Clamav funcation call flow(AC scan)

ac scan with regex sub-signature

by eqmcc

The call flow

When Clamav doing specific file scan (clamscan.c), there are following procedures:

- Initialize data structures
- Set engine parameters
- Load signatures
- Scan

Data structures

See *flow_normal_sgin_bm_scan.pdf*

Test case

test.txt

```
STARToooTESTkkkMYOtestTEST
```

ndb test
create ndb signature for test.txt
format
MalwareName:TargetType:Offset:HexSignature

File test.ndb

```
test_ndb_partsig:0:3,5:6f6f6f{4-6}6b6b6b
```

```
sudo cp test.ndb /var/lib/clamav/test.ndb
```

Above is a regex(with range) signature with offset info, so it will be loaded into ac pattern structure and will be used in ac scan mode, the signature has a range information and the signature will be split into two sub-sigs.
The virus record will match start offset between 3 and 8 with pattern as "ooo{4-6}kkk"(i.e.: any file with sub string "ooo'any 4-6 bytes'kkk" with the sub string's start at any position between absolute offset 3 and 8 will be identified as virus file)

run

```
user@ubuntu:~/clamav$ clamscan  test.txt
LibClamAV Warning: ***************************************************
LibClamAV Warning: ***  The virus database is older than 7 days!  ***
LibClamAV Warning: ***   Please update it as soon as possible.    ***
LibClamAV Warning: ***************************************************
LibClamAV info: DEBUG: add part sigs for virname=test_ndb_partsig
LibClamAV info: DEBUG: sig parts=2
LibClamAV info: DEBUG: add part 1 of the sig: [6f6f6f] with mindist=0, maxdist=0
LibClamAV info: DEBUG: sig(test_ndb_partsig) id=178 has 2 parts, this is part 1: [6f6f6f]
LibClamAV info: DEBUG: add part 2 of the sig: [6b6b6b] with mindist=4, maxdist=6
LibClamAV info: DEBUG: sig(test_ndb_partsig) id=178 has 2 parts, this is part 2: [6b6b6b]
LibClamAV info: DEBUG: in cli_ac_initdata, init offmatrix for part sigs
LibClamAV info: DEBUG: in cli_ac_initdata, init offmatrix for part sigs
LibClamAV info: DEBUG: in cli_ac_initdata, init offmatrix for part sigs
LibClamAV info: DEBUG: scan in bm_offmode=0 mode
LibClamAV info: DEBUG: ac scan
LibClamAV info: DEBUG: in cli_scanbuff and doing partial sig scan with acmode=3
LibClamAV info: DEBUG: in cli_scanbuff and doing partial sig scan with acmode=3
test.txt: test_ndb_partsig.UNOFFICIAL FOUND

----------- SCAN SUMMARY -----------
Known viruses: 1329150
Engine version: devel-a6558b5
Scanned directories: 0
Scanned files: 1
Infected files: 1
Data scanned: 0.00 MB
Data read: 0.00 MB (ratio 0.00:1)
Time: 12.933 sec (0 m 12 s)
```

Engine initialiazation and load signatures

```
scanmanager
 cl_load
  cli_load
   cli_loadndb
    cli_initroots
     cli_ac_init
      filter_init
     cli_bm_init
    cli_parse_add
     // change from "6f6f6f{4-6}6b6b6b" to
     // sub-sig 1: "6f6f6f"
     // mindist=4, maxdist=6
     //sub-sig 2: "6b6b6b"
     if((wild = strchr(hexsig, '{')))
      if(sscanf(wild, "%c%u%c", &l, &range, &r) == 3 && l == '{' && r == '}' && range > 0 &&
range < 128) // no for this case
       else
       // we have one more partial sigs
       root->ac_partsigs++;
       // count the parts, '{' or '*' will be treated as a part splitter
       for(i = 0; i < hexlen; i++) if(hexsig[i] == '{' || hexsig[i] == '*') parts++;
       if(parts) parts++;

       start = pt = hexcpy;
       // dealing with each part – adding each part as sub sig into ac tire
       for(i = 1; i <= parts; i++)
         // if format is "string1{a-b}string2"
         for(j = 0; j < strlen(start); j++) if(start[j] == '{') pt = start + j; break;
         *pt++ = 0; // mark the end of a sub sig
         // add the sub sig with info like:
         // start – start of a sub sig
         // root->ac_partsigs – mother sig id
         // parts – number of sub sigs in a sig
         // l – current part index
         // also for the first sub sig, there will be no mindist and maxdist as these two var will be
    reassigned when we get the info from regex {a-b}
         cli_ac_addsig(root, virname, start, root->ac_partsigs, parts, i, rtype, type,
mindist, maxdist, offset, lsigid, options)
         // after adding one part, trying to get the mindist and maxdist out of sig
         if((n = cli_strtok(pt, 0, "-"))) mindist = atoi(n)
         if((n = cli_strtok(pt, 1, "-"))) maxdist = atoi(n)
```

```
// change from "6f6f6f*6b6b6b" to
// sub-sig 1: "6f6f6f”
//sub-sig 2: “6b6b6b"
if((wild = strchr(hexsig, '*')))
  root->ac_partsigs++;
  // count the parts, ‘*’ will be treated as a part splitter
  for(i = 0; i < hexlen; i++) if(hexsig[i] == '*') parts++;
  if(parts) parts++;
  for(i = 1; i <= parts; i++)
    pt = cli_strtok(hexsig, i - 1, "*")
    // add the sub sig
    cli_ac_addsig(root, virname, pt, root->ac_partsigs, parts, i, rtype, type, 0, 0,
offset, lsigid, options)
```

the loading:

this signature “test_ndb_regex:0:3,5:6f6f6f{4-6}6b6b6b” has regular expression with floating range involved, so should be loaded into AC scan sturcture.

Meanwhile, if the signature doesn’t specific a target type, it should be loaded to root[0](generic).

load for ndb

```
#define NDB_TOKENS 6 // NDB have 6 fields
cli_loadndb
    cli_initroots
        for(i = 0; i < CLI_MTARGETS; i++) {
            if(cli_mtargets[i].ac_only || engine->ac_only) root->ac_only = 1;
            cli_ac_init // allocate memory for
                        // root->ac_root and root->ac_root->trans
                        // config and init filter filter_init, set all bits to 1:
                        // memset(m->B, ~0, sizeof(m->B));
                        // memset(m->end, ~0, sizeof(m->end));
            if(!root->ac_only) cli_bm_init // size = HASH(255, 255, 255) + 1;
              // allocate memory for root->bm_shift
              // root->bm_shift[i] = BM_MIN_LENGTH - BM_BLOCK_SIZE + 1;
        engine->root[1]->bm_offmode = 1; /* BM offset mode for PE files */
    target = (unsigned short) atoi(pt); // target is defined in each ndb record
```

root = engine->root[target];
cli_parse_add // add the pattern finally

add pattern: select algo – AC or BM

cli_parse_add
// see section **Engine initialiazation and load signatures**

add signature(pre processing for regular expression) - AC

cli_ac_addsig
// get mother-sig id, number of parts, part index, mindist and maxdist
new->sigid = sigid;
new->parts = parts;
new->partno = partno;
new->mindist = mindist;
new->maxdist = maxdist;
new->ch[0] |= CLI_MATCH_IGNORE;
new->ch[1] |= CLI_MATCH_IGNORE;
// dealing case as "[]" – "HEXSIG[x-y]aa or aa[x-y]HEXSIG"
if(strchr(hexsig, '[')) // with "[" – no match for this case
if(strchr(hexsig, '(')) // with "(" –no match for this case

// dealing other case
new->pattern = cli_mpool_hex2ui(root->mempool, hex ? hex : hexsig);
// new->pattern is uint16_t
cli_mpool_hex2ui
cli_realhex2ui // in this function, each byte of the pattern would be extended to uint16_t(low byte for the pattern byte and high byte for the matching type corresponding to the regular expression type)

```
#define CLI_MATCH_WILDCARD  0xff00
#define CLI_MATCH_CHAR      0x0000
#define CLI_MATCH_IGNORE    0x0100
#define CLI_MATCH_SPECIAL   0x0200
#define CLI_MATCH_NIBBLE_HIGH   0x0300
#define CLI_MATCH_NIBBLE_LOW    0x0400
```

if(hex[i] == '?' && hex[i + 1] == '?') val |= CLI_MATCH_IGNORE;
if(hex[i + 1] == '?') val |= CLI_MATCH_NIBBLE_HIGH;
if(hex[i] == '?') val |= CLI_MATCH_NIBBLE_LOW;
if(hex[i] == '(') val |= CLI_MATCH_SPECIAL;
filter_add_acpatt //* prefiltering
// check if there's regex in first letters
if(new->pattern[i] & CLI_MATCH_WILDCARD)

```
cli_caloff //“test_ndb_regex:0:3,5:6f6f6f{4-6}6b6b6b”
    if((pt = strchr(offcpy, ','))) offdata[2] = atoi(pt + 1); // which is 5
    offdata[0] = CLI_OFF_ABSOLUTE;
    *offset_min = offdata[1] = atoi(offcpy);    // which is 3
    *offset_max = *offset_min + offdata[2]; // which is 8
// add the pattern into ac tire
cli_ac_addpatt
```

add pattern to AC tire

```
cli_ac_addpatt
    uint16_t len = MIN(root->ac_maxdepth, pattern->length);
    // root->ac_maxdepth is set via CLI_DEFAULT_AC_MAXDEPTH
    for(i = 0; i < len; i++)
    next = pt->trans[(unsigned char) (pattern->pattern[i] & 0xff)];
      if(!next) // this tran does not yet exist
        next = (struct cli_ac_node *) mpool_calloc(root->mempool, 1, sizeof(struct cli_ac_node)); // allocate
        newtable = mpool_realloc(root->mempool, root->ac_nodetable, root->ac_nodes * sizeof(struct cli_ac_node *));  // allocate a new node table to copy over the old ones and store the new one, copy over is done automatically via mpool_realloc
        root->ac_nodetable = (struct cli_ac_node **) newtable;
        root->ac_nodetable[root->ac_nodes - 1] = next;
        // put into the tire-
        pt->trans[(unsigned char) (pattern->pattern[i] & 0xff)] = next;
      else
        pt = next // next char
    // create new pattern table and copy over
    newtable = mpool_realloc(root->mempool, root->ac_pattable, root->ac_patterns * sizeof(struct cli_ac_patt *));
    root->ac_pattable = (struct cli_ac_patt **) newtable;
    root->ac_pattable[root->ac_patterns - 1] = pattern;
    /*
    ac node would have a list of ac patterns that share the same prefix
    if there is pattern list, need to insert current one into it, sort according to the first 2 latters of the pattern
    also the ac tree only accept a max depth of 3
    */
    // pt is ac node and ph is ac pattern and now pt is pointing at leaf of this pattern in the ac tire
    ph = pt->list; // the list only exists when the last node in the ac tire is shared by
```

```
other patterns
      ph_add_after = ph_prev = NULL;

      while(ph) // no for this case
```

compile the tire to build the data structure for ac scan(build goto/fail/jump table)

```
cl_engine_compile
  cli_loadftm // load supported file format
  cli_ac_buildtrie
    ac_maketrans // compile the ac tire to build goto/fail/jump table

ac_maketrans
/*
three tables are needed: goto/fail/jump
1    goto table is automatically built via trans[] table
2    the size of each trans table is 256 - the size of ASCII table
3    fail and jump table are built in this function
     see 2_flow_normal_sgin_ac(regex)_scan for details
*/
```

Scan

scan logic design

there are 4 scan methods

1. BM
2. AC
3. Hash
4. Bytecode

There are 2 entry points to begin a scan: cli_map_scandesc and cli_magic_scandesc
cli_map_scandesc will scan a file that is mapped to virtual memory already, this method is not yet used except in unit test case.
cli_magic_scandesc however is used for now as the primary entry of a scan and actually in a later stage, the file to be scanned will be mapped to memory also.

Before the actual scan, the type of the file is assumed as CL_TYPE_ANY, and the

actual type of the incoming file would be decided with cli_filetype2 at magic_scandesc

After the filetype is decided, specific scan function dedicated to the file will be called directly. However, for ASCII file, - CL_TYPE_TEXT_ASCII, the scan will only be called with certain config. So for ascii file, cli_scanraw will be called to make the scan.

In raw scan, ASCII type will be assumed as CL_TYPE_ANY again and calling cli_fmap_scandesc to do further scan.

In cli_fmap_scandesc, according to **ftonly**(if configured as scan specific file type only) and **ftype**(the type of the file which will further decide the root to load) to decide the db to load and scan algo to use in match_run:

- Generic db or type specific db
- BM(normal signature mode or offset mode, currently off mode is only enabled for PE type) or AC or Hash scan
- Hash scan will be performed if BM and AC scan return clean
- If hash scan is clean also, then logic code scan/bytecode scan will be performed via calling cli_lsig_eval and further cli_magic_scandesc_type(mormal BM/AC scan), matchicon or cli_bytecode_runlsig(bytecode scan)
- Bytecode scan will be run finally via cli_bytecode_run
- Bytecode scan can also be triggered via cli_pdf and cli_scanpe
- The bytecode scan will be finally done at cli_vm_execute

In matcher_run, a prefiltering(filter_search_ext) is called to reduce the length of actual scan if possible. After that, BM scan firstly and AC scan later is performed to match against the virus db loaded

The ac scan

some prerequisite functions

========== ac_findmatch ==========

see 2_flow_normal_sgin_ac(regex)_scan for details

========== ac_addtype ==========

see 2_flow_normal_sgin_ac(regex)_scan for details

cli_ac_scanbuff

in cli_scanbuff, cli_ac_initdata will be called to init some data structure used for offset calc in cli_scanbuff corresponding to '{4-6}'

cli_ac_initdata

```
struct cli_ac_data *data
data->partsigs = partsigs;
if(partsigs)
    // allocate space of 4bytes*partsigs and inited as 0
    data->offmatrix = (int32_t ***) cli_calloc(partsigs, sizeof(int32_t **));
```

========== the scan ==========

see 2_flow_normal_sgin_ac(regex)_scan for codes/procedure not listed here

```
                /* it's a partial signature, yes for this case */
                if(pt->sigid)
                  // TBD TBD TBD

                /* old type signature, no for this case */
                else
                  // old type sig

                //try next pattern in next_same list

        // try next pattern in leaf node, end of while
//return identified file type if it's a AC_SCAN_FT scan
return (mode & AC_SCAN_FT) ? type : CL_CLEAN;
```

========== partial signature ==========

```
/*
    how to match sig like "string1{a-b}string2", with content like: "string3string1string0string2string4}" – length of string0 is between a and b
    1. match first sub-sig string1
    2. match second sub-sig string2
    3. make sure the distance between matched string1 and string2 is in between a(mindist) and b(maxdist)
*/
if(pt->sigid) /* it's a partial signature, yes for this case */
// offmatrix is a int32_t ***
```

```
//if the space is not yet exist
if(!mdata->offmatrix[pt->sigid - 1])
    /*
        following procedure will allocate memory of a three dimension array of int32_t tye as:
        dimension 1: number of signatures with sub sigs
        dimension 2: parts*4bytes
        dimension 3: parts*(CLI_DEFAULT_AC_TRACKLEN+2)*4bytes
    */
    // allocate memory parts*4bytes
    mdata->offmatrix[pt->sigid - 1] = cli_malloc(pt->parts * sizeof(int32_t *));
    // allocate memory parts*(CLI_DEFAULT_AC_TRACKLEN+2)*4bytes
    mdata->offmatrix[pt->sigid - 1][0] = cli_malloc(pt->parts * (CLI_DEFAULT_AC_TRACKLEN + 2) * sizeof(int32_t));
    // initialized as -1
    memset(mdata->offmatrix[pt->sigid - 1][0], -1, pt->parts * (CLI_DEFAULT_AC_TRACKLEN + 2) * sizeof(int32_t));
    // init first one in the array as 0
    mdata->offmatrix[pt->sigid - 1][0][0] = 0;
    for(j = 1; j < pt->parts; j++)
        // find the array's start of each part sig
        mdata->offmatrix[pt->sigid - 1][j] = mdata->offmatrix[pt->sigid - 1][0] + j * (CLI_DEFAULT_AC_TRACKLEN + 2);
        // init first one in the array as 0
        mdata->offmatrix[pt->sigid - 1][j][0] = 0;

//local reference of offmatrix
offmatrix = mdata->offmatrix[pt->sigid - 1];
found = 0;
// will be called in second round, offmatrix's data should be reassigned in first round
if(pt->partno != 1)
    // checking maxdist and mindist to see if match end of sub sig fits into the mindist and maxdist
    for(j = 1; j <= CLI_DEFAULT_AC_TRACKLEN + 1 && offmatrix[pt->partno - 2][j] != -1; j++)
        found = j;
            if(pt->maxdist)
                if(realoff - offmatrix[pt->partno - 2][j] > pt->maxdist)
                    // realoff - offmatrix[pt->partno - 2][j]=4
                    found = 0; // fail, exceed maxdist
            if(found && pt->mindist)
                if(realoff - offmatrix[pt->partno - 2][j] < pt->mindist)
                    found = 0; // fail, exceed mindist
            if(found) break; // succeed
```

```
if(pt->partno == 2 && found > 1)
    //swap(offmatrix[0][1] , offmatrix[0][found];)
    if(pt->type != CL_TYPE_MSEXE)
        //swap (offmatrix[pt->parts - 1][1] , offmatrix[pt->parts - 1][found])
        //match result is stored at offmatrix[pt->parts - 1][1]

// will be called in first round
if(pt->partno == 1 || (found && (pt->partno != pt->parts)))
    // if too many part sigs in a sig, re-init as 1
    if(offmatrix[pt->partno - 1][0] == CLI_DEFAULT_AC_TRACKLEN + 1)
        offmatrix[pt->partno - 1][0] = 1;
    // init as 1
    offmatrix[pt->partno - 1][0]++;
    // log current sub-sig's match end position in the buffer
    // will be 10 in this case 'NWSTARTooTESTkkkMYOtestTEST'
    offmatrix[pt->partno - 1][offmatrix[pt->partno - 1][0]] = offset + matchend;
    if(pt->partno == 1)
        //log first sub-sig's match end position in the buffer
        // will be 10 in this case 'NWSTARTooTESTkkkMYOtestTEST'
        offmatrix[pt->parts - 1][offmatrix[pt->partno - 1][0]] = realoff;

    // last sub-sig
    else if(found && pt->partno == pt->parts)
            // if pattern is for specific type
            if(pt->type) // no for this case
            // if is logic sig
            else
                if(pt->lsigid[0]) // no for this case
                if(res) // no for this case
                else // file in virname and return
                    *virname = pt->virname;
                    return CL_VIRUS;

    //after first round, will do pt = pt->next_same to try next pattern in same pattern
list, end of while(pt)
    //also will do patt = patt->next to try next pattern in leaf node, end of while(patt)
```