Clamav funcation call flow

by eqmcc

Clamav funcation call flow 1

The call flow 1

Data structures 1

Test case 5

Engine initialiazation 7

Load signatures 7

Scan 10

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

cl_engine

this is a global data structure controls the behavior of a scan engine

```
struct cl_engine {
    uint32_t refcount; /* reference counter */
    uint32_t sdb;
    uint32_t dboptions;
    uint32_t dbversion[2];
    uint32_t ac_only;
    uint32_t ac_mindepth;
    uint32_t ac_maxdepth;
    char *tmpdir;
    uint32_t keeptmp;

    /* Limits */
    uint64_t maxscansize;  /* during the scanning of archives this size
                    * will never be exceeded
                    */
    uint64_t maxfilesize;  /* compressed files will only be decompressed
                    * and scanned up to this size
                    */
    uint32_t maxreclevel;       /* maximum recursion level for archives */
    uint32_t maxfiles;      /* maximum number of files to be scanned
                    * within a single archive
                    */
    /* This is for structured data detection.  You can set the minimum
     * number of occurences of an CC# or SSN before the system will
     * generate a notification.
     */
    uint32_t min_cc_count;
    uint32_t min_ssn_count;

    /* Roots table */
    struct cli_matcher **root;

    /* hash matcher for standard MD5 sigs */
    struct cli_matcher *hm_hdb;
    /* hash matcher for MD5 sigs for PE sections */
    struct cli_matcher *hm_mdb;
    /* hash matcher for whitelist db */
    struct cli_matcher *hm_fp;


    /* Container metadata */
    struct cli_cdb *cdb;

    /* Phishing .pdb and .wdb databases*/
    struct regex_matcher *whitelist_matcher;
    struct regex_matcher *domainlist_matcher;
    struct phishcheck *phishcheck;

    /* Dynamic configuration */
    struct cli_dconf *dconf;
```

```
    /* Filetype definitions */
    struct cli_ftype *ftypes;

    /* Ignored signatures */
    struct cli_matcher *ignored;

    /* PUA categories (to be included or excluded) */
    char *pua_cats;

    /* Icon reference storage */
    struct icon_matcher *iconcheck;

    /* Negative cache storage */
    struct CACHE *cache;

    /* Database information from .info files */
    struct cli_dbinfo *dbinfo;

    /* Used for memory pools */
    mpool_t *mempool;

    /* crtmgr stuff */
    crtmgr cmgr;

    /* Callback(s) */
    clcb_pre_cache cb_pre_cache;
    clcb_pre_scan cb_pre_scan;
    clcb_post_scan cb_post_scan;
    clcb_sigload cb_sigload;
    void *cb_sigload_ctx;
    clcb_hash cb_hash;
    clcb_meta cb_meta;

    /* Used for bytecode */
    struct cli_all_bc bcs;
    unsigned *hooks[_BC_LAST_HOOK - _BC_START_HOOKS];
    unsigned hooks_cnt[_BC_LAST_HOOK - _BC_START_HOOKS];
    unsigned hook_lsig_ids;
    enum bytecode_security bytecode_security;
    uint32_t bytecode_timeout;
    enum bytecode_mode bytecode_mode;
};
```

cli_matcher

the data structure of BM, AC and HASH scan:

- cli_bm_patt
- cli_ac_patt
- cli_hash_patt

```
struct cli_matcher {
    unsigned int type;

    /* Extended Boyer-Moore */
    uint8_t *bm_shift;
    struct cli_bm_patt **bm_suffix, **bm_pattab;
    uint32_t *soff, soff_len; /* for PE section sigs */
    uint32_t bm_offmode, bm_patterns, bm_reloff_num, bm_absoff_num;

    /* HASH */
    struct cli_hash_patt hm;

    /* Extended Aho-Corasick */
    uint32_t ac_partsigs, ac_nodes, ac_patterns, ac_lsigs;
    struct cli_ac_lsig **ac_lsigtable;
    struct cli_ac_node *ac_root, **ac_nodetable;
    struct cli_ac_patt **ac_pattable;
    struct cli_ac_patt **ac_reloff;
    uint32_t ac_reloff_num, ac_absoff_num;
    uint8_t ac_mindepth, ac_maxdepth;
    struct filter *filter;

    uint16_t maxpatlen;
    uint8_t ac_only;
#ifdef USE_MPOOL
    mpool_t *mempool;
#endif
};
```

cli_bm_patt

```
struct cli_bm_patt {
    unsigned char *pattern, *prefix;
    char *virname;
    uint32_t offdata[4], offset_min, offset_max;
    struct cli_bm_patt *next;
    uint16_t length, prefix_length;
    uint16_t cnt;
    unsigned char pattern0;
    uint32_t boundary, filesize;
};
```

cli_ac_patt

```
struct cli_ac_patt {
    uint16_t *pattern, *prefix, length, prefix_length;
    uint32_t mindist, maxdist;
    uint32_t sigid;
    uint32_t lsigid[3];
    uint16_t ch[2];
    char *virname;
    void *customdata;
    uint16_t ch_mindist[2];
    uint16_t ch_maxdist[2];
    uint16_t parts, partno, special, special_pattern;
    struct cli_ac_special **special_table;
    struct cli_ac_patt *next, *next_same;
    uint16_t rtype, type;
    uint32_t offdata[4], offset_min, offset_max;
    uint32_t boundary;
    uint8_t depth;
};
```

cli_hash_patt

```
struct cli_hash_patt {
    struct cli_htu32 sizehashes[CLI_HASH_AVAIL_TYPES];
};
```

cli_mtarget

```
struct cli_mtarget {
    cli_file_t target;
    const char *name;
    uint8_t idx;    /* idx of matcher */
    uint8_t ac_only;
    uint8_t enable_prefiltering;
};

#define CLI_MTARGETS 11
static const struct cli_mtarget cli_mtargets[CLI_MTARGETS] =  {
    { 0,                    "GENERIC",      0,  0, 1 },
    { CL_TYPE_MSEXE,        "PE",           1,  0, 1 },
    { CL_TYPE_MSOLE2,       "OLE2",         2,  1, 0 },
    { CL_TYPE_HTML,         "HTML",         3,  1, 0 },
    { CL_TYPE_MAIL,         "MAIL",         4,  1, 1 },
    { CL_TYPE_GRAPHICS,     "GRAPHICS",     5,  1, 0 },
    { CL_TYPE_ELF,          "ELF",          6,  1, 0 },
    { CL_TYPE_TEXT_ASCII,   "ASCII",        7,  1, 1 },
    { CL_TYPE_ERROR,        "NOT USED",     8,  1, 0 },
    { CL_TYPE_MACHO,        "MACH-O",       9,  1, 0 },
    { CL_TYPE_PDF,          "PDF",         10,  1, 0 }
};
```

Have BM scan: GENERIC and PE, others are AC only
Have filter: GENERIC, PE, MAIL and ASCII
Have both BM and filter: GENERIC and PE with wildcard and other detailed signature formats specified in cli_parse_add

Test case

File: *test.txt*

testtesttesttestMYOtestTEST

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.

```
user@ubuntu:~/clamav$ sigtool --hex-dump
testtesttesttestMYOtestTEST
746573747465737474657374746573744d594f74657374544553540a
```

File test.ndb

```
test_ndb:0:0:746573747465737474657374746573744d594f74657374544553540a
```

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

Engine initialiazation

scanmanager
    cl_engine_new // init engine structure
        cli_mpool_dconf_init // dynamic configure
        crtmgr_init // certification
    cl_engine_set_str // config vars loaded from command line
    cl_engine_set_num // config vars loaded from command line
    cl_load // load virus database
        phishing_init // init phishing data structure
        cli_bytecode_init // init bytecode data structure
        cli_cache_init // init cache structure
        cli_load // load a virus file
        cli_loaddbdir // load a dir with virus files
    cl_engine_compile // compile the engine
        cli_loadftm   // load file types
        cli_ac_buildtrie // build ac trie
            cli_ac_buildtrie // build stat machine for AC match algo
        cli_build_regex_list
        cli_bytecode_prepare2 // Compile bytecode

Load signatures

Functions of loading virus database
cli_cvdload
cli_dconf_load
cli_loadcbc
cli_loadcdb
cli_loadcrt
cli_loaddb
cli_loadftm
cli_loadhash
cli_loadidb
cli_loadign
cli_loadinfo
cli_loadldb
cli_loadmd
cli_loadmscat
cli_loadndb
cli_loadpdb
cli_loadwdb

the loading:
if it's a normal signature(hex numbers only), then load to BM db only, otherwise, if there's wildcard involved, then should be loaded to AC db.

Meanwhile, if the signature specifics a target type, it should be loaded to a specified format's root, otherwise, it should b be loaded to root[0](generic). Also,only generic and PE target type will be having BM db, so only signature with targe type=0 or 1 will be loaded into BM db if the signature is using number only.

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

cli_bytecodde_load is called in cli_loadcdb to load the db for bytecode scan and in the process cl_engine_complie, and cli_bytecode_prepare2 is called to further initiate the bytecode engine

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, '{'))) // wildcard
        if(sscanf(wild, "%c%u%c", &l, &range, &r) == 3 && l == '{' && r == '}' &&
range > 0 && range < 128) // recursively add
```

cli_parse_add
root->ac_partsigs++;
cli_ac_addsig
if(strchr(hexsig, '*')) // *
cli_ac_addsig
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
cli_ac_addsig
if(the rest case) //numbers only
cli_bm_addpatt

add pattern - BM

cli_bm_addpatt

cli_caloff // calculate offset information before adding
// offdata[0]: type
// offdata[1]: offset value
// offdata[2]: max shift
// offdata[3]: section number

```
#define CLI_OFF_ANY          0xffffffff
#define CLI_OFF_NONE         0xfffffffe
#define CLI_OFF_ABSOLUTE     1
#define CLI_OFF_EOF_MINUS    2
#define CLI_OFF_EP_PLUS      3
#define CLI_OFF_EP_MINUS     4
#define CLI_OFF_SL_PLUS      5
#define CLI_OFF_SX_PLUS      6
#define CLI_OFF_VERSION      7
#define CLI_OFF_MACRO        8
#define CLI_OFF_SE       9
```

for cli_caloff, there would be absolute offset and relative offset, also there will be offset information defined in virus db record or passed in via cli_target_info which defines a offset info for a specific file type, and offset info via cli_target_info will be disabled in bm non offmode and enabled in bm offmode

offset_min and offset_max marks the range of the pattern if there's string like "3,6"

if(!info) /* decode offset string */
// will check offset string loaded from virus db record
// If contains "*|,|EP+|EP-|SL+|EOF-|VI|$", will calc the offset info
// Otherwise, just do "offdata[0] = CLI_OFF_ABSOLUTE"
else calc offset using specific file type's offset info

if(root->filter && !root->bm_offmode)
    filter_add_static // do prefiltering calculation

bellow code will take chance to see if certain hash has not yet defined in other signatures, if it is the case and current position is not the start of a pattern, then load balance the bm_suffix table via making the pattern shorter to the point where bm_suffix is not defined and the part of the pattern before this point are treated as prefix

```
#if BM_MIN_LENGTH == BM_BLOCK_SIZE
        /* try to load balance bm_suffix (at the cost of bm_shift) */
        for(i = 0; i < pattern->length - BM_BLOCK_SIZE + 1; i++) {
                idx = HASH(pt[i], pt[i + 1], pt[i + 2]);
                if(!root->bm_suffix[idx]) {
                        if(i) {
                                pattern->prefix = pattern->pattern;
                                pattern->prefix_length = i;
                                pattern->pattern = &pattern->pattern[i];
                                pattern->length -= i;
                                pt = pattern->pattern;
                        }
                        break;
                }
        }
#endif
```

// set bm shift
root->bm_shift[idx] = MIN(root->bm_shift[idx], BM_MIN_LENGTH - BM_BLOCK_SIZE - i);

// then insert the bm_suffix into hash chain where the item share the same index and the chain is sorted by first letter of the pattern

// inf in offset(PE) mode, will add to a data structure called bm_pattab indexed by a int counting the bumber of the bm patterns
if(root->bm_offmode)
    root->bm_pattab[root->bm_patterns] = pattern;

root->bm_patterns++;

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
```