June 17, 2020

GHSL-2020-075, GHSL-2020-079, GHSL-2020-080, GHSL-2020-081, GHSL-2020-082, GHSL-2020-083, GHSL-2020-084: Multiple vulnerabilities in SANE Backends (DoS, RCE)

Kevin Backhouse

Summary

SANE Backends contains several memory corruption vulnerabilities which can be triggered by a malicious device or computer that is connected to the same network. The vulnerabilities are triggered when an application such as simple-scan searches the network for scanners. In the specific case of simple-scan, this happens immediately when simple-scan starts, so there isn't even any need to trick the user into thinking that the scanner is genuine so that they will click on it.

We have also identified some other vulnerabilities in SANE Backends, which are less severe because they *do* require the user to click the "Scan" button after connecting to a malicious device.

Product

SANE Backends

Tested Version

libsane1 1.0.27-1~experimental3ubuntu2.2, tested on Ubuntu 18.04.4 LTS with simple-scan 3.28.0-0ubuntu1.

Details

Issue 1 (`GHSL-2020-075`, `CVE-2020-12867`): null pointer dereference in `sanei_epson_net_read`

The function `sanei_epson_net_read` has buggy code for handling the situation where it receives a response with an unexpected size:

```
/* receive net header */
size = sanei_epson_net_read_raw(s, header, 12, status);
if (size != 12) {
  return 0;
}

if (header[0] != 'I' || header[1] != 'S') {
  DBG(1, "header mismatch: %02X %02x\n", header[0], header[1]);
  *status = SANE_STATUS_IO_ERROR;
  return 0;
}

size = be32atoh(&header[6]);  <===== size is controlled by attacker

DBG(23, "%s: wanted = %lu, available = %lu\n", __func__,
  (u_long) wanted, (u_long) size);

*status = SANE_STATUS_GOOD;

if (size == wanted) {

  DBG(15, "%s: full read\n", __func__);

  read = sanei_epson_net_read_raw(s, buf, size, status);

  if (s->netbuf) {
    free(s->netbuf);
    s->netbuf = NULL;
    s->netlen = 0;
  }

  if (read < 0) {
    return 0;
  }

/*  } else if (wanted < size && s->netlen == size) { */
} else {
  DBG(23, "%s: partial read\n", __func__);

  read = sanei_epson_net_read_raw(s, s->netbuf, size, status);  <===== s->netbuf could be NULL
  if (read != size) {
    return 0;
  }

  s->netlen = size - wanted;
  s->netptr += wanted;
  read = wanted;

  DBG(23, "0,4 %02x %02x\n", s->netbuf[0], s->netbuf[4]);  <===== s->netbuf could be NULL
  DBG(23, "storing %lu to buffer at %p, next read at %p, %lu bytes left\n",
    (u_long) size, s->netbuf, s->netptr, (u_long) s->netlen);

  memcpy(buf, s->netbuf, wanted);
}

return read;
```

Notice that the value of `size` is read from an incoming message, so an attacker can set it to any value they like. In the `else` branch, which handles the case where `size != wanted`, there is no check that `s->netbuf` is large enough for `size` bytes. This could potentially lead to a buffer overflow. However, our proof-of-concept exploit for this bug triggers a case where `s->netbuf` hasn't even been initialized so the program crashes due to a null pointer dereference, rather than a buffer overflow.

Impact

This issue may lead to remote denial of service, where "remote" means a device or computer connected to the same network as the victim. For example, in a typical office environment the malicious device would need to be somewhere inside the building. Because the vulnerability causes `simple-scan` to crash as soon as it starts, it makes the application unusable.

Issue 2 (`GHSL-2020-079`, `CVE-2020-12866`): null pointer dereference in `epsonds_net_read`

The function `epsonds_net_read` has buggy code for handling the situation where it receives a response with an unexpected size:

```
/* receive net header */
size = epsonds_net_read_raw(s, header, 12, status);
if (size != 12) {
  return 0;
}

if (header[0] != 'I' || header[1] != 'S') {
  DBG(1, "header mismatch: %02X %02x\n", header[0], header[1]);
  *status = SANE_STATUS_IO_ERROR;
```

```
  return 0;
}

// incoming payload size
size = be32atoh(&header[6]);

DBG(23, "%s: wanted = %lu, available = %lu\n", __func__,
  (u_long) wanted, (u_long) size);

*status = SANE_STATUS_GOOD;

if (size == wanted) {

  DBG(15, "%s: full read\n", __func__);

  if (size) {
    read = epsonds_net_read_raw(s, buf, size, status);
  }

  if (s->netbuf) {
    free(s->netbuf);
    s->netbuf = NULL;
    s->netlen = 0;
  }

  if (read < 0) {
    return 0;
  }

} else if (wanted < size) {

  DBG(23, "%s: long tail\n", __func__);

  read = epsonds_net_read_raw(s, s->netbuf, size, status);  <===== no bounds check
  if (read != size) {
    return 0;
  }

  memcpy(buf, s->netbuf, wanted);
  read = wanted;

  free(s->netbuf);
  s->netbuf = NULL;
  s->netlen = 0;

} else {

  DBG(23, "%s: partial read\n", __func__);

  read = epsonds_net_read_raw(s, s->netbuf, size, status);
  if (read != size) {
    return 0;
  }

  s->netlen = size - wanted;  <===== negative integer overflow (because size < wanted)
  s->netptr += wanted;
  read = wanted;

  DBG(23, "0,4 %02x %02x\n", s->netbuf[0], s->netbuf[4]);
  DBG(23, "storing %lu to buffer at %p, next read at %p, %lu bytes left\n",
    (u_long) size, s->netbuf, s->netptr, (u_long) s->netlen);

  memcpy(buf, s->netbuf, wanted);  <===== no bounds check
}

return read;
```

This code is very similar to the code in `epson2_net.c` (see issue 1) and has similar bugs. The first of these is a NULL pointer exception at epsonds-net.c, line 160.

Impact

This issue may lead to remote denial of service, where “remote” means a device or computer connected to the same network as the victim. For example, in a typical office environment the malicious device would need to be somewhere inside the building. Because the vulnerability causes `simple-scan` to crash as soon as it starts, it makes the application unusable.

Issue 3 (GHSL-2020-080, CVE-2020-12861): heap buffer overflow in `epsonds_net_read`

This bug is in the same function as issue 2: `epsonds_net_read`. There is a heap buffer overflow at epsonds-net.c, line 135. The value of `size` is controlled by the attacker, so an arbitrary amount of attacker-controlled data is written to `s->netbuf`.

Impact

This issue may lead to remote code execution, where “remote” means a device or computer connected to the same network as the victim. For example, in a typical office environment the malicious device would need to be somewhere inside the building.

Issue 4 (GHSL-2020-081, CVE-2020-12864): reading uninitialized data in `epsonds_net_read`

This bug is in the same function as issue 2: `epsonds_net_read`. The `memcpy` at epsonds-net.c, line 164 can read uninitialized data. The value of `size` is controlled by the attacker, so the attacker can specify that `size == 0`. Since `s->netbuf` is a newly allocated heap buffer, it contains uninitialized memory.

Impact

By itself, the severity of this issue is very low. However, it may be very useful to an attacker who is attempting to exploit one of the buffer overflow vulnerabilities, such as issue 3, because it may enable the attacker to obtain the ASLR offsets of the program.

Issue 5 (GHSL-2020-082, CVE-2020-12862): out-of-bounds read in `decode_binary`

The function `decode_binary` has an out-of-bounds read:

```
/* h000 */
static char *decode_binary(char *buf)
{
  char tmp[6];
  int hl;

  memcpy(tmp, buf, 4);
  tmp[4] = '\0';

  if (buf[0] != 'h')
    return NULL;

  hl = strtol(tmp + 1, NULL, 16);
  if (hl) {

    char *v = malloc(hl + 1);
    memcpy(v, buf + 4, hl);
    v[hl] = '\0';

    return v;
  }

  return NULL;
}
```

The value of `hl` is controlled by the attacker and can be any value between 0 and 0xFFF (4095). `buf` is a pointer to a 64 byte stack buffer (`rbuf` in `esci2_cmd`), so the memcpy at line 273 can copy up to 4095 bytes from the stack into the newly allocated buffer.

Impact

By itself, the severity of this issue is very low. However, it may be very useful to an attacker who is attempting to exploit one of the buffer overflow vulnerabilities, such as issue 3, because it may enable the attacker to obtain the ASLR offsets of the program.

Issue 6 (GHSL-2020-083, CVE-2020-12863): out-of-bounds read in `esci2_check_header`

`esci2_check_header` uses `sscanf` to read a number from the message:

```
/* INFOx0000100#.... */

/* read the answer len */
if (buf[4] != 'x') {
  DBG(1, "unknown type in header: %c\n", buf[4]);
  return 0;
}

err = sscanf(&buf[5], "%x#", more);
if (err != 1) {
  DBG(1, "cannot decode length from header\n");
  return 0;
}
```

`buf` is a pointer to a 64 byte stack buffer (`rbuf` in `esci2_cmd`), the contents of which are entirely controlled by the attacker. If the attacker fills the buffer with the character '0', then `sscanf` will read off the end of the buffer. If the characters beyond the buffer are valid hexadecimal digits, then they will be converted to a number and written to the variable named `more`. The value of `more` is included in the next message that is sent to the malicious device, so this issue is an information leak vulnerability.

Impact

This issue may lead to remote information disclosure, where "remote" means a device or computer connected to the same network as the victim. In practice, though, this issue is very low severity, because the byte immediately following the buffer is usually not a valid hexadecimal digit, so no information disclosure occurs.

Issue 7 (GHSL-2020-084, CVE-2020-12865): buffer overflow in `esci2_img`

This issue requires the user to click the "Scan" button, so it is a one-click vulnerability, rather than a zero-click vulnerability. The function `esci2_img` has a heap buffer overflow:

```
SANE_Status
esci2_img(struct epsonds_scanner *s, SANE_Int *length)
{
  SANE_Status status = SANE_STATUS_GOOD;
  SANE_Status parse_status;
  unsigned int more;
  ssize_t read;

  *length = 0;

  if (s->canceling)
    return SANE_STATUS_CANCELLED;

  /* request image data */
  eds_send(s, "IMG x0000000", 12, &status, 64);
  if (status != SANE_STATUS_GOOD) {
    return status;
  }

  /* receive DataHeaderBlock */
  memset(s->buf, 0x00, 64);
  eds_recv(s, s->buf, 64, &status);
  if (status != SANE_STATUS_GOOD) {
    return status;
  }

  /* check if we need to read any image data */
  more = 0;
  if (!esci2_check_header("IMG ", (char *)s->buf, &more)) {  <===== attacker controls value of `more`
    return SANE_STATUS_IO_ERROR;
  }

  /* this handles eof and errors */
  parse_status = esci2_parse_block((char *)s->buf + 12, 64 - 12, s, &img_cb);

  /* no more data? return using the status of the esci2_parse_block
   * call, which might hold other error conditions.
   */
  if (!more) {
    return parse_status;
  }

  /* ALWAYS read image data */
  if (s->hw->connection == SANE_EPSONDS_NET) {
    epsonds_net_request_read(s, more);
  }

  read = eds_recv(s, s->buf, more, &status);  <===== heap buffer overflow
  if (status != SANE_STATUS_GOOD) {
    return status;
  }

  if (read != more) {
    return SANE_STATUS_IO_ERROR;
  }

  /* handle esci2_parse_block errors */
  if (parse_status != SANE_STATUS_GOOD) {
    return parse_status;
  }

  DBG(15, "%s: read %lu bytes, status: %d\n", __func__, (unsigned long) read, status);

  *length = read;

  if (s->canceling) {
    return SANE_STATUS_CANCELLED;
  }

  return SANE_STATUS_GOOD;
}
```

Impact

This issue may lead to remote code execution, where "remote" means a device or computer connected to the same network as the victim. For example, in a typical office environment the malicious device would need to be somewhere inside the building.

CVEs

- GHSL-2020-075 -> CVE-2020-12867
- GHSL-2020-079 -> CVE-2020-12866
- GHSL-2020-080 -> CVE-2020-12861
- GHSL-2020-081 -> CVE-2020-12864
- GHSL-2020-082 -> CVE-2020-12862
- GHSL-2020-083 -> CVE-2020-12863
- GHSL-2020-084 -> CVE-2020-12865

Coordinated Disclosure Timeline

This report was subject to the GHSL coordinated disclosure policy.

- 2020-04-21 reported: https://gitlab.com/sane-project/backends/-/issues/279
- 2020-04-21 acknowledged by Olaf Meeuwissen
- 2020-05-14 CVE request submitted to Mitre
- 2020-05-17 bugs announced on http://www.sane-project.org/ and on their mailing list
- 2020-05-30 issue 279 is derestricted. The original PoC is attached to the issue and is therefore also publicly visible.

Resources

- https://gitlab.com/sane-project/backends/-/issues/279
- https://alioth-lists.debian.net/pipermail/sane-announce/2020/000041.html
- https://github.com/github/securitylab/tree/38b182e96a48f19b412039c0b321d6faec2b5c55/SecurityExploits/SANE/epsonds_CVE-2020-12861

Credit

These issues were discovered and reported by GHSL team member @kevinbackhouse (Kevin Backhouse).

Contact

You can contact the GHSL team at `securitylab@github.com`, please include the relevant GHSL IDs in any communication regarding these issues.