic
     * because the dpd structure has a pointer to the stream api and stream5
     * needs to be configured first to set this */
    ConfigurePreprocessors(snort_conf, 0);
```

[1/2]

Código 3: Función SnortInit situada en snort.c (parte 1/2)

```
#ifdef DYNAMIC_PLUGIN
    InitDynamicEngines(snort_conf->dynamic_rules_path);

    if (ScRuleDumpMode())
    {
        if( snort_conf->dynamic_rules_path == NULL )
        {
            FatalError("%s(%d) Please specify the directory path for dumping the dynamic
                       rules \n", __FILE__, __LINE__);
        }

        DumpDetectionLibRules();
        CleanExit(0);
    }

    /* This will load each dynamic preprocessor module specified and set
     * the _dpd structure for each */
    InitDynamicPreprocessors();
#endif

    /* Now configure the dynamic preprocessors since the dpd structure
     * should be filled in and have the correct values */
    ConfigurePreprocessors(snort_conf, 1);

    ParseRules(snort_conf);
    RuleOptParseCleanup();

#ifdef DYNAMIC_PLUGIN
    InitDynamicDetectionPlugins(snort_conf);
#endif

    [...]

    fpCreateFastPacketDetection(snort_conf);

    [...]

}
```

[2/2]

Código 4: Función SnortInit situada en snort.c (parte 2/2)

```
typedef struct {

    PortTable * tcp_src, * tcp_dst;
    PortTable * udp_src, * udp_dst;
    PortTable * icmp_src,* icmp_dst;
    PortTable * ip_src,  * ip_dst;

    PortObject * tcp_anyany;
    PortObject * udp_anyany;
    PortObject * icmp_anyany;
    PortObject * ip_anyany;

    PortObject * tcp_nocontent;
    PortObject * udp_nocontent;
    PortObject * icmp_nocontent;
    PortObject * ip_nocontent;

}rule_port_tables_t;
```

Código 5: Estructura rule_port_tables_t situada en sfportobject.h

```
/*
*  Port list version
*
*  7/2007 - man
*
*  Build Pattern Groups for 1st pass of content searching using
*  multi-pattern search method.
*/
int fpCreateFastPacketDetection(SnortConfig *sc)
{
    rule_port_tables_t *port_tables;
    FastPatternConfig *fp;

    if(!rule_count || (sc == NULL))
        return 0;

    port_tables = sc->port_tables;
    fp = sc->fast_pattern_config;

    if ((port_tables == NULL) || (fp == NULL))
        return 0;

    [...]

    /* Use PortObjects to create PORT_GROUPs */
    if (fpDetectGetDebugPrintRuleGroupBuildDetails(fp))
        LogMessage("Creating Port Groups....\n");

    if (fpCreatePortGroups(sc, port_tables))
        FatalError("Could not create PortGroup objects for PortObjects\n");

    if (fpDetectGetDebugPrintRuleGroupBuildDetails(fp))
        LogMessage("Port Groups Done....\n");

    /* Create rule_maps */
    if (fpDetectGetDebugPrintRuleGroupBuildDetails(fp))
        LogMessage("Creating Rule Maps....\n");

    if (fpCreateRuleMaps(sc, port_tables))
        FatalError("Could not create rule maps\n");

    if (fpDetectGetDebugPrintRuleGroupBuildDetails(fp))
        LogMessage("Rule Maps Done....\n");

    [...]

    return 0;
}
```

Código 6: Función fpCreateFastPacketDetection situada en fpcreate.c

```
/*
 *  Create port group objects for all port tables
 *
 *  note: any-any ports are standard PortObjects not PortObject2's so we have to
 *  uprade them for the create port group function
 */
static int fpCreatePortGroups(SnortConfig *sc, rule_port_tables_t *p)
{
    PortObject2 *po2, *add_any_any = NULL;
    FastPatternConfig *fp = sc->fast_pattern_config;

    if (!rule_count)
        return 0 ;

    /* TCP */
    /* convert the tcp-any-any to a PortObject2 creature */
    po2 = PortObject2Dup(p->tcp_anyany);
    if (po2 == NULL)
        FatalError("Could not create a PortObject version 2 for tcp-any-any rules\n!");

    if (!fpDetectSplitAnyAny(fp))
        add_any_any = po2;

    if (fpDetectGetDebugPrintRuleGroupBuildDetails(fp))
        LogMessage("\nTCP-SRC ");

    if (fpCreatePortTablePortGroups(sc, p->tcp_src, add_any_any))
    {
        LogMessage("fpCreatePorTablePortGroups failed-tcp_src\n");
        return -1;
    }

    if (fpDetectGetDebugPrintRuleGroupBuildDetails(fp))
        LogMessage("\nTCP-DST ");

    if (fpCreatePortTablePortGroups(sc, p->tcp_dst, add_any_any))
    {
        LogMessage("fpCreatePorTablePortGroups failed-tcp_dst\n");
        return -1;
    }

    if (fpDetectGetDebugPrintRuleGroupBuildDetails(fp))
        LogMessage("\nTCP-ANYANY ");

    if (fpCreatePortObject2PortGroup(sc, po2, 0))
    {
        LogMessage("fpCreatePorTablePortGroups failed-tcp any-any\n");
        return -1;
    }

    /* save the any-any port group */
    p->tcp_anyany->data = po2->data;
    p->tcp_anyany->data_free = fpDeletePortGroup;
    po2->data = 0;
    /* release the dummy PortObject2 copy of tcp-any-any */
    PortObject2Free(po2);


    /* UDP */
    [...]
    /* ICMP */
    [...]
    /* IP */

    [...]

    return 0;
}
```

Código 7: Función fpCreatePortGroups situada en fpcreate.c

```
/*
 *  Create the port groups for this port table
 */
static int fpCreatePortTablePortGroups(SnortConfig *sc, PortTable *p, PortObject2 *poaa)
{
   SFGHASH_NODE * node;
   int cnt=1;
   FastPatternConfig *fp = sc->fast_pattern_config;

   if (fpDetectGetDebugPrintRuleGroupBuildDetails(fp))
       LogMessage("%d Port Groups in Port Table\n",p->pt_mpo_hash->count);

   for (node=sfghash_findfirst(p->pt_mpo_hash);  //p->pt_mpxo_hash
        node;
        node=sfghash_findnext(p->pt_mpo_hash) ) //p->pt->mpxo_hash
   {
        PortObject2 * po;

        po = (PortObject2*)node->data;
        if (po == NULL)
            continue;

        if (fpDetectGetDebugPrintRuleGroupBuildDetails(fp))
            LogMessage("Creating Port Group Object %d of %d\n",
                       cnt++,p->pt_mpo_hash->count);

        /* if the object is not referenced, don't add it to the PORT_GROUPs
         * as it may overwrite other objects that are more inclusive. */
        if (!po->port_cnt)
            continue;

        if (fpCreatePortObject2PortGroup(sc, po, poaa))
        {
            LogMessage("fpCreatePortObject2PortGroup() failed\n");
            return -1;
        }

        if (fpDetectGetDebugPrintRuleGroupBuildDetails(fp))
            mpsePrintSummary(fp->search_method);
   }

   return 0;
}
```

Código 8: Función fpCreatePortTablePortGroups situada en fpcreate.c

```
/*
 *  Create the PortGroup for these PortObject2 entitiies
 *
 *  This builds the 1st pass multi-pattern state machines for
 *  content and uricontent based on the rules in the PortObjects
 *  hash table.
 */
static int fpCreatePortObject2PortGroup(SnortConfig *sc, PortObject2 *po, PortObject2
*poaa)
{
    SFGHASH_NODE *node;     unsigned sid, gid;
    OptTreeNode * otn;      PORT_GROUP * pg;
    PortObject2 *pox;       FastPatternConfig *fp = sc->fast_pattern_config;

    /* verify we have a port object */
    if (po == NULL)
        return 0;

    po->data = 0;

    [...]

    /* Check if we have any rules */
    if (po->rule_hash == NULL)
        return 0;

    /* create a port_group */
    pg = (PORT_GROUP *)SnortAlloc(sizeof(PORT_GROUP));

    if (fpAllocPms(pg, fp) != 0)
    {
        free(pg);
        return -1;
    }

    pox = po;

    if (po == NULL)
        pox = poaa;

    while (pox != NULL)
    {
        for (node = sfghash_findfirst(pox->rule_hash);
                node;
                node = sfghash_findnext(pox->rule_hash))
        {
            [...]

            if (fpAddPortGroupRule(pg, otn, fp) != 0)
                continue;
        }

        [...]

        if (pox == poaa)
            break;

        pox = poaa;
    }

    /* This might happen if there was ip proto only rules
     * Don't return failure */
    if (fpFinishPortGroup(pg, fp) != 0)
        return 0;

    [...]

    return 0;
}
```

Código 9: Función fpCreatePortObject2PortGroup situada en fpcreate.c

```
/*
 *
 * Function: SnortMain(int, char *)
 *
 * Purpose:  The real place that the program handles entry and exit.  Called
 *           called by main(), or by SnortServiceMain().
 *
 * Arguments: See command line args in README file
 *
 * Returns: 0 => normal exit, 1 => exit on error
 *
 */
int SnortMain(int argc, char *argv[])
{
    const char* intf;
    int daqInit;

    SnortInit(argc, argv);

    intf = GetPacketSource();
    daqInit = intf || snort_conf->daq_type;

    if ( daqInit )
    {
        DAQ_Init(snort_conf);
        DAQ_New(snort_conf, intf);
    }

    [...]

    PacketLoop();

    // DAQ is shutdown in CleanExit() since we don't always return here
    CleanExit(0);

    return 0;
}
```

Código 10: Función SnortMain situada en snort.c (fase online)m

```
void PacketLoop (void)
{
    int error;
    int pkts_to_read = (int)snort_conf->pkt_cnt;

    TimeStart();

    while ( !exit_logged )
    {
        error = DAQ_Acquire(pkts_to_read, PacketCallback, NULL);

        if ( error )
        {
            if ( !ScReadMode() || !PQ_Next() )
            {
                /* If not read-mode or no next pcap, we're done */
                break;
            }
        }
        /* Check for any pending signals when no packets are read*/
        else
        {
            // TBD SnortIdle() only checks for perf file rotation
            // and that can only be done after calling SignalCheck()
            // so either move SnortIdle() to SignalCheck() or directly
            // set the flag in the signal handler (and then clear it
            // in SnortIdle()).

            // check for signals
            if ( SignalCheck() )
            {
#ifndef SNORT_RELOAD
                // Got SIGHUP
                Restart();
#endif
            }
            CheckForReload();
        }
        if ( pkts_to_read > 0 )
        {
            if ( (long)snort_conf->pkt_cnt <= (long)pc.total_from_daq )
                break;
            else
                pkts_to_read = (long)snort_conf->pkt_cnt - (long)pc.total_from_daq;
        }
        // idle time processing..quick things to check or do ...
        // TBD fix this per above ... and if it stays here, should
        // prolly change the name if acquire breaks due to a signal
        // (since in that case we aren't idle here)
        SnortIdle();
    }

    if ( !exit_logged && (error < 0) )
    {
        if ( error == DAQ_READFILE_EOF )
            error = 0;
        CleanExit(error);
    }
    done_processing = 1;
}
```

Código 11: Función PacketLoop situada en snort.c

```
static DAQ_Verdict PacketCallback(
    void* user, const DAQ_PktHdr_t* pkthdr, const uint8_t* pkt)
{
    DAQ_Verdict verdict = DAQ_VERDICT_PASS;

#ifdef EXIT_CHECK
    if (snort_conf->exit_check && (pc.total_from_daq >= snort_conf->exit_check))
        ExitCheckStart();
#endif

    /* First thing we do is process a Usr signal that we caught */
    if (SignalCheck())
    {
#ifndef SNORT_RELOAD
        /* Got SIGHUP */
        Restart();
#endif
    }

    pc.total_from_daq++;

    /* Increment counter that we're evaling rules for caching results */
    rule_eval_pkt_count++;

#ifdef TARGET_BASED
    /* Load in a new attribute table if we need to... */
    AttributeTableReloadCheck();
#endif

    CheckForReload();

    /* Save off the time of each and every packet */
    packet_time_update(pkthdr->ts.tv_sec);

    /* reset the thresholding subsystem checks for this packet */
    sfthreshold_reset();

    SnortEventqReset();
    Replace_ResetQueue();
#ifdef ACTIVE_RESPONSE
    Active_ResetQueue();
#endif

    [...]

    verdict = ProcessPacket(user, pkthdr, pkt, NULL);

    return verdict;
}
```

Código 12: Función PacketCallback situada en snort.c

```
DAQ_Verdict ProcessPacket(
    void* user, const DAQ_PktHdr_t* pkthdr, const uint8_t* pkt, void* ft)
{
    Packet p;
    DAQ_Verdict verdict = DAQ_VERDICT_PASS;
    int inject = 0;

    setRuntimePolicy(getDefaultPolicy());

    /* call the packet decoder */
    (*grinder) (&p, pkthdr, pkt);

    if(!p.pkth || !p.pkt)
        return verdict;

    /* Make sure this packet skips the rest of the preprocessors */
    /* Remove once the IPv6 frag code is moved into frag 3 */
    if(p.packet_flags & PKT_NO_DETECT)
        DisableAllDetect(&p);

    [...]

    /* just throw away the packet if we are configured to ignore this port */
    if ( !(p.packet_flags & PKT_IGNORE_PORT) )
    {
        /* start calling the detection processes */
        Preprocess(&p);
        log_func(&p);
    }

    if ( Active_SessionWasDropped() )
    {
        Active_DropAction(&p);

        if ( ScInlineMode() )
            verdict = DAQ_VERDICT_BLACKLIST;
        else
            verdict = DAQ_VERDICT_IGNORE;
    }
    if ( ft )
    {
        // we don't block, modify, pass, or count defrags
        // if the defrag trigged a block, this verdict will
        // be applied to the raw packet.
        return verdict;
    }

    [...]

    return verdict;
}
```

Código 13: Función ProcessPacket situada en snort.c

```
int Preprocess(Packet * p)
{
    ...
   // If the packet has errors, we won't analyze it.
    if ( p->error_flags )
    { ... }
    else
    {
        PreprocEvalFuncNode *idx = policy->preproc_eval_funcs;
        ...
        do_detect = do_detect_content = 1;
        ...
       /* Turn on all preprocessors */
        EnablePreprocessors(p);

        if ( p->dsize )
            for (; (idx != NULL) && !(p->packet_flags & PKT_PASS_RULE); idx = idx->next)
                if (((p->proto_bits & idx->proto_mask) ||
                     (idx->proto_mask == PROTO_BIT__ALL)) &&
                      IsPreprocBitSet(p, idx->preproc_bit))
                    idx->func(p, idx->context);
        else
        {
            for (; (idx != NULL) && !(p->packet_flags & PKT_PASS_RULE); idx = idx->next)
            {
                // short-circuit here if no app data
                if ( idx->priority >= PRIORITY_APPLICATION )
                    break;
                if (((p->proto_bits & idx->proto_mask) ||
                     (idx->proto_mask == PROTO_BIT__ALL)) &&
                      IsPreprocBitSet(p, idx->preproc_bit))
                    idx->func(p, idx->context);
            }
            DisableDetect(p);
        }
       if ((do_detect) && (p->bytes_to_inspect != -1))
        {
            /* Check if we are only inspecting a portion of this packet... */
            if (p->bytes_to_inspect > 0)
                p->dsize = (uint16_t)p->bytes_to_inspect;
            Detect(p);
        }
        ...
    }
    /* Simulate above behavior for preprocessor reassembled packets */
    if ((p->packet_flags & PKT_PREPROC_RPKT) && do_detect && (p->bytes_to_inspect !=-1))
    {
        PreprocReassemblyPktFuncNode *rpkt_idx = policy->preproc_reassembly_pkt_funcs;
        /* Loop through the preprocessors that have registered a
         * function to get a reassembled packet */
        while (rpkt_idx != NULL)
        {
            Packet *pp = NULL;
            /* If the preprocessor bit is set, get the reassembled packet */
            if (IsPreprocReassemblyPktBitSet(p, rpkt_idx->preproc_id))
                pp = (Packet *)rpkt_idx->func();
            if (pp != NULL)
            {
                if (p->bytes_to_inspect > 0)
                    pp->dsize = (uint16_t)p->bytes_to_inspect;

                Detect(pp);
            }
            rpkt_idx = rpkt_idx->next;
        }
    }
   return retval;
}
```

Código 14: Función Preprocess situada en detect.c

```
/*****************************************************************************
 *
 * Function: Detect(Packet *)
 *
 * Purpose: Apply the rules lists to the current packet
 *
 * Arguments: p => ptr to the decoded packet struct
 *
 * Returns: 1 == detection event
 *          0 == no detection
 *
 *****************************************************************************/
int Detect(Packet * p)
{
    int detected = 0;

    if ((p == NULL) || !IPH_IS_VALID(p))
        return 0;

    if (!snort_conf->ip_proto_array[GET_IPH_PROTO(p)])
    {
#ifdef GRE
# ifdef SUP_IP6
        switch (p->outer_family)
        {
            case AF_INET:
                if (!snort_conf->ip_proto_array[p->outer_ip4h.ip_proto])
                    return 0;
                break;

            case AF_INET6:
                if (!snort_conf->ip_proto_array[p->outer_ip6h.next])
                    return 0;
                break;

            default:
                return 0;
        }
# else
        if ((p->outer_iph == NULL) ||
             !snort_conf->ip_proto_array[p->outer_iph->ip_proto])
            return 0;
# endif  /* SUP_IP6 */
#else
        return 0;
#endif  /* GRE */
    }

    if (p->packet_flags & PKT_PASS_RULE)
    {
        /* If we've already seen a pass rule on this,
         * no need to continue do inspection.
         */
        return 0;
    }

    [...]

    /*
    **  This is where we short circuit so
    **  that we can do IP checks.
    */
    detected = fpEvalPacket(p);

    return detected;
}
```

Código 15: Función Detect situada en detect.c

```
/*
**
**  NAME
**    fpEvalPacket::
**
**  DESCRIPTION
**    This function is the interface to the Detect() routine.  Here
**    the IP protocol is processed.  If it is TCP, UDP, or ICMP, we
**    process the both that particular ruleset and the IP ruleset
**    with in the fpEvalHeader for that protocol.  If the protocol
**    is not TCP, UDP, or ICMP, we just process the packet against
**    the IP rules at the end of the fpEvalPacket routine.  Since
**    we are using a setwise methodology for snort rules, both the
**    network layer rules and the transport layer rules are done
**    at the same time.  While this is not the best for modularity,
**    it is the best for performance, which is what we are working
**    on currently.
**
**  FORMAL INPUTS
**    Packet * - the packet to inspect
**
**  FORMAL OUTPUT
**    int - 0 means that packet has been processed.
**
*/
int fpEvalPacket(Packet *p)
{
    int ip_proto = GET_IPH_PROTO(p);
    OTNX_MATCH_DATA *omd = snort_conf->omd;

    /* Run UDP rules against the UDP header of Teredo packets */
    [...]

    switch(ip_proto)
    {
        case IPPROTO_TCP:
            DEBUG_WRAP(DebugMessage(DEBUG_DETECT, "Detecting on TcpList\n"););
            if(p->tcph == NULL)
            {
                ip_proto = -1;
                break;
            }
            return fpEvalHeaderTcp(p, omd);
        case IPPROTO_UDP:
            DEBUG_WRAP(DebugMessage(DEBUG_DETECT, "Detecting on UdpList\n"););
            if(p->udph == NULL)
            {
                ip_proto = -1;
                break;
            }
            return fpEvalHeaderUdp(p, omd);
#ifdef SUP_IP6
        case IPPROTO_ICMPV6:
#endif
        case IPPROTO_ICMP:
            DEBUG_WRAP(DebugMessage(DEBUG_DETECT, "Detecting on IcmpList\n"););
            if(p->icmph == NULL)
            {
                ip_proto = -1;
                break;
            }
            return fpEvalHeaderIcmp(p, omd);
        default:
            break;
    }

    /* No Match on TCP/UDP, Do IP */
    return fpEvalHeaderIp(p, ip_proto, omd);
}
```

Código 16: Función fpEvalPacket situada en fpdetect.c

```
static inline int fpEvalHeaderTcp(Packet *p, OTNX_MATCH_DATA *omd)
{
    PORT_GROUP *src = NULL, *dst = NULL, *gen = NULL;

#ifdef TARGET_BASED
    if (IsAdaptiveConfigured(getRuntimePolicy(), 0))
    {
        int16_t proto_ordinal = GetProtocolReference(p);
        DEBUG_WRAP(DebugMessage(DEBUG_ATTRIBUTE, "proto_ordinal=%d\n", proto_ordinal););

        if (proto_ordinal > 0)
        {
            if (p->packet_flags & PKT_FROM_SERVER) /* to cli */
            {
                DEBUG_WRAP(DebugMessage(DEBUG_ATTRIBUTE, "pkt_from_server\n"););
                src = fpGetServicePortGroupByOrdinal(snort_conf->sopgTable, IPPROTO_TCP,
                                                     0 /*to_cli */,  proto_ordinal);
            }
            if (p->packet_flags & PKT_FROM_CLIENT) /* to srv */
            {
                DEBUG_WRAP(DebugMessage(DEBUG_ATTRIBUTE, "pkt_from_client\n"););
                dst = fpGetServicePortGroupByOrdinal(snort_conf->sopgTable, IPPROTO_TCP,
                                                     1 /*to_srv */,  proto_ordinal);
            }
            DEBUG_WRAP(DebugMessage(DEBUG_ATTRIBUTE,
                        "fpEvalHeaderTcp:targetbased-ordinal-lookup: "
                        "sport=%d, dport=%d, proto_ordinal=%d, src:%x, "
                        "dst:%x, gen:%x\n",p->sp,p->dp,proto_ordinal,src,dst,gen););
        }
    }

    if ((src == NULL) && (dst == NULL))
    {
        /* we did not have a target based group, use ports */
        if (!prmFindRuleGroupTcp(snort_conf->prmTcpRTNX,p->dp,p->sp, &src,&dst,&gen))
            return 0;

        DEBUG_WRAP(DebugMessage(DEBUG_ATTRIBUTE,
                    "fpEvalHeaderTcp: sport=%d, "
                    "dport=%d, src:%x, dst:%x, gen:%x\n",p->sp,p->dp,src,dst,gen););
    }
#else
    if (!prmFindRuleGroupTcp(snort_conf->prmTcpRTNX,p->dp,p->sp, &src,&dst,&gen))
        return 0;
#endif

    if (fpDetectGetDebugPrintNcRules(snort_conf->fast_pattern_config))
        LogMessage("fpEvalHeaderTcp: sport=%d, dport=%d, src:%p, dst:%p, gen:%p\n",
                    p->sp, p->dp, (void*)src, (void*)dst, (void*)gen);

    InitMatchInfo(omd);

    if (dst != NULL)
        if (fpEvalHeaderSW(dst, p, 1, 0, omd))
            return 1;

    if (src != NULL)
        if (fpEvalHeaderSW(src, p, 1, 0, omd))
            return 1;

    if (gen != NULL)
        if(fpEvalHeaderSW(gen, p, 1, 0, omd))
            return 1;

    return fpFinalSelectEvent(omd, p);
}
```

Código 17: Función fpEvalHeaderTcp situada en fpdetect.c

```
static inline int fpEvalHeaderUdp(Packet *p, OTNX_MATCH_DATA *omd)
{
    PORT_GROUP *src = NULL, *dst = NULL, *gen = NULL;

#ifdef TARGET_BASED
    if (IsAdaptiveConfigured(getRuntimePolicy(), 0))
    {
        /* Check for a service/protocol ordinal for this packet */
        int16_t proto_ordinal = GetProtocolReference(p);
        DEBUG_WRAP( DebugMessage(DEBUG_ATTRIBUTE,"proto_ordinal=%d\n",proto_ordinal););

        if (proto_ordinal > 0)
        {
            /* TODO:  To From Server ?, else we apply  */
            dst = fpGetServicePortGroupByOrdinal(snort_conf->sopgTable, IPPROTO_UDP,
                                                 TO_SERVER, proto_ordinal);
            src = fpGetServicePortGroupByOrdinal(snort_conf->sopgTable, IPPROTO_UDP,
                                                 TO_CLIENT, proto_ordinal);
            DEBUG_WRAP(DebugMessage(DEBUG_ATTRIBUTE,
                        "fpEvalHeaderUdpp:targetbased-ordinal-lookup: "
                        "sport=%d, dport=%d, proto_ordinal=%d, src:%x, "
                        "dst:%x, gen:%x\n",p->sp,p->dp,proto_ordinal,src,dst,gen););
        }
    }

    if ((src == NULL) && (dst == NULL))
    {
        /* we did not have a target based port group, use ports */
        if (!prmFindRuleGroupUdp(snort_conf->prmUdpRTNX,p->dp,p->sp, &src,&dst,&gen))
            return 0;

        DEBUG_WRAP(DebugMessage(DEBUG_ATTRIBUTE,
                    "fpEvalHeaderUdp: sport=%d, dport=%d, "
                    "src:%x, dst:%x, gen:%x\n",p->sp,p->dp,src,dst,gen););
    }
#else
    if (!prmFindRuleGroupUdp(snort_conf->prmUdpRTNX,p->dp,p->sp, &src,&dst,&gen))
        return 0;
#endif

    if (fpDetectGetDebugPrintNcRules(snort_conf->fast_pattern_config))
        LogMessage("fpEvalHeaderUdp: sport=%d, dport=%d, src:%p, dst:%p, gen:%p\n",
                    p->sp, p->dp, (void*)src, (void*)dst, (void*)gen);

    InitMatchInfo(omd);

    if (dst != NULL)
        if (fpEvalHeaderSW(dst, p, 1, 0, omd))
            return 1;

    if (src != NULL)
        if (fpEvalHeaderSW(src, p, 1, 0, omd))
            return 1;

    if (gen != NULL)
        if (fpEvalHeaderSW(gen, p, 1, 0, omd))
            return 1;

    return fpFinalSelectEvent(omd, p);
}
```

Código 18: Función fpEvalHeaderUdp situada en fpdetect.c

```
static inline int fpEvalHeaderIcmp(Packet *p, OTNX_MATCH_DATA *omd)
{
    PORT_GROUP *gen = NULL, *type = NULL;

    if (!prmFindRuleGroupIcmp(snort_conf->prmIcmpRTNX, p->icmph->type, &type, &gen))
        return 0;

    if (fpDetectGetDebugPrintNcRules(snort_conf->fast_pattern_config))
        LogMessage("fpEvalHeaderIcmp: icmp->type=%d type=%p gen=%p\n",
                    p->icmph->type, (void*)type, (void*)gen);

    InitMatchInfo(omd);

    if (type != NULL)
        if (fpEvalHeaderSW(type, p, 0, 0, omd))
            return 1;

    if (gen != NULL)
        if (fpEvalHeaderSW(gen, p, 0, 0, omd))
            return 1;

    return fpFinalSelectEvent(omd, p);
}
```

Código 19: Función fpEvalHeaderIcmp situada en fpdetect.c

```
static inline int fpEvalHeaderIp(Packet *p, int ip_proto, OTNX_MATCH_DATA *omd)
{
    PORT_GROUP *gen = NULL, *ip_group = NULL;

    if (!prmFindRuleGroupIp(snort_conf->prmIpRTNX, ip_proto, &ip_group, &gen))
        return 0;

    if(fpDetectGetDebugPrintNcRules(snort_conf->fast_pattern_config))
        LogMessage("fpEvalHeaderIp: ip_group=%p, gen=%p\n", (void*)ip_group,(void*)gen);

    InitMatchInfo(omd);

    if (ip_group != NULL)
        if (fpEvalHeaderSW(ip_group, p, 0, 1, omd))
            return 1;

    if (gen != NULL)
        if (fpEvalHeaderSW(gen, p, 0, 1, omd))
            return 1;

    return fpFinalSelectEvent(omd, p);
}
```

Código 20: Función fpEvalHeaderIp situada en fpdetect.c

```
/*
**  NAME
**    fpEvalHeaderSW::
**
**  DESCRIPTION
**    This function does a set-wise match on content, and walks an otn list
**    for non-content.  The otn list search will eventually be redone for
**    for performance purposes.
**
*/
static inline int fpEvalHeaderSW(PORT_GROUP *port_group, Packet *p,
        int check_ports, char ip_rule, OTNX_MATCH_DATA *omd)
{
    RULE_NODE *rnWalk;              void * so;
    int start_state;                const uint8_t *tmp_payload = p->data;
    uint16_t tmp_dsize = p->dsize;  void *tmp_iph = (void *)p->iph;
    FastPatternConfig *fp = snort_conf->fast_pattern_config;
    [...]
    if (do_detect_content)
    {
        /* First evaluate the detection functions. Namely those things
           that are between a preprocessor and rules. */
        [...]
        if (fp->inspect_stream_insert || !(p->packet_flags & PKT_STREAM_INSERT))
        {
            omd->pg = port_group; omd->p = p;
            omd->check_ports = check_ports;
            /* Uri-Content Match
               This check indicates that http_decode found at least one uri */
            if (p->uri_count > 0)
            {
                int i;
                for (i = HTTP_BUFFER_URI;
                    (i < p->uri_count) && (i <= HTTP_BUFFER_CLIENT_BODY); i++)
                {
                    if ((UriBufs[i].uri == NULL) || (UriBufs[i].length == 0))
                        continue;
                    switch (i)
                    {
                        case HTTP_BUFFER_URI:
                            so = (void *)port_group->pgPms[PM_TYPE__HTTP_URI_CONTENT];
                            break;
                        case HTTP_BUFFER_HEADER:
                            so = (void *)port_group->
                                pgPms[PM_TYPE__HTTP_HEADER_CONTENT];
                            break;
                        case HTTP_BUFFER_CLIENT_BODY:
                            so = (void *)port_group->
                                pgPms[PM_TYPE__HTTP_CLIENT_BODY_CONTENT];
                            break;
                        default:
                            so = NULL;
                            break;
                    }
                    if ((so != NULL) && (mpseGetPatternCount(so) > 0))
                    {
                        start_state = 0;
                        mpseSearch(so, UriBufs[i].uri, UriBufs[i].length,
                                rule_tree_match, omd, &start_state);
                    }
                }
            }
```

[1/2]

Código 21: Función fpEvalHeaderSW situada en fpdetect.c (parte 1/2)

```
            /*
             **  Decode Content Match
             **  We check to see if the packet has been normalized into
             **  the global (decode.c) DecodeBuffer.  Currently, only
             **  telnet normalization writes to this buffer.  So, if
             **  it is set, we do this the match against the normalized
             **  buffer and we do the check against the original
             **  payload, in case any of the rules have the
             **  'rawbytes' option.
             */
            so = (void *)port_group->pgPms[PM_TYPE__CONTENT];
            if ((so != NULL) && (mpseGetPatternCount(so) > 0))
            {
                if (Is_DetectFlag(FLAG_ALT_DECODE) && DecodeBuffer.len)
                {
                    start_state = 0;
                    mpseSearch(so, DecodeBuffer.data, DecodeBuffer.len,
                            rule_tree_match, omd, &start_state);
                }

                /* Adding this extra search on file data since we no more use
                   DecodeBuffer to decode now */
                if(file_data_ptr.len)
                {
                    start_state = 0;
                    mpseSearch(so, file_data_ptr.data, file_data_ptr.len,
                        rule_tree_match, omd, &start_state);
                }

                if (p->data && p->dsize)
                {
                    uint16_t pattern_match_size = p->dsize;

                    if ( IsLimitedDetect(p) && (p->alt_dsize < p->dsize) )
                        pattern_match_size = p->alt_dsize;

                    start_state = 0;
                    mpseSearch(so, p->data, pattern_match_size,
                            rule_tree_match, omd, &start_state);
                }
            }
        }
    }

    /*
     **  Walk and test the non-content OTNs
     */

    [...]

    return 0;
}
```


Código 22: Función fpEvalHeaderSW situada en fpdetect.c (parte 2/2)