Result Analysis

Suppress

▼ cJSON (target 5개)

- function calling count가 내부 branch에 많은 영향을 미치지만, 높다고해서 낮다고 해서 무조건적인 영향이 있는 것은 아니다. suppress했을 때 calling count가 높은데 내부 branch가 낮을 수도 있고, calling count가 낮은데 branch가 높을 수도 있다. 다른 함수에 의해 calling count는 영향을 받을 수 있지만 branch를 뚫으려는 시도는 영향을 받지 않는다.
- target function외 다른 함수들도 같이 suppress의 영향을 받긴하지만 약간 받기는 한다. 즉, target_function과 그 외 function이 둘다 suppress를 받기는 하지만, 그 정도는 다르다는 것이다.

▼ parse_number

- Function calling count: 10.5k → 4.99k

  코드 내부의 branch를 보면 약간씩 branch count가 상승한 곳도 있지만, 전체적으로 대폭 감소하였다.

  1. First evidence(약 2.11배 감소)

     a. 338| 33.7k| case 'E':
     339| 33.7k| number_c_string[i] = buffer_at_offset(input_buffer)[i];
     340| 33.7k| break;

     b. 338| 15.9k| case 'E':
     339| 15.9k| number_c_string[i] = buffer_at_offset(input_buffer)[i];
     340| 15.9k| break;

  2. Second Evidence (약 10.1배 감소)

     a. 362| 10.5k| if (number >= INT_MAX)
     363| 940| {

364| 940| item→valueint = INT_MAX;
365| 940| }

b. 362| 4.95k| if (number >= INT_MAX)
363| 93| {
364| 93| item→valueint = INT_MAX;
365| 93| }

▼ utf16

- Function calling count: 10.4k → 6.11k

해당 함수도 마찬가지로 내부 branch count가 눈에 띄게 줄었다. 그런데 유독 높은 branch count가 종종 있는데 이는 swith의 default같은 부분이다. suppress했음에도 어떤 branch의 count가 기존보다 높다면 이는 대부분 swith의 default같이 원래 count가 높은 branch이다. 예측컨데, 이는 branch를 뚫으려는 시도가 없이 카운트는 높으니 자연스럽게 흘러간 부분이라고 예측된다.

1. First Evidence (약 1.29배 감소)

a. 683| 10.4k| if ((first_code >= 0xD800) && (first_code <= 0xDBFF))
684| 175| {
685| 175| const unsigned char *second_sequence = first_sequence + 6;*
686| 175| unsigned int second_code = 0;
687| 175| sequence_length = 12; / \uXXXX\uXXXX */*

b. 684| 6.09k| if ((first_code >= 0xD800) && (first_code <= 0xDBFF))
685| 135| {
686| 135| const unsigned char *second_sequence = first_sequence + 6;*
687| 135| unsigned int second_code = 0;
688| 135| sequence_length = 12; / \uXXXX\uXXXX */*

2. Second Evidence (약 38배)

a. 729| 545| {
730| | /* two bytes, encoding 110xxxxx 10xxxxxx
/
731| 545| utf8_length = 2;
732| 545| first_byte_mark = 0xC0; /
11000000 */
733| 545| }

b. 730| 14| {
731| | /* two bytes, encoding 110xxxxx 10xxxxxx
/
732| 14| utf8_length = 2;
733| 14| first_byte_mark = 0xC0; /
11000000 */
734| 14| }

▼ print_string_ptr

- function calling count: 18.3k→24.6k

해당 function은 오히려 count가 증가하였다. 하지만 다른 함수의 suppress로 인한 영향을 받을 수 있다고 생각한다.

calling count가 증가하였지만 branch들은 분명히 약 1.2~3배 정도 count가 줄었고 특정 switch문에는 또 default 집중 현상을 보였다. 원래 fuzzing은 여러 branch를 골고루 방문하려고하는데 suppress되니 아무 곳으로 안가고 default로 빠져버리는 거 같다.

함수 내부 전체적으로 branch count는 줄었다.

1. First Evidence (약1.36배 감소)

a. 977| 894| output[0] = '\"';
978| 894| output_pointer = output + 1;

b. 978| 657| output[0] = '\"';
979| 657| output_pointer = output + 1;

c. Second Evidence (약 1.79배 감소)

d. 1011| 2.00k| case '\t':
1012| 2.00k| *output_pointer = 't';

1013| 2.00k| break;

e. 1012| 573| case '\t':
1013| 573| *output_pointer = 't';
1014| 573| break;

▼ print_value

- function calling count: 20.2k → 18.4k

count가 증가한 branch도 있지만 감소한 곳도 많다. 위 함수들과 같은 맥락으로 가장 만만한 branch에 계속 빠져서 해당 branch count는 증가하는 거 같다.

1. First Evidence 약 2.74배 감소

a. 1392| 379| case cJSON_False:
1393| 379| output = ensure(output_buffer, 6);
1394| 379| if (output == NULL)

b. 1393| 138| case cJSON_False:
1394| 138| output = ensure(output_buffer, 6);
1395| 138| if (output == NULL)

2. Secone Evidence 약 6.36배 감소

a. 1434| 3.99k| case cJSON_Array:
1435| 3.99k| return print_array(item, output_buffer);

b. 1435| 627| case cJSON_Array:
1436| 627| return print_array(item, output_buffer);

▼ parse_string

- function calling count: 27.7k → 36.5k

branch count가 더 증가한 곳도 있지만 전체적으로 count가 낮아짐. 그리고 또한 위 같은 현상과 같이 특정branch로 의미없는 집중하는 현상도 있음.

1. First Evidence 약 2.8배 감소

a. 846| 175| case 'f':
847| 175| *output_pointer++ = '\f';
848| 175| break;

b.
```
847|    61|            case 'f':
848|    61|               *output_pointer++ = '\f';
849|    61|               break;
```

2. Second Evidence 약 3.93배 감소

a.
```
852|   236|            case 'r':
853|   236|               *output_pointer++ = '\r';
854|   236|               break;
```

b.
```
853|    60|            case 'r':
854|    60|               *output_pointer++ = '\r';
855|    60|               break;
```

▼ libpng (target 5개)

- target: calc_image_size, zlib_check, zlib_advance, process_chunk, read_callback
- target function임에도 calling count가 더 높아지는 경우가 있다. 그렇지만 calling count가 높아졌다고 해서 절대적으로 branch coverage가 늘어나는 것은 아니다. 오히려 줄어든 경우가 많다.
- Enhance에서 봤던 특징은 어느 하나의 branch에 몰입하는 경우가 있다는 것이었는데, suppress에서는 오히려 평탄하게 접근하는 것을 보여준다. (switch문 내에서)→ target이 5개인 경우 보여주는 특성
- process_chunk에서 기존 fuzzing에서 뚫어내지 못했던 branch를 뚫어내는 결과를 얻었다.

▼ calc_image_size(target)

- function calling count: 1.10k → 1.71k

1. First evidence: switch가 실행된 횟수는 많지만, 그에 반해 내부 branch 눈에 띄게 count가 낮아진 것을 볼 수 있다.

a. 기존

```
1410|  1.10k|   switch (file→color_type)
1411|  1.10k|   {
1412|      5|     default:
1413|      5|       stop_invalid(file, "IHDR: colour type");
```

```
1414|       |
1415|     25|      invalid_bit_depth:
1416|     25|        stop_invalid(file, "IHDR: bit depth");
1417|       |
1418|    947|      case 0: /* g
/
1419|    947|        if (pd != 1 && pd != 2 && pd != 4 && pd != 8 && pd
!= 16)
1420|      5|          goto invalid_bit_depth;
1421|    942|        break;
1422|       |
1423|    942|      case 3:
1424|     40|        if (pd != 1 && pd != 2 && pd != 4 && pd != 8)
1425|      5|          goto invalid_bit_depth;
1426|     35|        break;
1427|       |
1428|     62|      case 2: /
 rgb /
1429|     62|        if (pd != 8 && pd != 16)
1430|      5|          goto invalid_bit_depth;
1431|       |
1432|     57|        pd = (png_uint_16)(pd * 3);
1433|     57|        break;
1434|       |
1435|     11|      case 4: /
 ga /
1436|     11|        if (pd != 8 && pd != 16)
1437|      5|          goto invalid_bit_depth;
1438|       |
1439|      6|        pd = (png_uint_16)(pd * 2);
1440|      6|        break;
1441|       |
1442|     35|      case 6: /
 rgba */
1443|     35|        if (pd != 8 && pd != 16)
1444|      5|          goto invalid_bit_depth;
```

```
1445|      |
1446|    30|        pd = (png_uint_16)(pd * 4);
1447|    30|        break;
1448|  1.10k|   }
```

b. Target

```
1410|  1.71k|   switch (file→color_type)
1411|  1.71k|   {
1412|     6|     default:
1413|     6|       stop_invalid(file, "IHDR: colour type");
1414|      |
1415|    29|     invalid_bit_depth:
1416|    29|       stop_invalid(file, "IHDR: bit depth");
1417|      |
1418|   169|     case 0: /* g
/
1419|   169|       if (pd != 1 && pd != 2 && pd != 4 && pd != 8 && pd
!= 16)
1420|     7|         goto invalid_bit_depth;
1421|   162|       break;
1422|      |
1423|   797|     case 3:
1424|   797|       if (pd != 1 && pd != 2 && pd != 4 && pd != 8)
1425|     7|         goto invalid_bit_depth;
1426|   790|       break;
1427|      |
1428|   790|     case 2: /
 rgb /
1429|   707|       if (pd != 8 && pd != 16)
1430|     5|         goto invalid_bit_depth;
1431|      |
1432|   702|       pd = (png_uint_16)(pd * 3);
1433|   702|       break;
1434|      |
1435|    15|     case 4: /
 ga /
```

```
1436|    15|        if (pd != 8 && pd != 16)
1437|     5|          goto invalid_bit_depth;
1438|      |
1439|    10|        pd = (png_uint_16)(pd * 2);
1440|    10|        break;
1441|      |
1442|    24|      case 6: /
 rgba */
1443|    24|        if (pd != 8 && pd != 16)
1444|     5|          goto invalid_bit_depth;
1445|      |
1446|    19|        pd = (png_uint_16)(pd * 4);
1447|    19|        break;
```

2. Second Evidence: suppress임에도 불구하고 많이 들어갔다. 이런 예외사항이 곳곳에 존재한다.
    a. 기존: 약 18.1배 차이나는 시도 횟수 대비 들어간 횟수

```
1462|  1.05k|   else switch (file->interlace_method)
1463|  1.05k|   {
1464|     58|     case PNG_INTERLACE_ADAM7:
```

    b. Target: 약 5.15배 차이나는 시도 횟수 대비 들어간 횟수

```
1462|  1.64k|   else switch (file->interlace_method)
1463|  1.64k|   {
1464|    318|     case PNG_INTERLACE_ADAM7:
```

▼ zlib_check(target)

- Function calling count: 1.70k → 1.58k

1. First Evidence: 첫 번째 case에 대해서는 branch count 비율이 증가했지만, 두 번째 case에 대해서는 확실히 떨어진 것을 볼 수 있다. 이는 suppress의 특징인 '평탄화'를 보여주고 있다.
    a. 기존

```
2625|  1.70k|     switch (rc)
2626|  1.70k|     {
2627|    105|       case ZLIB_TOO_FAR_BACK:
```

```
2628|       |          /* too far back error */
2629|    105|          file->status_code |= TOO_FAR_BACK;
2630|    105|          min_bits = zlib.window_bits + 1;
2631|    105|          max_bits = 15;
2632|    105|          break;
2633|       |
2634|  1.27k|        case ZLIB_STREAM_END:
2635|  1.27k|          if (!zlib.global->optimize_zlib &&
2636|  1.27k|              zlib.window_bits == zlib.file_bits &&
!zlib.cksum)
2637|  1.23k|          {
```

b. Target

```
2625|  1.58k|      switch (rc)
2626|  1.58k|      {
2627|    137|        case ZLIB_TOO_FAR_BACK:
2628|       |          /* too far back error */
2629|    137|          file->status_code |= TOO_FAR_BACK;
2630|    137|          min_bits = zlib.window_bits + 1;
2631|    137|          max_bits = 15;
2632|    137|          break;
2633|       |
2634|    791|        case ZLIB_STREAM_END:
2635|    791|          if (!zlib.global->optimize_zlib &&
2636|    791|              zlib.window_bits == zlib.file_bits &&
!zlib.cksum)
2637|    735|          {
```

2. Second Evidence: 어떻게 1번을 뚫었던 branch를 안뚫고 있다.

a. 기존

```
2714|      1|          default:
2715|       |            /* A fatal error; this happens if a too-far-back
error was
2716|       |             * hiding a more serious error, zlib_advance
has already
2717|       |             * output a zlib_message.
```

```
2718|      |                */
2719|     1|                zlib_end(&zlib);
2720|     1|                return 0;
```

b. target

```
2714|     0|            default:
2715|      |             /* A fatal error; this happens if a too-far-back
error was
2716|      |              * hiding a more serious error, zlib_advance
has already
2717|      |              * output a zlib_message.
2718|      |              */
2719|     0|              zlib_end(&zlib);
2720|     0|              return 0;
```

이외의 branch들은 대부분 증가된 count를 보여준다.

▼ zlib_advance(target)

- Function Calling count: 2.05k→1.98k

1. First Evidence: 이와 같이 suppress를 했음에도 enhance 효과가 나는 부분이 존재한다.
   a. 기존:약 107.3배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
2286|   220k|     switch (state)
2287|   220k|     {
2288|  2.05k|       case 0: /* first header byte */
2289|  2.05k|         {
```

   b. Target: 약 95.4배 차이나는 시도 횟수 대비 들어간 횟수

```
2286|   188k|     switch (state)
2287|   188k|     {
2288|  1.97k|       case 0: /* first header byte */
2289|  1.97k|         {
```

2. Second Evidence: 가끔 이렇게 엄청난 suppress도 보여주고는한다.
   a. 기존: 약 100.45배 차이나는 시도 횟수 대비 들어간 횟수

```
2370|   221k|          case Z_BUF_ERROR:
2371|   221k|            if (zlib->z.avail_out == 0)
2372|  2.20k|              continue; /* Try another output byte. */
```

b. Target: 약 23250배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
2370|   186k|          case Z_BUF_ERROR:
2371|   186k|            if (zlib->z.avail_out == 0)
2372|      8|              continue; /* Try another output byte. */
```

▼ process_chunk(target)

- Function Calling count: 5.5k→8.24k

1. First Evidence: 실행된 횟수가 많아서 그럴 수도 있지만, 새로운 branch를 뚫어냈다는 유의미한 결과가 있었습니다. 하지만 해당 Function은 Target function인 점을 인지해야합니다.

a. 기존:

```
2836|  5.50k|   if (type != png_IDAT)
2837|  4.64k|     file->alloc(file, 0/
chunk/);
2838|       |
2839|    862|   else if (file->idat == NULL)
2840|    862|     file->alloc(file, 1/
IDAT/);
2841|       |
2842|      0|   else
2843|      0|   {
2844|       |     /* The chunk length must be updated for
process_IDAT */
2845|      0|     assert(file->chunk != NULL);
2846|      0|     assert(file->chunk->chunk_type == png_IDAT);
2847|      0|     file->chunk->chunk_length = file->length;
2848|      0|   }
```

b. Target:

```
2836|  8.24k|   if (type != png_IDAT)
2837|  7.61k|     file->alloc(file, 0/
```

```
chunk/);
2838|      |
2839|   629|   else if (file->idat == NULL)
2840|   625|      file->alloc(file, 1/
IDAT/);
2841|      |
2842|     4|   else
2843|     4|   {
2844|      |      /* The chunk length must be updated for
process_IDAT */
2845|     4|      assert(file->chunk != NULL);
2846|     4|      assert(file->chunk->chunk_type == png_IDAT);
2847|     4|      file->chunk->chunk_length = file->length;
2848|     4|   }
```

2. Second Evidence: 원래 break까지 가지 못하던 Fuzzing이었는데 suppress가 적용되고나서 break까지 도달하고 있습니다.
   a. 기존

```
2913|   862|      case png_IDAT:
2914|   862|         if (process_IDAT(file))
2915|   589|            return;
2916|      |         /* First pass: */
2917|   273|         assert(next_type == png_IDAT);
2918|     0|         break;
```

   b. Target

```
2913|   629|      case png_IDAT:
2914|   629|         if (process_IDAT(file))
2915|   361|            return;
2916|      |         /* First pass: */
2917|   268|         assert(next_type == png_IDAT);
2918|     7|         break;
2919|  8.24k|   }
```

▼ read_callback(target)

- Function calling count: 19.2k → 34.4k

1. First Evidence: switch문 calling 횟수가 늘어가기는 했지만 비율적으로 따지자면 branch들이 덜 실행됐다.
    a. 기존:약 50.4배 차이나는 시도 횟수 대비 들어간 횟수

```
3290|   301k|     {
3291|  5.97k|       case 0: b = length >> 24; break;
3292|  5.97k|       case 1: b = length >> 16; break;
3293|  5.97k|       case 2: b = length >>  8; break;
3294|  5.97k|       case 3: b = length      ; break;
3295|       |
3296|  5.97k|       case 4: b = type >> 24; break;
3297|  5.97k|       case 5: b = type >> 16; break;
3298|  5.97k|       case 6: b = type >>  8; break;
3299|  5.97k|       case 7: b = type      ; break;
```

    b. target:약 61배 차이나는 시도 횟수 대비 들어간 횟수

```
3289|   540k|     switch (file→write_count)
3290|   540k|     {
3291|  8.85k|       case 0: b = length >> 24; break;
3292|  8.85k|       case 1: b = length >> 16; break;
3293|  8.85k|       case 2: b = length >>  8; break;
3294|  8.85k|       case 3: b = length      ; break;
3295|       |
3296|  8.85k|       case 4: b = type >> 24; break;
3297|  8.85k|       case 5: b = type >> 16; break;
3298|  8.85k|       case 6: b = type >>  8; break;
3299|  8.85k|       case 7: b = type      ; break;
```

2. Second Evidence: suppress가 적용되었지만 branch를 잘 뚫어 나갔다.
    a. 기존: 약 746배 차이나는 시도 횟수 대비 들어간 횟수

```
3410|   233k|             if (chunk→rewrite_length > 0)
3411|    312|             {
3412|    312|               if (chunk→rewrite_offset > 0)
3413|    224|                 --(chunk→rewrite_offset);
3414|       |
```

```
3415|    88|              else
3416|    88|              {
3417|    88|                b = chunk->rewrite_buffer[0];
3418|    88|                memmove(chunk->rewrite_buffer, chunk-
>rewrite_buffer+1,
3419|    88|                  (sizeof chunk->rewrite_buffer)-
3420|    88|                    (sizeof chunk->rewrite_buffer[0]));
3421|     |
3422|    88|                --(chunk->rewrite_length);
3423|    88|              }
3424|   312|            }
```

b. Target: 약 836배 차이나는 시도 횟수 대비 들어간 횟수

```
3410|  440k|            if (chunk->rewrite_length > 0)
3411|   526|            {
3412|   526|              if (chunk->rewrite_offset > 0)
3413|   416|                --(chunk->rewrite_offset);
3414|     |
3415|   110|              else
3416|   110|              {
3417|   110|                b = chunk->rewrite_buffer[0];
3418|   110|                memmove(chunk->rewrite_buffer, chunk-
>rewrite_buffer+1,
3419|   110|                  (sizeof chunk->rewrite_buffer)-
3420|   110|                    (sizeof chunk->rewrite_buffer[0]));
3421|     |
3422|   110|                --(chunk->rewrite_length);
3423|   110|              }
3424|   526|            }
```

▼ stop(non-target)

- Function calling count: 621→1.30k

1. First Evidence: 시도 횟수는 늘어나서 branch를 더 뚫기는 했는데, 그렇게 극적인 변화가 있지는 않다.

a. 기존

```
1139|    621|   if (file→global→quiet < 2) /* need two quiets to stop
this. /
1140|    621|   {
1141|    621|      png_uint_32 type;
1142|       |
1143|    621|      if (file→chunk != NULL)
1144|    414|         type = current_type(file, code); /
 Gropes in struct chunk and IDAT /
1145|       |
1146|    207|      else
1147|    207|         type = file→type;
1148|       |
1149|    621|      if (type)
1150|    611|         type_name(type, stdout);
1151|       |
1152|     10|      else /
 magic: an IDAT header, produces bogons for too many IDATs /
1153|     10|         fputs("HEAD", stdout); /
 not a registered chunk! */
```

b. Target

```
1143|  1.30k|      if (file→chunk != NULL)
1144|    610|         type = current_type(file, code); /* Gropes in struct
chunk and IDAT
/
1145|       |
1146|    690|      else
1147|    690|         type = file→type;
1148|       |
1149|  1.30k|      if (type)
1150|  1.28k|         type_name(type, stdout);
1151|       |
1152|     19|      else /
 magic: an IDAT header, produces bogons for too many IDATs /
```

1153| 19| fputs("HEAD", stdout); /
not a registered chunk! */

▼ process_iTxt(non-target)

- Function Calling count: 902→1.01k

1. First Evidence: 시도 횟수대비 뚫은 양은 기존 Fuzzing이 효과적이지만 branch를 뚫고 시도한 양은 suppress의 사이드 이펙트 영향이 효과적이었다.
   a. 기존:약 17.3배 차이나는 시도 횟수 대비 들어간 횟수

```
1666|  192|  while (length >= 9)
1667|  191|  {
1668|  191|    --length;
1669|  191|    ++index;
1670|  191|    if (reread_byte(file) == 0) /* keyword null terminator */
1671|  11|    {
1672|  11|      --length;
1673|  11|      ++index;
1674|  11|      (void)reread_byte(file); /* compression method */
1675|  11|      return zlib_check(file, index);
1676|  11|    }
1677|  191|  }
```

   b. target: 약 21.5배 차이나는 시도 횟수 대비 들어간 횟수

```
1666|  1.14k|  while (length >= 9)
1667|  1.14k|  {
1668|  1.14k|    --length;
1669|  1.14k|    ++index;
1670|  1.14k|    if (reread_byte(file) == 0) /* keyword null terminator */
1671|  53|    {
1672|  53|      --length;
1673|  53|      ++index;
1674|  53|      (void)reread_byte(file); /* compression method */
```

```
1675|     53|         return zlib_check(file, index);
1676|     53|     }
1677|  1.14k|   }
```

▼ cJSON(target 1개)

target 1개: parse_string

- parse_array에서 원래 뚫어내지 못했던 branch를 뚫었다.

▼ parse_string(target)

- Function calling count: 27.7k→26.1k(약 1.06배 감소)

1. First Evidence
   a. 기존: 약 33.7배 차이나는 시도 횟수 대비 들어간 횟수

```
786|  27.7k|    if (buffer_at_offset(input_buffer)[0] != '\"')
787|     82|    {
788|     82|        goto fail;
789|     82|    }
```

   b. Target: 약 41.4배 차이나는 시도 횟수 대비 들어간 횟수(WIN)

```
787|  26.1k|    if (buffer_at_offset(input_buffer)[0] != '\"')
788|     63|    {
789|     63|        goto fail;
790|     63|    }
```

2. Second Evidence: Enhance에서는 switch의 branch가 불확실하게 커버됐지만 suppress에서는 딱 suppress에 맞게 커버하였다.
   a. 기존

```
841|  17.5k|          switch (input_pointer[1])
842|  17.5k|          {
843|    276|              case 'b':
844|    276|                  *output_pointer++ = '\b';
845|    276|                  break;
846|    175|              case 'f':
847|    175|                  *output_pointer++ = '\f';
```

```
848|    175|                break;
849|    291|              case 'n':
850|    291|                *output_pointer++ = '\n';
851|    291|                break;
852|    236|              case 'r':
853|    236|                *output_pointer++ = '\r';
854|    236|                break;
855|    216|              case 't':
856|    216|                *output_pointer++ = '\t';
857|    216|                break;
858|  1.99k|              case '\"':
859|  5.82k|              case '\\':
860|  5.82k|              case '/':
861|  5.82k|
output_pointer++ = input_pointer[1];
862|  5.82k|                break;
863|       |
864|       |              /
 UTF-16 literal */
865|  10.4k|              case 'u':
```

b. Target

```
842|  8.16k|          switch (input_pointer[1])
843|  8.16k|          {
844|    314|            case 'b':
845|    314|              *output_pointer++ = '\b';
846|    314|              break;
847|     59|            case 'f':
848|     59|              *output_pointer++ = '\f';
849|     59|              break;
850|     40|            case 'n':
851|     40|              *output_pointer++ = '\n';
852|     40|              break;
853|     37|            case 'r':
854|     37|              *output_pointer++ = '\r';
855|     37|              break;
```

```
856|    228|             case 't':
857|    228|                 *output_pointer++ = '\t';
858|    228|                 break;
859|  1.12k|             case '\"':
860|  5.90k|              case '\\':
861|  5.90k|             case '/':
862|  5.90k|
output_pointer++ = input_pointer[1];
863|  5.90k|                 break;
864|       |
865|       |             /
 UTF-16 literal */
866|  1.53k|             case 'u':
```

▼ print_value(non-target)

- Function calling count: 20.2k→13.8k

1. First evidence: 숫자만 본다면 Target과 관련 없는 함수이지만 suppress됐다고 판단할 수 있다. suppress가 된 것은 맞으나 target function의 suppress의 정도와 비교한다면 오히려 적은 시도로 branch를 뚫어낸 것일 수도 있다.
    a. 기존: 약 8.38배 차이나는 시도 횟수 대비 들어간 횟수

    ```
    1381|  20.2k|    switch ((item→type) & 0xFF)
    1382|  20.2k|    {
    1383|    241|        case cJSON_NULL
    ```

    b. Target: 약 10.45배 차이나는 시도 횟수 대비 들어간 횟수

    ```
    1382|  13.8k|    switch ((item→type) & 0xFF)
    1383|  13.8k|    {
    1384|    132|        case cJSON_NULL:
    ```

2. Second Evidence: 해당 switch문의 실행횟수가 확연히 7.2k 적음에도 불구하고 적은 횟수로 비슷한 coverage를 보여주었다. 하지만 cJSON_Array는 아쉽다.
    a. 기존:

```
1431|  7.52k|        case cJSON_String:
1432|  7.52k|            return print_string(item, output_buffer);
1433|       |
1434|  3.99k|        case cJSON_Array:
1435|  3.99k|            return print_array(item, output_buffer);
1436|       |
1437|  3.15k|        case cJSON_Object:
1438|  3.15k|            return print_object(item, output_buffer);
```

b. Target

```
1432|  7.30k|        case cJSON_String:
1433|  7.30k|            return print_string(item, output_buffer);
1434|       |
1435|    558|        case cJSON_Array:
1436|    558|            return print_array(item, output_buffer);
1437|       |
1438|  2.41k|        case cJSON_Object:
1439|  2.41k|            return print_object(item, output_buffer);
```

▼ parse_array(non-target)

- function calling count: 12.4k→7.90(약 1.56배 감소)

1. First Evidence: 더 적은 횟수로 뚫지 못했던 branch를 뚫었다.

a. 기존

```
1451|  12.4k|    if (input_buffer->depth >= CJSON_NESTING_LIMIT)
1452|      0|    {
1453|      0|        return false; /* to deeply nested */
1454|      0|    }
```

b. Target

```
1452|  7.90k|    if (input_buffer->depth >= CJSON_NESTING_LIMIT)
1453|      5|    {
1454|      5|        return false; /* to deeply nested */
1455|      5|    }
```

2. Second Evidence: 왜 이렇게 많이 차이나는지 확인해봤는데 success뒤에는 의미있는 branch가 없었다. 그래서 이만큼만 시도한게 아닐까 싶다.

a. 기존: 약 3.22배 차이나는 시도 횟수 대비 들어간 횟수

```
1465|  12.4k|    if (can_access_at_index(input_buffer, 0) &&
(buffer_at_offset(input_buffer)[0] == ']'))
1466|  3.80k|    {
1467|       |        /* empty array */
1468|  3.80k|        goto success;
1469|  3.80k|    }
```

b. Target: 약 131.5배 차이나는 시도 횟수 대비 들어간 횟수

```
1466|  7.89k|    if (can_access_at_index(input_buffer, 0) &&
(buffer_at_offset(input_buffer)[0] == ']'))
1467|     60|    {
1468|       |        /* empty array */
1469|     60|        goto success;
1470|     60|    }
```

다른 함수들도 비슷

▼ libpng(target 1개)

target: process_iTXt

process_zTXt_iCCP

process_chunk

- 결과: target function에서 suppress된 곳이 군데군데 보이기는 하지만 enhance도 마찬가지로 군데군데 반전현상이 있기에 suppress되었다고 판단할 수 없다. 하지만, 다른 function들에서 branch를 많이 커버하고 원래 못가던 path를 뚫기도하는 결과를 보여줬다.
  target function은 suppress되지 않은거 같은데, 다른 Function들은 커버가 잘되었다.
  - 근거
    - Function calling count는 줄어 들었지만, 이 count는 fuzzing의 환경에 따라서 그때의 상황에 따라서 달라질 수 있는 것이기 때문에 function calling count가 줄었다는 것은 의미가 없다.

- 내부 branch coverage count가 얼마나 줄었느냐를 볼 때에는 Function calling count의 차이를 고려해서 비율로 증가폭, 감소폭을 계산해야한다. 이렇게 계산했을 때 보통 약 1.3배 정도 더 증가한 결과를 보였다.
- 다른 function들을 확인해보면 cover가 잘되었다.

▼ process_iTxt(Target function)

- Funtion calling count: 1.17k → 902(약 1.3배 감소)

1. First evidence: 시도횟수대비 더 많이 들어간 것을 볼 수 있다. Target에 Function calling count를 반영했을 때는 약 13.4배이다.
   a. 기존: 약 10.8배 차이나는 시도 횟수 대비 들어간 횟수

```
1699|  12.6k|      if (reread_byte(file) == 0) /* keyword null terminator */
1700|  1.16k|      {
1701|  1.16k|        --length;
1702|  1.16k|        ++index;
```

   b. Target: 약 10.3배 차이나는 시도 횟수 대비 들어간 횟수

```
1699|  9.09k|      if (reread_byte(file) == 0) /* keyword null terminator */
1700|    879|      {
1701|    879|        --length;
1702|    879|        ++index;
```

2. Second evidence: 시도횟수대비 더 많이 들어갔다. Function calling count 반영안해도 이미 많이 들어갔다.
   a. 기존: 약 4.14배차이나는 시도 횟수 대비 들어간 횟수

```
1722|  4.60k|              if (reread_byte(file) == 0) /* terminator */
1723|  1.08k|                return zlib_check(file, index);
```

   b. Target: 약 2.63배차이나는 시도 횟수 대비 들어간 횟수

```
1722|  2.20k|              if (reread_byte(file) == 0) /* terminator */
1723|    836|                return zlib_check(file, index);
```

▼ calc_image_size(side effected function)

- Function calling count: 1.73k → 1.10k (약 1.5배 감소)

1. First evidence: switch문을 시도하는 횟수는 차이나지만 branch를 뚫어내는 횟수는 크게 다르지 않다. 비율적으로 보았을 때 오히려 path를 잘 뚫었다. 하지만, weight를 가했을 때 switch문에서 나타나는 현상이 만만한 branch에 집중적으로 들어간다는 것다. 그래서 특정 branch만 유독 count가 높은 상황이 있다.

   a. 기존

```
1410|  1.73k|   switch (file->color_type)
1411|  1.73k|   {
1412|      5|     default:
1413|      5|       stop_invalid(file, "IHDR: colour type");
1414|       |
1415|     31|     invalid_bit_depth:
1416|     31|       stop_invalid(file, "IHDR: bit depth");
1417|       |
1418|  1.00k|     case 0: /* g */
1419|  1.00k|       if (pd != 1 && pd != 2 && pd != 4 && pd != 8 &&
pd != 16)
1420|      9|         goto invalid_bit_depth;
1421|    995|       break;
1422|       |
1423|    995|     case 3:
1424|    644|       if (pd != 1 && pd != 2 && pd != 4 && pd != 8)
1425|      6|         goto invalid_bit_depth;
1426|    638|       break;
```

   b. Target

```
1410|  1.10k|   switch (file->color_type)
1411|  1.10k|   {
1412|      5|     default:
1413|      5|       stop_invalid(file, "IHDR: colour type");
1414|       |
1415|     25|     invalid_bit_depth:
```

```
1416|     25|        stop_invalid(file, "IHDR: bit depth");
1417|       |
1418|    947|     case 0: /* g */
1419|    947|       if (pd != 1 && pd != 2 && pd != 4 && pd != 8 &&
pd != 16)
1420|      5|         goto invalid_bit_depth;
1421|    942|       break;
1422|       |
1423|    942|     case 3:
1424|     40|       if (pd != 1 && pd != 2 && pd != 4 && pd != 8)
1425|      5|         goto invalid_bit_depth;
1426|     35|       break;
```

2. Secnode Evidence: 해당 branch가 유독 suppress가 잘된 branch이다. 이때 유의할 점은 weight를 가했을 때 특정 branch 집중 현상을 보였어서 이에 대한 영향일 수도 있다.
    a. 기존: 약 5배차이나는 시도 횟수 대비 들어간 횟수

```
1462|  1.66k|   else switch (file→interlace_method)
1463|  1.66k|   {
1464|    326|     case PNG_INTERLACE_ADAM7:
```

    b. Target: 약 18배차이나는 시도 횟수 대비 들어간 횟수

```
1462|  1.05k|   else switch (file→interlace_method)
1463|  1.05k|   {
1464|     58|     case PNG_INTERLACE_ADAM7:
```

▼ stop(side effected function)

- Function calling count: 1.21k → 621

1. First Evidence: Target이 비율적으로 더 많이 들어간다.
    a. 기존: 약 2배차이나는 시도 횟수 대비 들어간 횟수

```
1143|  1.21k|     if (file→chunk != NULL)
1144|    578|       type = current_type(file, code); /* Gropes in
struct chunk and IDA
```

    b. Target: 약 1.5배차이나는 시도 횟수 대비 들어간 횟수

1143| 621| if (file→chunk != NULL)
1144| 414| type = current_type(file, code); /* Gropes in struct chunk and IDAT */

2. Second Evidence: Target이 비율적으로 더 많이 들어간다.
    a. 기존: 약 67배차이나는 시도 횟수 대비 들어간 횟수

1150| 1.20k| type_name(type, stdout);
1151| |
1152| 18| else /* magic: an IDAT header, produces bogons for too many IDATs */

    b. Target: 약 62배차이나는 시도 횟수 대비 들어간 횟수

1150| 611| type_name(type, stdout);
1151| |
1152| 10| else /* magic: an IDAT header, produces bogons for too many I

▼ process_chunk(side effected function)

- Function calling count: 5.50k→8.95k

1. First Evidence: 기존에 뚫지 못했던 branch를 cover해냈다.
    a. 기존

2836| 5.50k| if (type != png_IDAT)
2837| 4.64k| file→alloc(file, 0/*chunk*/);
2838| |
2839| 862| else if (file→idat == NULL)
2840| 862| file→alloc(file, 1/*IDAT*/);
2841| |
2842| 0| else
2843| 0| {
2844| | /* The chunk length must be updated for process_IDAT */
2845| 0| assert(file→chunk != NULL);
2846| 0| assert(file→chunk→chunk_type == png_IDAT);

```
2847|      0|     file->chunk->chunk_length = file->length;
2848|      0|   }
```

b. Target

```
2836|  8.95k|   if (type != png_IDAT)
2837|  8.12k|     file->alloc(file, 0/
chunk/);
2838|       |
2839|    828|   else if (file->idat == NULL)
2840|    822|     file->alloc(file, 1/
IDAT/);
2841|       |
2842|      6|   else
2843|      6|   {
2844|       |     /* The chunk length must be updated for
process_IDAT */
2845|      6|     assert(file->chunk != NULL);
2846|      6|     assert(file->chunk->chunk_type == png_IDAT);
2847|      6|     file->chunk->chunk_length = file->length;
2848|      6|   }
```

2. Second Evidence: Target이 걸린 프로그램에서 더 많이 커버하였다.

a. 기존: 약 91.6배 차이나는 시도 횟수 대비 들어간 횟수

```
2899|     12|     case png_zTXt: case png_iCCP:
2900|     12|       if (process_zTXt_iCCP(file))
2901|      0|         return;
2902|     12|       chunk_end(&file->chunk);
2903|     12|       file_setpos(file, &file->data_pos);
2904|     12|       break;
```

b. Target: 약 18.2배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
2899|     95|     case png_zTXt: case png_iCCP:
2900|     95|       if (process_zTXt_iCCP(file))
2901|      0|         return;
2902|     95|       chunk_end(&file->chunk);
```

```
2903|    95|        file_setpos(file, &file→data_pos);
2904|    95|        break;
```

▼ process_zTXt_iCCP

- Function Calling count: 12 → 95

1. First Evidence
    a. 기존

```
1666|   192|   while (length >= 9)
1667|   191|   {
1668|   191|     --length;
1669|   191|     ++index;
1670|   191|     if (reread_byte(file) == 0) /* keyword null
terminator
/
1671|    11|     {
1672|    11|       --length;
1673|    11|       ++index;
1674|    11|       (void)reread_byte(file); /
 compression method */
1675|    11|       return zlib_check(file, index);
1676|    11|     }
1677|   191|   }
```

    b. Target

```
1666|  1.69k|   while (length >= 9)
1667|  1.68k|   {
1668|  1.68k|     --length;
1669|  1.68k|     ++index;
1670|  1.68k|     if (reread_byte(file) == 0) /* keyword null
terminator
/
1671|    85|     {
1672|    85|       --length;
1673|    85|       ++index;
1674|    85|       (void)reread_byte(file); /
```

```
 compression method */
1675|     85|         return zlib_check(file, index);
1676|     85|     }
1677|  1.68k|   }
```

Enhance

▼ cJSON(target 5개)

- Target function을 5개로 했을 때(target : parse_number, utf16, parse_string, print_string_ptr, parse_value)
- Target1보다 드라마틱한 결과를 보여주지는 않는다. Target function이라고 할지라도 별로 좋지 않은 branch coverage를 보여주고는 한다. 그렇지만 좋은 경우도 있다.
- 전체적으로 enhance된 함수들도 있지만, weight에 의해 선택받지 못한 testcase가 원래 실행시키던 함수를 실행 못시키게 되어서, 이 영향이 target function에 미치기도 하였다. 그래서 오히려 coverage가 줄어든 target function도 있었다.
- 전반적으로 10%정도의 개선사항이 있는 거 같다.

▼ parse_number(Target)

- Function calling count: 10.5k → 7.06k (약 1.48배 감소)

1. First Evidence: 0.6배 정도 개입이 줄기는 했지만 function calling count를 고려하고 Fuzzing의 랜덤성을 고려한다면 마냥 개입이 줄었다고 생각할 수는 없다.
    a. 기존: 약 1.8배 차이나는 시도 횟수 대비 들어간 횟수(branch count들 평균내서 44.7k에서 나눔.)

```
322|  44.7k|   {
323|  44.7k|       switch (buffer_at_offset(input_buffer)[i])
324|  44.7k|       {
325|  13.4k|           case '0':
326|  21.4k|           case '1':
327|  24.2k|           case '2':
328|  25.5k|           case '3':
329|  26.1k|           case '4':
330|  26.9k|           case '5':
331|  27.8k|           case '6':
```

```
332|  28.8k|          case '7':
333|  29.7k|          case '8':
334|  31.2k|          case '9':
335|  31.2k|          case '+':
336|  32.7k|          case '-':
337|  32.8k|          case 'e':
338|  33.7k|          case 'E':
339|  33.7k|              number_c_string[i] =
buffer_at_offset(input_buffer)[i];
340|  33.7k|              break;
341|       |
342|    584|          case '.':
343|    584|              number_c_string[i] = decimal_point;
344|    584|              break;
345|       |
346|  10.4k|          default:
347|  10.4k|              goto loop_end;
```

b. Target: 약 1.74배 차이나는 시도 횟수 대비 들어간 횟수

```
322|  39.5k|    {
323|  39.5k|        switch (buffer_at_offset(input_buffer)[i])
324|  39.5k|        {
325|  5.68k|          case '0':
326|  9.67k|          case '1':
327|  24.8k|          case '2':
328|  25.6k|          case '3':
329|  25.8k|          case '4':
330|  26.9k|          case '5':
331|  27.6k|          case '6':
332|  28.3k|          case '7':
333|  28.7k|          case '8':
334|  29.5k|          case '9':
335|  29.5k|          case '+':
336|  30.8k|          case '-':
337|  31.2k|          case 'e':
338|  31.2k|          case 'E':
```

```
339|  31.2k|                number_c_string[i] =
buffer_at_offset(input_buffer)[i];
340|  31.2k|                break;
341|       |
342|  1.26k|            case '.':
343|  1.26k|                number_c_string[i] = decimal_point;
344|  1.26k|                break;
345|       |
346|  7.05k|            default:
347|  7.05k|                goto loop_end;
```

2. Second Evidence
    - 첫 번째 branch
        - 기존: 약 1.17배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)
        - Target: 약 6.95배 차이나는 시도 횟수 대비 들어간 횟수
    - 두 번째 branch
        - 기존: 약 20배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)
        - Target: 약 120배 차이나는 시도 횟수 대비 들어간 횟수

    a. 기존

```
362|  10.5k|    if (number >= INT_MAX)
363|    940|    {
364|    940|        item->valueint = INT_MAX;
365|    940|    }
366|  9.56k|    else if (number <= (double)INT_MIN)
367|    466|    {
368|    466|        item->valueint = INT_MIN;
369|    466|    }
370|  9.09k|    else
371|  9.09k|    {
372|  9.09k|        item->valueint = (int)number;
373|  9.09k|    }
```

    b. Target

```
362|  7.02k|    if (number >= INT_MAX)
363|  1.01k|    {
364|  1.01k|        item->valueint = INT_MAX;
365|  1.01k|    }
366|  6.00k|    else if (number <= (double)INT_MIN)
367|     50|    {
368|     50|        item->valueint = INT_MIN;
369|     50|    }
370|  5.95k|    else
371|  5.95k|    {
372|  5.95k|        item->valueint = (int)number;
373|  5.95k|    }
```

▼ utf16_literal_to_utf8(Target)

- Function calling count: 10.4k → 10.0k (약 1.04배 감소)

1. First Evidence: 비슷한 시도횟수지만 Target한게 더 잘 들어갔다.
    a. 기존: 약 5.94배 차이나는 시도 횟수 대비 들어간 횟수

```
683|  10.4k|    if ((first_code >= 0xD800) && (first_code <= 0xDBFF))
684|    175|    {
685|    175|        const unsigned char *second_sequence = first_sequence + 6;
686|    175|        unsigned int second_code = 0;
687|    175|        sequence_length = 12; /* \uXXXX\uXXXX */
```

    b. Target: 약 2.3배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
684|  10.0k|    if ((first_code >= 0xD800) && (first_code <= 0xDBFF))
685|    434|    {
686|    434|        const unsigned char *second_sequence = first_sequence + 6;
687|    434|        unsigned int second_code = 0;
688|    434|        sequence_length = 12; /* \uXXXX\uXXXX */
```

2. Second Evidence:
    a. 기존: 약 4.49배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
728|  2.45k|    else if (codepoint < 0x800)
729|    545|    {
```

    b. Target: 약 6.23배 차이나는 시도 횟수 대비 들어간 횟수

```
729|    979|    else if (codepoint < 0x800)
730|    157|    {
```

▼ parse_string(Target)

- Function Calling count: 27.7k → 26.3k(약 1.05배 감소)

1. First Evidence
    a. 기존: 약 21.4배 차이나는 시도 횟수 대비 들어간 횟수

```
798|   409k|          if (input_end[0] == '\\')
799|  19.1k|          {
800|  19.1k|              if ((size_t)(input_end + 1 - input_buffer-
>content) >= input_buffer->length)
```

    b. Target: 약 19.17배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
799|   418k|          if (input_end[0] == '\\')
800|  21.8k|          {
801|  21.8k|              if ((size_t)(input_end + 1 - input_buffer-
>content) >= input_buffer->length)
```

2. Second Evidence: 특정 branch에 대해서는 잘들어가고 특정 branch는 오히려 못들어감. 특이사항은 일반 fuzzing처럼 골고루 branch count를 나눠갖는 것이 아닌, 특정 branch에 몰아서 들어가는 경향을 보임.
    a. 기존

```
841|  17.5k|          switch (input_pointer[1])
842|  17.5k|          {
843|    276|              case 'b':
844|    276|                  *output_pointer++ = '\b';
845|    276|                  break;
846|    175|              case 'f':
```

```
847|    175|                *output_pointer++ = '\f';
848|    175|                break;
849|    291|              case 'n':
850|    291|                *output_pointer++ = '\n';
851|    291|                break;
852|    236|              case 'r':
853|    236|                *output_pointer++ = '\r';
854|    236|                break;
855|    216|              case 't':
856|    216|                *output_pointer++ = '\t';
857|    216|                break;
858|  1.99k|              case '\"':
859|  5.82k|              case '\\':
860|  5.82k|              case '/':
861|  5.82k|
output_pointer++ = input_pointer[1];
862|  5.82k|                break;
863|       |
864|       |            /
 UTF-16 literal */
865|  10.4k|              case 'u':
```

b. Target

```
842|  19.4k|          switch (input_pointer[1])
843|  19.4k|          {
844|     76|            case 'b':
845|     76|              *output_pointer++ = '\b';
846|     76|              break;
847|    132|            case 'f':
848|    132|              *output_pointer++ = '\f';
849|    132|              break;
850|    355|            case 'n':
851|    355|              *output_pointer++ = '\n';
852|    355|              break;
853|     99|            case 'r':
854|     99|              *output_pointer++ = '\r';
```

```
855|    99|                 break;
856|   185|             case 't':
857|   185|                 *output_pointer++ = '\t';
858|   185|                 break;
859|   531|             case '\"':
860|  8.39k|              case '\\':
861|  8.40k|             case '/':
862|  8.40k|
output_pointer++ = input_pointer[1];
863|  8.40k|                 break;
864|      |
865|      |             /
 UTF-16 literal */
866|  10.0k|             case 'u':
```

▼ print_string_ptr(Target)

- Function calling count: 18.3k → 19.6k(약 1.07배 증가)

1. First Evidence: Enhance에 됐다고 볼 수 없는 branch의 증가량이다. 여기서도 특정한 branch에 몰아서 힘을 쓰는 현상을 보인다.(default)
    a. 기존

```
937|   179k|       switch (input_pointer)
938|   179k|       {
939|    804|           case '\"':
940|  2.15k|           case '\\':
941|  4.43k|           case '\b':
942|  7.44k|           case '\f':
943|  10.7k|           case '\n':
944|  12.5k|           case '\r':
945|  14.5k|           case '\t':
946|      |           /
 one character escape sequence */
947|  14.5k|             escape_characters++;
948|  14.5k|             break;
949|   164k|         default:
```

b. Target

```
938|   247k|       switch (input_pointer)
939|   247k|       {
940|    329|          case '\"':
941|  6.15k|          case '\\':
942|  6.18k|          case '\b':
943|  6.31k|          case '\f':
944|  7.36k|          case '\n':
945|  7.43k|          case '\r':
946|  8.08k|          case '\t':
947|       |          /
 one character escape sequence */
948|  8.08k|             escape_characters++;
949|  8.08k|             break;
950|   239k|          default:
```

2. Second Evidence: First Evidence와 마찬가지의 상황.

a. 기존

```
991|  17.5k|          switch (*input_pointer)
992|  17.5k|          {
993|  1.34k|             case '\\':
994|  1.34k|                *output_pointer = '\\';
995|  1.34k|                break;
996|    804|             case '\"':
997|    804|                *output_pointer = '\"';
998|    804|                break;
999|  2.28k|             case '\b':
1000|  2.28k|                *output_pointer = 'b';
1001|  2.28k|                break;
1002|  3.01k|             case '\f':
1003|  3.01k|                *output_pointer = 'f';
1004|  3.01k|                break;
1005|  3.29k|             case '\n':
1006|  3.29k|                *output_pointer = 'n';
1007|  3.29k|                break;
```

```
1008|  1.82k|            case '\r':
1009|  1.82k|                *output_pointer = 'r';
1010|  1.82k|                break;
1011|  2.00k|            case '\t':
1012|  2.00k|                *output_pointer = 't';
1013|  2.00k|                break;
1014|  3.03k|            default:
```

b. Target

```
992|  11.8k|        switch (*input_pointer)
993|  11.8k|        {
994|  5.82k|            case '\\':
995|  5.82k|                *output_pointer = '\\';
996|  5.82k|                break;
997|   329|            case '\"':
998|   329|                *output_pointer = '\"';
999|   329|                break;
1000|    32|            case '\b':
1001|    32|                *output_pointer = 'b';
1002|    32|                break;
1003|   134|            case '\f':
1004|   134|                *output_pointer = 'f';
1005|   134|                break;
1006|  1.04k|            case '\n':
1007|  1.04k|                *output_pointer = 'n';
1008|  1.04k|                break;
1009|    69|            case '\r':
1010|    69|                *output_pointer = 'r';
1011|    69|                break;
1012|   647|            case '\t':
1013|   647|                *output_pointer = 't';
1014|   647|                break;
1015|  3.80k|            default:
```

▼ parse_value(Target)

- Function calling count: 40.4k → 29.4k (약 1.37배 감소)

1. First Evidence
    a. 기존: 약 11.25배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

        1326| 40.4k| if (can_read(input_buffer, 4) && (strncmp((const char*)buffer_at_offset(input_buffer), "null", 4) == 0))
        1327| 359| {

    b. Target: 약 18.14배 차이나는 시도 횟수 대비 들어간 횟수

        1327| 29.4k| if (can_read(input_buffer, 4) && (strncmp((const char*)buffer_at_offset(input_buffer), "null", 4) == 0))
        1328| 162| {

2. Second Evidence
    a. 기존: 약 25.38배 차이나는 시도 횟수 대비 들어간 횟수

        1340| 39.6k| if (can_read(input_buffer, 4) && (strncmp((const char*)buffer_at_offset(input_buffer), "true", 4) == 0))
        1341| 156| {

    b. Target: 약 13.74배 차이나는 시도 횟수 대비 들어간 횟수(WIN)

        1341| 29.0k| if (can_read(input_buffer, 4) && (strncmp((const char*)buffer_at_offset(input_buffer), "true", 4) == 0))
        1342| 211| {

▼ parse_array(non-Target)

- Function calling count: 12.4→7.11k

1. First evidence: 새로운 path를 뚫으려는 시도가 확연히 작아짐
    a. 기존: 약 3.26배 차이나는 시도 횟수 대비 들어간 횟수

        1465| 12.4k| if (can_access_at_index(input_buffer, 0) && (buffer_at_offset(input_buffer)[0] == ']'))
        1466| 3.80k| {
        1467| | /* empty array */
        1468| 3.80k| goto success;
        1469| 3.80k| }

    b. Target: 약 58배 차이나는 시도 횟수 대비 들어간 횟수

```
 1466|  7.10k|    if (can_access_at_index(input_buffer, 0) &&
(buffer_at_offset(input_buffer)[0] == ']'))
1467|    121|    {
1468|      |        /* empty array */
1469|    121|        goto success;
1470|    121|    }
```

2. Second Evidence: 새로운 path를 뚫으려는 시도가 확연히 작아짐
    a. 기존: 약 1.63배 차이나는 시도 횟수 대비 들어간 횟수

```
1491|  22.1k|        if (head == NULL)
1497|  13.5k|        {
1498|      |            /* add to the end and advance */
1499|  13.5k|            current_item->next = new_item;
1500|  13.5k|            new_item->prev = current_item;
1501|  13.5k|            current_item = new_item;
1502|  13.5k|        }
```

    b. Target: 약 2.33배 차이나는 시도 횟수 대비 들어간 횟수

```
1492|  12.2k|        if (head == NULL)
1498|  5.23k|        {
1499|      |            /* add to the end and advance */
1500|  5.23k|            current_item->next = new_item;
1501|  5.23k|            new_item->prev = current_item;
1502|  5.23k|            current_item = new_item;
1503|  5.23k|        }
```

▼ print_object(non_Target)

- Function calling count: 3.15k→2.51k

1. First Evidence
    a. 기존: 약 1.12배 차이나는 시도 횟수 대비 들어간 횟수

```
1777|  10.8k|        if (output_buffer->format)
1778|  9.57k|        {
1779|  9.57k|            *output_pointer++ = '\t';
1780|  9.57k|        }
```

b. Target: 약 1.18배 차이나는 시도 횟수 대비 들어간 횟수

```
1783|  11.4k|       if (output_buffer→format)
1784|  9.65k|       {
1785|  9.65k|          *output_pointer++ = '\t';
1786|  9.65k|       }
```

▼ libpng(target 5개)

- target 5개: calc_image_size, zlib_check, zlib_advance, process_chunk, read_callback)
- 결과가 항상 suppress하게 일정하지는 않다. 하지만 suppress된 결과를 보여주는 것은 맞다.

▼ calc_image_size(target)

- Function calling count: 1.10k → 1.81k (약 1.64배)

1. First Function: 기존 Fuzzing에서 골고루 들어가지 못했던 switch문에 대하여 골고루 들어갈 수 있게 해주는 역할을 했음.

   a. 기존

```
1418|    947|     case 0: /* g /
1419|    947|       if (pd != 1 && pd != 2 && pd != 4 && pd != 8 && pd
!= 16)
1420|      5|         goto invalid_bit_depth;
1421|    942|       break;
1422|       |
1423|    942|     case 3:
1424|     40|       if (pd != 1 && pd != 2 && pd != 4 && pd != 8)
1425|      5|         goto invalid_bit_depth;
1426|     35|       break;
1427|       |
1428|     62|     case 2: /
 rgb /
1429|     62|       if (pd != 8 && pd != 16)
1430|      5|         goto invalid_bit_depth;
1431|       |
1432|     57|       pd = (png_uint_16)(pd * 3);
```

```
1433|    57|        break;
1434|     |
1435|    11|    case 4: /
 ga /
1436|    11|      if (pd != 8 && pd != 16)
1437|     5|        goto invalid_bit_depth;
1438|     |
1439|     6|      pd = (png_uint_16)(pd * 2);
1440|     6|      break;
1441|     |
1442|    35|    case 6: /
 rgba */
1443|    35|      if (pd != 8 && pd != 16)
1444|     5|        goto invalid_bit_depth;
```

b. Target

```
1418|   497|    case 0: /* g /
1419|   497|      if (pd != 1 && pd != 2 && pd != 4 && pd != 8 && pd
!= 16)
1420|     7|        goto invalid_bit_depth;
1421|   490|      break;
1422|     |
1423|   627|    case 3:
1424|   627|      if (pd != 1 && pd != 2 && pd != 4 && pd != 8)
1425|     6|        goto invalid_bit_depth;
1426|   621|      break;
1427|     |
1428|   659|    case 2: /
 rgb /
1429|   659|      if (pd != 8 && pd != 16)
1430|     6|        goto invalid_bit_depth;
1431|     |
1432|   653|      pd = (png_uint_16)(pd * 3);
1433|   653|      break;
1434|     |
1435|    11|    case 4: /
```

```
 ga /
1436|    11|        if (pd != 8 && pd != 16)
1437|     5|          goto invalid_bit_depth;
1438|      |
1439|     6|        pd = (png_uint_16)(pd * 2);
1440|     6|        break;
1441|      |
1442|    15|     case 6: /
 rgba */
1443|    15|        if (pd != 8 && pd != 16)
1444|     5|          goto invalid_bit_depth;
```

2. Second Evidence
    a. 기존: 약 18배 차이나는 시도 횟수 대비 들어간 횟수

```
1463|  1.05k|   {
1464|     58|      case PNG_INTERLACE_ADAM7:
```

    b. Target: 약 4.48배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
1463|  1.74k|   {
1464|    388|      case PNG_INTERLACE_ADAM7:
```

▼ zlib_check(target)

- Function calling count: 1.70k → 1.95k

1. First Evidence
    a. 기존: 약 16배 차이나는 시도 횟수 대비 들어간 횟수

```
2625|  1.70k|      switch (rc)
2626|  1.70k|      {
2627|    105|        case ZLIB_TOO_FAR_BACK:
```

    b. Target: 약 11배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
2625|  1.95k|      switch (rc)
2626|  1.95k|      {
2627|    168|        case ZLIB_TOO_FAR_BACK:
```

2. Second Evidence

a. 기존: 약 3배 차이나는 시도 횟수 대비 들어간 횟수

```
2692|    298|              if (min_bits > max_bits)
2693|     99|              {
```

b. Target: 약 2.6배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
2692|    435|              if (min_bits > max_bits)
2693|    166|              {
```

▼ zlib_advance(target)

- Function calling count: 2.05k → 2.43k (약 1.18배 증가)

1. First Evidence

a. 기존: 약 4배 차이나는 시도 횟수 대비 들어간 횟수

```
2319|  2.04k|            if (bln != b2)
2320|    507|
```

b. Target: 약 3.3배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
2319|  2.41k|            if (bln != b2)
2320|    719|            {
```

2. Second Evidence

a. 기존: 약 553,333배 차이나는 시도 횟수 대비 들어간 횟수

```
 2368|  49.8M|        switch (zlib→rc)
2395|      9|          case Z_NEED_DICT:
2396|      9|            zlib_message(zlib, 0/
stream error/);
2397|      9|            endrc = ZLIB_FATAL;
2398|      9|            break;
```

b. Target: 약 456,666배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

```
 2368|  68.5M|        switch (zlib→rc)
2395|     15|          case Z_NEED_DICT:
2396|     15|            zlib_message(zlib, 0/
stream error/);
2397|     15|            endrc = ZLIB_FATAL;
2398|     15|            break;
```

▼ process_chunk(target)

- Function calling count: 5.50k → 9.40k

1. First Evidence
   a. 기존:약 4.36배 차이나는 시도 횟수 대비 들어간 횟수

      2798| 5.50k| if ((file→crc ^ 0xffffffff) != file_crc)
      2799| 1.26k| {

   b. Target:약 2.94배 차이나는 시도 횟수 대비 들어간 횟수 (WIN)

      2798| 9.40k| if ((file→crc ^ 0xffffffff) != file_crc)
      2799| 3.19k| {

2. Second Evidence: 뚫지 못했던 branch를 뚫었다.
   a. 기존

      2836| 5.50k| if (type != png_IDAT)
      2837| 4.64k| file→alloc(file, 0/*chunk*/);
      2838| | |
      2839| 862| else if (file→idat == NULL)
      2840| 862| file→alloc(file, 1/*IDAT*/);
      2841| | |
      2842| 0| else
      2843| 0| {
      2844| | /* The chunk length must be updated for process_IDAT */
      2845| 0| assert(file→chunk != NULL);
      2846| 0| assert(file→chunk→chunk_type == png_IDAT);
      2847| 0| file→chunk→chunk_length = file→length;
      2848| 0| }

   b. Target

      2836| 9.40k| if (type != png_IDAT)
      2837| 8.51k| file→alloc(file, 0/*chunk*/);

```
2838|      |
2839|    896|   else if (file→idat == NULL)
2840|    886|     file→alloc(file, 1/
IDAT/);
2841|      |
2842|     10|   else
2843|     10|   {
2844|      |     /* The chunk length must be updated for
process_IDAT */
2845|     10|     assert(file→chunk != NULL);
2846|     10|     assert(file→chunk→chunk_type == png_IDAT);
2847|     10|     file→chunk→chunk_length = file→length;
2848|     10|   }
```

▼ read_callback(target)

- Function calling count: 19.2k → 37.1k

카운트는 늘었지만 coverage는 거의 비슷

▼ cJSON(target 1개)

- Target function을 1개로 했을 때(target: parse_string)
- 전체적으로 관련 Target function branch count가 약 1.68배 정도 상승하였다.
- 전체적으로 관련되지 않은 function branch count가 약 8.6배 정도 감소하였다.
- target함수와 관련된 function들은 branch count가 늘어나지만, 관련이 없는 함수들은 줄어든 것이 보인다. (e,g, print_number, parse_number, parse_array, print_array 줄어듦)

▼ parse_string(target)

- Function calling count: 27.7k → 52.8k (약 1.9배)

1. first evidence(약 1.71배)
    a. 19.1k| if ((size_t)(input_end + 1 - input_buffer→content) >= input_buffer→length)

b. 32.8k| if ((size_t)(input_end + 1 - input_buffer→content) >= input_buffer→length)

2. Seconde evidence(약 1.6배)

a. 10.4k| case 'u':
10.4k| sequence_length = utf16_literal_to_utf8(input_poin

b. 17.0k| case 'u':
17.0k| sequence_length = utf16_literal_to_utf8(input_poin

3. Third evidence(약 1.25배)

a. 262| if (output != NULL)
144| {
144| input_buffer→hooks.deallocate(output);
144| }

b. 320| if (output != NULL)
180| {
180| input_buffer→hooks.deallocate(output);
180| }

▼ parse_value(같이 enhance 영향 받음)

- Function calling count: 40.4k→49.9k (약 1.23배)

1. First Evidence(약 1.66배)

a. 40.4k| if (can_read(input_buffer, 4) && (strncmp((const char*)buffer_at_offset(input_buffer), "null", 4) == 0))
359| {
359| item→type = cJSON_NULL;
359| input_buffer→offset += 4;
359| return true;
359| }

b. 49.9k| if (can_read(input_buffer, 4) && (strncmp((const char*)buffer_at_offset(input_buffer), "null", 4) == 0))
596| {
596| item→type = cJSON_NULL;
596| input_buffer→offset += 4;

596| return true;
596| }

2. Second Evidence (약 1.9배)

    a. 39.4k| if (can_access_at_index(input_buffer, 0) &&
    (buffer_at_offset(input_buffer)[0] == '\"'))
    11.1k| {
    11.1k| return parse_string(item, input_buffer);
    11.1k| }

    b. 48.5k| if (can_access_at_index(input_buffer, 0) &&
    (buffer_at_offset(input_buffer)[0] == '\"'))
    21.8k| {
    21.8k| return parse_string(item, input_buffer);
    21.8k| }

▼ parse_object(같이 enhance 영향 받음)

- Function calling count: 5.25k → 7.54k (약 1.43배)

1. First evidence (약 2배)

    a. 1656| 23.5k| {
    1657| | /* add to the end and advance */
    1658| 23.5k| current_item→next = new_item;
    1659| 23.5k| new_item→prev = current_item;
    1660| 23.5k| current_item = new_item;
    1661| 23.5k| }

    b. 1655| 11.5k| {
    1656| | /* add to the end and advance */
    1657| 11.5k| current_item→next = new_item;
    1658| 11.5k| new_item→prev = current_item;
    1659| 11.5k| current_item = new_item;
    1660| 11.5k| }

2. Second evidence (약 1.67배)

    a. 1684| 622| {
    1685| 622| goto fail; /* failed to parse value */
    1686| 622| }

b. 1690| 1.04k| {
1691| 1.04k| goto fail; /* failed to parse value */
1692| 1.04k| }

▼ print_number(관련 없는 함수라 less covered)

- Function calling count: 4.85k→5.62k (약 1.15배)

1. First evidence (약 19배 감소)

a. 567| 499| {
568| 499| length = sprintf((char*)number_buffer, "null");
569| 499| }

b. 568| 26| {
569| 26| length = sprintf((char*)number_buffer, "null");
570| 26| }

2. Second evidence (약 3.68배 감소)

a. 581| 166| {
582| | /* If not, print with 17 decimal places of precision */
583| 166| *length = sprintf((char*)number_buffer, "%1.17g", d);*
584| 166| }

b. 582| 45| {
583| | /* If not, print with 17 decimal places of precision */
584| 45| *length = sprintf((char*)number_buffer, "%1.17g", d);*
585| 45| }

▼ parse_number(관련 없는 함수라 less covered)

- function calling count: 10.5k → 9.50k (약 1.1배 감소)

1. First evidence(약 1.25배 감소)

a. 325| 9.28k| case '0':
326| 14.7k| case '1':
327| 17.2k| case '2':

328| 18.5k| case '3':
329| 18.8k| case '4':
330| 21.3k| case '5':
331| 22.5k| case '6':
332| 23.9k| case '7':
333| 24.7k| case '8':
334| 26.0k| case '9':
335| 26.1k| case '+':
336| 27.6k| case '-':
337| 28.1k| case 'e':
338| 28.1k| case 'E':

b. 325| 13.4k| case '0':
326| 21.4k| case '1':
327| 24.2k| case '2':
328| 25.5k| case '3':
329| 26.1k| case '4':
330| 26.9k| case '5':
331| 27.8k| case '6':
332| 28.8k| case '7':
333| 29.7k| case '8':
334| 31.2k| case '9':
335| 31.2k| case '+':
336| 32.7k| case '-':
337| 32.8k| case 'e':
338| 33.7k| case 'E':

2. Second evidence (약 10.6배 감소)

a. 363| 940| {
364| 940| item→valueint = INT_MAX;
365| 940| }

b. 363| 88| {
364| 88| item→valueint = INT_MAX;
365| 88| }

▼ libpng(target 1개)

- Target function을 1개로 했을 때(target: calc_image_size)
- target function과 관련된 함수들은 branch count가 전부 증가했다.
- 전체적으로 Target function과 관련된 함수의 branch count는 약 2.7배 상승하였다.
- 전체적으로 Target function과 관련 없는 함수의 branch count는 약 3배 감소하였다.

▼ calc_image_size(target)

- function calling count: 1.10k → 1.64k(약 1.49배)

1. First evidence (약 5.31배 증가)
    a. 1464| 58| case PNG_INTERLACE_ADAM7:
    1465| | /* Interlacing makes the image larger because of the replication of
    1466| | * both the filter byte and the padding to a byte boundary.
    1467| | */
    1468| 58| {
    b. 1464| 308| case PNG_INTERLACE_ADAM7:
    1465| | /* Interlacing makes the image larger because of the replication of
    1466| | * both the filter byte and the padding to a byte boundary.
    1467| | */
2. Second evidence (약 1.27배 증가)
    a. 987| case PNG_INTERLACE_NONE:
    b. 1499| 1.26k| case PNG_INTERLACE_NONE:

▼ uarb_mult_digit(같이 enhance 영향 받음)

- Function calling count: 2.79k → 8.42k (약 3.01배)

1. First evidence (약 3.1배 증가)

a. 275| 3.53k| if (out_digits < n_digits)
276| 3.10k| carry += (png_uint_32)num[out_digits] * val;

b. 275| 11.8k| if (out_digits < n_digits)
276| 9.64k| carry += (png_uint_32)num[out_digits] * val;

2. Seconde evidence (약 2.34배 증가)

a. 285| 2.76k| if (out_digits > a_digits)
286| 1.29k| return out_digits;
287| 2.76k| }

b. 285| 7.89k| if (out_digits > a_digits)
286| 3.03k| return out_digits;
287| 7.89k| }

▼ uarb_inc(같이 enhance 영향 받음)

- Function calling count: 360k→444k (약 1.23배 증가)

1. First evidence (약 1.82배 증가)

a. 172| 52.7k| while (val > 0)
173| 6.73k| {
174| 6.73k| result[ndigits++] = (png_uint_16)(val & 0xffff);
175| 6.73k| val >>= 16;
176| 6.73k| }

b. 172| 67.9k| while (val > 0)
173| 12.3k| {
174| 12.3k| result[ndigits++] = (png_uint_16)(val & 0xffff);
175| 12.3k| val >>= 16;
176| 12.3k| }

▼ uarb_copy(관련 없는 함수라 less covered)

- Function calling count: 2.64k → 1.51k (약 1.74배 감소)

1. Evidence (약 1.68배 감소)

a. 188| 2.84k| if ((to[d] = from[d]) != 0)
189| 2.84k| odigits = d+1;

b. 188| 1.69k| if ((to[d] = from[d]) != 0)
189| 1.69k| odigits = d+1;

▼ uarb_cmp(관련 없는 함수라 less covered)

- Function calling count: 718 → 311 (약 2.3배 감소)

1. First Evidence (약 4.25배 감소)
   a. 350| 693| if (adigits > bdigits)
   351| 85| return 1;
   b. 350| 310| if (adigits > bdigits)
   351| 20| return 1;
2. Second Evidence (약 3.14배 감소)
   a. 357| 673| else if (a[adigits] > b[adigits])
   358| 44| return 1;
   b. 357| 421| else if (a[adigits] > b[adigits])
   358| 14| return 1;