ClueHunter Documentation

Release 0.1a1

Ke Yang

January 05, 2017

Contents

ClueHunter is an auxiliary tool for crash point reverse data flow analysis. It generate data flow graph according to the gdb debug log(C program source code level). It receives manually specified sink variables that cause the last line crash and perform interprocedural analysis on the log trace. For obtaining the auto-debug trace, the tool *robot_dbg.exp* in ClueHunter requires the program under debug to be compiled with profiled code information (gcc **-g -save-temps** operation). During the current develop stage, only command line program is supported.

CHAPTER 1

Target Problem: Where does the bad data come from? And how?

A common question for program debugging is "Where is the bad data come from? And how?" In this case we need to analyse reversely from the crash or wrongly executed statement, trace the relative calculation, then infer and locate the wrong code logic. Slicing is an effective technique to prune away irrelative calculations. It help analyst to focus on the error relative code snipet. Visulizing the complex code logic as a colorful property graph is also preferable. In fact, we always use IDE or gdb to view the code snipets and call stacks to keep in mind the whole picture of the data flow, but few of them provide the trace visualization service. ClueHunter is then designed to relax our mind and help us to infer the executed data transform logic.

Contents:

1.1 Quick Start

1.1.1 Install

ClueHunter depends on graphviz to generate the picture from the dot file. Others dependencies are installed in Ubuntu-14.04 system by default.

List of dependencies:

- gcc >=4.8
- gdb >=7.7
- expect 1.1
- python 2.7
- graphviz>=2.36

For Ubuntu:

```
sudo apt-get install git
sudo apt-get instsall graphviz
git clone https://github.com/yangke/cluehunter.git
```

That's done.

1.1.2 Start Funny

1. Compile the Program Under Analysis

First compile your C project with gcc **-g -save-temps** option. In most cases you can specify this in the configure procedure like this:

```
$./configure CFLAGS="-g -save-temps" CXXFLAGS="-g -save-temps" --prefix=$YOUR_INSTALL_PATH
```

Otherwise you may have to change the *Makefile*.

2. Provide the Execution Command for Auto-debug

   Then modify line 15 in `cluehunter/robot_dbg.exp` to fit with your debug scenario. Here is an example for executable program `swf2xml` test in swfmill-0.3.3.

```
15:spawn gdb -q --args swfmill swf2xml exploit_it_to_crash
```

   The input file `exploit_it_to_crash` will cause the crash of `swf2xml`.

3. Run the Modified Script

   Then use `robot_dbg.exp` to debug your program automatically. It executes gdb `next` command when meeting lines which contains library or system call site, other cases it executes `step` command of gdb. If `robot_dbg.exp` mistakenly steps into a call with no source code, it will then use `finish` command to execute through it to jump out. Copy the `robot_dbg.exp` into the directory of binary executable program: `swf2xml` and the exploit input: `exploit_it_to_crash`. This will make the former command valid(`spawn gdb -q --args swfmill swf2xml exploit_it_to_crash`).

```
swfmill-0.3.3_install_bin_path$ls
        ... exploit_it_to_crash ... robot_dbg.exp ... swf2xml ...
swfmill-0.3.3_install_bin_path$./robot_dbg.exp
...
(gdb) q
A debugging session is active.

        Inferior 1 [process 30695] will be killed.

Quit anyway? (y or n) ^Cswfmill-0.3.3_install_bin_path$ls
        ... exploit_it_to_crash ... gdb.txt ... robot_dbg.exp ... swf2xml ...
```

4. **Use cluehunter.py to analyse the gdb.txt** Every thing come handy, we got the debug trace `gdb.txt` besides them. Then we can use `cluehunter.py` to analyze this trace.

```
python cluehunter.py -t path_to/gdb.txt\
        -vs length -ps N -o . -n telescope -l 1
```

   This command will use the test trace located at gdb.txt to perform reverse data flow analysis for variable `length` from the last parsed line(as the default). To specify the line number, you can use the option `-i {line number in trace.txt}` (see bellow for detail). The sensitive crash data `length` itself are marked as tainted. The access pattern of `length`, `'N'`, means direct access. Another mark `'*'` means we need to dereference this pointer to access sensitive sink data we cared about. Note that the `*` must be quoted with *“”* or *‘’* in command line. This command will cause ClueHunter output `telescope.dot` and use graphviz to generate `telescope.svg` beside it. `-vs`, `-ps` and `-t` are three mandatory options which specify the names of sink variables, patterns and the trace to analysis respectively. `-o` option specified the output directory. `-l` specified the parsed trace redundancy level. `0` means only remove the line redundancy in same function and `1` means remove both the inner function and inter-function reduandancy.

   If you want to analyze variables on specific trace line, you may need `-i` option. For example: `-i -1` specifies the last line in `trace.txt`, and `-i -2` specifies the line of last but one. You can also use positive line number. For instance, `-i 100` means the 100 line in the `trace.txt`. **Note that the lines we talk here are the lines in the parsed middle file: ‘‘trace.txt‘‘**. The last line(`-i -1`) in `trace.txt` corresponds to the last none empty line above the error information `Program receive ...` in `gdb.txt`.

1.1.3 Macro Expansion

ClueHunter can analyze the function call caused by macros by expanding them. It references the preprocessed `*.i` files generated by **-save-temps** option of gcc to make a macro expansion. To use this function, you have to specify the path of the compiled C project corresponding to the log trace under analysis. And make sure the under analysis program is compiled with **-save-temps**. This function is not available by default, please use `-m` to specify the compiled C project path.

1.1.4 Executable Test Command

Here is an executable test command which analyzes the trace `gdb-swfmill-0.3.3.txt` provided in test module.

```
python cluehunter.py -t test/gdb_logs/swfmill-0.3.3/gdb-swfmill-0.3.3.txt\
        -vs length -ps 'N' -o . -n telescope -l 1 -m test/gdb_logs/swfmill-0.3.3/swfmill-0.3.3
```

1.1.5 Complete Usage

```
usage: cluehunter.py [-h] -ps PATTERNS [PATTERNS ...] -vs VARIABLES
                [VARIABLES ...] [-l LEVEL] -t TRACE [-o OUTPUT_PATH]
                [-m C_PROJECT_DIR] [-n NAME] [-d | -v | -q]

optional arguments:
  -h, --help            show this help message and exit
  -l LEVEL, --level LEVEL
                        Redundancy level of the parsing. 0 means just remove
                        inline or innner function redundancy; 1 means remove
                        both of the inline and interprocedural reduandancy.
  -i INDEX, --index INDEX
                        The start trace line for tracking. Default value is -1
                        which means start from the last line. Positive integer
                        means the {line number}-1 in the parsed result
                        cluhunter/test/trace.txt. Negative integer means the
                        last but what line of the cluhunter/test/trace.txt. 0
                        is useless, but it still can be regarded as the
                        first line.
  -t TRACE, --trace TRACE
                        The file path of gdb trace log, for example,
                        ./gdb.txt. This log should be generated by
                        robot_dbg.exp.
  -o OUTPUT_PATH, --output-directory OUTPUT_PATH
                        The output directory in which .dot and .png files will
                        be dumped in this path.
  -m C_PROJECT_DIR, --c-project-dir C_PROJECT_DIR
                        The C project directory with the .i files maked by gcc
                        '-save-temps' option. Usually the we add this flags
                        during configure: ./configure CFLAGS='-g -save-temps'.
  -n NAME, --name NAME  The prefix name of the generated .dot and .png files.
  -d, --debug           Enable debug output.
  -v, --verbose         Increase verbosity.
  -q, --quiet           Be quiet during processing.

sinks:
  -ps PATTERNS [PATTERNS ...], --patterns PATTERNS [PATTERNS ...]
                        Specify the access pattern list of the sink
                        identifiers. Patterns must be "*" or "N" separated
```

```
                        with blanks. "N" means direct access, "*" means this
                        is a pointer of the cared data.
  -vs VARIABLES [VARIABLES ...], --variables VARIABLES [VARIABLES ...]
                        Specify the identifier name of the sink variables.
                        Example:"father->baby.toy"
```

1.2 Output Examples

1.2.1 CVE-2008-1686 speex null pointer dereference

The following table shows the summary of CVE-2008-1686.

Table 1.1: Table 1 CVE-2008-1686 summary

CVE ID	CWE ID	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained	Access Level
CVE-2008-1686	189	Exec Code	2008-04-08	2011-05-19	9.3	Admin	Remote
Array index vulnerability in Speex 1.1.12 and earlier, as used in libfishsound 0.9.0 and earlier, including Illiminable DirectShow Filters and Annodex Plugins for Firefox, xine-lib before 1.1.12, and many other products, allows remote attackers to execute arbitrary code via a header structure containing a negative offset, which is used to dereference a function pointer.							

Construct a crash exploit and use the robot_dbg.exp to record the source code execution trace. Then let ClueHunter to perform the interprocedural analysis on it. It will output the svg graph file by default.

```
.../cluehunter$python cluehunter.py -ps '*' -vs mode -l 1 \
        -t test/gdb_logs/speex/CVE-2008-1686/speex-1.1.12/speexdec/gdb-speex-1.1.12_speexdec_mode.txt
```

Figure 1 shows the dependencies of variable `mode` which cause the crash.

Fig. 1.1: Figure 1 Data dependency graph of variable `mode` generated by cluehunter

Table 2 shows the meaning of the node and edge shape .

Table 1.2: Table 2 The Meaning of Shape for Node and Edge

elipse node	statement
square node	call info
solid red edge	innner function data flow
dashed green edge	connection of call info and callsite
dashed yellow edge	cross function data flow (mainly caused by argument definition)
dashed orange edge	represent the data flow beteen the callsite's return statement and the call assignment

The the crash is caused by the NULL pointer returned by `speex_lib_get_mode(modeID)` located at line 166 in log trace. ClueHunter provides the inner statement that cause this problem which is a if check that judges the `modeID` as an invalid value:

```
168#719   if (mode < 0 || mode > SPEEX_NB_MODES) return NULL;
```

To continue tracking the modeID, change the start line to the trace index(166) of the call site:

```
166#327    mode = speex_lib_get_mode (modeID);
```

Go into the cluhunter directory, type the following command then we got the answer.

```
.../cluehunter$python cluehunter.py -ps N -vs mode -l 1 -i 166 \
        -t test/gdb_logs/speex/CVE-2008-1686/speex-1.1.12/speexdec/gdb-speex-1.1.12_speexdec_mode.txt
```

(In this case we disabled the macro analysis as we does not specify *-m $BUILD_PATH_OF_SPEEX*.)

Fig. 1.2: Figure 2 Data dependency graph of variable `modeID` generated by cluehunter

For cluehunter the upper ogg page operation in trace line 73 is not an obvious assignment. This is because cluehunter haven't included the ogg library tainting rules. For a backup solution, we can still find the data flow by interleaving the manual effort and cluehunter. Because we can control the start point to be analysed in the execution trace.

```
58#main (argc=2, argv=0xbfffe884) at speexdec.c:583
59#583        data = ogg_sync_buffer(&oy, 200);
60#585        nb_read = fread(data, sizeof(char), 200, fin);
61#586        ogg_sync_wrote(&oy, nb_read);
62#589        while (ogg_sync_pageout(&oy, &og)==1)
63#592           if (stream_init == 0) {
64#593              ogg_stream_init(&os, ogg_page_serialno(&og));
65#594              stream_init = 1;
66#597           ogg_stream_pagein(&os, &og);
67#598           page_granule = ogg_page_granulepos(&og);
68#599           page_nb_packets = ogg_page_packets(&og);
69#600           if (page_granule>0 && frame_size)
70#609              skip_samples = 0;
71#612           last_granule = page_granule;
72#614           packet_no=0;
73#615           while (!eos && ogg_stream_packetout(&os, &op)==1)
74#618              if (packet_count==0)
75#620                 st = process_header(&op, enh_enabled, &frame_size, &rate, &nframes, forceMode,
```