Exercise L01-01: Qemu

```
/* CVE-2017-15118
     ref. https://bugzilla.redhat.com/attachment.cgi?id=1358264&action=diff

   a qemu client can send a request to the Network Block Device (NBD) qemu server:
     $ qemu-io f raw nbd://localhost:10809/path
*/

#define NBD_MAX_NAME_SIZE 256

static int nbd_negotiate_handle_info(NBDClient *client, uint32_t length,
                                     uint32_t opt, uint16_t myflags,
                                     Error **errp) {
    char name[NBD_MAX_NAME_SIZE + 1];
    uint16_t requests;
    uint32_t namelen;
    const char *msg;

    /* Client sends:
        4 bytes: L, name length (can be 0)
        L bytes: export name
        2 bytes: N, number of requests (can be 0)
        N * 2 bytes: N requests
    */
    if (length < sizeof(namelen) + sizeof(requests)) {
        msg = "overall request too short";
        goto invalid;
    }
    if (nbd_read(client->ioc, &namelen, sizeof(namelen), errp) < 0) {
        return -EIO;
    }

    be32_to_cpus(&namelen);
    length -= sizeof(namelen);
    if (namelen > length - sizeof(requests) || (length - namelen) % 2) {
        msg = "name length is incorrect";
        goto invalid;
    }
    if (nbd_read(client->ioc, name, namelen, errp) < 0) {
        return -EIO;
    }
    name[namelen] = '\0';
    ...
}

/* nbd_read
 * Reads @size bytes from @ioc. Returns 0 on success. */
static inline int nbd_read(QIOChannel *ioc, void *buffer, size_t size,
                           Error **errp);
```

Exercise L01-02: Ruby

```c
/* CVE-2014-4975 */
/*  ref. https://svn.ruby-lang.org/cgi-bin/viewvc.cgi/trunk/pack.c?r1=45921&r2=46778 */

/* @pack.c
    ["a"*3070].pack("m4000")
      => encode(var, "aaa..", 3070, .., true) */
static void
encodes(VALUE str, const char *s, long len, int type, int tail_lf) {
    char buff[4096];
    long i = 0;
    const char *trans = type == 'u' ? uu_table : b64_table;
    char padding;

    if (type == 'u') {
        buff[i++] = (char)len + ' ';
        padding = '`';
    }
    else {
        padding = '=';
    }
    while (len >= 3) {
        while (len >= 3 && sizeof(buff)-i >= 4) {
            buff[i++] = trans[077 & (*s >> 2)];
            buff[i++] = trans[077 & (((*s << 4) & 060) | ((s[1] >> 4) & 017))];
            buff[i++] = trans[077 & (((s[1] << 2) & 074) | ((s[2] >> 6) & 03))];
            buff[i++] = trans[077 & s[2]];
            s += 3;
            len -= 3;
        }
        if (sizeof(buff)-i < 4) {
            rb_str_buf_cat(str, buff, i);
            i = 0;
        }
    }

    if (len == 2) {
        buff[i++] = trans[077 & (*s >> 2)];
        buff[i++] = trans[077 & (((*s << 4) & 060) | ((s[1] >> 4) & 017))];
        buff[i++] = trans[077 & (((s[1] << 2) & 074) | (('\0' >> 6) & 03))];
        buff[i++] = padding;
    }
    else if (len == 1) {
        buff[i++] = trans[077 & (*s >> 2)];
        buff[i++] = trans[077 & (((*s << 4) & 060) | (('\0' >> 4) & 017))];
        buff[i++] = padding;
        buff[i++] = padding;
    }
    if (tail_lf) buff[i++] = '\n';
    rb_str_buf_cat(str, buff, i);
}
```

Exercise L01-03: Libc

```
/* CVE-2015-7547
     ref. https://sourceware.org/ml/libc-alpha/2016-02/msg00416.html (>1000 lines!)

@glibc-2.22
  getaddrinfo(): given a url, returns a set of addrinfo
    gaih_inet()
      gethostbyname4_r()
       : sends out parallel A (ipv4) and AAAA (ipv6) queries if PF_UNSPEC
*/
enum nss_status _nss_dns_gethostbyname4_r(...) {
   ...
   ansp = (querybuf *) alloca (2048);
   __libc_res_nsearch (&_res, name, C_IN, T_UNSPEC,
                       &ansp, 2048, &ansp,
                       &ansp2, &anssizp2, &resplen2, &ans2p_malloced);
   ...
}

/*
gethostbyname4_r()              <- alloca-ed
  __libc_res_nsearch()
    __libc_res_nquerydomain()
      __libc_res_nquery()
        __libc_res_nsend()
          send_dg()             <- overflow
*/
int __libc_res_nsend(...) {
next_ns:
  /* - buf/buflen: A query
     - buf2/buflen2: AAAA query
     - ansp: 'alloca-ed' buffer (host_buffer.buf->buf)
     - anssizp: 2048
     - anscp: ansp (fine to realloc if necessary)
     - ansp2: NULL (fine to malloc if necessary) */
  n = send_dg(statp, buf, buflen, buf2, buflen2,
              ansp, anssizp, &terrno,
              ns, &v_circuit, &gotsomewhere, ansp,
              ansp2, nansp2, resplen2, ansp2_malloced);
  /*
  When send_dg() returns:
    ansp  -> answer to A (or AAAA) query
    ansp2 -> answer to AAAA (or A) query

  1) both are in stack (alloca)
  2) ansp in stack but ansp2 is in heap (no more space left after the first answer)
     (ans2p_malloced = 1)
  3) both are in heap (both answers were too big)
     (ansp != anscp)
  */
  ...
  /* try another name server */
  if (n == 0 && (buf2 == NULL || *resplen2 == 0))
    goto next_ns;
}

/* (snippet of document in glibc-2.23)

   The send_dg function is responsible for sending a DNS query over UDP
   to the nameserver.
```

```
60
61      The query stored in BUF of BUFLEN length is sent first followed by
62      the query stored in BUF2 of BUFLEN2 length.  Queries are sent
63      in parallel (default) or serially (RES_SINGLKUP or RES_SNGLKUPREOP).
64
65      Answers to the query are stored firstly in *ANSP up to a max of
66      *ANSSIZP bytes.  If more than *ANSSIZP bytes are needed and ANSCP
67      is non-NULL (to indicate that modifying the answer buffer is allowed)
68      then malloc is used to allocate a new response buffer and ANSCP and
69      ANSP will both point to the new buffer.
70
71      Answers to the query are stored secondly in *ANSP2 up to a max of
72      *ANSSIZP2 bytes, with the actual response length stored in
73      *RESPLEN2.  If more than *ANSSIZP bytes are needed and ANSP2
74      is non-NULL (required for a second query) then malloc is used to
75      allocate a new response buffer, *ANSSIZP2 is set to the new buffer
76      size and *ANSP2_MALLOCED is set to 1.
77
78      Note that the answers may arrive in any order from the server and
79      therefore the first and second answer buffers may not correspond to
80      the first and second queries.
81
82      It is the caller's responsibility to free the malloc allocated
83      buffers by detecting that the pointers have changed from their
84      original values i.e. *ANSCP or *ANSP2 has changed.
85  */
86  static int
87  send_dg(res_state statp,
88      const u_char *buf, int buflen, const u_char *buf2, int buflen2,
89      u_char **ansp, int *anssizp,
90      int *terrno, int ns, int *v_circuit, int *gotsomewhere, u_char **anscp,
91      u_char **ansp2, int *anssizp2, int *resplen2, int *ansp2_malloced)
92  {
93      u_char *ans = *ansp;
94      int orig_anssizp = *anssizp;
95
96    /* both resps haven't arrived yet */
97    int recvresp1 = 0;
98    int recvresp2 = 0;
99
100   wait:
101    __poll (pfd, 1, 0);
102    ...
103    if (pfd[0].revents & POLLOUT) {...}
104      else if (pfd[0].revents & POLLIN) {
105      /* responses are arriving (on the wire). */
106          int *thisanssizp;
107          u_char **thisansp;
108          int *thisresplenp;
109
110          if ((recvresp1 | recvresp2) == 0) {
111        /* We have not received any responses yet */
112              thisanssizp = anssizp;
113              thisansp = anscp ?: ansp;
114              thisresplenp = &resplen;
115          } else {
116              if (*anssizp != MAXPACKET) {
117                  /* No buffer allocated for the first reply. We can
118              try to use the rest of the user-provided buffer.  */
119                  *anssizp2 = orig_anssizp - resplen;
120                  *ansp2 = *ansp + resplen;
121              } else {
```

```
122                 /* The first reply did not fit into the user-provided buffer.
123                    Maybe the second answer will.  */
124                 *anssizp2 = orig_anssizp;
125                 *ansp2 = *ansp;
126             }
127
128             thisanssizp = anssizp2;
129             thisansp = ansp2;
130             thisresplenp = resplen2;
131         }
132
133         if (*thisanssizp < MAXPACKET
134             /* Yes, we test ANSCP here. If we have two buffers
135                both will be allocatable. */
136             && anscp
137             && (ioctl (pfd[0].fd, FIONREAD, thisresplenp) < 0
138             || *thisanssizp < *thisresplenp)) {
139             u_char *newp = malloc(MAXPACKET);
140             if (newp != NULL) {
141                 *anssizp = MAXPACKET;
142                 *thisansp = ans = newp;
143
144         /* BUG:
145             - failed to set *ansp to the new buffer
146             - failed to set *thisanssizp to the new size
147
148             *ansp -> allocaed (2048)
149             *anssizp -> MAXPACKET */
150
151                 if (thisansp == ansp2)
152                   *ansp2_malloced = 1;
153             }
154         }
155         *thisresplenp = recvfrom(pfd[0].fd, (char*)*thisansp,
156                                  *thisanssizp, 0, &from, &fromlen);
157         if (*thisresplenp <= 0)
158       goto err_out;
159
160         /* Mark which reply we received.  */
161         if (recvresp1 == 0 && hp->id == anhp->id)
162             recvresp1 = 1;
163         else
164             recvresp2 = 1;
165
166         /* Repeat waiting if we have a second answer to arrive.  */
167         if ((recvresp1 & recvresp2) == 0) {
168             goto wait;
169         }
170   ...
171     }
172
173  err_out:
174   return 0;
175 }
```

Exercise L02-01: Linux block

```c
/* CVE-2013-2851 */
/*  ref. https://github.com/torvalds/linux, @ffc8b30866879ed9ba62bd0a86fecdbd51cd3d19 */

/* block/genhd.c */
static void register_disk(struct gendisk *disk) {
  struct device *ddev = disk_to_dev(disk);

  ddev->parent = disk->driverfs_dev;
  dev_set_name(ddev, disk->disk_name);

  /* delay uevents, until we scanned partition table */
  dev_set_uevent_suppress(ddev, 1);

  if (device_add(ddev))
    return;
  ...
}

/* drivers/block/ndb.c */
static int __nbd_ioctl(struct block_device *bdev, struct nbd_device *nbd,
                       unsigned int cmd, unsigned long arg) {
   switch (cmd) {
   ...
   case NBD_DO_IT: {
     struct task_struct *thread;
     ...
     thread = kthread_create(nbd_thread, nbd, nbd->disk->disk_name);
     if (IS_ERR(thread)) {
       return PTR_ERR(thread);
     }
   }
   ...
  }
}

/**
 * dev_set_name - set a device name
 * @dev: device
 * @fmt: format string for the device's name
 */
int dev_set_name(struct device *dev, const char *fmt, ...) {
    va_list vargs;
    int err;

    va_start(vargs, fmt);
    err = kobject_set_name_vargs(&dev->kobj, fmt, vargs);
    va_end(vargs);
    return err;
}

/**
 * kthread_create - create a kthread on the current node
 * @threadfn: the function to run in the thread
 * @namefmt: printf-style format string for the thread name
 * @arg...: arguments for @namefmt.
 */
#define kthread_create(threadfn, data, namefmt, arg...) ...
```

Exercise L02-02: Linux ext3

```c
/* CVE-2013-1848 */
/*  ref. https://github.com/torvalds/linux, @8d0c2d10dd72c5292eda7a06231056a4c972e4cc */

/* fs/ext3/super.c */
void ext3_msg(struct super_block *sb, const char *prefix,
        const char *fmt, ...)
{
    struct va_format vaf;
    va_list args;

    va_start(args, fmt);

    vaf.fmt = fmt;
    vaf.va = &args;

    printk("%sEXT3-fs (%s): %pV\n", prefix, sb->s_id, &vaf);

    va_end(args);
}

/*
 * Get the superblock
 */
static ext3_fsblk_t get_sb_block(void **data, struct super_block *sb)
{
    char *options = (char *) *data;

    if (!options || strncmp(options, "sb=", 3) != 0)
        return 1;   /* Default location */
    options += 3;

    if (*options && *options != ',') {
        ext3_msg(sb, "error: invalid sb specification: %s", (char *) *data);
        return 1;
    }
  ...
}

/*
 * Open the external journal device
 */
static struct block_device *ext3_blkdev_get(dev_t dev, struct super_block *sb)
{
    struct block_device *bdev;
    char b[BDEVNAME_SIZE];

    bdev = blkdev_get_by_dev(dev, FMODE_READ|FMODE_WRITE|FMODE_EXCL, sb);
    if (IS_ERR(bdev))
        goto fail;
    return bdev;

fail:
    ext3_msg(sb, "error: failed to open journal device %s: %ld",
        __bdevname(dev, b), PTR_ERR(bdev));

    return NULL;
}
```

Exercise L02-03: Sudo

```c
/* CVE-2012-0809 */
/*  ref: https://github.com/millert/sudo, @697caf8df32270a2676cd54e69d1f72d8d172d1f */

/* src/sudo.c */
int main(int argc, char *argv[], char *envp[]) {
  ...
  /* Parse command line arguments. */
  sudo_mode = parse_args(argc, argv, &nargc, &nargv, &settings, &env_add);
  sudo_debug(9, "sudo_mode %d", sudo_mode);
  ...
}

/*
 * Simple debugging/logging.
 */
void sudo_debug(int level, const char *fmt, ...) {
  va_list ap;
  char *fmt2;

  if (level > debug_level)
    return;

  /* Backet fmt with program name and a newline to make it a single write */
  easprintf(&fmt2, "%s: %s\n", getprogname(), fmt);
  va_start(ap, fmt);
  vfprintf(stderr, fmt2, ap);
  va_end(ap);
  efree(fmt2);
}

/* NOTE. */
/*  easprintf: an error-free version of asprintf() */
/*  efree: an error-free version of free() */
```

Exercise L03-00: Integer Overflow and Undefined Behaviors

1. (in x86_64) what does the expression `1 > 0` evaluate to?
   (a) 0 (b) 1 (c) NaN (d) -1 (e) undefined

2. `(unsigned short)1 > -1`?
   (a) 1 (b) 0 (c) -1 (d) undefined

3. `-1U > 0`?
   (a) 1 (b) 0 (c) -1 (d) undefined

4. `UINT_MAX + 1`?
   (a) 0 (b) 1 (c) INT_MAX (d) UINT_MAX (e) undefined

5. `abs(-2147483648)`?
   (a) == 0 (b) < 0 (c) > 0 (d) == NaN

6. `1U << 0`?
   (a) 1 (b) 4 (c) UINT_MAX (d) 0 (e) undefined

7. `1U << 32`?
   (a) 1 (b) 4 (c) UINT_MAX (d) INT_MIN (e) 0 (f) undefined

8. `-1L << 2`?
   (a) 0 (b) 4 (c) INT_MAX (d) INT_MIN (e) undefined

9. `INT_MAX + 1`?
   (a) 0 (b) 1 (c) INT_MAX (d) UINT_MAX (e) undefined

10. `UINT_MAX + 1`?
    (a) 0 (b) 1 (c) INT_MAX (d) UINT_MAX (e) undefined

11. `-INT_MIN`?
    (a) 0 (b) 1 (c) INT_MAX (d) UINT_MAX (e) INT_MIN (f) undefined

12. `-1L > 1U`? on x86_64 and x86
    (a) (0, 0) (b) (1, 1) (c) (0, 1) (d) (1, 0) (e) undefined

BONUS. is it possible that `a / b < 0` when `a < 0` and `b < 0`?

Exercise L03-01: Android

```
/* CVE-2015-1538 and CVE-2015-3824
   ref. https://android.googlesource.com/platform/frameworks/av/+/edd4a76%5E!/ */

/* CVE-2015-1538: parsing mp4 file from MMS */
status_t SampleTable::setTimeToSampleParams(off64_t data_offset, size_t data_size) {
  if (mTimeToSample != NULL || data_size < 8)
    return ERROR_MALFORMED;

  uint8_t header[8];
  if (mDataSource->readAt(data_offset, header, sizeof(header)) < (ssize_t)sizeof(header))
    return ERROR_IO;
  ...
  mTimeToSampleCount = U32_AT(&header[4]);
  mTimeToSample = new uint32_t[mTimeToSampleCount * 2];
  size_t size = sizeof(uint32_t) * mTimeToSampleCount * 2;
  if (mDataSource->readAt(data_offset + 8, mTimeToSample, size) < (ssize_t)size)
    return ERROR_IO;
  for (uint32_t i = 0; i < mTimeToSampleCount * 2; ++i)
    mTimeToSample[i] = ntohl(mTimeToSample[i]);
  return OK;
}

/* CVE-2015-3824: parsing mp4 file from MMS */
status_t MPEG4Extractor::parseChunk(off64_t *offset, int depth) {
  uint32_t hdr[2];
  mDataSource->readAt(*offset, hdr, 8);
  uint64_t chunk_size = ntohl(hdr[0]);
  uint32_t chunk_type = ntohl(hdr[1]);
  ...
    switch(chunk_type) {
     ...
    case FOURCC('t', 'x', '3', 'g'): {
      uint32_t type;
      const void *data;
      size_t size = 0;
      if (!mLastTrack->meta->findData(kKeyTextFormatData, &type, &data, &size))
        size = 0;

      uint8_t *buffer = new (std::nothrow) uint8_t[size + chunk_size];
      if (buffer == NULL)
        return ERROR_MALFORMED;
      if (size > 0)
        memcpy(buffer, data, size);

      if ((size_t)(mDataSource->readAt(*offset, buffer + size, chunk_size)) < chunk_size) {
        delete[] buffer;
        buffer = NULL;
        // advance read pointer so we don't end up reading this again
        *offset += chunk_size;
        return ERROR_IO;
      }
      mLastTrack->meta->setData(kKeyTextFormatData, 0, buffer, size + chunk_size);
      delete[] buffer;
      *offset += chunk_size;
      break;
    }
    ...
  }
}
```

Exercise L03-02: Linux perf

```c
/* CVE-2009-3234 and double fetching found in 2017
   ref. Linux, @b3e62e35058fc744ac794611f4e79bcd1c5a4b83, @f12f42acdbb577a12eecfcebbbec41c81505c4dc */

SYSCALL_DEFINE5(perf_event_open, struct perf_event_attr __user *, attr_uptr, ...) {
  err = perf_copy_attr(attr_uptr, &attr); ...
}

static int perf_copy_attr(struct perf_counter_attr __user *uattr,
                          struct perf_counter_attr *attr) {
  int ret;
  u32 size;

  if (!access_ok(VERIFY_WRITE, uattr, PERF_ATTR_SIZE_VER0))
    return -EFAULT;

  /* zero the full structure, so that a short copy will be nice. */
  memset(attr, 0, sizeof(*attr));
  ret = get_user(size, &uattr->size);
  if (ret)
    return ret;

  if (size > PAGE_SIZE) /* silly large */
    goto err_size;
  if (!size)            /* abi compat */
    size = PERF_ATTR_SIZE_VER0;
  if (size < PERF_ATTR_SIZE_VER0)
    goto err_size;

  /* If we're handed a bigger struct than we know of,
   * ensure all the unknown bits are 0.  */
  if (size > sizeof(*attr)) {
    unsigned long val;
    unsigned long __user *addr, __user *end;

    addr = PTR_ALIGN((void __user *)uattr + sizeof(*attr),
                     sizeof(unsigned long));
    end  = PTR_ALIGN((void __user *)uattr + size,
                     sizeof(unsigned long));

    for (; addr < end; addr += sizeof(unsigned long)) {
      ret = get_user(val, addr);
      if (ret)
        return ret;
      if (val)
        goto err_size;
    }
  }

  ret = copy_from_user(attr, uattr, size);
  if (ret)
    return -EFAULT;
  ...
out:
  return ret;
err_size:
  put_user(sizeof(*attr), &uattr->size);
  ret = -E2BIG;
  goto out;
}
```

Exercise L03-03: Linux usb

```c
/* CVE-2016-4482
   ref. Linux, @681fef8380eb818c0b845fca5d2ab1dcbab114ee */

// @include/uapi/linux/usbdevice_fs.h
struct usbdevfs_connectinfo {
  unsigned int devnum;
  unsigned char slow;
};

static long usbdev_do_ioctl(struct file *file, unsigned int cmd, void __user *p) {
  struct usb_dev_state *ps = file->private_data;
  struct usb_device *dev = ps->dev;
  int ret = -ENOTTY;

  if (!(file->f_mode & FMODE_WRITE))
    return -EPERM;

  usb_lock_device(dev);
  ...
  switch (cmd) {
  ...
  case USBDEVFS_CONNECTINFO:
    snoop(&dev->dev, "%s: CONNECTINFO\n", __func__);
    ret = proc_connectinfo(ps, p);
    break;
  ...
  }
  return ret;
}

static int proc_connectinfo(struct usb_dev_state *ps, void __user *arg) {
  struct usbdevfs_connectinfo ci = {
    .devnum = ps->dev->devnum,
    .slow = ps->dev->speed == USB_SPEED_LOW
  };

  if (copy_to_user(arg, &ci, sizeof(ci)))
    return -EFAULT;
  return 0;
}
```

Exercise L04-01: Heartbleed

```
/* CVE-2014-0160
   ref. https://git.openssl.org/gitweb/?p=openssl.git;a=commit;h=96db9023b881d7cd9f379b0c154650d6c108e9a3 */
int tls1_process_heartbeat(SSL *s) {
  unsigned char *p = &s->s3->rrec.data[0], *pl;
  unsigned short hbtype;
  unsigned int payload;
  unsigned int padding = 16; /* Use minimum padding */

  /* Read type and payload length first */
  hbtype = *p++;
  n2s(p, payload);
  pl = p;

  if (s->msg_callback)
    s->msg_callback(0, s->version, TLS1_RT_HEARTBEAT,
                    &s->s3->rrec.data[0], s->s3->rrec.length, s,
                    s->msg_callback_arg);

  if (hbtype == TLS1_HB_REQUEST) {
    unsigned char *buffer, *bp;
    int r;

    /* Allocate memory for the response, size is 1 bytes
     * message type, plus 2 bytes payload length, plus
     * payload, plus padding */
    buffer = OPENSSL_malloc(1 + 2 + payload + padding);
    bp = buffer;

    /* Enter response type, length and copy payload */
    *bp++ = TLS1_HB_RESPONSE;
    s2n(payload, bp);
    memcpy(bp, pl, payload);
    bp += payload;
    /* Random padding */
    RAND_pseudo_bytes(bp, padding);

    r = ssl3_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + payload + padding);
    if (r >= 0 && s->msg_callback)
      s->msg_callback(1, s->version, TLS1_RT_HEARTBEAT,
                      buffer, 3 + payload + padding,
                      s, s->msg_callback_arg);

    OPENSSL_free(buffer);
    ...
  } else if (hbtype == TLS1_HB_RESPONSE) {
    unsigned int seq;

    /* We only send sequence numbers (2 bytes unsigned int),
     * and 16 random bytes, so we just try to read the
     * sequence number */
    n2s(pl, seq);

    if (payload == 18 && seq == s->tlsext_hb_seq) {
      s->tlsext_hb_seq++;
      s->tlsext_hb_pending = 0;
    }
  }
  return 0;
}
```

Exercise L04-02: Wireshark

```
/* CVE-2018-11360
   ref. https://code.wireshark.org, @47a5fa850b388fcf4ea762073806f01b459820fe */

static guint16
de_sub_addr(tvbuff_t *tvb, proto_tree *tree, packet_info *pinfo, guint32 offset,
            guint len, gchar **extracted_address){
  ...
  ia5_string = (guint8 *)tvb_memdup(wmem_packet_scope(), tvb,
                                    curr_offset, ia5_string_len);
  *extracted_address = (gchar *)wmem_alloc(wmem_packet_scope(), ia5_string_len);

  invalid_ia5_char = FALSE;
  for(i = 0; i < ia5_string_len; i++) {
    dig1 = (ia5_string[i] & 0xf0) >> 4;
    dig2 = ia5_string[i] & 0x0f;
    oct = (dig1 * 10) + dig2 + 32;
    if (oct > 127)
      invalid_ia5_char = TRUE;
    ia5_string[i] = oct;
  }

  IA5_7BIT_decode(*extracted_address, ia5_string, ia5_string_len);
  ...
}

void
IA5_7BIT_decode(unsigned char *dest, const unsigned char *src, int len) {
  int i, j;
  gunichar buf;

  for (i = 0, j = 0; j < len;  j++) {
    buf = char_def_ia5_alphabet_decode(src[j]);
    i += g_unichar_to_utf8(buf,&(dest[i]));
  }
  dest[i]=0;
  return;
}
```

Exercise L04-03: Linux keyring

```c
/* CVE-2016-0728
   ref. Linux, @23567fd052a9abb6d67fe8e7a9ccdd9800a540f2 */

/* Join the named keyring as the session keyring if possible else
 * attempt to create a new one of that name and join that. */
long join_session_keyring(const char *name) {
  struct cred *new = prepare_creds();
  ...

  /* allow the user to join or create a named keyring */
  mutex_lock(&key_session_mutex);

  /* look for an existing keyring of this name */
  keyring = find_keyring_by_name(name, false);
  if (PTR_ERR(keyring) == -ENOKEY) {
    /* not found - try and create a new one */
    keyring = keyring_alloc(
      name, old->uid, old->gid, old,
      KEY_POS_ALL | KEY_USR_VIEW | KEY_USR_READ | KEY_USR_LINK,
      KEY_ALLOC_IN_QUOTA, NULL);
    if (IS_ERR(keyring)) {
      ret = PTR_ERR(keyring);
      goto error2;
    }
  } else if (IS_ERR(keyring)) {
    ret = PTR_ERR(keyring);
    goto error2;
  } else if (keyring == new->session_keyring) {
    ret = 0;
    goto error2;
  }

  /* we've got a keyring - now to install it */
  ret = install_session_keyring_to_cred(new, keyring);
  if (ret < 0)
    goto error2;

  commit_creds(new);
  mutex_unlock(&key_session_mutex);

  ret = keyring->serial;
  key_put(keyring);
okay:
  return ret;

error2:
  mutex_unlock(&key_session_mutex);
error:
  abort_creds(new);
  return ret;
}

/* Find a keyring with the specified name.
 * ...
 * Returns a pointer to the keyring with the keyring's refcount having being
 * incremented on success.  -ENOKEY is returned if a key could not be found. */
struct key *find_keyring_by_name(const char *name, bool skip_perm_check) { ... }
```

Exercise L04-04: Linux vma

```c
/* CVE-2018-17182
   ref. https://googleprojectzero.blogspot.com/2018/09/a-cache-invalidation-bug-in-linux.html */

/* mm by a process, vmacache per thread
   current->vmcache.seqnum indicates the current version of vmcache
   mm->vmcache_seqnum indicates the global version

   current->vmcache.seqnum != mm->vmcache_seqnum indicates that the vmcache
   contains dangled (i.e., free()) pointers. Is there any path that a
   dangled pointer might be considered valid (i.e., wrapped?)? */

/* find vma of addr in mm */
struct vm_area_struct *vmacache_find(struct mm_struct *mm, unsigned long addr) {
  int idx = VMACACHE_HASH(addr);
  if (!vmacache_valid(mm))
    return NULL;
  for (int i = 0; i < VMACACHE_SIZE; i++) {
    struct vm_area_struct *vma = current->vmacache.vmas[idx];
    if (vma)
      if (vma->vm_start <= addr && vma->vm_end > addr)
        return vma;
    if (++idx == VMACACHE_SIZE)
      idx = 0;
  }
  return NULL;
}

/* Flush vma caches for threads that share a given mm. */
void vmacache_flush_all(struct mm_struct *mm) {
  struct task_struct *g, *p;
  /* Single threaded tasks need not iterate the entire list of
   * process. We can avoid the flushing as well since the mm's seqnum
   * was increased and don't have to worry about other threads'
   * seqnum. Current's flush will occur upon the next lookup. */
  if (atomic_read(&mm->mm_users) == 1)
    return;
  rcu_read_lock();
  for_each_process_thread(g, p) {
    /* Only flush the vmacache pointers as the mm seqnum is already
     * set and curr's will be set upon invalidation when the next
     * lookup is done. */
    if (mm == p->mm)
      vmacache_flush(p);
  }
  rcu_read_unlock();
}

static bool vmacache_valid(struct mm_struct *mm) {
  if (!vmacache_valid_mm(mm))
    return false;
  if (mm->vmacache_seqnum != current->vmacache.seqnum) {
    /* First attempt will always be invalid, initialize the new cache
     * for this task here. */
    current->vmacache.seqnum = mm->vmacache_seqnum;
    vmacache_flush(current);
    return false;
  }
  return true;
}
```