USB Autorun attacks against Linux

ShmooCon 2011 – Washington, DC

Jon Larimer
IBM X-Force Advanced R&D

jlarimer@us.ibm.com
jlarimer@gmail.com

Autorun malware

- On Windows, the **`autorun.inf`** file could be used to automatically launch programs when a CD/Floppy/USB driver was inserted
- Many people disable this feature now!
- Windows 7 changed how **AutoRun** works from USB drives
- Malware and hackers could still take advantage of vulnerabilities...
  - LNK vulnerability (**Stuxnet**)
  - PDF previews
  - Embedded BMP thumbnail vulnerability

Autorun malware on Linux?

- Some desktop environments support autorun scripts on external media
- freedesktop.org specifications allow: **`.autorun`**, **`autorun`**, **`autorun.sh`**
- Specs specifically forbid running these scripts automatically – the desktop environment should always ask
- What about taking advantage of security vulnerabilities?

Autorun vulnerabilities

- Lots of code executes when a new mass storage device is connected...
- Removable storage subsystem drivers
  - –USB, eSATA, FireWire, PCMCIA
- File system drivers
  - –Kernel drivers: **ext3**, **ext4**, etc
  - –User mode (FUSE) drivers: **ntfs-3g**
- Desktop applications
  - –GNOME desktop thumbnailers
- Vulnerabilities could exist at any of these layers!

Attacks on physical systems

- Physical access is 'game over'
- What about full disk encryption?
- IEEE 1394 (FireWire) DMA physical memory access
  - Requires FireWire port and drivers
- Cold boot attack
  - Requires being able to boot from external media, stealing the RAM, or swapping out an internal drive
- Removable storage attacks!
  - Most desktop OS's will automatically mount file systems on USB
  - Physical access not really necessary, just find someone to plug a device into their PC
  - If an exploit runs while the PC is already booted and the user is logged on, full disk encryption could be defeated

USB on Linux

- **usbcore** in **`drivers/usb/core`**
- Host controller driver framework is **`drivers/usb/core/hdc.c`**
  - –UHCI: **`drivers/usb/host/usb-uhci.c`**
  - –EHCI: **`drivers/usb/host/usb-ehci.c`**
- Hub driver in **`drivers/usb/core/hub.c`**
- Interface drivers register by calling **`usb_register()`** or **`usb_register_driver()`**, specifying which vendor/product IDs they work with
- **`drivers/core/usb/driver.c usb_match_id()`** takes care of the matching, then the driver is loaded

USB Vulnerabilities on Linux

- MWR InfoSecurity Auerswald Linux USB driver bug, 2009
  - Driver provides support for Auerswald USB ISDN devices
  - Had a problem handing USB descriptors, resulting in a buffer overflow
- Fuzzing USB drivers
  - Mortiz Jodiet – hardware+software (2009)
  - Tobias Mueller – QEMU-based fuzzer (2010)
- Vulnerabilities in USB drivers can be exploited with cheap, small, off-the-shelf programmable USB development boards

USB mass storage on Linux

- Storage class driver in `drivers/usb/storage/usb.c`
- `storage_probe()`
  - Sets up a **SCSI** host structure
  - adds **SCSI** host to **SCSI** subsystem
  - `scsiglue.c` and `protocol.c` take care of converting **SRBs** to **URBs** for the USB drivers
- **SCSI** subsystem adds a block device (`/dev/sdb`)
- **udev** is notified

udev, udisks, and D-Bus

- **udev**
  - –device manager for Linux
  - –adds/remove entries in **`/dev`**
  - –can trigger events based on rules or through a netlink socket
- **D-Bus**
  - –IPC mechanism
  - –allows applications to register for system device events
- **udisks**
  - –provides a **D-Bus** interface for dealing with disk devices
  - –uses **GUdev** library (part of **udev**) to subscribe to **udev** events through a **netlink** socket, republishes them through **D-Bus**

File systems in Linux

- Traditionally lived in **`fs/`** branch of kernel source tree
- File systems operate between low level disk bus drivers and virtual file system
- **FUSE** – file system in userspace
- **GVFS** – GNOME Virtual File System
  - –not a traditional file system
  - –can only be access through **GVFS**, **GIO**, or the **`~/.gvfs`** **FUSE** mountpoint
  - –Can access also access files through SMB, FTP, DAV, etc

File system driver vulnerabilities

- Vulnerabilities in FS drivers could be exploited by malformed FS images on a USB drive
- Successfully exploited vulnerabilities result in root access – file system drivers run in kernel mode
- User mode file system driver exploits run in the context of whoever mounted the volume
- Would be considered a 'local' kernel-mode bug because it requires physical access
- For the purpose of exploitation, it can be considered remote since you don't already have access to the OS

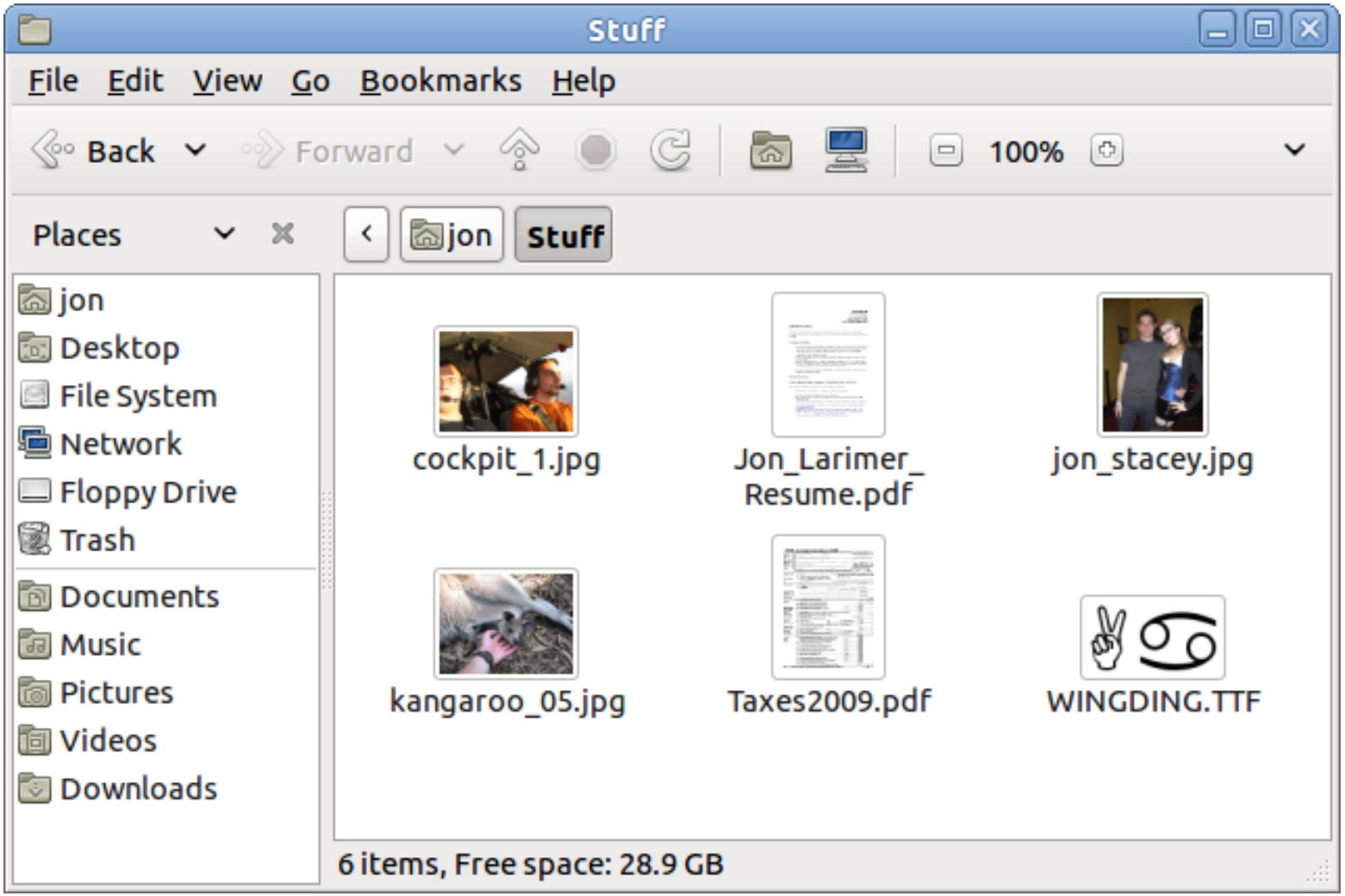
Finding file system driver vulns

- Manually auditing the code (read *The Art of Software Security Assessment* by Dowd/McDonald/Schuh)
  - Concentrate on how structures on disk sectors are parsed
- Static code analysis (lint, clang static analyzer, etc)
- Fuzzing
  - In Linux, any block device (including a file) can be mounted as a volume
  - Write code to modify a FS image, mount, perform various operations, then unmount
  - Fuzz smarter by understanding FS structure, use code coverage/taint analysis tools

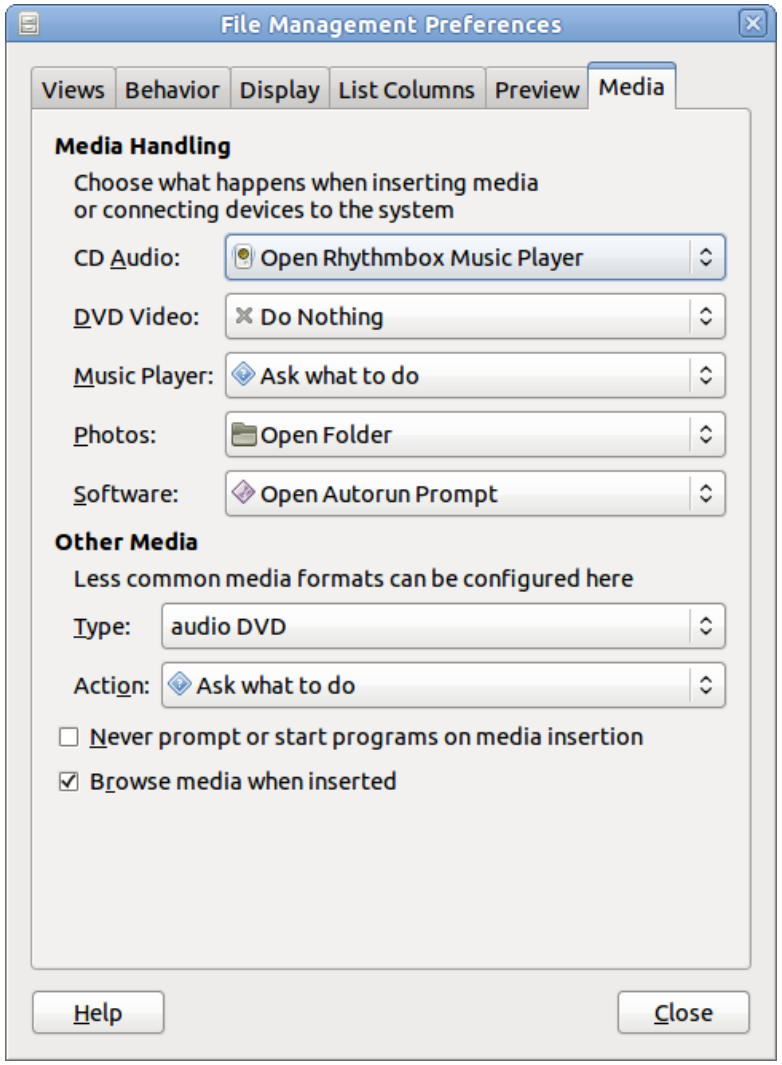
GNOME Nautilus

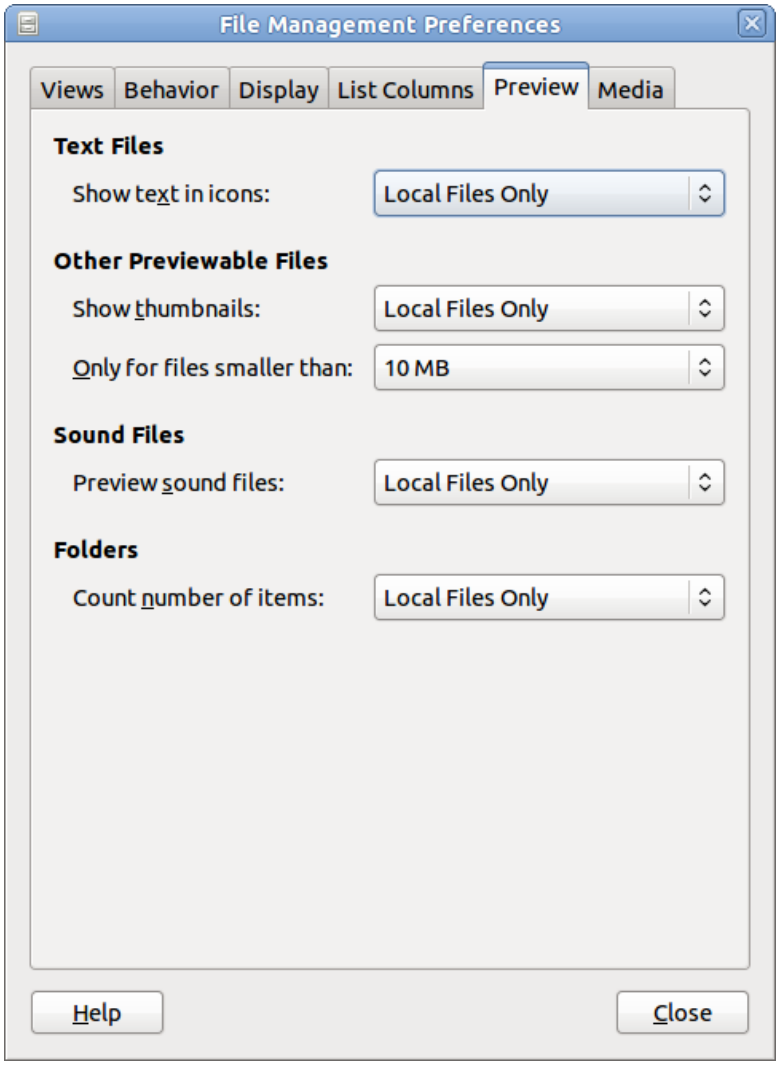
- GNOME Nautilus is the file browser used by Ubuntu Desktop Linux10.10
- Supports most of the freedesktop.org specifications
- Will automatically mount known file systems on USB drives by default
  - –Volume mounted in **`/media/XXX`**, where **`XXX`** is the volume name
- Will automatically open a browsing window when a new file system is mounted
- File browsing window will generate thumbnails for all files in the root directory of the device
- It does this even with the screensaver running and locked!

GNOME Nautilus file browser



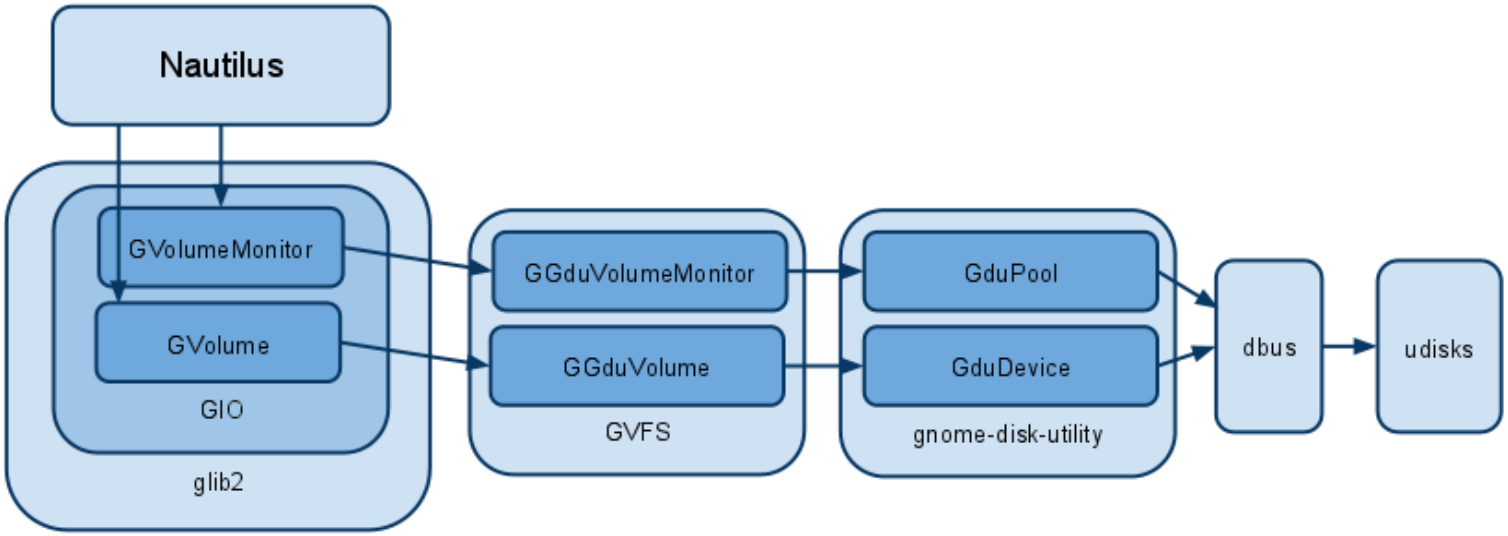
Nautilus thumbnail/media settings





GNOME Nautilus operation

- Detects when new storage devices are connected
- Uses **GVFS** to access browse file systems over SMB, FTP, DAV, etc
- Uses **GVFS** to be notified of newly mounted file systems



GNOME Nautilus – thumbnailers

- Nautilus will generate thumbnail images for images, movies, documents, and other file types to use as icons in the file browser
- Settings stored in **gconf** system
- Image icons generated internally using **GdkPixBuf**
- Also allows 3rd party icon handlers
  - **evince-thumbnailer**: document files
  - **totem-video-thumbnailer**: video and audio files
  - **gnome-thumbnail-font**: font files
- Thumbnails cached in **`~/.thumbnails/normal`**

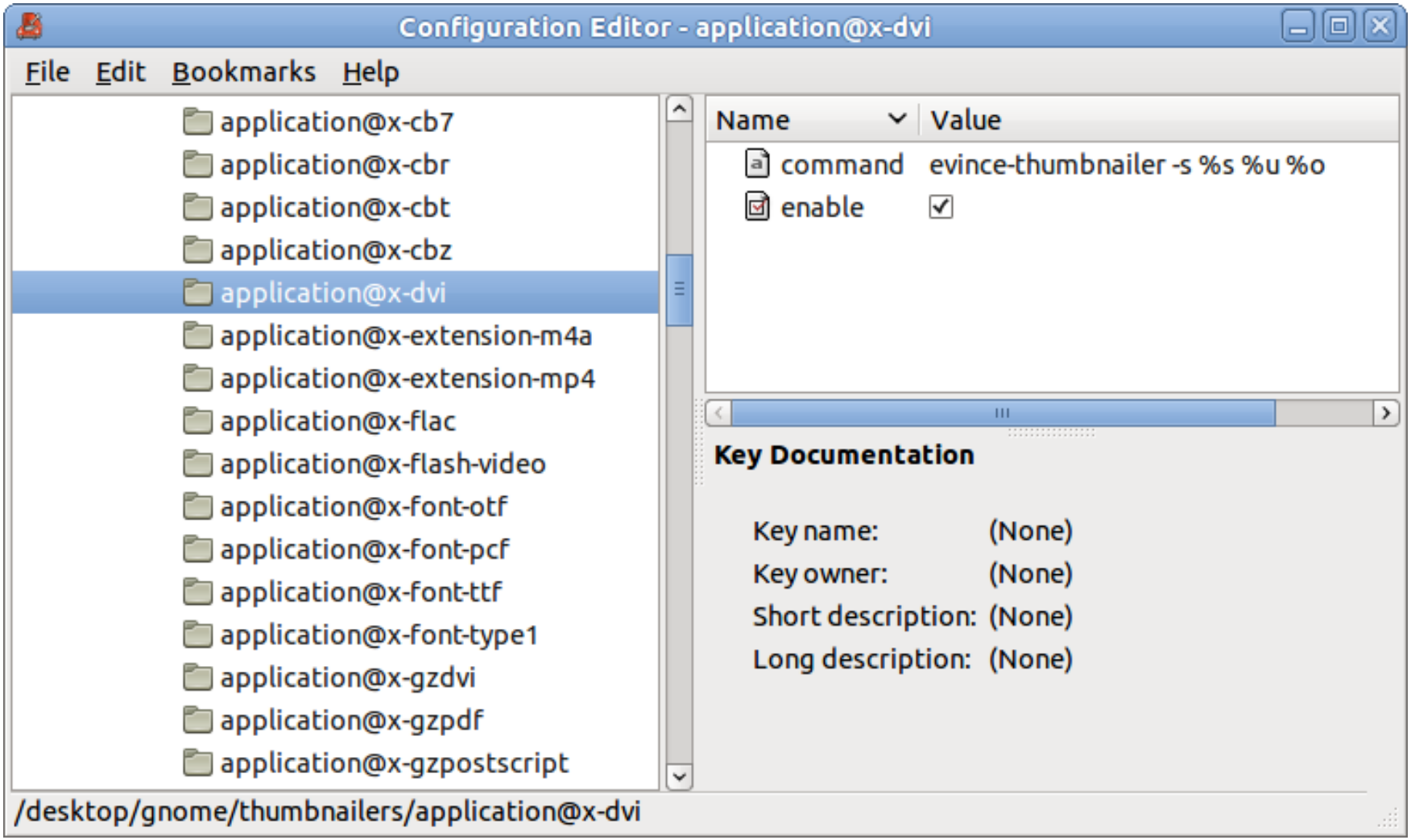
GdkPixBuf thumbnails

- Relies on some 3rd party libraries for some image formats (**libpng**, **libtiff**, **libjpeg**)
- All 3 of those libraries have had security vulnerabilities before...
- Contains built-in code for other formats (**bmp**, **gif**, **ico**, **tga**, **xpm**, and others)
- Full list of supported extensions (in Ubuntu 10.10):
  - wmf, apm, ani, bmp, gif, icns, ico, cur, jp2, jpc, jpx, j2k, jpf, jpeg, jpe, jpg, pcx, png, pnm, pbm, pgm, ppm, qtif, qif, ras, svg, tga, targa, tiff, tif, wbmp, xbm

Exploiting bugs in GdkPixBuf

- Thumbnailing happens in the Nautilus process
- If the process crashes, the file browsing window goes away
- Difficult to defeat **NX** and **ASLR** (can't brute force against **ASLR**...)
- Nautilus isn't protected by **AppArmor**

Nautilus external thumbnail settings



External thumbnailers

- Configured with gconf:
  - `gconftool -R /desktop/gnome/thumbnailers`
- Example:
  - `/usr/bin/totem-video-thumbnailer -s %s %u %o`
    - `%s` = size
    - `%u` = input file
    - `%o` = output file
- Nautilus looks up thumbnailer application for each file based on the MIME type
- Separate process is launched for each file that gets thumbnailed

evince-thumbnailer

- Part of GNOME evince, the document reader
- Supports file types:
  - pdf, djvu, djv, pdf.bz2, cbr, cbz, cbt, dvi, pdf.gz, ps.bz2, ps, ps.gz, eps.bz2, epsi.bz2, epsf.bz2, eps, epsi, epsf, dvi.gz, dvi.bz2, eps.gz, epsi.gz, epsf.gz, cb7
- Renders the first page of a document to use as the icon
- Relies on 3rd party libraries for some formats, internal code for others
- Protected with **PIE**, **AppArmor**

totem-video-thumbnailer

- totem-video-thumbnailer thumbnails these extensions:
  - anim[1-9j], mp4, m4v, m2t, m2ts, ts, mts, cpi, clpi, mpl, mpls, bdm, bdmv, asf, ogx, shn, mxf, gvp, avi, divx, qt, mov, moov, qtvr, wmv, webm, wmx, ra, rm, ram, ogv, ram, mpeg, mpg, mp2, mpe, vob, dv, mkv, wpl, fli, rm, rmj, rmm, rms, rmx, rmvb, wvx, rv, rvx, rp, flv, pict, pict1, pict2, nsc, fli, flc, wm, sdp, qtl, 3gp, 3g2, 3gpp, 3ga, nsv, viv, vivo
- Relies on many 3rd party libraries
- Not protected by **PIE** or **AppArmor**!

gnome-thumbnail-font

- Provides thumbnails for:
  - –ttf, ttc, otf, pfa, pfb, gsf, pcf, pcf.Z, pcf.gz
- Uses the **FreeType** library for rendering fonts
- There have been vulnerabilities in **FreeType** reported in the past
- Not protected by **PIE** or **AppArmor**!

What about mitigations?

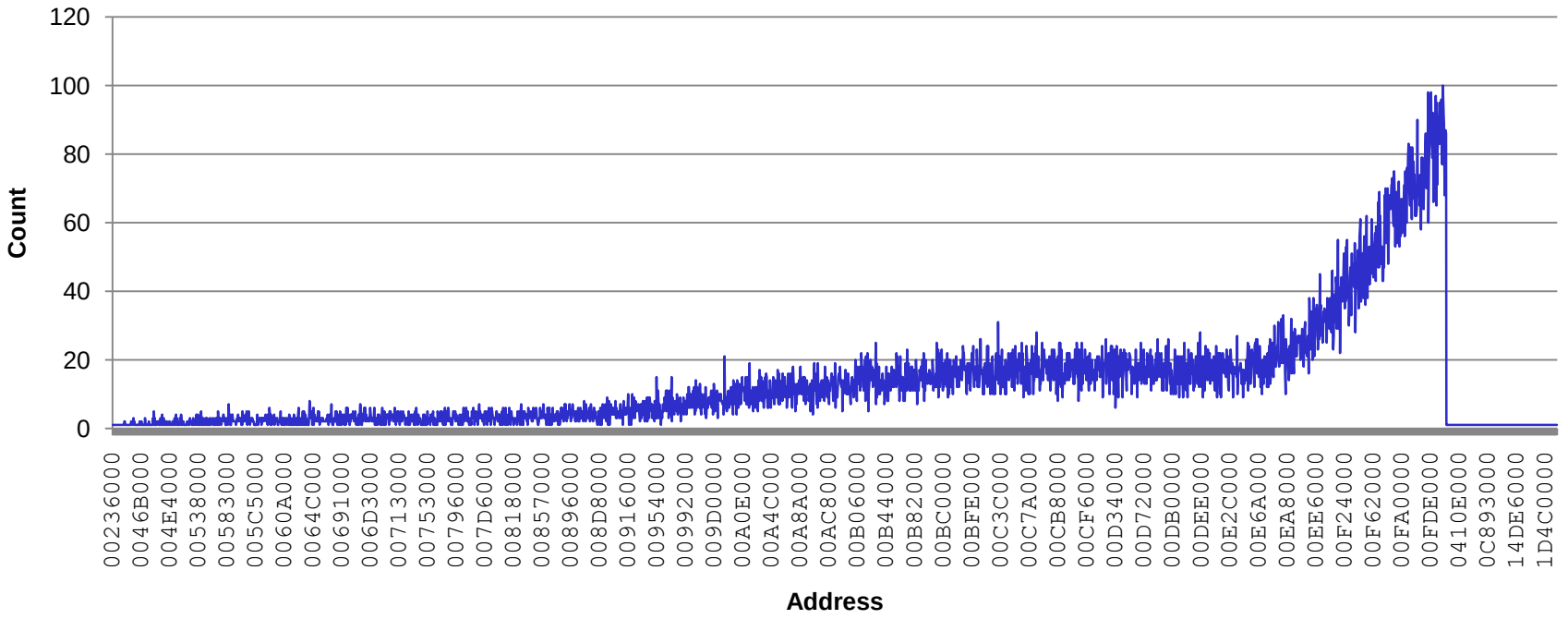
- Ubuntu 10.10 has many security features and exploit mitigations in place by default
- **NX** (non-executable memory), **ASLR** (address space layout randomization) on everything, **PIE** (position independent executable) on some files
- **AppArmor** – kind of a firewall for system calls
- **NX** is not very effective, attackers can use **ret2libc** or return-oriented-programming (**ROP**) exploitation techniques
- **ASLR** mitigates **ROP**...
- **AppArmor** can be tough to defeat, but there are weaknesses
  - –Protection is defined by per-application profiles
  - –There are some things **AppArmor** can't protect against

Defeating ASLR/PIE

- **ASLR** can be brute-forced
- External thumbnailers are launched as a separate process for each file
  - –If the process crashes, the other files will continue to be thumbnailed
- Figure out which addresses **`libc`** (or other target library) can be loaded at
- There are only around ~3000 addresses that will be used with the standard Linux kernel **ASLR** implementation on 32 bit systems
- Sometimes less are needed...

ASLR Weaknesses?

Base address of `libc` per 40960 runs of `evince-thumbnailer`



Defeating AppArmor

- **AppArmor** is only as strong as the application's profile
  - **evince-thumbnailer**'s profile used to allow writing to `~/.config/autostart`
  - Could be used to install malware to start when the user logs in
  - See https://code.launchpad.net/bugs/698194
- There are some things that **AppArmor** can't protect against
  - X11 library calls (could block network access in some cases)
  - Kill screensaver, sniff keystrokes, inject keystrokes, etc

Autorun exploit payloads?

- Copy files from the user's home directory – useful for defeating full disk encryption, **TrueCrypt**, etc
  - –Get browser cookies, documents
- Kill screensaver process
  - –`killall gnome-screensaver`
  - –Gives you full access to the user's desktop
- Install a backdoor
  - –Add script to `~/.bash_profile`, `~/.profile`
  - –Add desktop file to `~/.config/autostart`
- Elevate privileges
  - –Install rootkit

Killing the screensaver

- The screen saver is just a process that runs as a window at the top of the X11 window tree
- **`killall gnome-screensaver`** will kill the process, bypassing the authentication dialog
- Alternatively, locate the X11 library in the process and use that to kill the screen saver window:
  - **`XOpenDisplay(":0.0")`**
  - **`XQueryTree()`** to enumerate windows
  - **`XFetchName()`** to look for "gnome-screensaver"
  - **`XKillClient()`** to terminate the process

Install a backdoor?

- Can be done without root
- **`~/.config/autostart`** is analogous to the Windows "Startup" folder
  - Drop a shortcut there (**`.desktop`** file) that points to a script and it'll be launched whenever the user logs in
  - **`.desktop`** file format is part of freedesktop.org specifications
- Or use **`.profile`**, **`.bash_profile`**...
- Script could download remote access trojan from the web, or just copy one from the USB drive

evince vulnerabilities

- Vulnerabilities in handling external font files for **DVI** documents (CVE-2010-2640, CVE-2010-2641, CVE-2010-2642, CVE-2010-2643)
- http://www.ubuntu.com/usn/usn-1035-1
- **DVI** files are generated from **LaTeX** documents
- **DVI** files can reference external fonts that get loaded when the **DVI** file is processed
- External fonts can be specified with an absolute path (**`/media/XXX`**)
- Easier to exploit from USB than any remote vector...

CVE-2010-2640 – PK font parsing

backend/dvi/mdvi-lib/pk.c

```
424                             int     pl;
425                             int     cc;
426                             int     w, h;
427                             int     x, y;
428                             int     offset;
429                             long    tfm;
430
431                             switch(flag_byte & 0x7) {
432                             case 7:
433                                     pl = fuget4(p);
434                                     cc = fuget4(p);
435                                     offset = ftell(p) + pl;
436                                     tfm = fuget4(p);
437                                     fsget4(p); /* skip dx */
438                                     fsget4(p); /* skip dy */
439                                     w  = fuget4(p);
440                                     h  = fuget4(p);
441                                     x  = fsget4(p);
442                                     y  = fsget4(p);
443                                     break;
```

CVE-2010-2640

backend/dvi/mdvi-lib/pk.c

```
 483                             font->chars[cc].code = cc;
 484                             font->chars[cc].flags = flag_byte;
 485                             font->chars[cc].offset = ftell(p);
 486                             font->chars[cc].width = w;
 487                             font->chars[cc].height = h;
 488                             font->chars[cc].glyph.data = NULL;
 489                             font->chars[cc].x = x;
 490                             font->chars[cc].y = y;
 491                             font->chars[cc].glyph.x = x;
 492                             font->chars[cc].glyph.y = y;
 493                             font->chars[cc].glyph.w = w;
 494                             font->chars[cc].glyph.h = h;
 495                             font->chars[cc].grey.data = NULL;
 496                             font->chars[cc].shrunk.data = NULL;
 497                             font->chars[cc].tfmwidth = TFMSCALE(z,
tfm, alpha, beta);
 498                             font->chars[cc].loaded = 0;
```

CVE-2010-2640

- So we can write an arbitrary value to a semi-arbitrary location in memory
- We don't know where the stack is, can't overwrite the return address
- The write is relative to the heap, so **ASLR** won't impact our ability to overwrite a function pointer on the heap

backend/dvi/mdvi-lib/fontsrch.c

```
 173          /*
 174           * If the font type registered a function to do the
lookup, use that.
 175           * Otherwise we use kpathsea.
 176           */
 177          if(ptr->info.lookup)
 178                  filename = ptr->info.lookup(name, h, v);
```

CVE-2010-2640

```
165 struct _DviFontInfo {
166         char    *name;  /* human-readable format identifying string */
167         int     scalable; /* does it support scaling natively? */
168         DviFontLoadFunc         load;
169         DviFontGetGlyphFunc     getglyph;
170         DviFontShrinkFunc       shrink0;
171         DviFontShrinkFunc       shrink1;
172         DviFontFreeFunc         freedata;
173         DviFontResetFunc        reset;
174         DviFontLookupFunc       lookup;
175         int                     kpse_type;
176         void *                  private;
177 };
178
179 struct _DviFontChar {
180         Uint32  offset;
181         Int16   code;           /* format-dependent, not used by MDVI */
182         Int16   width;
183         Int16   height;
184         Int16   x;
185         Int16   y;
186         Int32   tfmwidth;
187         Ushort  flags;
188 #ifdef __STRICT_ANSI__
189         Ushort  loaded;
190         Ushort  missing;
191 #else
192         Ushort  loaded : 1,
193                 missing : 1;
194 #endif
195         Ulong   fg;
196         Ulong   bg;
197         BITMAP  *glyph_data;
198         /* data for shrunk bitimaps */
199         DviGlyph glyph;
200         DviGlyph shrunk;
201         DviGlyph grey;
202 };
```

CVE-2010-2640

- We can overwrite **`ptr->info.lookup`** with the address of **`system()`** in **`libc`**
- **`name`** is a string representing the font file it's looking for
- To write this exploit:
  - –figure out what **`cc`** needs to be so that **`w`**, **`h`**, **`x`**, or **`y`** overwrites **`ptr->info.lookup`** for one of the fonts
  - –specify that **`cc`** value for the first font, and put the address in system in **`w`**, **`h`**, **`x`**, **`y`**
  - –for the 2nd font, specify the name to be **`/media/XXX/kill.sh`**, where ***`XXX`*** is volume name of USB device
  - –**`/media/XXX/kill.sh`** can be a shell script to do whatever you want – mine kills the screensaver

Problems...

- **AppArmor** will prevent loading a `.pk600` file, but creating a symlink from the `.pk600` file to a file ending in `.png` will get around this restriction
- **AppArmor** won't let you execute a process
- How do we get around this?
  - –Write a ROP 2nd stage shellcode loader
  - –`mmap/open/read`
  - –**AppArmor** won't let you map executable files, but you can create an anonymous W+X mapping
  - –2nd stage shellcode can search for **X11** library, use **X11** APIs to enumerate root windows then kill the one labeled "gnome-screensaver"
- Still working on it...

Demo!

D E M O

DEMO

DEMO

Conclusion

- It's possible for software vulnerabilities to be used for autorun attacks against Linux
- Not just GNOME, KDE has similar functionality
- Recommendations:
  - Disable auto-mounting and auto-browsing of removable storage and media
  - Disable thumbnailing of files
  - Make use of technologies like **AppArmor** to provide enhanced protection for desktop user interface processes
  - **PaX (grsecurity)** offer more bits of entropy for **ASLR** than the default Linux kernel and other security features
  - Use a 64 bit OS, which makes it even harder to brute force **ASLR**