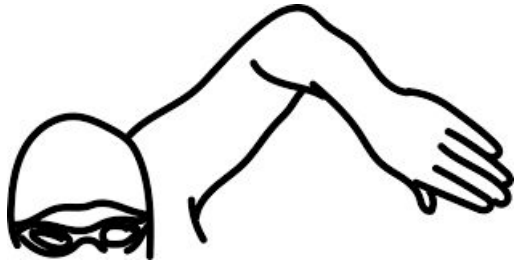

ccrawl

@bdcht, bdcht3@gmail.com

(https://github.com/bdcht/ccrawl)

ccrawl: search engine for C/C++ data structures

Outline

1. Motivation [3min]
2. Requirements [
3. Usage
4. Implementation [3
5. Limitations & further devels

1.Motivation

- Static/Dynamic Binary Code Analysis :
  - how to identify structured data types
  - how to identify constants (macros or enums) or bit masks (constants aggregation)

1.Motivation

- Static/Dynamic Binary Code Analysis :
  - how to identify structured data types
  - how to identify constants (macros or enums) or bit masks (constants aggregation)

```
unused_rs = 0LL;
*((_QWORD *)fs + 196) = 0LL;                    // fs->regs.prev
while ( insn_ptr < insn_end )
{
  pc = (char *)*((_QWORD *)fs + 201);           // fs->pc
  if ( pc >= (char *)context->ra + (*(_QWORD *)&context->flags >> 63) )
    break;
  insn = (unsigned __int8)*insn_ptr++;
  v11 = insn & 0xC0;
  if ( v11 == 64 )                              // DW_FCA_advance_loc
  {
    *((_QWORD *)fs + 201) = &pc[(insn & 0x3F) * *((_QWORD *)fs + 204)];
    continue;
  }
  if ( v11 != 128 )
  {
    if ( v11 == 192 )
    {
      *((_DWORD *)fs + 4 * (insn & 0x3F) + 2) = 0;
      continue;
    }
```

execute_cfa_program (in libgcc.so)

1.Motivation(s)

- static / dynamic Binary Code Analysis:
  - how to identify structured data types
  - how to identify constants (macros or enums) or bit masks (constants aggregation)
- other analysis like forensics dumps (dd, ram, pcap, ...):
  - how to locate structured data based on some characteristics (crypto, …)
  - how to extract/fix these data structures
- other analysis like source code analysis:
  - how to have an overview of the dependency graph between data structures

- Tools:
  - **IDA Pro** + Plugins (TILIB,FLIRT,pdb,...),
  - gdb/gef + python **ctypes**
  - **Volatility, binwalk, hachoir, ...**
  - **kaitai_struct**
  - **protobuf**

2.Requirements

a “search engine” that would allow to “query” a large database of definitions:

- collect all definitions for types/prototypes/constants even if the files set is “incomplete” (missing types, missing includes, …)
- query the database for structured C types with constraints like
  - having a pointer at offset +196 and an int at offset 0
  - having a pointer to char at offset +201, a total size of 244,
  - …
- query for symbols (#define ou enums) based on constants
  - which symbols have value **0x64 ?**
  - which OR-ed sequence of symbols have value 0x42 ?
- output all dependencies between structures/types
- translate results in python (**ctypes**, **amoco**), or other formats (**kaitai**,…)

3.Usage

```
$ ccrawl [-v, --verbose]                increase output verbosity
         [-q, --quiet]                  remove all outputs
         [-c, --configfile <file>]      optional configuration file
         [-l, --local <file>]           local database file
         [-b, --db <url>]               remote database url
         [-g, --tag <name>]             tag used to filter documents of the database
         [<command> [options] [args]]
```

- commands :
  - **collect** : import all definitions of types/consts into the database
  - **match** : search for regular expression in all symbols (types, fields, ...)
  - **find** : search for structured types based on sizes/offsets/values
  - **show** : output the definition in C or some extern format (ctypes, …)
  - info : output informations about a database “document”
  - store : send local database to a remote **mongodb**
  - tags : output the list of all ‘tags’ in the database
  - sources : output the list of source files that have been collected in the database
  - stats : output various statistics about the database

3.Usage: collect

```
$ ccrawl [global options] collect [options] <src>

  options: [-a, --all]          by default, only header files with '*.h' extension are
                                considered, this option forces extraction from all provided
                                files.
           [-t, --types]        extraction is limited to types (typedefs, struct, union, enum)
           [-f, --functions]    extraction is limited to function prototypes
           [-m, --macros]       extraction is limited to macros

           [-s, --strict]       collect in "strict" mode: in this mode, all errors reported by
                                libclang are blocking. It is thus mandatory to provide the
                                complete set of input files and precise clang options such that
                                clang is able to compile successfully the provided <src> files.
           [--clang "<opts>"] pass <opts> string directly to clang as options

           <src> ...            directory name(s) or file name(s) of C source(s) from which
                                selected definitions shall be extracted and collected in the
                                local database.
```

example: **$ cd gcc; ccrawl -l gcc.db --tag “gcc” collect libgcc/ include/**

3.Usage: match

```
$ ccrawl [global options] match <rex>

          <rex>              python (re) regular expression matched against local database
                             documents keys 'id' and 'val'. Documents are filtered with
                             'tag' as well if the --tag global options is used.
```

examples:

$ ccrawl -l gcc.db --tag “gcc” match DW_CFA

```
found cStruct identifer "struct frame_state_reg_info" with matching value
found cFunc identifer "get_DW_CFA_name"
found cMacro identifer "DW_CFA"
found cMacro identifer "DW_CFA_extended"
found cStruct identifer "struct frame_state_reg_info" with matching value
found cMacro identifer "DW_CFA_DUP"
found cEnum identifer "enum dwarf_call_frame_info" with matching value
```

3.Usage: find

```
$ ccrawl [global options] find [-a, --ands <str>]
                               [-o, --ors  <str>]
                               [<find_command> [options] [args]]

          [-a, --ands <str>] filters <str> of the form "key=value" added to current query
                             with operator AND:
                             Equivalent to "Q &= where(key).search(value)".
          [-o, --ors <str>]  same form, but added to current query with operator OR:
                             Equivalent to "Q |= where(key).search(value)".
```

- find_commands :
  - prototype : search for a prototype with argument of a given type
  - constant : search for symbols with given value/mask
  - struct : search for struct/union/class with given type/size constraint on a field/offset

examples: **$ ccrawl -l gcc.db --tag "gcc" find prototype ...**

$ ccrawl -l gcc.db --tag "gcc" find const ...

$ ccrawl -l gcc.db --tag "gcc" find struct ...

3.Usage: find prototype

```
<find_command>:

prototype "<pos>:<type>" ...
          Find prototypes (cls=cFunc) for which constraints of the form
          "<pos>:<type>" matches. Such constraint indicates that
          argument located at <pos> index has C type <type>
          (position index 0 designates the return value of the function).
```

example: **$ ccrawl -l gcc.db --tag “gcc” find prototype “1:struct _Unwind_Context *”**

3.Usage: find constant

```
constant [-m, --mask] <value>
          Find which macro definition or enum field name matches constant <value>.
          Option --mask allows to look for the set of macros or enum symbols
          that equals <value> when OR-ed.
```

examples: **$ ccrawl -l gcc.db --tag “gcc” find constant “64”**

3.Usage: find struct

```
struct "<offset>:<type>" ...
          Find structures (cls=cStruct) satisfying constraints of the form:
          "<offset>:<type>" where offset indicates a byte offset value (or '*')
          and type is a C type name, symbol '?', '*' or a byte size value:
          If <type> is "?", match any type at given offset,
          If <type> is "*", match any pointer type at given offset,
          If <type> is "+<val>", match if sizeof(type)==val at given offset.
          Si "*:+<val>", match struct only if sizeof(struct)==val.
```

examples: **$ ccrawl -l gcc.db --tag “gcc” find struct “+24:struct _Unwind_Context *”**

$ ccrawl -l gcc.db --tag “gcc” find struct “*:0x960”

$ ccrawl -l gcc.db --tag “gcc” find struct “+8:_Unwind_Word”

3.Usage: show

```
$ ccrawl [global options] show [options] <identifier>

  options: [-r, --recursive]     recursively include all required definitions in the output
                                 such that type <identifier> is fully defined.
           [-f, --format <fmt>]  use output format <fmt>. Defaults to C, other formats are
                                 "ctypes", "amoco", "volatility", "kaitaistruct", "protobuf".
```

examples: **$ ccrawl -l gcc.db --tag “gcc” show ‘struct _Unwind_Context’**

$ ccrawl -l gcc.db --tag “gcc” show -r ‘_Unwind_FrameState’

$ ccrawl -l gcc.db --tag “gcc” show -r -f ctypes ‘struct dwarf_cie’

4.Implementation

- **Python** (2.7 + 3.5)
- commands & sub-commands: **Click**
- locale database (json): **TinyDB**
- remote (efficient) database : **MongoDB** (or **CouchDB**)
- C/C++ sources parser: **libclang**
- configure: **traitlets**

Clang options:

```
-M  -MG   -MF/tmp/ccrawl-xxx -ferror-limit=0 -fparse-all-comments
[-x c++ -std=c++11]
```

4.Implementation. internals

- local bases in JSON:
  - cls: ‘cStruct’, ‘cUnion’, ‘cEnum’, ‘cTypedef’, ‘cMacro’, ‘cFunc’
  - id: symbol (name) of collected objet
  - src: source filename
  - tag: arbitrary string used for filtering (defaults to timestamp)
  - val: actual definition of the collected objet (a list/dict/int/str)

```
{'cls': '<class>',
 'id' : '<identifier>',
 'src': '<path>',
 'tag': '<name>',
 'val': <object>}
```

example:

```
struct _mystruct {
    myinteger   I;                  /* comment for field I */
    signed int   tab[MYMACRO(8)];        // modern comment for tab
    unsigned char p[MYCONST];
    short int *s;
    struct _mystruct* next;
    foo func;
    struct _bar {
        enum X x;
    } bar[2];
};
```

```
{'cls': 'cStruct',
   'id': 'struct _mystruct',
   'src': 'samples/header.h',
   'tag': '1549141214.805588',
   'val': [['myinteger', 'I', 'comment for field I'],
           ['int [12]', 'tab', 'modern comment for tab'],
           ['unsigned char [16]', 'p', None],
           ['short *', 's', None],
           ['struct _mystruct *', 'next', None],
           ['foo', 'func', None],
           ['struct _bar [2]', 'bar', None]
          ]}
```

5.Limitations and further developments

- add more extern output formats (kaitai, protobuf, ...)
- add a web frontend
- output coverage informations
- add option to limit recursion depth and replace pointers by void*
- add support for parsing a function body and compute some “signature”
- ...