Clamav funcation call flow

ac scan with regex 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

```
STARTooo TESTkkkMYOtestTEST
```

ndb test

create ndb signature for test.txt

format

MalwareName:TargetType:Offset:HexSignature

where TargetType is one of the following numbers specifying the type of the target file:

0: Any file

1: Portable Executable

2: OLE2 component (eg: VBA script)

3: HTML (normalized)

4: Mail File

5: Graphics

6: ELF

7: ASCII text file (normalized)

And Offset is an asterisk or a decimal number n possibly combined with a special modifier:

- * = any
- n = absolute offset
- EOF-n = end of file minus n bytes

Signatures for PE and ELF files additionally support:

- EP+n = entry point plus n bytes (EP+0 for EP)
- EP-n = entry point minus n bytes
- Sx+n = start of section x's (counted from 0) data plus n bytes
- Sx-n = start of section x's data minus n bytes
- SL+n = start of last section plus n bytes
- SL-n = start of last section minus n bytes

All the above offsets except * can be turned into **floating offsets** and represented as Offset,MaxShift where MaxShift is an unsigned integer. A floating offset will match every offset between Offset and Offset+MaxShift, eg. 10,5 will match all offsets from 10 to 15 and EP+n,y will match all offsets from EP+n to EP+n+y. Versions of ClamAV older than 0.91 will silently ignore the MaxShift extension and only use Offset.

HexSignature

Wildcards

ClamAV supports the following extensions inside hex signatures:

- ??

Match any byte.

- a?

Match a high nibble (the four high bits). **IMPORTANT NOTE:** The nibble matching is only available in libclamav with the functionality level 17 and higher therefore please only use it with .ndb signatures followed by ":17" (MinEngineFunctionalityLevel, see 2.3.4).

- ?a

Match a low nibble (the four low bits).
• `*`
Match any number of bytes.
• `{n}`
Match n bytes.
• `{-n}`
Match n or less bytes.
• `{n-}`
Match n or more bytes.
• `(aa|bb|cc|..)`
Match aa or bb or cc..
• `HEXSIG[x-y]aa` or `aa[x-y]HEXSIG`
Match aa anchored to a hex-signature, see `https://wwws.clamav.net/bugzilla/show_bug.cgi?id=776` for a discussion and examples.
The range signatures `*` and `{}` virtually separate a hex-signature into two parts, eg. `aabbcc*bbaacc` is treated as two sub-signatures `aabbcc` and `bbaacc` with any number of bytes between them. It's a requirement that each sub-signature includes a block of two static characters somewhere in its body.

```
user@ubuntu:~/clamav$ sigtool --hex-dump
ooo*kkk
6f6f6f2a6b6b6b
```

File test.ndb

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

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

Above is a regex based signature with offset info, so it will be loaded into ac pattern structure and will be used in ac scan mode.
The virus record will match start offset between 3 and 5 with pattern as "ooo{4}kkk"(i.e.: any file with sub string "ooo'any 4 bytes'kkk" with the sub string's start at any position between absolute offset 3 and 5 will be identified as virus file)

run

```
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: scan in bm_offmode=0 mode
LibClamAV info: DEBUG: ac scan
test.txt: test_ndb_regex.UNOFFICIAL FOUND
LibClamAV info: DEBUG: scan in bm_offmode=0 mode
LibClamAV info: DEBUG: ac scan
test1.txt: test_ndb_regex.UNOFFICIAL FOUND

----------- SCAN SUMMARY -----------
Known viruses: 1329150
Engine version: devel-a6558b5
Scanned directories: 0
Scanned files: 2
Infected files: 2
Data scanned: 0.00 MB
Data read: 0.00 MB (ratio 0.00:1)
Time: 5.562 sec (0 m 5 s)
user@ubuntu:~/clamav$
```

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}6b6b6b" to "6f6f6f????????6b6b6b"
          if((wild = strchr(hexsig, '{')))
            if(sscanf(wild, "%c%u%c", &l, &range, &r) == 3 && l == '{' && r == '}' && range > 0 &&
range < 128)
                hexcpy = cli_calloc(hexlen + 2 * range, sizeof(char));
                strncpy(hexcpy, hexsig, wild - hexsig);
                strcat(hexcpy, "??");
                wild = strchr(wild, '}')
                strcat(hexcpy, ++wild);
                //call again
                cli_parse_add(root, virname, hexcpy, rtype, type, offset, target, lsigid, options);
                    if(root->ac_only || type || lsigid || strpbrk(hexsig, "?([") || (root->bm_offmode &&
(!strcmp(offset, "*") || strchr(offset, ','))) || strstr(offset, "VI") || strchr(offset, '$'))
          cli_ac_addsig
          cli_ac_addpatt
        cli_ac_addpatt //*
        filter_add_acpatt
        cli_caloff //*
```

the loading:
this signature "test_ndb_regex:0:3,5:6f6f6f{4}6b6b6b" has regular expression 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).

During the db loading process, filter_add_acpatt would be called to calculate prefiltering(using shift or FSM) data of the signatures which will speed up following bm scan a little bit.

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
    if (hexsig[0] == '$') // macro
        cli_ac_addpatt

    if((wild = strchr(hexsig, '{'))) // regular expression
        if(sscanf(wild, "%c%u%c", &l, &range, &r) == 3 && l == '{' && r == '}' &&
range > 0 && range < 128) // dealing case as "{a,b}"
          // change from "6f6f6f{4}6b6b6b" to "6f6f6f????????6b6b6b"
          hexcpy = cli_calloc(hexlen + 2 * range, sizeof(char));
```

```
         strncpy(hexcpy, hexsig, wild - hexsig);
         strcat(hexcpy, "??");
         wild = strchr(wild, '}')
         strcat(hexcpy, ++wild);
         //call again
         cli_parse_add(root, virname, hexcpy, rtype, type, offset, target, lsigid,
options);
        else // dealing case as “string{a,b}string{c,d}” - partial sigs
         root->ac_partsigs++;
         // find all the partial sigs
         for(i = 0; i < hexlen; i++)
            // each hex string besides “{}” or {*} will be split into two partial sigs
            if(hexsig[i] == '{' || hexsig[i] == '*')    parts++;
         // adding each sig into ac tire
         start = pt = hexcpy;
         for(i = 1; i <= parts; i++)
            for(j = 0; j < strlen(start); j++)
                if(start[j] == '{') asterisk = 0; // has not asterisk
                // dealing case as “string{a,b}string*string{c,d}” - partial sigs
                if(start[j] == '*') asterisk = 1; // has asterisk
           ret = cli_ac_addsig(root, virname, start, root->ac_partsigs, parts, i, rtype,
type, mindist, maxdist, offset, lsigid, options)

    // each hex string besides “{}” or {*} will be split into two partial sigs
    if(strchr(hexsig, '*'))
        root->ac_partsigs++;
        for(i = 0; i < hexlen; i++)    if(hexsig[i] == '*')       parts++;
        for(i = 1; i <= parts; i++)
            pt = cli_strtok(hexsig, i - 1, "*")
            ret = cli_ac_addsig(root, virname, pt, root->ac_partsigs, parts, i, rtype,
type, 0, 0, offset, lsigid, options)

    if(root->ac_only || type || lsigid || strpbrk(hexsig, "?([") || (root->bm_offmode
&& (!strcmp(offset, "*") || strchr(offset, ','))) || strstr(offset, "VI") || strchr(offset,
'$')) // cases that also applies ac algo
    // ac_only
    // targeting specific file type instead of generic
    // PE’s bm offset mode with offset defined in signature
    // have VI(version information) offset
    // enters here with ‘?’
            cli_ac_addsig

    if(the rest case) //numbers only
        cli_bm_addpatt
```

add signature(pre processing for regular expression) - AC

cli_ac_addsig

```
	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 "[" – [] means a range, special case
		for(i = 0; i < 2; i++)
			pt = strchr(hex, '[')
			pt2 = strchr(pt, ']')
			sscanf(pt, "%u-%u", &n1, &n2) // AC_CH_MAXDIST=3
			if(strlen(hex) == 2)
				dec = cli_hex2ui(hex); // case "aa[x-y]HEXSIG"
				new->ch[i] = *dec;
				new->ch_mindist[i] = n1;
				new->ch_maxdist[i] = n2;
			if(strlen(pt2) == 2)
				dec = cli_hex2ui(pt2); // case "HEXSIG[x-y]aa"
				new->ch[i] = *dec;
				new->ch_mindist[i] = n1;
				new->ch_maxdist[i] = n2;
// special types
```

```
#define AC_SPECIAL_ALT_CHAR 1
#define AC_SPECIAL_ALT_STR  2
#define AC_SPECIAL_LINE_MARKER  3
#define AC_SPECIAL_BOUNDARY 4

#define AC_BOUNDARY_LEFT         1
#define AC_BOUNDARY_LEFT_NEGATIVE    2
#define AC_BOUNDARY_RIGHT        4
#define AC_BOUNDARY_RIGHT_NEGATIVE   8
#define AC_LINE_MARKER_LEFT      16
#define AC_LINE_MARKER_LEFT_NEGATIVE     32
#define AC_LINE_MARKER_RIGHT         64
#define AC_LINE_MARKER_RIGHT_NEGATIVE    128
```

```
	// dealing case as "()" – "(aa|bb|cc|..) or ! (aa|bb|cc|..) or (B) or (L)"
	if(strchr(hexsig, '(')) // with "(" – () means or, special case
		start = pt = hexcpy;
		while((pt = strchr(start, '('))) // for each "()"
			/* struct cli_ac_special {
				unsigned char *str;
				struct cli_ac_special *next;
				uint16_t len, num;
				uint8_t type, negative;
			}; */
			newspecial = (struct cli_ac_special *) mpool_calloc(root->mempool, 1,
sizeof(struct cli_ac_special));
			if(pt >= hexcpy + 2) if(pt[-2] == '!') // case "! (aa|bb|cc|..)"
```

```
                newspecial->negative=1; // case “(aa|bb|cc|..)”
                                         // newspecial->negative = 0
            start = strchr(pt, ')')
            if(!strcmp(pt, "B")) // case “(B)”
                if(!*start)
                    new->boundary |= AC_BOUNDARY_RIGHT;
                    if(newspecial->negative)
                        new->boundary |= AC_BOUNDARY_RIGHT_NEGATIVE;
                if(pt - 1 == hexcpy)
                    new->boundary |= AC_BOUNDARY_LEFT;
                    if(newspecial->negative)
                        new->boundary |= AC_BOUNDARY_LEFT_NEGATIVE;
            if(!strcmp(pt, "L")) // case “(L)”
                if(!*start)
                    new->boundary |= AC_LINE_MARKER_RIGHT;
                    if(newspecial->negative)
                        new->boundary |= AC_LINE_MARKER_RIGHT_NEGATIVE;
                if(pt - 1 == hexcpy)
                    new->boundary |= AC_LINE_MARKER_LEFT;
                    if(newspecial->negative)
                        new->boundary |= AC_LINE_MARKER_LEFT_NEGATIVE;

        // create new special table with old one copied over
        new->special++;
        newtable = (struct cli_ac_special **) mpool_realloc(root->mempool,
new->special_table, new->special * sizeof(struct cli_ac_special *));
        newtable[new->special - 1] = newspecial;
        new->special_table = newtable;

        if(!strcmp(pt, "B"))    newspecial->type = AC_SPECIAL_BOUNDARY;
        if(!strcmp(pt, "L"))    newspecial->type = AC_SPECIAL_LINE_MARKER;
        else // case “(xx|yy|zz) or (a|b|c)”
            newspecial->num = 1;
            for(i = 0; i < strlen(pt); i++)
                if(pt[i] == '|') newspecial->num++;
            // case “(a|b|c)”
            if(3 * newspecial->num - 1 == (uint16_t) strlen(pt))
                newspecial->type = AC_SPECIAL_ALT_CHAR;
                newspecial->str = (unsigned char *) mpool_malloc(root->mempool,
newspecial->num);
            // case “(xx|yy|zz)
            else    newspecial->type = AC_SPECIAL_ALT_STR;

        for(i = 0; i < newspecial->num; i++)
```

```
                if(newspecial->num == 1) // case of only 1 "|"
                    c = (char *) cli_mpool_hex2str(root->mempool, pt);
                else // case multiple "|"
                    (h = cli_strtok(pt, i, "|")
                    c = (char *) cli_mpool_hex2str(root->mempool, h);\
                // alternative chars stored in array and alternative strings stored in chain
                if(newspecial->type == AC_SPECIAL_ALT_CHAR)
                    newspecial->str[i] = *c; // set the char
                else // string case
                    if(i)
                        specialpt = newspecial;
                        // insert the string into chain of alterative
                        while(specialpt->next)
                            specialpt = specialpt->next;
                        specialpt->next = (struct cli_ac_special *)
mpool_calloc(root->mempool, 1, sizeof(struct cli_ac_special));
                        specialpt->next->str = (unsigned char *) c;
                    else newspecial->str = (unsigned char *) c;
                // sort the char array
                if(newspecial->num>1 && newspecial->type == AC_SPECIAL_ALT_CHAR)
                 cli_qsort(newspecial->str, newspecial->num, sizeof(unsigned char),
qcompare);

        // 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}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

        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) // if leaf is shared by other patterns which is highly possible as only first 3 bytes of the signature is used to build the ac tire, then try to insert it to the
```

```
shared pattern list, also of the pattern or subpattern are same, should also add into a structure called pattern->next_same
    // compare partno???

    if(!ph_add_after && ph->partno <= pattern->partno && (!ph->next || ph->next->partno > pattern->partno))
        ph_add_after = ph;
    // same pattern length, same prefix length and same first two letters
    // ending in same leaf, need to further confirm if the two pattern are same or similar

    if((ph->length == pattern->length) && (ph->prefix_length == pattern->prefix_length) && (ph->ch[0] == pattern->ch[0]) && (ph->ch[1] == pattern->ch[1]))
        // if the characters part of the two pattern are exact the same, compare other info in the signature

        if(!memcmp(ph->pattern, pattern->pattern, ph->length * sizeof(uint16_t))
    && !memcmp(ph->prefix, pattern->prefix, ph->prefix_length * sizeof(uint16_t)))
            // if no other regex special case, the two sig are exact match
            if(!ph->special && !pattern->special) match = 1
            if(ph->special == pattern->special)
                //compare the special info
                a1 = ph->special_table[i];
                a2 = pattern->special_table[i];
            else match = 0;
            if(match) // sig info is the same
                // insert into next_same(same signature list) and sorting according to partno
                if(pattern->partno < ph->partno)
                    pattern->next_same = ph; // insert into same pattern list
                    if(ph_prev) ph_prev->next = ph->next; // remove ph from the leaf node's pattern list since it is added into same pattern list of current pattern
                    else pt->list = ph->next; // removing from current pattern's list
                else
                    while(ph->next_same && ph->next_same->partno < pattern->partno)
                        ph = ph->next_same;
                    pattern->next_same = ph->next_same;
                    ph->next_same = pattern;

        else
            // try next pattern in the list
            ph_prev = ph;
```

```
                ph = ph->next;

        if(ph_add_after) // insert
            pattern->next = ph_add_after->next;
            ph_add_after->next = pattern;
        else // append in head
            pattern->next = pt->list;
            pt->list = pattern;
```

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

cl_engine_compile

  cli_loadftm

```
in cli_loadftm loading: MPEG video stream,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: MPEG sys stream,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: GZip,CL_TYPE_ANY,CL_TYPE_GZ
in cli_loadftm loading: SCRENC,CL_TYPE_ANY,CL_TYPE_SCRENC
in cli_loadftm loading: PostScript,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: BinHex,CL_TYPE_ANY,CL_TYPE_BINHEX
in cli_loadftm loading: Real Media File,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: BMP,CL_TYPE_ANY,CL_TYPE_GRAPHICS
in cli_loadftm loading: BZip,CL_TYPE_ANY,CL_TYPE_BZ
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Eserv mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: MBox,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Exim mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: GIF,CL_TYPE_ANY,CL_TYPE_GRAPHICS
in cli_loadftm loading: Qmail bounce,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: MP3,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: MS CHM,CL_TYPE_ANY,CL_TYPE_MSCHM
in cli_loadftm loading: MS-EXE/DLL,CL_TYPE_ANY,CL_TYPE_MSEXE
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Ogg Stream,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: ZIP,CL_TYPE_ANY,CL_TYPE_ZIP
in cli_loadftm loading: ZIP,CL_TYPE_ANY,CL_TYPE_ZIP
in cli_loadftm loading: RIFF,CL_TYPE_ANY,CL_TYPE_RIFF
in cli_loadftm loading: RIFX,CL_TYPE_ANY,CL_TYPE_RIFF
in cli_loadftm loading: RAR,CL_TYPE_ANY,CL_TYPE_RAR
in cli_loadftm loading: Raw mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Maildir,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Maildir,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: compress.exed,CL_TYPE_ANY,CL_TYPE_MSSZDD
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: EVS mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Symantec,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: ARJ,CL_TYPE_ANY,CL_TYPE_ARJ
in cli_loadftm loading: UUencoded,CL_TYPE_ANY,CL_TYPE_UUENCODED
in cli_loadftm loading: VPOP3 Mail (UNIX),CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: VPOP3 Mail (DOS),CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: TNEF,CL_TYPE_ANY,CL_TYPE_TNEF
in cli_loadftm loading: ELF,CL_TYPE_ANY,CL_TYPE_ELF
in cli_loadftm loading: PNG,CL_TYPE_ANY,CL_TYPE_GRAPHICS
in cli_loadftm loading: CryptFF,CL_TYPE_ANY,CL_TYPE_CRYPTFF
in cli_loadftm loading: OLE2 container,CL_TYPE_ANY,CL_TYPE_MSOLE2
in cli_loadftm loading: JPEG,CL_TYPE_ANY,CL_TYPE_GRAPHICS
in cli_loadftm loading: MP3,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: PE,CL_TYPE_ANY,CL_TYPE_MSEXE
in cli_loadftm loading: ZIP-SFX,CL_TYPE_ANY,CL_TYPE_ZIPSFX
in cli_loadftm loading: RAR-SFX,CL_TYPE_ANY,CL_TYPE_RARSFX
in cli_loadftm loading: ARJ-SFX,CL_TYPE_ANY,CL_TYPE_ARJSFX
in cli_loadftm loading: ARJ-SFX,CL_TYPE_ANY,CL_TYPE_ARJSFX
in cli_loadftm loading: ARJ-SFX,CL_TYPE_ANY,CL_TYPE_ARJSFX
in cli_loadftm loading: AUTOIT,CL_TYPE_ANY,CL_TYPE_AUTOIT
in cli_loadftm loading: NSIS,CL_TYPE_ANY,CL_TYPE_NULSFT
in cli_loadftm loading: SIP log,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: HTML data,CL_TYPE_ANY,CL_TYPE_HTML
in cli_loadftm loading: RTF,CL_TYPE_ANY,CL_TYPE_RTF
in cli_loadftm loading: TAR-POSIX,CL_TYPE_ANY,CL_TYPE_POSIX_TAR
in cli_loadftm loading: mirc ini,CL_TYPE_ANY,CL_TYPE_SCRIPT
in cli_loadftm loading: Mail file,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail file,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail file,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail file,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mach-O LE,CL_TYPE_ANY,CL_TYPE_MACHO
in cli_loadftm loading: Mach-O LE 64-bit,CL_TYPE_ANY,CL_TYPE_MACHO
in cli_loadftm loading: Mach-O BE,CL_TYPE_ANY,CL_TYPE_MACHO
in cli_loadftm loading: Mach-O BE 64-bit,CL_TYPE_ANY,CL_TYPE_MACHO
in cli_loadftm loading: Universal Binary/Java Bytecode,CL_TYPE_ANY,CL_TYPE_MACHO_UNIBIN
in cli_loadftm loading: 7zip,CL_TYPE_ANY,CL_TYPE_7Z
in cli_loadftm loading: Mail file,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: Mail file,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: CPIO NEWC,CL_TYPE_ANY,CL_TYPE_CPIO_NEWC
in cli_loadftm loading: CPIO CRC,CL_TYPE_ANY,CL_TYPE_CPIO_CRC
in cli_loadftm loading: CPIO ODC,CL_TYPE_ANY,CL_TYPE_CPIO_ODC
in cli_loadftm loading: CPIO OLD BINARY BE,CL_TYPE_ANY,CL_TYPE_CPIO_OLD
in cli_loadftm loading: CPIO OLD BINARY LE,CL_TYPE_ANY,CL_TYPE_CPIO_OLD
in cli_loadftm loading: ISHIELD-MSI,CL_TYPE_ANY,CL_TYPE_ISHIELD_MSI
in cli_loadftm loading: PDF document,CL_TYPE_ANY,CL_TYPE_PDF
in cli_loadftm loading: PDF,CL_TYPE_ANY,CL_TYPE_PDF
in cli_loadftm loading: PDF,CL_TYPE_ANY,CL_TYPE_PDF
in cli_loadftm loading: PDF,CL_TYPE_ANY,CL_TYPE_PDF
in cli_loadftm loading: PDF,CL_TYPE_ANY,CL_TYPE_PDF
in cli_loadftm loading: PDF document,CL_TYPE_ANY,CL_TYPE_PDF
in cli_loadftm loading: PDF document,CL_TYPE_ANY,CL_TYPE_PDF
in cli_loadftm loading: PDF,CL_TYPE_ANY,CL_TYPE_PDF
in cli_loadftm loading: PDF,CL_TYPE_ANY,CL_TYPE_PDF
in cli_loadftm loading: SYM DATFILE,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: PDF image,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: SQLite WAL,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: SQLite WAL,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: SQLite database,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: SQLite journal,CL_TYPE_ANY,CL_TYPE_IGNORED
in cli_loadftm loading: SWF (compressed),CL_TYPE_ANY,CL_TYPE_SWF
in cli_loadftm loading: SWF (uncompressed),CL_TYPE_ANY,CL_TYPE_SWF
in cli_loadftm loading: MS CAB,CL_TYPE_ANY,CL_TYPE_MSCAB
in cli_loadftm loading: CAB-SFX,CL_TYPE_ANY,CL_TYPE_CABSFX
in cli_loadftm loading: ISO9660,CL_TYPE_ANY,CL_TYPE_ISO9660
in cli_loadftm loading: TAR-POSIX-CVE-2012-1419,CL_TYPE_ANY,CL_TYPE_POSIX_TAR
in cli_loadftm loading: SIS,CL_TYPE_ANY,CL_TYPE_SIS
in cli_loadftm loading: Mail file,CL_TYPE_ANY,CL_TYPE_MAIL
in cli_loadftm loading: JPEG2000,CL_TYPE_ANY,CL_TYPE_GRAPHICS
```

  cli_ac_buildtrie
    ac_maketrans

ac_maketrans

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

Case of scanning a text file

scanfile

```
 cl_scandesc_callback

   scan_common
     // in normal case, argument 'map' passed into will be NULL
     //except for cl_scanmap_callback in unit test

     cli_magic_scandesc
     // fmap function defined at libclamav/fmap.c
     if(!(*ctx->fmap = fmap(desc, 0, sb.st_size)))
     // call magic_scandesc with type=CL_TYPE_ANY
     // in cli_magic_scandesc_type, will call magic_scandesc with specific type

       magic_scandesc
         if(type == CL_TYPE_ANY)
             type = cli_filetype2(*ctx->fmap, ctx->engine);
               call cli_filetype
               call cli_texttype
         filetype = cli_ftname(type);
         cache_check // calculate hash for a file and do first hash scan???
         // what is following doing???
         hashed_size = (*ctx->fmap)->len;
```

```
        old_hook_lsig_matches = ctx->hook_lsig_matches;
        ctx->hook_lsig_matches = NULL;
         … …
         ctx->hook_lsig_matches = cli_bitset_init();
         in case CL_TYPE_TEXT_ASCII, will not do cli_scan_structured

         cli_scanraw
           unsigned int acmode = AC_SCAN_VIR
            if(typercg) acmode |= AC_SCAN_FT; // specific value for acmode will be
used in cli_ac_scanbuff

             cli_fmap_scandesc(ctx, type == CL_TYPE_TEXT_ASCII ? 0 : type, 0,
&ftoffset, acmode, NULL, refhash)
```

cli_fmap_scandesc

```
// ftonly is the file type [assed ion, if is not CL_TYPE_ANY(=0), then ftonly is true
// ret=cli_fmap_scandesc(ctx, type == CL_TYPE_TEXT_ASCII ? 0 : type, 0, &ftoffset,
acmode, NULL, refhash), so ftonly is set here called from cli_scandesc
if(!ftonly) groot = ctx->engine->root[0]; /* generic signatures */
if(ftype) // in ascii text case, it is 0 which is converted to before
  //now pick up a root for targets
/* the metrix is:
If ftonly is set then use generic root, if recognized specific file type, then use
corresponding root
ftony is set:
means engine will only scan structured file, hence will not use generic root
ftype is set:
means incoming file is a structured file so should pick a specific root
*/
targetinfo(&info, i, map); // get offset and other info according to decided target
type(root[index])
    if(target == 1)
      einfo = cli_peheader; // PE
    else if(target == 6)
      einfo = cli_elfheader; // ELF
    else if(target == 9)
      einfo = cli_machoheader; // MACHO
    else
      return;
if(!ftonly)
    cli_ac_initdata // init data for groot(generic root)
if(troot) // if use specific root for a file type, not applicable in this case
    cli_ac_initdata
    cli_ac_caloff
    if(troot->bm_offmode) // if in bm offset mode
        cli_bm_initoff
if(!ftonly && hdb) // if it's a specific file type and has hash db loaded, try do hash can
preparation
if(troot)
    matcher_run with troot and ac mode decided via acmode
if(!ftonly)
    matcher_run with groot and ac mode decided via acmode

in matcher_run calls
cli_bm_scanbuff
    // byte by byte scan
    shift=root->bmsift[idx]
    if(shift==0)
        //scan over whole pattern
    else
        i+=shift
cli_ac_scanbuff
```