Multiple vulnerabilities in xrdp

Discovered by: Hamid Ebadi
CSIRT Team Member
Amirkabir University CSIRT Laboratory (APA Laboratory)

autcert@aut.ac.ir

Introduction

Based on the work of rdesktop, xrdp uses the remote desktop protocol to present a GUI to the user. The goal of this project is to provide a fully functional Linux terminal server, capable of accepting connections from rdesktop and Microsoft's own terminal server / remote desktop clients. (http://xrdp.sourceforge.net)
There are multiple buffer overflow vulnerabilities in xrdp which could be used by malicious attackers to execute arbitary code on the system.

Vulnerable version

xrdp <=0.4.1

Vulnerability

There are multiple vulnerabilities in Xrdp server which may allow a remote attacker to execute arbitary code.

1. A buffer overflow exists at line 880 in function “xrdp_bitmap_invalidate()” which is defined in "xrdp/xrdp_bitmap.c". This vulnerability could be exploited when processing specially crafted requests. The technical details of the vulnerability is as follows:

```
xrdp_bitmap_invalidate(struct xrdp_bitmap* self, struct xrdp_rect* rect)
{
  .
  .
  .
  char text[256];  //The variable which is overflowed later
.
.
.
  if (self->password_char != 0)
  {
    g_memset(text, self->password_char, self->edit_pos);
    text[self->edit_pos] = 0;
  }
  else
     g_strncpy(text, self->caption1, self->edit_pos); //text is overflowed by long size edit_pos
.
.
.
}
```

Table 1 – Vulnerability in xrdp_bitmap_invalidate

Function “g_strncpy()” uses “self->edit_pos” as the size parameter for copying “self->caption1” into text. We can increase self->edit_pos value in xrdp_bitmap_def_proc() function as described below. Since there is no limitation for self->edit_pos, attacker can

overflow text array when copying self->caption1 into it. Successful exploitation allows execution of arbitrary code.

Line 1205 , xrdp_bitmap.c, xrdp_bitmap_def_proc() function:

```
.
.
.
      c = get_char_from_scan_code(param2, scan_code, self->wm->keys,
                                  self->wm->caps_lock,
                                  self->wm->num_lock,
                                  self->wm->scroll_lock,
                                  self->wm->session->client_info->keylayout);
      if ((unsigned char)c >= 32)
      {
       add_char_at(self->caption1, c, self->edit_pos);
       self->edit_pos++;  //edit_pos is incremented by one
       xrdp_bitmap_invalidate(self, 0);
      }
.
.
.
```

Table 2 – edit_pos can be controlloed by attacker.

There is also another attack vector using edit_pos in /xrdp/funcs.c, in xrdp_bitmap_def_proc() function. There is no boundry checking in add_char_at() function when this function try to insert a character (c) in special position (self->edit_pos) into string (self->caption1).

```
/* returns error */
int APP_CC
xrdp_bitmap_def_proc(struct xrdp_bitmap* self, int msg,
                     int param1, int param2)
{
// code is summarized
 if (pressed_key== (left or up arrow) || pressed_key==(right or down arrow) ||
pressed_key==backspace || pressed_key==delete || pressed_key==end || pressed_key==home
... )
{
//do proper action.
}
     else
     {
      c = get_char_from_scan_code(param2, scan_code, self->wm->keys,
                                  self->wm->caps_lock,
                                  self->wm->num_lock,
                                  self->wm->scroll_lock,
                                  self->wm->session->client_info->keylayout);
      if ((unsigned char)c >= 32)
      {
       add_char_at(self->caption1, c, self->edit_pos);

       xrdp_bitmap_invalidate(self, 0);
      }
```

```
    }
}
```

Table 3 – Another attack vector in xrdp_bitmap_def_proc()

```
/* add a ch at index position in text, index starts at 0 */
/* if index = -1 add it to the end */
int APP_CC
add_char_at(char* text, char ch, int index)
{
  int len;
  int i;

  len = g_strlen(text);
  if (index >= len || index < 0)
  {
    text[len] = ch;
    text[len + 1] = 0;
    return 0;
  }
  for (i = len - 1; i >= index; i--)
  {
    text[i + 1] = text[i];
  }
  text[i + 1] = ch;
  text[len + 1] = 0;
  return 0;
}
```

Table 4 – add_char_at() implementation

2. Another vulnerability exists in the “rdp_rdp_process_color_pointer_pdu ()” function in "rdp/rdp_rdp.c " file when xrdp tries to establish a connection to another rdp server. In this function both ‘dlen’ and ‘mlen’ are initialized from input data and both of these variables are used as the size parameter for memcpy without any boundry checking, so attacker can overflow cursor->data and cursor->mask. The technical details of the vulnerability is as follows:

```
/* Process a color pointer PDU */
static void APP_CC
rdp_rdp_process_color_pointer_pdu(struct rdp_rdp* self, struct stream* s)
{
  int cache_idx;
  int dlen;
  int mlen;
  struct rdp_cursor* cursor;

  in_uint16_le(s, cache_idx);
  cursor = self->cursors + cache_idx;
  in_uint16_le(s, cursor->x);
  in_uint16_le(s, cursor->y);
  in_uint16_le(s, cursor->width);
  in_uint16_le(s, cursor->height);
```

```
  in_uint16_le(s, mlen); /* mask length */  //s is of type stream(see below)
  in_uint16_le(s, dlen); /* data length */  //
  in_uint8a(s, cursor->data, dlen);  //Vulnerable code. cursor->data may be overflowed
  in_uint8a(s, cursor->mask, mlen); //Vulnerable code. cursor->mask may be overflowed
  self->mod->server_set_cursor(self->mod, cursor->x, cursor->y,
                               cursor->data, cursor->mask);
}
```

"stream" is a structure to store input data.

```
/* parser state */
struct stream
{
  char* p;
  char* end;
  char* data;
  int size;
  /* offsets of various headers */
  char* iso_hdr;
  char* mcs_hdr;
  char* sec_hdr;
  char* rdp_hdr;
  char* channel_hdr;
  char* next_packet;
};
```

"in_uint8a" is a macro to copy data from pointer "p" in stream to specified location

```
#define in_uint8a(s, v, n) \
{ \
  g_memcpy((v), (s)->p, (n)); \
  (s)->p += (n); \
}
struct rdp_cursor
{
  int x;
  int y;
  int width;
  int height;
  char mask[(32 * 32) / 8];
  char data[(32 * 32) * 3];
};
```

In this vulnerability we confirmed the overflow but no crash happens here.

Workaround

There are currently no patches available for these vulnerabilities.

Credit

This vulnerability has been discovered by *Hamid Ebadi* from Amirkabir university CSIRT laboratory.

autcert@aut.ac.ir
https://www.ircert.cc