Anexo III: Integración

```
[...]
#define DETECTION_OPT__SEARCH_METHOD                            "search-method"
#define DETECTION_OPT__SEARCH_OPTIMIZE                          "search-optimize"
#define DETECTION_OPT__SPLIT_ANY_ANY                            "split-any-any"
#define DETECTION_OPT__MAX_PATTERN_LEN                          "max-pattern-len"
//+rbw
#define DETECTION_OPT__FILTER                                   "filter"
//
[...]

void ConfigDetection(SnortConfig *sc, char *args)
{
    int i;
    char **toks;
    int num_toks;
    FastPatternConfig *fp;
    [...]
    fp = sc->fast_pattern_config;
    [...]
    toks = mSplit(args, ", ", 20, &num_toks, 0);
    for (i = 0; i < num_toks; i++)
    {
        if (strcasecmp(toks[i], DETECTION_OPT__SEARCH_OPTIMIZE) == 0)
            fpSetDetectSearchOpt(fp, 1);
        [...]
        else if (strcasecmp(toks[i], DETECTION_OPT__SEARCH_METHOD) == 0)
        {
            i++;
            if (i < num_toks)
                if (fpSetDetectSearchMethod(fp, toks[i]) == -1)
                    ParseError("Invalid argument to 'search-method': %s.", toks[i]);
            else
                ParseError("Need an argument to 'search-method'.");
        }
        [...]
        else if (strcasecmp(toks[i], DETECTION_OPT__SPLIT_ANY_ANY) == 0)
            fpDetectSetSplitAnyAny(fp, 1);
        else if (strcasecmp(toks[i], DETECTION_OPT__MAX_PATTERN_LEN) == 0)
        {
            i++;
            if (i < num_toks)
            {
                char *endptr;
                int n = SnortStrtol(toks[i], &endptr, 0);
                if ((errno == ERANGE) || (*endptr != '\0') || (n < 0))
                    ParseError("Invalid argument for max-pattern-len: %s.  "
                               "Need a non-negative integer.", toks[i]);
                fpSetMaxPatternLen(fp, n);
            }
            else
                ParseError("Missing argument to 'max-pattern-len'.");
        }
//+rbw
        else if (strcasecmp(toks[i], DETECTION_OPT__FILTER) == 0)
        {
            i++;
            if (i < num_toks)
                if (fpSetDetectFilter(fp, toks[i]) == -1)
                    ParseError("Invalid argument to 'filter': %s.", toks[i]);
            else
                ParseError("Need an argument to 'filter'.");
        }
//
        [...]
    }
    [...]
}
[...]
```

Código 1: Fichero parser.c modificado

```
[...]
/*
*  Pattern Matching Methods
*/
//#define MPSE_MWM       1
#define MPSE_AC        2
//#define MPSE_KTBM      3
#define MPSE_LOWMEM    4
//#define MPSE_AUTO      5
#define MPSE_ACF       6
#define MPSE_ACS       7
#define MPSE_ACB       8
#define MPSE_ACSB      9
#define MPSE_AC_BNFA   10
#define MPSE_AC_BNFA_Q 11
#define MPSE_ACF_Q     12
#define MPSE_LOWMEM_Q  13

#ifdef INTEL_SOFT_CPM
#define MPSE_INTEL_CPM 14
#endif /* INTEL_SOFT_CPM */

//+rbw
/*
*  Filters
*/
#define MPSE_FILTER_RB      21      //redborder filter
#define MPSE_FILTER_JRB     22      //just redborder filter
//
[...]
```

Código 2: Fichero mpse.h modificado

```
[...]
/*
**  This structure holds configuration options for the detection engine.
*/
typedef struct _FastPatternConfig
{
    int inspect_stream_insert;
    int search_method;
    int search_opt;
    int search_method_verbose;
    int debug;
    unsigned int max_queue_events;
    unsigned int bleedover_port_limit;
    int configured;
    int portlists_flags;
    int split_any_any;
    int max_pattern_len;
    int num_patterns_truncated;  /* due to max_pattern_len */
    int num_patterns_trimmed;    /* due to zero byte prefix */
    int debug_print_fast_pattern;
//+rbw
    int filter;
//

} FastPatternConfig;
[...]
int fpSetDetectSearchMethod(FastPatternConfig *, char *);
//+rbw
int fpSetDetectFilter(FastPatternConfig *, char *);
//
void fpSetDetectSearchOpt(FastPatternConfig *, int flag);
void fpDetectSetSplitAnyAny(FastPatternConfig *, int);
void fpSetMaxPatternLen(FastPatternConfig *, unsigned int);
[...]
```

Código 3: Fichero fpcreate.h modificado

```
[...]

/*
   Search method is set using:
   config detect: search-method ac-bnfa | ac | ac-full | ac-sparsebands |
                                ac-sparse | ac-banded | ac-std | verbose
*/
int fpSetDetectSearchMethod(FastPatternConfig *fp, char *method)
{
    LogMessage("Detection:\n");

//+rbw
    fp->filter = 0;
//
    if( !strcasecmp(method,"ac-std") )
    {
       fp->search_method = MPSE_AC;
       LogMessage("   Search-Method = AC-Std\n");
    }
    [...]
    else if( !strcasecmp(method,"ac-q") || !strcasecmp(method,"ac") )
    {
       fp->search_method = MPSE_ACF_Q;
       LogMessage("   Search-Method = AC-Full-Q\n");
    }
    else if( !strcasecmp(method,"ac-split") )
    {
        fp->search_method = MPSE_ACF_Q;
        fp->split_any_any = 1;
        LogMessage("   Search-Method = AC-Full-Q\n");
        LogMessage("    Split Any/Any group = enabled\n");
    }
    [...]
    else
       return -1;

    return 0;
}

//+rbw
/*
*  Traffic filtering is set using: redborder | justredb
*/
int fpSetDetectFilter(FastPatternConfig *fp, char *filter)
{
    if( !strcasecmp(filter,"redborder") )
    {
        fp->filter = MPSE_FILTER_RB;
        LogMessage("    Traffic Filtering = redborder\n");
    }
    else if( !strcasecmp(filter,"justredb") )
    {
        fp->filter = MPSE_FILTER_JRB;
        LogMessage("    Traffic Filtering = just redBorder (for test purposes)\n");
    }
    else
       return -1;

    return 0;
}
//

[...]
```

Código 4: Fichero fpcreate.c modificado

```
typedef struct _mpse_struct {

    int    method;
    void * obj;
//+rbw
    void * obj_filter;
    int filter;
//
    int    verbose;
    uint64_t bcnt;
    char   inc_global_counter;

} MPSE;
```

Código 5: Estructura MPSE modificada y situada en mpse.c

```
int mpsePrintSummary(int method)
{
//+rbw
    int filter = snort_conf->fast_pattern_config->filter;
    if (filter)
    {
        switch (filter)
        {
            case MPSE_FILTER_RB:
            case MPSE_FILTER_JRB:
                redbPrintSummaryInfo();
                break;
        }
    }
//
    switch (method)
    {
        case MPSE_AC_BNFA:
        case MPSE_AC_BNFA_Q:
            bnfaPrintSummary();
            break;
        case MPSE_AC:
            acsmPrintSummaryInfo();
            break;
        case MPSE_ACF:
        case MPSE_ACF_Q:
        case MPSE_ACS:
        case MPSE_ACB:
        case MPSE_ACSB:
            acsmPrintSummaryInfo2();
            break;
        case MPSE_LOWMEM:
        case MPSE_LOWMEM_Q:
            if( KTrieMemUsed() )
            {
                double x;
                x = (double) KTrieMemUsed();
                LogMessage("[ LowMem Search-Method Memory Used : %g %s ]\n",
                        (x > 1.e+6) ?  x/1.e+6 : x/1.e+3,
                        (x > 1.e+6) ? "MBytes" : "KBytes" );
            }
            break;
        default:
            break;
    }
#ifdef INTEL_SOFT_CPM
    IntelPmPrintSummary();
#endif
    return 0;
}
```

Código 6: Función mpsePrintSummary modificada y situada en mpse.c

```
void * mpseNew(int method, int use_global_counter_flag, void (*userfree)(void *p),
               void (*optiontreefree)(void **p), void (*neg_list_free)(void **p))
{
    MPSE * p;
    p = (MPSE*)calloc( 1,sizeof(MPSE) );
    if( !p ) return NULL;

    p->method = method;
    p->verbose = 0;
    p->obj = NULL;
//+rbw
    p->obj_filter = NULL;
    p->filter = snort_conf->fast_pattern_config->filter;
    if (p->filter)
    {
        switch (p->filter)
        {
            case MPSE_FILTER_RB:
            case MPSE_FILTER_JRB:
                p->obj_filter = redbNew();
                break;
        }
    }
//
    p->bcnt = 0;
    p->inc_global_counter = (char)use_global_counter_flag;
    switch( method )
    {
        [...]
        case MPSE_AC:
            p->obj = acsmNew(userfree, optiontreefree, neg_list_free);
            break;
        case MPSE_ACF:
            p->obj = acsmNew2(userfree, optiontreefree, neg_list_free);
            if(p->obj)acsmSelectFormat2((ACSM_STRUCT2*)p->obj,ACF_FULL  );
            break;
        case MPSE_ACF_Q:
            p->obj = acsmNew2(userfree, optiontreefree, neg_list_free);
            if(p->obj)acsmSelectFormat2((ACSM_STRUCT2*)p->obj,ACF_FULLQ  );
            break;
        case MPSE_ACS:
            p->obj = acsmNew2(userfree, optiontreefree, neg_list_free);
            if(p->obj)acsmSelectFormat2((ACSM_STRUCT2*)p->obj,ACF_SPARSE  );
            break;
        case MPSE_ACB:
            p->obj = acsmNew2(userfree, optiontreefree, neg_list_free);
            if(p->obj)acsmSelectFormat2((ACSM_STRUCT2*)p->obj,ACF_BANDED  );
            break;
        case MPSE_ACSB:
            p->obj = acsmNew2(userfree, optiontreefree, neg_list_free);
            if(p->obj)acsmSelectFormat2((ACSM_STRUCT2*)p->obj,ACF_SPARSEBANDS  );
            break;
        [...]
        default:
            /* p is free'd below if no case */
            break;
    }

    if( !p->obj )
    {
//+rbw-5jul
        if (p->obj_filter) free (p->obj_filter);
//
        free(p); p = NULL;
    }

    return (void *)p;
}
```

Código 7: Función mpseNew modificada y situada en mpse.c

```
int mpseAddPattern(void * pvoid, void * P, int m, unsigned noCase, unsigned offset,
                   unsigned depth, unsigned negative, void* ID, int IID)
{
  MPSE * p = (MPSE*)pvoid;

//+rbw
    if (p->obj_filter)
    {
        switch (p->filter)
        {
            case MPSE_FILTER_RB:
            case MPSE_FILTER_JRB:
                redbAddPattern( (REDB_STRUCT *)p->obj_filter, (unsigned char *)P, m);
                break;
        }
    }
//

  switch( p->method )
   {
     case MPSE_AC_BNFA:
     case MPSE_AC_BNFA_Q:
       return bnfaAddPattern( (bnfa_struct_t*)p->obj, (unsigned char *)P, m,
              noCase, negative, ID );

     case MPSE_AC:
       return acsmAddPattern( (ACSM_STRUCT*)p->obj, (unsigned char *)P, m,
              noCase, offset, depth, negative, ID, IID );

     case MPSE_ACF:
     case MPSE_ACF_Q:
     case MPSE_ACS:
     case MPSE_ACB:
     case MPSE_ACSB:
       return acsmAddPattern2( (ACSM_STRUCT2*)p->obj, (unsigned char *)P, m,
              noCase, offset, depth, negative, ID, IID );

     case MPSE_LOWMEM:
     case MPSE_LOWMEM_Q:
       return KTrieAddPattern( (KTRIE_STRUCT *)p->obj, (unsigned char *)P, m,
                               noCase, negative, ID );

#ifdef INTEL_SOFT_CPM
     case MPSE_INTEL_CPM:
       return IntelPmAddPattern((IntelPm *)p->obj, (unsigned char *)P, m,
               noCase, negative, ID, IID);
#endif

     default:
       return -1;
   }
}
```

Código 8: Función mpseAddPattern modificada y situada en mpse.c

```
int mpsePrepPatterns (void *pvoid, int (*build_tree)(void *id, void **existing_tree),
                      int (*neg_list_func)(void *id, void **list) )
{
  int retv;
  MPSE * p = (MPSE*)pvoid;

//+rbw
    if (p->obj_filter)
    {
        int retf;
        switch (p->filter)
        {
            case MPSE_FILTER_RB:
            case MPSE_FILTER_JRB:
                retf = redbCompile((REDB_STRUCT *)p->obj_filter);
                break;
        }
    }
//

  switch( p->method )
   {
     case MPSE_AC_BNFA:
     case MPSE_AC_BNFA_Q:
       retv = bnfaCompile( (bnfa_struct_t*) p->obj, build_tree, neg_list_func );
     break;

     case MPSE_AC:
       retv = acsmCompile( (ACSM_STRUCT*) p->obj, build_tree, neg_list_func );
     break;

     case MPSE_ACF:
     case MPSE_ACF_Q:
     case MPSE_ACS:
     case MPSE_ACB:
     case MPSE_ACSB:
       retv = acsmCompile2( (ACSM_STRUCT2*) p->obj, build_tree, neg_list_func );
     break;

     case MPSE_LOWMEM:
     case MPSE_LOWMEM_Q:
       return KTrieCompile( (KTRIE_STRUCT *)p->obj, build_tree, neg_list_func );

#ifdef INTEL_SOFT_CPM
     case MPSE_INTEL_CPM:
       return IntelPmFinishGroup((IntelPm *)p->obj, build_tree, neg_list_func);
#endif

     default:
       retv = 1;
     break;
   }

  return retv;
}
```

Código 9: Función mpsePrepPatterns modificada y situada en mpse.c

```
int mpseSearch(void *pvoid, const unsigned char *T, int n,
               int (*action)(void *id,void *tree,int index,void *data,void *neg_list),
               void *data, int *current_state )
{
    MPSE * p = (MPSE*)pvoid;
    int ret;
    PROFILE_VARS;
    PREPROC_PROFILE_START(mpsePerfStats);
    p->bcnt += n;
    if(p->inc_global_counter)
        s_bcnt += n;
//+rbw
    int w = 0; //Aho-Corasick will search for patterns from this position in the payload
    int retrb = 1;
    if (p->obj_filter)
        switch (p->filter)
        {
            case MPSE_FILTER_RB:
                if (!(REDBSEARCH((REDB_STRUCT *)p->obj_filter,(unsigned char *)T,n,&w)))
                {
                    PREPROC_PROFILE_END(mpsePerfStats); return 0;
                }
                break;
            case MPSE_FILTER_JRB:
                PREPROC_PROFILE_END(mpsePerfStats); return 1;
        }
//
    switch( p->method )
    {
        case MPSE_AC_BNFA:     // ac-bnfa-nq
        case MPSE_AC_BNFA_Q:   // ac-bnfa-q | ac-bnfa
            /* return is actually the state */
//+rbw
            //ret = bnfaSearch( (bnfa_struct_t*) p->obj, (unsigned char *)T, n,
                              //action, data, 0 /* start-state */, current_state );
            ret = bnfaSearch( (bnfa_struct_t*) p->obj, (unsigned char *)T, n,
                              action, data, 0 /* start-state */, current_state, w );
//
            PREPROC_PROFILE_END(mpsePerfStats); return ret;
        case MPSE_AC:      // ac-std
//+rbw
            //ret = acsmSearch( (ACSM_STRUCT*) p->obj, (unsigned char *)T, n,
                              //action, data, current_state );
            ret = acsmSearch( (ACSM_STRUCT*) p->obj, (unsigned char *)T, n,
                              action, data, current_state, w );
//
            PREPROC_PROFILE_END(mpsePerfStats); return ret;
        case MPSE_ACF:     // ac-nq
        case MPSE_ACF_Q:   // ac-q | ac | ac-split (also set split_any_any)
        case MPSE_ACS:     // acs
        case MPSE_ACB:     // ac-banded
        case MPSE_ACSB:    // ac-sparsebands
//+rbw
            //ret = acsmSearch2( (ACSM_STRUCT2*) p->obj, (unsigned char *)T, n,
                               //action, data, current_state );
            ret = acsmSearch2( (ACSM_STRUCT2*) p->obj, (unsigned char *)T, n,
                               action, data, current_state, w );
//
            PREPROC_PROFILE_END(mpsePerfStats); return ret;
        [...]
    }
}
```

Código 10: Función mpseSearch modificada y situada en mpse.c