CodeFlowVis

Visualizer of Code Coverage and Execution Flow

Jaehoon Jang
IS661, 2024.05.16

KAIST
Programming Systems Laboratory

Background

- **Code Coverage**
  - Executed source code during the testing
  - Useful in identifying errors or vulnerabilities

```
int main(){
    int a = input();
    int b = 2;
    if (a > b){
        a = a + 1;
    } else {
        b = b + 1;
    }
    return b;
}
```

Background

- **Code Coverage**
  - Executed source code during the testing
  - Useful in identifying errors or vulnerabilities

```
int main(){
    int a = input();
    int b = 2;
    if (a > b){
        a = a + 1;
    } else {
        b = b + 1;
    }
    return b;
}
```

a = 1

Background

- **Code Coverage**
  - Executed source code during the testing
  - Useful in identifying errors or vulnerabilities
- **Code Coverage Tools**
  - JaCoCo[1], GCOV[2]
  - Measure & Visualize code coverage

```c
int main(){
    int a = input();
    int b = 2;
    if (a > b){
        a = a + 1;
    } else {
        b = b + 1;
    }
    return b;
}
```

a = 1

```
-:    0:Runs:2
-:    1:#include <stdio.h>
-:    2:
2:    3:void print_hello(){
2:    4:    printf("hello, world!\n");
2:    4-block  0
2:    5:}
-:    6:
2:    7:int main(){
2:    8:    print_hello();
2:    8-block  0
2:    9:    return 0;
-:   10:}
```

[1] JaCoCo, https://www.jacoco.org/jacoco/
[2] GCOV, https://gcc.gnu.org/onlinedocs/gcc/Gcov.html

Problem

- Difficult to identify the **execution flow**
- It is easy when it is a single function & file.

```
int main(){
    int a = input();
    int b = 2;
    if (a > b){
        f();
    } else {
        g();
    }
    return b;
}
```

a = 1

Problem

- Difficult to identify the **execution flow**
- It is easy when it is a single function & file

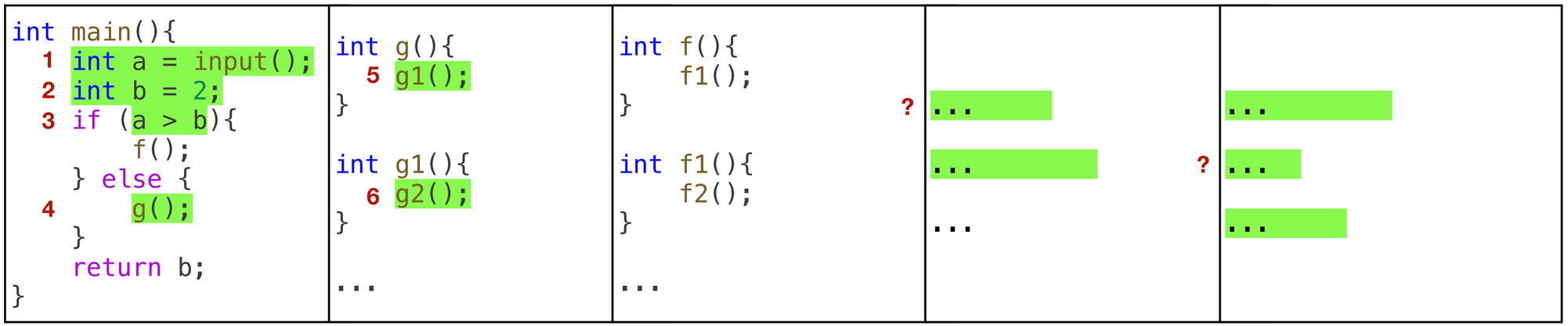
```
int main(){
  1 int a = input();
  2 int b = 2;
  3 if (a > b){
        f();
    } else {
  4     g();
    }
    return b;
}
```

main.c

a = 1

Problem

- Difficult to identify the **execution flow**
- It is easy when it is a single function & file
- But It's hard when there are complex and big codes



Problem

- Difficult to identify the execution flow
- It is easy when it is a single function & file.
- But it is hard when there are complex and big codes.

Why is it important to know the **execution flow**?

```
int main(){
    int a = input();
    int b = 2;
    if (a > b){
        f();
    } else {
        g();
    }
    return b;
}
```

```
int f(){
    f1();
}

int f1(){
    f2();
}

...
```

```
int g(){
    g1();
}

int g1(){
    g2();
}

...
```

```
...

...

...
```

```
...

...

...
```

a = 1

Importance of Execution Flow

- It's difficult to pinpoint exactly **where** and **why** a bug occurs when it arises
- Debugging is challenging
  - understanding the detailed cause of the issue is difficult
- Sometimes, it's impossible to determine why the error occurred
- Therefore, we need to know the **execution flow**

Example

- CVE-2017-14940[3]
  - nm (GNU Binary Utilities), binutils-2.29
  - Denial of Service

Agostino Sarubbo 2017-09-21 12:47:50 UTC Description

Created attachment 10436 [details]
screenshot of the issue

To reproduce:
```
# nm -A -a -l -S -s --special-syms --synthetic --with-symbol-versions -D $FILE
```

I don't get failures but it eats ~230GB of ram.

```
# nm -V
GNU nm (Gentoo git) 2.29.51.20170921
```

[3] CVE-2017-14940, https://nvd.nist.gov/vuln/detail/CVE-2017-14940

Example

- CVE-2017-14940 Demo

Example

- In this bug case,
  - No stack trace log
  - No debug information
  - But, the binutils codebase has almost 5 million lines

Example

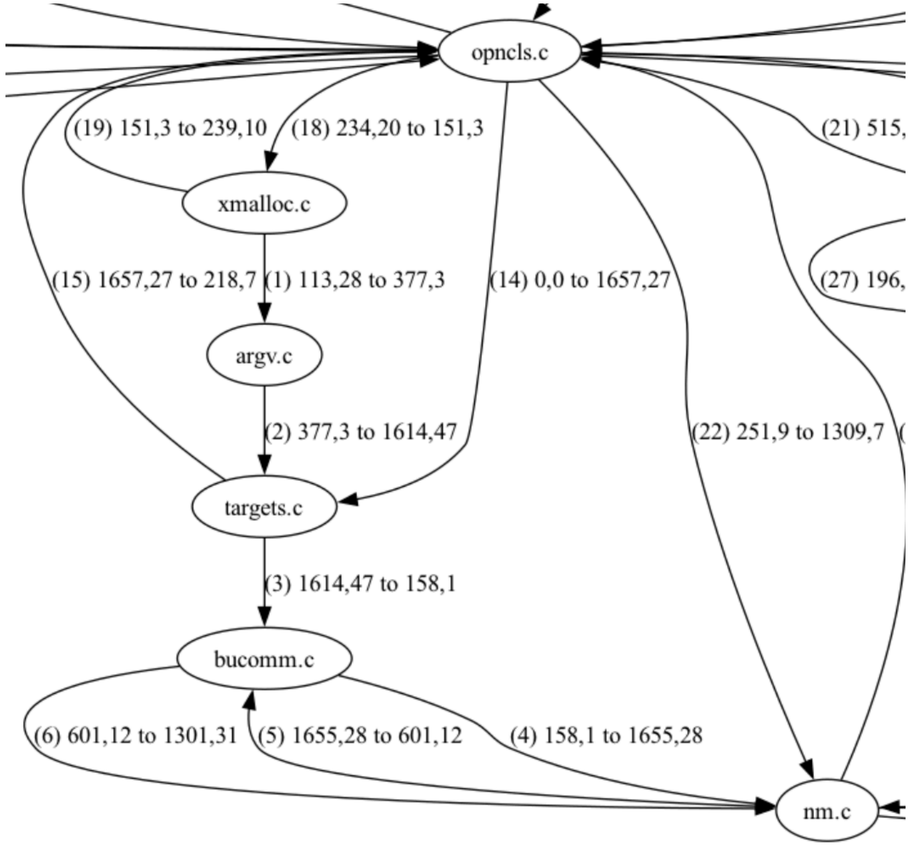
- In this bug case,
  - No stack trace log
  - No debug information
  - But, the binutils codebase has almost 5 million lines

How can we debug and fix this bug?

CodeFlowVis

- Provides code coverage and execution flow results at the same time
- Can know what code a particular input went through when an error occurred
  - Based on VSCode, LLVM

CodeFlowVis

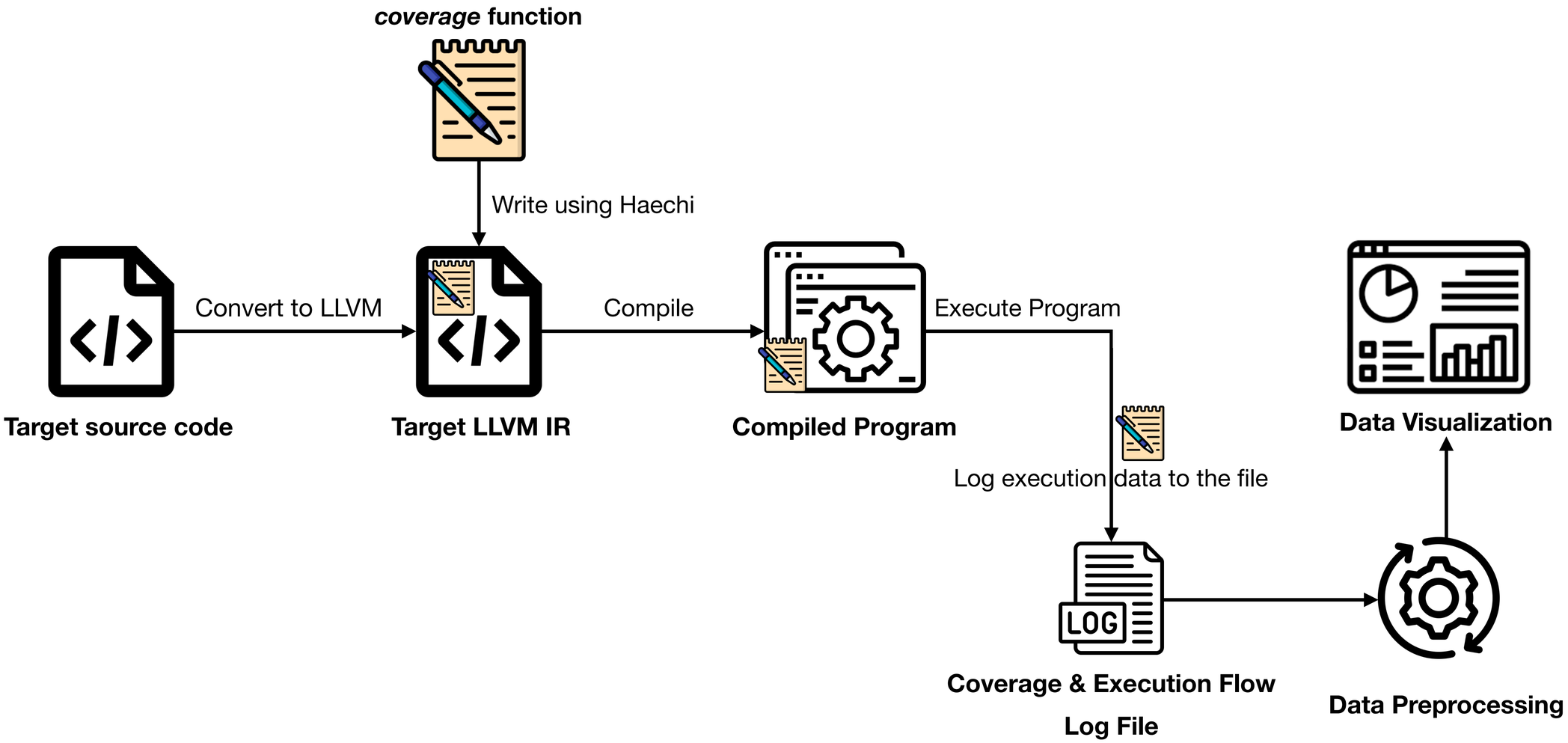


State transition diagram

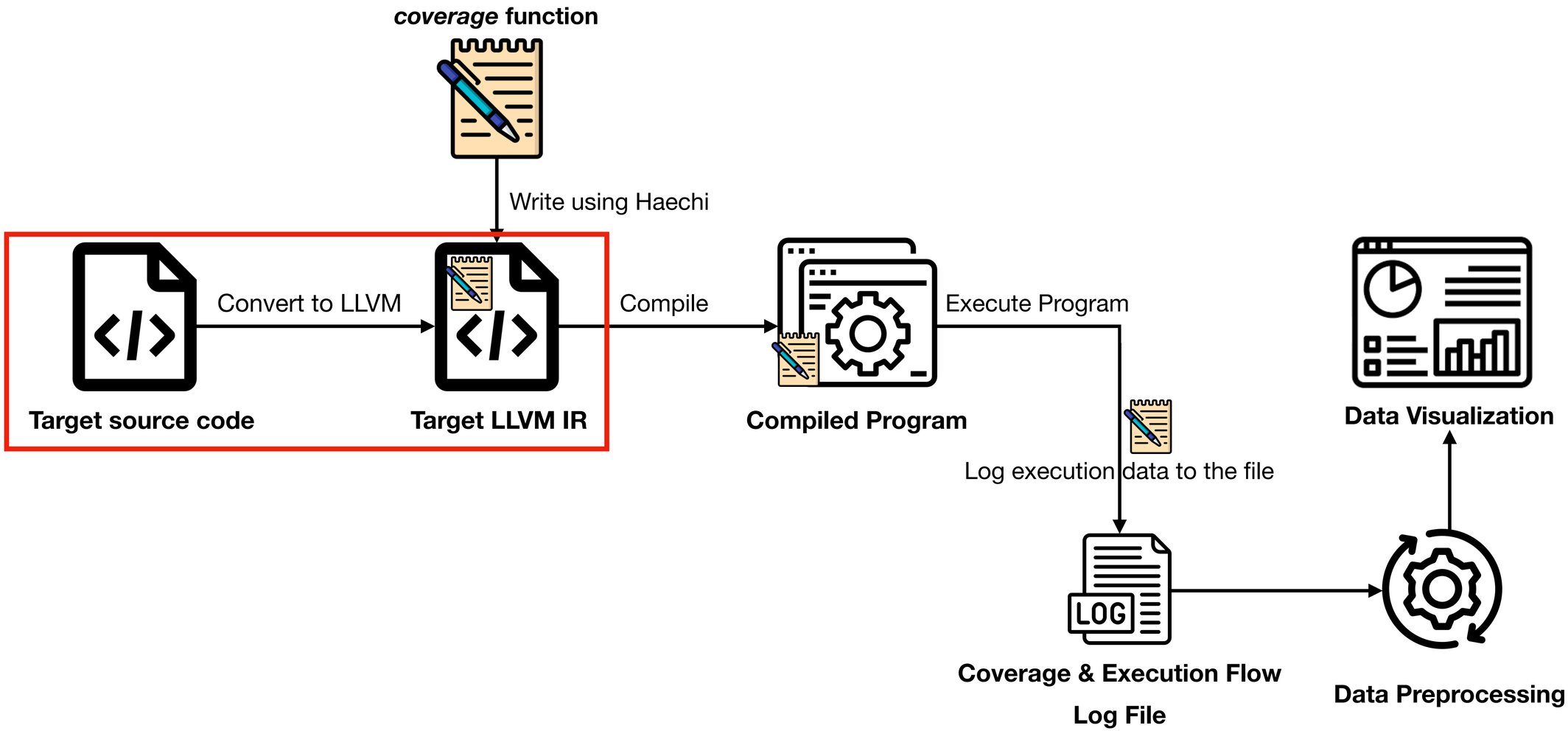
```
24 1301   file = bfd_openr (filename, target ? target : plugin_target);
   1302   if (file == NULL)
   1303     {
   1304       bfd_nonfatal (filename);
   1305       return FALSE;
   1306     }
   1307
   1308   /* If printing line numbers, decompress the debug sections.  */
67 1309   if (line_numbers)
68 1310     file->flags |= BFD_DECOMPRESS;
   1311
69 1312   if (bfd_check_format (file, bfd_archive))
   1313     {
   1314       display_archive (file);
   1315     }
   1316   else if (bfd_check_format_matches (file, bfd_object, &matching))
   1317     {
   1318       set_print_width (file);
416 1319       format->print_object_filename (filename);
```

Highlighting in VSCode

Overview


coverage function
Write using Haechi
Convert to LLVM
Compile
Execute Program
Target source code
Target LLVM IR
Compiled Program
Log execution data to the file
Data Visualization
LOG
Coverage & Execution Flow Log File
Data Preprocessing

Overview


coverage function
Write using Haechi
Convert to LLVM
Compile
Execute Program
Target source code
Target LLVM IR
Compiled Program
Log execution data to the file
Data Visualization
LOG
Coverage & Execution Flow Log File
Data Preprocessing

Convert to LLVM IR

- We collect information by inserting instrumentation based on LLVM
  - We chose LLVM because it supports various platforms
- For a single file, you can convert it to LLVM IR using the *clang* tool
- However, converting an entire project to LLVM is a challenging task
- To solve this problem, we use gllvm[4]

[4] gllvm, https://github.com/SRI-CSL/gllvm

GLLVM?

- Generate LLVM bitcode from C/C++ projects
- Replace the compiler with ***gclang***
- Use ***get-bc*** to extract bitcode

Convert to LLVM IR

- You just need to use *gclang* as the compiler
- You can obtain the bitcode by using *get-bc*
- You can convert the bitcode to LLVM IR with *llvm-dis*
- Through this process, you can convert the entire project to LLVM IR

Convert to LLVM IR

- You just need to use *gclang* as the compiler
- You can obtain the bitcode by using *get-bc*
- You can convert the bitcode to LLVM IR with *llvm-dis*
- Through this process, you can convert the entire project to LLVM IR

```
CC=gclang CXX=gclang++ ./configure $CONFIG_OPTIONS || exit 1
CC=gclang CXX=gclang++ make -j || exit 1
```

Compile with *gclang*

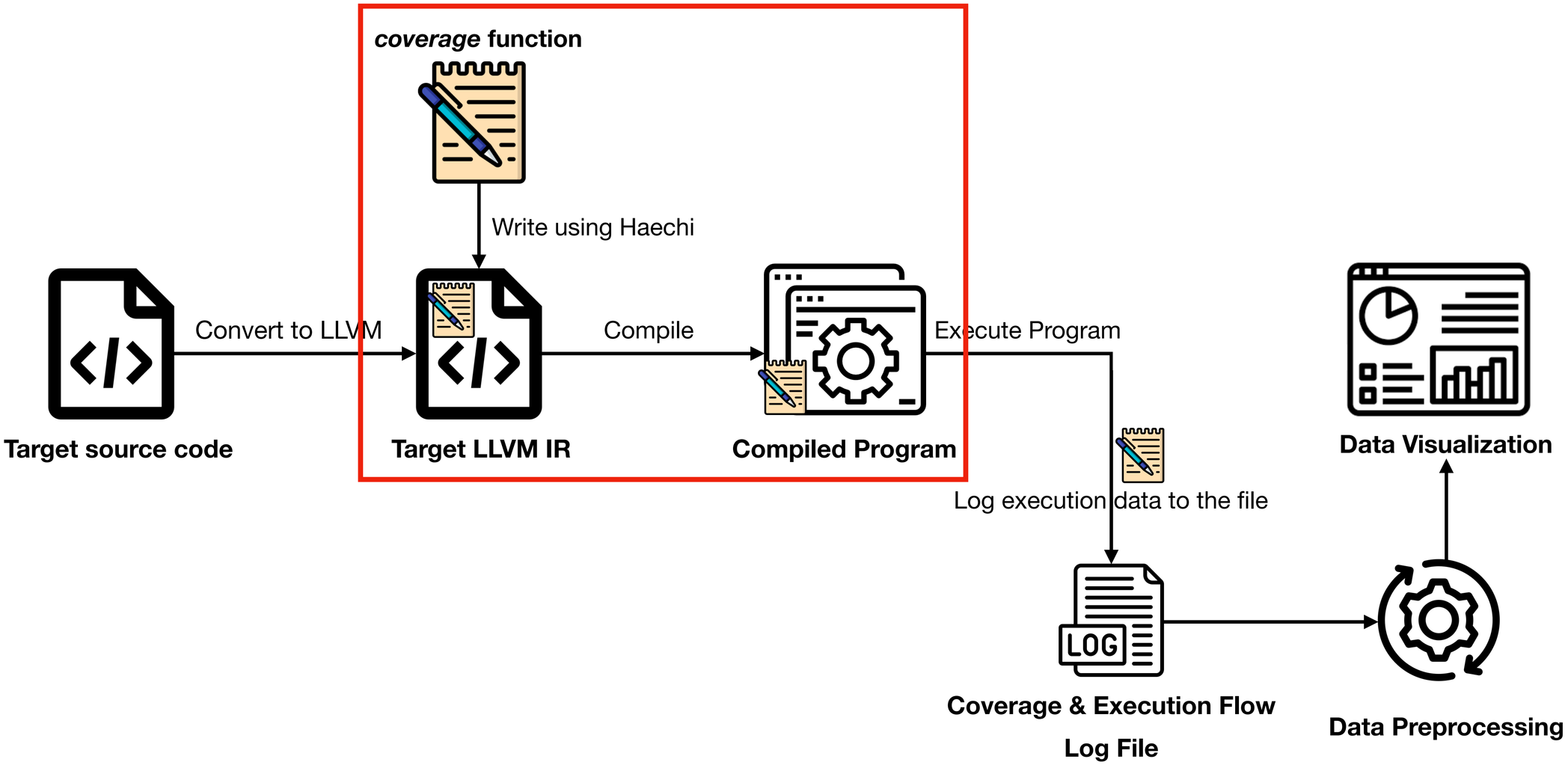
```
root@c2617159fea5:/benchmark/project/binutils-2.29# get-bc nm-haechi
Bitcode file extracted to: nm-haechi.bc.
```

Get bitcode with *get-bc*

```
root@c2617159fea5:/benchmark/project/binutils-2.29# llvm-dis-12 nm-haechi.bc
root@c2617159fea5:/benchmark/project/binutils-2.29# ll nm-haechi.ll
-rw-r--r-- 1 root root 47684731 May 15 04:01 nm-haechi.ll
```

Get readable LLVM with *llvm-dis*

Overview


coverage function
Write using Haechi
Convert to LLVM
Compile
Execute Program
Target source code
Target LLVM IR
Compiled Program
Log execution data to the file
Data Visualization
LOG
Coverage & Execution Flow Log File
Data Preprocessing

Insert Coverage Function

- Define the coverage function in C
- Insert this function into the target LLVM IR
- Compile the LLVM IR with the inserted instrumentation

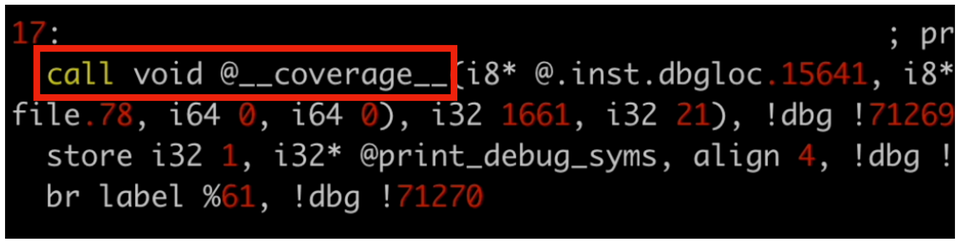
Insert Coverage Function

- Define the coverage function in C
  - Define the *__coverage__* function in a C file
  - It takes *filename*, *line*, and *col* as inputs
  - Saves them to a file
    - [filename]:[line],[col]
    - objcopy.c:152,10

```
void __coverage__(
  char *checked,
  char *filename,
  int line,
  int col)
```

Insert Coverage Function

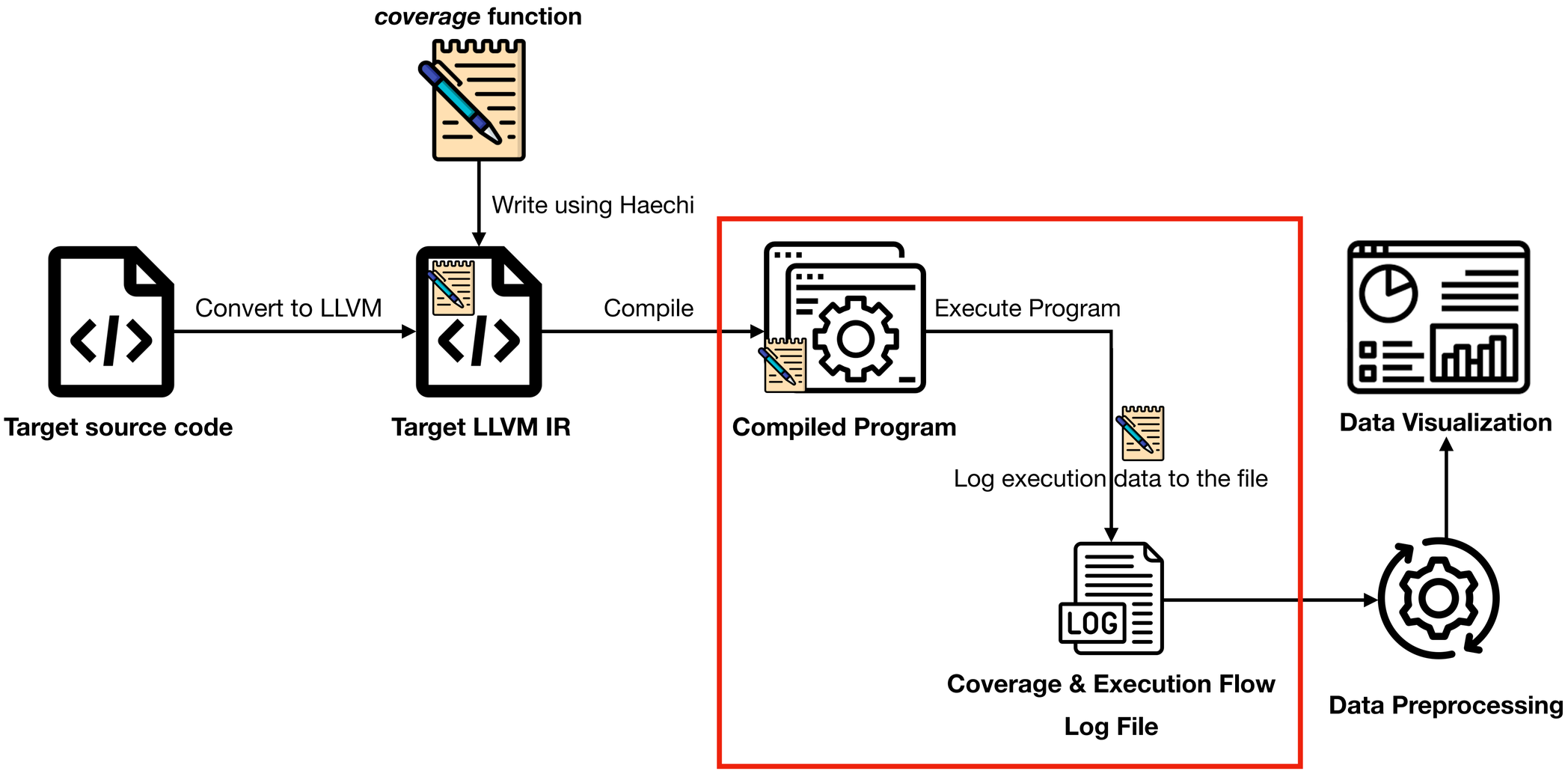
- Insert this function into the target LLVM IR
  - Traverse the entire LLVM IR
    - Insert a "*call __coverage__*" function for every instruction
- Retrieve the filename and line number using the LLVM API



Insert Coverage Function

- Compile the LLVM IR with the inserted instrumentation
  - The coverage function is an external function
    - so it goes through the linking process
    - *llvm-link*
  - The instrumented binary is compiled by
    - *gclang*
- This feature is implemented using a static analysis tool called *Haechi* in our lab

Overview


coverage function
Write using Haechi
Convert to LLVM
Compile
Execute Program
Log execution data to the file
Target source code
Target LLVM IR
Compiled Program
Data Visualization
LOG
Coverage & Execution Flow Log File
Data Preprocessing

Execute Program

- Execute the binary obtained through the previous process
- After execution, data is recorded by the coverage function inserted by Haechi
- The data is saved to a *.cov* file
- The format is [filename]:[line],[column]

```
nm.c:1301,31
opncls.c:62,18
libbfd.c:193,9
libbfd.c:281,5
libbfd.c:283,3
opncls.c:66,7
```

.cov file example

Execute Program

- Execute the binary obtained through the previous process
- After execution, data is recorded by the coverage function inserted by Haechi
- The data is saved to a *.cov* file
- The format is [filename]:[line],[column]
  - capturing both coverage and execution flow
  - The execution flow can be determined
    - by the order in which the data is saved

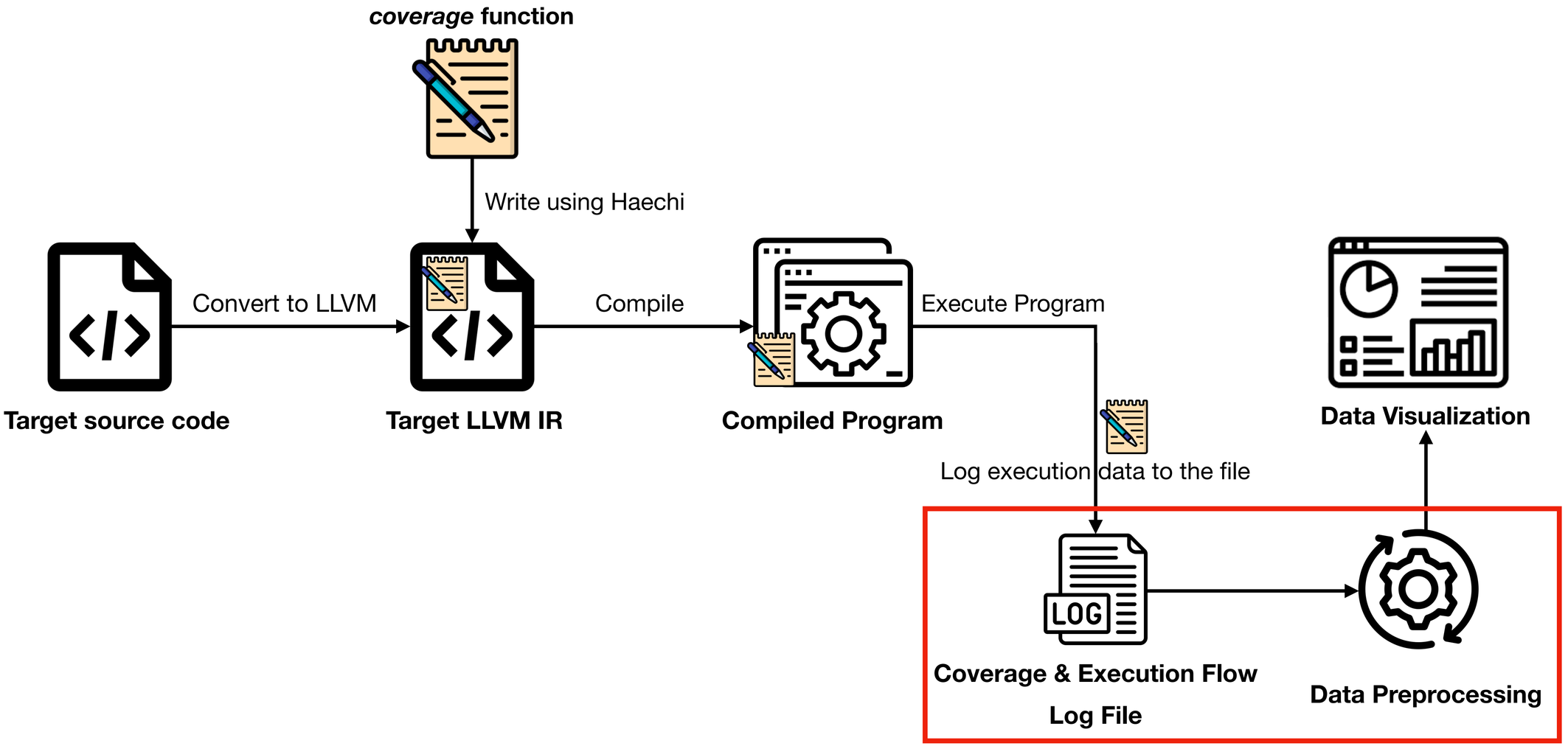
```
1 nm.c:1301,31
2 opncls.c:62,18
3 libbfd.c:193,9
4 libbfd.c:281,5
5 libbfd.c:283,3
6 opncls.c:66,7
```

.cov file example

Execute Program

- I will demonstrate the entire process in a video

Overview


coverage function
Write using Haechi
Convert to LLVM
Compile
Execute Program
Target source code
Target LLVM IR
Compiled Program
Data Visualization
Log execution data to the file
LOG
Coverage & Execution Flow Log File
Data Preprocessing

Data Preprocessing

- The generated .cov file goes through two main processes

1. Change the filename to the project path
2. Generate a State Transition Diagram

- This process is performed using a python script
  - Run the script with
  - python3 *process_cov_file.py* [cov filename] [project path]

Data Preprocessing

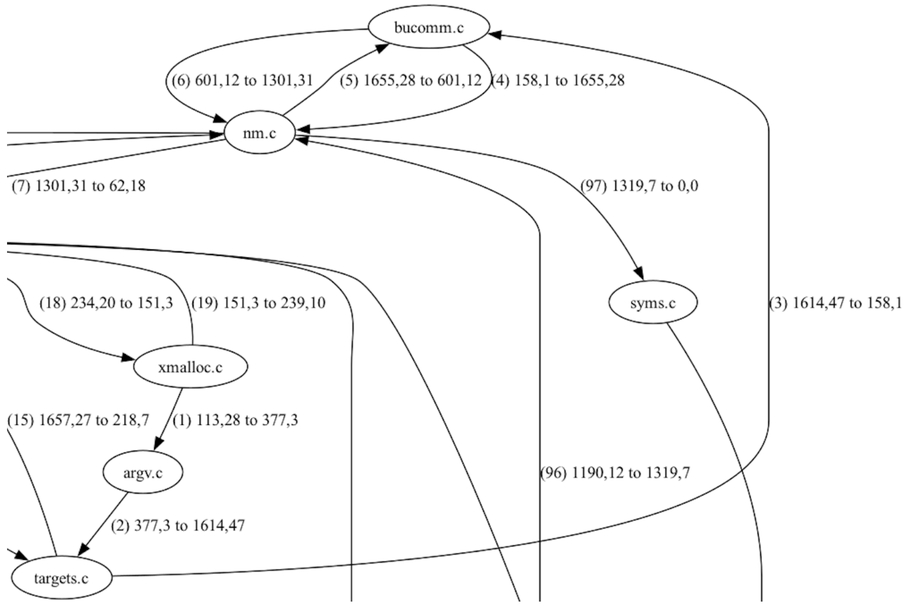
1. Change the filename to the project path
   - This is for highlighting later

Data Preprocessing

2. Generate a State Transition Diagram
   - Generate a state transition diagram to visualize the overall flow
   - with *graphviz* python module

```
libiberty/argv.c:377,3
bfd/targets.c:1614,47
bfd/targets.c:1581,67
binutils/bucomm.c:158,1
binutils/nm.c:1655,28
```

Preprocess



Data Preprocessing

libiberty/argv.c:377,3

bfd/targets.c:1614,47

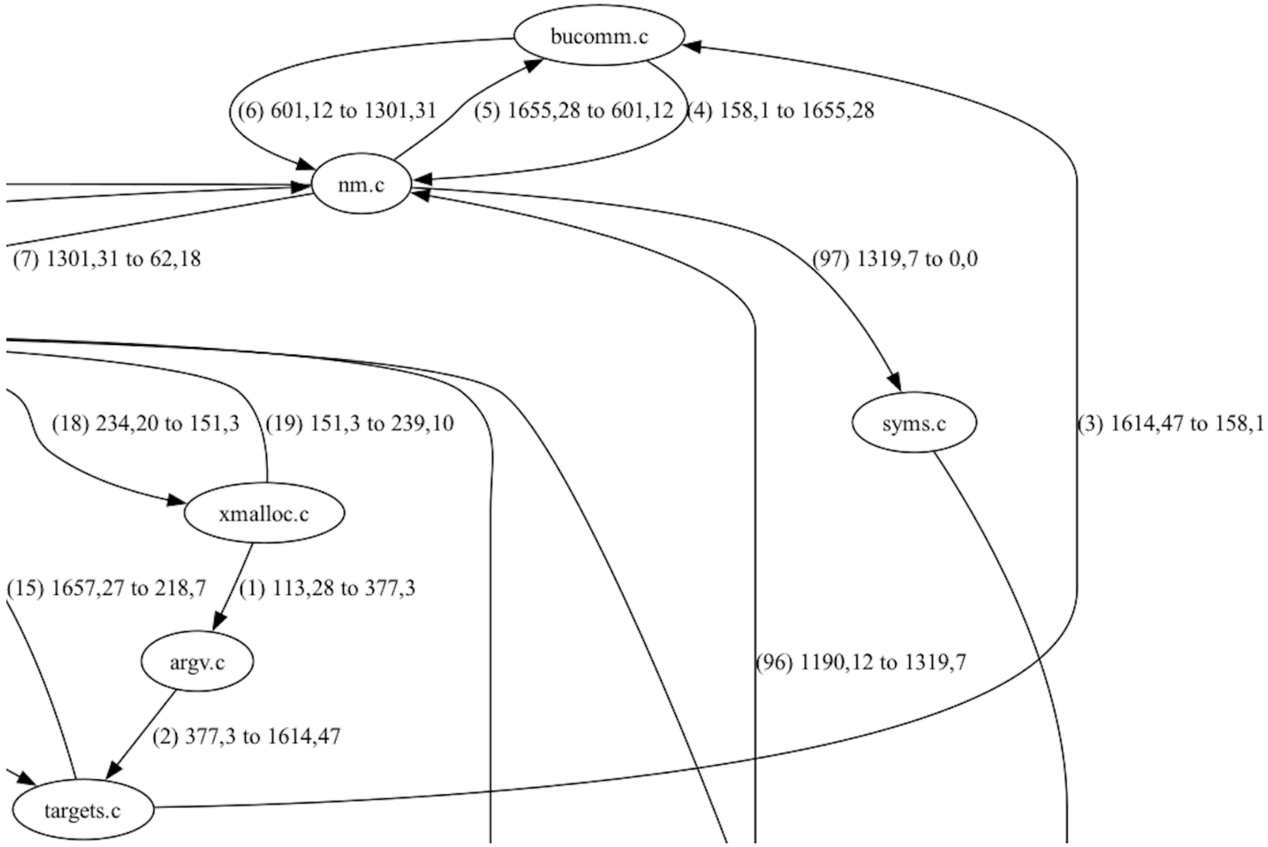
bfd/targets.c:1581,67

binutils/bucomm.c:158,1

binutils/nm.c:1655,28

.cov file

Preprocess



State transition diagram

Data Preprocessing

libiberty/argv.c:377,3

bfd/targets.c:1614,47

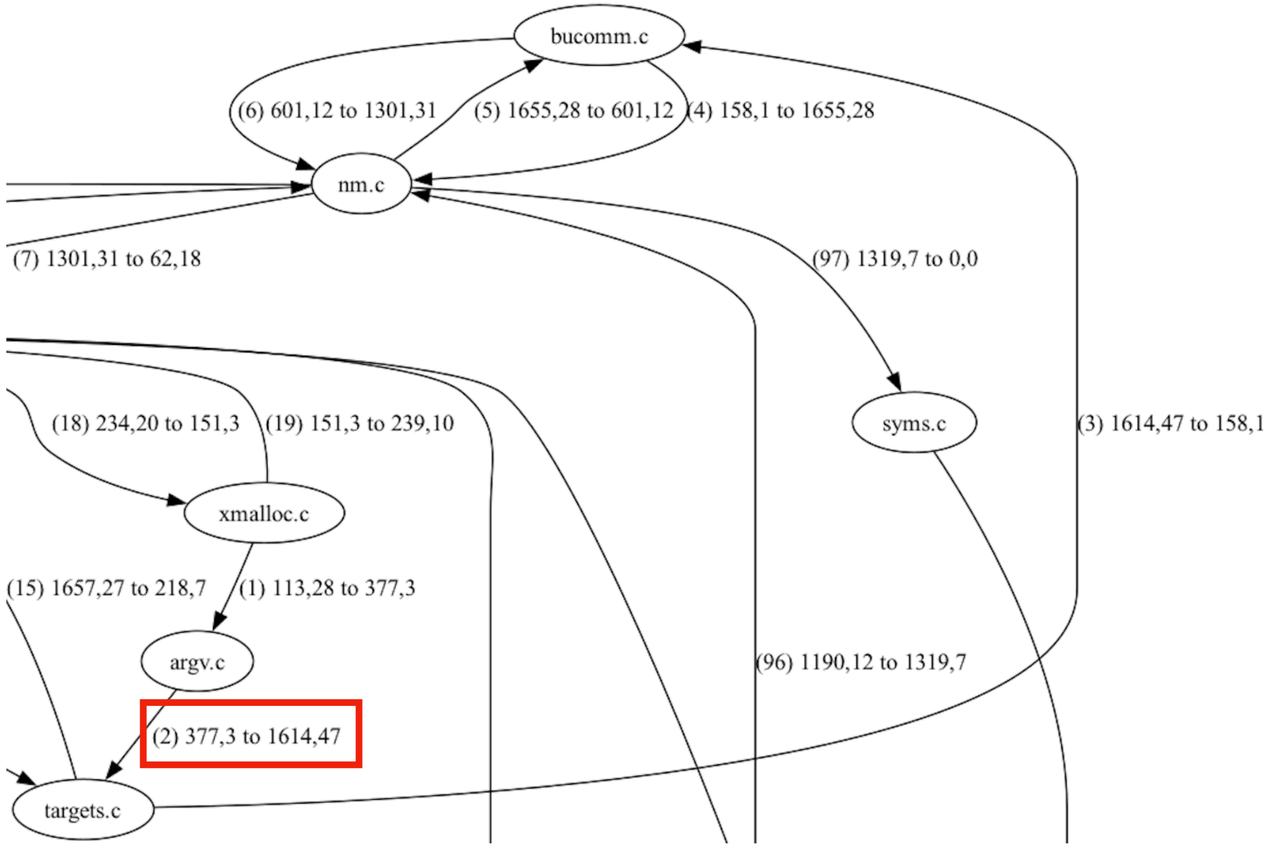
bfd/targets.c:1581,67

binutils/bucomm.c:158,1

binutils/nm.c:1655,28

.cov file

Preprocess



State transition diagram

Data Preprocessing

libiberty/argv.c:377,3

bfd/targets.c:1614,47

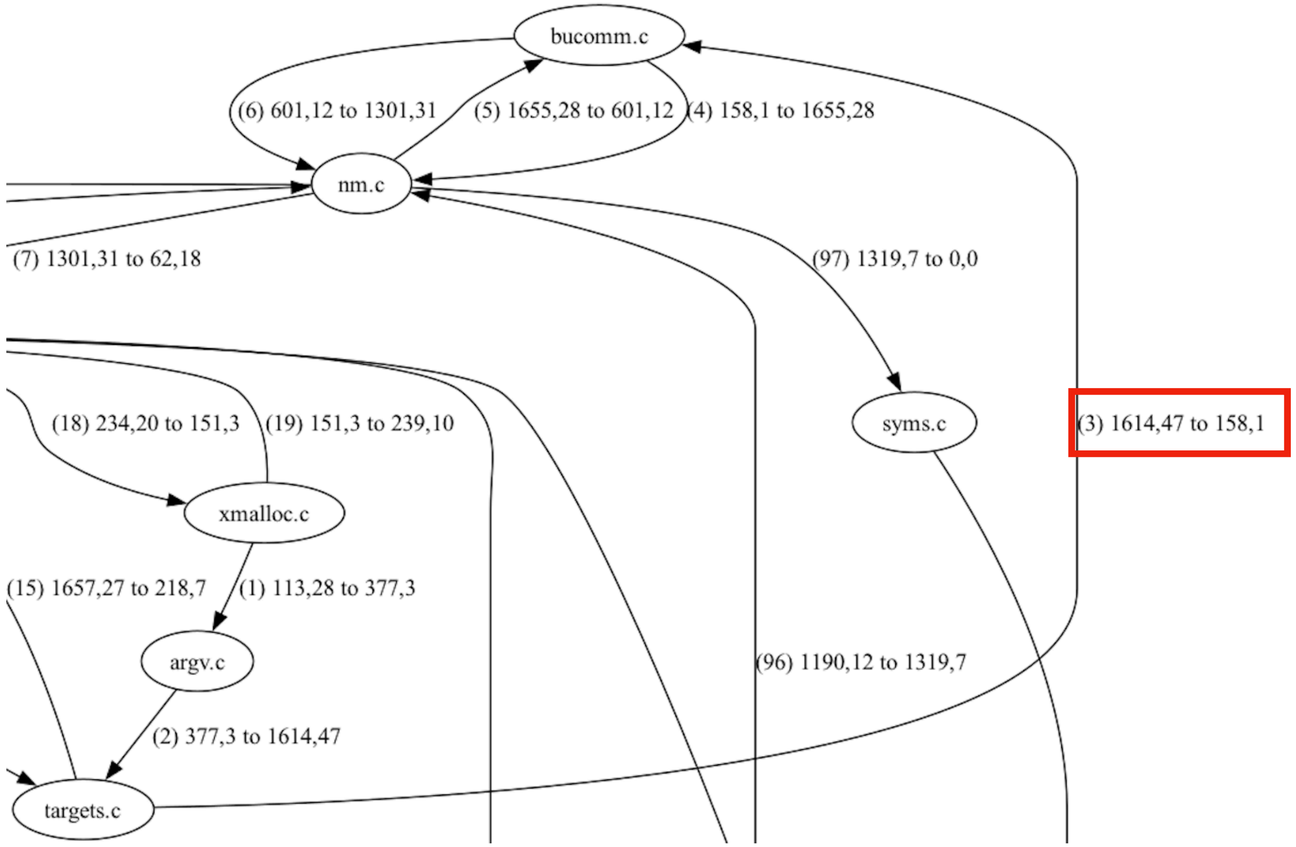
bfd/targets.c:1581,67

binutils/bucomm.c:158,1

binutils/nm.c:1655,28

.cov file

Preprocess



State transition diagram

Data Preprocessing

libiberty/argv.c:377,3

bfd/targets.c:1614,47

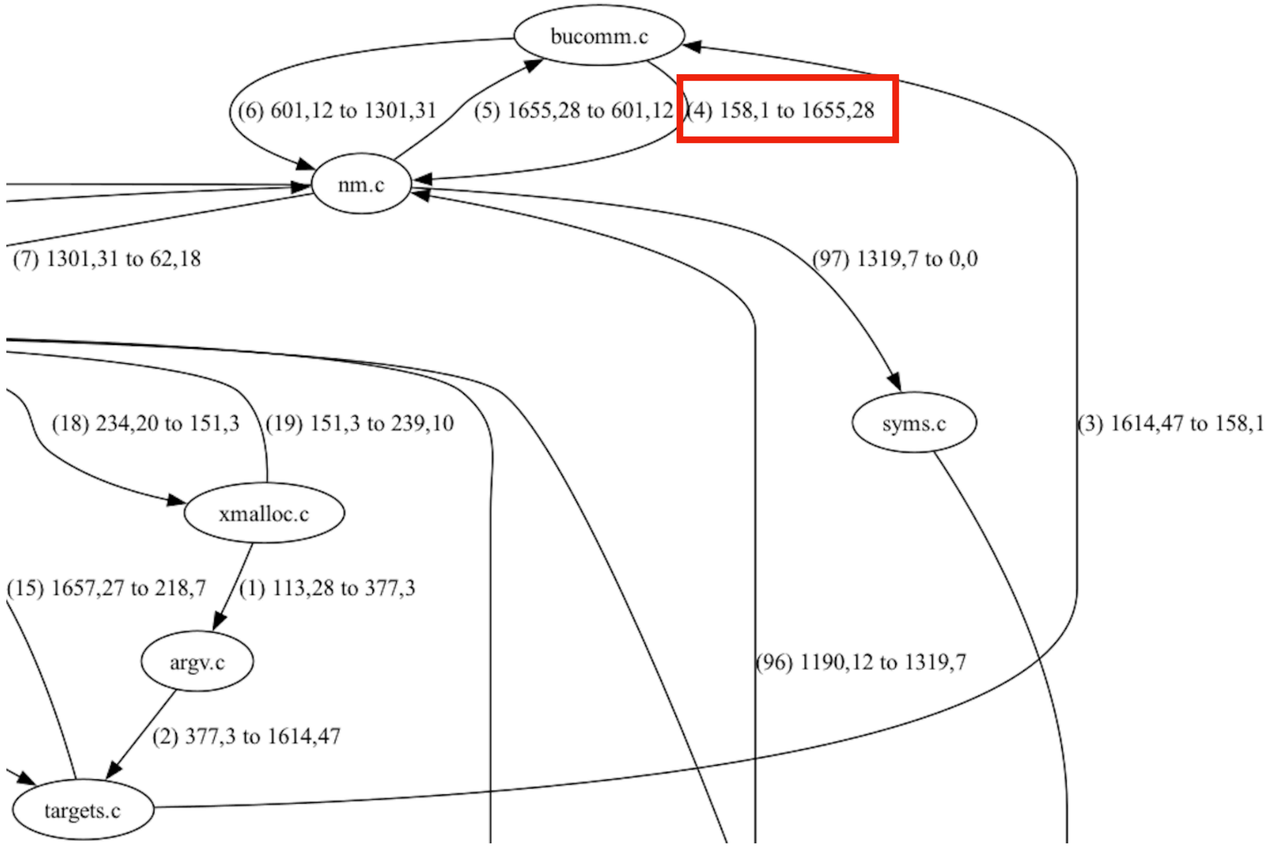
bfd/targets.c:1581,67

binutils/bucomm.c:158,1

binutils/nm.c:1655,28

.cov file

Preprocess

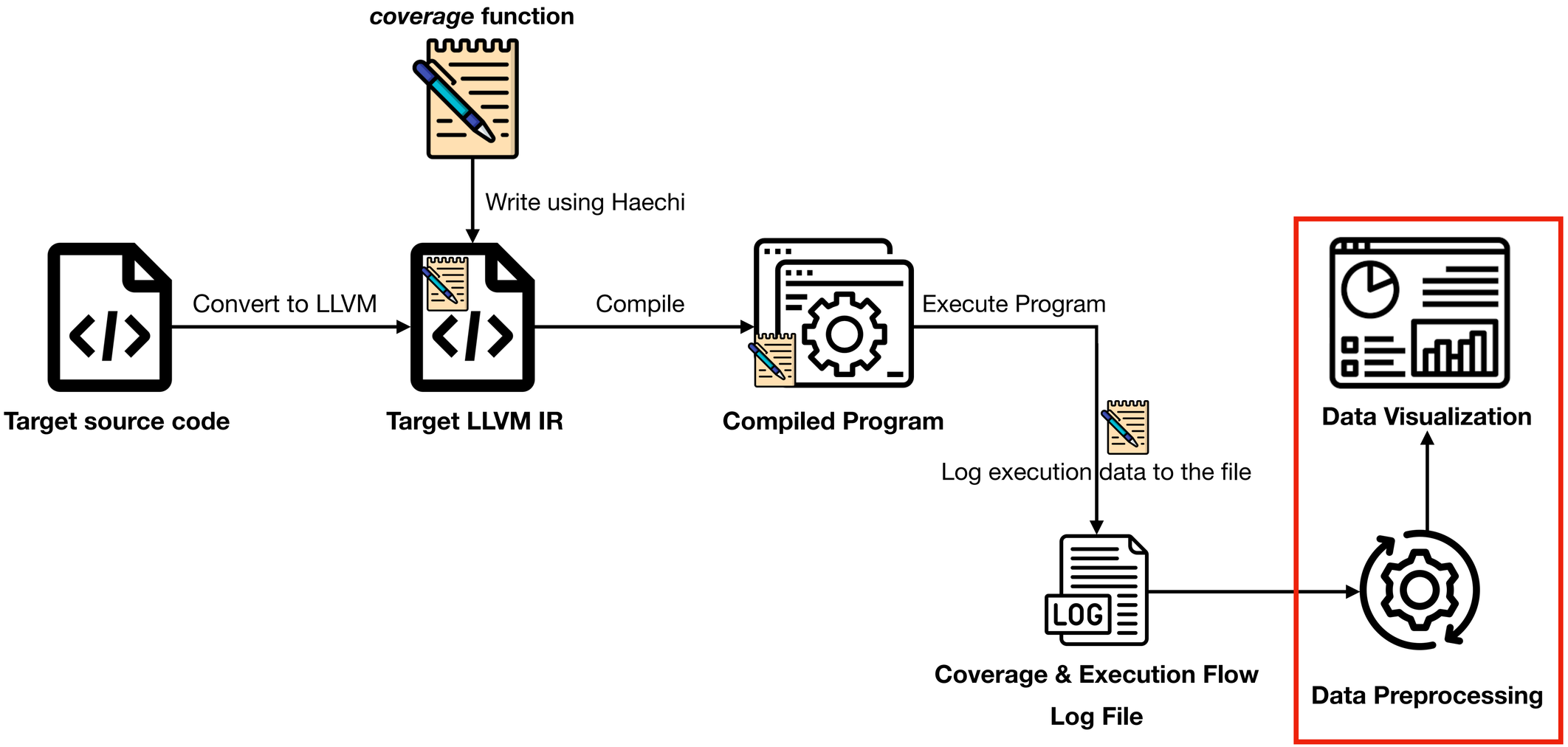


State transition diagram

Data Preprocessing

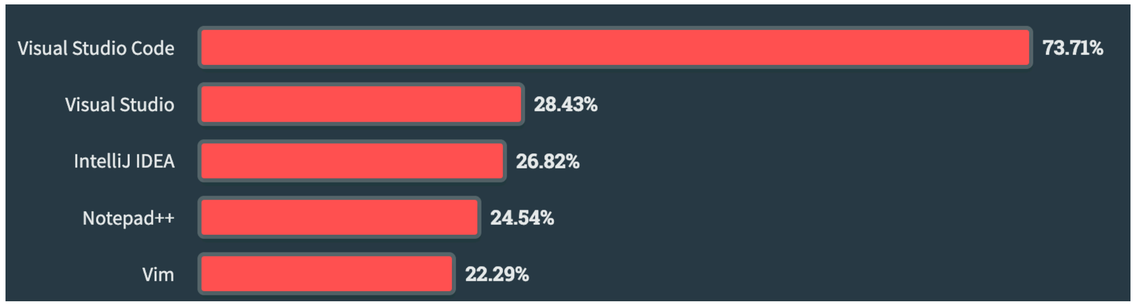
- I will demonstrate the data preprocessing in a video

Overview


coverage function
Write using Haechi
Convert to LLVM
Compile
Execute Program
Target source code
Target LLVM IR
Compiled Program
Log execution data to the file
LOG
Coverage & Execution Flow Log File
Data Visualization
Data Preprocessing

Data Visualization

- Visualize the .cov file obtained through the above process
- The visualization was implemented as a VSCode extension
- VSCode was chosen because it is used by the most developers[5]

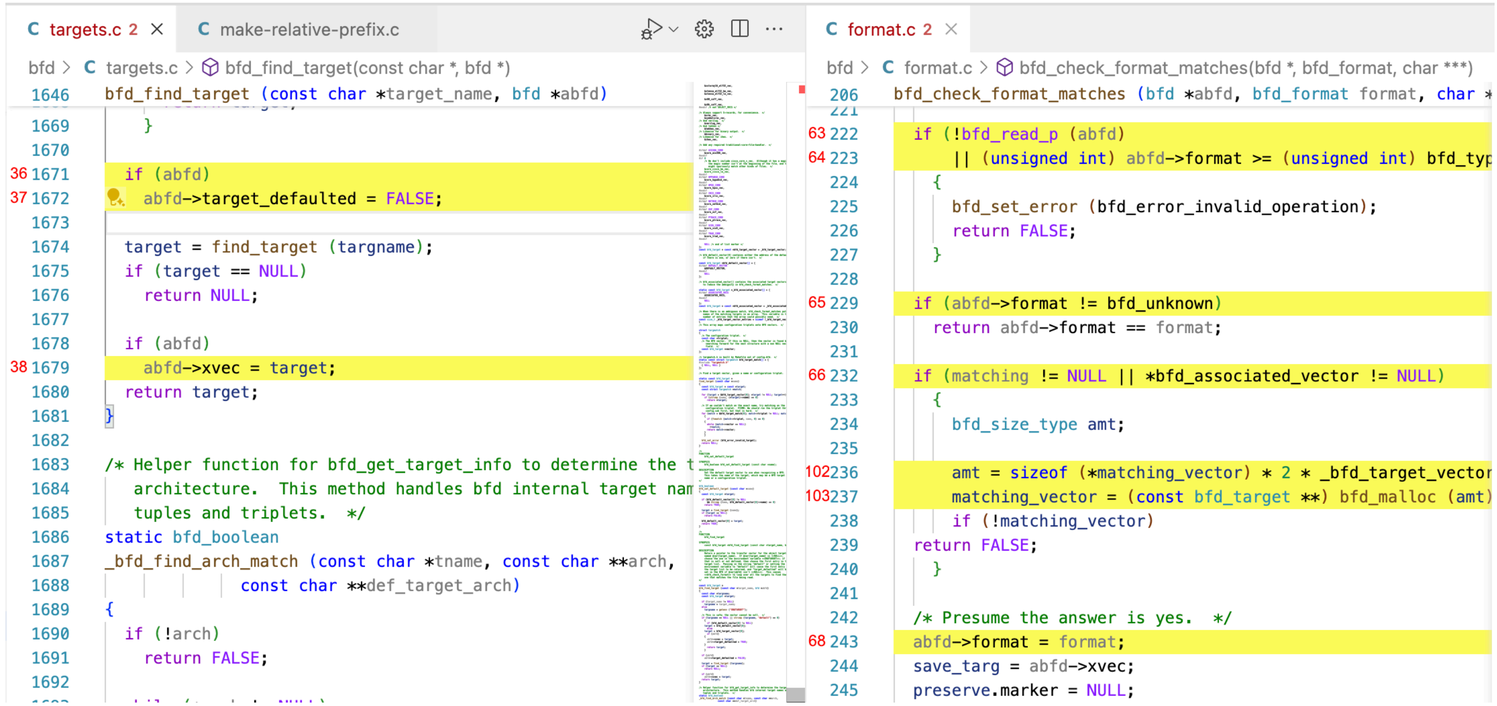


Visual Studio Code remains the preferred IDE across all developers

[5] Stack Overflow Survey 2023, "most popular technologies new collab tools", https://survey.stackoverflow.co/2023/#most-popular-technologies-new-collab-tools

Data Visualization

- Parse the .cov file and highlight the corresponding lines in the file
- Display the execution flow next to the line in red text



Data Visualization

- Since we're using VSCode, we can take advantage of its built-in shortcuts
  - F12 : Go to Definition
  - Cmd+U : Go to Last Cursor Position

Demo

- Let's trace CVE-2017-14940

Demo

- User can control *hdr->sh_info*

```
407 8196        contents = (bfd_byte *) bfd_malloc (hdr->sh_size);
    8197        if (contents == NULL)
    8198      goto error_return_verref;
    8199
408 8200        if (bfd_seek (abfd, hdr->sh_offset, SEEK_SET) != 0
409 8201        || bfd_bread (contents, hdr->sh_size, abfd) != hdr->sh_size)
    8202      goto error_return_verref;
    8203
    8204        elf_tdata (abfd)->verref = (Elf_Internal_Verneed *)
410 8205      bfd_zalloc2 (abfd, hdr->sh_info, sizeof (Elf_Internal_Verneed));
```

Demo

- User can control *hdr->sh_info*
- *hdr->sh_info* becomes *nmemb*
- Multiply *size* and *nmemb*
- Allocate as much as *size*

```
1026  bfd_zalloc2 (bfd *abfd, bfd_size_type nmemb, bfd_size_type size)
1027  {
1028    void *res;
1029
1030    if ((nmemb | size) >= HALF_BFD_SIZE_TYPE
1031        && size != 0
1032        && nmemb > ~(bfd_size_type) 0 / size)
1033      {
1034        bfd_set_error (bfd_error_no_memory);
1035        return NULL;
1036      }
1037
411 1038    size *= nmemb;
1039
1040    res = bfd_alloc (abfd, size);
1041    if (res)
1042      memset (res, 0, (size_t) size);
```

size can be a very large value

Demo

- User can control *hdr->sh_info*
- *hdr->sh_info* becomes *nmemb*
- Mult
- Allocate as much as *size*

```
root@c2617159fea5:/benchmark/RUNDIR-binutils-2.29/binutils-2.29# binutils/nm-new -A -a -l -S -s --special-
syms --synthetic --with-symbol-versions -D /benchmark/poc/nm/2017-14940
size: 250181845056
^C
```

```
1026  bfd_zalloc2 (bfd *abfd, bfd_size_type nmemb, bfd_size_type size)
1027  {
1028    void *res;
1029
1030    if ((nmemb | size) >= HALF_BFD_SIZE_TYPE
1031        && size != 0
1032        && nmemb > ~(bfd_size_type) 0 / size)
1033      {
1039
1040    res = bfd_alloc (abfd, size);
1041    if (res)
1042      memset (res, 0, (size_t) size);
```

size can be a very large value

Demo

- User can control *hdr->sh_info*
- *hdr->sh_info* becomes *nmemb*
- Mult
- Allocate as much as *size*

```
root@c2617159fea5:/benchmark/RUNDIR-binutils-2.29/binutils-2.29# binutils/nm-new -A -a -l -S -s --special-
syms --synthetic --with-symbol-versions -D /benchmark/poc/nm/2017-14940
size: 250181845056
^C
```

```
1026  bfd_zalloc2 (bfd *abfd, bfd_size_type nmemb, bfd_size_type size)
1027  {
1028    void *res;
1029
1030    if ((nmemb | size) >= HALF_BFD_SIZE_TYPE
1031        && size != 0
1032        && nmemb > ~(bfd_size_type) 0 / size)
1033      {
1039
1040    res = bfd_alloc (abfd, size);
1041    if (res)
1042      memset (res, 0, (size_t) size);
```

size can be a very large value

Demo

- User can control *hdr->sh_info*
- *hdr->sh_info* becomes *nmemb*
- Mult
- Allocate as much as *size*

```
root@c2617159fea5:/benchmark/RUNDIR-binutils-2.29/binutils-2.29# binutils/nm-new -A -a -l -S -s --special-
syms --synthetic --with-symbol-versions -D /benchmark/poc/nm/2017-14940
size: 250181845056
^C
```

```
1026  bfd_zalloc2 (bfd *abfd, bfd_size_type nmemb, bfd_size_type size)
1027  {
1028    void *res;
1029
1030    if ((nmemb | size) >= HALF_BFD_SIZE_TYPE
1031        && size != 0
1032        && nmemb > ~(bfd_size_type) 0 / size)
1033      {
1039
1040    res = bfd_alloc (abfd, size);
1041    if (res)
1042      memset (res, 0, (size_t) size);
```

size can be a very large value

We can trace the bug with *CodeFlowVis*, without debug info

Evaluation

- RQ1. Is code coverage clearly provided according to the input?
- RQ2. Is the program execution flow clearly provided according to the input?

Evaluation

- RQ1. Is code coverage clearly provided according to the input?

```
1026  bfd_zalloc2 (bfd *abfd, bfd_size_type nmemb, bfd_size_type size)
1027  {
1028    void *res;
1029
1030    if ((nmemb | size) >= HALF_BFD_SIZE_TYPE
1031        && size != 0
1032        && nmemb > ~(bfd_size_type) 0 / size)
1033      {
1034        bfd_set_error (bfd_error_no_memory);
1035        return NULL;
1036      }
1037
411 1038   size *= nmemb;
1039
1040    res = bfd_alloc (abfd, size);
1041    if (res)
1042      memset (res, 0, (size_t) size);
```

Evaluation

- RQ1. Is code coverage clearly provided according to the input?

```
1026  bfd_zalloc2 (bfd *abfd, bfd_size_type nmemb, bfd_size_type size)
1027  {
1028    void *res;
1029
1030    if ((nmemb | size) >= HALF_BFD_SIZE_TYPE
1031        && size != 0
1032        && nmemb > ~(bfd_size_type) 0 / size)
1033      {
1034        bfd_set_error (bfd_error_no_memory);
1035        return NULL;
1036      }
1037
411 1038   size *= nmemb;
1039
1040    res = bfd_alloc (abfd, size);
1041    if (res)
1042      memset (res, 0, (size_t) size);
```

Even though it was run, it was not displayed.

Evaluation

- RQ2. Is the program execution flow clearly provided according to the input?

Evaluation

- RQ2. Is the program execution flow clearly provided according to the input?

```
READ of size 1 at 0x00000089fce4 thread T0
    #0 0x42f686 in strcmp /src/llvm-project/compiler-rt/lib/asan/../sanitizer_common/sanitizer_common_interceptors.inc:449:5
    #1 0x60f8e2 in _bfd_elf_get_reloc_section /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:3562:10
    #2 0x613371 in assign_section_numbers /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:3836:8
    #3 0x613371 in _bfd_elf_compute_section_file_positions /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:4176:8
    #4 0x62bc26 in _bfd_elf_write_object_contents /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:6222:12
    #5 0x5a060a in bfd_close /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/opncls.c:734:13
    #6 0x4d2276 in copy_file /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:2886:51
    #7 0x4ce203 in copy_main /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:4792:3
    #8 0x4c74ca in main /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:4893:5
    #9 0x7f7d8fbdd082 in __libc_start_main /build/glibc-SzIz7B/glibc-2.31/csu/../csu/libc-start.c:308:16
    #10 0x41c57d in _start (/benchmark/bin/DAFL/objcopy-2017-8393+0x41c57d)
```

CVE-2017-8393

Evaluation

- RQ2. Is the program execution flow clearly provided according to the input?

```
READ of size 1 at 0x00000089fce4 thread T0
    #0 0x42f686 in strcmp /src/llvm-project/compiler-rt/lib/asan/../sanitizer_common/sanitizer_common_interceptors.inc:449:5
    #1 0x60f8e2 in _bfd_elf_get_reloc_section /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:3562:10
    #2 0x613371 in assign_section_numbers /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:3836:8
    #3 0x613371 in _bfd_elf_compute_section_file_positions /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:4176:8
    #4 0x62bc26 in _bfd_elf_write_object_contents /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:6222:12
    #5 0x5a060a in bfd_close /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/opncls.c:734:13
    #6 0x4d2276 in copy_file /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:2886:51
    #7 0x4ce203 in copy_main /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:4792:3
    #8 0x4c74ca in main /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:4893:5
    #9 0x7f7d8fbdd082 in __libc_start_main /build/glibc-SzIz7B/glibc-2.31/csu/../csu/libc-start.c:308:16
    #10 0x41e57d in _start (/benchmark/bin/RAFL/objcopy-2017-8393+0x41c57d)
```

```
999  3547    type = elf_section_data (reloc_sec)->this_hdr.sh_type;
     3548    if (type != SHT_REL && type != SHT_RELA)
     3549      return NULL;
     3550
     3551    /* We look up the section the relocs apply to by name.  */
1000 3552    name = reloc_sec->name;
     3553    if (type == SHT_REL)
1001 3554      name += 4;
     3555    else
     3556      name += 5;
     3557
     3558    /* If a target needs .got.plt section, relocations in rela.plt/rel.plt
     3559       section apply to .got.plt section.  */
1002 3560    abfd = reloc_sec->owner;
     3561    if (get_elf_backend_data (abfd)->want_got_plt
1003 3562        && strcmp (name, ".plt") == 0)
```

Evaluation

- RQ2. Is the program execution flow clearly provided according to the input?

```
READ of size 1 at 0x00000089fce4 thread T0
    #0 0x42f686 in strcmp /src/llvm-project/compiler-rt/lib/asan/../sanitizer_common/sanitizer_common_interceptors.inc:449:5
    #1 0x60f8e2 in _bfd_elf_get_reloc_section /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:3562:10
    #2 0x613371 in assign_section_numbers /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:3836:8
    #3 0x613371 in _bfd_elf_compute_section_file_positions /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:4176:8
    #4 0x62bc26 in _bfd_elf_write_object_contents /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:6222:12
    #5 0x5a060a in bfd_close /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/opncls.c:734:13
    #6 0x4d2276 in copy_file /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:2886:51
    #7 0x4ce203 in copy_main /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:4792:3
    #8 0x4c74ca in main /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:4893:5
    #9 0x7f7d8fbdd082 in __libc_start_main /build/glibc-SzIz7B/glibc-2.31/csu/../csu/libc-start.c:308:16
    #10 0x41e57d in _start (/benchmark/bin/RAFL/objcopy-2017-8393+0x41c57d)
```

```
999  3547    type = elf_section_data (reloc_sec)->this_hdr.sh_type;
     3548    if (type != SHT_REL && type != SHT_RELA)
     3549      return NULL;
     3550
     3551    /* We look up the section the relocs apply to by name.  */
1000 3552    name = reloc_sec->name;
     3553    if (type == SHT_REL)
1001 3554      name += 4;
     3555    else
     3556      name += 5;
     3557
     3558    /* If a target needs .got.plt section, relocations in rela.plt/rel.plt
     3559       section apply to .got.plt section.  */
1002 3560    abfd = reloc_sec->owner;
     3561    if (get_elf_backend_data (abfd)->want_got_plt
1003 3562        && strcmp (name, ".plt") == 0)
```

```
3868            if (elf_section_data (s)->this_hdr.sh_entsize == 0)
3869              elf_section_data (s)->this_hdr.sh_entsize
3870                = 4 + 2 * bfd_get_arch_size (abfd) / 8;
3871          }
```

Evaluation

- RQ2. Is the program execution flow clearly provided according to the input?

```
READ of size 1 at 0x00000089fce4 thread T0
    #0 0x42f686 in strcmp /src/llvm-project/compiler-rt/lib/asan/../sanitizer_common/sanitizer_common_interceptors.inc:449:5
    #1 0x60f8e2 in _bfd_elf_get_reloc_section /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:3562:10
    #2 0x613371 in assign_section_numbers /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:3836:8
    #3 0x613371 in _bfd_elf_compute_section_file_positions /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:4176:8
    #4 0x62bc26 in _bfd_elf_write_object_contents /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/elf.c:6222:12
    #5 0x5a060a in bfd_close /benchmark/RUNDIR-binutils-2.28/binutils-2.28/bfd/opncls.c:734:13
    #6 0x4d2276 in copy_file /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:2886:51
    #7 0x4ce203 in copy_main /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:4792:3
    #8 0x4c74ca in main /benchmark/RUNDIR-binutils-2.28/binutils-2.28/binutils/objcopy.c:4893:5
    #9 0x7f7d8fbdd082 in __libc_start_main /build/glibc-SzIz7B/glibc-2.31/csu/../csu/libc-start.c:308:16
    #10 0x41e57d in _start (/benchmark/bin/RAFL/objcopy-2017-8393+0x41c57d)
```

```
999  3547    type = elf_section_data (reloc_sec)->this_hdr.sh_type;
     3548    if (type != SHT_REL && type != SHT_RELA)
     3549      return NULL;
     3550
     3551    /* We look up the section the relocs apply to by name.  */
1000 3552    name = reloc_sec->name;
     3553    if (type == SHT_REL)
1001 3554      name += 4;
     3555    else
     3556      name += 5;
     3557
     3558    /* If a target needs .got.plt section, relocations in rela.plt/rel.plt
     3559       section apply to .got.plt section.  */
1002 3560    abfd = reloc_sec->owner;
     3561    if (get_elf_backend_data (abfd)->want_got_plt
1003 3562        && strcmp (name, ".plt") == 0)
```

```
3868            if (elf_section_data (s)->this_hdr.sh_entsize == 0)
3869              elf_section_data (s)->this_hdr.sh_entsize
3870                = 4 + 2 * bfd_get_arch_size (abfd) / 8;
3871          }
```

```
862 4161  if (bed->elf_backend_begin_write_processing)
    4162    (*bed->elf_backend_begin_write_processing) (abfd, link_info);
    4163
863 4164  if (! prep_headers (abfd))
    4165    return FALSE;
    4166
    4167  /* Post process the headers if necessary.  */
896 4168  (*bed->elf_backend_post_process_headers) (abfd, link_info);
    4169
    4170  fsargs.failed = FALSE;
    4171  fsargs.link_info = link_info;
    4172  bfd_map_over_sections (abfd, elf_fake_sections, &fsargs);
    4173  if (fsargs.failed)
    4174    return FALSE;
    4175
942 4176  if (!assign_section_numbers (abfd, link_info))
    4177    return FALSE;
    4178
    4179  /* The backend linker builds symbol table information itself.  */
1009 4180  need_symtab = (link_info == NULL
1010 4181         && (bfd_get_symcount (abfd) > 0
1011 4182             || ((abfd->flags & (EXEC_P | DYNAMIC | HAS_RELOC))
    4183                 == HAS_RELOC)));
```

Evaluation

- RQ2. Is the program execution flow clearly provided according to the input?

It does not display **100%** of both code coverage and execution flow

However, it provides **enough information** to trace bugs effectively

Future Work

- Make the state transition diagram interactive
- It does not provide complete code coverage information
  - Some lines are not displayed during the C ↔ LLVM conversion process
  - Combine with GCOV

Summary

- Coverage alone makes it **difficult** to understand the **execution flow**
  - Tracing becomes challenging when a bug occurs
- To address these issues, we propose the **CodeFlowVis** visualizer
- It stores coverage and execution flow information based on LLVM
- It visualizes this information to facilitate **efficient debugging**
- We have **successfully traced real bugs** using this tool