Cross Translational Unit Analysis in Clang Static Analyzer: Prototype and Measurements

Gabor Horvath (xazax¹), Peter Szecsi (ps95¹), Zoltan Gera (gerazor¹)
Daniel Krupp (daniel.krupp²), Zoltan Porkolab (zoltan.porkolab²)

[1] @caesar.elte.hu
[2] @ericsson.com
• Motivation
• Overview of the Cross Translation Unit Analysis architecture
• Evaluation on open source projects
  • Findings
  • Performance
• Design questions
  • How to organize CTU related code
  • What to reanalyze, how to scale
• Future work
Find bugs without running the code

Exploded Graph

```c
void test(int b) {
    int a,c;
    switch (b){
        case 1: a = b / 0; break;
        case 4: c = b - 4;
            a = b / c; break;
    }
}
```
Motivation

- We saw useful CTU results reported by closed source analysis tools
- Can we achieve the same using Clang SA?
High Level Architecture

1st Pass
CTU Build

Global Call Graph

Function Definition Index

2nd Pass
Analyzer

Analysis Results (PLISTS)

Source Code & JSON Compilation Database

AST dumps
Evaluation

- Open source C projects:
  - OpenSSL, Curl, Vim, Memcached, ffmpeg, PostgreSQL, ...
  - Full details at: http://cc.inf.elte.hu
  - Improvements needed for C++ support

- Metrics:
  - Number of new bugs reported
  - Number of lost bug reports
  - Quality of new bug reports
  - Analysis time
  - Peak memory usage (per process)
<table>
<thead>
<tr>
<th>Run ID (project + timestamp)</th>
<th>Total files of project</th>
<th>Files XTU Passed</th>
<th>Files XTU Failed</th>
<th>Time of XTU (sec)</th>
<th>Time of noXTU (sec)</th>
<th>Time of XTU-BUILD (sec)</th>
<th>Max heap usage of NOXTU (B)</th>
<th>Max heap usage of XTU (B)</th>
<th>Analysis Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>curl_2017-03-23_23:50:26</td>
<td>293</td>
<td>293</td>
<td>0</td>
<td>644.54</td>
<td>362.41</td>
<td>8.37</td>
<td>97912954</td>
<td>24380678</td>
<td>No XTU Coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XTU Coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Diff Coverage</td>
</tr>
<tr>
<td>ffmpeg_2017-03-24_00:10:12</td>
<td>1563</td>
<td>1559</td>
<td>4</td>
<td>6260.98</td>
<td>2941.50</td>
<td>57.89</td>
<td>157030632</td>
<td>202068780</td>
<td>No XTU Coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XTU Coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Diff Coverage</td>
</tr>
<tr>
<td>memcached_2017-03-24_03:10:12</td>
<td>35</td>
<td>35</td>
<td>0</td>
<td>372.16</td>
<td>230.57</td>
<td>2.20</td>
<td>72261684</td>
<td>95777660</td>
<td>No XTU Coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XTU Coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Diff Coverage</td>
</tr>
<tr>
<td>nginx_2017-03-24_03:21:37</td>
<td>114</td>
<td>114</td>
<td>0</td>
<td>766.14</td>
<td>336.93</td>
<td>5.28</td>
<td>81624439</td>
<td>161235414</td>
<td>No XTU Coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>XTU Coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Diff Coverage</td>
</tr>
<tr>
<td>File</td>
<td>Lines</td>
<td>Branches</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------</td>
<td>--------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/base64.c</td>
<td>90.7 % 107 / 118</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/conncache.c</td>
<td>100.0 % 139 / 139</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/connect.c</td>
<td>85.2 % 439 / 515</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/content_encoding.c</td>
<td>100.0 % 160 / 160</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/cookie.c</td>
<td>62.7 % 381 / 608</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/curl_addrinfo.c</td>
<td>96.7 % 174 / 180</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/curl_endian.c</td>
<td>100.0 % 32 / 32</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/curl_fnmatch.c</td>
<td>91.0 % 213 / 234</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/curl_gethostname.c</td>
<td>100.0 % 11 / 11</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/curl_memrchr.c</td>
<td>100.0 % 8 / 8</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/curl_ntlm_core.c</td>
<td>98.4 % 122 / 124</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/curl_ntlm_wb.c</td>
<td>100.0 % 196 / 196</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/curl_sasl.c</td>
<td>98.2 % 224 / 228</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/dict.c</td>
<td>100.0 % 127 / 127</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/dotdot.c</td>
<td>84.1 % 53 / 63</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/easy.c</td>
<td>98.0 % 251 / 256</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/escape.c</td>
<td>90.0 % 81 / 90</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/file.c</td>
<td>100.0 % 234 / 234</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/fileinfo.c</td>
<td>100.0 % 11 / 11</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>lib/formdata.c</td>
<td>66.4 % 417 / 628</td>
<td>100.0 % 0 / 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
if(strcmp(lineptr, "#HttpOnly", 10) == 0) {
    lineptr += 10;
    co->httponly = TRUE;
}

if(lineptr[0]=='#') {
    /* don't even try the comments */
    free(co);
    return NULL;
}

/* strip off the possible end-of-line characters */
ptr=strchr(lineptr, '\r');
if(ptr)
    *ptr=0; /* clear it */
ptr=strchr(lineptr, '\n');
if(ptr)
    *ptr=0; /* clear it */
firstptr=strtok_r(lineptr, "\t", &tok_buf); /* tokenize it on the TAB */

/* Now loop through the fields and init the struct we already have allocated */
for(ptr=firstptr, fields=0; ptr && !badcookie; ptr=strtok_r(NULL, "\t", &tok_buf), fields++) {
    switch(fields) {
    case 0:
        if(ptr[0]=='.') /* skip preceding dots */
            ptr++;
        co->domain = strdup(ptr);
        if(!co->domain)
            badcookie = TRUE;
        break;
• **2.4X** average, **2.1X** median, **5X** peak
Bug Path Length of Bug Reports

- The reason for false positives was never the CTU
• True positive example:
  http://cc.inf.elte.hu:8080/#baseline=177&newcheck=178&report=17539

• One Definition Rule violation found
AVInputFormat *iformat = NULL;
AVFormatContext *format_ctx = NULL;
AVCodec *codec;
AVCodecContext *codec_ctx;
AVFrame *frame;
int frame_decoded, ret = 0;
AVPacket pkt;
AVDictionary *opt=NULL;

av_init_packet(&pkt);

av_register_all();

iformat = av_find_input_format("image2");
if ((ret = avformat_open_input(&format_ctx, filename, iformat, NULL)) < 0) {
    assuming the condition is false
    av_log(log_ctx, AV_LOG_ERROR,
        "Failed to open input file '%s'\n", filename);
    return ret;
}

if ((ret = avformat_find_stream_info(format_ctx, NULL)) < 0) {
    assuming the condition is false
    av_log(log_ctx, AV_LOG_ERROR, "Find stream info failed\n");
    return ret;
}

codec_ctx = format_ctx->streams[0]->codec;
codec = avcodec_find_decoder(codec_ctx->codec_id);
if (!codec) {
    av_log(log_ctx, AV_LOG_ERROR, "Failed to find codec\n");
    ret = AVERROR(EINVAL);
    goto end;
}

av_dict_set(&opt, "thread_type", "slice", 0);
if ((ret = avcodec_open2(codec_ctx, codec, &opt)) < 0) {
    av_log(log_ctx, AV_LOG_ERROR, "Failed to open codec\n");
    goto end;
}

if (!(frame = av_frame_alloc())) {
end:
    av_packet_unref(&pkt);
    avcodec_close(codec_ctx);
    avformat_close_input(&format_ctx);
    av_frame_free(&frame);
    Calling 'av_frame_free'

    bugpath in:
    frame.c

    av_dict_free(&opt);
A.cpp
void neg(int *x);
void g(int *x) {
    ...
    neg(NULL);
    ...
}
void h(int *x) {
    ...
    neg(NULL);
    ...
}

B.cpp
void neg(int *x) {
    *x = -(*x);
}

Same bug multiple times?
Evaluation – Analysis time

- **2.5X** average, **2.1X** median, **6.4X** peak
Evaluation - Memory

● 2.3X average, 2.3X median, 5.5X peak
● AST dumps consume disk space temporarily
  • ~40GB for LLVM
Current Implementation

- Artem Dergachev, Aleksei Sidorin, et al.
  - Prototype: both for naive CTU and summary based interprocedural analysis, based on Clang 3.4
- Improved version contributed by Ericsson, only contains the CTU part, ready for review
  - [https://reviews.llvm.org/D30691](https://reviews.llvm.org/D30691)
  - Patch is relatively small, CTU off by default
  - **No changes required to checker implementations**
Order of the Analysis of Functions

A.cpp
void f(int x);
void g(int x) {
    f(x);
}
void h(int x) {
    g(x);
}

B.cpp
void i(int x) {
}
void f(int x) {
    i(x);
}
void f(int *x);
void g(int *x) {
    f(NULL);
}

void f(int *x) {
    *x = -(*x);
}

A.cpp

B.cpp

void f(int *x) {
    *x = -(*x);
}
Incrementality

A.cpp

```c
void f(int *x);
void g(int *x) {
    f(NULL);
}
```

B.cpp

```c
void f(int *x) {
    *x = -(*x);
    if (x == 0)
        return;
}
```

- We need to reanalyze A.cpp too
AST Importer

• Import (merge) one AST into another
• Can import one function/type at a time
• Caches the results to avoid multiple imports
• Used by LLDB
• Not a mature component of Clang yet
Issues with importing source locations from macros

Suboptimal results for C++ projects
  - We concentrated on C projects
  - Fixed C related bugs in the importer

The analysis can find AST Importer bugs
  - Running analysis on the imported AST can trigger asserts
  - Found invariant violations on imported AST that otherwise very challenging to write a test for
Coverage

- Increased for some files
  - Functions evaluated in more contexts
- Decreased for others
  - Analysis budget runs out due to DFS
  - Prune more infeasible paths
  - More issues reported implies stops
- Small overall decrease

```c
void external(int x);
void g(int x) {
    external(x);
    x / 0;
}
```
Coverage

- Increased for some files
  - Functions evaluated in more contexts
- Decreased for others
  - Analysis budget runs out due to DFS
  - Prune more infeasible paths
  - More issues reported implies stops
- Small overall decrease

```c
void external(int x);
void g(int x) {
    external(x);
    x / 0;
}
```

Might exhaust budget
Getting Started

- Run CTU on your project if interested in additional results
- Run both CTU and non-CTU to get maximal coverage
- Give us feedback about the quality of reports
  - Analysis errors
  - True positives
  - False positives
- CodeChecker supports viewing CTU results!
  - https://github.com/Ericsson/codechecker
Future Work

- Extend the C++ support of ASTImporter
- New strategies to build an exploded graph with good shape?
- Tune default budget for CTU
- Incremental CTU analysis
- Make the binary AST dumps smaller

- Grouping of bug paths in viewers (CodeChecker, XCode, ...
Summary

- Improved the CTU prototype
- Evaluated the results on open source projects
  - CTU found many new potential bugs
  - Analysis time scales well with CPUs
  - Bug/time, bug/memory ratio is good
  - Coverage, quality of reports satisfying
- Works well for C programs
- Improvements needed for C++
- Prepared a patch for upstreaming
Thank you! Questions?