Using Xterm to Navigate the Huge Color Space

An interesting, and probably very old (1986?) flaw in libX11 allow crafted color names to mess up the client-server communication protocol and inject malicious X server requests. The flaw is comparable to SQLi injecting commands into database connections granting an attacker access to all features of the connection protocol. Searching for the root cause and also the origin of the problematic code turned into an excursion into software archeology.

Looking back to the early age of computer graphics, color space was quite limited due to the small color depth with ancient hard- and software. Often the whole color space was reduced to just black/white (1 bit), 16 colors (4 bits), later 8 or 16 bits. Even being a child of those times the early X-window system protocol specification considered way bigger color spaces, e.g. by allowing 16 bit color depth per color channel in AllocColor, see X.V11R1.tar.gz from 1987, file *doc/Protocol/spec*.

```
3896 AllocColor
3897         cmap: COLORMAP
3898         red, green, blue: CARD16
3899     =>
3900         pixel: CARD32
3901         red, green, blue: CARD16
3902
3903         Errors: Colormap, Alloc
3904
3905         Allocates a read-only colormap entry corresponding
             to the closest RGB
```

Also Xterm, a simple graphical terminal, is of similar age and it was even maintained together with the ancient xfree86 X implementation, see a CVS commit entry from 19970618 changelog (older resources should exist, but I cold not locate them).

```
Revision 3.491 / (view) - annotate - [select for diffs] , Wed Jun 18 07:25:49 1997 UTC (9 years, 6 months ago) by dawes
Branch: MAIN
Changes since 3.490: +6 -2 lines
Diff to previous 3.490

more xterm termcap updates
```

While being that old, still the procolol specification, a free server implementation (now Xorg) and even Xterm still exist and are maintained.

Exploring the Color Space Using OSCs

Even with *AllocColor* having 48 bits to specifiy a color, quite some hardware back then could only produce 16 different colors using a fixed palette instead of a per-pixel RGB color. The palette could be changed but therefore different programs sharing a screen via the X server for output had to negotiate the colors on the palette or the server had to tell, which are the colors available looking similar to the specification. The X protocol helped to do that by providing the *AllocColor* but also the *LookupColor* request. The later could be used to lookup a color by name.

```
LookupColor
1     92          opcode
1                 unused
2     3+(n+p)/4   request length
4     COLORMAP    cmap
2     n           length of name
2                 unused
n     STRING8     name              | opcode, x, len
p                 unused, p=pad(n)
```

Graphical terminals like Xterm, Rxvt, ... use(d) exactly such requests to negotiate the color then used to display graphical output. But they did this not only for themselves but also for programs run by the terminal. To allow hilighting of text output, programs can emit terminal control sequences to change the color or font. The terminal will react to the control sequences, select an appropriate color and paint the program output using X protocol requests sent to the graphics server. For example *ls --color=always /home* creates a colored directory listing:

```
$ ls --color=always /home | xxd
00000000: 1b5b 306d 1b5b 3031 3b33 346d 7465 7374  .[0m.[01;34mtest
00000010: 1b5b 306d 0a                             .[0m.
```

While the terminal control sequence used by *ls* may only select a predefined color, OSCs (Operating System Command Sequences) allow requesting any color by name or RGB specification, e.g. using those Python3 commands:

```
sys.stdout.buffer.write(b'\x07\x1b]11;purple4\x07')
sys.stdout.buffer.write(b'\x07\x1b]11;#006f00015858\x07')
```

The color *#006f00015858* is just a nice dark blue and is submitted to the X server using an *AllocColor* request. The full X protocol packet data is:

```
[op]    [length][colormap id   ][red   ][green ][blue  ][pad   ]
\x54\x75\x04\x00\x20\x00\x00\x00\x6f\x00\x01\x00\x58\x58\x00\x65
                              [op][mo][length][next X frame ..
```

The 0x54 is the opcode (AllocColor), 0x04 the length of the packet (0x4 * 4 bytes), then color map id and the color specification *\x6f\x00\x01\x00\x58\x58\x00\x65*. But the color specification itself (without the AllocColor header) would also be a valid X protocol request, but one for *SetAccessControl*. With mode set to 0 it will grant anyone access to the X server session, also key loggers, screen shot tools, ...:

```
SetAccessControl
1      111    opcode (0x6f)
1             mode
       0      Disable
       1      Enable
2      1      request length
```

Enjoying Exotic Colors With Loooong Names

Using the control sequence *b'\x07\x1b]11;purple4\x07'* allows color lookup by name. Emiting such sequence to Xterm will cause Xterm to perform a *LookupColor* X protocol request with the given name:

```
LookupColor
1     92              opcode
1                     unused
2     3+(n+p)/4       request length
4     COLORMAP        cmap
2     n               length of name
2                     unused
n     STRING8         name
p                     unused, p=pad(n)
```

As there is no maximum OSC (operating system control) sequence length specified (at least I did not find any limits), Xterm takes whatever the program run inside Xterm requests. As the Xterm C program code uses *size_t* (64 bit on amd64 architecture), Xterm might allocate memory for color names many GBs large. Such long names may exhaust computer RAM in general, but XTerm itself is not harmed by it (unless OOM-killer is triggered). But when submitting such long names via the X protocol, the [CARD16] packet length (4 bytes per unit) will overflow but nevertheless the full packet data is transmitted. Therefore the X-server handles a truncated *LookupColor* packet. The part of the color name that did not fit in the packet is treated as subsequent X protocol request. See *libx11/libx11-1.7.0/src/LookupCol.c* how the request packet is created:

```
35 Status
36 XLookupColor (
```

```
...
81          n = (int) strlen (spec);
82          LockDisplay(dpy);
83          GetReq (LookupColor, req);
84          req->cmap = cmap;
85          req->nbytes = n;
86          req->length += (n + 3) >> 2;
87          Data (dpy, spec, (long)n);
88          if (!_XReply (dpy, (xReply *) &reply, 0, xTrue)) {
```

Thus looking up an overlong color name allows injection of arbitrary X protocol requests with user controlled data and most likely loss of correct reading frame on the connection. The OSC data may contain only printable characters as color name. With such data it is not possible to inject e.g. a valid *SetAccessControl* packet directly. But other valid X protocol requests can be injected that consume not only data from the botched *LookupColor* request but also the first 8 bytes of a following *AllocColor* X protocol request. Due to the loss of the reading frame, the RGB color specification data is then executed as a separate X request with fully user controlled data. The server will process it and disable any X server authentication as requested.

The PoC enjoy-all-the-colors.py also injects a *ChangeKeyboardMapping* request (configuration option) to mess up the keyboard map (making the keyboard unusable). The *SetAccessControl* request then turns of X-server authentication as can be seen by "xhost". If the attacker for any reason can reach the X-server socket he can then take full control of the logged on user session and usually execute code as that user. Another interesting X protocol request to inject is the *XTestFakeInput* request to inject keystrokes to "type" shell commands in the terminal.

The PoC disabled access control in 10/10 attempts while keymap messup plus access requests worked only 5/10 attempts in the testing environment.

```
$ xhost
access control enabled, only authorized clients can connect
SI:localuser:test
$ xterm -e ./enjoy-all-the-colors.py
xterm: warning, error event received:
X Error of failed request:  BadFont (invalid Font parameter)
  Major opcode of failed request:  48 (X_QueryTextExtents)
  Resource id in failed request:  0x54545454
  Serial number of failed request:  312
  Current serial number in output stream:  312
$ xhost
access control disabled, clients can connect from any host
SI:localuser:test
```

The exploit also works when the control sequences are embedded in log files, then (hazardously) displayed using *cat* or *tail -f*. Also running the PoC via SSH on a remote host will disable authentication on the **local** host.

```
$ xhost
access control enabled, only authorized clients can connect
SI:localuser:test
$ xterm -e tail -f x.out
xterm: warning, error event received:
...
$ xhost
access control disabled, clients can connect from any host
SI:localuser:test
$ xhost -
access control enabled, only authorized clients can connect
$ xterm -e ssh -F test@test ./enjoy-all-the-colors.py
xterm: warning, error event received:
...
$ xhost
access control disabled, clients can connect from any host
SI:localuser:test
```

All Fun Comes To an End

Meanwhile patches were released. Upstream patches in libX11-1.7.1 make various requests with oversized string data fail with an error without attempting to transmit the data to the server. See libx11 git commit 8d2e02ae650f00c4a53deb625211a0527126c605.

A mitigation patch was also to xterm (patch #367) to ignore GB-sized color name requests in control sequences.

For more information see UNPAR-2021-1 advisory or CVE-2021-31535. Various Linux distribution related bugtrackers show how the patches reach the end users, e.g. [BugArchLinux-1968], [BugDebian-988737], [BugGentoo-790824], [BugRedHat-1961822], [BugSuse-1182506].

References:

- [X11-R1]: Specification from 19870915: https://www.x.org/releases/X11R1/X.V11R1.tar.gz
- [OCS]: OSCs (Operating System Command Sequences)
- [PoC]: enjoy-all-the-colors.py
- [UNPAR-2021-1]: UNPAR-2021-1 advisory
- [LibX11Patch]: Git commit 8d2e02ae650f00c4a53deb625211a0527126c605
- [XtermPatch]: Patch #367
- [CVE-Mitre] CVE-2021-31535
- [BugArchLinux-1968] Arch Linux vulnerability group 1968
- [BugDebian-988737] Debian bug report 988737
- [BugGentoo-790824] Gentoo bug report 790824
- [BugRedHat-1961822] RedHat bug report 1961822
- [BugSuse-1182506] Suse bug report 1182506

Notes:

Comments are welcome, but there is no forum system im place yet. If there is something important to be added to this page, please send it as e-mail. Legal: Appropriate comments will be published, there is no right for you to get them published, use a "Nick:" entry in your comment otherwise for attribution "Anonymous" is used, comment mails are deleted after processing (GDPR), IPR rights for your comment stay with you except that the content may be used to correct or improve the page while referencing to your comment as source of the change, comment data is not submitted to third parties. Phuuu, inhale!