LLVM SanitizerCoverage 를 활용한 coverage 측정

스마트보안전공 202334843 이동하

What is Code Coverage?

AKA. Test coverage

software testing strategy

a metric that measures how much of a program's source code is executed during testing

Programs with a high code coverage percentage are less likely to have undetected bugs

```java
/**
 * @param price   The price to set.
 */
public void setPrice(String price) throws RecipeException{
    int amtPrice = 0;
    try {
        amtPrice = Integer.parseInt(price);
    } catch (NumberFormatException e) {
        throw new RecipeException("Price must be a positive integer");
    }
    if (amtPrice >= 0) {
        this.price = amtPrice;
    } else {
        throw new RecipeException("Price must be a positive integer");
    }
}
```

Ref : https://blog.mergify.com/code-coverage-everything-you-need-to-know/

What is Code Coverage?

AKA. Test coverage

software testing strategy

a metric that measures how much of a program's source code is executed during testing

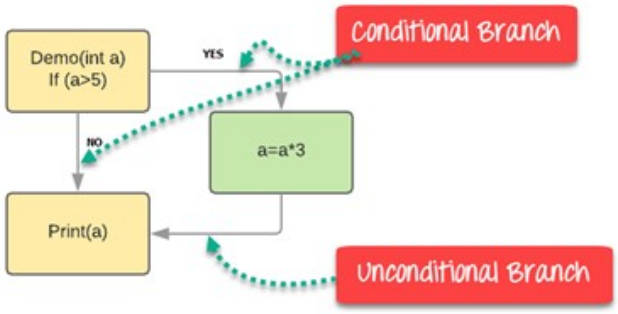
Programs with a **high code coverage percentage are less likely to have undetected bugs**

```
/**
 * @param price   The price to set.
 */
public void setPrice(String price) throws RecipeException{
    int amtPrice = 0;
    try {
        amtPrice = Integer.parseInt(price);
    } catch (NumberFormatException e) {
        throw new RecipeException("Price must be a positive integer");
    }
    if (amtPrice >= 0) {
        this.price = amtPrice;
    } else {
        throw new RecipeException("Price must be a positive integer");
    }
}
```

HIT!

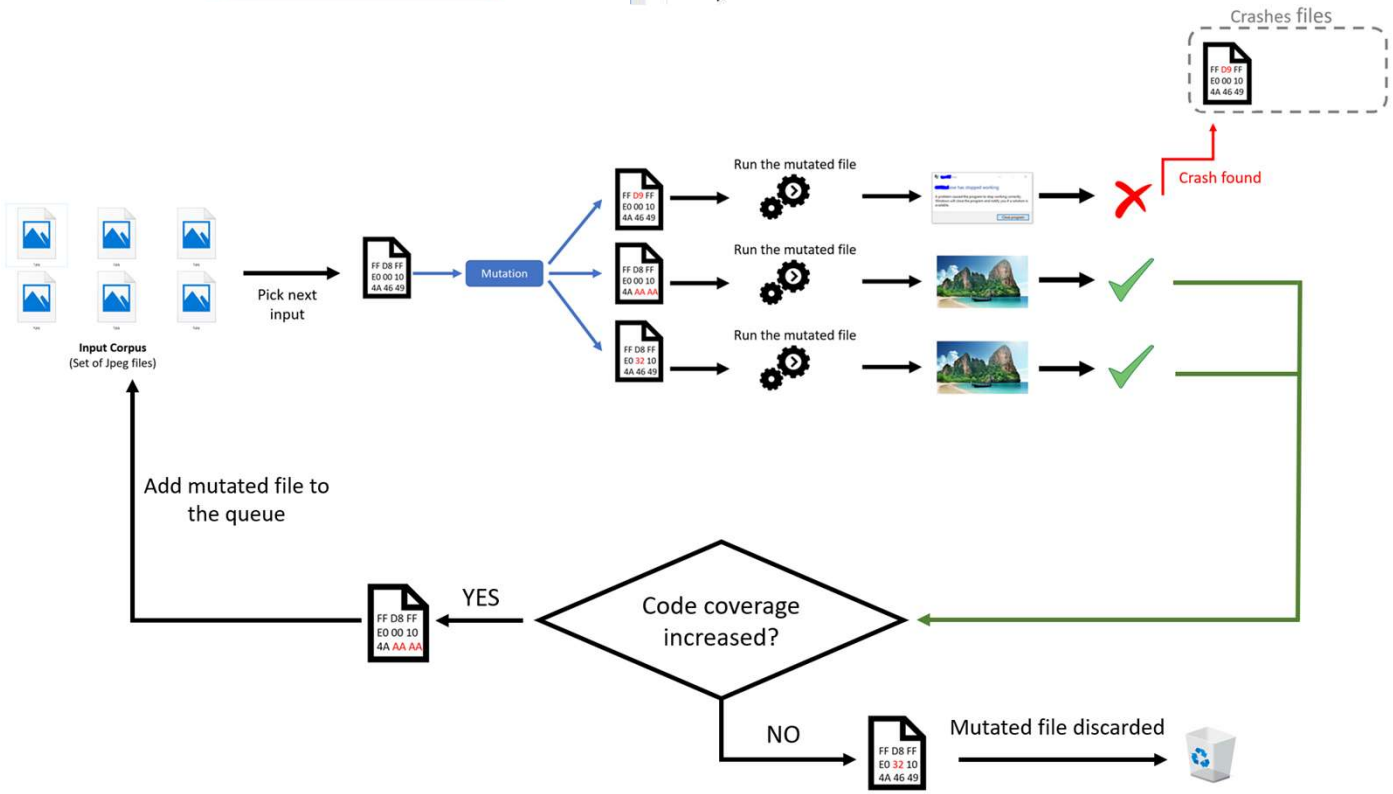
MISS

Ref : https://blog.mergify.com/code-coverage-everything-you-need-to-know/

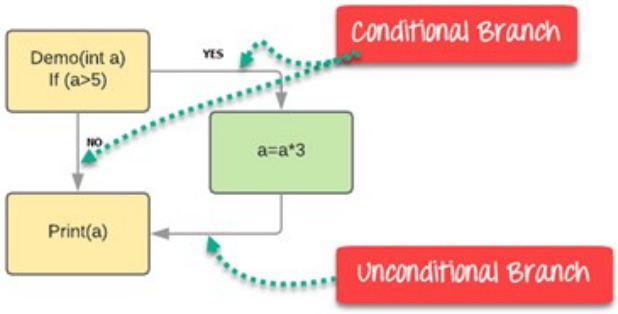



```
public boolean addAll(int index, Collection c) {
    if(c.isEmpty()) {
        return false;
    } else if(_size == index || _size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            pred = insertListable(pred,
        }
        return true;
    }
}
```

1 of 2 branches missed.
Press 'F2' for focus

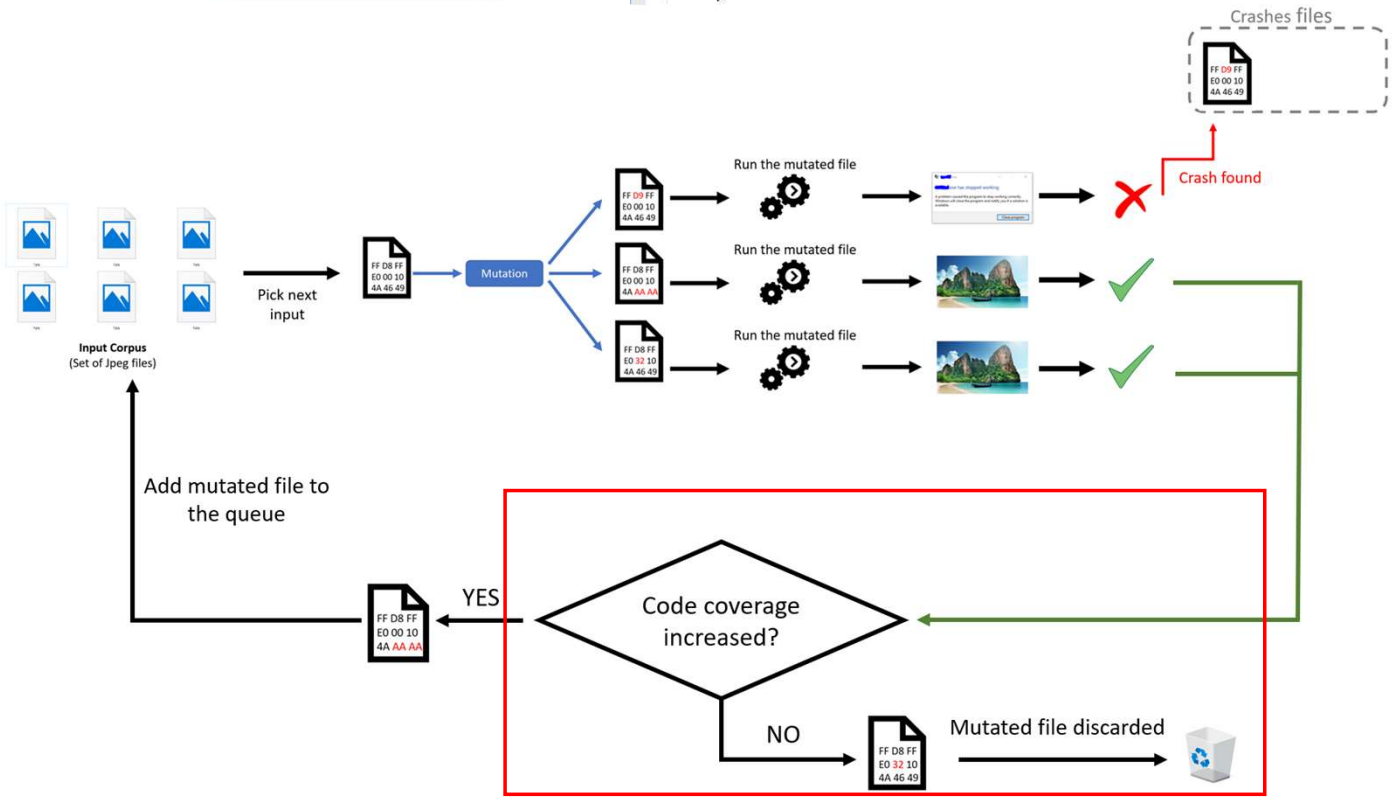


Ref : missing...




```
public boolean addAll(int index, Collection c) {
    if(c.isEmpty()) {
        return false;
    } else if(_size == index || _size == 0) {
        return addAll(c);
    } else {
        Listable succ = getListableAt(index);
        Listable pred = (null == succ) ? null : succ.prev();
        Iterator it = c.iterator();
        while(it.hasNext()) {
            pred = insertListable(pred,
        }
        return true;
    }
}
```

1 of 2 branches missed.
Press 'F2' for focus



Programs with a high code coverage percentage are less likely to have undetected bugs

How to check Code Coverage?

Black box : Binary Only

Grey box : hybrid (with source)

White box : compile time (with full source)

How to check Code Coverage?

Black box : Binary Only

Grey box : hybrid (with source)

White box : compile time (with full source)

How to check Code Coverage?

Black box : Binary Only

Grey box : hybrid (with source)

White box : compile time (with full source) → GCC's gcov

LLVM's SanitizerCoverage

LLVM's Source-based Code Coverage

SanitizerCoverage VS Source-based Code Coverage

SanitizerCoverage

상대적으로 간단&가벼운 측정 방식
Output 도 디테일하지 않음

컴파일 시 fsanitize 옵션으로 활성화

Simple하다는 장점이 있음

Source-based Code Coverage

라인, 함수, 분기 수준의 정교한 커버리지 제공

시각화 등 외부 기능 지원

operates on AST and preprocessor

<test.c>

```c
#include <stdio.h>


// 점수 계산 함수
char calculate_grade(int score) {
    if (score < 0 || score > 100) {
        printf("Error: 점수는 0~100 사이여야 합니다.₩n");
        return 'E'; // Error
    }

    if (score >= 90)      return 'A';
    else if (score >= 80) return 'B';
    else if (score >= 70) return 'C';
    else if (score >= 60) return 'D';
    else                  return 'F';
}

// 메인 함수
int main() {
    int score = 0;
    printf("score? : ");
    scanf("%d", &score);

    char grade = calculate_grade(score);
    printf("score : %d, grade : %c₩n", score, grade);

    return 0;
}
```

```
whoami@whoami:~/jeon_cov/test$ ./test
score? : 34
score : 34, grade : F
```

SanitizerCoverage

Tracing PCs with Guards

clang **-fsanitize-coverage=trace-pc-guard** -o test test.c

```
void __sanitizer_cov_trace_pc_guard_init(uint32_t* start, uint32_t* stop) {
            static uint64_t N;  // Counter for the guards.
            if (start == stop || *start) return;  // Initialize only once.
            printf("INIT: %p %p₩n", start, stop);
            for (uint32_t* x = start; x < stop; x++)
                        *x = ++N;  // Guards should start from 1.
}

void __sanitizer_cov_trace_pc_guard(uint32_t* guard) {
            if (!*guard) return;  // Duplicate the guard check.
            void* PC = __builtin_return_address(0);
            char PcDescr[1024];
            __sanitizer_symbolize_pc(PC, "%p %F %L", PcDescr, sizeof(PcDescr));
            printf("guard: %p %x PC %s₩n", guard, *guard, PcDescr);
}
```

SanitizerCoverage

Tracing PCs with Guards

-fsanitize-coverage=trace-pc-guard

컴파일 시 모든 **edge**에

__sanitizer_cov_trace_pc_guard(&guard_variable) 를 삽입한다.

=trace-pc,indirect-calls 옵션이 추가로 있을 경우,

모든 **간접 호출**에 __sanitizer_cov_trace_pc_indirect(void *callee) 가 삽입된다.

- `edge` (default): edges are instrumented (see below).
- `bb`: basic blocks are instrumented.
- `func`: only the entry block of every function will be instrumented.

Use these flags together with `trace-pc-guard` or `trace-pc`, like this: `-fsanitize-coverage=func,trace-pc-guard`.

Ref : https://clang.llvm.org/docs/SanitizerCoverage.html#edge-coverage

SanitizerCoverage

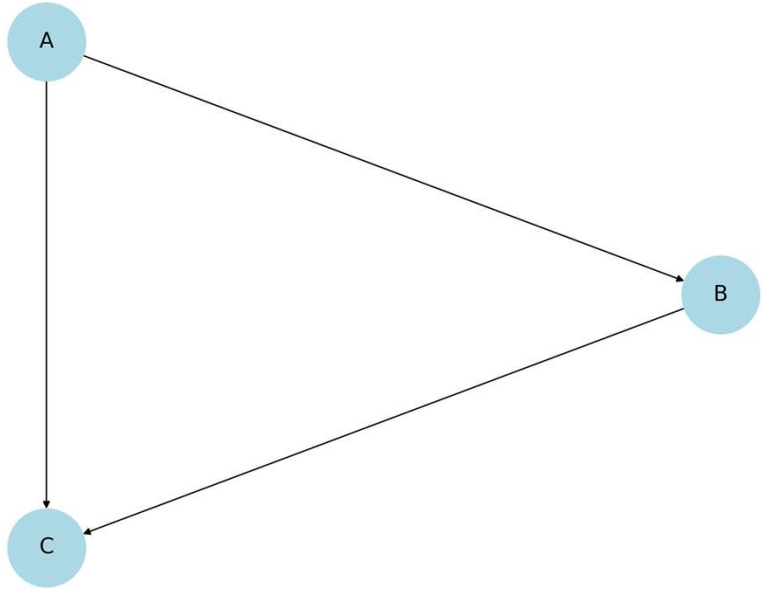
Tracing PCs with Guards

edge : block간의 연결 지점

```
void foo(int* a) {
    if (a)
        *a = 0;
}
```

block
- **A**: if (a) 조건 평가
- **B**: *a = 0 실행
- **C**: 조건이 거짓일 때 실행되는 경로

edge
- A->B
- B->C
- A->C



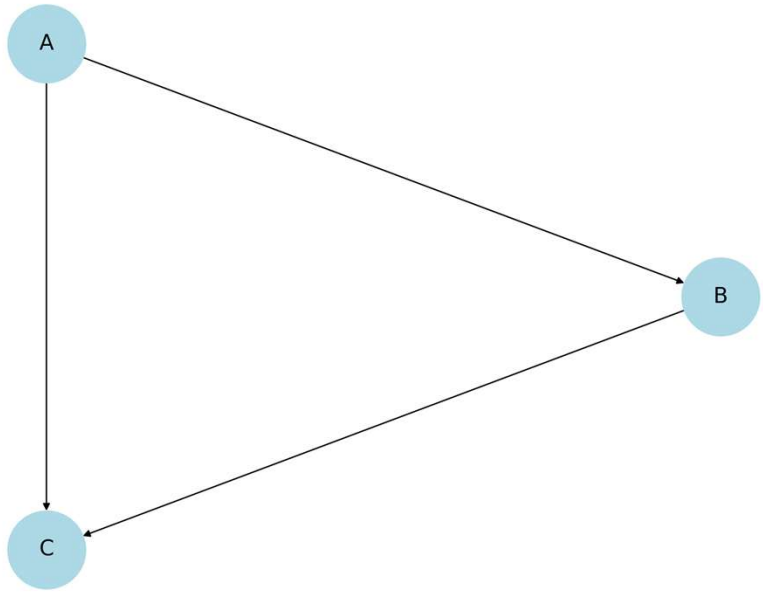
SanitizerCoverage

Tracing PCs with Guards

block
- **A**: if (a) 조건 평가
- **B**: *a = 0 실행
- **C**: 조건이 거짓일 때 실행되는 경로

edge
- A->B
- B->C
- **A->C**

A에서 C로 가는 경로가 실행되었는지 알 수 없음
이를 **Critical Edge** 라고 함



SanitizerCoverage

Tracing PCs with Guards

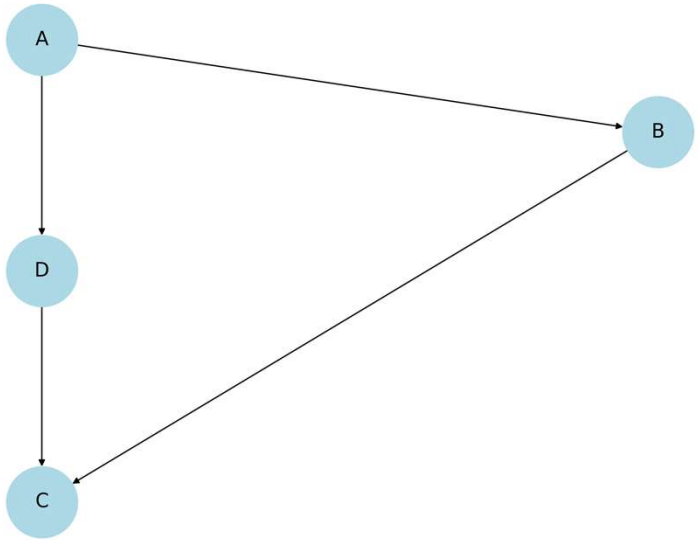
block

- **A**: if (a) 조건 평가
- **B**: *a = 0 실행
- **C**: 조건이 거짓일 때 실행되는 경로
- **D**: dummy block -> critical edge를 측정하기 위함

edge

- A->B
- B->C
- **A->C**

A에서 C로 가는 경로가 실행되었는지 알 수 없음
이를 **Critical Edge** 라고 함



SanitizerCoverage

Tracing PCs with Guards

```
whoami@whoami:~/jeon_cov/test$ ./test
INIT: 0x5eb9a7a85b80 0x5eb9a7a85ba4
guard: 0x5eb9a7a85ba0 9 PC 0x5eb9a7a6df83 in main (/home/whoami/jeon_cov/test/test+0x2cf83) (BuildId: f525844384651a8deeaf32249cd368a9c24eee87)
score? : 70
guard: 0x5eb9a7a85b80 1 PC 0x5eb9a7a6de66 in calculate_grade (/home/whoami/jeon_cov/test/test+0x2ce66) (BuildId: f525844384651a8deeaf32249cd368a9c24eee87)
guard: 0x5eb9a7a85b94 6 PC 0x5eb9a7a6df1c in calculate_grade (/home/whoami/jeon_cov/test/test+0x2cf1c) (BuildId: f525844384651a8deeaf32249cd368a9c24eee87)
score : 70, grade : C
whoami@whoami:~/jeon_cov/test$
```

clang **-fsanitize-coverage=trace-pc-guard** -o test test.c

SanitizerCoverage

Tracing PCs with Guards

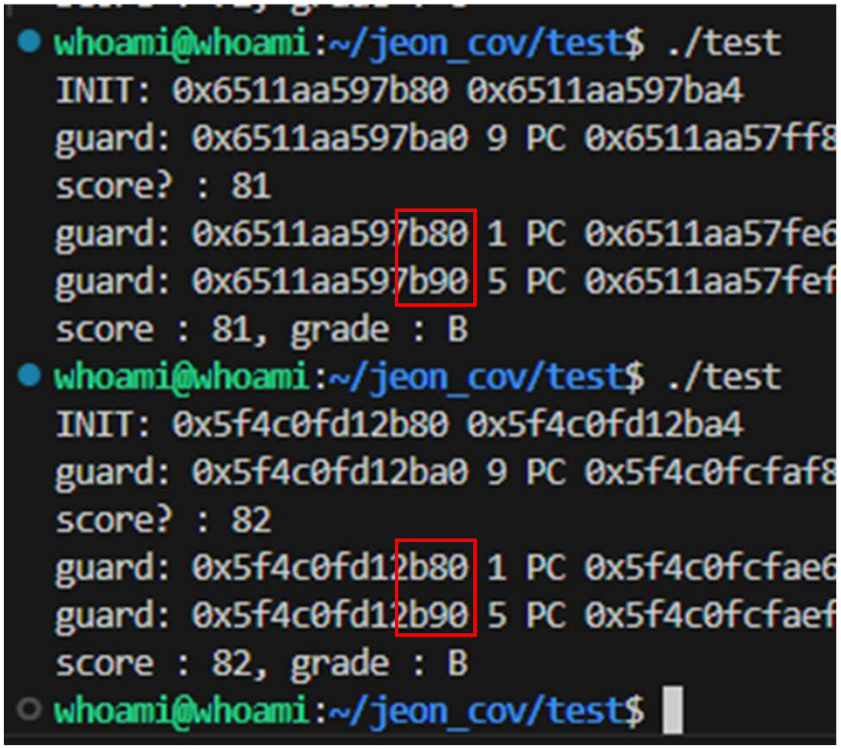
```
whoami@whoami:~/jeon_cov/test$ ./test
INIT: 0x5eb9a7a85b80 0x5eb9a7a85ba4
guard: 0x5eb9a7a85ba0 9 PC 0x5eb9a7a6df83 in main (/home/whoa
score? : 70
guard: 0x5eb9a7a85b80 1 PC 0x5eb9a7a6de66 in calculate_grade
guard: 0x5eb9a7a85b94 6 PC 0x5eb9a7a6df1c in calculate_grade
score : 70, grade : C
whoami@whoami:~/jeon_cov/test$ ./test
INIT: 0x5d138a6dab80 0x5d138a6daba4
guard: 0x5d138a6daba0 9 PC 0x5d138a6c2f83 in main (/home/whoa
score? : 71
guard: 0x5d138a6dab80 1 PC 0x5d138a6c2e66 in calculate_grade
guard: 0x5d138a6dab94 6 PC 0x5d138a6c2f1c in calculate_grade
score : 71, grade : C
whoami@whoami:~/jeon_cov/test$ ./test
INIT: 0x5a26ecb7ab80 0x5a26ecb7aba4
guard: 0x5a26ecb7aba0 9 PC 0x5a26ecb62f83 in main (/home/whoa
score? : 72
guard: 0x5a26ecb7ab80 1 PC 0x5a26ecb62e66 in calculate_grade
guard: 0x5a26ecb7ab94 6 PC 0x5a26ecb62f1c in calculate_grade
score : 72, grade : C
```

ASLR로 인해 주소의 상위 비트는
무작위화되지만, 하위 비트(마지막 3자리)는
메모리 정렬 규칙에 따라 일정하게 유지

Grade : C 분기에서는 0x~b94 가 hit 되었음

SanitizerCoverage

Tracing PCs with Guards



ASLR로 인해 주소의 상위 비트는
무작위화되지만, 하위 비트(마지막 3자리)는
메모리 정렬 규칙에 따라 일정하게 유지

Grade : B 분기에서는 0x~b90 가 hit 되었음

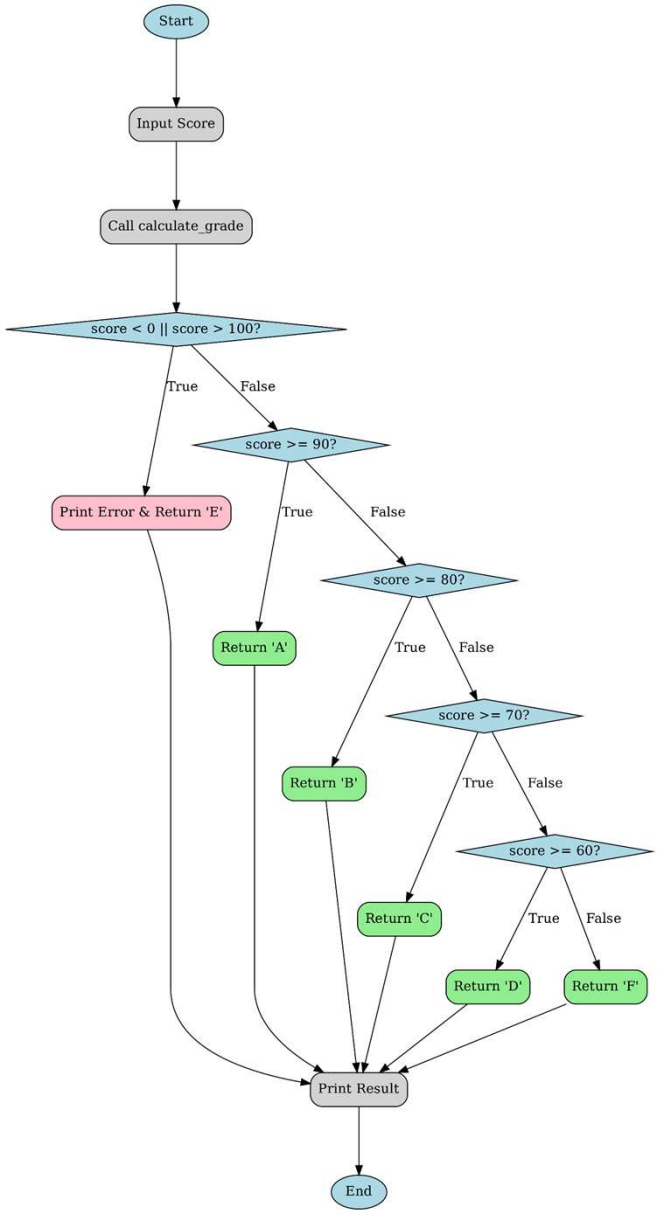
예상 엣지

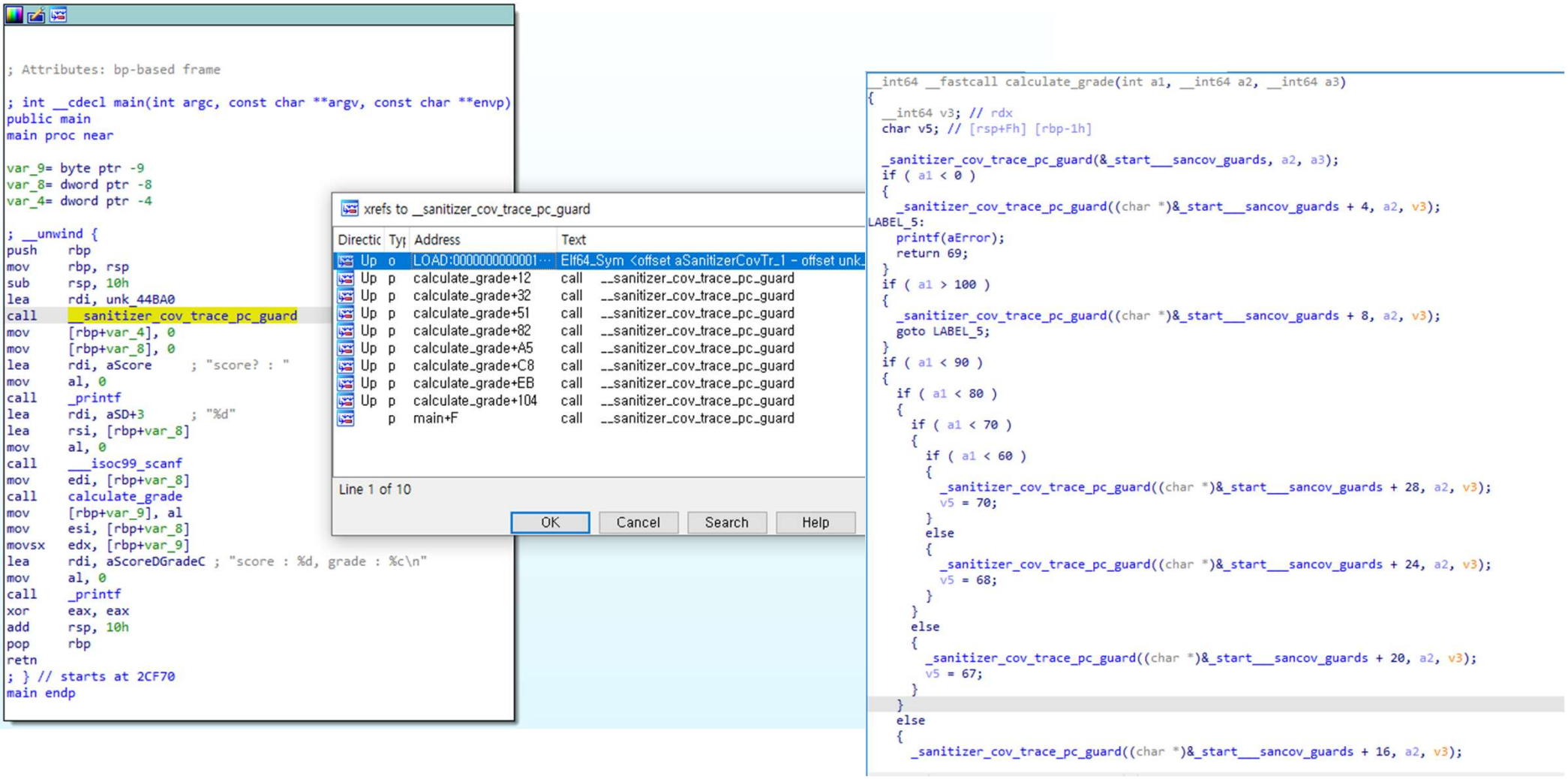
(1) calculate_grade

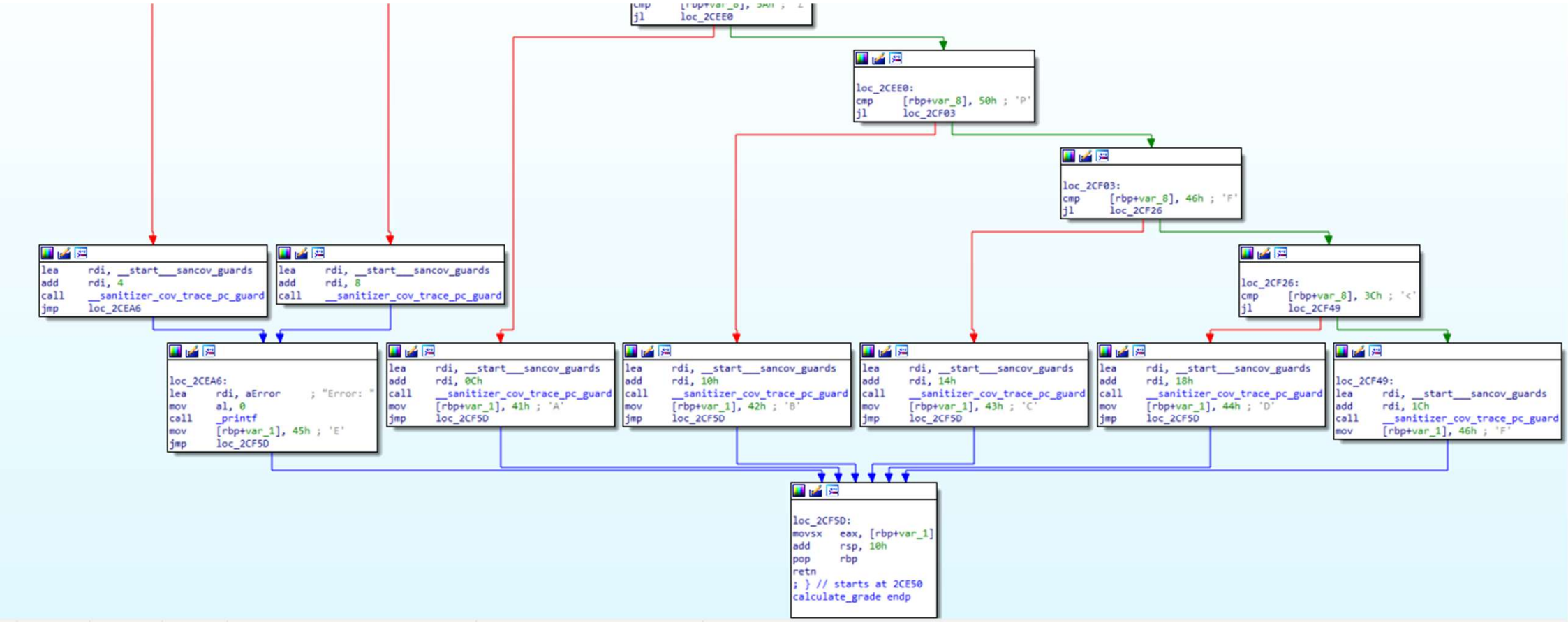
- **Edge 1**: 함수 시작 → if (score < 0 || score > 100)
- **Edge 2**: if (score < 0 || score > 100) 참(True) → return 'E'
- **Edge 3**: if (score < 0 || score > 100) 거짓(False) → if (score >= 90)
- **Edge 4**: if (score >= 90) 참(True) → return 'A'
- **Edge 5**: if (score >= 90) 거짓(False) → if (score >= 80)
- **Edge 6**: if (score >= 80) 참(True) → return 'B'
- **Edge 7**: if (score >= 80) 거짓(False) → if (score >= 70)
- **Edge 8**: if (score >= 70) 참(True) → return 'C'
- **Edge 9**: if (score >= 70) 거짓(False) → if (score >= 60)
- **Edge 10**: if (score >= 60) 참(True) → return 'D'
- **Edge 11**: if (score >= 60) 거짓(False) → return 'F'

(2) main

- **Edge 12**: main 함수 시작 → scanf
- **Edge 13**: scanf → calculate_grade 호출
- **Edge 14**: calculate_grade → 결과 출력
- **Edge 15**: 결과 출력 → return 0




; Attributes: bp-based frame
; int __cdecl main(int argc, const char **argv, const char **envp)
public main
main proc near
var_9= byte ptr -9
var_8= dword ptr -8
var_4= dword ptr -4
; __unwind {
push rbp
mov rbp, rsp
sub rsp, 10h
lea rdi, unk_44BA0
call __sanitizer_cov_trace_pc_guard
mov [rbp+var_4], 0
mov [rbp+var_8], 0
lea rdi, aScore ; "score? : "
mov al, 0
call _printf
lea rdi, aSD+3 ; "%d"
lea rsi, [rbp+var_8]
mov al, 0
call ___isoc99_scanf
mov edi, [rbp+var_8]
call calculate_grade
mov [rbp+var_9], al
mov esi, [rbp+var_8]
movsx edx, [rbp+var_9]
lea rdi, aScoreDGradeC ; "score : %d, grade : %c\n"
mov al, 0
call _printf
xor eax, eax
add rsp, 10h
pop rbp
retn
; } // starts at 2CF70
main endp
xrefs to __sanitizer_cov_trace_pc_guard
Directic Typ Address Text
Up o LOAD:0000000000001··· Elf64_Sym <offset aSanitizerCovTr_1 - offset unk_
Up p calculate_grade+12 call __sanitizer_cov_trace_pc_guard
Up p calculate_grade+32 call __sanitizer_cov_trace_pc_guard
Up p calculate_grade+51 call __sanitizer_cov_trace_pc_guard
Up p calculate_grade+82 call __sanitizer_cov_trace_pc_guard
Up p calculate_grade+A5 call __sanitizer_cov_trace_pc_guard
Up p calculate_grade+C8 call __sanitizer_cov_trace_pc_guard
Up p calculate_grade+EB call __sanitizer_cov_trace_pc_guard
Up p calculate_grade+104 call __sanitizer_cov_trace_pc_guard
p main+F call __sanitizer_cov_trace_pc_guard
Line 1 of 10
OK
Cancel
Search
Help
__int64 __fastcall calculate_grade(int a1, __int64 a2, __int64 a3)
{
__int64 v3; // rdx
char v5; // [rsp+Fh] [rbp-1h]
_sanitizer_cov_trace_pc_guard(&_start___sancov_guards, a2, a3);
if (a1 < 0)
{
_sanitizer_cov_trace_pc_guard((char *)&_start___sancov_guards + 4, a2, v3);
LABEL_5:
printf(aError);
return 69;
}
if (a1 > 100)
{
_sanitizer_cov_trace_pc_guard((char *)&_start___sancov_guards + 8, a2, v3);
goto LABEL_5;
}
if (a1 < 90)
{
if (a1 < 80)
{
if (a1 < 70)
{
if (a1 < 60)
{
_sanitizer_cov_trace_pc_guard((char *)&_start___sancov_guards + 28, a2, v3);
v5 = 70;
}
else
{
_sanitizer_cov_trace_pc_guard((char *)&_start___sancov_guards + 24, a2, v3);
v5 = 68;
}
}
else
{
_sanitizer_cov_trace_pc_guard((char *)&_start___sancov_guards + 20, a2, v3);
v5 = 67;
}
}
else
{
_sanitizer_cov_trace_pc_guard((char *)&_start___sancov_guards + 16, a2, v3);


jl loc_2CEE0
loc_2CEE0:
cmp [rbp+var_8], 50h ; 'P'
jl loc_2CF03
loc_2CF03:
cmp [rbp+var_8], 46h ; 'F'
jl loc_2CF26
loc_2CF26:
cmp [rbp+var_8], 3Ch ; '<'
jl loc_2CF49
lea rdi, __start___sancov_guards
add rdi, 4
call __sanitizer_cov_trace_pc_guard
jmp loc_2CEA6
lea rdi, __start___sancov_guards
add rdi, 8
call __sanitizer_cov_trace_pc_guard
loc_2CEA6:
lea rdi, aError ; "Error: "
mov al, 0
call _printf
mov [rbp+var_1], 45h ; 'E'
jmp loc_2CF5D
lea rdi, __start___sancov_guards
add rdi, 0Ch
call __sanitizer_cov_trace_pc_guard
mov [rbp+var_1], 41h ; 'A'
jmp loc_2CF5D
lea rdi, __start___sancov_guards
add rdi, 10h
call __sanitizer_cov_trace_pc_guard
mov [rbp+var_1], 42h ; 'B'
jmp loc_2CF5D
lea rdi, __start___sancov_guards
add rdi, 14h
call __sanitizer_cov_trace_pc_guard
mov [rbp+var_1], 43h ; 'C'
jmp loc_2CF5D
lea rdi, __start___sancov_guards
add rdi, 18h
call __sanitizer_cov_trace_pc_guard
mov [rbp+var_1], 44h ; 'D'
jmp loc_2CF5D
loc_2CF49:
lea rdi, __start___sancov_guards
add rdi, 1Ch
call __sanitizer_cov_trace_pc_guard
mov [rbp+var_1], 46h ; 'F'
loc_2CF5D:
movsx eax, [rbp+var_1]
add rsp, 10h
pop rbp
retn
; } // starts at 2CE50
calculate_grade endp

실제 엣지

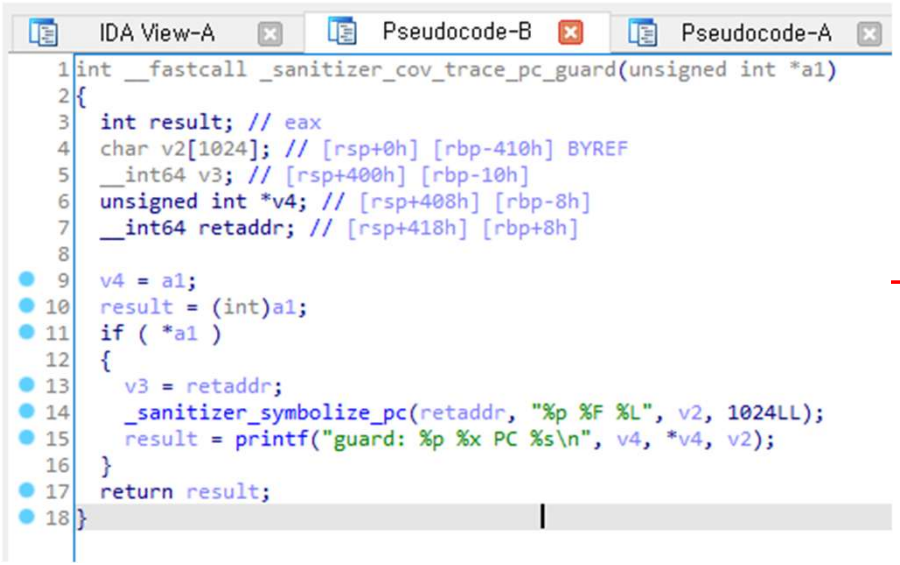
(1) calculate_grade

- ~~**Edge 1**: 함수 시작 → if (score < 0 || score > 100)~~
- **Edge 2**: if (score < 0 || score > 100) 참(True) → return 'E'
- ~~**Edge 3**: if (score < 0 || score > 100) 거짓(False) → if (score >= 90)~~
- **Edge 4**: if (score >= 90) 참(True) → return 'A'
- ~~**Edge 5**: if (score >= 90) 거짓(False) → if (score >= 80)~~
- **Edge 6**: if (score >= 80) 참(True) → return 'B'
- ~~**Edge 7**: if (score >= 80) 거짓(False) → if (score >= 70)~~
- **Edge 8**: if (score >= 70) 참(True) → return 'C'
- ~~**Edge 9**: if (score >= 70) 거짓(False) → if (score >= 60)~~
- **Edge 10**: if (score >= 60) 참(True) → return 'D'
- **Edge 11**: if (score >= 60) 거짓(False) → return ' F'
- **New : score < 0**
- **New : score > 100**

(2) main

- **Edge 12**: main 함수 시작 → scanf
- **Edge 13**: scanf → calculate_grade 호출
- **Edge 14**: calculate_grade → 결과 출력
- **Edge 15**: 결과 출력 → return 0

```
if ( a1 < 0 )
{
  _sanitizer_cov_trace_pc_guard((char *)&_start___sancov_guards + 4, a2, v3);
LABEL_5:
  printf(aError);
  return 69;
}
if ( a1 > 100 )
{
  _sanitizer_cov_trace_pc_guard((char *)&_start___sancov_guards + 8, a2, v3);
  goto LABEL_5;
}
```

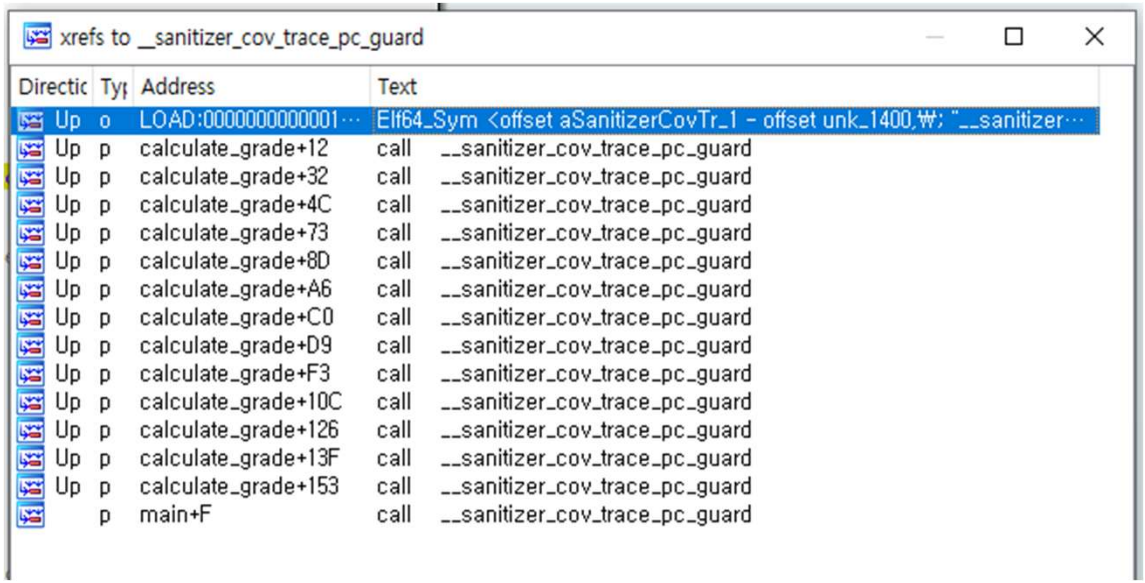



```
void __sanitizer_cov_trace_pc_guard(uint32_t* guard) {
            if (!*guard) return;  // Duplicate the guard check.
            void* PC = __builtin_return_address(0);
            char PcDescr[1024];
            __sanitizer_symbolize_pc(PC, "%p %F %L", PcDescr, sizeof(PcDescr));
            printf("guard: %p %x PC %s₩n", guard, *guard, PcDescr);
}
```

User code

-fsanitize-coverage=**bb,no-prune,trace-pc-guard**

```
whoami@whoami:~/jeon_cov/test$ ./test
INIT: 0x615c5a7fdb80 0x615c5a7fdbb8
guard: 0x615c5a7fdbb4 e PC 0x615c5a7e5fd3 in main (/home/whoami/jeon_cov/test/test+0x2cfd3) (BuildId: c03695bfe4d5cbf846ed749cb807d76640f937c2)
score? : 92
guard: 0x615c5a7fdb80 1 PC 0x615c5a7e5e66 in calculate_grade (/home/whoami/jeon_cov/test/test+0x2ce66) (BuildId: c03695bfe4d5cbf846ed749cb807d76640f937c2)
guard: 0x615c5a7fdb84 2 PC 0x615c5a7e5e86 in calculate_grade (/home/whoami/jeon_cov/test/test+0x2ce86) (BuildId: c03695bfe4d5cbf846ed749cb807d76640f937c2)
guard: 0x615c5a7fdb8c 4 PC 0x615c5a7e5ec7 in calculate_grade (/home/whoami/jeon_cov/test/test+0x2cec7) (BuildId: c03695bfe4d5cbf846ed749cb807d76640f937c2)
guard: 0x615c5a7fdb90 5 PC 0x615c5a7e5ee1 in calculate_grade (/home/whoami/jeon_cov/test/test+0x2cee1) (BuildId: c03695bfe4d5cbf846ed749cb807d76640f937c2)
guard: 0x615c5a7fdbb0 d PC 0x615c5a7e5fa7 in calculate_grade (/home/whoami/jeon_cov/test/test+0x2cfa7) (BuildId: c03695bfe4d5cbf846ed749cb807d76640f937c2)
score : 92, grade : A
```



- **edge (기본값)**
  - 코드의 **에지(Edge)**, 즉 블록 간의 연결(분기점)을 계측
  - 코드의 흐름이 어떤 에지를 통과했는지 추적할 수 있다.
- **bb (Basic Block)**
  - 코드의 **기본 블록(Basic Block)**을 계측
  - 기본 블록은 분기나 함수 호출 없이 순차적으로 실행되는 명령어 집합
- **func**
  - **함수의 진입 지점**(Entry Block)만 계측
  - 함수가 호출되었는지 여부만 확인

기본적으로, edge 또는 bb 계측을 사용하면 일부 중복되는 블록/에지는 계측에서 제외
no-prune 옵션을 추가하여 모든 에지/블록을 계측하도록 설정 (예상한 위치와 유사하게 적용)

SanitizerCoverage

Inline 8bit-counters

clang **-fsanitize-coverage=inline-8bit-counters** -o test test.c

```
// 전역 변수로 카운터 배열 저장
static char* counter_start = NULL;
static char* counter_end = NULL;

// 카운터 초기화 함수
void __sanitizer_cov_8bit_counters_init(char* start, char* end) {
    printf("Counters initialized: start=%p, end=%p₩n", (void*)start, (void*)end);

    // 카운터 배열 저장
    counter_start = start;
    counter_end = end;

    // 카운터 초기화 (0으로 설정)
    for (char* ptr = start; ptr < end; ++ptr) {
        *ptr = 0;
    }
}
```

SanitizerCoverage

Inline 8bit-counters

clang **-fsanitize-coverage=inline-8bit-counters** -o test test.c

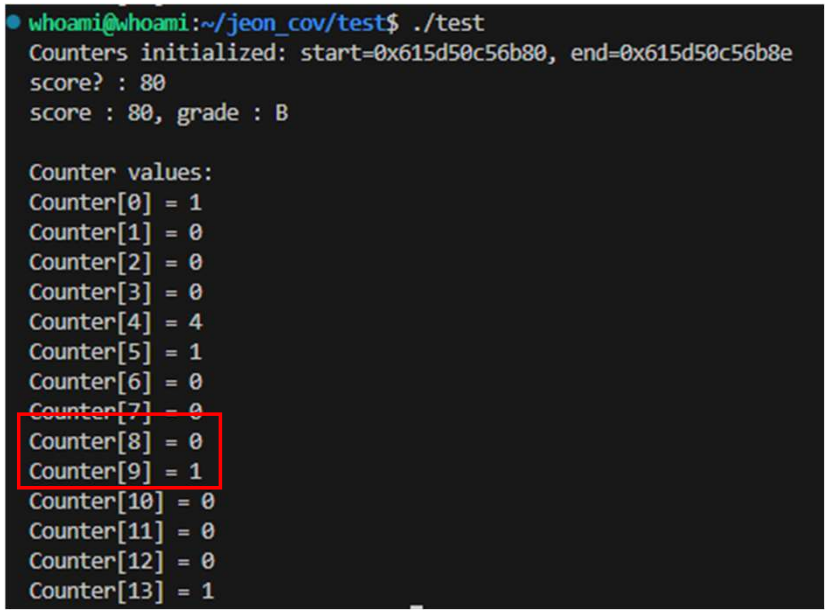
```
void print_counters_at_exit() {
    if (counter_start == NULL || counter_end == NULL) {
        printf("No counters to display.₩n");
        return;
    }

    printf("₩nCounter values:₩n");
    size_t counter_index = 0;
    for (char* ptr = counter_start; ptr < counter_end; ++ptr, ++counter_index) {
        printf("Counter[%zu] = %d₩n", counter_index, (unsigned char)*ptr);
    }
}
```

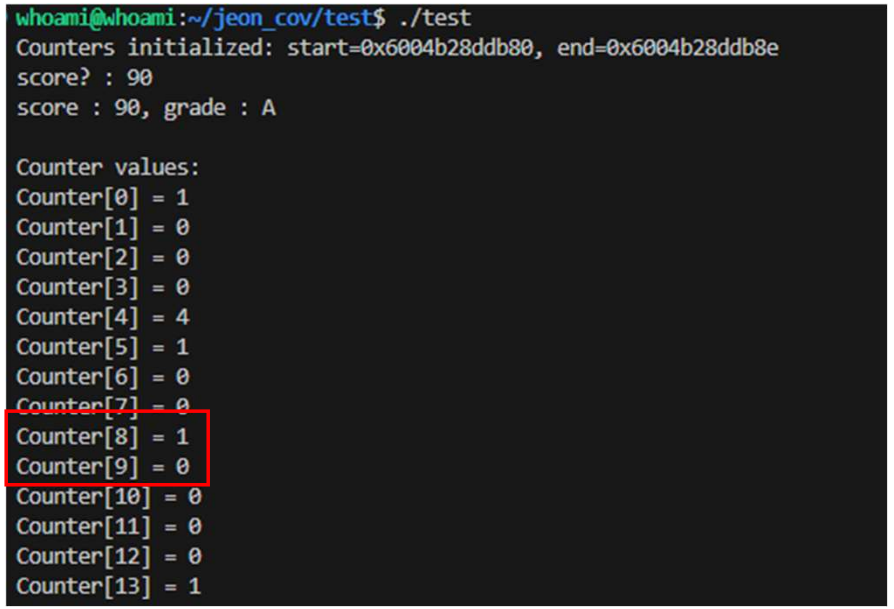
SanitizerCoverage

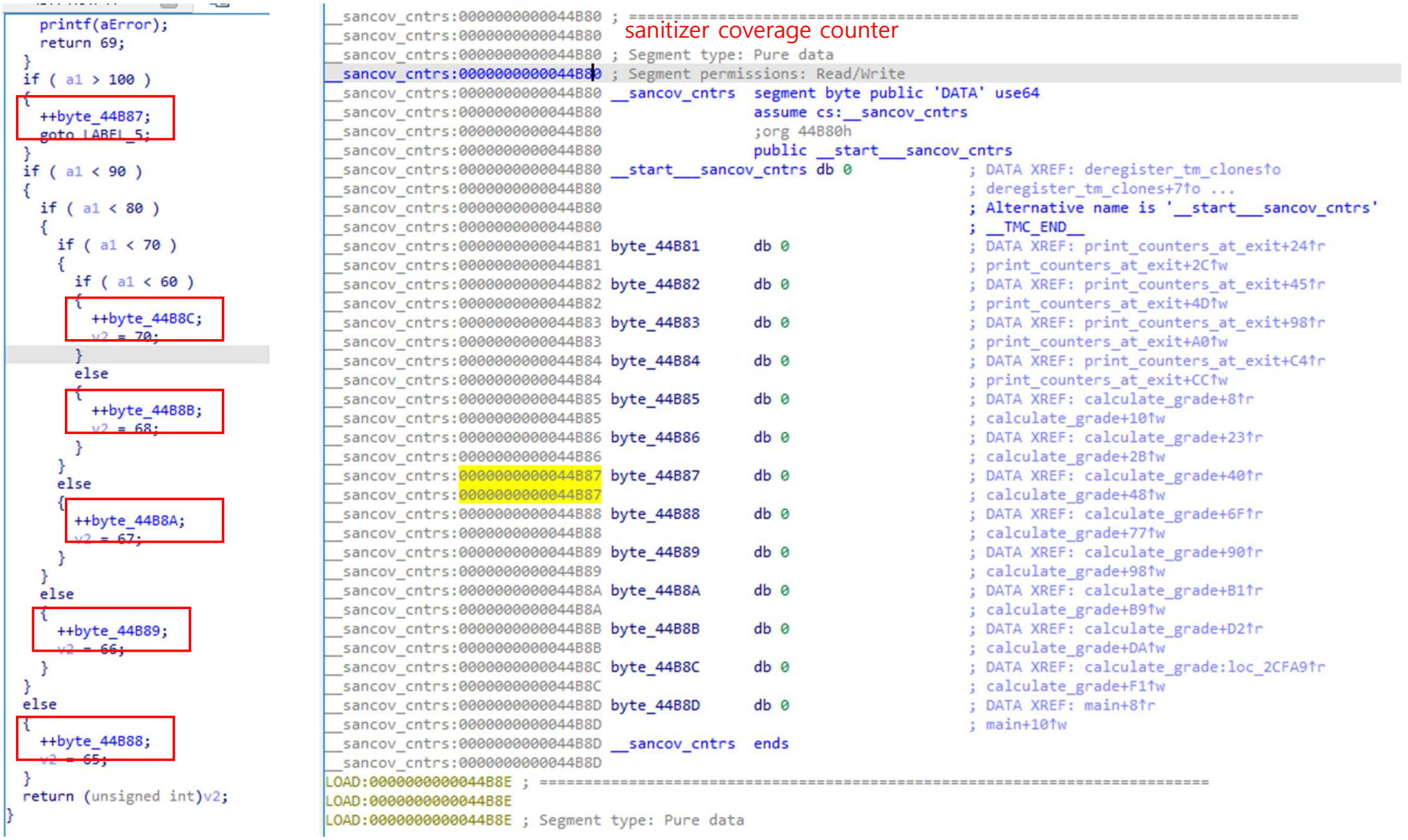
Inline 8bit-counters

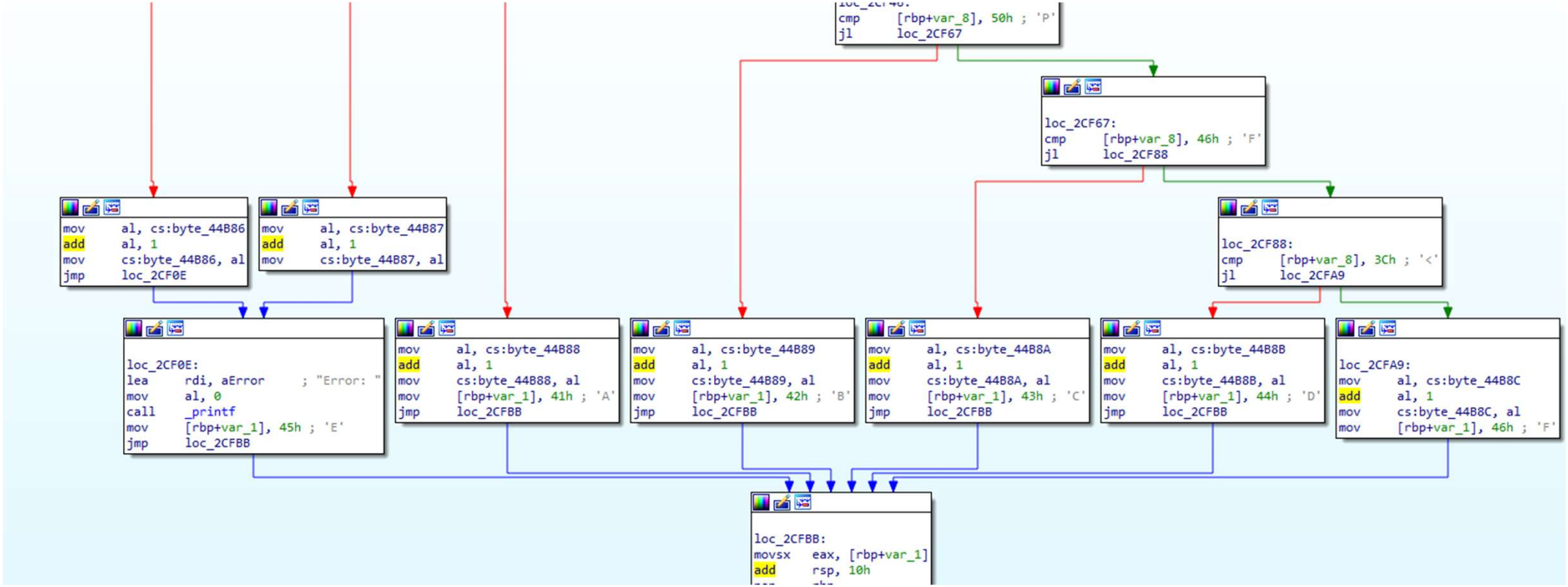
Input : 80



Input : 90




printf(aError);
return 69;
}
if (a1 > 100)
{
++byte_44B87;
goto LABEL_5;
}
if (a1 < 90)
{
if (a1 < 80)
{
if (a1 < 70)
{
if (a1 < 60)
{
++byte_44B8C;
v2 = 70;
}
else
{
++byte_44B8B;
v2 = 68;
}
}
else
{
++byte_44B8A;
v2 = 67;
}
}
else
{
++byte_44B89;
v2 = 66;
}
}
else
{
++byte_44B88;
v2 = 65;
}
return (unsigned int)v2;
}
sanitizer coverage counter
__sancov_cntrs:0000000000044B80 ; ===
__sancov_cntrs:0000000000044B80
__sancov_cntrs:0000000000044B80 ; Segment type: Pure data
__sancov_cntrs:0000000000044B80 ; Segment permissions: Read/Write
__sancov_cntrs:0000000000044B80 __sancov_cntrs segment byte public 'DATA' use64
__sancov_cntrs:0000000000044B80 assume cs:__sancov_cntrs
__sancov_cntrs:0000000000044B80 ;org 44B80h
__sancov_cntrs:0000000000044B80 public __start___sancov_cntrs
__sancov_cntrs:0000000000044B80 __start___sancov_cntrs db 0 ; DATA XREF: deregister_tm_clones↑o
__sancov_cntrs:0000000000044B80 ; deregister_tm_clones+7↑o ...
__sancov_cntrs:0000000000044B80 ; Alternative name is '__start___sancov_cntrs'
__sancov_cntrs:0000000000044B80 ; __TMC_END__
__sancov_cntrs:0000000000044B81 byte_44B81 db 0 ; DATA XREF: print_counters_at_exit+24↑r
__sancov_cntrs:0000000000044B81 ; print_counters_at_exit+2C↑w
__sancov_cntrs:0000000000044B82 byte_44B82 db 0 ; DATA XREF: print_counters_at_exit+45↑r
__sancov_cntrs:0000000000044B82 ; print_counters_at_exit+4D↑w
__sancov_cntrs:0000000000044B83 byte_44B83 db 0 ; DATA XREF: print_counters_at_exit+98↑r
__sancov_cntrs:0000000000044B83 ; print_counters_at_exit+A0↑w
__sancov_cntrs:0000000000044B84 byte_44B84 db 0 ; DATA XREF: print_counters_at_exit+C4↑r
__sancov_cntrs:0000000000044B84 ; print_counters_at_exit+CC↑w
__sancov_cntrs:0000000000044B85 byte_44B85 db 0 ; DATA XREF: calculate_grade+8↑r
__sancov_cntrs:0000000000044B85 ; calculate_grade+10↑w
__sancov_cntrs:0000000000044B86 byte_44B86 db 0 ; DATA XREF: calculate_grade+23↑r
__sancov_cntrs:0000000000044B86 ; calculate_grade+2B↑w
__sancov_cntrs:0000000000044B87 byte_44B87 db 0 ; DATA XREF: calculate_grade+40↑r
__sancov_cntrs:0000000000044B87 ; calculate_grade+48↑w
__sancov_cntrs:0000000000044B88 byte_44B88 db 0 ; DATA XREF: calculate_grade+6F↑r
__sancov_cntrs:0000000000044B88 ; calculate_grade+77↑w
__sancov_cntrs:0000000000044B89 byte_44B89 db 0 ; DATA XREF: calculate_grade+90↑r
__sancov_cntrs:0000000000044B89 ; calculate_grade+98↑w
__sancov_cntrs:0000000000044B8A byte_44B8A db 0 ; DATA XREF: calculate_grade+B1↑r
__sancov_cntrs:0000000000044B8A ; calculate_grade+B9↑w
__sancov_cntrs:0000000000044B8B byte_44B8B db 0 ; DATA XREF: calculate_grade+D2↑r
__sancov_cntrs:0000000000044B8B ; calculate_grade+DA↑w
__sancov_cntrs:0000000000044B8C byte_44B8C db 0 ; DATA XREF: calculate_grade:loc_2CFA9↑r
__sancov_cntrs:0000000000044B8C ; calculate_grade+F1↑w
__sancov_cntrs:0000000000044B8D byte_44B8D db 0 ; DATA XREF: main+8↑r
__sancov_cntrs:0000000000044B8D ; main+10↑w
__sancov_cntrs:0000000000044B8D __sancov_cntrs ends
__sancov_cntrs:0000000000044B8D
LOAD:0000000000044B8E ; ===
LOAD:0000000000044B8E
LOAD:0000000000044B8E ; Segment type: Pure data



적용되는 위치에는 유의미한 변화가 없다

SanitizerCoverage

Inline bool-flag

clang **-fsanitize-coverage=inline-bool-flag** -o test test.c

```c
#include <stdbool.h> // C99 표준 bool 타입 사용

// 전역 변수로 플래그 배열 저장
static bool* flag_start = NULL;
static bool* flag_end = NULL;

// 플래그 초기화 함수
void __sanitizer_cov_bool_flag_init(bool* start, bool* end) {
    printf("Boolean flags initialized: start=%p, end=%p₩n", (void*)start, (void*)end);

    // 플래그 배열 저장
    flag_start = start;
    flag_end = end;

    // 플래그 초기화 (false로 설정)
    for (bool* ptr = start; ptr < end; ++ptr) {
        *ptr = false;
    }
}
```

SanitizerCoverage

Inline bool-flag

clang **-fsanitize-coverage=inline-bool-flag** -o test test.c

```
// 프로그램 종료 시 플래그 값을 출력하는 함수
void print_flags_at_exit() {
    if (flag_start == NULL || flag_end == NULL) {
        printf("No flags to display.₩n");
        return;
    }

    printf("₩nBoolean flag values:₩n");
    size_t flag_index = 0;
    for (bool* ptr = flag_start; ptr < flag_end; ++ptr, ++flag_index) {
        printf("Flag[%zu] = %s₩n", flag_index, *ptr ? "true" : "false");
    }
}
```

SanitizerCoverage

Inline bool-flag

```
whoami@whoami:~/jeon_cov/test$ ./test
Boolean flags initialized: start=0x5f9c4ca28b80, end=0x5f9c4ca28b8e
score? : 90
score : 90, grade : A

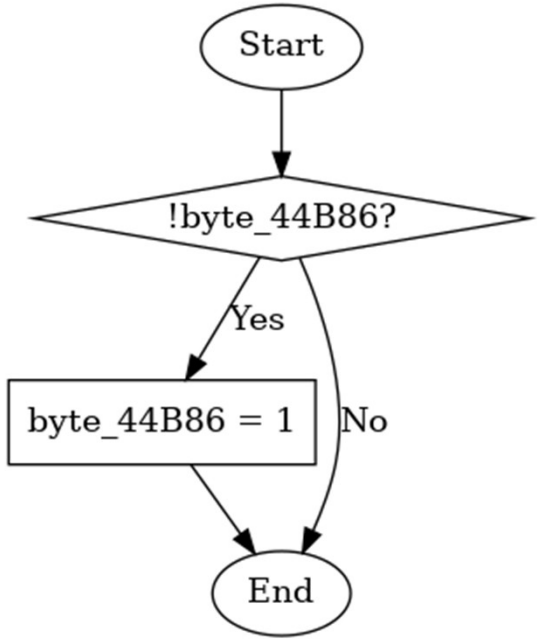
Boolean flag values:
Flag[0] = true
Flag[1] = false
Flag[2] = false
Flag[3] = false
Flag[4] = true
Flag[5] = true
Flag[6] = false
Flag[7] = false
Flag[8] = true
Flag[9] = false
Flag[10] = false
Flag[11] = false
Flag[12] = false
Flag[13] = true
```

```
whoami@whoami:~/jeon_cov/test$ ./test
Boolean flags initialized: start=0x618c4275bb80, end=0x618c4275bb8e
score? : 70
score : 70, grade : C

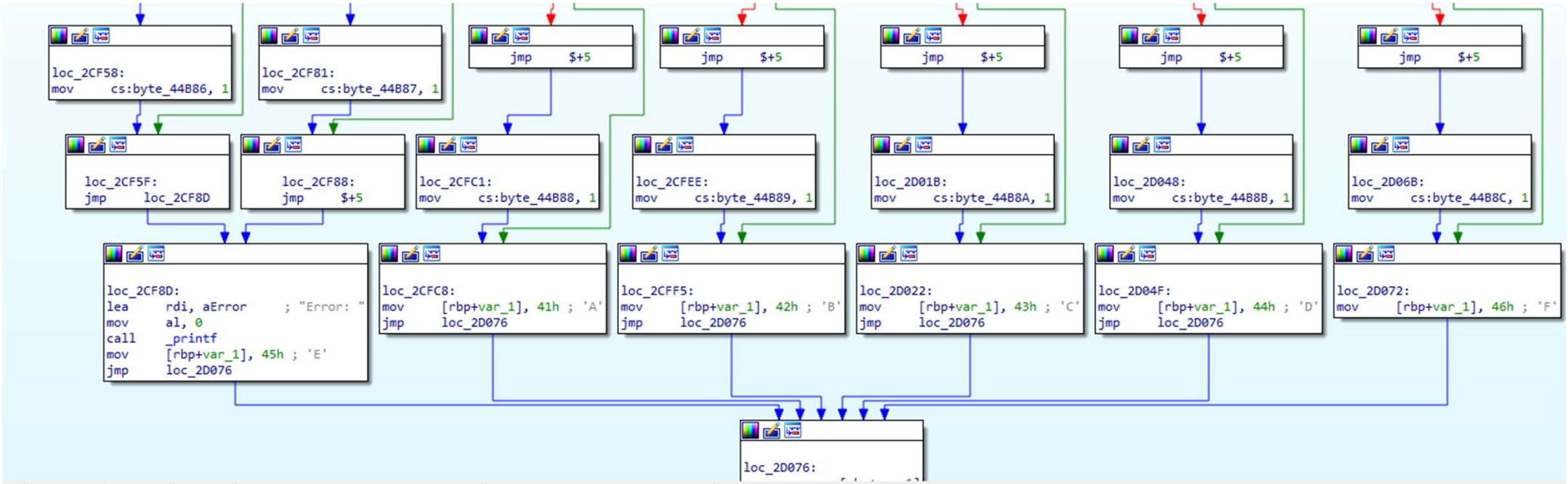
Boolean flag values:
Flag[0] = true
Flag[1] = false
Flag[2] = false
Flag[3] = false
Flag[4] = true
Flag[5] = true
Flag[6] = false
Flag[7] = false
Flag[8] = false
Flag[9] = false
Flag[10] = true
Flag[11] = false
Flag[12] = false
Flag[13] = true
```

```
if ( a1 < 0 )
{
  if ( !byte_44B86 )
    byte_44B86 = 1;
LABEL_10:
  printf(aError);
  return 69;
}
if ( a1 > 100 )
{
  if ( !byte_44B87 )
    byte_44B87 = 1;
  goto LABEL_10;
}
if ( a1 < 90 )
{
  if ( a1 < 80 )
  {
    if ( a1 < 70 )
    {
      if ( a1 < 60 )
      {
        if ( !byte_44B8C )
          byte_44B8C = 1;
        v2 = 70;
      }
      else
      {
        if ( !byte_44B8B )
          byte_44B8B = 1;
        v2 = 68;
      }
    }
    else
    {
      if ( !byte_44B8A )
        byte_44B8A = 1;
      v2 = 67;
    }
  }
  else
  {
    if ( !byte_44B89 )
      byte_44B89 = 1;
```

```
__sancov_bools:0000000000044B80 ; ======================
__sancov_bools:0000000000044B80
__sancov_bools:0000000000044B80 ; Segment type: Pure dat
__sancov_bools:0000000000044B80 ; Segment permissions: R
__sancov_bools:0000000000044B80 __sancov_bools  segment
__sancov_bools:0000000000044B80                 assume c
__sancov_bools:0000000000044B80                 ;org 44B
__sancov_bools:0000000000044B80                 public _
__sancov_bools:0000000000044B80 __start___sancov_bools d
__sancov_bools:0000000000044B80
__sancov_bools:0000000000044B80
__sancov_bools:0000000000044B80
__sancov_bools:0000000000044B81 byte_44B81      db 0
__sancov_bools:0000000000044B81
__sancov_bools:0000000000044B82 byte_44B82      db 0
__sancov_bools:0000000000044B82
__sancov_bools:0000000000044B83 byte_44B83      db 0
__sancov_bools:0000000000044B83
__sancov_bools:0000000000044B84 byte_44B84      db 0
__sancov_bools:0000000000044B84
__sancov_bools:0000000000044B85 byte_44B85      db 0
__sancov_bools:0000000000044B85
__sancov_bools:0000000000044B86 byte_44B86      db 0
__sancov_bools:0000000000044B86
__sancov_bools:0000000000044B87 byte_44B87      db 0
__sancov_bools:0000000000044B87
__sancov_bools:0000000000044B88 byte_44B88      db 0
__sancov_bools:0000000000044B88
__sancov_bools:0000000000044B89 byte_44B89      db 0
__sancov_bools:0000000000044B89
__sancov_bools:0000000000044B8A byte_44B8A      db 0
__sancov_bools:0000000000044B8A
__sancov_bools:0000000000044B8B byte_44B8B      db 0
__sancov_bools:0000000000044B8B
__sancov_bools:0000000000044B8C byte_44B8C      db 0
__sancov_bools:0000000000044B8C
__sancov_bools:0000000000044B8D byte_44B8D      db 0
__sancov_bools:0000000000044B8D
__sancov_bools:0000000000044B8D __sancov_bools  ends
__sancov_bools:0000000000044B8D
LOAD:0000000000044B8E ; ==============================
```



초기 : 00000000000000000000000000 -> 특정 비트 1로 설정, 이미 1이면 skip



적용되는 위치에는 유의미한 변화가 없다

SanitizerCoverage

PC-Table

clang **-fsanitize-coverage=pc-table** -o test test.c
clang **-fsanitize-coverage=pc-table,trace-pc-guard** -o test test.c

PC-Table은 단독으로 쓰일 수 없다
.
trace-pc-guard 계측 결과가 PC 테이블에 저장된다.

Note: this instrumentation might be incompatible with dead code stripping
(ref : https://bugs.llvm.org/show_bug.cgi?id=34636)

SanitizerCoverage

PC-Table

```
// 전역 변수로 PC 테이블을 저장
static const uintptr_t* pc_table_start = NULL;
static const uintptr_t* pc_table_end = NULL;

// PC 테이블 초기화 함수
void __sanitizer_cov_pcs_init(const uintptr_t* pcs_beg, const uintptr_t* pcs_end) {
    pc_table_start = pcs_beg;
    pc_table_end = pcs_end;
    printf("PC Table initialized: %p - %p₩n", pcs_beg, pcs_end);
}

// 프로그램 종료 시 테이블 내용을 출력하는 함수
void print_pc_table() {
    if (pc_table_start == NULL || pc_table_end == NULL) {
        printf("PC Table not initialized.₩n");
        return;
    }

    printf("PC Table Contents:₩n");
    for (const uintptr_t* ptr = pc_table_start; ptr < pc_table_end; ptr += 2) {
        uintptr_t pc = ptr[0];       // 첫 번째 값은 PC
        uintptr_t flags = ptr[1];   // 두 번째 값은 PCFlags
        printf("PC: 0x%lx, Flags: 0x%lx₩n", pc, flags);
    }
}
```

SanitizerCoverage

PC-Table

```
whoami@whoami:~/jeon_cov/test$ ./test
PC Table initialized: 0x5b0aae31cbb8 - 0x5b0aae31cc98
score? : 90
score : 90, grade : A
PC Table Contents:
PC: 0x5b0aae304da0, Flags: 0x1
PC: 0x5b0aae304dc2, Flags: 0x0
PC: 0x5b0aae304de5, Flags: 0x0
PC: 0x5b0aae304e32, Flags: 0x0
PC: 0x5b0aae304e74, Flags: 0x0
PC: 0x5b0aae304ea0, Flags: 0x1
PC: 0x5b0aae304ec7, Flags: 0x0
PC: 0x5b0aae304ee6, Flags: 0x0
PC: 0x5b0aae304f17, Flags: 0x0
PC: 0x5b0aae304f3a, Flags: 0x0
PC: 0x5b0aae304f5d, Flags: 0x0
PC: 0x5b0aae304f80, Flags: 0x0
PC: 0x5b0aae304f99, Flags: 0x0
PC: 0x5b0aae304fc0, Flags: 0x1
whoami@whoami:~/jeon_cov/test$
```

```
int __cdecl main(int argc, const char **argv, const char **envp)
{
  char v4; // [rsp+7h] [rbp-9h]
  int v5[2]; // [rsp+8h] [rbp-8h] BYREF

  _sanitizer_cov_trace_pc_guard(&unk_44BB4, argv, envp);
  v5[1] = 0;
  v5[0] = 0;
  printf("score? : ");
  __isoc99_scanf("%d", v5);
  v4 = calculate_grade((unsigned int)v5[0]);
  printf("score : %d, grade : %c\n", (unsigned int)v5[0], (unsigned int)v4);
  print_pc_table();
  return 0;
}
```

trace-pc-guard 와 동일한 instrument가 삽입된다

SanitizerCoverage

PC-Table

```
__sancov_guards:0000000000044B80                public __start___sancov_guards
__sancov_guards:0000000000044B80 ; _DWORD _start___sancov_guards[5]
__sancov_guards:0000000000044B80 __start___sancov_guards dd 5 dup(0)     ; DATA XREF: deregister_tm_clones↑o
__sancov_guards:0000000000044B80                                         ; deregister_tm_clones+7↑o ...
__sancov_guards:0000000000044B80                                         ; Alternative name is '__start___sancov_guards'
__sancov_guards:0000000000044B80                                         ; __TMC_END__
__sancov_guards:0000000000044B94 unk_44B94      db    0                  ; DATA XREF: calculate_grade+B↑o
__sancov_guards:0000000000044B94                                         ; calculate_grade:loc_2CEC7↑o ...
__sancov_guards:0000000000044B95                db    0
__sancov_guards:0000000000044B96                db    0
__sancov_guards:0000000000044B97                db    0
__sancov_guards:0000000000044B98                db    0
__sancov_guards:0000000000044B99                db    0
__sancov_guards:0000000000044B9A                db    0
__sancov_guards:0000000000044B9B                db    0
__sancov_guards:0000000000044B9C                db    0
__sancov_guards:0000000000044B9D                db    0
__sancov_guards:0000000000044B9E                db    0
__sancov_guards:0000000000044B9F                db    0
__sancov_guards:0000000000044BA0                db    0
__sancov_guards:0000000000044BA1                db    0
__sancov_guards:0000000000044BA2                db    0
__sancov_guards:0000000000044BA3                db    0
__sancov_guards:0000000000044BA4                db    0
__sancov_guards:0000000000044BA5                db    0
__sancov_guards:0000000000044BA6                db    0
__sancov_guards:0000000000044BA7                db    0
__sancov_guards:0000000000044BA8                db    0
__sancov_guards:0000000000044BA9                db    0
__sancov_guards:0000000000044BAA                db    0
__sancov_guards:0000000000044BAB                db    0
```

trace-pc-guard 와 동일

SanitizerCoverage

PC-Table

```
__sancov_pcs:0000000000044BB8 __sancov_pcs     segment qword public 'DATA' use64
__sancov_pcs:0000000000044BB8                  assume cs:__sancov_pcs
__sancov_pcs:0000000000044BB8                  ;org 44BB8h
__sancov_pcs:0000000000044BB8                  public __start___sancov_pcs
__sancov_pcs:0000000000044BB8 __start___sancov_pcs dq offset print_pc_table
__sancov_pcs:0000000000044BB8                                          ; DATA XREF: sancov_module_ctor_trace_pc_guard+B↑o
__sancov_pcs:0000000000044BB8                                          ; sancov_module_ctor_trace_pc_guard+17↑o
__sancov_pcs:0000000000044BB8                                          ; Alternative name is '__start___sancov_pcs'
__sancov_pcs:0000000000044BB8                                          ; __stop___sancov_guards
__sancov_pcs:0000000000044BC0                  db    1
__sancov_pcs:0000000000044BC1                  db    0
__sancov_pcs:0000000000044BC2                  db    0
__sancov_pcs:0000000000044BC3                  db    0
__sancov_pcs:0000000000044BC4                  db    0
__sancov_pcs:0000000000044BC5                  db    0
__sancov_pcs:0000000000044BC6                  db    0
__sancov_pcs:0000000000044BC7                  db    0
__sancov_pcs:0000000000044BC8                  dq offset loc_2CDC2
__sancov_pcs:0000000000044BD0                  db    0
__sancov_pcs:0000000000044BD1                  db    0
__sancov_pcs:0000000000044BD2                  db    0
__sancov_pcs:0000000000044BD3                  db    0
__sancov_pcs:0000000000044BD4                  db    0
__sancov_pcs:0000000000044BD5                  db    0
__sancov_pcs:0000000000044BD6                  db    0
__sancov_pcs:0000000000044BD7                  db    0
__sancov_pcs:0000000000044BD8                  dq offset loc_2CDE5
__sancov_pcs:0000000000044BE0                  db    0
__sancov_pcs:0000000000044BE1                  db    0
__sancov_pcs:0000000000044BE2                  db    0
__sancov_pcs:0000000000044BE3                  db    0
__sancov_pcs:0000000000044BE4                  db    0
```

PC(addr in 64bit env) : 8byte

SanitizerCoverage

Inline 8bit-counters

Inline bool-flag

PC-Table

} Experimental, may change or disappear in future

LLVM Bugzilla is read-only and represents the historical archive of all LLVM issues filled before November 26, 2021. Use github to submit LLVM bugs

Bug 34636 - **make -fsanitize-coverage=pc-table friendly with -ffunction-sections -Wl,-gc-sections**

Status: NEW
Alias: None
Product: new-bugs
Component: new bugs (show other bugs)
Version: unspecified
Hardware: PC Linux
Importance: P enhancement
Assignee: Matt Morehouse

Reported: 2017-09-15 16:24 PDT by Kostya Serebryany
Modified: 2020-01-18 19:06 PST (History)
CC List: 13 users (show)
See Also:
Fixed By Commit(s):

SanitizerCoverage

Trace-PC

clang **-fsanitize-coverage=trace-pc** -o test test.c

__sanitizer_cov_trace_pc() 를 모든 edge에 삽입

-fsanitize-coverage=indirect-calls 함께 사용 시, __sanitizer_cov_trace_pc_indirect(void *callee) 가 모든 간접 호출에 삽입

Linux Kernel Fuzzing에 사용된다.
(Syzkaller 에서 발견)

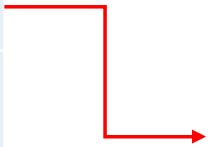


Ref : https://github.com/google/syzkaller

SanitizerCoverage

Trace-PC VS Trace-PC-Guard

	-fsanitize-coverage=trace-pc	-fsanitize-coverage=trace-pc-guard
콜백 함수	__sanitizer_cov_trace_pc()	__sanitizer_cov_trace_pc_guard()
추적 단위	프로그램 카운터(PC)	Guard 변수 (정수 배열)
코드 삽입량	많음	상대적으로 적음
효율성	추적 정확도가 높음	효율적이고 퍼포먼스 우수
사용 사례	Linux 커널 퍼징	libFuzzer, 유저랜드 퍼징



왜 그런건지
아직 이해 못함
추후 공부해 볼 예정

SanitizerCoverage

Tracing Data Flow

주요 플래그

- **-fsanitize-coverage=trace-cmp**: 비교(comparison) 및 switch 문에 대한 추가 계측.
- **-fsanitize-coverage=trace-div**: 정수 나눗셈의 피연산자를 계측.
- **-fsanitize-coverage=trace-gep**: 배열 인덱스 접근(GEP, GetElementPtr) 계측.
- **-fsanitize-coverage=trace-loads / -fsanitize-coverage=trace-stores**: 메모리 읽기(load) 및 쓰기(store) 명령 계측.

사용 조건

단독으로는 작동하지 않으며, 반드시 다음 중 하나의 플래그와 함께 사용해야 함

- **-fsanitize-coverage=trace-pc**
- **-fsanitize-coverage=inline-8bit-counters**
- **-fsanitize-coverage=inline-bool**

SanitizerCoverage

Tracing Control Flow

-fsanitize-coverage=control-flow :

함수의 각 기본 블록(basic block)에 대해 후속 블록(successor) 및 호출된 함수(callee)를 기록하는 테이블 생성.

후속 블록 리스트: 해당 블록에서 다음에 실행 가능한 블록들.
호출된 함수 리스트: 해당 블록에서 호출된 함수(내장 함수 제외).

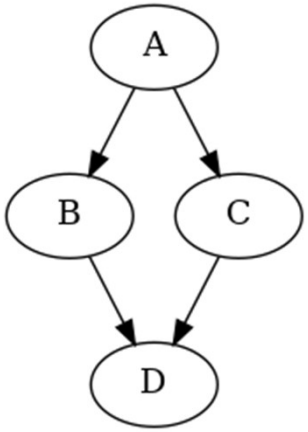
Indirect Calls은 아직 지원되지 않는다. -1으로 표시

SanitizerCoverage

Tracing Control Flow

```
int foo(int x) {
    if (x > 0)
        bar(x);
    else
        x = 0;
    return x;
}
```

- **A**: 시작 블록 (조건문 실행).
- **B**: x > 0일 때 호출되는 bar(x) 실행 블록.
- **C**: x <= 0일 때 x = 0을 실행하는 블록.
- **D**: 종료 블록.



가능한 경우의 수

A -> B

A -> C

B -> D

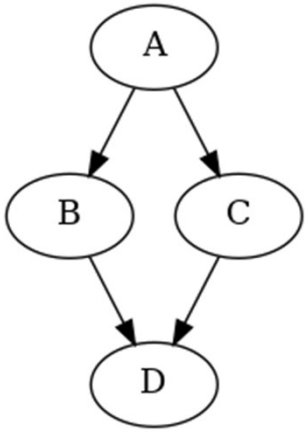
C -> D

SanitizerCoverage

Tracing Control Flow

```
int foo(int x) {
    if (x > 0)
        bar(x);
    else
        x = 0;
    return x;
}
```

- **A**: 시작 블록 (조건문 실행).
- **B**: x > 0일 때 호출되는 bar(x) 실행 블록.
- **C**: x <= 0일 때 x = 0을 실행하는 블록.
- **D**: 종료 블록.



가능한 경우의 수

A -> B

A -> C

B -> D

C -> D

Tracing Control Flow Table

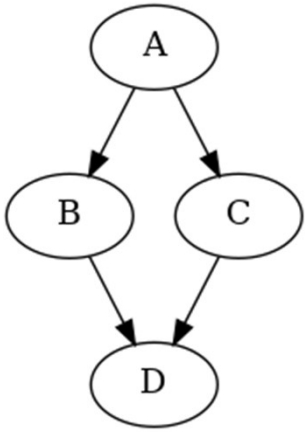
A, B, C, null, null,
B, D, null, @bar, null,
C, D, null, null,
D, null, null

SanitizerCoverage

Tracing Control Flow

```
int foo(int x) {
    if (x > 0)
        bar(x);
    else
        x = 0;
    return x;
}
```

- **A**: 시작 블록 (조건문 실행).
- **B**: x > 0일 때 호출되는 bar(x) 실행 블록.
- **C**: x <= 0일 때 x = 0을 실행하는 블록.
- **D**: 종료 블록.



가능한 경우의 수

A -> B

A -> C

B -> D

C -> D

Tracing Control Flow Table

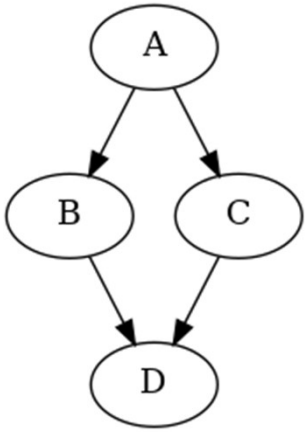
A, B, C,	**null**,	**null**,
B, D,	**null**, @bar,	**null**,
C, D,	**null**,	**null**,
D,	**null**,	**null**

SanitizerCoverage

Tracing Control Flow

```
int foo(int x) {
    if (x > 0)
        bar(x);
    else
        x = 0;
    return x;
}
```

- **A**: 시작 블록 (조건문 실행).
- **B**: x > 0일 때 호출되는 bar(x) 실행 블록.
- **C**: x <= 0일 때 x = 0을 실행하는 블록.
- **D**: 종료 블록.



가능한 경우의 수

A -> B

A -> C

B -> D

C -> D

기본 블록(basic block)

↓

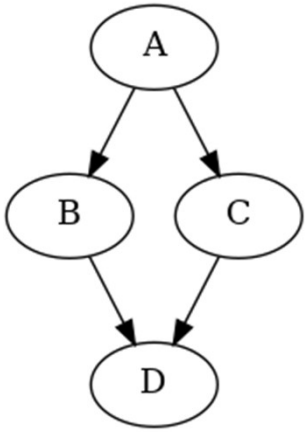
A, B, C, **null**, **null**,
B, D, **null**, @bar, **null**,
C, D, **null**, **null**,
D, **null**, **null**

SanitizerCoverage

Tracing Control Flow

```
int foo(int x) {
    if (x > 0)
        bar(x);
    else
        x = 0;
    return x;
}
```

- **A**: 시작 블록 (조건문 실행).
- **B**: x > 0일 때 호출되는 bar(x) 실행 블록.
- **C**: x <= 0일 때 x = 0을 실행하는 블록.
- **D**: 종료 블록.



가능한 경우의 수

A -> B

A -> C

B -> D

C -> D

후속 블록(successor)

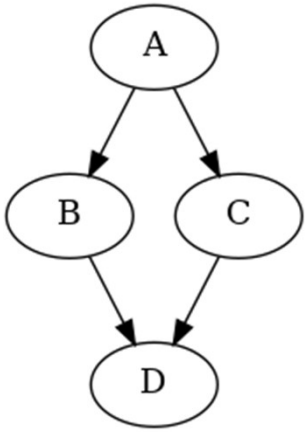
A, B, C, **null**, **null**,
B, D, **null**, @bar, **null**,
C, D, **null**, **null**,
D, **null**, **null**

SanitizerCoverage

Tracing Control Flow

```
int foo(int x) {
    if (x > 0)
        bar(x);
    else
        x = 0;
    return x;
}
```

- **A**: 시작 블록 (조건문 실행).
- **B**: x > 0일 때 호출되는 bar(x) 실행 블록.
- **C**: x <= 0일 때 x = 0을 실행하는 블록.
- **D**: 종료 블록.



가능한 경우의 수

A -> B

A -> C

B -> D

C -> D

호출된 함수(callee)

A, B, C, **null**, **null**,
B, D, **null**, @bar, **null**,
C, D, **null**, **null**,
D, **null**, **null**

```
// Table data storage variables (global)
const uintptr_t* cfs_table_begin = NULL;
const uintptr_t* cfs_table_end = NULL;

// Control flow table initialization function
extern "C" void __sanitizer_cov_cfs_init(const uintptr_t * cfs_beg,
    const uintptr_t * cfs_end) {
    // Save the start and end pointers of the table
    cfs_table_begin = cfs_beg;
    cfs_table_end = cfs_end;
}
```

```
// Function to print the control flow table
void print_cfs_table() {
    if (cfs_table_begin == NULL || cfs_table_end == NULL) {
        printf("Control flow table is not initialized.₩n");
        return;
    }

    printf("Control Flow Table:₩n");
    const uintptr_t* current = cfs_table_begin;
    while (current < cfs_table_end) {
        printf("Basic Block Address: 0x%lx₩n", *current++);

        // Print successor blocks
        printf("  Successor Blocks: ");
        while (*current != 0) {
            printf("0x%lx ", *current++);
        }
        current++; // Skip null
        printf("₩n");

        // Print called functions
        printf("  Called Functions: ");
        while (*current != 0) {
            if (*current == (uintptr_t)-1) {
                printf("[Indirect Call] ");
            }
            else {
                printf("0x%lx ", *current);
            }
            current++;
        }
        current++; // Skip null
        printf("₩n");
    }
}
```

```
whoami@whoami:~/jeon_cov/test$ clang -fsanitize-coverage=control-flow -o test test.c
clang: error: unsupported argument 'control-flow' to option 'fsanitize-coverage='
```

```
whoami@whoami:~/jeon_cov/test$ clang -v
Ubuntu clang version 14.0.0-1ubuntu1.1
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
Found candidate GCC installation: /usr/bin/../lib/gcc/x86_64-linux-gnu/11
Found candidate GCC installation: /usr/bin/../lib/gcc/x86_64-linux-gnu/12
Selected GCC installation: /usr/bin/../lib/gcc/x86_64-linux-gnu/12
Candidate multilib: .;@m64
Selected multilib: .;@m64
```

https://clang.llvm.org/docs/SanitizerCoverage.html#edge-coverage 문서에 있는데

https://releases.llvm.org/13.0.0/tools/clang/docs/SanitizerCoverage.html 문서에는 빠짐

https://releases.llvm.org/14.0.0/tools/clang/docs/SanitizerCoverage.html 도 없음

모든 구버전 문서를 조회하였으나 별 소득이 없어 중단.

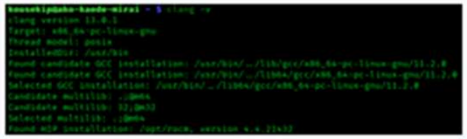
구글링을 해봐도 자료가 없었음

https://clang.llvm.org/docs/SanitizerCoverage.html#edge-coverage

Clang 20.0.0git documentation

SANITIZERCOVERAGE

Clang



Clang 13.0.1

Original author(s)	Chris Lattner
Developer(s)	LLVM Developer Group
Initial release	September 26, 2007; 17 years ago[1]
Stable release	19.1.6[2] / 17 December 2024; 5 days ago
Repository	github.com/llvm/llvm-project

llvm.org

Readme

View license

Code of conduct

Security policy

Activity

Custom properties

29.8k stars

581 watching

12.3k forks

Report repository

Releases 120

LLVM 19.1.6 Latest

2 weeks ago

+ 119 releases

Branches **Tags**

llvmorg-20-init

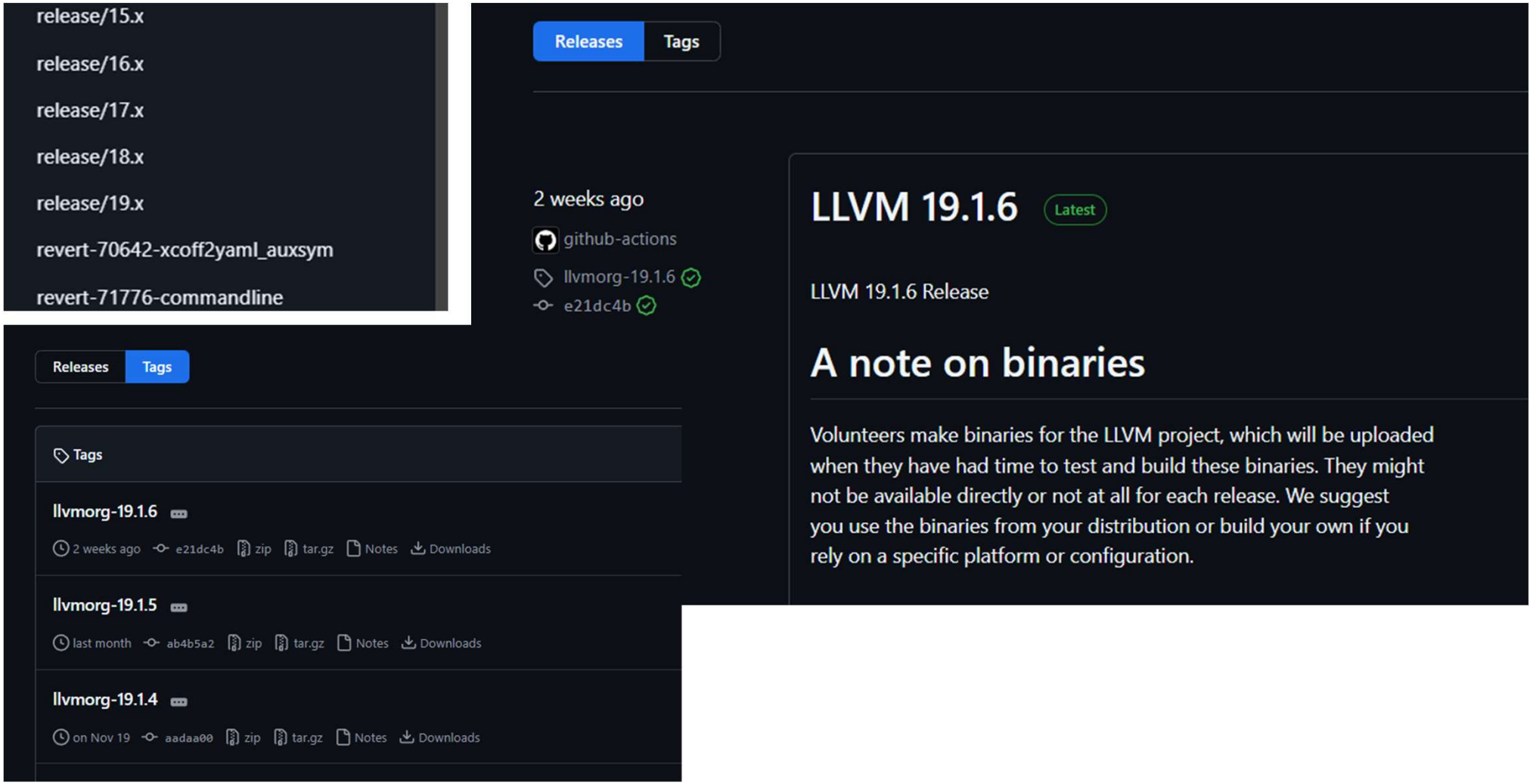
llvmorg-19.1.6

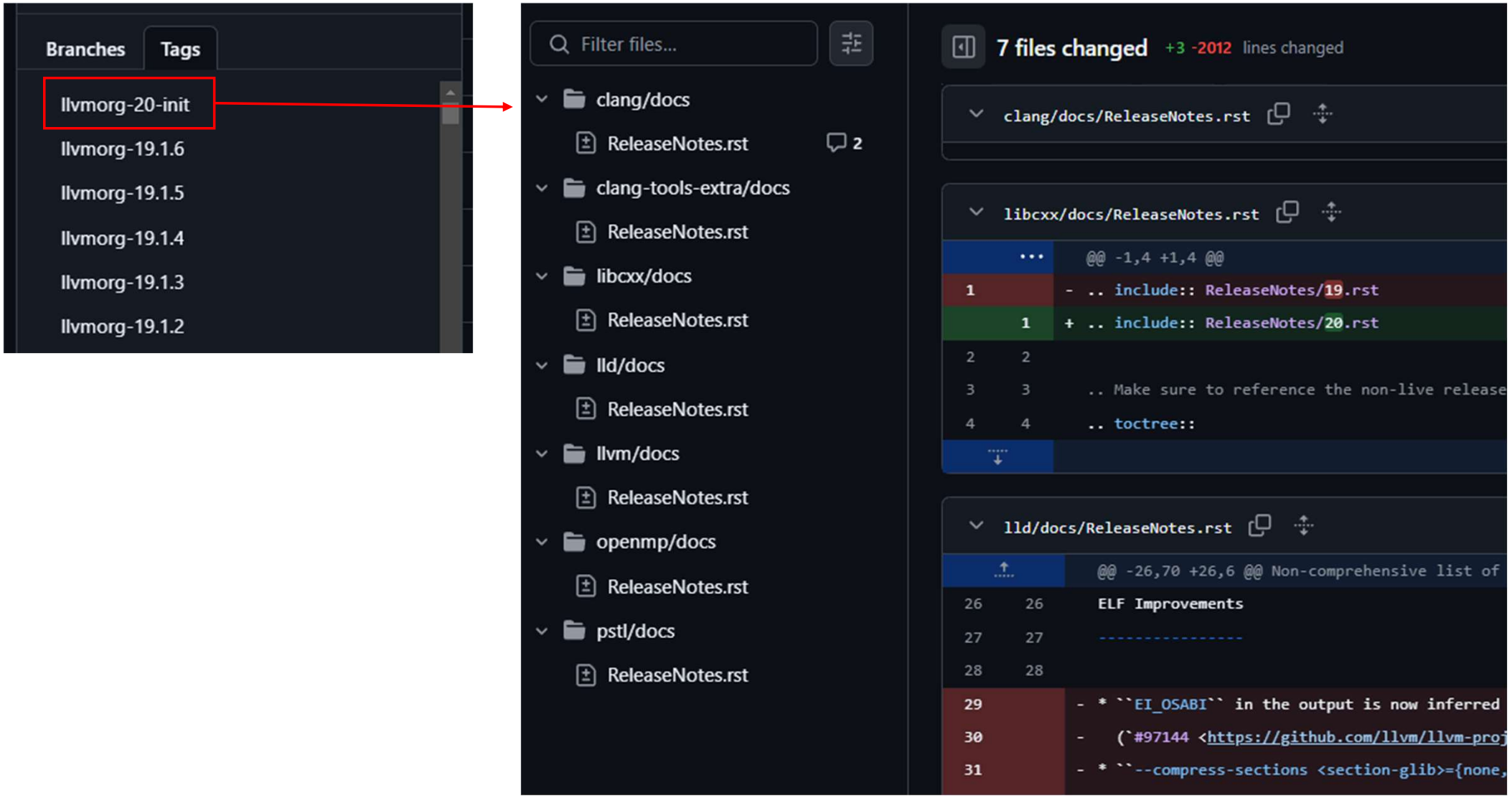
llvmorg-19.1.5

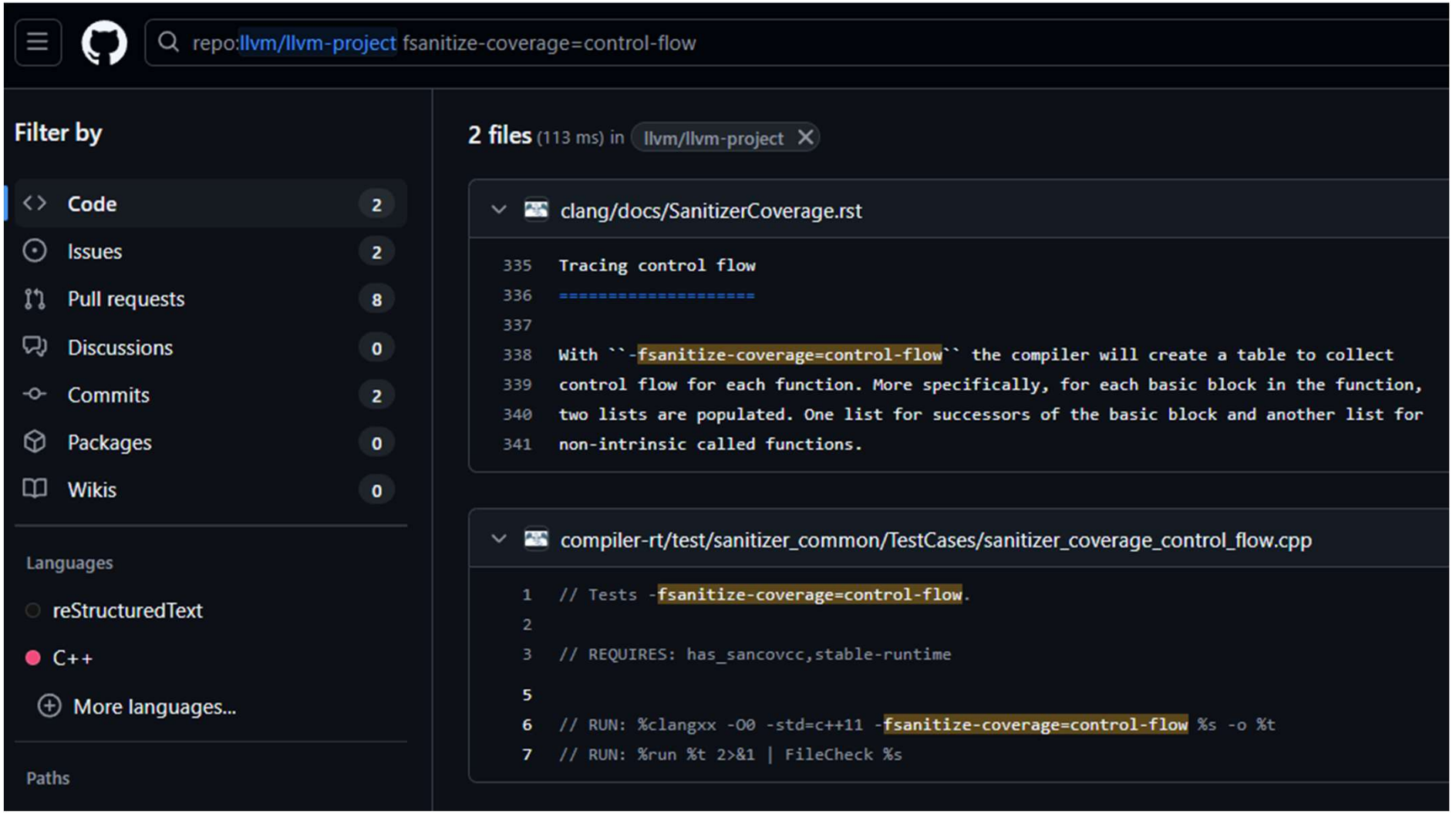
llvmorg-19.1.4

llvmorg-19.1.3

llvmorg-19.1.2


release/15.x
release/16.x
release/17.x
release/18.x
release/19.x
revert-70642-xcoff2yaml_auxsym
revert-71776-commandline
Releases
Tags
2 weeks ago
github-actions
llvmorg-19.1.6
e21dc4b
LLVM 19.1.6
Latest
LLVM 19.1.6 Release
A note on binaries
Volunteers make binaries for the LLVM project, which will be uploaded when they have had time to test and build these binaries. They might not be available directly or not at all for each release. We suggest you use the binaries from your distribution or build your own if you rely on a specific platform or configuration.
Releases
Tags
Tags
llvmorg-19.1.6
2 weeks ago e21dc4b zip tar.gz Notes Downloads
llvmorg-19.1.5
last month ab4b5a2 zip tar.gz Notes Downloads
llvmorg-19.1.4
on Nov 19 aadaa00 zip tar.gz Notes Downloads


Branches
Tags
llvmorg-20-init
llvmorg-19.1.6
llvmorg-19.1.5
llvmorg-19.1.4
llvmorg-19.1.3
llvmorg-19.1.2
Filter files...
clang/docs
ReleaseNotes.rst
2
clang-tools-extra/docs
ReleaseNotes.rst
libcxx/docs
ReleaseNotes.rst
lld/docs
ReleaseNotes.rst
llvm/docs
ReleaseNotes.rst
openmp/docs
ReleaseNotes.rst
pstl/docs
ReleaseNotes.rst
7 files changed +3 -2012 lines changed
clang/docs/ReleaseNotes.rst
libcxx/docs/ReleaseNotes.rst
@@ -1,4 +1,4 @@
1 - .. include:: ReleaseNotes/19.rst
1 + .. include:: ReleaseNotes/20.rst
2 2
3 3 .. Make sure to reference the non-live release
4 4 .. toctree::
lld/docs/ReleaseNotes.rst
@@ -26,70 +26,6 @@ Non-comprehensive list of
26 26 ELF Improvements
27 27 ----------------
28 28
29 - * ``EI_OSABI`` in the output is now inferred
30 - (`#97144 <https://github.com/llvm/llvm-proj
31 - * ``--compress-sections <section-glib>={none,



Git에도 구현 코드는 찾을 수 없었다

SanitizerCoverage

Selective Coverage(정식 명칭은 아님)

AFL의 경우, __afl_coverage_off/__afl_coverage_on 등의 함수로 coverage를 런타임에서 조절 할 수 있었음
이는 소스코드에 해당 함수를 사용하는 방식이었음

CLANG에는 allowlist와 blocklist로 선택적인 coverage 특정 전략을 compile time에 지정할 수 있다

SanitizerCoverage

Selective Coverage(정식 명칭은 아님)

문법 1. #을 기준으로 주석처리된다

문법 2. src:<경로> 로 대상 파일을 지정한다

문법 3. src:<경로>/* 를 사용할 수 있다.

문법 4. fun:<함수이름> 을 사용할 수 있다.

문법 5. fun:*<함수이름>* 과 같이 wildcard 사용을 강력히 권장한다(안쓸거면 mangling 된 이름을 사용)

이는 allow, block 모두 동일하게 적용된다

Compiler가 함수 이름 임의로 수정하는 것
C++ 특성 상 function overloading 이 필요하기에 발생한다고 함

SanitizerCoverage

Selective Coverage(정식 명칭은 아님)

allowlist

```
# Enable instrumentation for a whole folder
src:bar/*
# Enable instrumentation for a specific source file
src:foo/a.cpp
# Enable instrumentation for all functions in those files
fun:*
```

Folder, File 지정하고 내부에 모든 함수로 지정
느슨한 셋팅

blocklist

```
# Disable instrumentation for a specific source file that the
allowlist allowed
src:bar/b.cpp
# Disable instrumentation for a specific function that the
allowlist allowed
fun:*myFunc*
```

특정 파일에 특정 패턴의 함수를 지정
Specific한 셋팅

SanitizerCoverage

Selective Coverage(정식 명칭은 아님)

test.c

```
#include <stdio.h>

void includedFunction() {
    printf("Included Function");
}

// 이 함수는 차단 리스트에 의해 제외
void excludedFunction() {
    printf("Excluded Function");
}

int main() {
    includedFunction();
    excludedFunction();
    return 0;
}
```

blocklsit

```
fun:*excludedFunction*
```

SanitizerCoverage

Selective Coverage(정식 명칭은 아님)

clang -fsanitize-coverage=trace-pc-guard -o test test.c **-fsanitize-coverage-ignorelist=blocklist**

```
; __unwind {
push    rbp
mov     rbp, rsp
sub     rsp, 10h
lea     rdi, unk_43874
call    __sanitizer_cov_trace_pc_guard
mov     [rbp+var_4], 0
call    includedFunction
call    excludedFunction
xor     eax, eax
add     rsp, 10h
pop     rbp
retn
; } // starts at 2CD80
main endp
```

<main>

```
includedFunction proc near
; __unwind {
push    rbp
mov     rbp, rsp
lea     rdi, __start___sancov_guards
call    __sanitizer_cov_trace_pc_guard
lea     rdi, aIncludedFuncti ; "Included Function"
mov     al, 0
call    _printf
pop     rbp
retn
; } // starts at 2CD60
includedFunction endp
```

<IncludedFunction>

```
public excludedFunction
excludedFunction proc near
; __unwind {
push    rbp
mov     rbp, rsp
lea     rdi, aExcludedFuncti ; "Excluded Function"
mov     al, 0
call    _printf
pop     rbp
retn
; } // starts at 2CD40
excludedFunction endp
```

<ExcludedFunction>

SanitizerCoverage

Default implementation

ASAN, MemorySanitizer, etc. -> default implementations of some of the coverage callback

clang -fsanitize-coverage=trace-pc-guard **-fsanitize=address** -o test test.c

ASAN_OPTIONS=coverage=1 ./test

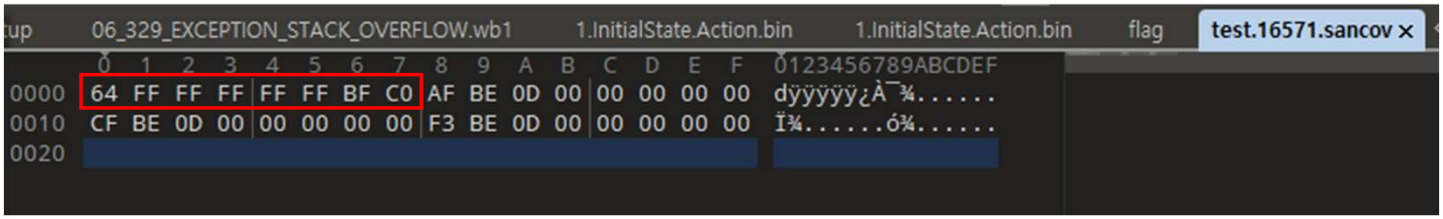
```
whoami@whoami:~/jeon_cov/test$ ASAN_OPTIONS=coverage=1 ./test
SanitizerCoverage: ./test.16571.sancov: 3 PCs written
```

generate

```
whoami@whoami:~/jeon_cov/test$ ls test.16571.sancov
test.16571.sancov
```

SanitizerCoverage

Default implementation

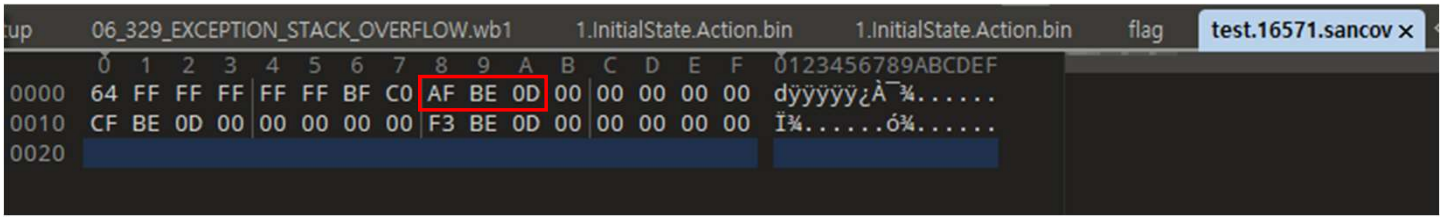


첫 8바이트: Magic 값
0xC0BFFFFFFFFFFF64/0xC0BFFFFFFFFFFF32

Little endian이라는 문서는 없지만, 추정가능

SanitizerCoverage

Default implementation

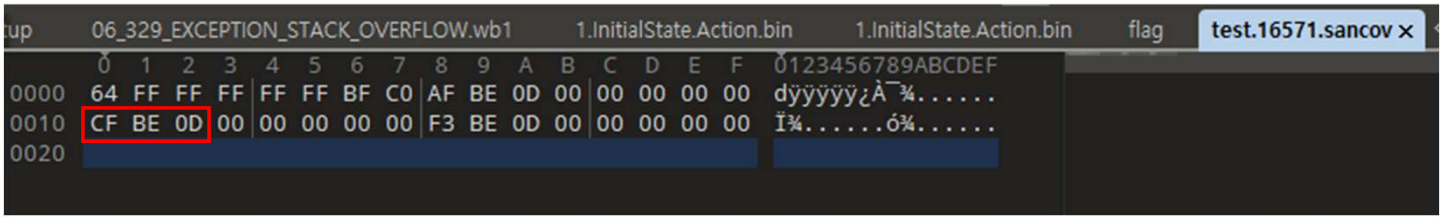


Offset : 0x0DBEAF

```
.text:00000000000DBEA0                  public includedFunction
.text:00000000000DBEA0 includedFunction proc near               ; CODE XREF: main+1B↓p
.text:00000000000DBEA0 ; __unwind {
.text:00000000000DBEA0                  push    rbp             ; char
.text:00000000000DBEA1                  mov     rbp, rsp
.text:00000000000DBEA4                  lea     rdi, __start___sancov_guards
.text:00000000000DBEAB                  call    __sanitizer_cov_trace_pc_guard
.text:00000000000DBEB0                  lea     rdi, _str       ; "Included Function"
.text:00000000000DBEB7                  mov     al, 0
.text:00000000000DBEB9                  call    printf
.text:00000000000DBEBE                  pop     rbp
.text:00000000000DBEBF                  retn
.text:00000000000DBEBF ; } // starts at DBEA0
.text:00000000000DBEBF includedFunction endp
```

SanitizerCoverage

Default implementation

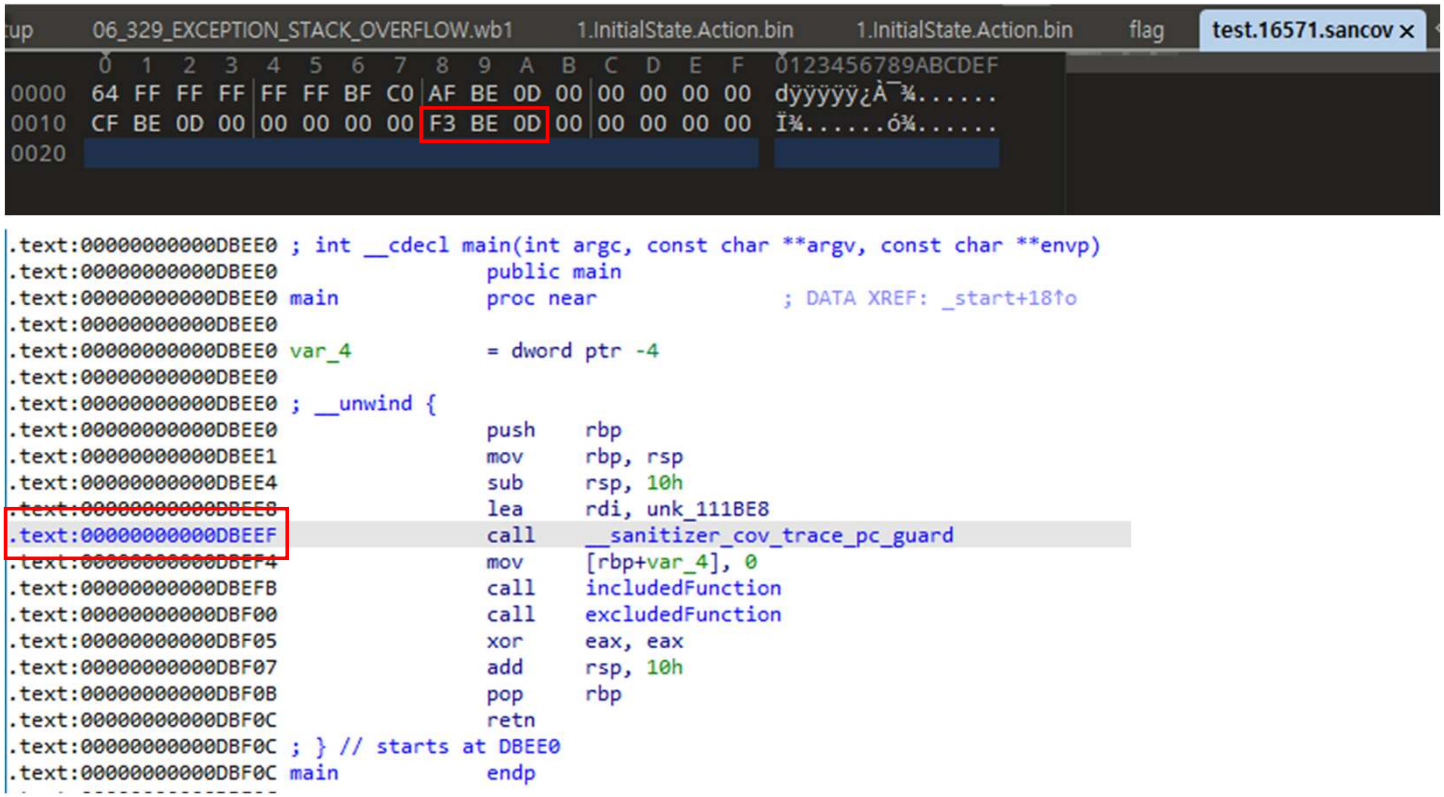


Offset : 0x0DBECF

```
.text:00000000000DBEC0 excludedFunction proc near              ; CODE XREF: main+20↓p
.text:00000000000DBEC0 ; __unwind {
.text:00000000000DBEC0                 push    rbp             ; char
.text:00000000000DBEC1                 mov     rbp, rsp
.text:00000000000DBEC4                 lea     rdi, unk_111BE4
.text:00000000000DBECB                 call    __sanitizer_cov_trace_pc_guard
.text:00000000000DBED0                 lea     rdi, _str_1     ; "Excluded Function"
.text:00000000000DBED7                 mov     al, 0
.text:00000000000DBED9                 call    printf
.text:00000000000DBEDE                 pop     rbp
.text:00000000000DBEDF                 retn
.text:00000000000DBEDF ; } // starts at DBEC0
.text:00000000000DBEDF excludedFunction endp
```

SanitizerCoverage

Default implementation



요구사항 :

- LLVM SanitizerCoverage 사용
- Code coverage 측정

Plan :

- 가장 기본적인 trace-pc-guard만 사용
- Dumb fuzzing VS Coverage based fuzzing 비교

<test.c>

```c
#include <stdio.h>


// 점수 계산 함수
char calculate_grade(int score) {
    if (score < 0 || score > 100) {
        printf("Error: 점수는 0~100 사이여야 합니다.₩n");
        return 'E'; // Error
    }

    if (score >= 90)      return 'A';
    else if (score >= 80) return 'B';
    else if (score >= 70) return 'C';
    else if (score >= 60) return 'D';
    else                  return 'F';
}

// 메인 함수
int main() {
    int score = 0;
    printf("score? : ");
    scanf("%d", &score);

    char grade = calculate_grade(score);
    printf("score : %d, grade : %c₩n", score, grade);

    return 0;
}
```

```
whoami@whoami:~/jeon_cov/test$ ./test
score? : 34
score : 34, grade : F
```

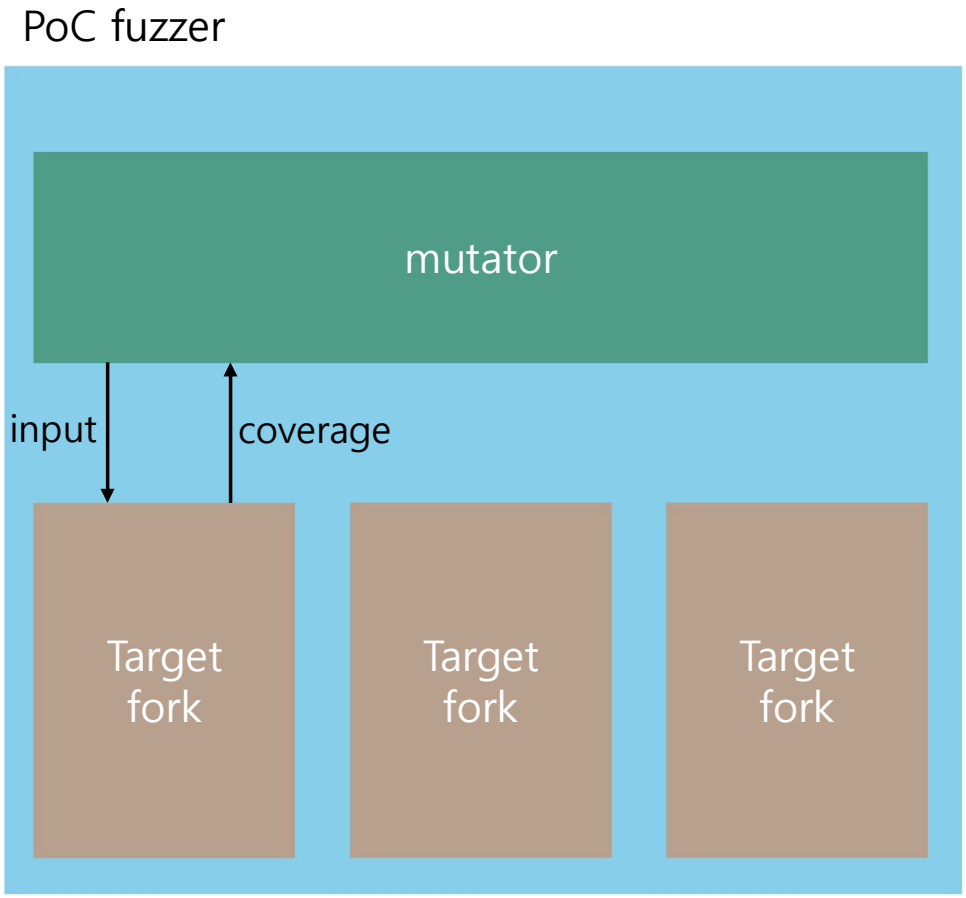
```
// 전역 파이프 (커버리지 수집용)
int ipc_pipe[2];

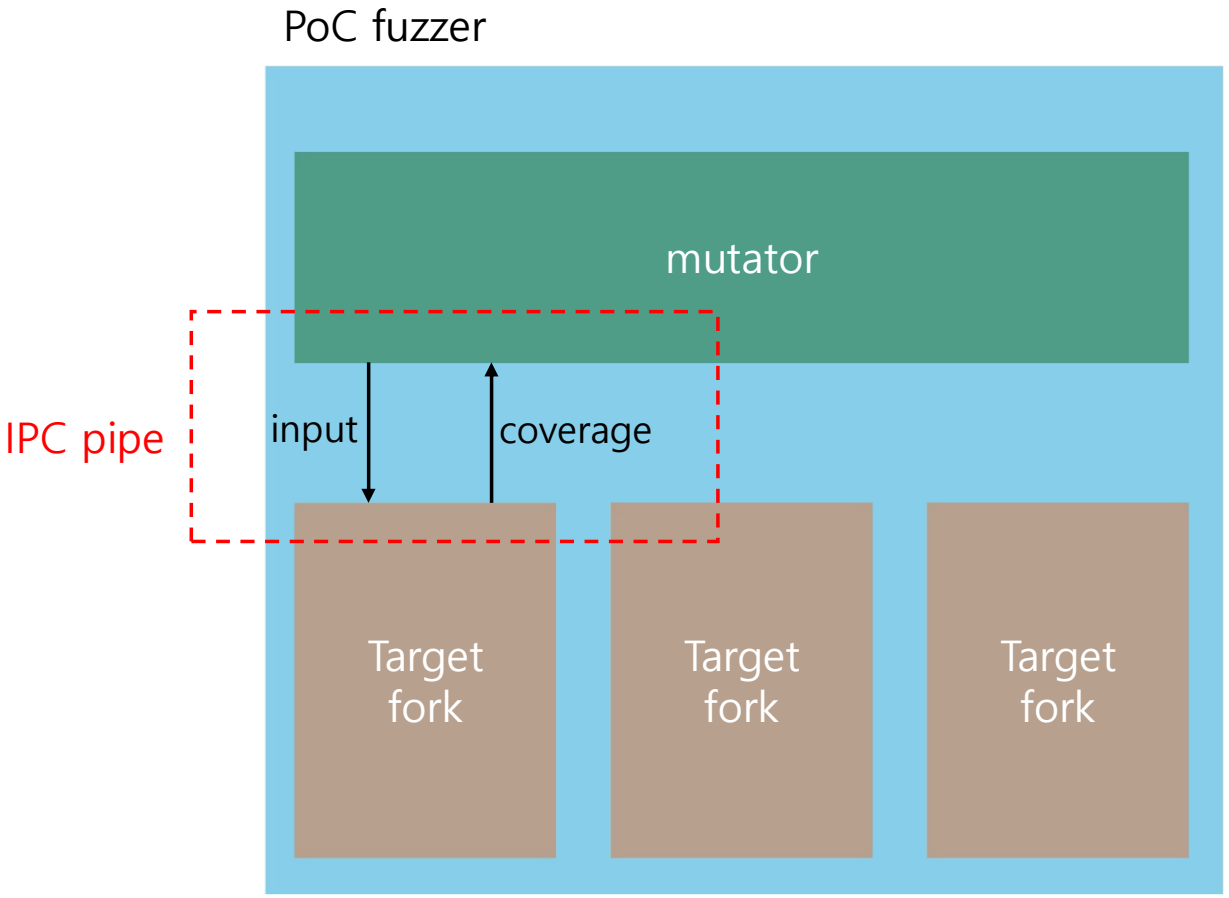
// Sanitizer Coverage 콜백 함수
void __sanitizer_cov_trace_pc_guard(uint32_t* guard) {
    //if (!*guard) return; // guard가 0이면 이미 방문된 코드

    void* PC = __builtin_return_address(0);

    // 디버깅 출력
    /////fprintf(stderr, "[Debug] Writing guard=%p, PC=%p₩n", *guard, PC);

    // guard와 PC 주소를 파이프로 전송
    write(ipc_pipe[1], guard, sizeof(*guard));  // guard 자체의 값
    write(ipc_pipe[1], &PC, sizeof(PC));        // PC(리턴 어드레스)
}
```


PoC fuzzer
mutator
input
coverage
Target fork
Target fork
Target fork


PoC fuzzer
mutator
IPC pipe
input
coverage
Target fork
Target fork
Target fork

COV.C

```
<생략>
pid_t
 pid = fork();
if (pid < 0) {
    perror("fork");
    exit(1);
}

if (pid == 0) {
    // 자식 프로세스

    // 부모가 읽을 파이프 닫음
    close(coverage_pipe[0]);

    // coverage_pipe[1]을 전역 변수 ipc_pipe[1]로 사용
    dup2(coverage_pipe[1], 101);  // fd 101에 복사
    close(coverage_pipe[1]);

    // 자식이 읽을 input_pipe[0]을 STDIN으로 설정
    close(input_pipe[1]);
    dup2(input_pipe[0], STDIN_FILENO);
    close(input_pipe[0]);

    // target 바이너리 실행
    execl("./target", "./target", NULL);

    // 실행 실패 시
    perror("execl");
    exit(1);
}
else {
```

```
// 부모 프로세스

    // 자식이 쓸 파이프 닫음
    close(coverage_pipe[1]);
    close(input_pipe[0]);
    // 자식 프로세스에 STDIN으로 점수 전달
    char buf[32];
    snprintf(buf, sizeof(buf), "%d₩n", input);
    write(input_pipe[1], buf, strlen(buf));
    close(input_pipe[1]);  // 더 이상 필요 없으므로 닫음
    // coverage_pipe[0]에서 커버리지 정보 읽기
    printf("[Parent] Test score: %d₩n", input);
    while (1) {
        uint32_t guard_val;
        void* pc_val;
        // guard 읽기
        ssize_t ret = read(coverage_pipe[0], &guard_val, sizeof(guard_val));
        if (ret == 0) break;  // 자식 종료 시 파이프 닫힘
        if (ret < 0) {
            perror("read(guard)");
            break;
        }
        // PC 읽기
        ret = read(coverage_pipe[0], &pc_val, sizeof(pc_val));
        if (ret <= 0) {
            perror("read(pc)");
            break;
        }
        // 읽은 데이터 출력
        printf("  >> guard=%u, pc=%p₩n", guard_val, pc_val);
        map[guard_val] = 1;
    }
```

```
whoami@whoami:~/jeon_cov/grade$ ./cov
Usage: ./cov <score1> <score2> <score3> <score4>
whoami@whoami:~/jeon_cov/grade$ ./cov 90 80 70 50
Updated test_scores:
test_scores[0] = 90
test_scores[1] = 80
test_scores[2] = 70
test_scores[3] = 50
[Parent] Test score: 90
  >> guard=1, pc=0x59b516712dc7
  >> guard=4, pc=0x59b516712e37
[Parent] Child exited (score 90)

[Parent] Test score: 80
  >> guard=1, pc=0x62c2a02acdc7
  >> guard=5, pc=0x62c2a02ace5a
[Parent] Child exited (score 80)

[Parent] Test score: 70
  >> guard=1, pc=0x5d02ff3f0dc7
  >> guard=6, pc=0x5d02ff3f0e7d
[Parent] Child exited (score 70)

[Parent] Test score: 50
  >> guard=1, pc=0x5cefd8f9edc7
  >> guard=8, pc=0x5cefd8f9eeb9
[Parent] Child exited (score 50)

0100111010000000
hit : 31 %
```

```
whoami@whoami:~/jeon_cov/grade$ ./cov 90 92 93 99
Updated test_scores:
test_scores[0] = 90
test_scores[1] = 92
test_scores[2] = 93
test_scores[3] = 99
[Parent] Test score: 90
  >> guard=1, pc=0x570ec3776dc7
  >> guard=4, pc=0x570ec3776e37
[Parent] Child exited (score 90)

[Parent] Test score: 92
  >> guard=1, pc=0x5dc0ae02ddc7
  >> guard=4, pc=0x5dc0ae02de37
[Parent] Child exited (score 92)

[Parent] Test score: 93
  >> guard=1, pc=0x590f283cadc7
  >> guard=4, pc=0x590f283cae37
[Parent] Child exited (score 93)

[Parent] Test score: 99
  >> guard=1, pc=0x63bee02fddc7
  >> guard=4, pc=0x63bee02fde37
[Parent] Child exited (score 99)

0100100000000000
hit : 12 %
```

```
whoami@whoami:~/jeon_cov/grade$ ./cov
Usage: ./cov <score1> <score2> <score3> <score4>
whoami@whoami:~/jeon_cov/grade$ ./cov 90 80 70 50
Updated test_scores:
test_scores[0] = 90
test_scores[1] = 80
test_scores[2] = 70
test_scores[3] = 50
[Parent] Test score: 90
  >> guard=1, pc=0x59b516712dc7
  >> guard=4, pc=0x59b516712e37
[Parent] Child exited (score 90)

[Parent] Test score: 80
  >> guard=1, pc=0x62c2a02acdc7
  >> guard=5, pc=0x62c2a02ace5a
[Parent] Child exited (score 80)

[Parent] Test score: 70
  >> guard=1, pc=0x5d02ff3f0dc7
  >> guard=6, pc=0x5d02ff3f0e7d
[Parent] Child exited (score 70)

[Parent] Test score: 50
  >> guard=1, pc=0x5cefd8f9edc7
  >> guard=8, pc=0x5cefd8f9eeb9
[Parent] Child exited (score 50)

0100111010000000
hit : 31 %
```

```c
char calculate_grade(int score) {
    if (score < 0 || score > 100) {
        printf("Error: 점수는 0~100 사이여야 합니다.₩n");
        return 'E'; // Error
    }

    if (score >= 90)      return 'A';
    else if (score >= 80) return 'B';
    else if (score >= 70) return 'C';
    else if (score >= 60) return 'D';
    else                  return 'F';
}
```

target.c

```c
char calculate_grade(int score) {
    if (score < 0 || score > 100) {
        printf("Error: 점수는 0~100 사이여야 합니다.₩n");
        return 'E'; // Error
    }

    if (score >= 90)      return 'A';
    else if (score >= 80) return 'B';
    else if (score >= 70) return 'C';
    else if (score >= 60) return 'D';
    else                  return 'F';
}
```

target.c

```
whoami@whoami:~/jeon_cov/grade$ ./cov 90 92 93 99
Updated test_scores:
test_scores[0] = 90
test_scores[1] = 92
test_scores[2] = 93
test_scores[3] = 99
[Parent] Test score: 90
  >> guard=1, pc=0x570ec3776dc7
  >> guard=4, pc=0x570ec3776e37
[Parent] Child exited (score 90)

[Parent] Test score: 92
  >> guard=1, pc=0x5dc0ae02ddc7
  >> guard=4, pc=0x5dc0ae02de37
[Parent] Child exited (score 92)

[Parent] Test score: 93
  >> guard=1, pc=0x590f283cadc7
  >> guard=4, pc=0x590f283cae37
[Parent] Child exited (score 93)

[Parent] Test score: 99
  >> guard=1, pc=0x63bee02fddc7
  >> guard=4, pc=0x63bee02fde37
[Parent] Child exited (score 99)

0100100000000000
hit : 12 %
```

dumb_fuzz.c

```
void random_testing() {
    int best_fitness = 0;
    int best_inputs[4] = { 0 };

    for (int iteration = 0; iteration < ITERATIONS; iteration++) {
        int inputs[4];
        for (int i = 0; i < 4; i++) {
            inputs[i] = rand() % 101; // 0부터 100까지 랜덤 숫자 생성
        }

        memset(cov_map, 0, MAP_SIZE * sizeof(bool));

        for (int i = 0; i < 4; i++) {
            cov_run(cov_map, inputs[i]);
        }

        int fitness = count_ones(cov_map, MAP_SIZE);

        if (fitness > best_fitness) {
            best_fitness = fitness;
            memcpy(best_inputs, inputs, sizeof(best_inputs));
        }
```

dumb_fuzz.c

```
        printf("Iteration %d: Fitness = %d%%₩n", iteration, fitness * 100 / MAP_SIZE);
        if (best_fitness * 100 / MAP_SIZE == 31) break; // 최대 31% 달성 시 종료
    }

    printf("₩nBest fitness: %d%%₩n", best_fitness * 100 / MAP_SIZE);
    printf("Best inputs: ");
    for (int i = 0; i < 4; i++) {
        printf("%d ", best_inputs[i]);
    }
    printf("₩n");
}

int main() {
    srand(time(NULL));
    random_testing();
    return 0;
}
```

비교를 위해 31%가 최대 coverage라는 것을 안다는 전제 하에 실험하였다.
실제 환경에서는 coverage가 충분히 커졌다는 것을 보장할 수 없다.

Makefile

```
# Makefile

CC = clang
CFLAGS = -fsanitize-coverage=trace-pc-guard

# Targets
all: cov target dumb_fuzz cov_fuzz

cov: cov.c
	$(CC) $< -o $@

target: target.c
	$(CC) $(CFLAGS) $< -o $@

dumb_fuzz: dumb_fuzz.c
	$(CC) $< -o $@

cov_fuzz: cov_fuzz.c
	$(CC) $< -o $@

# Cleanup
clean:
	rm -f cov target dumb_fuzz cov_fuzz
```

```
whoami@whoami:~/jeon_cov/grade$ ./dumb_fuzz
Iteration 0: Fitness = 18%
Iteration 1: Fitness = 18%
Iteration 2: Fitness = 12%
Iteration 3: Fitness = 18%
Iteration 4: Fitness = 25%
Iteration 5: Fitness = 25%
Iteration 6: Fitness = 12%
Iteration 7: Fitness = 18%
Iteration 8: Fitness = 12%
Iteration 9: Fitness = 18%
Iteration 10: Fitness = 18%
Iteration 11: Fitness = 18%
Iteration 12: Fitness = 18%
Iteration 13: Fitness = 18%
Iteration 14: Fitness = 18%
Iteration 15: Fitness = 25%
Iteration 16: Fitness = 12%
Iteration 17: Fitness = 12%
Iteration 18: Fitness = 12%
Iteration 19: Fitness = 25%
Iteration 20: Fitness = 25%
Iteration 21: Fitness = 18%
Iteration 22: Fitness = 25%
Iteration 23: Fitness = 12%
Iteration 24: Fitness = 18%
Iteration 25: Fitness = 18%
Iteration 26: Fitness = 18%
Iteration 27: Fitness = 18%
Iteration 28: Fitness = 12%
Iteration 29: Fitness = 25%
Iteration 30: Fitness = 18%
Iteration 31: Fitness = 18%
Iteration 32: Fitness = 25%
Iteration 33: Fitness = 18%
Iteration 34: Fitness = 31%

Best fitness: 31%
Best inputs: 31 68 78 96
whoami@whoami:~/jeon_cov/grade$
```

```
whoami@whoami:~/jeon_cov/grade$ ./dumb_fuzz
Iteration 0: Fitness = 18%
Iteration 1: Fitness = 25%
Iteration 2: Fitness = 18%
Iteration 3: Fitness = 25%
Iteration 4: Fitness = 31%

Best fitness: 31%
Best inputs: 37 83 94 63
```

```
whoami@whoami:~/jeon_cov/grade$ ./dumb_fuzz
Iteration 0: Fitness = 25%
Iteration 1: Fitness = 12%
Iteration 2: Fitness = 25%
Iteration 3: Fitness = 25%
Iteration 4: Fitness = 25%
Iteration 5: Fitness = 18%
Iteration 6: Fitness = 18%
Iteration 7: Fitness = 25%
Iteration 8: Fitness = 18%
Iteration 9: Fitness = 18%
Iteration 10: Fitness = 18%
Iteration 11: Fitness = 25%
Iteration 12: Fitness = 12%
Iteration 13: Fitness = 12%
Iteration 14: Fitness = 25%
Iteration 15: Fitness = 25%
Iteration 16: Fitness = 12%
Iteration 17: Fitness = 18%
Iteration 18: Fitness = 18%
Iteration 19: Fitness = 25%
Iteration 20: Fitness = 18%
Iteration 21: Fitness = 31%
```

```
whoami@whoami:~/jeon_cov/grade$ ./dumb_fuzz
Iteration 0: Fitness = 25%
Iteration 1: Fitness = 12%
Iteration 2: Fitness = 25%
Iteration 3: Fitness = 25%
Iteration 4: Fitness = 25%
Iteration 5: Fitness = 18%
Iteration 6: Fitness = 18%
Iteration 7: Fitness = 25%
Iteration 8: Fitness = 18%
Iteration 9: Fitness = 18%
Iteration 10: Fitness = 18%
Iteration 11: Fitness = 25%
Iteration 12: Fitness = 12%
Iteration 13: Fitness = 12%
Iteration 14: Fitness = 25%
Iteration 15: Fitness = 25%
Iteration 16: Fitness = 12%
Iteration 17: Fitness = 18%
Iteration 18: Fitness = 18%
Iteration 19: Fitness = 25%
Iteration 20: Fitness = 18%
Iteration 21: Fitness = 31%

Best fitness: 31%
Best inputs: 37 95 67 76
```

Fitness가 random하다

(올라갔다 내려갔다함)

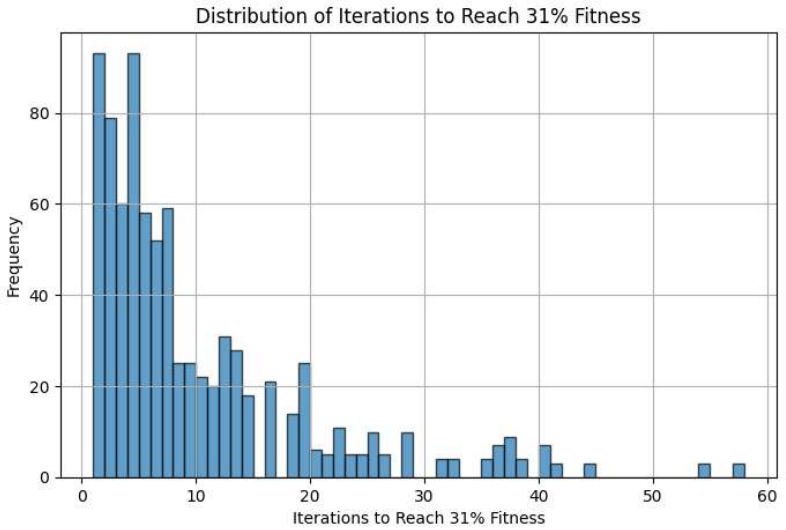
```python
import subprocess
import matplotlib.pyplot as plt

# Function to run . / dumb_fuzz and parse its output
def run_dumb_fuzz() :
    iterations_to_31 = []

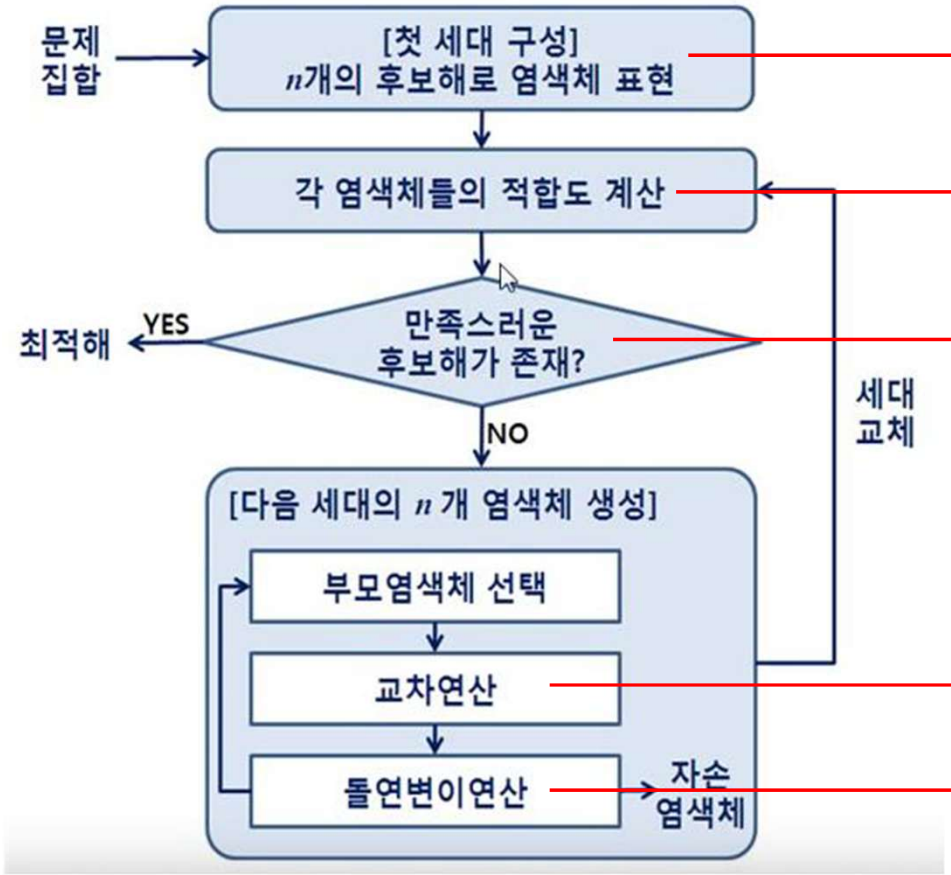
    for _ in range(1000) : # Run the program 100 times
        process = subprocess.Popen("./dumb_fuzz", stdout = subprocess.PIPE,
stderr = subprocess.PIPE, text = True)
        output, _ = process.communicate()

        # Parse the output to find the iteration where 31 % fitness is
reached
        for line in output.splitlines() :
            if "Fitness = 31%" in line :
iteration = int(line.split()[1].strip(":"))  # Extract the iteration number
iterations_to_31.append(iteration)
break
```

benchmark.py




```
Average Iterations to Reach 31%: 8.10
Maximum Iterations: 58
Minimum Iterations: 0
whoami@whoami:~/jeon_cov/grade$
```



Rand % 101

Coverage 조사

Coverage 증가?

Idx 를 랜덤하게

고정 확률(10%)
[Rand % 101]

cov_fuzz.c

```
for (int i = 0; i < POP_SIZE; i++) {
    for (int j = 0; j < EXEC_COUNT; j++) {
        population[i][j] = rand() % 101; // 초기값 랜덤 설정
    }
}
```

초기 값 뽑기

```
#define MUTATION_RATE 0.1
```

```
void mutate(int individual[EXEC_COUNT]) {
    for (int i = 0; i < EXEC_COUNT; i++) {
        if ((rand() / (double)RAND_MAX) < MUTATION_RATE) {
            individual[i] = rand() % 101;
        }
    }
}
```

돌연변이 생성

```
void crossover(int parent1[EXEC_COUNT], int parent2[EXEC_COUNT], int
child1[EXEC_COUNT], int child2[EXEC_COUNT]) {
    int crossover_point = rand() % EXEC_COUNT;
    for (int i = 0; i < EXEC_COUNT; i++) {
        if (i < crossover_point) {
            child1[i] = parent1[i];
            child2[i] = parent2[i];
        }
        else {
            child1[i] = parent2[i];
            child2[i] = parent1[i];
        }
    }
}
```

교배 (랜덤한 idx 전/후 로 부모 유전자 받음)

cov_fuzz.c

```
void genetic_algorithm() {
    int population[POP_SIZE][4];
    int fitness[POP_SIZE];

    for (int i = 0; i < POP_SIZE; i++) {
        for (int j = 0; j < 4; j++) {
            population[i][j] = rand() % 101;
        }
    }

    for (int generation = 0; generation < GENERATIONS; generation++) {
        evaluate_population(cov_map, population, fitness);

        int new_population[POP_SIZE][4];

        for (int i = 0; i < POP_SIZE / 2; i++) {
            int parent1_idx = rand() % POP_SIZE;
            int parent2_idx = rand() % POP_SIZE;

            crossover(population[parent1_idx], population[parent2_idx],
new_population[i * 2], new_population[i * 2 + 1]);

            mutate(new_population[i * 2]);
            mutate(new_population[i * 2 + 1]);
        }
```

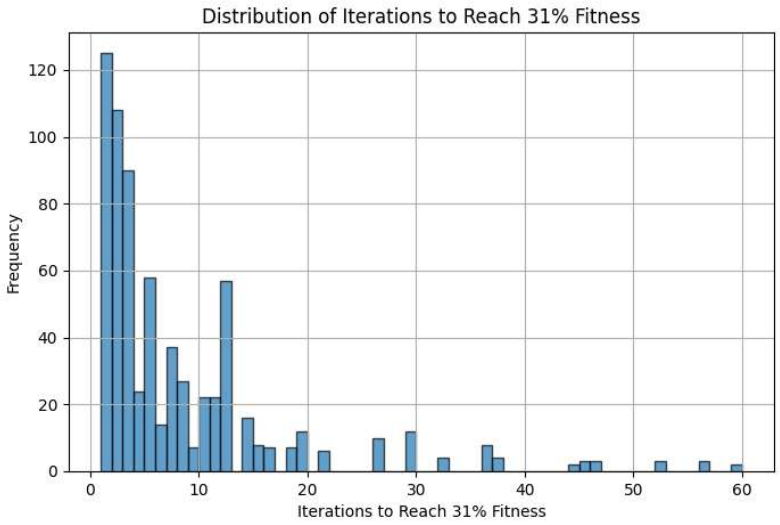
부모 선택(무작위)

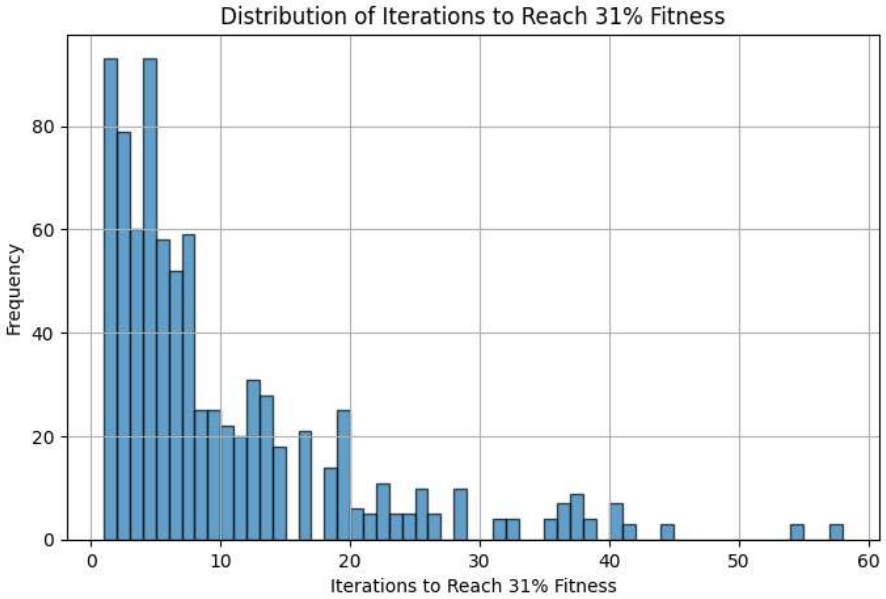
```
whoami@whoami:~/jeon_cov/grade$ ./cov_fuzz
Generation 0: Best fitness = 25%
Generation 1: Best fitness = 25%
Generation 2: Best fitness = 18%
Generation 3: Best fitness = 25%
Generation 4: Best fitness = 25%
Generation 5: Best fitness = 31%

whoami@whoami:~/jeon_cov/grade$ ./cov_fuzz
Generation 0: Best fitness = 18%
Generation 1: Best fitness = 12%
Generation 2: Best fitness = 18%
Generation 3: Best fitness = 18%
Generation 4: Best fitness = 18%
Generation 5: Best fitness = 18%
Generation 6: Best fitness = 25%
Generation 7: Best fitness = 12%
Generation 8: Best fitness = 18%
Generation 9: Best fitness = 12%
Generation 10: Best fitness = 18%
Generation 11: Best fitness = 18%
Generation 12: Best fitness = 18%
Generation 13: Best fitness = 25%
Generation 14: Best fitness = 18%
Generation 15: Best fitness = 18%
Generation 16: Best fitness = 18%
Generation 17: Best fitness = 12%
Generation 18: Best fitness = 12%
Generation 19: Best fitness = 18%
Generation 20: Best fitness = 25%
Generation 21: Best fitness = 18%
Generation 22: Best fitness = 25%
Generation 23: Best fitness = 12%
Generation 24: Best fitness = 25%
Generation 25: Best fitness = 25%
Generation 26: Best fitness = 12%
```

대부분의 경우에서 dumb fuzz 보다 빠르게 best fitness인 31%에 도달하지만, 유전 알고리즘 연산으로 인해 1 cycle이 느려짐에 있어서 시간 관점에서 큰 효율을 얻지 못하였다.

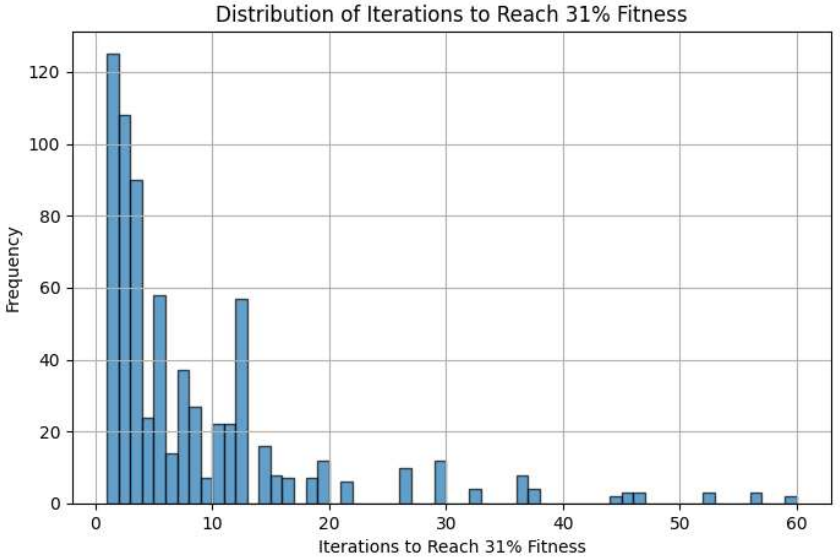
```
Average Iterations to Reach 31%: 5.68
Maximum Iterations: 60
Minimum Iterations: 0
```






```
Average Iterations to Reach 31%: 8.10
Maximum Iterations: 58
Minimum Iterations: 0
whoami@whoami:~/jeon_cov/grade$
```

dumb fuzz




```
Average Iterations to Reach 31%: 5.68
Maximum Iterations: 60
Minimum Iterations: 0
```

Coverage based fuzz
(little bit faster)

```
char calculate_grade(int score) {
    if (score < 0 || score > 100) {
        printf("Error: 점수는 0~100 사이여야 합니다.₩n");
        return 'E'; // Error
    }

    if (score >= 90) {
        if (score >= 95) {
            return 'A'; // A+
        }
        else {
            return 'A'; // A
        }
    }
    else if (score >= 80) {
        if (score >= 85) {
            if (score >= 88) {
                return 'B'; // B+
            }
            else {
                return 'B'; // B
            }
        }
        else {
            return 'B'; // B-
        }
    }
    else if (score >= 70) {
        if (score >= 75) {
            if (score >= 78) {
                return 'C'; // C+
            }
            else {
                return 'C'; // C
            }
        }
        else {
            return 'C'; // C-
        }
    }
    else if (score >= 60) {
        if (score >= 65) {
            if (score >= 68) {
                return 'D'; // D+
            }
            else {
                return 'D'; // D
            }
        }
        else {
            return 'D'; // D-
        }
    }
    else {
        if (score >= 50) {
            if (score >= 55) {
                return 'F'; // F+
            }
            else {
                return 'F'; // F
            }
        }
        else {
            return 'F'; // F-
        }
    }
}
```

IDEA

기존 코드는 매우 단순하여, random하게 시도해도 충분히 모든 커버리지를 hit할 수 있었다

복잡한 로직의 경우, 단순한 random 방식으로는 모든 경우의 수를 찾을 수 없을 것이라고 추측하였다

dumb_fuzz.c 는 3000번 실행하고 최대 커버리지를 출력하는 방식으로 수정하였다.
(Maximum Coverage에 도달하지 못하기에 / 68%가 이론 상 최대 cov)

기존에는 4개의 정수를 랜덤하게 생성하여 1개 배치가 4번의 실행을 포함하였다.

이를 10개의 정수를 1개 배치로 수정하였다.
(real-world에서는 보다 복잡한 input이 대부분이라고 생각한다, ex:image)

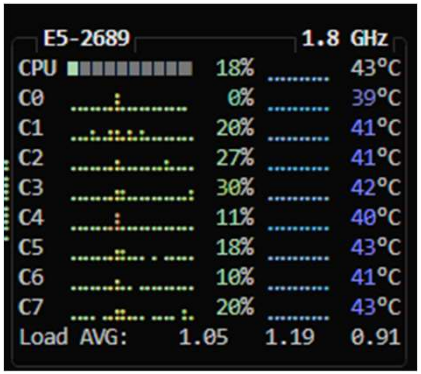
dumb_fuzz

```
Iteration 2974: Fitness = 43%
Iteration 2975: Fitness = 37%
Iteration 2976: Fitness = 43%
Iteration 2977: Fitness = 37%
Iteration 2978: Fitness = 12%
Iteration 2979: Fitness = 37%
Iteration 2980: Fitness = 25%
Iteration 2981: Fitness = 31%
Iteration 2982: Fitness = 31%
Iteration 2983: Fitness = 37%
Iteration 2984: Fitness = 37%
Iteration 2985: Fitness = 31%
Iteration 2986: Fitness = 18%
Iteration 2987: Fitness = 31%
Iteration 2988: Fitness = 37%
Iteration 2989: Fitness = 31%
Iteration 2990: Fitness = 37%
Iteration 2991: Fitness = 31%
Iteration 2992: Fitness = 37%
Iteration 2993: Fitness = 25%
Iteration 2994: Fitness = 31%
Iteration 2995: Fitness = 31%
Iteration 2996: Fitness = 50%
Iteration 2997: Fitness = 43%
Iteration 2998: Fitness = 31%
Iteration 2999: Fitness = 25%

Best fitness: 56%
Best inputs: 78 87 57 57 84 28 64 66 73 88
```

dumb_fuzz 3000번의 시도를 수행
대부분 best fit 기준 50~60%의 커버리지에 도달한다.

최대 커버리지에 도달하지 못하였다



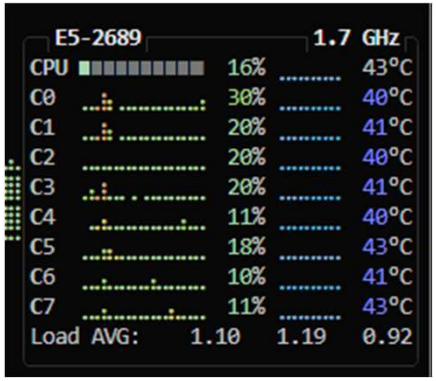
cov_fuzz

```
Generation 33: Best fitness = 62%
Generation 34: Best fitness = 62%
Generation 35: Best fitness = 62%
Generation 36: Best fitness = 62%
Generation 37: Best fitness = 62%
Generation 38: Best fitness = 62%
Generation 39: Best fitness = 62%
Generation 40: Best fitness = 62%
Generation 41: Best fitness = 62%
Generation 42: Best fitness = 62%
Generation 43: Best fitness = 62%
Generation 44: Best fitness = 62%
Generation 45: Best fitness = 62%
Generation 46: Best fitness = 62%
Generation 47: Best fitness = 62%
Generation 48: Best fitness = 62%
Generation 49: Best fitness = 62%
Generation 50: Best fitness = 62%
Generation 51: Best fitness = 62%
Generation 52: Best fitness = 68%
Best individual: 100 60 88 56 68 80 78 77 91 73
```

cov_fuzz 평균 45~55번의 세대에서 최대 커버리지 도달

실제 실행 횟수는 더 많기에 dumb_fuzz와 iteration으로 비교할 순 없다
(유전자 score check으로 인한)

시간으로 비교하였을 때 매우 큰 차이가 나타난다.
(cov_fuzz 는 10초 안쪽으로 최대 커버리지 도달)



```
char calculate_grade(int score) {
    if (score < 0 || score > 100) {
        printf("Error: 점수는 0~100 사이여야 합니다.₩n");
        return 'E'; // Error
    }

    if (score >= 90) {
        if (score >= 95) {
            return 'A'; // A+
        }
        else {
            return 'A'; // A
        }
    }
    else if (score >= 80) {
        if (score >= 85) {
            if (score >= 88) {
                return 'B'; // B+
            }
            else {
                return 'B'; // B
            }
        }
        else {
            return 'B'; // B-
        }
    }
    else if (score >= 70) {
        if (score >= 75) {
            if (score >= 78) {
                return 'C'; // C+
            }
            else {
                return 'C'; // C
            }
        }
        else {
            return 'C'; // C-
        }
    }
    else if (score >= 60) {
        if (score >= 65) {
            if (score >= 68) {
                return 'D'; // D+
            }
            else {
                return 'D'; // D
            }
        }
        else {
            return 'D'; // D-
        }
    }
    else {
        if (score >= 50) {
            if (score >= 55) {
                return 'F'; // F+
            }
            else {
                return 'F'; // F
            }
        }
        else {
            return 'F'; // F-
        }
    }
}
```

cov_fuzz 결과 :

76 58 99 85 91 82 62 72 67 68

10회 실행 모두 다양한 분기문에 도달하였음

dumb_fuzz	cov_fuzz
최대 커버리지 도달 실패	최대 커버리지 도달
N/A	10초 내의 런타임
일관성 없는 fitness (최대 cov 도달 여부 확인 불가)	지속적인 증가 추세의 fitness (최대 cov 도달 여부 추측 가능)

IDEA

충분히 단순한 input에 대해서는, mutation으로 인한 overhead가 더 critical 할 수 있다.

그러나 충분히 복잡한 프로그램을 대상으로 fuzzing할 때에는, coverage기반의 mutation이 필수적이라고 생각한다.

Mutation algorithm으로 GA(Genetic Algorithm)를 사용하였다. (블로그에 공개된 코드를 사용)
Input 형태에 따른 적절한 방법의 mutation logic을 사용하여 더 빠른 커버리지 도달을 시도해볼 수 있을 것이라 판단하였다.