IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Assessing the Linux Desktop's Security

Ilja van Sprundel <ivansprundel@ioactive.com>

Who am I?

- IOActive
- Director of Penetration Testing
- Pentest
- Code review
- Break stuff for fun and profit ☺

agenda

- Intro
- Observations
- Problems
- More observations
- More problems summary
- Solutions ?

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

What this talk is about

- Local security of linux on the desktop

Intro

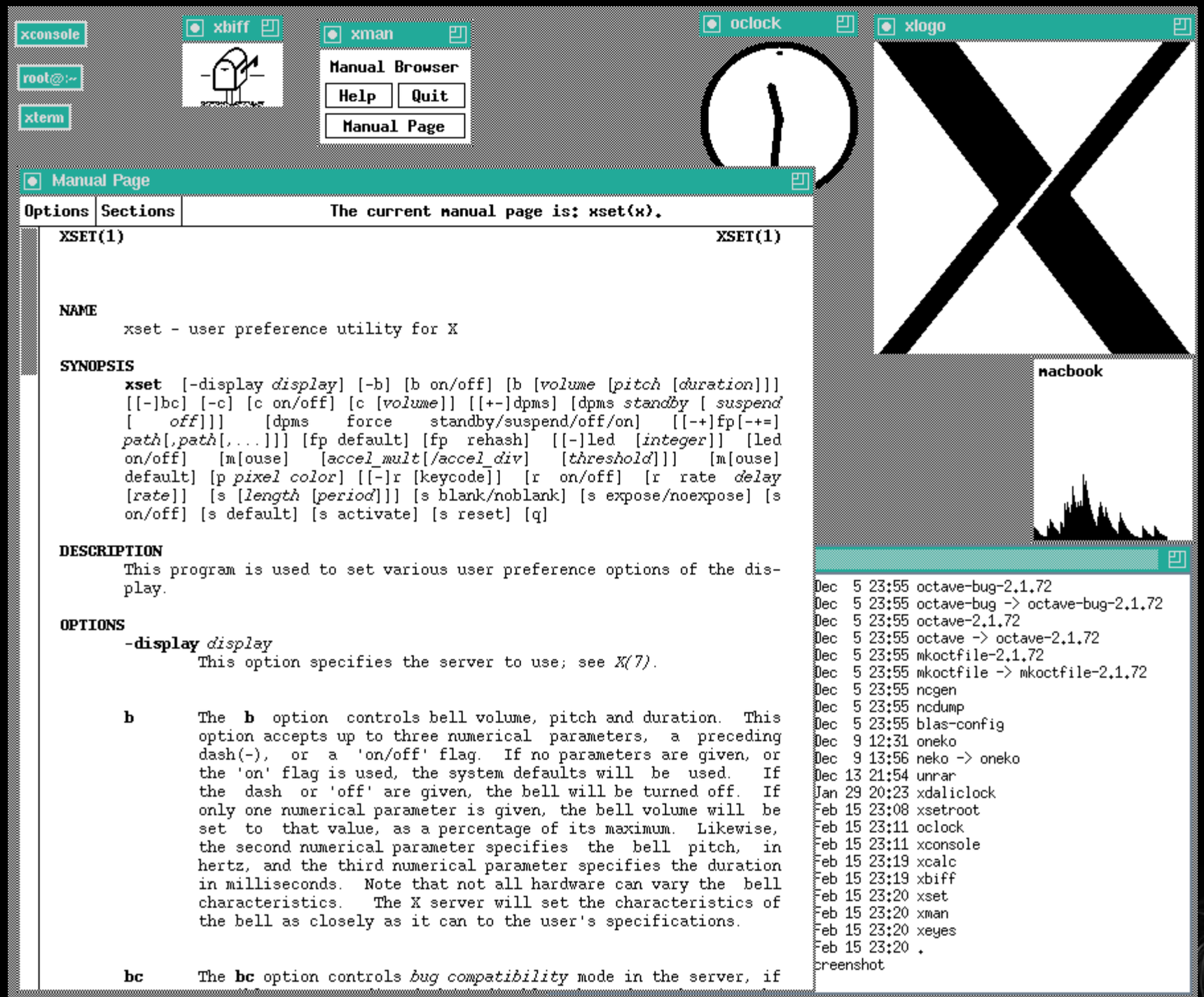
- Used to use linux as my main desktop machine
- Switched to windows about 7 years ago
  - Mainly for work reasons
- Have used linux sporadically since then
  - However, not as a desktop OS
- Things seem to have changed somewhat

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

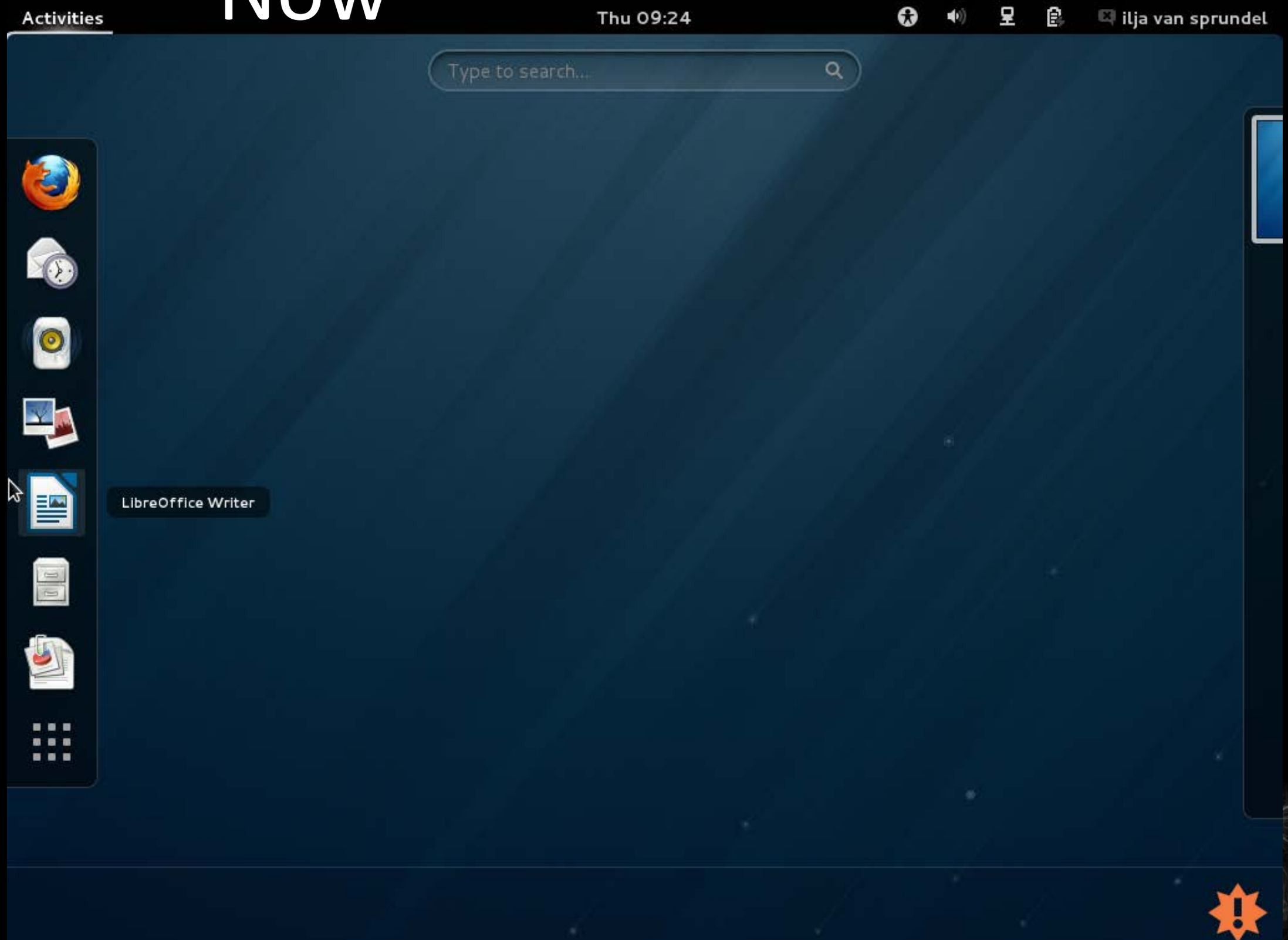
Intro

- Installed ubuntu during the holidays last year
- Boots into a gui, looks like some mix of windows and osx gui
- It's pretty clear they want to matter as a desktop os
  - 7-10 years ago, it looked like, well, X
  - Today, it looks like something my grandmother can use
- Also looked at fedora, opensuse, knoppix
- Also took a quick look at osx and opensolaris (openindiana)

A decade ago


xconsole
root@:~
xterm
xbiff
xman
Manual Browser
Help
Quit
Manual Page
oclock
xlogo
Manual Page
Options
Sections
The current manual page is: xset(x).
XSET(1)
XSET(1)
NAME
xset - user preference utility for X
SYNOPSIS
xset [-display display] [-b] [b on/off] [b [volume [pitch [duration]]] [[-]bc] [-c] [c on/off] [c [volume]] [[+-]dpms] [dpms standby [suspend [off]]] [dpms force standby/suspend/off/on] [[-+]fp[-+=] path[,path[,...]]] [fp default] [fp rehash] [[-]led [integer]] [led on/off] [m[ouse] [accel_mult[/accel_div] [threshold]]] [m[ouse] default] [p pixel color] [[-]r [keycode]] [r on/off] [r rate delay [rate]] [s [length [period]]] [s blank/noblank] [s expose/noexpose] [s on/off] [s default] [s activate] [s reset] [q]
DESCRIPTION
This program is used to set various user preference options of the display.
OPTIONS
-display display
This option specifies the server to use; see X(7).
b
The b option controls bell volume, pitch and duration. This option accepts up to three numerical parameters, a preceding dash(-), or a 'on/off' flag. If no parameters are given, or the 'on' flag is used, the system defaults will be used. If the dash or 'off' are given, the bell will be turned off. If only one numerical parameter is given, the bell volume will be set to that value, as a percentage of its maximum. Likewise, the second numerical parameter specifies the bell pitch, in hertz, and the third numerical parameter specifies the duration in milliseconds. Note that not all hardware can vary the bell characteristics. The X server will set the characteristics of the bell as closely as it can to the user's specifications.
bc
The bc option controls bug compatibility mode in the server, if
macbook
Dec 5 23:55 octave-bug-2.1.72
Dec 5 23:55 octave-bug -> octave-bug-2.1.72
Dec 5 23:55 octave-2.1.72
Dec 5 23:55 octave -> octave-2.1.72
Dec 5 23:55 mkoctfile-2.1.72
Dec 5 23:55 mkoctfile -> mkoctfile-2.1.72
Dec 5 23:55 ncgen
Dec 5 23:55 ncdump
Dec 5 23:55 blas-config
Dec 9 12:31 oneko
Dec 9 13:56 neko -> oneko
Dec 13 21:54 unrar
Jan 29 20:23 xdaliclock
Feb 15 23:08 xsetroot
Feb 15 23:11 oclock
Feb 15 23:11 xconsole
Feb 15 23:19 xcalc
Feb 15 23:19 xbiff
Feb 15 23:20 xset
Feb 15 23:20 xman
Feb 15 23:20 xeyes
Feb 15 23:20 .
creenshot

Now


Activities
Thu 09:24
ilja van sprundel
Type to search...
LibreOffice Writer

Intro

- Initially had about a week or so of time to play around with this
- Made some interesting observations
  - Simple command line tools
  - Some code reading
- Found some clear problems
- Maybe a solution or two

What was actually done (observations)

- Started off with very simple commands to enumerate some entrypoints
- Wanted to see:
  - shared memory (and it's acl's) (ipcs)
  - Udp/tcp/unix sockets exposed locally (netstat –pln)
  - Look at cron scripts
  - …

What was actually done (observations)

- …
- Wanted to see:
  - Look for world writable files and directories
    - find / -perm -0666 -type f
    - find / -perm -0666 -type d
  - Enumerate suid files
    - find / -perm +2000 -o -perm +4000 -type f
  - Enumerate dbus system endpoints
    - dbus-send --system --type=method_call --print-reply --dest=org.freedesktop.DBus /org/freedesktop/DBus org.freedesktop.DBus.ListNames

What was actually done (observations)

- Expected this to be pretty boring and coming up almost empty handed
- Varying results for various distro's and operating systems
- There seem to be some systemic issues across all of them
- Is no one doing trivial entrypoint analysis before shipping ?

Overall finds (problems)

- Without disclosing details (bugs aren't fixed)
  - world writeable shared memory
  - World writable scripts
  - Really really bloated suid binaries
  - misconfigurations
  - Over 60 finds in less than a week
- The goal of this talk isn't any specific bug

More observations

- Dbus
- Relatively new attack surface
- X/Gnome/KDE specific
- Ipc mechanism to pull information about the system or the current session
- Session is probably not that interesting
- System could be!

More observations

- Dbus
- Loads of new attack surface
  - Configuration
  - Design (e.g. repurposing)
  - Implementation (e.g. buffer overflow)
- There seem to be piles and piles of these installed on default linux distro's (40-60)

More observations

- Dbus system
- Configure who can read / write to it
  - Under what circumstances (root, console, group, default, …)
  - Where (what interface, …)
- /etc/dbus-1/*
- Xml-alike file that specifies this

More observations

- Have been configuration bugs here in the past:
  - http://pkgs.fedoraproject.org/cgit/sectool.git/tree/sectool-0.9.5-dbus.patch?id=aedb3ef7f7e5ab22d5438bfb7eee63489ccf3244;id2=4859832281f0e08c6fbe48fc252c4199a0e9e322

More observations

blob: aedb3ef7f7e5ab22d5438bfb7eee63489ccf3244 (plain)

```
1  diff -up sectool-0.9.5/org.fedoraproject.sectool.mechanism.conf.dbus sectool-0.9.5/org.fedoraproject.sectool.mechanism.conf
2  --- sectool-0.9.5/org.fedoraproject.sectool.mechanism.conf.dbus 2012-04-03 15:21:05.521186717 +0200
3  +++ sectool-0.9.5/org.fedoraproject.sectool.mechanism.conf      2012-04-03 15:23:57.602490428 +0200
4  @@ -9,7 +9,6 @@
5         <allow own="org.fedoraproject.sectool.mechanism"/>
6     </policy>
7     <policy context="default">
8  -      <allow send_destination="org.fedoraproject.sectool.mechanism"/>
9  -      <allow send_type="method_call"/>
10 +      <allow send_destination="org.fedoraproject.sectool.mechanism" send_type="method_call"/>
11    </policy>
12 </busconfig>
```

More observations

- Easy to make config mistakes
- Similar to android intent permissions being set in their AndroidManifest.xml file

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

More observations

- I wanted to look a bit closer at the suids
- Asked readelf to give me a list of the library dependencies (readelf –d)
- All those libraries themselves are attack surface as well
- Some have just libc
- Others depends on huge blobs of network parsers (e.g. X).

More observations

```
ilja@ilja-VirtualBox:~$ readelf -d /usr/bin/kppp

Dynamic section at offset 0x82ec8 contains 33 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [libkde3support.so.4]
 0x00000001 (NEEDED)                     Shared library: [libQt3Support.so.4]
 0x00000001 (NEEDED)                     Shared library: [libkio.so.5]
 0x00000001 (NEEDED)                     Shared library: [libkdeui.so.5]
 0x00000001 (NEEDED)                     Shared library: [libkdecore.so.5]
 0x00000001 (NEEDED)                     Shared library: [libQtCore.so.4]
 0x00000001 (NEEDED)                     Shared library: [libQtDBus.so.4]
 0x00000001 (NEEDED)                     Shared library: [libQtGui.so.4]
 0x00000001 (NEEDED)                     Shared library: [libstdc++.so.6]
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
```

More observations

File: /usr/games/gnomine

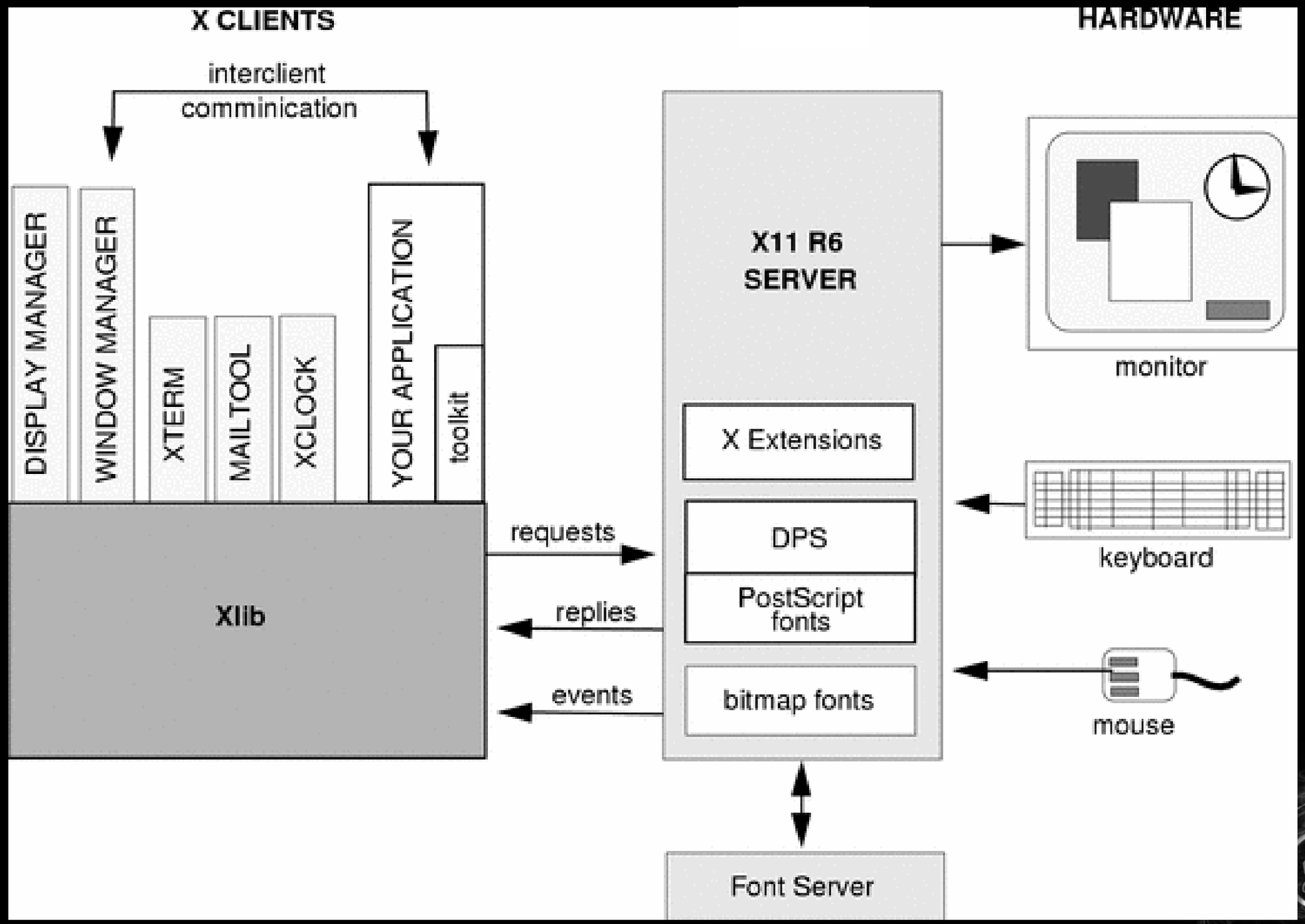
Dynamic section at offset 0x17ea4 contains 35 entries:

Tag	Type	Name/Value
0x00000001	(NEEDED)	Shared library: [libgtk-3.so.0]
0x00000001	(NEEDED)	Shared library: [libgdk-3.so.0]
0x00000001	(NEEDED)	Shared library: [libpangocairo-1.0.so.0]
0x00000001	(NEEDED)	Shared library: [libpango-1.0.so.0]
0x00000001	(NEEDED)	Shared library: [librsvg-2.so.2]
0x00000001	(NEEDED)	Shared library: [libgio-2.0.so.0]
0x00000001	(NEEDED)	Shared library: [libgdk_pixbuf-2.0.so.0]
0x00000001	(NEEDED)	Shared library: [libgobject-2.0.so.0]
0x00000001	(NEEDED)	Shared library: [libglib-2.0.so.0]
0x00000001	(NEEDED)	Shared library: [libcairo.so.2]
0x00000001	(NEEDED)	Shared library: [libpthread.so.0]
0x00000001	(NEEDED)	Shared library: [libc.so.6]

More observations

- I spend some time zooming in on those using X
- X is a client/server protocol to be used for the gui in most unices
  - Including most linux distributions
- It’s networked (can be tcp/ip, ipc, …)
  - Binary protocol
- suids in question are basically clients
- You can make them connect to arbitrary X servers using the DISPLAY variable
  - Who says these have to be well behaving X servers ?

X network architecture



More observations

- The X client libraries (Xlib) are clear attack surface
- Spend about a day looking at the network parsing code in Xlib
- Things are ***really really*** bad
  - Binary protocol parsers in C.
    - A lot of them written in the 80's
  - Server data appears to be trusted. Very little validation
  - > 60 trivial bugs
- It's clear that code was not written with trust boundaries in mind at all.

More observations

```
Status XAllocColorCells(
    register Display *dpy,
    Colormap cmap,
    Bool contig,
    unsigned long *masks, /* LISTofCARD32 */ /* RETURN */
    unsigned int nplanes, /* CARD16 */
    unsigned long *pixels, /* LISTofCARD32 */ /* RETURN */
    unsigned int ncolors) /* CARD16 */
{
...
    xAllocColorCellsReply rep;
...
    status = _XReply(dpy, (xReply *)&rep, 0, xFalse);

    if (status) {
        _XRead32 (dpy, (long *) pixels, 4L * (long) (rep.nPixels));
        _XRead32 (dpy, (long *) masks, 4L * (long) (rep.nMasks));
    }
...
}
```

More observations

```
XTimeCoord *XGetMotionEvents(register Display *dpy, Window w, Time start, Time stop, int *nEvents)
/* RETURN */
{
    xGetMotionEventsReply rep;
...
    if (!_XReply (dpy, (xReply *)&rep, 0, xFalse)) {

        }
...
        if (! (tc = (XTimeCoord *)
                Xmalloc( (unsigned)
                        (nbytes = (long) rep.nEvents * sizeof(XTimeCoord))))) {
...
        }
...
        for (i = rep.nEvents, tcptr = tc; i > 0; i--, tcptr++) {
            _XRead (dpy, (char *) &xtc, nbytes);
            tcptr->time = xtc.time;

        }
...}
```

More observations

```
_XkbReadGetDeviceInfoReply(   Display *                dpy,
                              xkbGetDeviceInfoReply *  rep,
                              XkbDeviceInfoPtr         devi)
{
...
    if (rep->nBtnsWanted>0) {
        act= &devi->btn_acts[rep->firstBtnWanted];
        bzero((char *)act,(rep->nBtnsWanted*sizeof(XkbAction)));
    }
...
        int size;
        act= &devi->btn_acts[rep->firstBtnRtrn];
        size= rep->nBtnsRtrn*SIZEOF(xkbActionWireDesc);
        if (!_XkbCopyFromReadBuffer(&buf,(char *)act,size))
            goto BAILOUT;
...
}
```

More observations

```
Private Bool
_XimXGetReadData(
    Xim                 im,
    char            *buf,
    int                 buf_len,
    int                *ret_len,
    XEvent             *event)
{
...
        return_code = XGetWindowProperty(im->core.display,
                spec->lib_connect_wid, prop, 0L,
                (long)((length + 3)/ 4), True, AnyPropertyType,
                &type_ret, &format_ret, &nitems, &bytes_after_ret, &prop_ret);
...
        if (buf_len >= length) {
            (void)memcpy(buf, prop_ret, (int)nitems);
...
        }
...
}
```

More observations

```
static char * ReadInFile(_Xconst char *filename)
{
    register int fd, size;
...
    if ( (fd = _XOpenFile (filename, O_RDONLY)) == -1 )
        return (char *)NULL;
...
        if ( (fstat(fd, &status_buffer)) == -1 ) {
            close (fd);
            return (char *)NULL;
        } else
            size = status_buffer.st_size;
    }

    if (!(filebuf = Xmalloc(size + 1))) { /* leave room for '\0' */
        close(fd);
        return (char *)NULL;
    }
    size = read (fd, filebuf, size);
...
}
```

More observations

```
static char*
TransFileName(Xim im, char *name) {
    char *home = NULL, *lcCompose = NULL;
    char dir[XLC_BUFSIZE];
    char *i = name, *ret, *j;
    int l = 0;

    while (*i) {
        if (*i == '%') {
                i++;
                switch (*i) {
                    case 'H':
                        home = getenv("HOME");
                        if (home)
                          l += strlen(home); ← possible int overflow (long HOME and loads of %H's)
                        break;
                    case 'L':
                     if (lcCompose == NULL)
                          lcCompose = _XlcFileName(im->core.lcd, COMPOSE_FILE);
                     if (lcCompose)
                          l += strlen(lcCompose); ← possible integer overflow (long lcCompose and loads of %L's)
                        break;
                    case 'S':
                     xlocaledir(dir, XLC_BUFSIZE);
                     l += strlen(dir); ← possible integer overflow (long dir and loads of %S's)
                        break;
```

More observations (cont)

```
 j = ret = Xmalloc(l+1); ← integer overflow, alloc too short if any of the int overflows occured
...
    i = name;
    while (*i) {
        if (*i == '%') {
                i++;
                switch (*i) {
                     case '%':
                      *j++ = '%';
                        break;
                     case 'H':
                        if (home) {
                            strcpy(j, home); ← buffer overflow if integer overflow occured
                            j += strlen(home);
                        }
                        break;
                case 'L':
                        if (lcCompose) {
                         strcpy(j, lcCompose); ← buffer overflow if integer overflow occured
                         j += strlen(lcCompose);
                 }
                        break;
                case 'S':
                 strcpy(j, dir); ← buffer overflow if integer overflow occured
                 j += strlen(dir);
                        break;
```

More observations

```
XFixesCursorImage *XFixesGetCursorImage (Display *dpy) {
...
    xXFixesGetCursorImageAndNameReply          rep;
    int                                        npixels;
...
    if (!_XReply (dpy, (xReply *) &rep, 0, xFalse))
    {
...
    }
...
    npixels = rep.width * rep.height; ← ushort * ushort
...
    rlength = (sizeof (XFixesCursorImage) +
               npixels * sizeof (unsigned long) +
               nbytes_name + 1);
...
    image = (XFixesCursorImage *) Xmalloc (rlength);
...
    image->x = rep.x;
...
}
```

More observations

```
Bool DMXGetScreenAttributes(Display *dpy, int physical_screen,
                            DMXScreenAttributes *attr)
{
...
    xDMXGetScreenAttributesReply rep;
...
    if (!_XReply(dpy, (xReply *)&rep,
                 (SIZEOF(xDMXGetScreenAttributesReply) - 32) >> 2, xFalse)) {
        UnlockDisplay(dpy);
        SyncHandle();
        return False;
    }
    attr->displayName  = Xmalloc(rep.displayNameLength + 1 + 4 /* for pad */);
    _XReadPad(dpy, attr->displayName, rep.displayNameLength);
    attr->displayName[rep.displayNameLength] = '\0';
...
}
```

More observations

```
int
XGetDeviceButtonMapping(
      register Display        *dpy,
      XDevice                 *device,
      unsigned char            map[],
      unsigned int    nmap)
{
      int status = 0;
      unsigned char mapping[256];   /* known fixed size */
...
      xGetDeviceButtonMappingReply rep;
...
      status = _XReply(dpy, (xReply *) & rep, 0, xFalse);
      if (status == 1) {
            nbytes = (long)rep.length << 2;
            _XRead(dpy, (char *)mapping, nbytes);
...
      }
...
}
```

More observations

```
XEventClass *
XGetDeviceDontPropagateList(
    register Display        *dpy,
    Window                   window,
    int                     *count)
{
...
    xGetDeviceDontPropagateListReply rep;
...
    if (!_XReply(dpy, (xReply *) & rep, 0, xFalse)) {
...
    }
...
        list = (XEventClass *) Xmalloc(rep.length * sizeof(XEventClass));
        if (list) {
...
            for (i = 0; i < rep.length; i++) {
                _XRead(dpy, (char *)(&ec), sizeof(CARD32));
                list[i] = (XEventClass) ec;
            }
        }
```

More observations

```
static int
_XIPassiveGrabDevice(Display* dpy, int deviceid, int grabtype, int detail,
                     Window grab_window, Cursor cursor,
                     int grab_mode, int paired_device_mode,
                     Bool owner_events, XIEventMask *mask,
                     int num_modifiers, XIGrabModifiers *modifiers_inout)
{
...
    if (!_XReply(dpy, (xReply *)&reply, 0, xFalse))
    {
...
    }

    failed_mods = calloc(reply.num_modifiers, sizeof(xXIGrabModifierInfo));
...
    _XRead(dpy, (char*)failed_mods, reply.num_modifiers * sizeof(xXIGrabModifierInfo));

    for (i = 0; i < reply.num_modifiers; i++)
    {
        modifiers_inout[i].status = failed_mods[i].status;
    }
```

More observations

```
XineramaScreenInfo *
XineramaQueryScreens(
    Display *dpy,
    int      *number
)
{
     XExtDisplayInfo                *info = find_display (dpy);
     xXineramaQueryScreensReplyrep;
....
          if((scrnInfo = Xmalloc(sizeof(XineramaScreenInfo) * rep.number))) {
...
               for(i = 0; i < rep.number; i++) {
                    _XRead(dpy, (char*)(&scratch), sz_XineramaScreenInfo);
                    scrnInfo[i].screen_number = i;
                    scrnInfo[i].x_org              = scratch.x_org;
                    scrnInfo[i].y_org              = scratch.y_org;
                    scrnInfo[i].width    = scratch.width;
                    scrnInfo[i].height    = scratch.height;
               }
...
     }
```

More observations

```
XvImageFormatValues * XvListImageFormats (
    Display          *dpy,
    XvPortID         port,
    int              *num
){
...
   xvListImageFormatsReply rep;
...
   if (_XReply(dpy, (xReply *)&rep, 0, xFalse) == 0) {
...
   }
...
        int size = (rep.num_formats * sizeof(XvImageFormatValues));
...
        if((ret = Xmalloc(size))) {
...
             for(i = 0; i < rep.num_formats; i++) {
...
             }
        }
...
```

More observations

```
Bool
XF86VidModeGetGammaRamp (
    Display *dpy,
    int screen,
    int size,
    unsigned short *red,
    unsigned short *green,
    unsigned short *blue
)
{
...
    xXF86VidModeGetGammaRampReply rep;
...
    if (!_XReply (dpy, (xReply *) &rep, 0, xFalse)) {
...
    }
    if(rep.size) {
        _XRead(dpy, (char*)red, rep.size << 1);
        _XRead(dpy, (char*)green, rep.size << 1);
        _XRead(dpy, (char*)blue, rep.size << 1);
    }
... }
```

More observations

```
Bool
uniDRIOpenConnection(dpy, screen, hSAREA, busIdString)
      Display *dpy;
      int screen;
      drm_handle_t *hSAREA;
      char **busIdString;
{
...
...
    xXF86DRIOpenConnectionReply rep;
...
          if (!(*busIdString = (char *)Xcalloc(rep.busIdStringLength + 1, 1))) {
...
          }
          _XReadPad(dpy, *busIdString, rep.busIdStringLength);
...
}
```

More observations

```
static XcursorFileHeader *
_XcursorReadFileHeader (XcursorFile *file)
{
...
    if (!_XcursorReadUInt (file, &head.ntoc)) ← unsigned 32bit ntoc var read from file
        return NULL;
...
    fileHeader = _XcursorFileHeaderCreate (head.ntoc); ← passed on to allocate buffer
...
    for (n = 0; n < fileHeader->ntoc; n++)
    {
        if (!_XcursorReadUInt (file, &fileHeader->tocs[n].type))
            break;
        if (!_XcursorReadUInt (file, &fileHeader->tocs[n].subtype))
            break;
        if (!_XcursorReadUInt (file, &fileHeader->tocs[n].position))
            break;
    }
...
}
```

More observations (cont)

```
static XcursorFileHeader *
_XcursorFileHeaderCreate (int ntoc)
{
    XcursorFileHeader    *fileHeader;

    if (ntoc > 0x10000)
        return NULL;
    fileHeader = malloc (sizeof (XcursorFileHeader) +
                         ntoc * sizeof (XcursorFileToc));
    if (!fileHeader)
        return NULL;
    fileHeader->magic = XCURSOR_MAGIC;
    fileHeader->header = XCURSOR_FILE_HEADER_LEN;
    fileHeader->version = XCURSOR_FILE_VERSION;
    fileHeader->ntoc = ntoc;
    fileHeader->tocs = (XcursorFileToc *) (fileHeader + 1);
    return fileHeader;
}
```

More observations

```
static void ReqCleanup(
    Widget widget,
    XtPointer closure,
    XEvent *ev,
    Boolean *cont)
{
...
    char *value;
...
        (void) XGetWindowProperty(event->display, XtWindow(widget),
                                  event->atom, 0L, 1000000, True, AnyPropertyType,
                                  &target, &format, &length, &bytesafter,
                                  (unsigned char **) &value); ← should check return value.
         XFree(value);
...
}
```

More observations

- The X client libraries (Xlib)
- All discovered X bugs are ~~being~~ fixed
- The developer involved is actually ~~quite good~~
  - Amazing
  - Alan Coopersmith
  - Very deep understanding of X and the bugs involved
  - No pushback, no handholding
  - Worked tirelessly
  - 104 patches, with reviews and some tests in < 3 months
- And had some interesting comments

More observations

™I don't know how many setuid X clients still exist these days (is xterm still setuid on any platforms, or did they all get grantpt() or similar calls to avoid needing root?), but since we know there's more X clients than we can keep track of (especially once you get to home grown apps in various companies they've been using for decades), we have to assume there still may be some. It would be good to put a reminder in the security advisory that best practice is to separate out the parts of an application that require elevated privileges from the GUI to avoid such issues - GTK requires this, but not all toolkits do."

More observations

™Shoot me now. And then shoot daniels for not freeing us from XKB yet. And then shoot anyone who volunteers to try to fix XKB, before it's too late for them too."

™Here's my initial analysis of the first part of the Xlib set, until I got so tired my head started spinning trying to figure them out"

™Really, if your window shape is anywhere near 2^32 rectangles, what are you doing?"

™Yes, these [bugs] all seem possible, and far more feasible now than when this code was written, back when disk sizes were still measured in megabytes."

More observations

- Comment on LWN from one of the people that introduced some of these bugs in the ‘80’s

Numerous security issues in X Window System clients

Posted May 24, 2013 16:33 UTC (Fri) by **jg** (subscriber, #17537) [Link]

All I can say is that in 1984-1988 it was a very, very different world. We're talking about > 25 years ago; many LWN readers weren't born then.

There weren't bad guys out there. I remember the Morris worm hitting when I was working on the X11 protocol bindings. And Kevin Mitnick had just reared his head a year or two before.

Some of the bugs I clearly wrote in that period, and others copied my mistakes (and clearly made more of their own). Others of the just couldn't happen in practice on machines of that era (say, 2 megabyte MicroVax's).

And X is still used widely remotely: just usually via SSH tunnels.

But the case you really worry about is different: remoting you application to an untrusted/untrustworthy X server, which may want to break into your machine/device. This is still a thing of desire (but SSH has highly discouraged this capability). So these problems, in practice, don't happen much.

I'm still happy to see the bugs get fixed, of course.

More observations

- Debian's turn around on these bugs was ridiculously fast!
- 104 patches to merge in
- 2 week embargo
- Full releases and advisory on day embargo expired
  - ***No one else*** managed to do this
- I think Moritz Mühlenhoff deserves most of the credit for that one

More observations

- However, Raw X is rarely used nowadays
- There’s stuff build on top
- And they use raw X

- gtk+
- KDE (which uses QT which uses raw X)
- Rare direct calls to Xlib code
- other

More observations

- KDE / QT
- Crappyness is about on par with Xlib
- Trivial bugs!

More observations

- Several instances of this

```
Qkeymapper_x11.cpp
void QKeyMapperPrivate::clearMappings()
{
...
        uchar *data = 0;
        if (XGetWindowProperty(X11->display, RootWindow(X11->display, 0), ATOM(_XKB_RULES_NAMES), 0, 1024,
                               false, XA_STRING, &type, &format, &nitems, &bytesAfter, &data) == Success
            && type == XA_STRING && format == 8 && nitems > 2) {
...
            char *names[5] = { 0, 0, 0, 0, 0 };
            char *p = reinterpret_cast<char *>(data), *end = p + nitems;
            int i = 0;
            do {
                names[i++] = p;
                p += qstrlen(p) + 1;
            } while (p < end);
...
}
```

More observations

- In QT init code (affects all QT applications)

```
Qapplication_x11.cpp
void qt_init(QApplicationPrivate *priv, int,
                Display *display, Qt::HANDLE visual, Qt::HANDLE colormap)
{
...
        } else if (arg == "-name") {
            if (++i < argc)
                appName = argv[i]; ← if it was previously new'ed, it isn't anymore.
        }
...
}


void qt_cleanup()
{
...
    if (X11->foreignDisplay) {
        delete [] (char *)appName; ← could delete [] a pointer that isn't new'ed and possibly corrupt memory
        appName = 0;
    }
...
}
```

More observations

- So I reported these bugs
- They didn’t seem to think it was a security issues
- Quote from X developer (previous slide) is dead on
  - Suid remark
- QT does not seem to agree with this

More observations

- "KDE has precisely one setuid application, kcheckpass, for this reason. I suspect that someone running an suid Qt application would fall into a huge number of problems, the most obvious one being a malicious style that would allow them to trivially execute arbitrary code“

- Wait, did we just get a free Code exec bug from the QT security team ?

More observations

- I respond back, saying there are more KDE suid binaries, and specifically mention kppp, and question him on the styles thing

More observations

™> I am aware of this, regardless, this is library code, as such, chances are, there are suid applications out there that will use it.

That would be a security hole in those applications rather than in Qt, there are many ways that people can abuse a library to create unsafe applications.

> Do styles contain executable code ?

Yes."

More observations

™> also, does kppp no longer run suid ?

kppp should not be installed setuid. Here's a quote from its FAQ:

"There is no need for the setuid bit, if you know a bit of UNIX® systems administration. Simply create a modem group, add all users that you want to give access to the modem to that group and make the modem device read/writable for that group."

I doubt any modern distro would install it suid, in fact most are extremely careful about what they allow to be suid and are actively working to minimise what is."

More observations

- That kppp FAQ quote in incomplete, it goes on to say:

™ The KPPP team has lately done a lot of work to make KPPP setuid-safe. But it's up to you to decide if you install and how you install it."

More observations

- In fact, distro's do still have it suid.
- E.g. Ubuntu
- This is library code! They should not set policy for the apps that use them.
- They're sitting on the fence, because it's easy
  - you don't actually have to do anything
- Either defend it, and shut up
- Or do a suid check and exit()

More observations

- None of those bugs are fixed
- Got the ok from QT security team to disclose:

™> Ok, since you guys don't consider this a security issue, you're ok with me talking about this publicly?

Yes, that's fine"

More observations

- So loaders have LD_PRELOAD
- And has been made setuid safe years ago
- KDE/QT
  - QT_PLUGIN_PATH
- Gnome
  - GTK_MODULES
- Neither are setuid safe !

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

More observations

- The GTK+ people seem to be doing somewhat better.
- They do not allow suid GTK+ applications.
- And clearly explain why on their webpage

More observations (http://www.gtk.org/setuid.html)

The GTK+ Project

The GTK+ Project

About Features Download Screenshots Documentation Development

Why GTK_MODULES is not a security hole

GTK+ supports the environment variable `GTK_MODULES` which specifies arbitrary dynamic modules to be loaded and executed when GTK+ is initialized. It is somewhat similar to the `LD_PRELOAD` environment variable. However, this (and similar functionality such as specifying theme engines) is not disabled when running setuid or setgid. Is this a security hole? No. Writing setuid and setgid programs using GTK+ is bad idea and will never be supported by the GTK+ team.

You should not write setuid GTK+ programs because:

- GTK+ is too big. GTK+-1.2 and its dependent libraries (ignoring Xlib) total over 200,000 lines of code. For GTK+-2.0 (ignoring Xlib and image loading libraries), this figure will be around 500,000 lines of code.
- GTK+ is too complex. GTK+ takes input from dozens of sources, from drag-and-drop, to root-window properties, to keyboard input, to configuration files. This is a much broader scope for compromises than a typical server and makes auditing GTK+ especially tricky.
- Security of GTK+ requires the security of Xlib. The GTK+ team is not prepared to make that guarantee. Security bugs have been found in the recent past in such areas of Xlib as the input method code.
- You should not make your GUI setuid at all. Why run the risk of security bugs in code that does not need to be running with elevated privileges?

In the opinion of the GTK+ team, the only correct way to write a setuid program with a graphical user interface is to have a setuid backend that communicates with the non-setuid graphical user interface via a mechanism such as a pipe and that considers the input it receives to be untrusted.

For this reason, no effort is made in GTK+ to disable the obvious ways that you could compromise a setuid GTK+ program - GTK_MODULES and the ability for the user to specify theme engines, because we consider this to be only papering over the fundamental problems of writing setuid programs with *any* GUI toolkit. GTK+ may be modified in the future to simply refuse to run with elevated privileges, though it does not do this currently.

Does this mean that there are no security considerations for GTK+? No. In particular image loaders have been and will continue to be an area of special care, since users may load images from untrusted sources. And in addition to the possibility of this variety of exploit, most potential security holes are essentially bugs and even as mere bugs, must be squashed. To help accomplish this goal, GTK+ extensively uses high-level data structure abstractions which minimize the risk of most traditional buffer overflows.

However, the secure setuid program is a 500 line program that does only what it needs to, rather than a 500,000 line library whose essential task is user interfaces.

More observations

- This is beautiful, well though out and sane!
- “Security of GTK+ requires the security of Xlib. The GTK+ team is not prepared to make that guarantee”
- Or is it ?

More observations

Gtkmain.c

```
/* This checks to see if the process is running suid or sgid
 * at the current time. If so, we don't allow GTK+ to be initialized.
 * This is meant to be a mild check - we only error out if we
 * can prove the programmer is doing something wrong, not if
 * they could be doing something wrong. For this reason, we
 * don't use issetugid() on BSD or prctl (PR_GET_DUMPABLE).
 */
static gboolean
check_setugid (void)
```

More observations

- What does that mean ?
- Suid binaries can use GTK+, ***BUT*** …
- … they must acquire the privileged resources they want first
- And then drop privileges
- After that it’s ok to use GTK+
- Want to have their cake and eat it too
- Check should be stronger!

More observations

- games are a great example
- They are suid
  - Share a highscore database
- Once aquired, privs are dropped
- Only thing an attacker would have access to is that db
  - assuming a bug was found and exploited
- That db is considered trusted.
  - Any security bug in db parsing allows for further escalation
- Any user now playing any of those games gets pwned

More observations

- Spend a little bit of time looking at x display managers
- There's a lot of them
- Uses xdmcp protocol (goes over udp)
- Most have dependency on libxdmcp for this
- Libxdmcp's api's quite easily lend themselves to abuse
  - Leaves a lot of stuff uninitialized on failure
- This is ~~being~~ fixed

More observations

- LightDM
  - Used by ubuntu
- Has so called greeters that allow you to customize the gui
- Unpriv’ed greeters talk to LightDM
  - Using a pipe
- Parser for that pipe wasn’t great
  - Not that bad either
- Bugs are ~~being~~ fixed

```
static gboolean
read_cb (GIOChannel *source, GIOCondition condition, gpointer data)
{
    Greeter *greeter = data;
    gsize n_to_read, n_read, offset;
...
    n_to_read = HEADER_SIZE;
    if (greeter->priv->n_read >= HEADER_SIZE)
    {
        offset = int_length ();
        n_to_read += read_int (greeter, &offset); [2]
    }
    status = g_io_channel_read_chars (greeter->priv->from_greeter_channel,
                                      (gchar *) greeter->priv->read_buffer + greeter->priv->n_read,
                                      n_to_read - greeter->priv->n_read, [3]
                                      &n_read,
                                      &error);
...
    greeter->priv->n_read += n_read;
    if (greeter->priv->n_read != n_to_read)
        return TRUE;
    /* If have header, rerun for content */
    if (greeter->priv->n_read == HEADER_SIZE)
    {
        gsize offset = int_length ();
        n_to_read = read_int (greeter, &offset);
        if (n_to_read > 0)
        {
            greeter->priv->read_buffer = secure_realloc (greeter, greeter->priv->read_buffer, HEADER_SIZE + n_to_read); [1]
            read_cb (source, condition, greeter);
            return TRUE;
        }
    }
...
}
```

More observations

1. integer overflow when allocating buffer
2. Integer overflow (really small n_to_read)
3. Integer underflow, size to read becomes really large

More observations

- As mentioned earlier, libraries build on top of X use Xlib
- Apps will sometimes also call some X api's to query certain things
  - Using the XGetWindowProperty() api or any number of api's build on top of it (e.g. XGetClassHint(), XGetRGBColormaps(), …)
- Looked at the use of Xlib api's
- This too wasn't great

More observations

- By far the most common bug when using Xlib

```
void fn()
{
...
    SomeFormat *sf;
...
        (void) XGetWindowProperty(dpy, w, property, 0L,
                                  10000000, False, SomePropertyType, &type, &format,
                                  &length, &bytesafter, (unsigned char **) &sf);
...
        XFree((char*)sf);
...
}
```

- Check return values !
- XLib defense in depth fixes. Now guarantees NULL init of arguments on failure.

More observations

- Developers using Xlib don’t seem to realize that most of the api’s they use parse potentially untrusted network data
  - XReply
  - _XReply
  - _XRead32
  - _XRead
  - _XGetAsyncReply
  - XGetWindowProperty
  - XNextEvent
  - XPeekEventXIfEvent
  - XCheckIfEvent
  - XPeekIfEvent
  - XCheckTypedWindowEvent
  - XSetErrorHandler
  - XQueryFont

More observations

- derived from XGetWindowProperty:

- XFetchName
- XGetIconName
- XGetSizeHints
- XGetWMHints
- XGetWMSizeHints
- XGetIconSizes
- XGetTransientForHint
- XGetClassHint
- XGetRGBColormaps
- XGetTextProperty
- XGetWMName
- XGetWMIconName
- XGetWMClientMachine
- XGetCommand
- XGetWMColormapWindows
- XGetWMProtocols

- XScreenResourceString
- XFetchBuffer
- XFetchBytes
- XkbRF_GetNamesProp

More observations

- Conceptually there's a couple of X suid apps around that you'll see:
  - Config tools (e.g. kppp)
  - Games (e.g. swell foop)
  - Screen locking utils (e.g. Xlock, Xlockmore, Xscreensaver, …)
- Virtually all of these apps do drop privileges

More observations

- the screenlocking utils
- Only seem to capture your hashed pw entry (and optionally root).
  - Getspnam()
- W00t. That's not much of a resource
- Or is it ?

More observations

- The linux shadow library is responsible for api's for reading from and writing to the shadow file
- Shadow.h
  - The code uses FILE stream api's to read and write
    - Uses heap buffers internally, Can't clear memory.
  - Stores read data in local stack buffers
    - Doesn't clear memory
  - Basically leaks the entire shadow file onto the heap and the stack

More problems summary

- Xlib in suids is a bad idea
- GTK+ kinda sorta still allowed in suids
- Very common sloppy misuse of Xlib api's
- Linux shadow library handles sensitive data in a sloppy manner

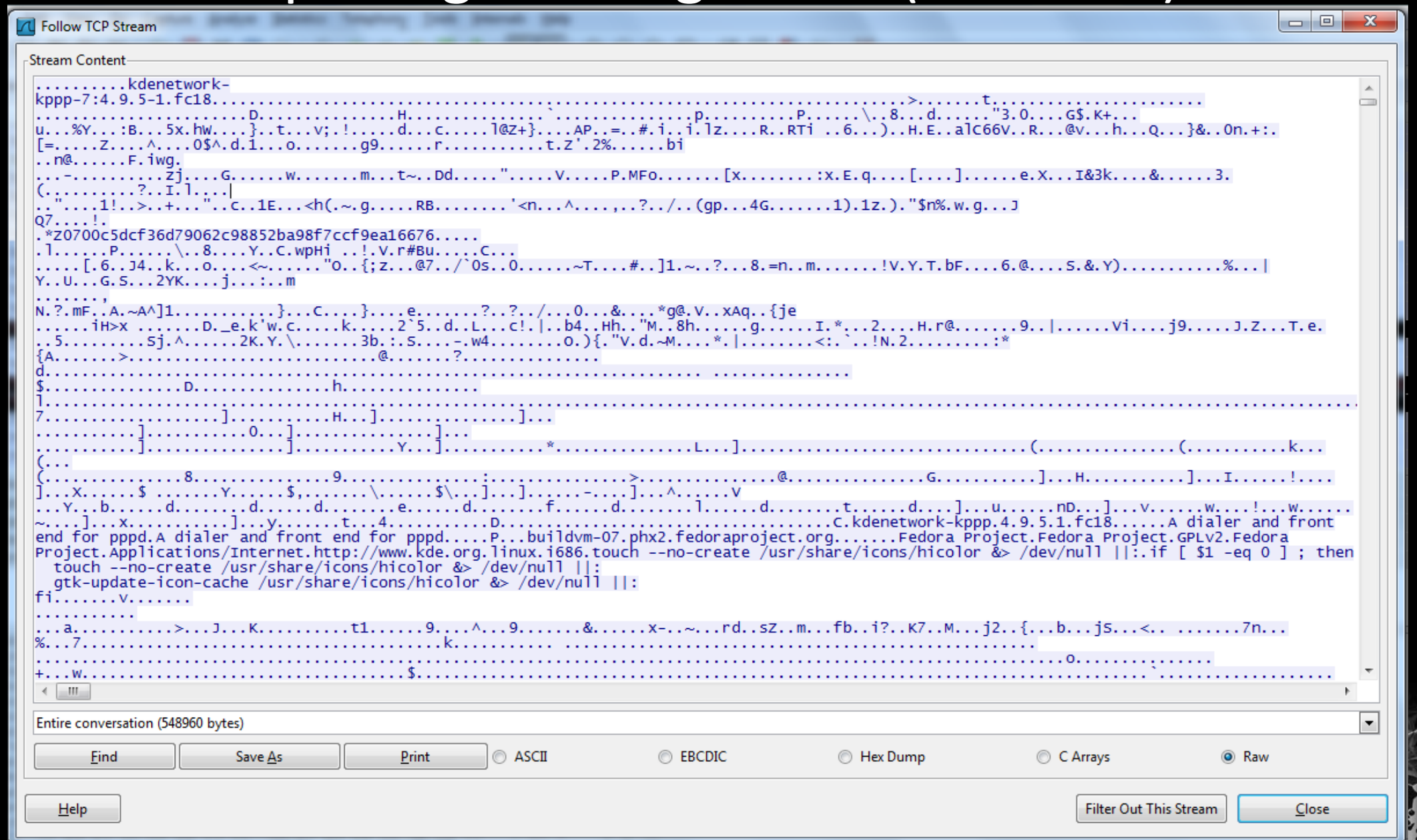
todo

- There’s some other things we wanted to look at but didn’t get around too
- Package managers
- Clipboard

- Ok, we did look for about 5 seconds
- It did look bad ….

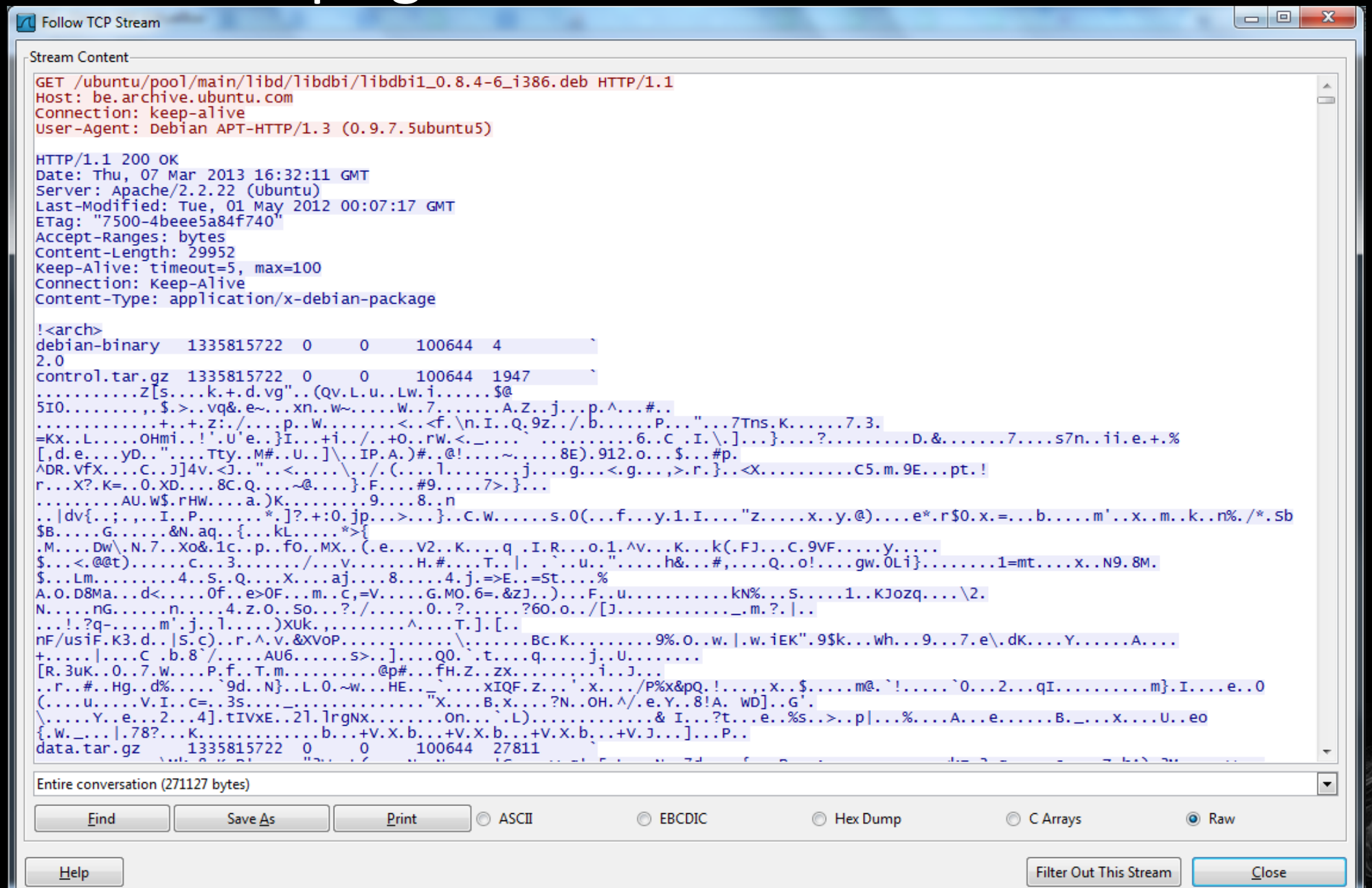
todo

- Fedora package management (software)



todo

- Ubuntu apt-get



todo

- OpenSuse zypper

Follow TCP Stream

Stream Content

```
GET /distribution/12.2/repo/oss/suse/i586/rrdtool-1.4.7-4.1.2.i586.rpm HTTP/1.1
User-Agent: ZYpp 11.6.2 (curl 7.25.0) openSUSE-12.2-i586
Host: download.opensuse.org
X-ZYpp-AnonymousId: 222b313d-28bf-4d67-a6ea-fcbefb1eebc1
X-ZYpp-DistributionFlavor: dvd
Accept: */*, application/metalink+xml, application/metalink4+xml

HTTP/1.1 200 OK
Date: Fri, 08 Mar 2013 00:09:00 GMT
Server: Apache/2.2.12 (Linux/SUSE)
X-Prefix: 208.181.206.0/23
X-AS: 852
Vary: accept
Content-Disposition: attachment; filename="rrdtool-1.4.7-4.1.2.i586.rpm.meta4"
Transfer-Encoding: chunked
Content-Type: application/metalink4+xml; charset=UTF-8

1f89
<?xml version="1.0" encoding="UTF-8"?>
<metalink xmlns="urn:ietf:params:xml:ns:metalink">
  <generator>MirrorBrain/2.15.0</generator>
  <origin dynamic="true">http://download.opensuse.org/distribution/12.2/repo/oss/suse/i586/rrdtool-1.4.7-4.1.2.i586.rpm.meta4</origin>
  <published>2013-03-08T00:09:00Z</published>
  <publisher>
    <name>openSUSE</name>
    <url>http://download.opensuse.org</url>
  </publisher>

  <file name="rrdtool-1.4.7-4.1.2.i586.rpm">
    <size>616657</size>
```

Entire conversation (16111 bytes)

Find | Save As | Print | ASCII | EBCDIC | Hex Dump | C Arrays | Raw

Help | Filter Out This Stream | Close

todo

- Lets hope they sign stuff
- And check it
  - And have signatures locally
  - Or fetch them securely from a remote host
- And don’t use md5 …
- Even if you’ve got all this correct
- Network protocol is unencrypted
- Adds a lot of remote attack surface

todo

- Clipboard
- ICCCM
- It smells rotten
- http://lists.slug.org.au/archives/slug-chat/2001/July/msg00054.html
  - Srsly, go read it
  - No, really!
  - It’s all I know about ICCCM, but it speaks volumes

todo

™d00d, that document is devilspawn . what sick evil twisted mind wrote this damn spec?"

™The ICCCM is the coding equivalent of the Medieval rack, except its advertised as some kind of X11 swingers party."

™I've seen more elegant protocols in unlikely places. When blowflies fight over a pile of elephant shit, their pecking order is a more elegant protocol than ICCCM."

™I. C. C. C. M.

Inter-
Client
Communications
Conventions
Manual!

Manual, like in "manual labour", like in "pain"
Conventions, like in "not required, just do ALL OF IT or you SUCK!"
Communications, like in "fucking overengineered carrier pigeons"
Client, like in "see that guy with the limp, he was one of my "clients""
Inter-, like in "Inter-nal bleeding™"

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Solutions ?

- The shadow library thing is easy to fix
- It’s not really a bug in the first place
  - But exposes too much sensitive information to an already compromised suid program
- Drop all FILE stream usage.
  - Use open/read/write syscalls instead
- Clear all memory after use
  - Make sure memset() doesn’t get optimized out by compiler

Solutions ?

- Most suids on linux (and most unices) have been dropping privileges for a long time
  - Nothing has changed since
- This isn’t good enough.
- Those privileged resources include:
  - read fd to /dev/kmem
  - read/write fd to /etc/resolv.conf
  - Full content of /etc/shadow
  - …

Solutions ?

- The suid processes still have their suid bit set in kernel
- Attacker still needs some kind of bug + exploit
- reduced what can be gained from uid 0 to those resources
- Is however still very significant

Solutions ?

- A model where priv dropping and priv separation is combined would make more sense
- Would add more defense in depth
- Probably not that hard to implement for some suids

Solutions ?

- Here’s what it would looks like:
  - pipe
  - fork
  - client drops all privs
  - server gets resources
  - server drops privs
  - very small and well defined interface between client and server

Solutions ?

- Client retains it's suid bit
  - Pipe to server is protected from injection
- readelf –d on some of the suids
  - HUGE list of library dependencies!
- We don't want that in service code.
- fork() is out!
  - If you fork, all that stuff is still in memory.

Solutions ?

- fork() is out, fork() + execve() is in.
- Pass fd to excve’ed process.
- Server has to be a very small piece of c code.
- Only access to libc.

Solutions ?

- Actually, glibc (default on most distributions) is super bloated.
- > 100mb of source code (2.1.7)
  - memfrob() ?
  - strfry() ???
- Can you really trust that ?
- Should not be used in server app
- Instead use something like dietlibc, uClibc, klibc or musl

Solutions ?

- One last piece of code bloat left
- Dynamic loader.
- Takes input though environment variables
- Have been bugs in there in the past
- Do you really want to trust it ?
- Fix: static binaries

Conclusion?

- Guess there should be a conclusion
- Run for the hills ?
- Things could be better …
- … by several orders of magnitude

- There’s really a lot of work to be done here
  - most of that code is not written with a trust boundary in mind

IOActive™
COMPREHENSIVE COMPUTER SECURITY SERVICES

Questions ?