The Forgotten Treasure In Classic Targets

About Us

- Juntao Wu(@Dawuge3)
- Security Researcher
- Founder of Shuffle Team(@ShuffleTea38197)
- Huawei,Honor,Samsung Mobile Security Hall of Fame
- Mainly Focus on Mobile/Web3 research
- Spoken at Zer0Con, KCon, BlackHat

- Hangyu Hua(@HBh25Y)
- Security Researcher at Numen Cyber
- Co-Founder of Shuffle Team(@ShuffleTea38197)
- Mainly Focus on Low Level Security research

Agenda

- Part I: The Forgotten Treasure
- Part II: Review The Targets
- Part III: Enhance Fuzzers

Part I: The Forgotten Treasure

Background

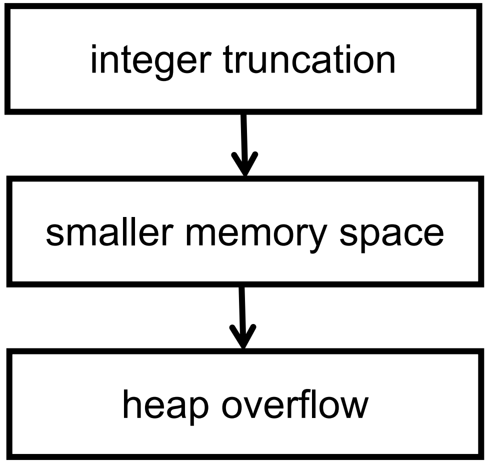
- As security researchers continue to discover vulnerabilities and various *nix operating system versions are iterated, the number of vulnerabilities with known attack vectors has gradually decreased. As a result, security researchers have paid less attention to them.

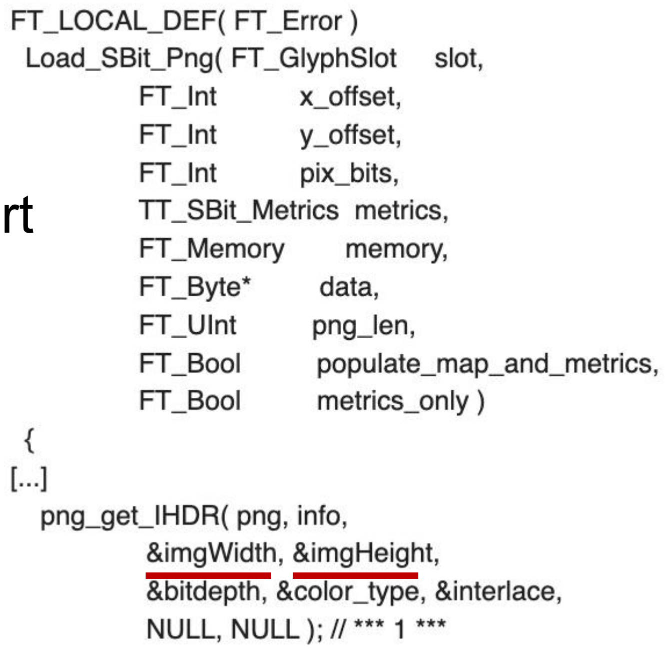
- Is this one of the reasons why wild exploit chains continue to appear?

- web2/web3

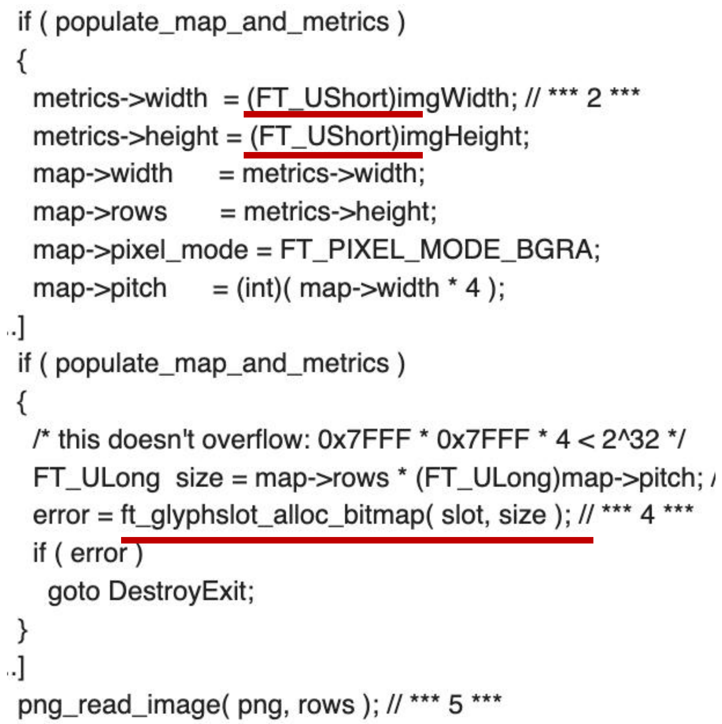
CVE-2020-15999

- in the wild
- FreeType
- src/sfnt/pngshim.c
- typedef unsigned short
- FT_UShort



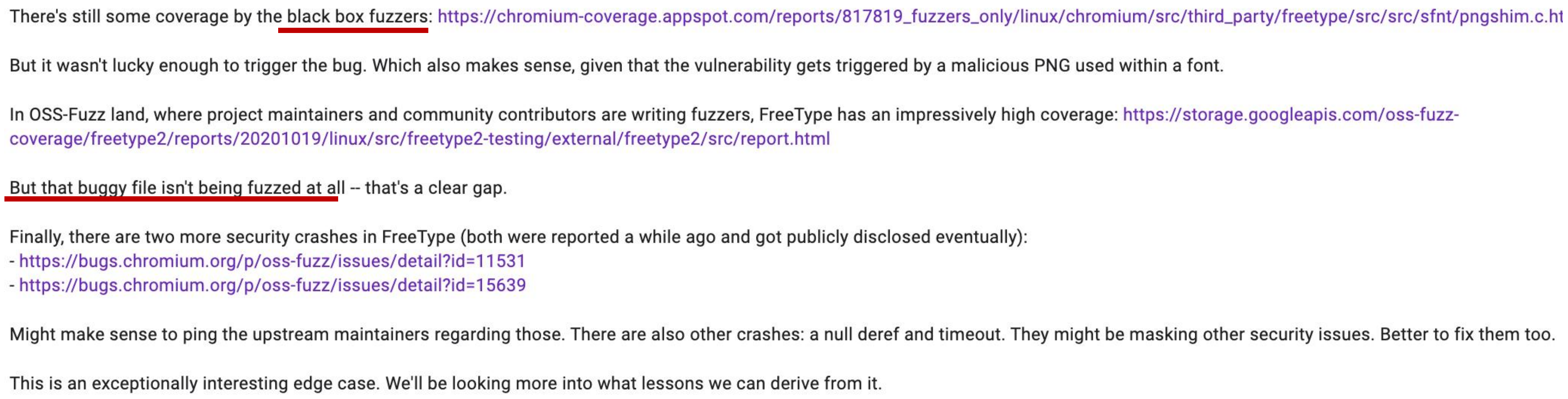


```
FT_LOCAL_DEF( FT_Error )
 Load_SBit_Png( FT_GlyphSlot     slot,
                FT_Int           x_offset,
                FT_Int           y_offset,
                FT_Int           pix_bits,
                TT_SBit_Metrics  metrics,
                FT_Memory        memory,
                FT_Byte*         data,
                FT_UInt          png_len,
                FT_Bool          populate_map_and_metrics,
                FT_Bool          metrics_only )
 {
[...]
   png_get_IHDR( png, info,
                 &imgWidth, &imgHeight,
                 &bitdepth, &color_type, &interlace,
                 NULL, NULL ); // *** 1 ***
```



```
if ( populate_map_and_metrics )
{
  metrics->width  = (FT_UShort)imgWidth; // *** 2 ***
  metrics->height = (FT_UShort)imgHeight;
  map->width      = metrics->width;
  map->rows       = metrics->height;
  map->pixel_mode = FT_PIXEL_MODE_BGRA;
  map->pitch      = (int)( map->width * 4 );
.]
if ( populate_map_and_metrics )
{
  /* this doesn't overflow: 0x7FFF * 0x7FFF * 4 < 2^32 */
  FT_ULong  size = map->rows * (FT_ULong)map->pitch; /
  error = ft_glyphslot_alloc_bitmap( slot, size ); // *** 4 ***
  if ( error )
    goto DestroyExit;
}
.]
png_read_image( png, rows ); // *** 5 ***
```

The Fuzzing Result



There's still some coverage by the black box fuzzers: https://chromium-coverage.appspot.com/reports/817819_fuzzers_only/linux/chromium/src/third_party/freetype/src/src/sfnt/pngshim.c.ht

But it wasn't lucky enough to trigger the bug. Which also makes sense, given that the vulnerability gets triggered by a malicious PNG used within a font.

In OSS-Fuzz land, where project maintainers and community contributors are writing fuzzers, FreeType has an impressively high coverage: https://storage.googleapis.com/oss-fuzz-coverage/freetype2/reports/20201019/linux/src/freetype2-testing/external/freetype2/src/report.html

But that buggy file isn't being fuzzed at all -- that's a clear gap.

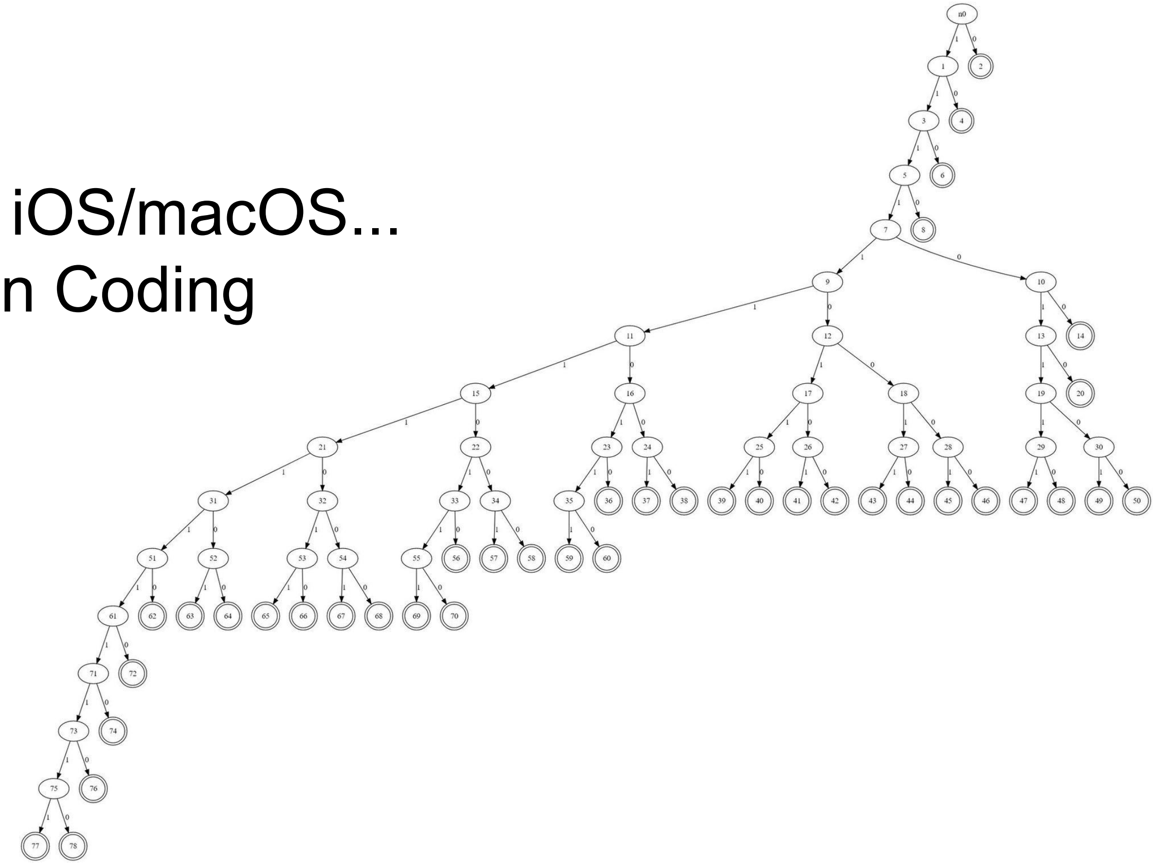
Finally, there are two more security crashes in FreeType (both were reported a while ago and got publicly disclosed eventually):
- https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=11531
- https://bugs.chromium.org/p/oss-fuzz/issues/detail?id=15639

Might make sense to ping the upstream maintainers regarding those. There are also other crashes: a null deref and timeout. They might be masking other security issues. Better to fix them too.

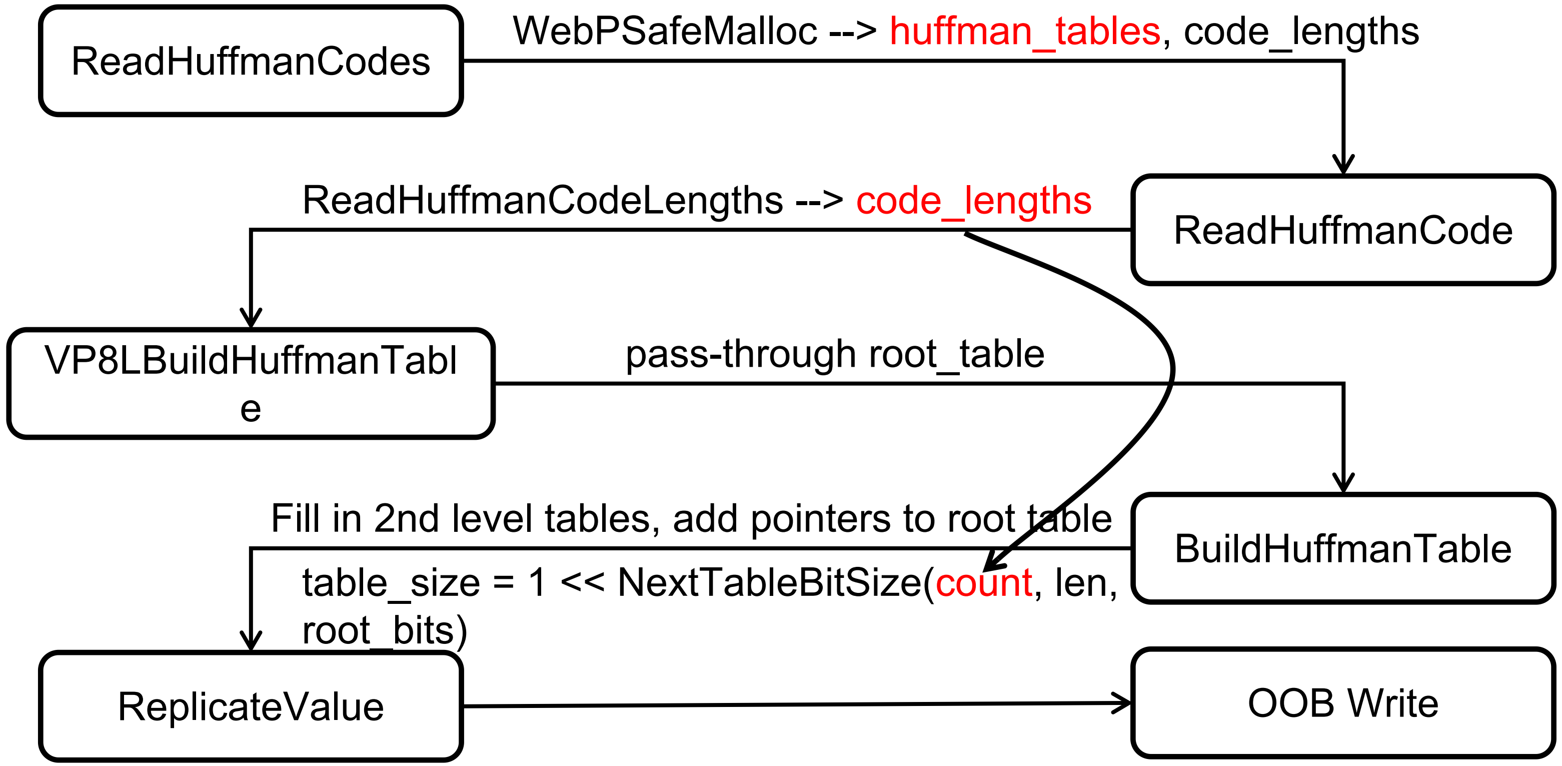
This is an exceptionally interesting edge case. We'll be looking more into what lessons we can derive from it.

CVE-2023-4863, CVE-2023-41064

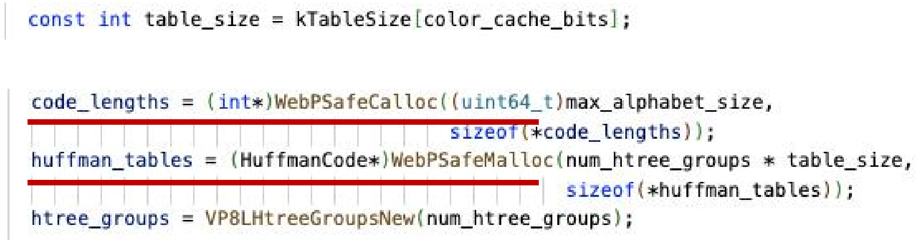
- WebP/Libwebp
- In the wild
- Android, Chrome, iOS/macOS...
- Canonical Huffman Coding



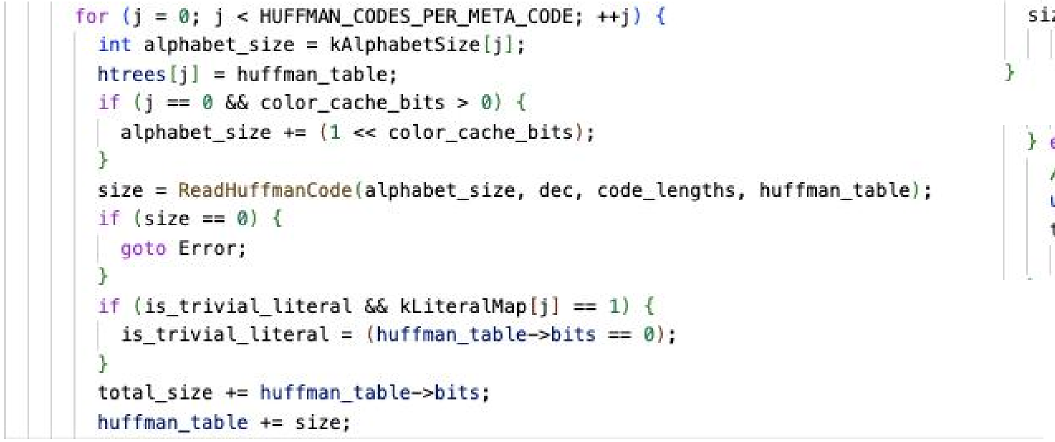
Trigger flow


ReadHuffmanCodes
WebPSafeMalloc --> huffman_tables, code_lengths
ReadHuffmanCode
ReadHuffmanCodeLengths --> code_lengths
VP8LBuildHuffmanTable
pass-through root_table
BuildHuffmanTable
Fill in 2nd level tables, add pointers to root table
table_size = 1 << NextTableBitSize(count, len, root_bits)
ReplicateValue
OOB Write

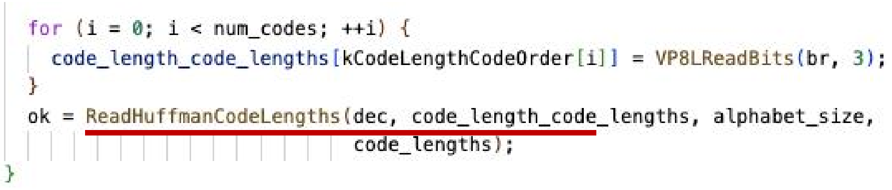
Trigger flow



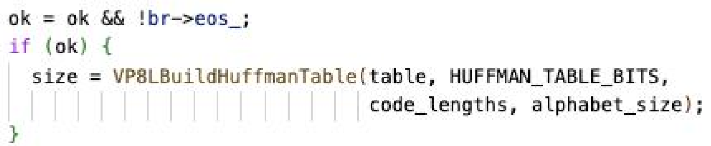
```
const int table_size = kTableSize[color_cache_bits];

code_lengths = (int*)WebPSafeCalloc((uint64_t)max_alphabet_size,
                                     sizeof(*code_lengths));
huffman_tables = (HuffmanCode*)WebPSafeMalloc(num_htree_groups * table_size,
                                               sizeof(*huffman_tables));
htree_groups = VP8LHtreeGroupsNew(num_htree_groups);
```



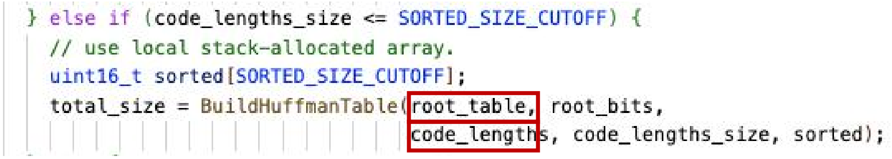
```
    for (j = 0; j < HUFFMAN_CODES_PER_META_CODE; ++j) {
      int alphabet_size = kAlphabetSize[j];
      htrees[j] = huffman_table;
      if (j == 0 && color_cache_bits > 0) {
        alphabet_size += (1 << color_cache_bits);
      }
      size = ReadHuffmanCode(alphabet_size, dec, code_lengths, huffman_table);
      if (size == 0) {
        goto Error;
      }
      if (is_trivial_literal && kLiteralMap[j] == 1) {
        is_trivial_literal = (huffman_table->bits == 0);
      }
      total_size += huffman_table->bits;
      huffman_table += size;
```



```
  for (i = 0; i < num_codes; ++i) {
    code_length_code_lengths[kCodeLengthCodeOrder[i]] = VP8LReadBits(br, 3);
  }
  ok = ReadHuffmanCodeLengths(dec, code_length_code_lengths, alphabet_size,
                              code_lengths);
}
```

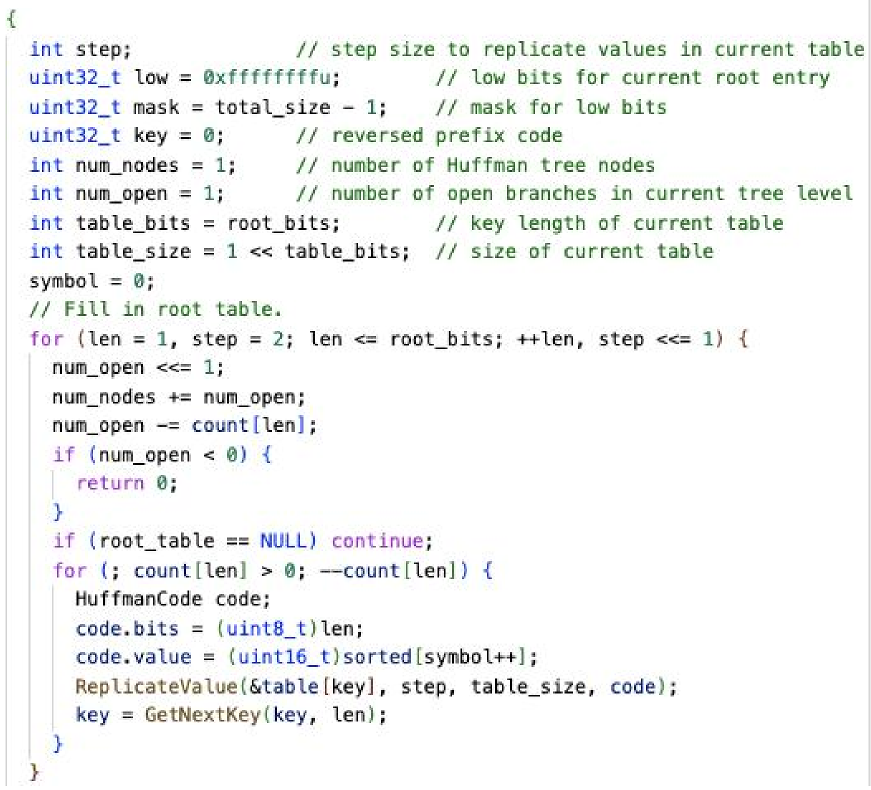


```
ok = ok && !br->eos_;
if (ok) {
  size = VP8LBuildHuffmanTable(table, HUFFMAN_TABLE_BITS,
                               code_lengths, alphabet_size);
}
```

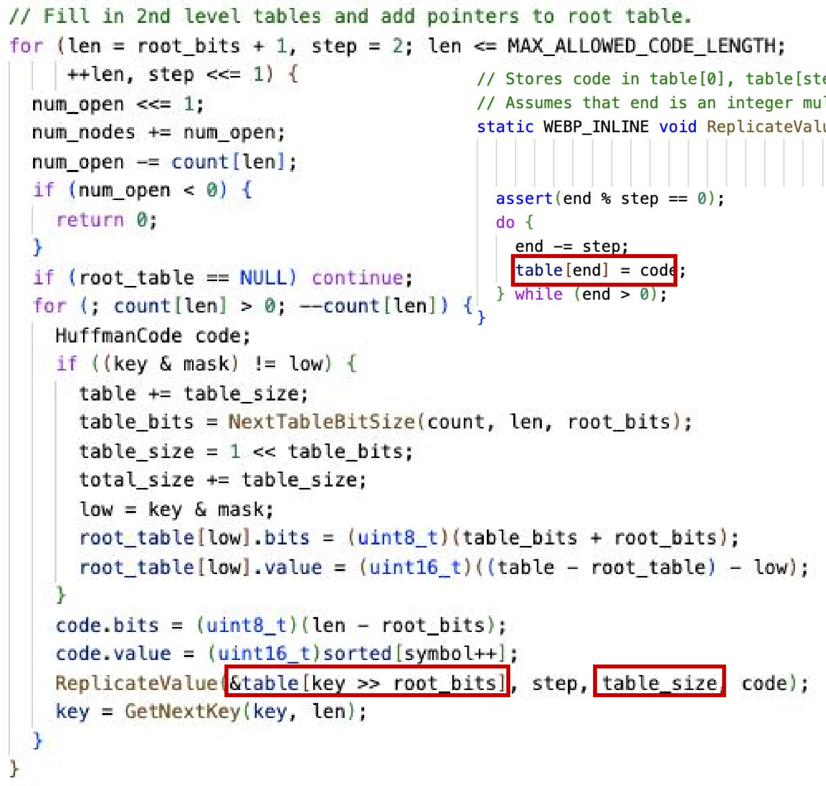


```
  } else if (code_lengths_size <= SORTED_SIZE_CUTOFF) {
    // use local stack-allocated array.
    uint16_t sorted[SORTED_SIZE_CUTOFF];
    total_size = BuildHuffmanTable(root_table, root_bits,
                                   code_lengths, code_lengths_size, sorted);
```

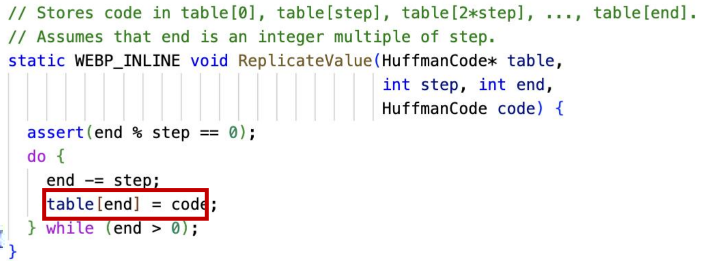
Trigger flow



```
{
  int step;              // step size to replicate values in current table
  uint32_t low = 0xffffffffu;        // low bits for current root entry
  uint32_t mask = total_size - 1;    // mask for low bits
  uint32_t key = 0;      // reversed prefix code
  int num_nodes = 1;     // number of Huffman tree nodes
  int num_open = 1;      // number of open branches in current tree level
  int table_bits = root_bits;        // key length of current table
  int table_size = 1 << table_bits;  // size of current table
  symbol = 0;
  // Fill in root table.
  for (len = 1, step = 2; len <= root_bits; ++len, step <<= 1) {
    num_open <<= 1;
    num_nodes += num_open;
    num_open -= count[len];
    if (num_open < 0) {
      return 0;
    }
    if (root_table == NULL) continue;
    for (; count[len] > 0; --count[len]) {
      HuffmanCode code;
      code.bits = (uint8_t)len;
      code.value = (uint16_t)sorted[symbol++];
      ReplicateValue(&table[key], step, table_size, code);
      key = GetNextKey(key, len);
    }
  }
```

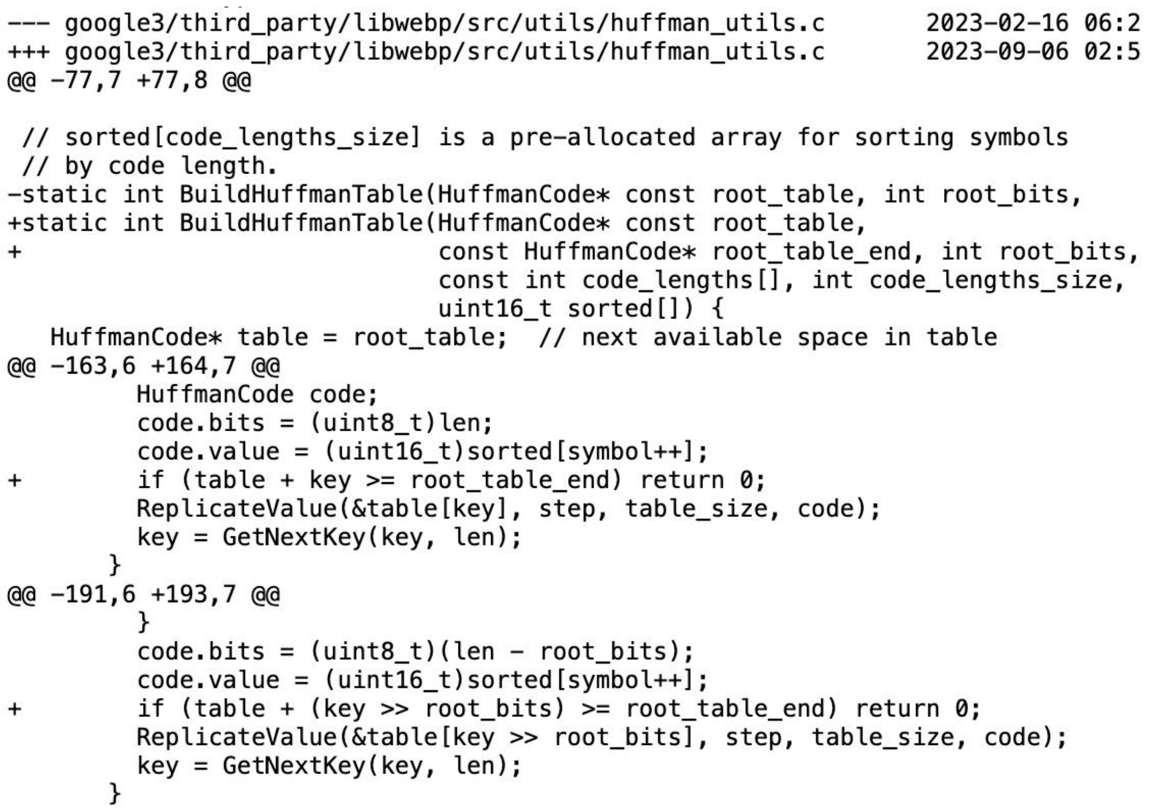


```
// Fill in 2nd level tables and add pointers to root table.
for (len = root_bits + 1, step = 2; len <= MAX_ALLOWED_CODE_LENGTH;
     ++len, step <<= 1) {
  num_open <<= 1;
  num_nodes += num_open;
  num_open -= count[len];
  if (num_open < 0) {
    return 0;
  }
  if (root_table == NULL) continue;
  for (; count[len] > 0; --count[len]) {
    HuffmanCode code;
    if ((key & mask) != low) {
      table += table_size;
      table_bits = NextTableBitSize(count, len, root_bits);
      table_size = 1 << table_bits;
      total_size += table_size;
      low = key & mask;
      root_table[low].bits = (uint8_t)(table_bits + root_bits);
      root_table[low].value = (uint16_t)((table - root_table) - low);
    }
    code.bits = (uint8_t)(len - root_bits);
    code.value = (uint16_t)sorted[symbol++];
    ReplicateValue(&table[key >> root_bits], step, table_size, code);
    key = GetNextKey(key, len);
  }
}
```



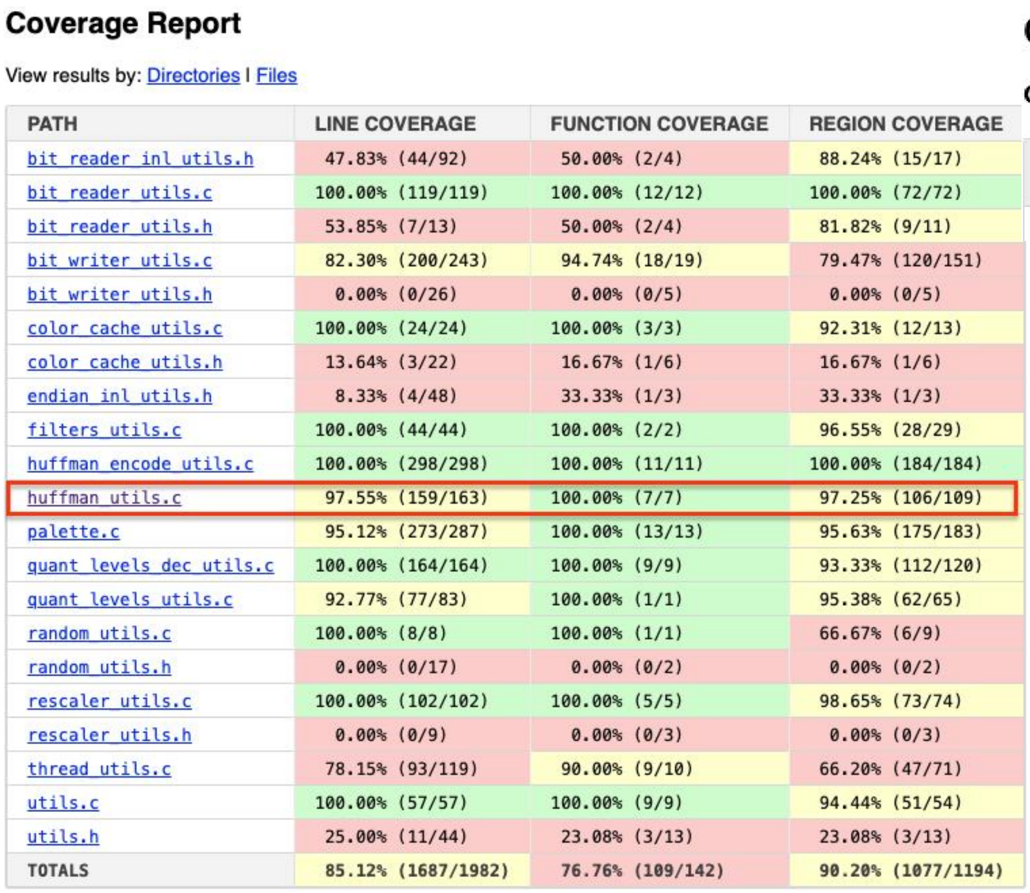
```
// Stores code in table[0], table[step], table[2*step], ..., table[end].
// Assumes that end is an integer multiple of step.
static WEBP_INLINE void ReplicateValue(HuffmanCode* table,
                                       int step, int end,
                                       HuffmanCode code) {
  assert(end % step == 0);
  do {
    end -= step;
    table[end] = code;
  } while (end > 0);
}
```

Patch



```
--- google3/third_party/libwebp/src/utils/huffman_utils.c        2023-02-16 06:2
+++ google3/third_party/libwebp/src/utils/huffman_utils.c        2023-09-06 02:5
@@ -77,7 +77,8 @@

 // sorted[code_lengths_size] is a pre-allocated array for sorting symbols
 // by code length.
-static int BuildHuffmanTable(HuffmanCode* const root_table, int root_bits,
+static int BuildHuffmanTable(HuffmanCode* const root_table,
+                             const HuffmanCode* root_table_end, int root_bits,
                              const int code_lengths[], int code_lengths_size,
                              uint16_t sorted[]) {
   HuffmanCode* table = root_table;  // next available space in table
@@ -163,6 +164,7 @@
         HuffmanCode code;
         code.bits = (uint8_t)len;
         code.value = (uint16_t)sorted[symbol++];
+        if (table + key >= root_table_end) return 0;
         ReplicateValue(&table[key], step, table_size, code);
         key = GetNextKey(key, len);
       }
@@ -191,6 +193,7 @@
         }
         code.bits = (uint8_t)(len - root_bits);
         code.value = (uint16_t)sorted[symbol++];
+        if (table + (key >> root_bits) >= root_table_end) return 0;
         ReplicateValue(&table[key >> root_bits], step, table_size, code);
         key = GetNextKey(key, len);
       }
```

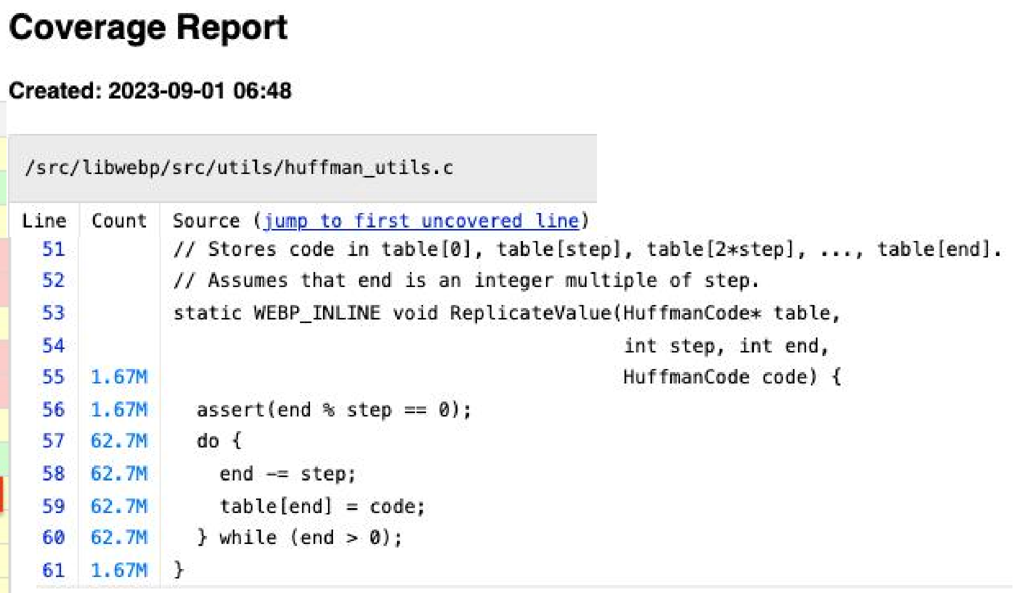
The Fuzzing Result



Coverage Report

View results by: Directories | Files

PATH	LINE COVERAGE	FUNCTION COVERAGE	REGION COVERAGE
bit_reader_inl_utils.h	47.83% (44/92)	50.00% (2/4)	88.24% (15/17)
bit_reader_utils.c	100.00% (119/119)	100.00% (12/12)	100.00% (72/72)
bit_reader_utils.h	53.85% (7/13)	50.00% (2/4)	81.82% (9/11)
bit_writer_utils.c	82.30% (200/243)	94.74% (18/19)	79.47% (120/151)
bit_writer_utils.h	0.00% (0/26)	0.00% (0/5)	0.00% (0/5)
color_cache_utils.c	100.00% (24/24)	100.00% (3/3)	92.31% (12/13)
color_cache_utils.h	13.64% (3/22)	16.67% (1/6)	16.67% (1/6)
endian_inl_utils.h	8.33% (4/48)	33.33% (1/3)	33.33% (1/3)
filters_utils.c	100.00% (44/44)	100.00% (2/2)	96.55% (28/29)
huffman_encode_utils.c	100.00% (298/298)	100.00% (11/11)	100.00% (184/184)
huffman_utils.c	97.55% (159/163)	100.00% (7/7)	97.25% (106/109)
palette.c	95.12% (273/287)	100.00% (13/13)	95.63% (175/183)
quant_levels_dec_utils.c	100.00% (164/164)	100.00% (9/9)	93.33% (112/120)
quant_levels_utils.c	92.77% (77/83)	100.00% (1/1)	95.38% (62/65)
random_utils.c	100.00% (8/8)	100.00% (1/1)	66.67% (6/9)
random_utils.h	0.00% (0/17)	0.00% (0/2)	0.00% (0/2)
rescaler_utils.c	100.00% (102/102)	100.00% (5/5)	98.65% (73/74)
rescaler_utils.h	0.00% (0/9)	0.00% (0/3)	0.00% (0/3)
thread_utils.c	78.15% (93/119)	90.00% (9/10)	66.20% (47/71)
utils.c	100.00% (57/57)	100.00% (9/9)	94.44% (51/54)
utils.h	25.00% (11/44)	23.08% (3/13)	23.08% (3/13)
TOTALS	85.12% (1687/1982)	76.76% (109/142)	90.20% (1077/1194)



Coverage Report

Created: 2023-09-01 06:48

/src/libwebp/src/utils/huffman_utils.c

```
Line  Count  Source (jump to first uncovered line)
 51          // Stores code in table[0], table[step], table[2*step], ..., table[end].
 52          // Assumes that end is an integer multiple of step.
 53          static WEBP_INLINE void ReplicateValue(HuffmanCode* table,
 54                                                 int step, int end,
 55   1.67M                                         HuffmanCode code) {
 56   1.67M    assert(end % step == 0);
 57   62.7M    do {
 58   62.7M      end -= step;
 59   62.7M      table[end] = code;
 60   62.7M    } while (end > 0);
 61   1.67M  }
```

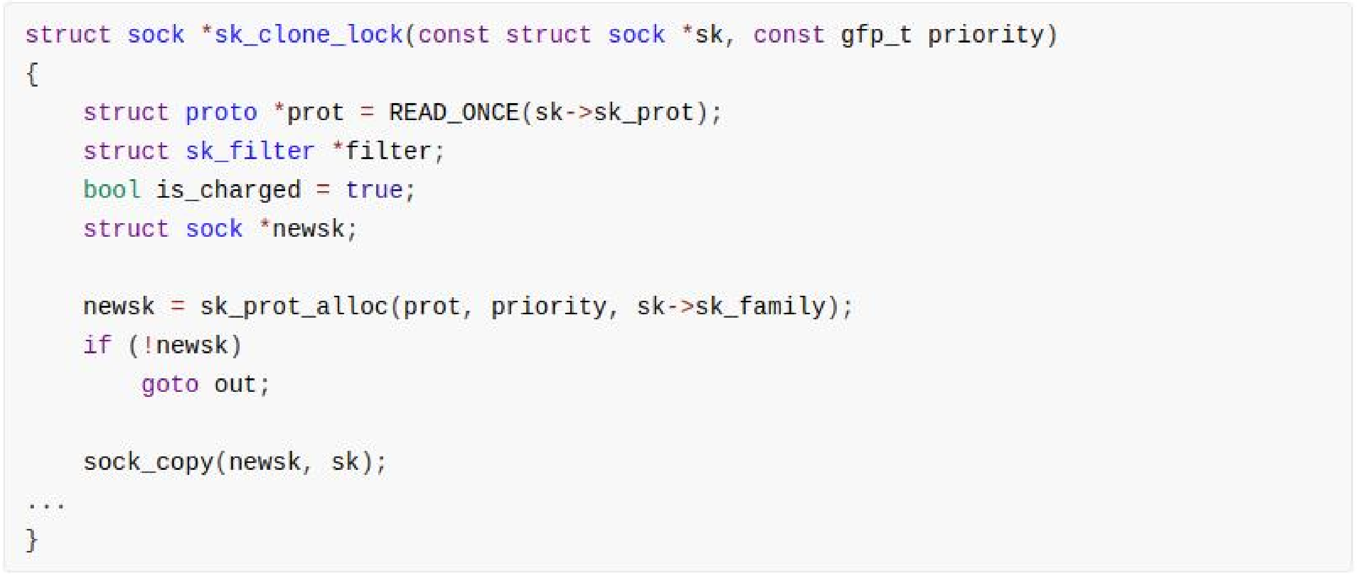
Reflection

- Why hadn't this bug been found earlier?
- Had the library not been fuzzed enough?
- Had it not been fuzzed right?

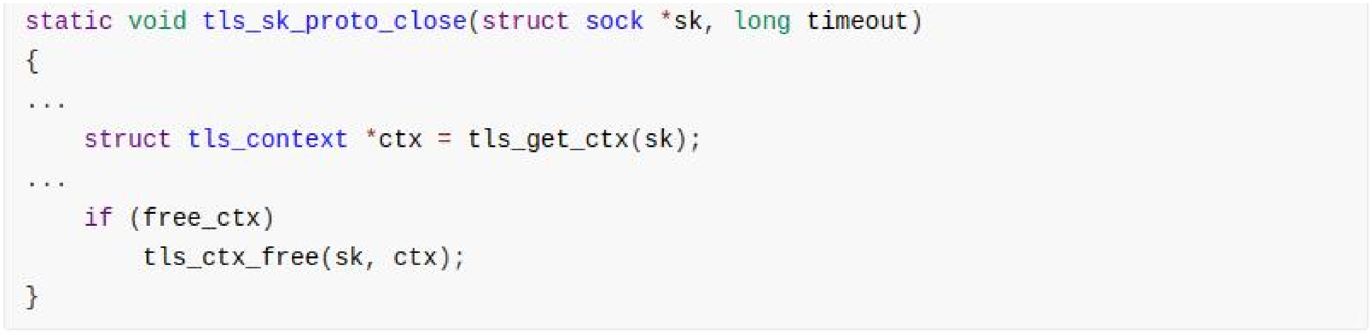
-------- blog from Ben Hawkes

CVE-2023-0461

- Linux kernel
- TCP_ULP



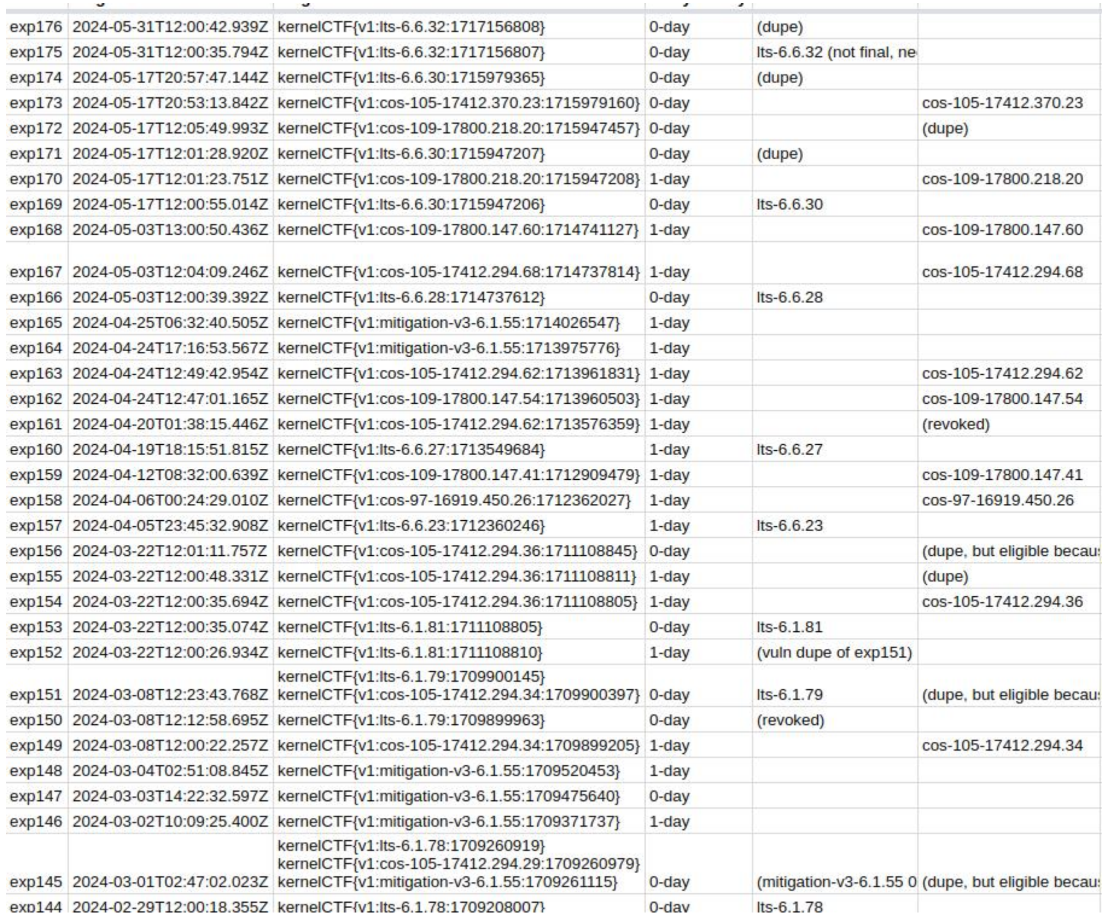
```c
struct sock *sk_clone_lock(const struct sock *sk, const gfp_t priority)
{
    struct proto *prot = READ_ONCE(sk->sk_prot);
    struct sk_filter *filter;
    bool is_charged = true;
    struct sock *newsk;

    newsk = sk_prot_alloc(prot, priority, sk->sk_family);
    if (!newsk)
        goto out;

    sock_copy(newsk, sk);
...
}
```



```c
static void tls_sk_proto_close(struct sock *sk, long timeout)
{
...
    struct tls_context *ctx = tls_get_ctx(sk);
...
    if (free_ctx)
        tls_ctx_free(sk, ctx);
}
```

Part II: Review The Targets

Exploitation



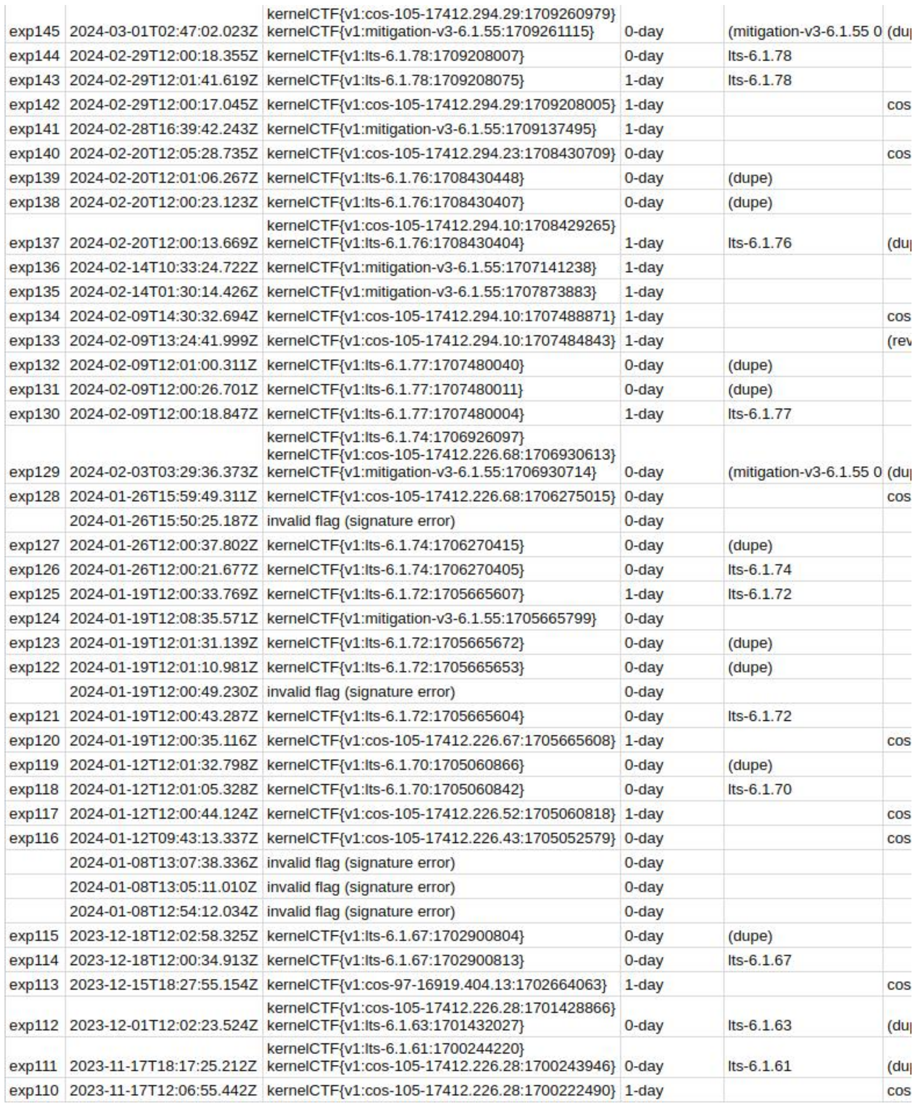
exp176	2024-05-31T12:00:42.939Z	kernelCTF{v1:lts-6.6.32:1717156808}	0-day	(dupe)	
exp175	2024-05-31T12:00:35.794Z	kernelCTF{v1:lts-6.6.32:1717156807}	0-day	lts-6.6.32 (not final, ne	
exp174	2024-05-17T20:57:47.144Z	kernelCTF{v1:lts-6.6.30:1715979365}	0-day	(dupe)	
exp173	2024-05-17T20:53:13.842Z	kernelCTF{v1:cos-105-17412.370.23:1715979160}	0-day		cos-105-17412.370.23
exp172	2024-05-17T12:05:49.993Z	kernelCTF{v1:cos-109-17800.218.20:1715947457}	0-day		(dupe)
exp171	2024-05-17T12:01:28.920Z	kernelCTF{v1:lts-6.6.30:1715947207}	0-day	(dupe)	
exp170	2024-05-17T12:01:23.751Z	kernelCTF{v1:cos-109-17800.218.20:1715947208}	1-day		cos-109-17800.218.20
exp169	2024-05-17T12:00:55.014Z	kernelCTF{v1:lts-6.6.30:1715947206}	0-day	lts-6.6.30	
exp168	2024-05-03T13:00:50.436Z	kernelCTF{v1:cos-109-17800.147.60:1714741127}	1-day		cos-109-17800.147.60
exp167	2024-05-03T12:04:09.246Z	kernelCTF{v1:cos-105-17412.294.68:1714737814}	1-day		cos-105-17412.294.68
exp166	2024-05-03T12:00:39.392Z	kernelCTF{v1:lts-6.6.28:1714737612}	0-day	lts-6.6.28	
exp165	2024-04-25T06:32:40.505Z	kernelCTF{v1:mitigation-v3-6.1.55:1714026547}	1-day		
exp164	2024-04-24T17:16:53.567Z	kernelCTF{v1:mitigation-v3-6.1.55:1713975776}	1-day		
exp163	2024-04-24T12:49:42.954Z	kernelCTF{v1:cos-105-17412.294.62:1713961831}	1-day		cos-105-17412.294.62
exp162	2024-04-24T12:47:01.165Z	kernelCTF{v1:cos-109-17800.147.54:1713960503}	1-day		cos-109-17800.147.54
exp161	2024-04-20T01:38:15.446Z	kernelCTF{v1:cos-105-17412.294.62:1713576359}	1-day		(revoked)
exp160	2024-04-19T18:15:51.815Z	kernelCTF{v1:lts-6.6.27:1713549684}	1-day	lts-6.6.27	
exp159	2024-04-12T08:32:00.639Z	kernelCTF{v1:cos-109-17800.147.41:1712909479}	1-day		cos-109-17800.147.41
exp158	2024-04-06T00:24:29.010Z	kernelCTF{v1:cos-97-16919.450.26:1712362027}	1-day		cos-97-16919.450.26
exp157	2024-04-05T23:45:32.908Z	kernelCTF{v1:lts-6.6.23:1712360246}	1-day	lts-6.6.23	
exp156	2024-03-22T12:01:11.757Z	kernelCTF{v1:cos-105-17412.294.36:1711108845}	0-day		(dupe, but eligible becau
exp155	2024-03-22T12:00:48.331Z	kernelCTF{v1:cos-105-17412.294.36:1711108811}	1-day		(dupe)
exp154	2024-03-22T12:00:35.694Z	kernelCTF{v1:cos-105-17412.294.36:1711108805}	1-day		cos-105-17412.294.36
exp153	2024-03-22T12:00:35.074Z	kernelCTF{v1:lts-6.1.81:1711108805}	0-day	lts-6.1.81	
exp152	2024-03-22T12:00:26.934Z	kernelCTF{v1:lts-6.1.81:1711108810}	1-day	(vuln dupe of exp151)	

exp151	2024-03-08T12:23:43.768Z	kernelCTF{v1:lts-6.1.79:1709900145}
		kernelCTF{v1:cos-105-17412.294.34:1709900397}
			0-day	lts-6.1.79	(dupe, but eligible becau

exp150	2024-03-08T12:12:58.695Z	kernelCTF{v1:lts-6.1.79:1709899963}	0-day	(revoked)	
exp149	2024-03-08T12:00:22.257Z	kernelCTF{v1:cos-105-17412.294.34:1709899205}	1-day		cos-105-17412.294.34
exp148	2024-03-04T02:51:08.845Z	kernelCTF{v1:mitigation-v3-6.1.55:1709520453}	1-day		
exp147	2024-03-03T14:22:32.597Z	kernelCTF{v1:mitigation-v3-6.1.55:1709475640}	0-day		
exp146	2024-03-02T10:09:25.400Z	kernelCTF{v1:mitigation-v3-6.1.55:1709371737}	1-day		

exp145	2024-03-01T02:47:02.023Z	kernelCTF{v1:lts-6.1.78:1709260919}
		kernelCTF{v1:cos-105-17412.294.29:1709260979}
		kernelCTF{v1:mitigation-v3-6.1.55:1709261115}
			0-day	(mitigation-v3-6.1.55 0	(dupe, but eligible becau

exp144	2024-02-29T12:00:18.355Z	kernelCTF{v1:lts-6.1.78:1709208007}	0-day	lts-6.1.78	



exp145	2024-03-01T02:47:02.023Z	kernelCTF{v1:cos-105-17412.294.29:1709260979}
		kernelCTF{v1:mitigation-v3-6.1.55:1709261115}
			0-day	(mitigation-v3-6.1.55 0	(du

exp144	2024-02-29T12:00:18.355Z	kernelCTF{v1:lts-6.1.78:1709208007}	0-day	lts-6.1.78	
exp143	2024-02-29T12:01:41.619Z	kernelCTF{v1:lts-6.1.78:1709208075}	1-day	lts-6.1.78	
exp142	2024-02-29T12:00:17.045Z	kernelCTF{v1:cos-105-17412.294.29:1709208005}	1-day		cos
exp141	2024-02-28T16:39:42.243Z	kernelCTF{v1:mitigation-v3-6.1.55:1709137495}	1-day		
exp140	2024-02-20T12:05:28.735Z	kernelCTF{v1:cos-105-17412.294.23:1708430709}	0-day		cos
exp139	2024-02-20T12:01:06.267Z	kernelCTF{v1:lts-6.1.76:1708430448}	0-day	(dupe)	
exp138	2024-02-20T12:00:23.123Z	kernelCTF{v1:lts-6.1.76:1708430407}	0-day	(dupe)	

exp137	2024-02-20T12:00:13.669Z	kernelCTF{v1:cos-105-17412.294.10:1708429265}
		kernelCTF{v1:lts-6.1.76:1708430404}
			1-day	lts-6.1.76	(du

exp136	2024-02-14T10:33:24.722Z	kernelCTF{v1:mitigation-v3-6.1.55:1707141238}	1-day		
exp135	2024-02-14T01:30:14.426Z	kernelCTF{v1:mitigation-v3-6.1.55:1707873883}	1-day		
exp134	2024-02-09T14:30:32.694Z	kernelCTF{v1:cos-105-17412.294.10:1707488871}	1-day		cos
exp133	2024-02-09T13:24:41.999Z	kernelCTF{v1:cos-105-17412.294.10:1707484843}	1-day		(rev
exp132	2024-02-09T12:01:00.311Z	kernelCTF{v1:lts-6.1.77:1707480040}	0-day	(dupe)	
exp131	2024-02-09T12:00:26.701Z	kernelCTF{v1:lts-6.1.77:1707480011}	0-day	(dupe)	
exp130	2024-02-09T12:00:18.847Z	kernelCTF{v1:lts-6.1.77:1707480004}	1-day	lts-6.1.77	

exp129	2024-02-03T03:29:36.373Z	kernelCTF{v1:lts-6.1.74:1706926097}
		kernelCTF{v1:cos-105-17412.226.68:1706930613}
		kernelCTF{v1:mitigation-v3-6.1.55:1706930714}
			0-day	(mitigation-v3-6.1.55 0	(du

exp128	2024-01-26T15:59:49.311Z	kernelCTF{v1:cos-105-17412.226.68:1706275015}	0-day		cos
	2024-01-26T15:50:25.187Z	invalid flag (signature error)	0-day		
exp127	2024-01-26T12:00:37.802Z	kernelCTF{v1:lts-6.1.74:1706270415}	0-day	(dupe)	
exp126	2024-01-26T12:00:21.677Z	kernelCTF{v1:lts-6.1.74:1706270405}	0-day	lts-6.1.74	
exp125	2024-01-19T12:00:33.769Z	kernelCTF{v1:lts-6.1.72:1705665607}	1-day	lts-6.1.72	
exp124	2024-01-19T12:08:35.571Z	kernelCTF{v1:mitigation-v3-6.1.55:1705665799}	0-day		
exp123	2024-01-19T12:01:31.139Z	kernelCTF{v1:lts-6.1.72:1705665672}	0-day	(dupe)	
exp122	2024-01-19T12:01:10.981Z	kernelCTF{v1:lts-6.1.72:1705665653}	0-day	(dupe)	
	2024-01-19T12:00:49.230Z	invalid flag (signature error)	0-day		
exp121	2024-01-19T12:00:43.287Z	kernelCTF{v1:lts-6.1.72:1705665604}	0-day	lts-6.1.72	
exp120	2024-01-19T12:00:35.116Z	kernelCTF{v1:cos-105-17412.226.67:1705665608}	1-day		cos
exp119	2024-01-12T12:01:32.798Z	kernelCTF{v1:lts-6.1.70:1705060866}	0-day	(dupe)	
exp118	2024-01-12T12:01:05.328Z	kernelCTF{v1:lts-6.1.70:1705060842}	0-day	lts-6.1.70	
exp117	2024-01-12T12:00:44.124Z	kernelCTF{v1:cos-105-17412.226.52:1705060818}	1-day		cos
exp116	2024-01-12T09:43:13.337Z	kernelCTF{v1:cos-105-17412.226.43:1705052579}	0-day		cos
	2024-01-08T13:07:38.336Z	invalid flag (signature error)	0-day		
	2024-01-08T13:05:11.010Z	invalid flag (signature error)	0-day		
	2024-01-08T12:54:12.034Z	invalid flag (signature error)	0-day		
exp115	2023-12-18T12:02:58.325Z	kernelCTF{v1:lts-6.1.67:1702900804}	0-day	(dupe)	
exp114	2023-12-18T12:00:34.913Z	kernelCTF{v1:lts-6.1.67:1702900813}	0-day	lts-6.1.67	
exp113	2023-12-15T18:27:55.154Z	kernelCTF{v1:cos-97-16919.404.13:1702664063}	1-day		cos

exp112	2023-12-01T12:02:23.524Z	kernelCTF{v1:cos-105-17412.226.28:1701428866}
		kernelCTF{v1:lts-6.1.63:1701432027}
			0-day	lts-6.1.63	(du

exp111	2023-11-17T18:17:25.212Z	kernelCTF{v1:lts-6.1.61:1700244220}
		kernelCTF{v1:cos-105-17412.226.28:1700243946}
			0-day	lts-6.1.61	(du

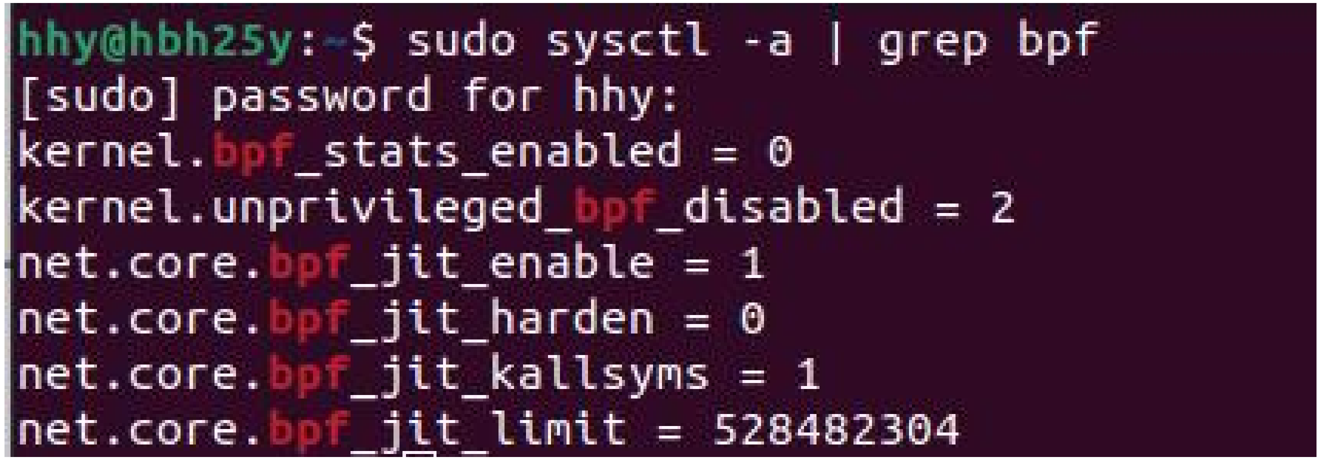
exp110	2023-11-17T12:06:55.442Z	kernelCTF{v1:cos-105-17412.226.28:1700222490}	1-day		cos

Attack Surface

0x00 Extended Berkeley Packet Filter(eBPF)

- CVE-2021-3490
- CVE-2021-34886
- CVE-2022-23222

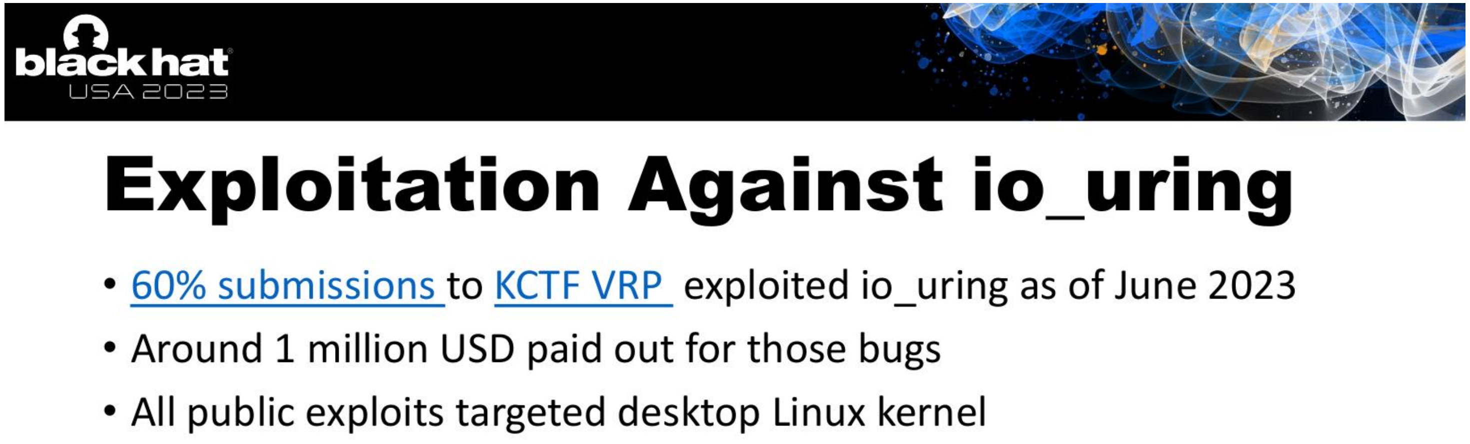
...



```
hhy@hbh25y:~$ sudo sysctl -a | grep bpf
[sudo] password for hhy:
kernel.bpf_stats_enabled = 0
kernel.unprivileged_bpf_disabled = 2
net.core.bpf_jit_enable = 1
net.core.bpf_jit_harden = 0
net.core.bpf_jit_kallsyms = 1
net.core.bpf_jit_limit = 528482304
```

Attack Surface

0x01 io_uring



black hat USA 2023

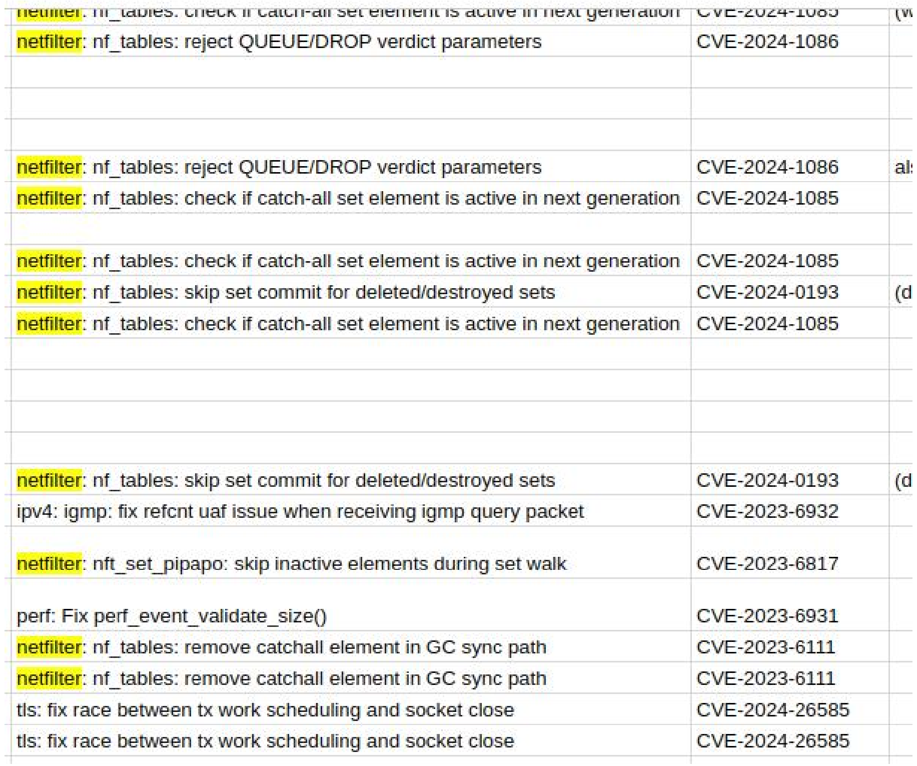
Exploitation Against io_uring

- 60% submissions to KCTF VRP exploited io_uring as of June 2023
- Around 1 million USD paid out for those bugs
- All public exploits targeted desktop Linux kernel

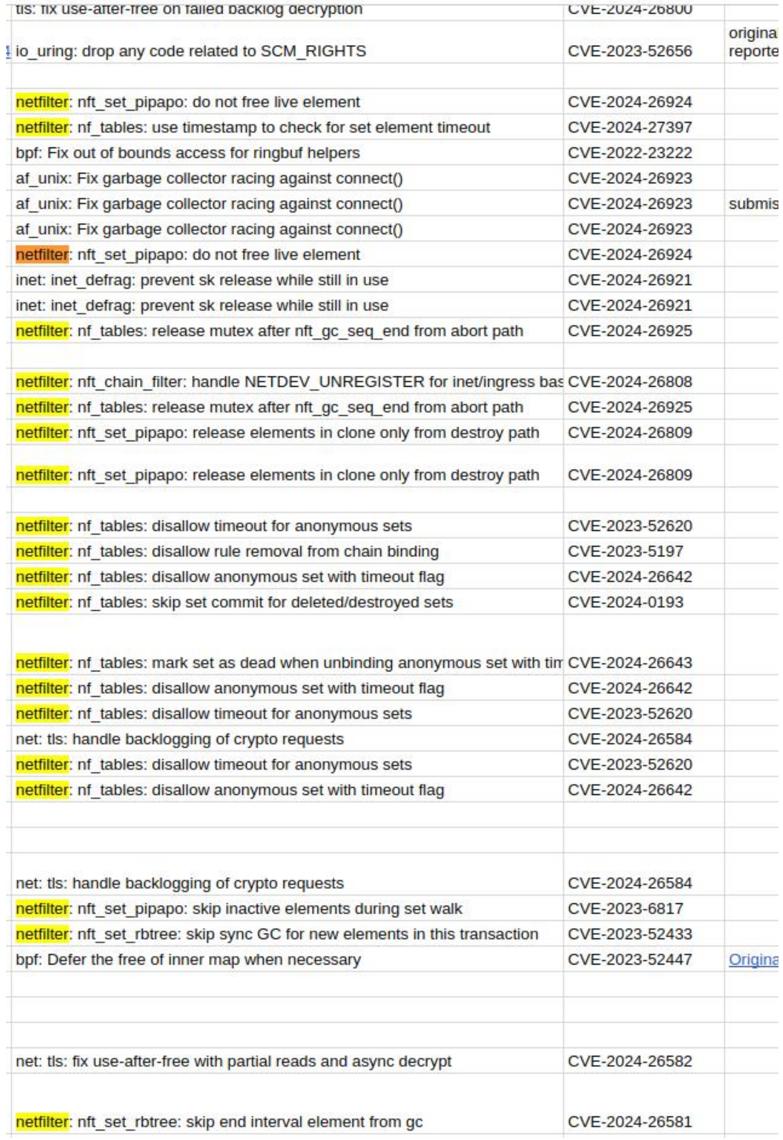
Bad io_uring: A New Era of Rooting for Android (black hat USA 2023)

Attack Surface

0x02 netfilter



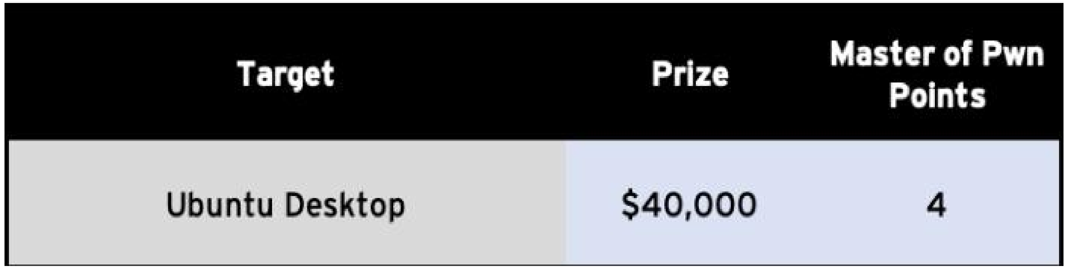
netfilter: nf_tables: check if catch-all set element is active in next generation	CVE-2024-1085	(w
netfilter: nf_tables: reject QUEUE/DROP verdict parameters	CVE-2024-1086	
netfilter: nf_tables: reject QUEUE/DROP verdict parameters	CVE-2024-1086	al
netfilter: nf_tables: check if catch-all set element is active in next generation	CVE-2024-1085	
netfilter: nf_tables: check if catch-all set element is active in next generation	CVE-2024-1085	
netfilter: nf_tables: skip set commit for deleted/destroyed sets	CVE-2024-0193	(d
netfilter: nf_tables: check if catch-all set element is active in next generation	CVE-2024-1085	
netfilter: nf_tables: skip set commit for deleted/destroyed sets	CVE-2024-0193	(d
ipv4: igmp: fix refcnt uaf issue when receiving igmp query packet	CVE-2023-6932	
netfilter: nft_set_pipapo: skip inactive elements during set walk	CVE-2023-6817	
perf: Fix perf_event_validate_size()	CVE-2023-6931	
netfilter: nf_tables: remove catchall element in GC sync path	CVE-2023-6111	
netfilter: nf_tables: remove catchall element in GC sync path	CVE-2023-6111	
tls: fix race between tx work scheduling and socket close	CVE-2024-26585	
tls: fix race between tx work scheduling and socket close	CVE-2024-26585	



tls: fix use-after-free on failed backlog decryption	CVE-2024-26800	
io_uring: drop any code related to SCM_RIGHTS	CVE-2023-52656	origina reporte
netfilter: nft_set_pipapo: do not free live element	CVE-2024-26924	
netfilter: nf_tables: use timestamp to check for set element timeout	CVE-2024-27397	
bpf: Fix out of bounds access for ringbuf helpers	CVE-2022-23222	
af_unix: Fix garbage collector racing against connect()	CVE-2024-26923	
af_unix: Fix garbage collector racing against connect()	CVE-2024-26923	submis
af_unix: Fix garbage collector racing against connect()	CVE-2024-26923	
netfilter: nft_set_pipapo: do not free live element	CVE-2024-26924	
inet: inet_defrag: prevent sk release while still in use	CVE-2024-26921	
inet: inet_defrag: prevent sk release while still in use	CVE-2024-26921	
netfilter: nf_tables: release mutex after nft_gc_seq_end from abort path	CVE-2024-26925	
netfilter: nft_chain_filter: handle NETDEV_UNREGISTER for inet/ingress bas	CVE-2024-26808	
netfilter: nf_tables: release mutex after nft_gc_seq_end from abort path	CVE-2024-26925	
netfilter: nft_set_pipapo: release elements in clone only from destroy path	CVE-2024-26809	
netfilter: nft_set_pipapo: release elements in clone only from destroy path	CVE-2024-26809	
netfilter: nf_tables: disallow timeout for anonymous sets	CVE-2023-52620	
netfilter: nf_tables: disallow rule removal from chain binding	CVE-2023-5197	
netfilter: nf_tables: disallow anonymous set with timeout flag	CVE-2024-26642	
netfilter: nf_tables: skip set commit for deleted/destroyed sets	CVE-2024-0193	
netfilter: nf_tables: mark set as dead when unbinding anonymous set with tin	CVE-2024-26643	
netfilter: nf_tables: disallow anonymous set with timeout flag	CVE-2024-26642	
netfilter: nf_tables: disallow timeout for anonymous sets	CVE-2023-52620	
net: tls: handle backlogging of crypto requests	CVE-2024-26584	
netfilter: nf_tables: disallow timeout for anonymous sets	CVE-2023-52620	
netfilter: nf_tables: disallow anonymous set with timeout flag	CVE-2024-26642	
net: tls: handle backlogging of crypto requests	CVE-2024-26584	
netfilter: nft_set_pipapo: skip inactive elements during set walk	CVE-2023-6817	
netfilter: nft_set_rbtree: skip sync GC for new elements in this transaction	CVE-2023-52433	
bpf: Defer the free of inner map when necessary	CVE-2023-52447	Origina
net: tls: fix use-after-free with partial reads and async decrypt	CVE-2024-26582	
netfilter: nft_set_rbtree: skip end interval element from gc	CVE-2024-26581	

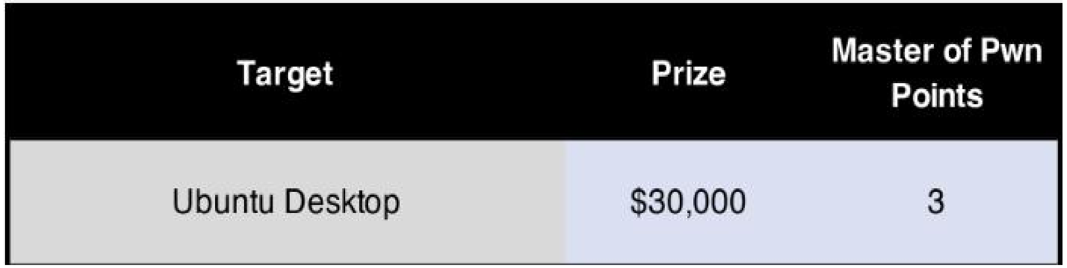
Pwn2own

2022



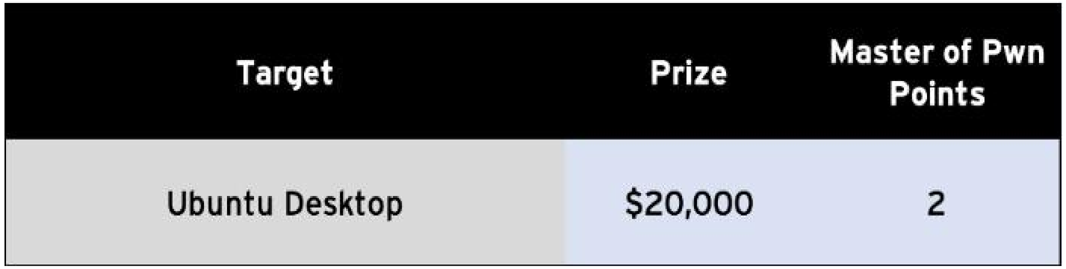
Target	Prize	Master of Pwn Points
Ubuntu Desktop	$40,000	4

2023



Target	Prize	Master of Pwn Points
Ubuntu Desktop	$30,000	3

2024



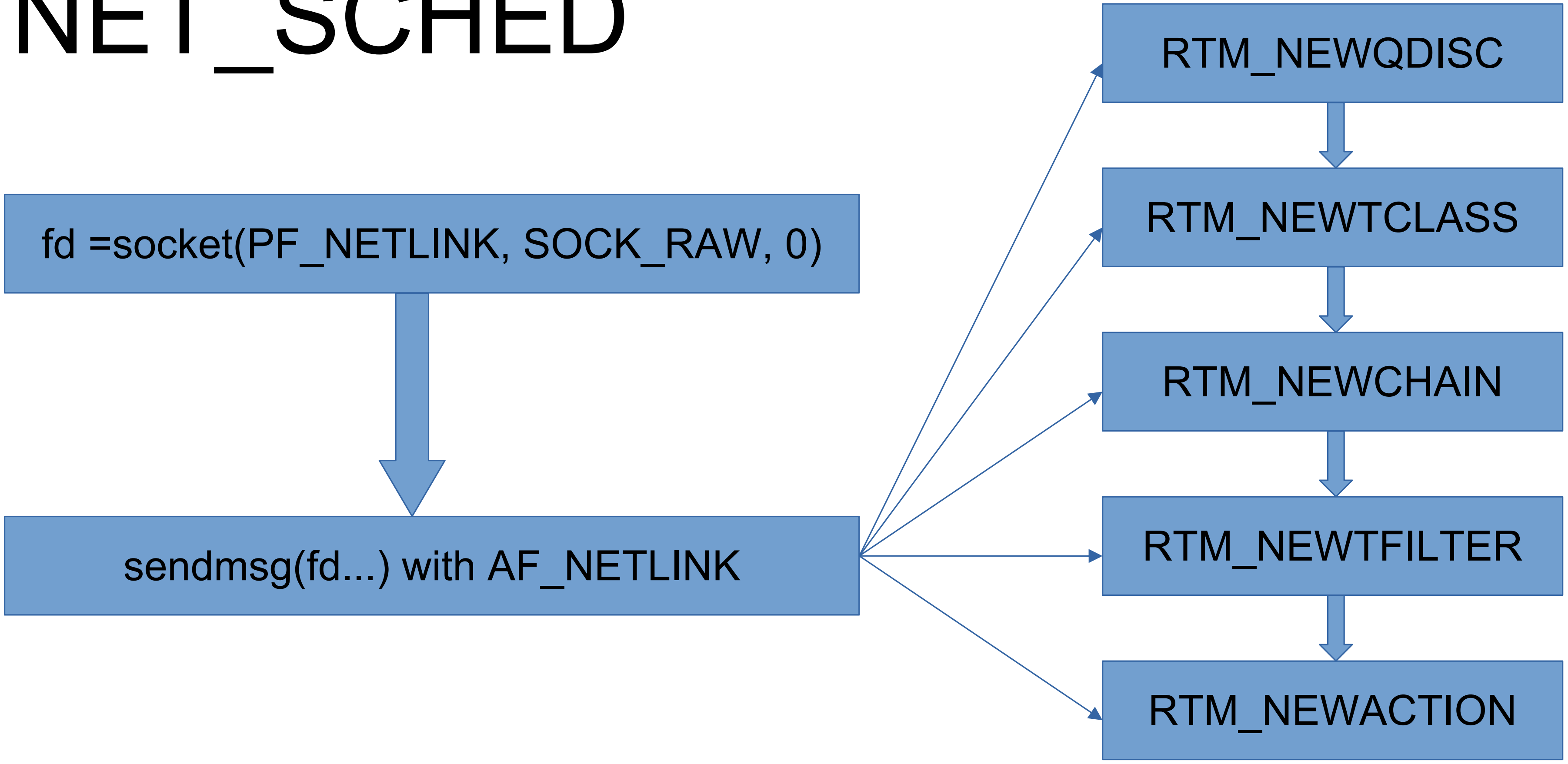
Target	Prize	Master of Pwn Points
Ubuntu Desktop	$20,000	2

Attack Surface

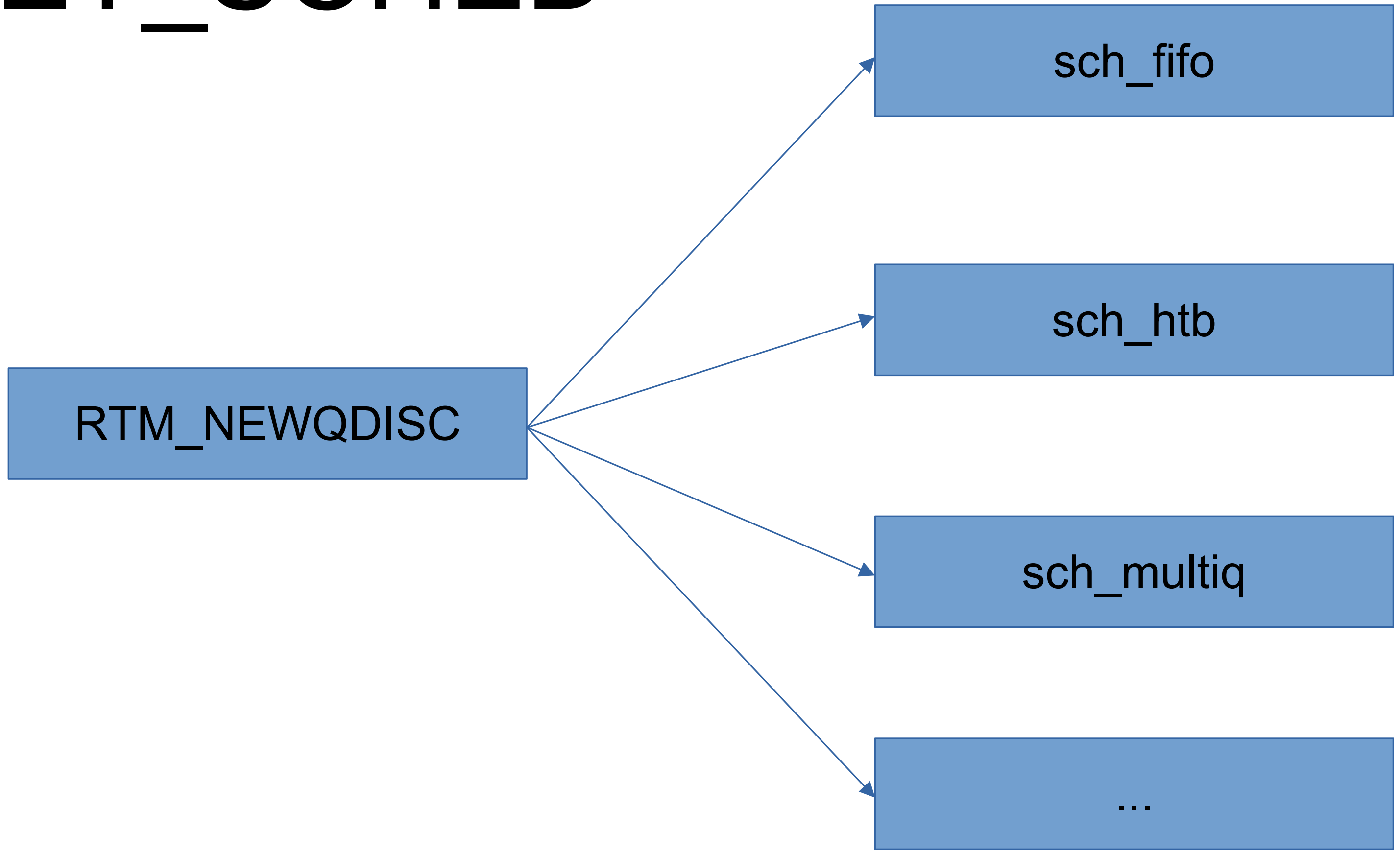
0x03 packet scheduler

- Advantage
  - Less attention
  - Complex
  - No privileges
- Disadvantage
  - Poor generalizability
  - New namespace

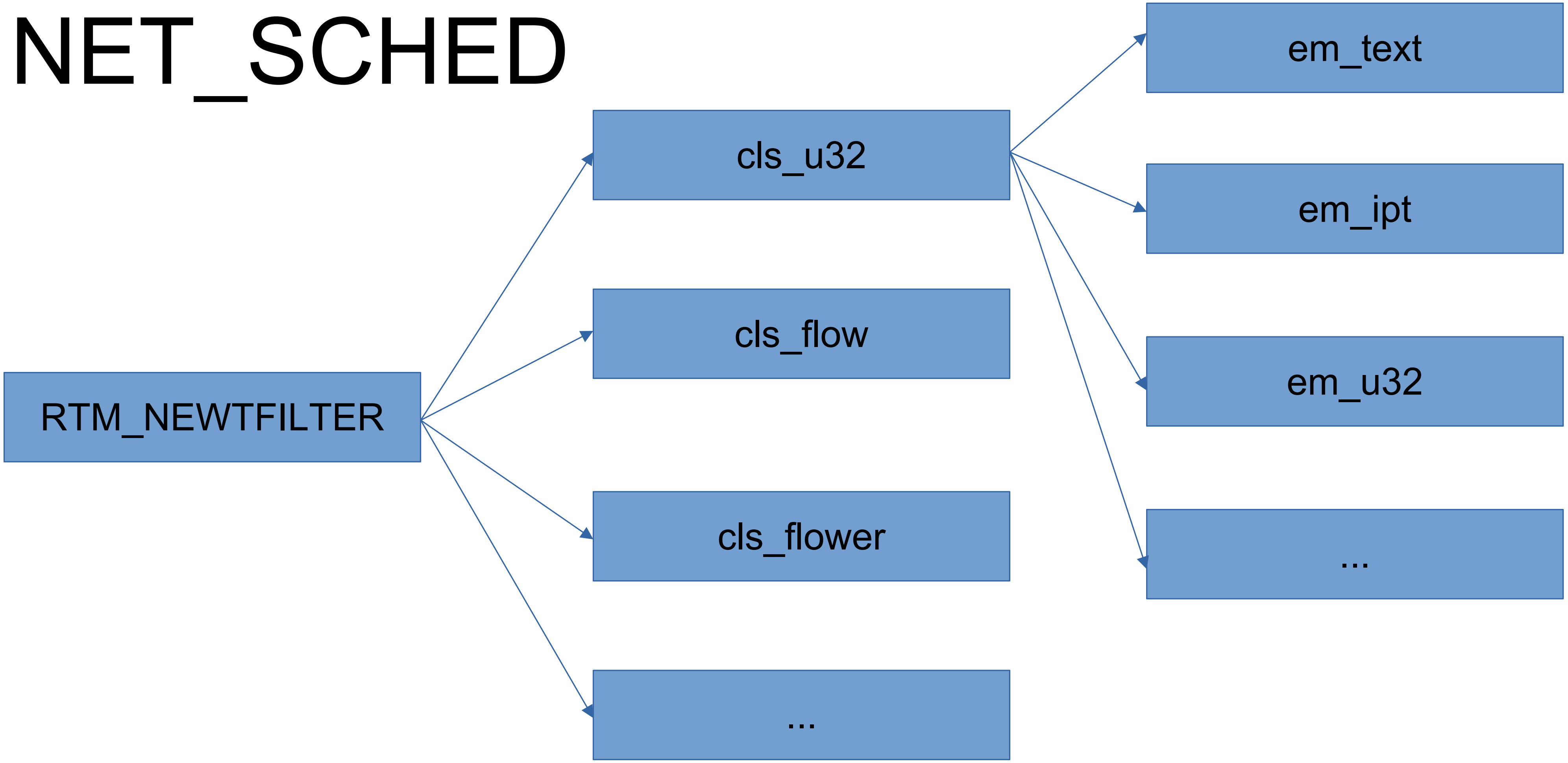
NET_SCHED


fd =socket(PF_NETLINK, SOCK_RAW, 0)
sendmsg(fd...) with AF_NETLINK
RTM_NEWQDISC
RTM_NEWTCLASS
RTM_NEWCHAIN
RTM_NEWTFILTER
RTM_NEWACTION

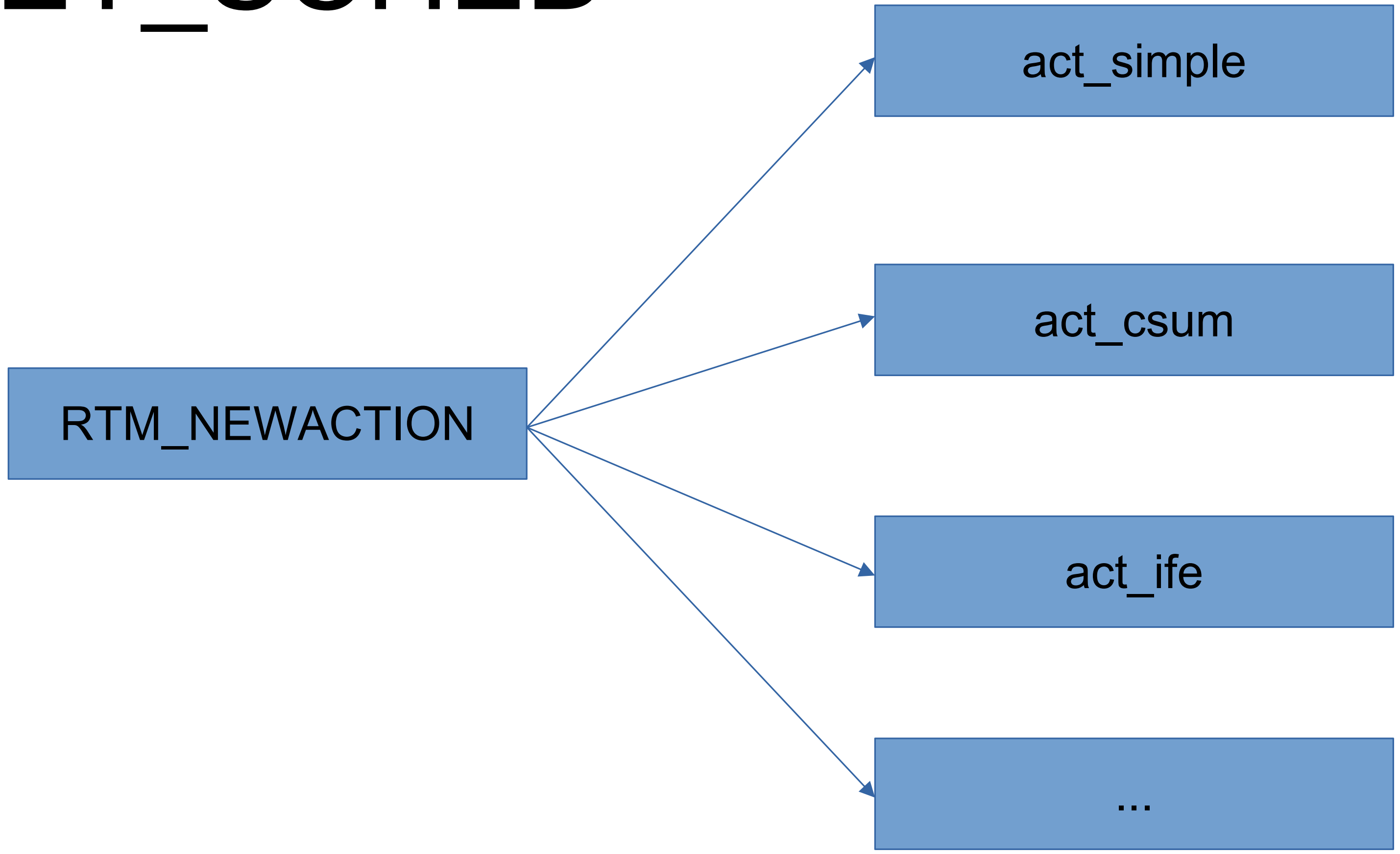
NET_SCHED



NET_SCHED


RTM_NEWTFILTER
cls_u32
cls_flow
cls_flower
...
em_text
em_ipt
em_u32
...

NET_SCHED



NET_SCHED

Fuzz……

NET_SCHED

No result.

NET_SCHED

Code reviewing…

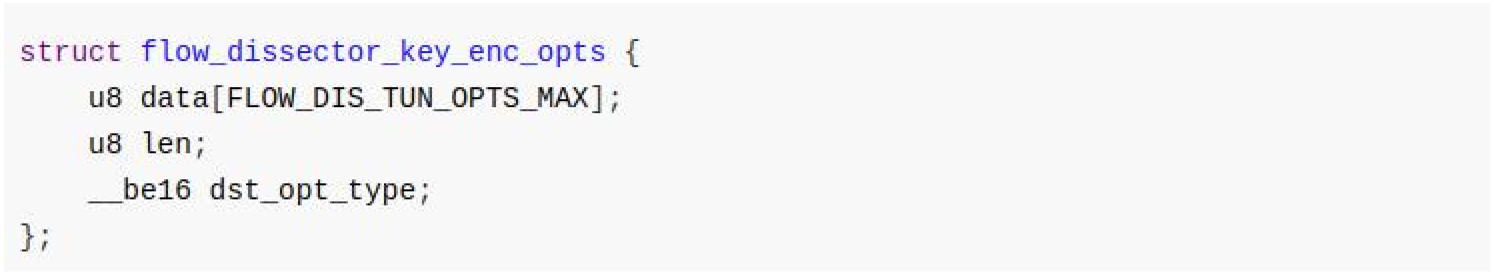
CVE-2023-35788

```c
static int fl_set_enc_opt()
{
...
    option_len = fl_set_geneve_opt(nla_opt_key, key,
                key_depth, option_len, extack);
    if (option_len < 0)
        return option_len;
        key->enc_opts.len += option_len;
...
}
```

```c
static int fl_set_geneve_opt(...)
{
...
    opt = (struct geneve_opt *)&key->enc_opts.data[key->enc_opts.len];  <--- [1]
    memset(opt, 0xff, option_len);
    opt->length = data_len / 4;
    opt->r1 = 0;
    opt->r2 = 0;
    opt->r3 = 0;
...
}
```

CVE-2023-35788

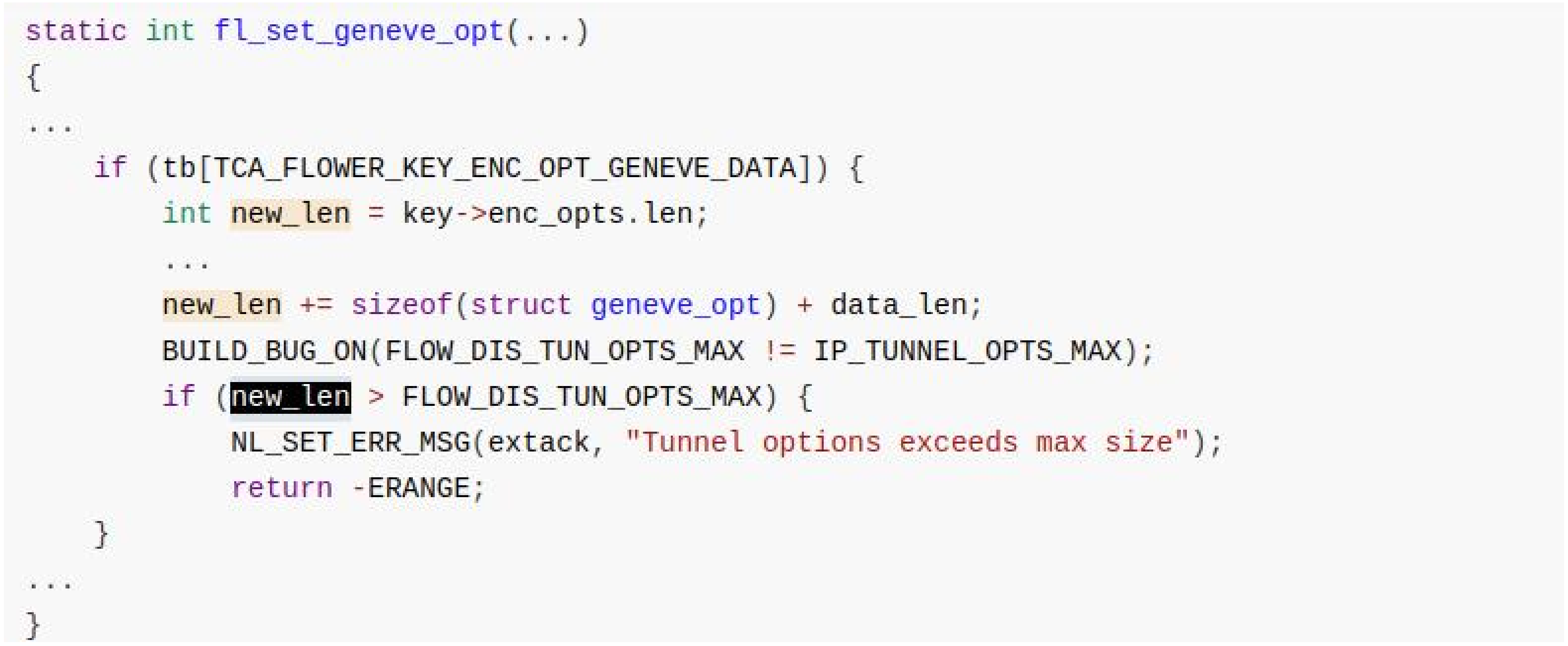
#define FLOW_DIS_TUN_OPTS_MAX 255



```
struct flow_dissector_key_enc_opts {
    u8 data[FLOW_DIS_TUN_OPTS_MAX];
    u8 len;
    __be16 dst_opt_type;
};
```

CVE-2023-35788

#define FLOW_DIS_TUN_OPTS_MAX 255

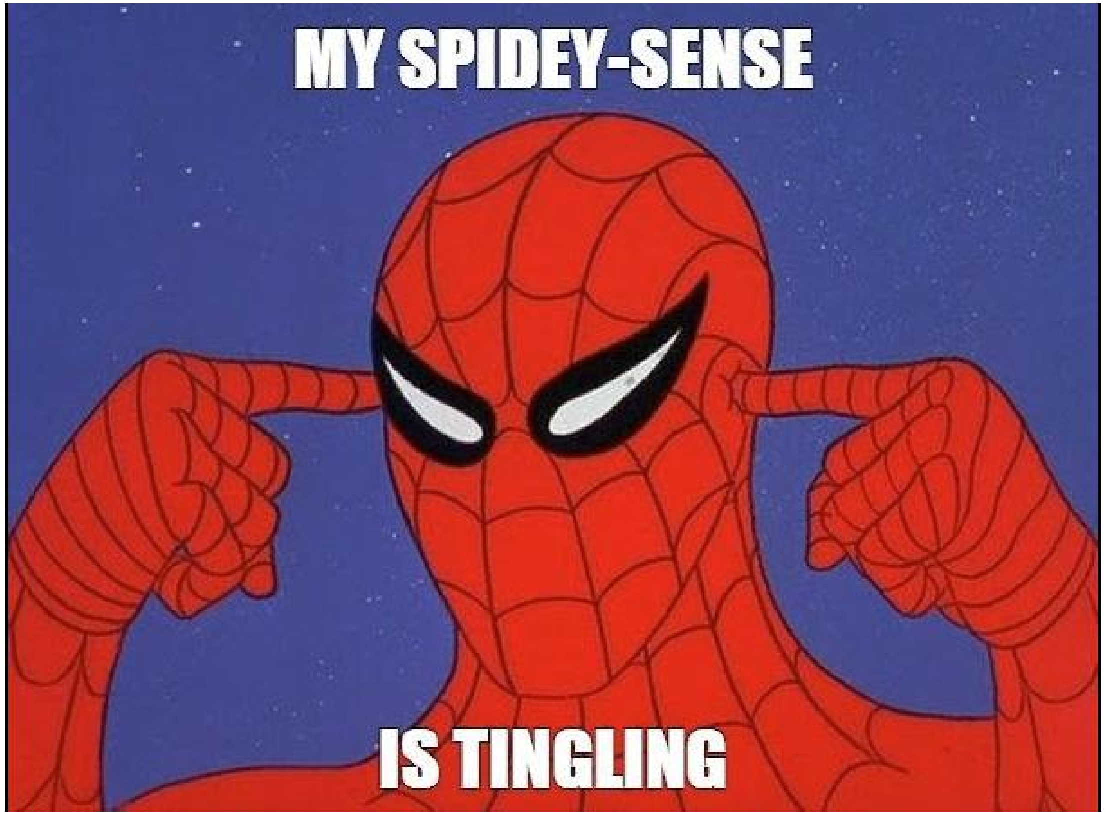


```
static int fl_set_geneve_opt(...)
{
...
    if (tb[TCA_FLOWER_KEY_ENC_OPT_GENEVE_DATA]) {
        int new_len = key->enc_opts.len;
        ...
        new_len += sizeof(struct geneve_opt) + data_len;
        BUILD_BUG_ON(FLOW_DIS_TUN_OPTS_MAX != IP_TUNNEL_OPTS_MAX);
        if (new_len > FLOW_DIS_TUN_OPTS_MAX) {
            NL_SET_ERR_MSG(extack, "Tunnel options exceeds max size");
            return -ERANGE;
    }
...
}
```

CVE-2023-35788

Not a bug?

CVE-2023-35788



CVE-2023-35788

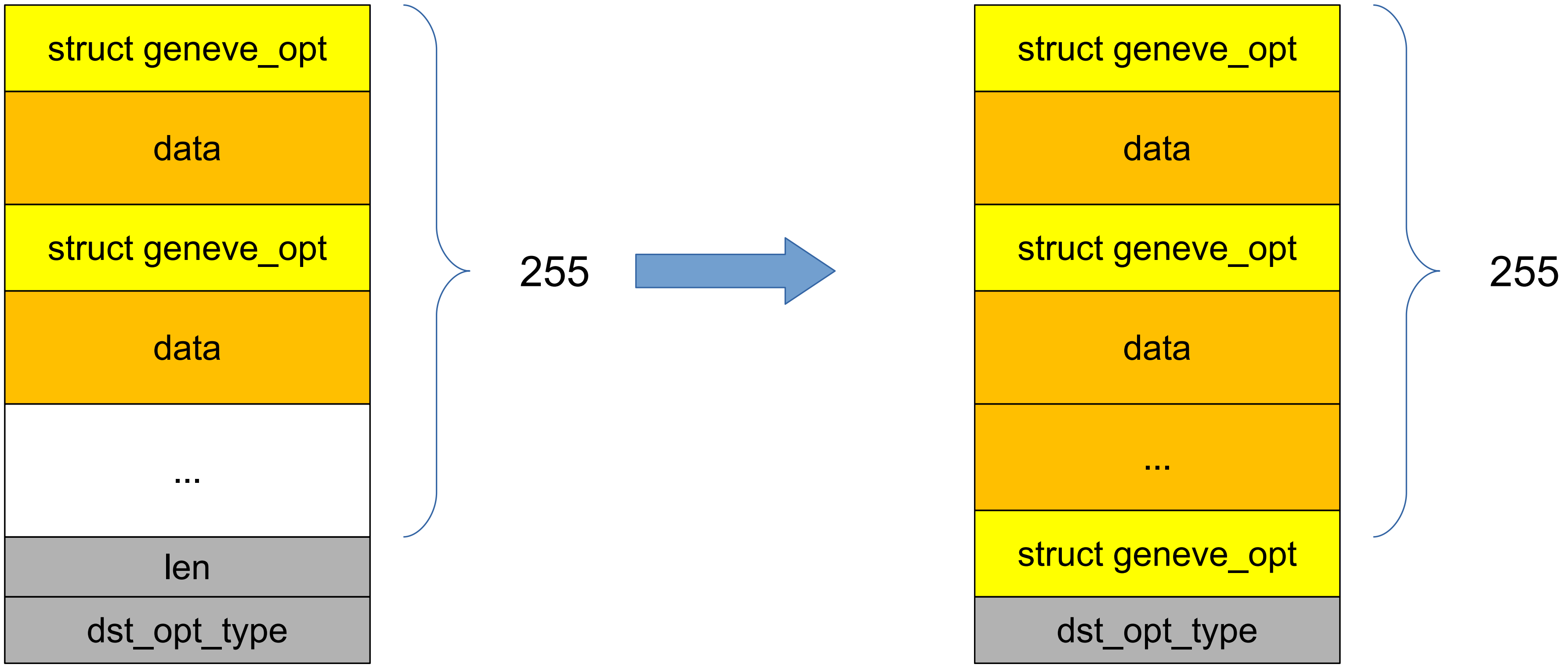
```
static int fl_set_geneve_opt(...)
{
...
    opt = (struct geneve_opt *)&key->enc_opts.data[key->enc_opts.len];  <--- [1]
    memset(opt, 0xff, option_len);
    opt->length = data_len / 4;
    opt->r1 = 0;
    opt->r2 = 0;
    opt->r3 = 0;
...
}
```

```
struct geneve_opt {
    __be16  opt_class;
    u8  type;
#ifdef __LITTLE_ENDIAN_BITFIELD
    u8  length:5;
    u8  r3:1;
    u8  r2:1;
    u8  r1:1;
#else
    u8  r1:1;
    u8  r2:1;
    u8  r3:1;
    u8  length:5;
#endif
    u8  opt_data[];
};
```

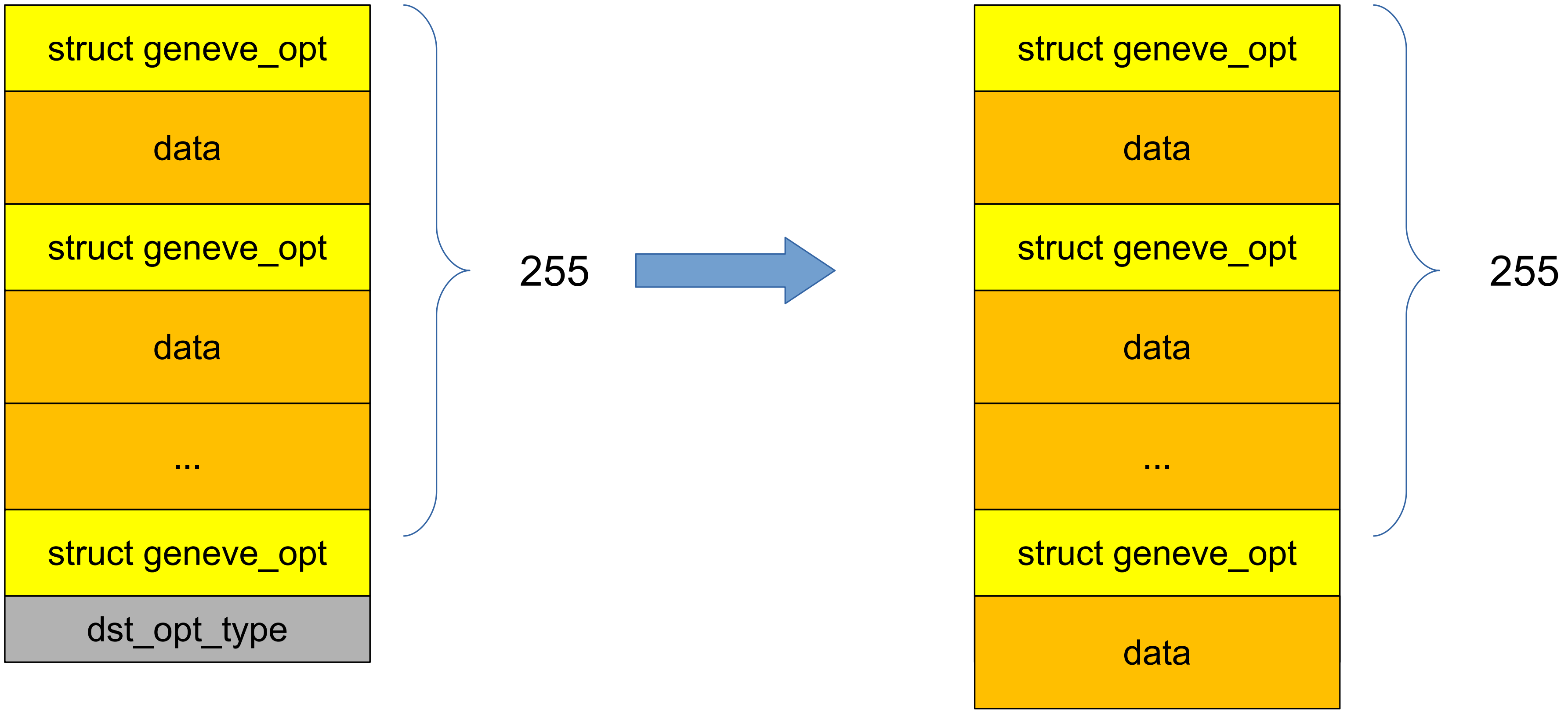
Off-by-one?

```
struct flow_dissector_key_enc_opts {
    u8 data[FLOW_DIS_TUN_OPTS_MAX];
    u8 len;
    __be16 dst_opt_type;
};
```

OFF-BY-ONE?


struct geneve_opt
data
struct geneve_opt
data
...
len
dst_opt_type
255
struct geneve_opt
data
struct geneve_opt
data
...
struct geneve_opt
dst_opt_type
255

OOB write


struct geneve_opt
data
struct geneve_opt
data
...
struct geneve_opt
dst_opt_type
255
struct geneve_opt
data
struct geneve_opt
data
...
struct geneve_opt
data
255

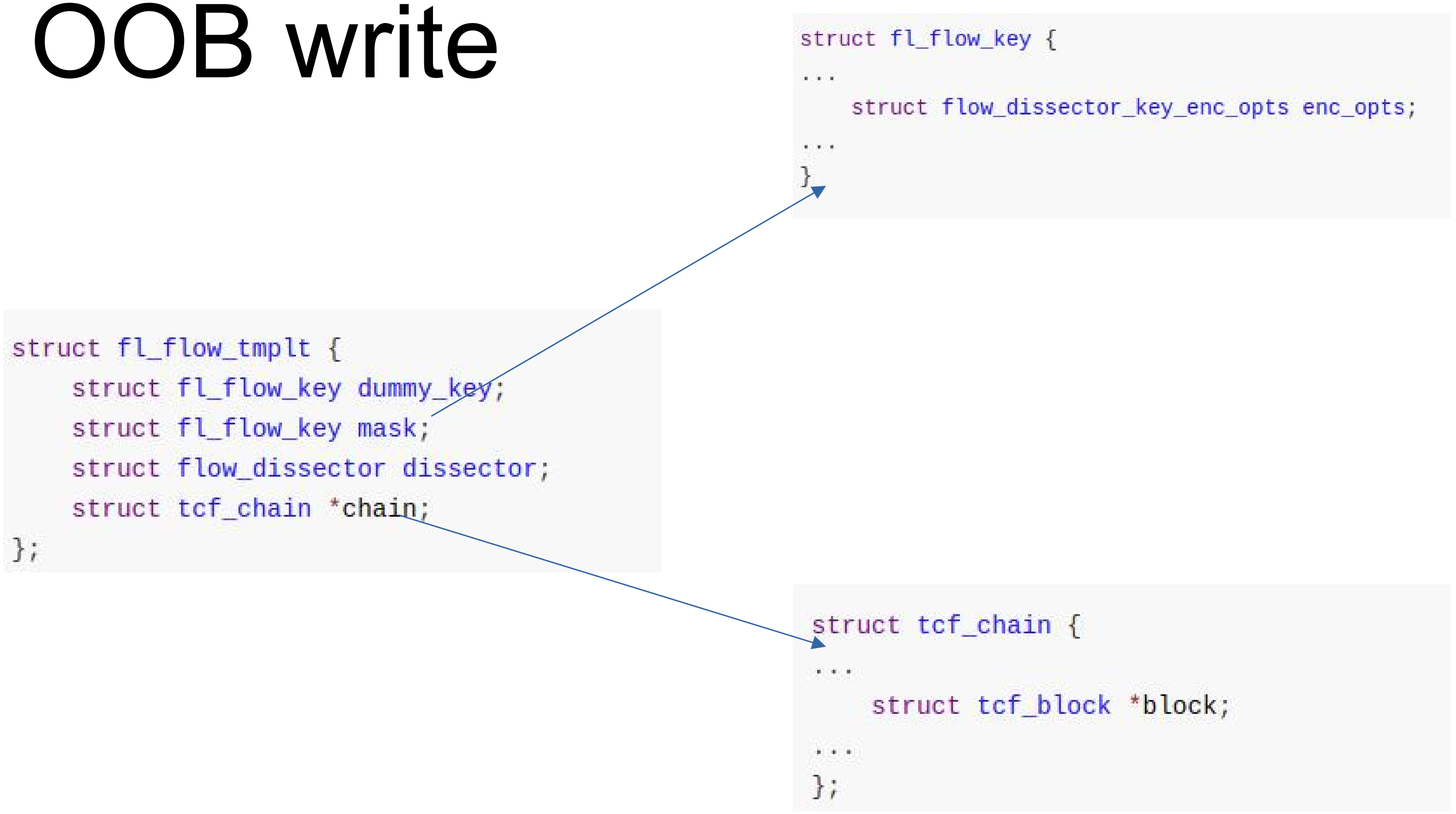
OOB write

```c
static int fl_set_geneve_opt(const struct nlattr *nla, struct fl_flow_key *key,
                             int depth, int option_len,
                             struct netlink_ext_ack *extack)
{
...
    err = nla_parse_nested_deprecated(tb,
                                      TCA_FLOWER_KEY_ENC_OPT_GENEVE_MAX,
                                      nla, geneve_opt_policy, extack);
    if (err < 0)
        return err;
...
}
```

```c
static const struct nla_policy
geneve_opt_policy[TCA_FLOWER_KEY_ENC_OPT_GENEVE_MAX + 1] = {
    [TCA_FLOWER_KEY_ENC_OPT_GENEVE_CLASS]      = { .type = NLA_U16 },
    [TCA_FLOWER_KEY_ENC_OPT_GENEVE_TYPE]       = { .type = NLA_U8 },
    [TCA_FLOWER_KEY_ENC_OPT_GENEVE_DATA]       = { .type = NLA_BINARY,
                                .len = 128 },
};
```

Only 128 each time

OOB write

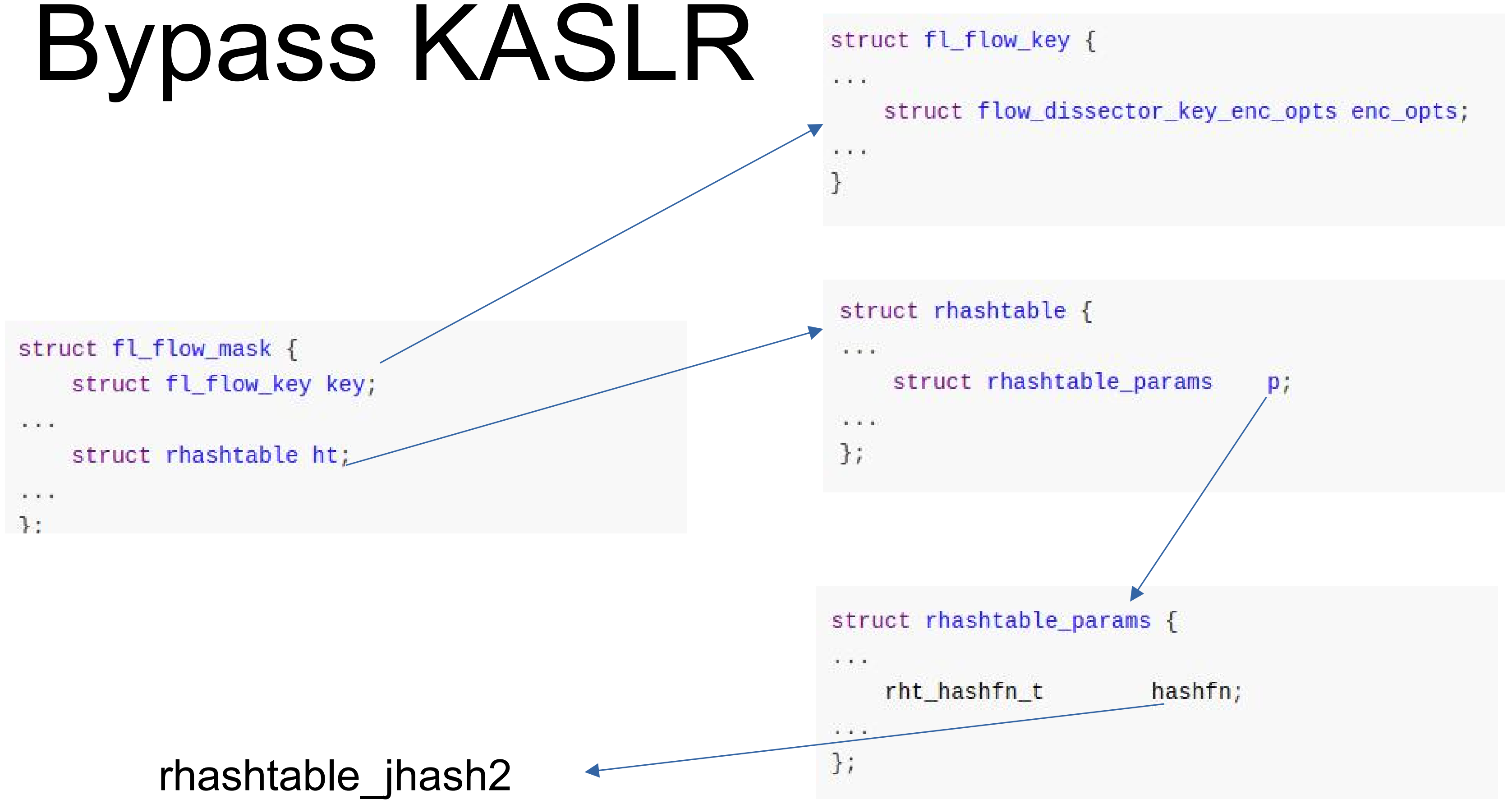


```
struct fl_flow_key {
...
    struct flow_dissector_key_enc_opts enc_opts;
...
}
```

```
struct fl_flow_tmplt {
    struct fl_flow_key dummy_key;
    struct fl_flow_key mask;
    struct flow_dissector dissector;
    struct tcf_chain *chain;
};
```

```
struct tcf_chain {
...
    struct tcf_block *block;
...
};
```

Bypass KASLR



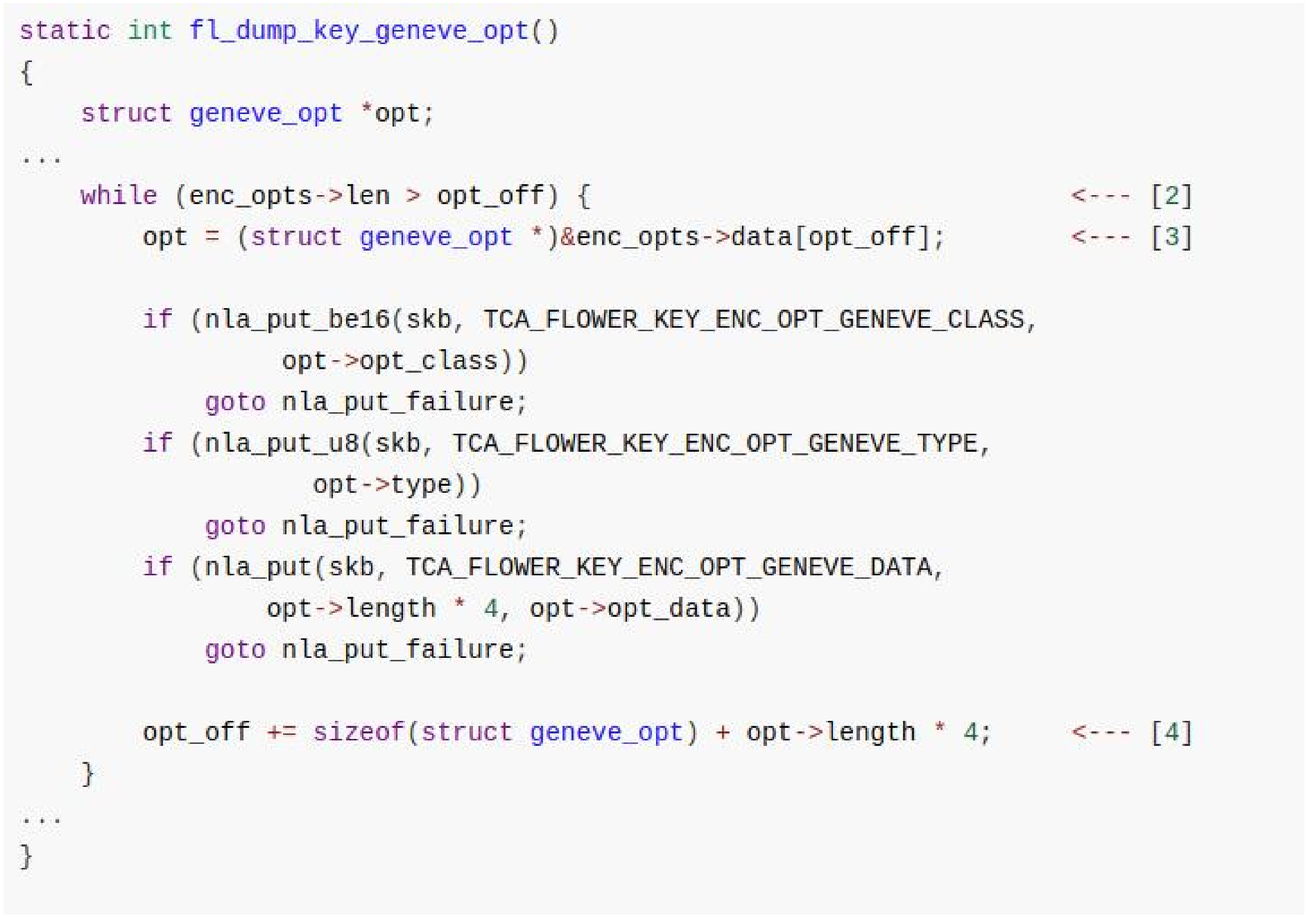
```
struct fl_flow_key {
...
    struct flow_dissector_key_enc_opts enc_opts;
...
}
```

```
struct rhashtable {
...
    struct rhashtable_params    p;
...
};
```

```
struct fl_flow_mask {
    struct fl_flow_key key;
...
    struct rhashtable ht;
...
};
```

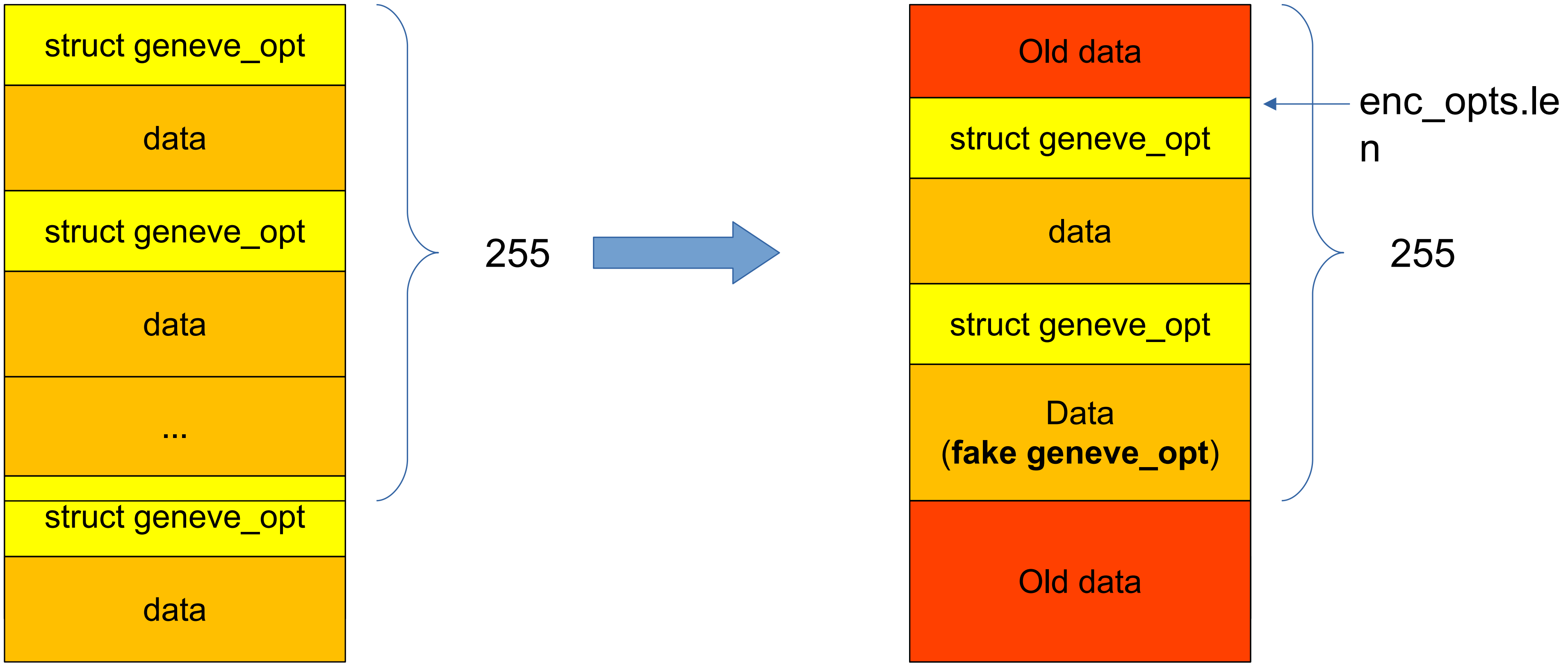
```
struct rhashtable_params {
...
    rht_hashfn_t         hashfn;
...
};
```

Bypass KASLR

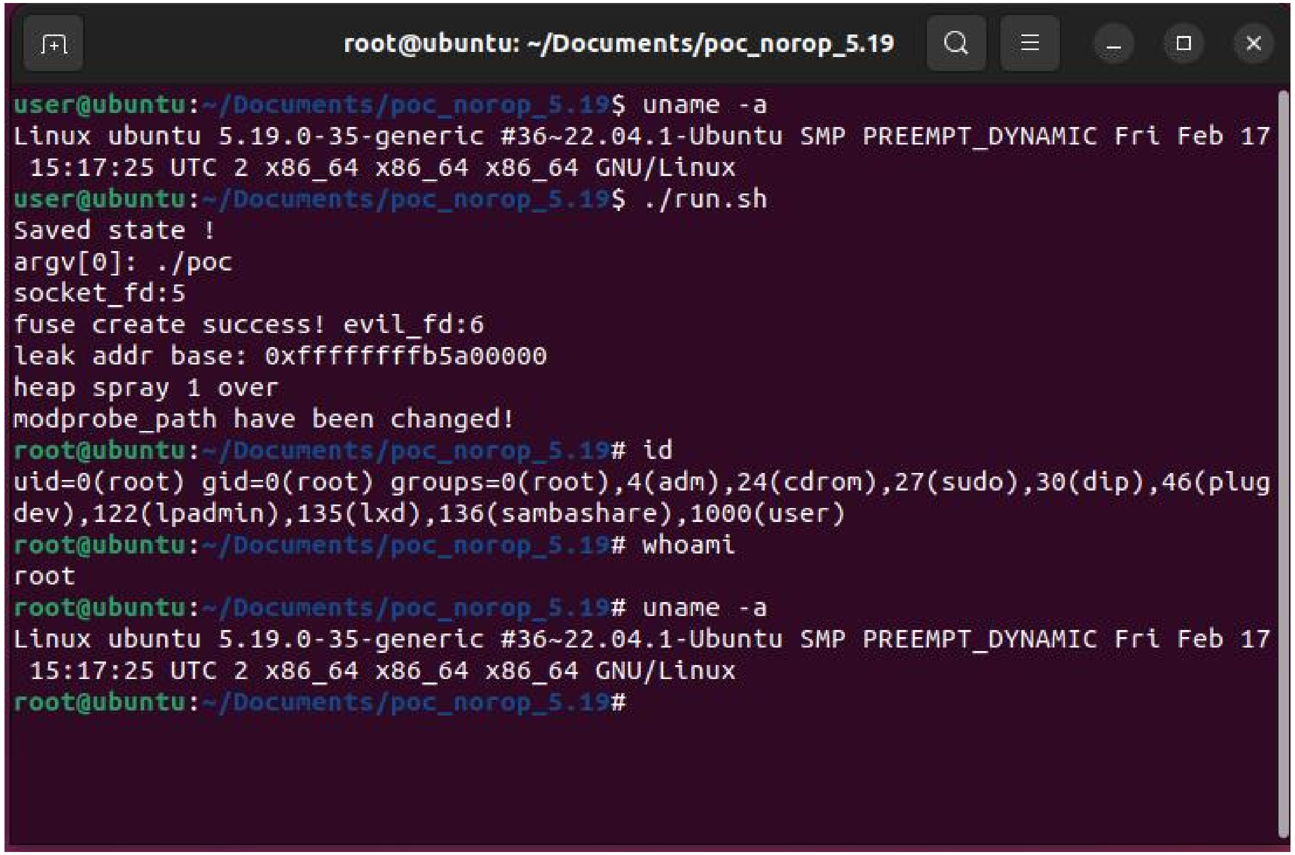


```
static int fl_dump_key_geneve_opt()
{
    struct geneve_opt *opt;
...
    while (enc_opts->len > opt_off) {                               <--- [2]
        opt = (struct geneve_opt *)&enc_opts->data[opt_off];        <--- [3]

        if (nla_put_be16(skb, TCA_FLOWER_KEY_ENC_OPT_GENEVE_CLASS,
                 opt->opt_class))
            goto nla_put_failure;
        if (nla_put_u8(skb, TCA_FLOWER_KEY_ENC_OPT_GENEVE_TYPE,
                   opt->type))
            goto nla_put_failure;
        if (nla_put(skb, TCA_FLOWER_KEY_ENC_OPT_GENEVE_DATA,
                opt->length * 4, opt->opt_data))
            goto nla_put_failure;

        opt_off += sizeof(struct geneve_opt) + opt->length * 4;     <--- [4]
    }
...
}
```

Bypass KASLR



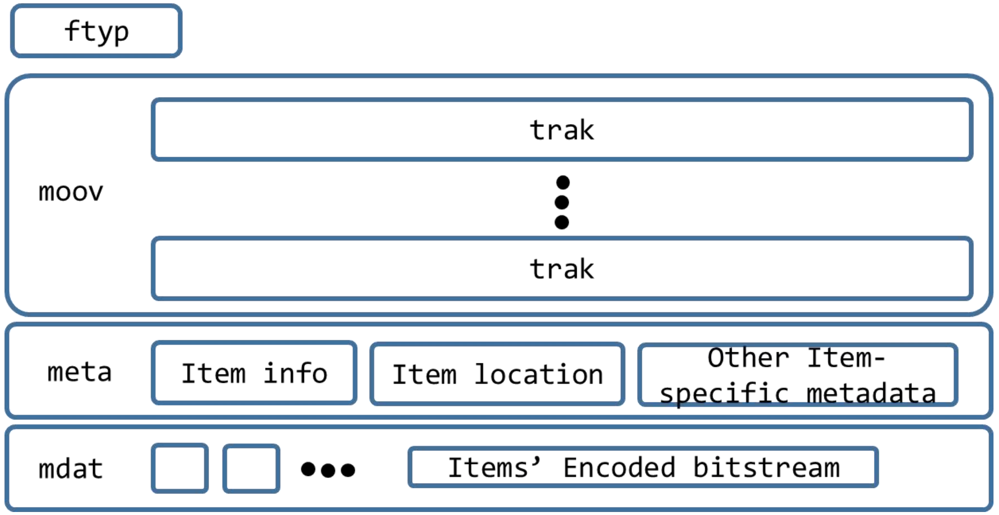
CVE-2023-35788

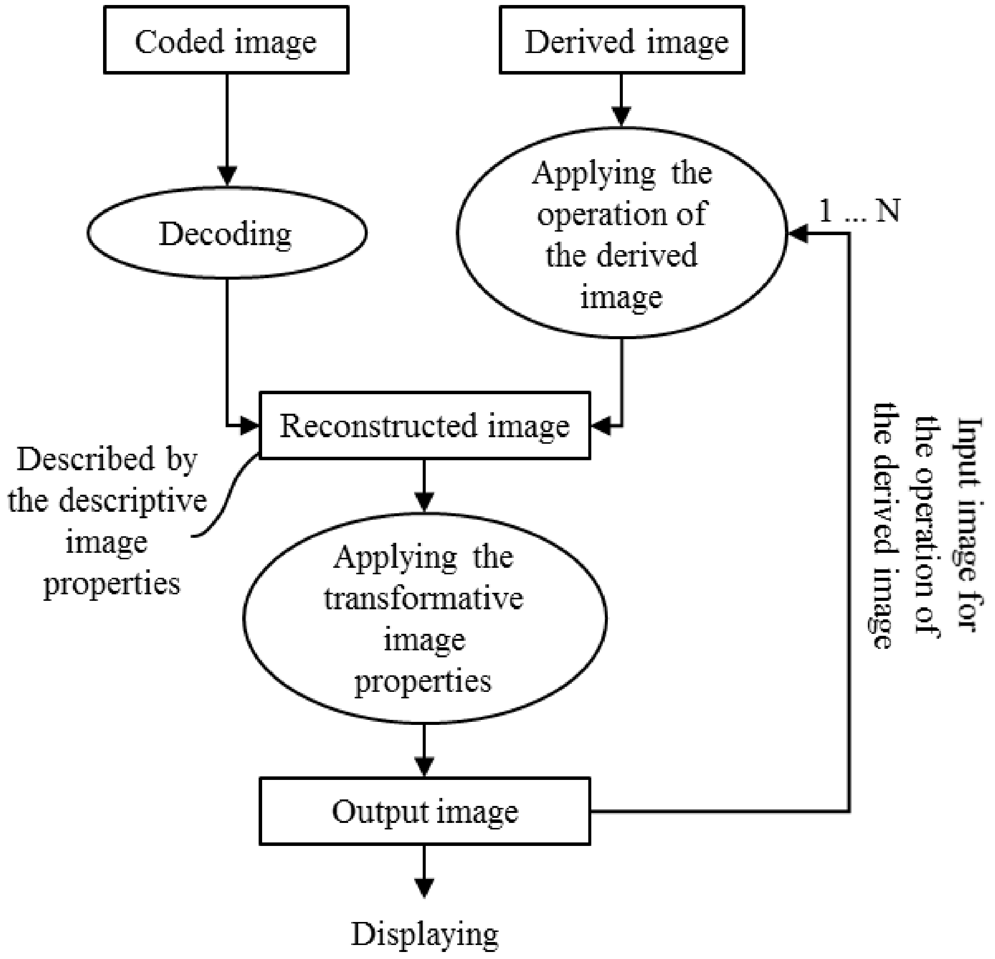


```
user@ubuntu:~/Documents/poc_norop_5.19$ uname -a
Linux ubuntu 5.19.0-35-generic #36~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Feb 17
 15:17:25 UTC 2 x86_64 x86_64 x86_64 GNU/Linux
user@ubuntu:~/Documents/poc_norop_5.19$ ./run.sh
Saved state !
argv[0]: ./poc
socket_fd:5
fuse create success! evil_fd:6
leak addr base: 0xffffffffb5a00000
heap spray 1 over
modprobe_path have been changed!
root@ubuntu:~/Documents/poc_norop_5.19# id
uid=0(root) gid=0(root) groups=0(root),4(adm),24(cdrom),27(sudo),30(dip),46(plug
dev),122(lpadmin),135(lxd),136(sambashare),1000(user)
root@ubuntu:~/Documents/poc_norop_5.19# whoami
root
root@ubuntu:~/Documents/poc_norop_5.19# uname -a
Linux ubuntu 5.19.0-35-generic #36~22.04.1-Ubuntu SMP PREEMPT_DYNAMIC Fri Feb 17
 15:17:25 UTC 2 x86_64 x86_64 x86_64 GNU/Linux
root@ubuntu:~/Documents/poc_norop_5.19#
```

https://shuffleteam.github.io/research/CVE-2023-35788/

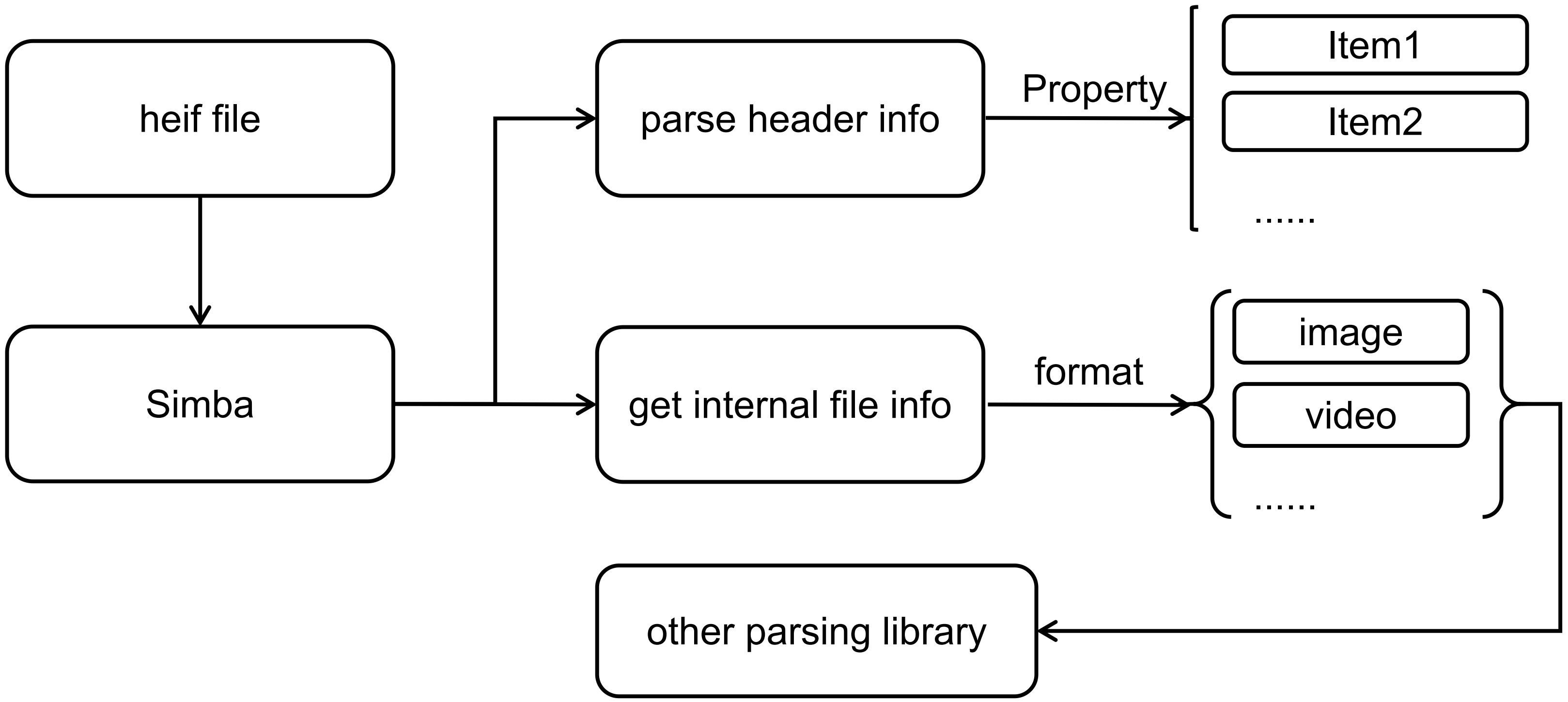
What is .Heif


ftyp
moov
trak
trak
meta
Item info
Item location
Other Item-specific metadata
mdat
Items' Encoded bitstream


Coded image
Derived image
Decoding
Applying the operation of the derived image
1 ... N
Reconstructed image
Described by the descriptive image properties
Applying the transformative image properties
Input image for the operation of the derived image
Output image
Displaying

Simba - The parsing logic

- The image parsing library for heif on Samsung

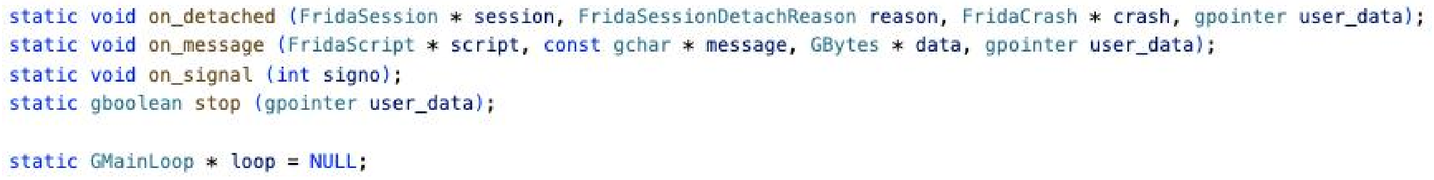


Fuzzing

- Approximately 5-6 vulnerabilities found
- some are actually similar errors
- occurred in the earlier parse function
- The fuzzer keeps triggering crashes on these old issues
- Which means that the fuzz testing is not accurate enough(seeds, cov..)
- improve -> "Not too bad" fuzzer

  -> radamsa + frida gum on phone

  -> frida-core-example.c



```
static void on_detached (FridaSession * session, FridaSessionDetachReason reason, FridaCrash * crash, gpointer user_data);
static void on_message (FridaScript * script, const gchar * message, GBytes * data, gpointer user_data);
static void on_signal (int signo);
static gboolean stop (gpointer user_data);

static GMainLoop * loop = NULL;
```

My CVEs(2022.4)

- simba(samsung)
- CVE-2022-26093, CVE-2022-26094, CVE-2022-26095, CVE-2022-26096, CVE-2022-26097, CVE-2022-26098, CVE-2022-26099, CVE-2022-27567, CVE-2022-27568, CVE-2022-27569, CVE-2022-27570, CVE-2022-27571, CVE-2022-27572, CVE-2022-27573, CVE-2022-27574,
- heif(aosp)
- CVE-2021-39804

Fuzzing method compared

- Mateusz Jurczyk(Project Zero),: Exploit Samsung MMS (zero-click) through qmg files.
- Flanker, DroidCorn: Closed-source Android Binary Fuzzing
- Dawuge: Frida fuzzer
- _pox_(Singular Security Lab), Android JNI Fuzzing, Zer0Con2022
- Hao Xiong, Qinming Dai, Rui Chang, Yajin Zhou(Zhejiang University): Fuzzing Samsung's closed-source libraries as if on a real device, Zer0Con2023

Fuzzing test has been a general approach to find bugs in closed-source native libraries.

Fuzzing method compared

- For mobile testing targets, code coverage should be the most effective fuzz testing metric.

Interface / Target	Parameters are memory addresses or objects, string, fd, etc.	Parameters are objects of the java layer, etc.
Shared Library	user-mode + qemu	jvm + qemu/frida
Service Process	frida	jvm + frida
Network Process/Session	frida	jvm + frida

- So is there anything else we should pay attention to?

Fuzzing is the best?

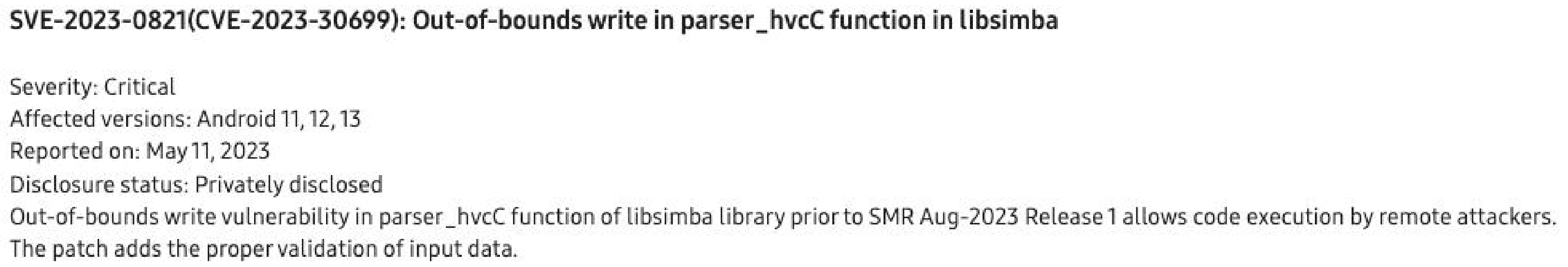
- CVE-2020-15999
  - Missed by fuzz testing options
- CVE-2023-4863, CVE-2023-41064
  - Missed because of complex conditions and deep trigger path
- fuzzing corpus
- problems with the fuzzer itself
- ...

It's time to review

- But this time we focused on digging out previously fixed patches and some types of vulnerabilities that are not easily discovered by fuzz testing.

- Boom! CVE-2023-30699!

CVE-2023-30699

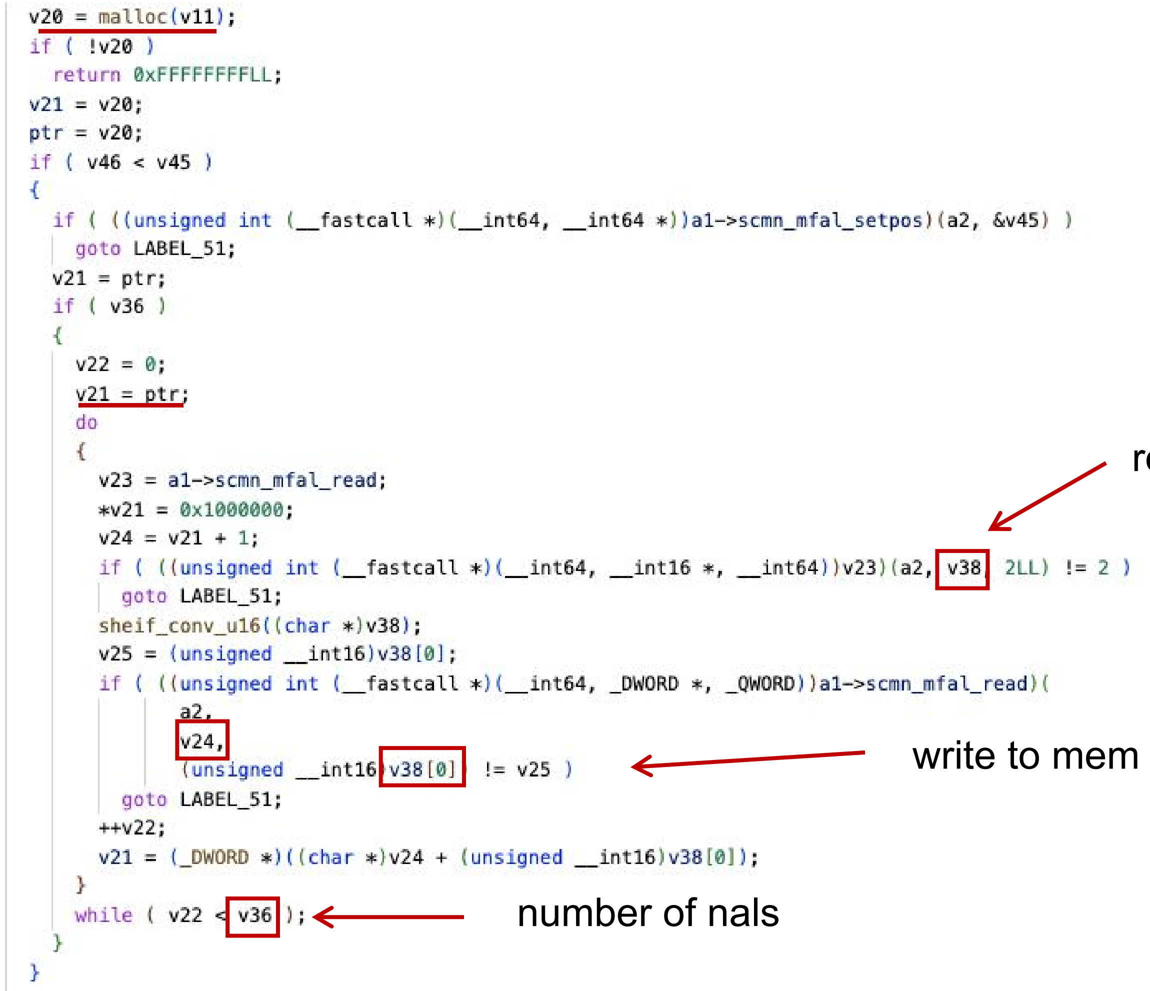


SVE-2023-0821(CVE-2023-30699): Out-of-bounds write in parser_hvcC function in libsimba

Severity: Critical
Affected versions: Android 11, 12, 13
Reported on: May 11, 2023
Disclosure status: Privately disclosed
Out-of-bounds write vulnerability in parser_hvcC function of libsimba library prior to SMR Aug-2023 Release 1 allows code execution by remote attackers.
The patch adds the proper validation of input data.

In fact, this is not a complex type of vulnerability
Why hadn't this bug been found by fuzzer?

CVE-2023-30699



```
v20 = malloc(v11);
if ( !v20 )
  return 0xFFFFFFFFLL;
v21 = v20;
ptr = v20;
if ( v46 < v45 )
{
  if ( ((unsigned int (__fastcall *)(__int64, __int64 *))a1->scmn_mfal_setpos)(a2, &v45) )
    goto LABEL_51;
  v21 = ptr;
  if ( v36 )
  {
    v22 = 0;
    v21 = ptr;
    do
    {
      v23 = a1->scmn_mfal_read;
      *v21 = 0x1000000;
      v24 = v21 + 1;
      if ( ((unsigned int (__fastcall *)(__int64, __int16 *, __int64))v23)(a2, v38, 2LL) != 2 )
        goto LABEL_51;
      sheif_conv_u16((char *)v38);
      v25 = (unsigned __int16)v38[0];
      if ( ((unsigned int (__fastcall *)(__int64, _DWORD *, _QWORD))a1->scmn_mfal_read)(
             a2,
             v24,
             (unsigned __int16)v38[0]) != v25 )
        goto LABEL_51;
      ++v22;
      v21 = (_DWORD *)((char *)v24 + (unsigned __int16)v38[0]);
    }
    while ( v22 < v36 );
  }
}
```

read size from heif file data

write to mem with control size

number of nals

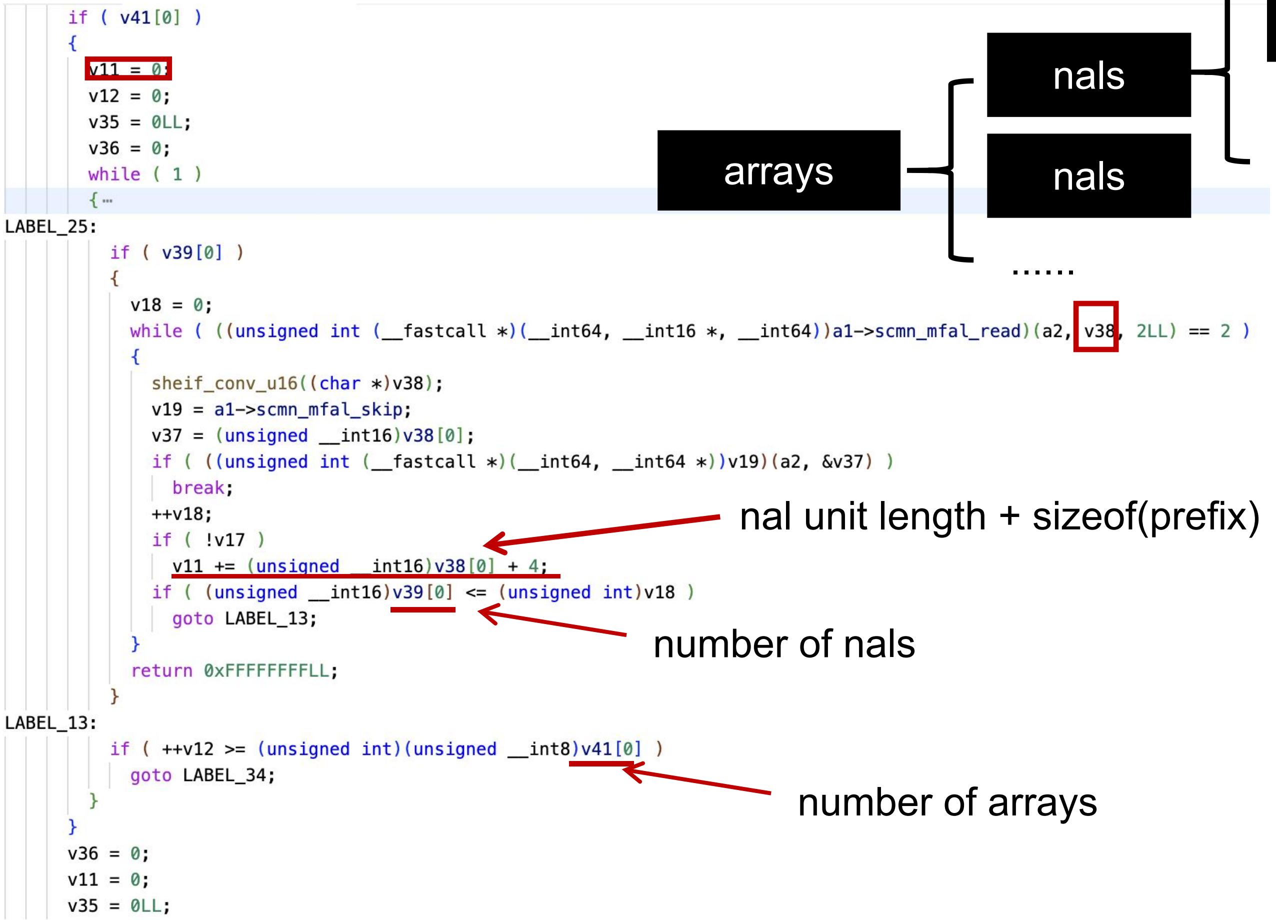
CVE-2023-30699

The reason why this malloc can allocate a smaller value is due to an integer truncation issue. The parameter passed to malloc undergoes a type conversion, being converted to an unsigned int parameter. The maximum value of this unsigned int is only 0xffffffff. By controlling v41[0] (maximum value of 0xff), v39[0] (maximum value of 0xffff), v38[0] (0xffff), we can achieve a theoretical maximum value of 0xff * 0xffff * (0xffff + 4). This value is already small enough after the truncation of the malloc parameter, laying the groundwork for subsequent out-of-bounds writes.

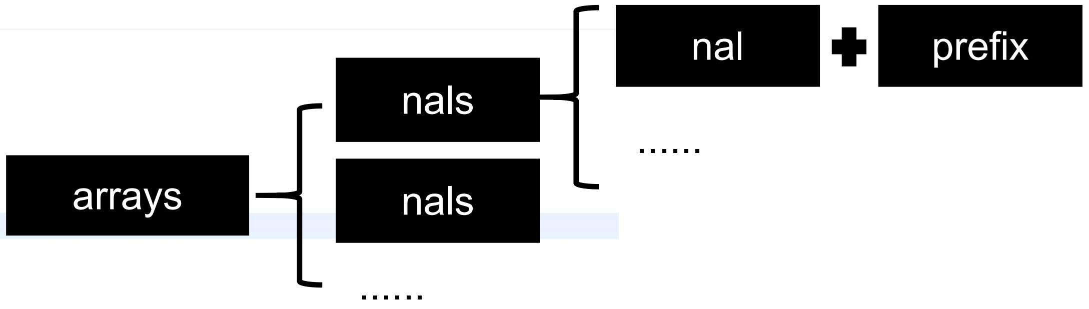
```
.text:0000000000030F40 loc_30F40                              ; CODE XREF: parser_hvcC+19C↑j
.text:0000000000030F40                AND             X0, X28, #0xFFFFFFFF ; size           ---->  convert to uint32
.text:0000000000030F44                BL              .malloc
```

```
v41[0] = number of nals
v39[0] = number of arrays ?
```

CVE-2023-30699



```
    if ( v41[0] )
    {
      v11 = 0;
      v12 = 0;
      v35 = 0LL;
      v36 = 0;
      while ( 1 )
      {…
LABEL_25:
        if ( v39[0] )
        {
          v18 = 0;
          while ( ((unsigned int (__fastcall *)(__int64, __int16 *, __int64))a1->scmn_mfal_read)(a2, v38, 2LL) == 2 )
          {
            sheif_conv_u16((char *)v38);
            v19 = a1->scmn_mfal_skip;
            v37 = (unsigned __int16)v38[0];
            if ( ((unsigned int (__fastcall *)(__int64, __int64 *))v19)(a2, &v37) )
              break;
            ++v18;
            if ( !v17 )
              v11 += (unsigned __int16)v38[0] + 4;
            if ( (unsigned __int16)v39[0] <= (unsigned int)v18 )
              goto LABEL_13;
          }
          return 0xFFFFFFFFLL;
        }
LABEL_13:
        if ( ++v12 >= (unsigned int)(unsigned __int8)v41[0] )
          goto LABEL_34;
      }
    }
    v36 = 0;
    v11 = 0;
    v35 = 0LL;
```



Part III: Enhance Fuzzers

Why fuzz failed ?

- The model of this vulnerability is difficult to construct
- Deep trigger path
- Complex trigger conditions
- Need to enter the same path repeatedly
- Requires unique value to trigger
- Overflow within a structure
- Virtual network card

$\Big\}$

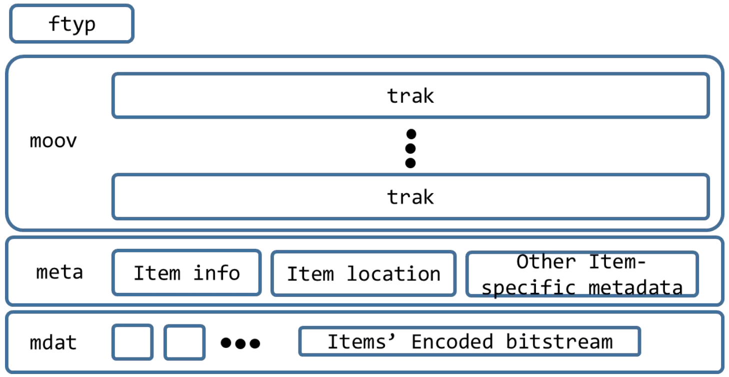
- Structure
- Path
- DFG

Enhance Fuzzers

- Structure -> substructure(Loop)

- Path -> Load, Cmp, Switch
  - Multiple tests on the same path
  - Different/unique values

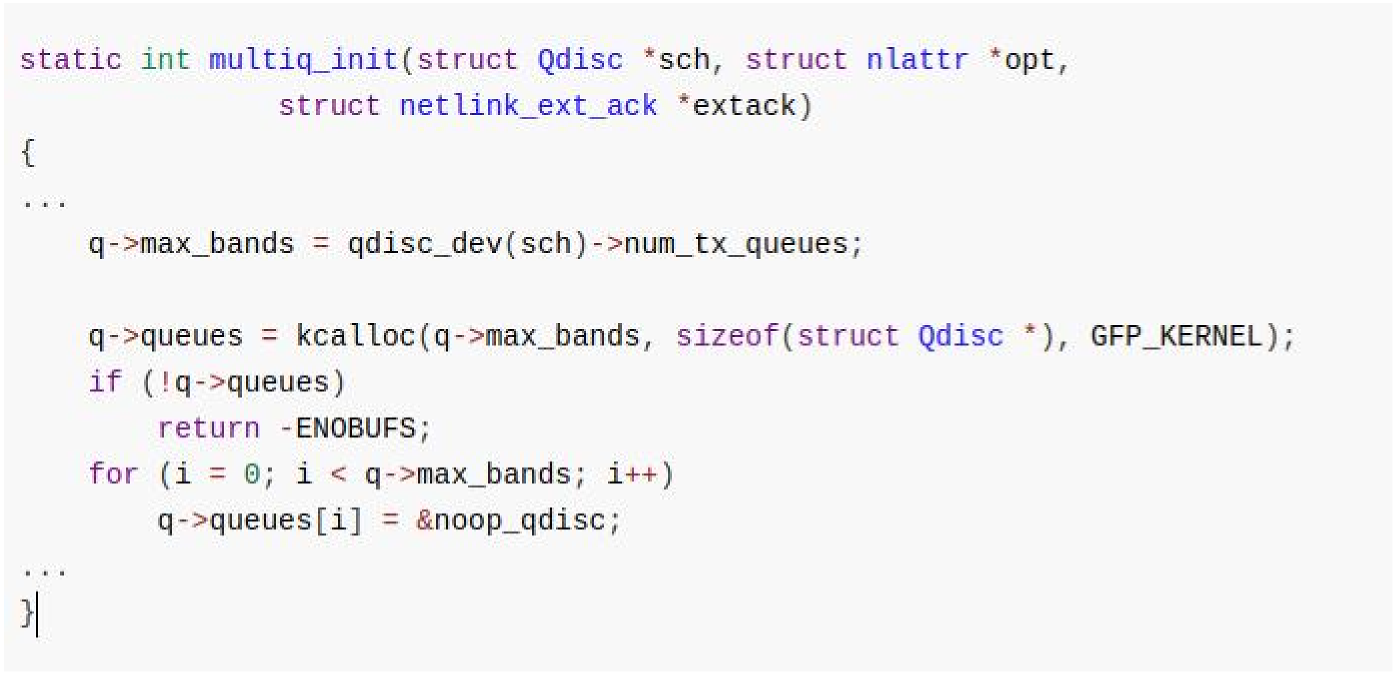
- DFG



New Vulns

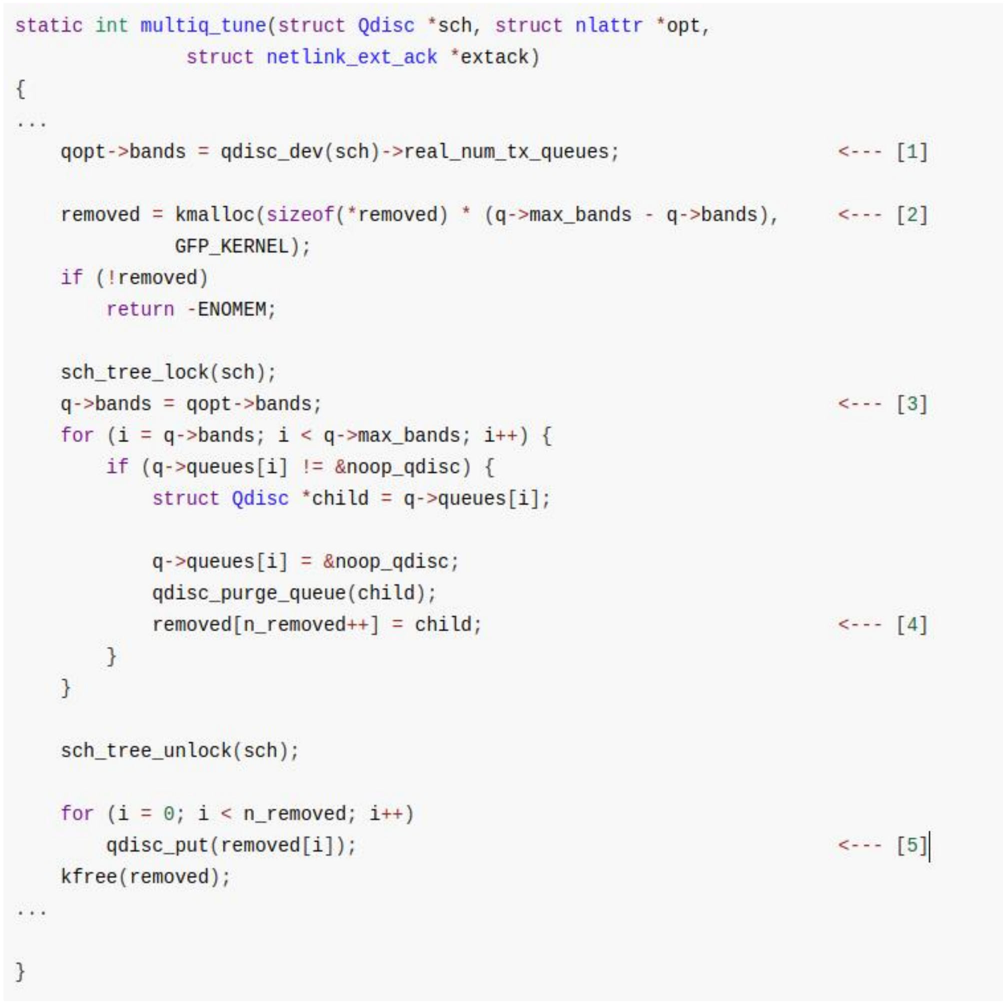
- Samsung: 8 vulnerabilities found （CVE-2024-20812～CVE-2024-20814, CVE-2024-20817～CVE-2024-20819...）
  - Bootloader(S* series): CVE-2023-42561, CVE-2023-21489
- Honor: 3 vulnerabilities found
- Huawei: 2 vulnerabilities found
- Linux Kernel: CVE-2024-36978

CVE-2024-36978



```
static int multiq_init(struct Qdisc *sch, struct nlattr *opt,
               struct netlink_ext_ack *extack)
{
...
    q->max_bands = qdisc_dev(sch)->num_tx_queues;

    q->queues = kcalloc(q->max_bands, sizeof(struct Qdisc *), GFP_KERNEL);
    if (!q->queues)
        return -ENOBUFS;
    for (i = 0; i < q->max_bands; i++)
        q->queues[i] = &noop_qdisc;
...
}
```

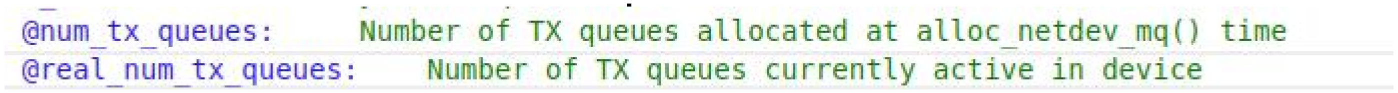
CVE-2024-36978



```
static int multiq_tune(struct Qdisc *sch, struct nlattr *opt,
               struct netlink_ext_ack *extack)
{
...
    qopt->bands = qdisc_dev(sch)->real_num_tx_queues;                   <--- [1]

    removed = kmalloc(sizeof(*removed) * (q->max_bands - q->bands),     <--- [2]
              GFP_KERNEL);
    if (!removed)
        return -ENOMEM;

    sch_tree_lock(sch);
    q->bands = qopt->bands;                                             <--- [3]
    for (i = q->bands; i < q->max_bands; i++) {
        if (q->queues[i] != &noop_qdisc) {
            struct Qdisc *child = q->queues[i];

            q->queues[i] = &noop_qdisc;
            qdisc_purge_queue(child);
            removed[n_removed++] = child;                               <--- [4]
        }
    }

    sch_tree_unlock(sch);

    for (i = 0; i < n_removed; i++)
        qdisc_put(removed[i]);                                          <--- [5]
    kfree(removed);
...

}
```

Hardware support?

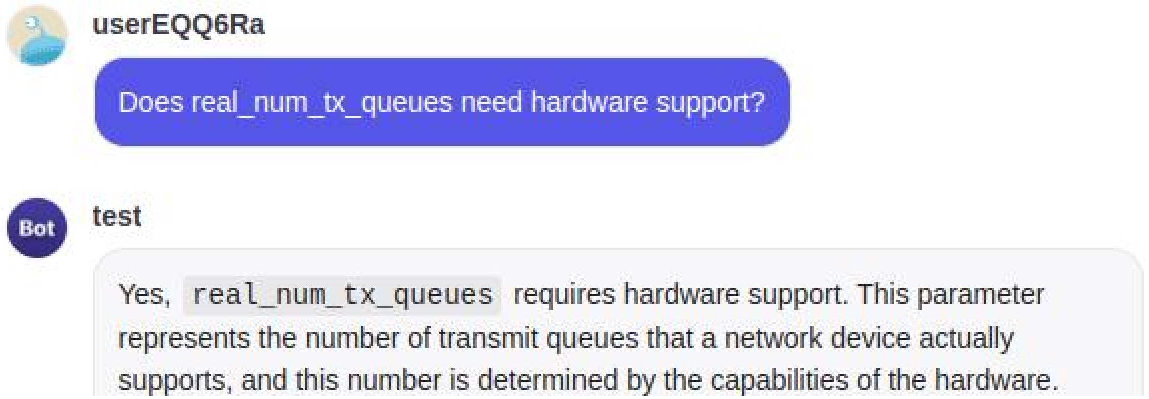
Source



```
@num_tx_queues:       Number of TX queues allocated at alloc_netdev_mq() time
@real_num_tx_queues:     Number of TX queues currently active in device
```

AI

My thinkpad

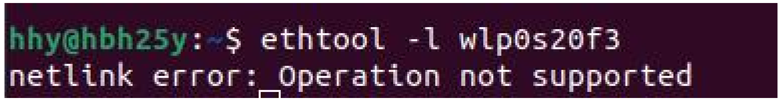


userEQQ6Ra

Does real_num_tx_queues need hardware support?

test

Yes, `real_num_tx_queues` requires hardware support. This parameter represents the number of transmit queues that a network device actually supports, and this number is determined by the capabilities of the hardware.



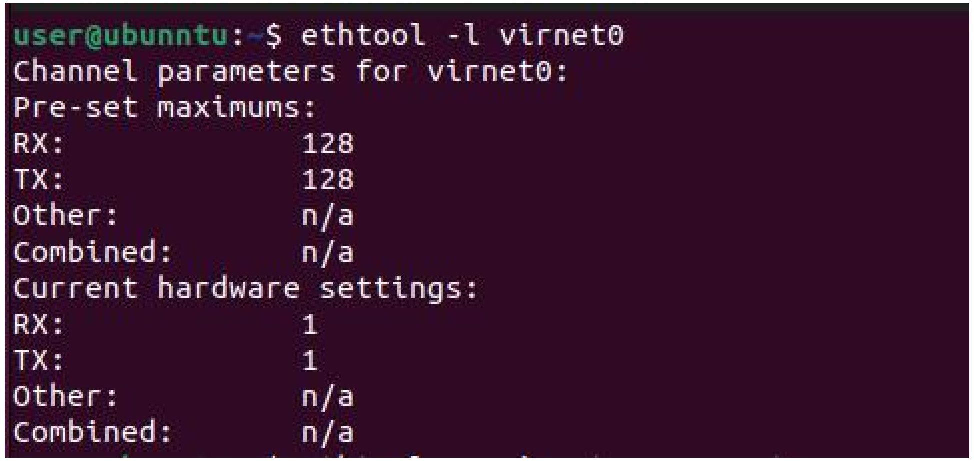
```
hhy@hbh25y:~$ ethtool -l wlp0s20f3
netlink error: Operation not supported
```

CVE-2024-36978

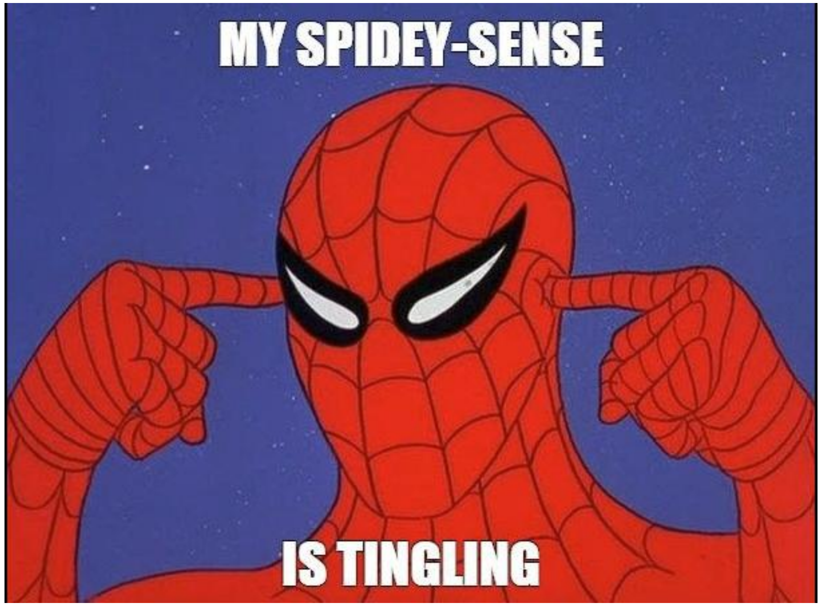
Useless bug?

virtual network card

- create a new namespace
- ip link add dev virnet0 type veth
- ip addr add 192.168.99.2/24 dev virnet0
- ip link set virnet0 up



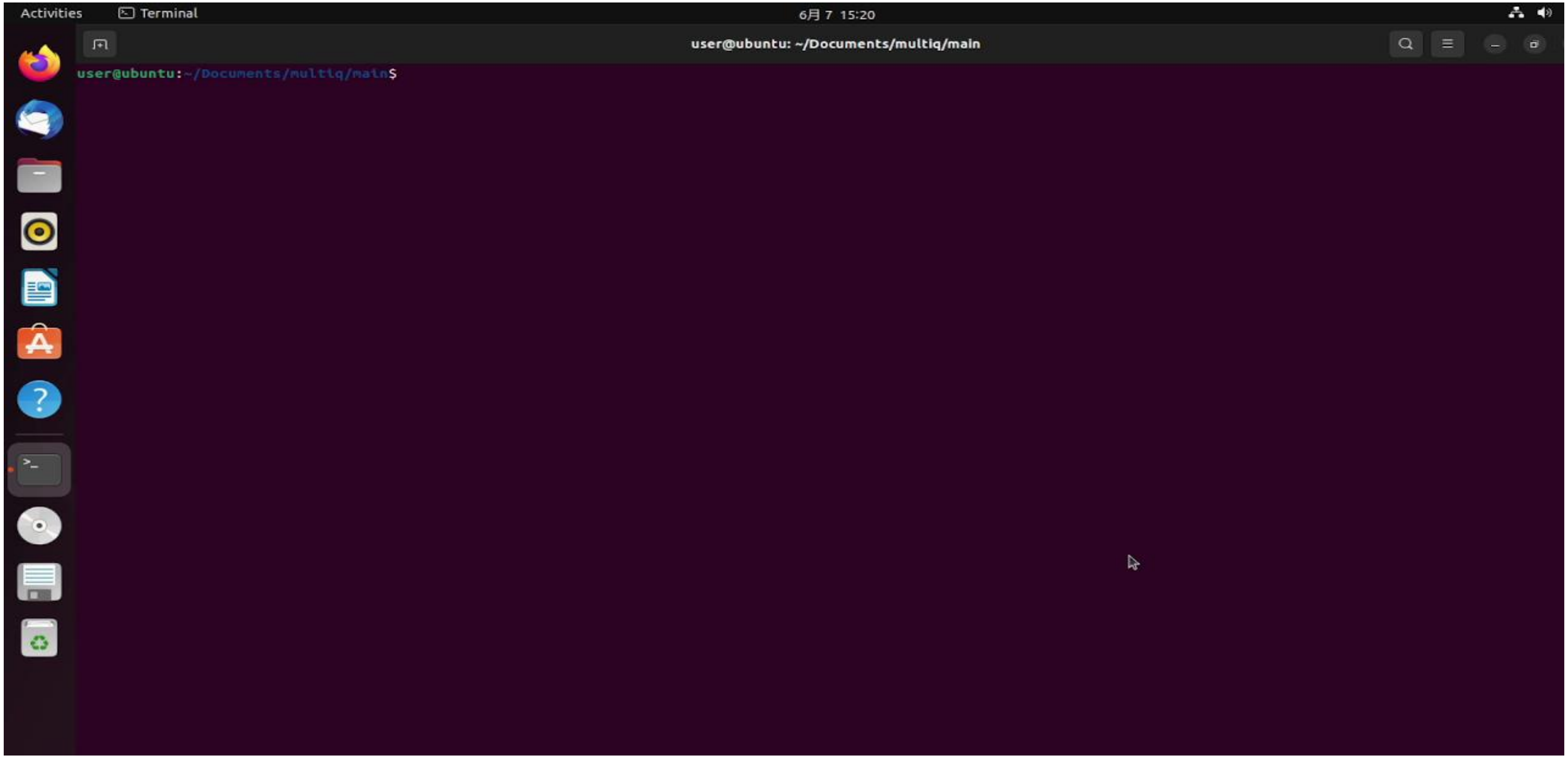
```
user@ubunntu:~$ ethtool -l virnet0
Channel parameters for virnet0:
Pre-set maximums:
RX:             128
TX:             128
Other:          n/a
Combined:       n/a
Current hardware settings:
RX:             1
TX:             1
Other:          n/a
Combined:       n/a
```



CVE-2024-36978

$$\begin{tikzcd}
\text{virnet0 rx create} \arrow[r] \arrow[d] & \text{num\_tx\_queues = 128} \\
\text{ethtool -L virnet0 rx 124 tx 124} \arrow[r] \arrow[d] & \text{real\_num\_tx\_queues = 124} \\
\text{call multiq\_tune} \arrow[r] \arrow[d] & \text{qopt->bands = 124, q->bands = 128} \\
\text{ethtool -L virnet0 rx 119 tx 119} \arrow[r] \arrow[d] & \text{real\_num\_tx\_queues = 119} \\
\text{call multiq\_tune} \arrow[r] \arrow[d] & \text{qopt->bands = 119, q->bands = 124} \\
\text{OOB write} &
\end{tikzcd}$$

EXP Demo



```
user@ubuntu:~/Documents/multiq/main$
```

END

@Dawuge3
@HBh25Y