Interoperability and code reuse

A Sticky Mess

Developing cross-platform apps can be difficult and error prone. We offer some tips to ease the work. *By Kurt Seifried*

Interoperability is a good thing; it enables us to use security tools across multiple platforms (like OpenID and OSSEC) and combine data from multiple platforms into tools like Prelude, right? Code reuse is also good – why reinvent the wheel if someone is giving away really nice all-weather tires? Unfortunately, when it comes to interoperability, not everyone does a good job of implementation. In fact, efforts to achieve compatibility and interoperability often can make a real security mess.

Embedded Libraries

I can't imagine the horror that developers face. For example, say you're building an app, and it needs to do some font rendering. No problem, right? You can just call a font library such as FreeType2 and use it to parse and output the data. Well … on Linux, this will probably work without too much trouble, because FreeType2 is a widely supported and used library. Thus, package maintainers will probably be able to create package dependencies so that the library is automatically installed when your package is, and life is good.

However, say you also want your software to install and work on Windows. Now, you are faced with a choice: How do you deal with the need for FreeType2? Do you document the dependency and simply hope users will figure out how to download and install FreeType2 on Windows? That's probably not your best choice.

Alternatively, you could find a copy of FreeType2 for Windows and bundle it with the Windows installer, but now you're installing two applications and having to maintain an internal build of FreeType2 (or at least verify that the build you're using works on Windows as expected).

Or, you can simply get a copy of the FreeType2 source code and include it as a component or module of your software. When you build the program, you can embed FreeType2 into it, and, presto, you have FreeType2 support in your application. Everything works, and you don't have to deal with external dependencies.

But, software isn't a static entity. During the last year, several security flaws have been found in FreeType2. Applications like Firefox that embed FreeType2 might not be affected by all the vulnerabilities, but it's probably affected by some of them. So, as a user or administrator, you can't simply upgrade FreeType2 on your systems and avoid FreeType2 issues.

Because Firefox uses an embedded copy of FreeType2, you'll need to install updated versions for Firefox once they provide an update with their internal version of FreeType2 that has been corrected. The good news, however, is that Firefox is a pretty responsive project and the developers fix security issues promptly. But, even a responsive project can makes mistakes or miss a security

KURT SEIFRIED

Kurt Seifried is an Information Security Consultant specializing in Linux and networks since 1996. He often wonders how it is that technology works on a large scale but often fails on a small scale.

update from the upstream software provider.

Find Embedded Libraries

So, how do you find embedded copies of libraries like FreeType2? The answer is: Not easily. Sometimes an embedded library will only require a single file, and if the name has changed, good luck finding it. You could manually grep files for version and identification strings, but this process is error prone and could miss something (what if the version information was removed?).

Package Clone Detection

Fortunately, Silvio Cesare decided to work on this problem [1]. Using a combination of file names and fuzzy file hashing (fuzzy hashing allows similar inputs to be identified), a large graph (nodes representing software are connected to other nodes where a match is indicated) is created. That graph can then be searched for cliques, which tend to indicate software packages that contain embedded copies of other software packages. The software is available under the GPL version 3 and works for RPM- and DEB-based distributions, giving pretty wide coverage. Once you've run the software and identified outstanding issues, how do you prevent the problems from occurring in the future?

The Debian project [2] has taken the stance that embedded libraries (sometimes called convenience copies) should not be used and that a separate package should be created if the library is needed but does not exist. A proposal from the Fedora project [3] has taken a slightly different view, acknowledging that embedded libraries might be needed in certain cases. In this case, a `provides` statement should be included in the spec file for the package, so that later embedded copies of libraries can be found using the command `rpm -q --provides <package name>`, which will provide output such as:

```
Provides: bundled(zlib) = 1.1.14
```

This can be grepped easily and used with automated tools.

Other Cross-Platform Problems

Some developers are not content simply to embed libraries. One example of how this can go horribly wrong is the Calibre e-book management software. In an effort to allow users simply to plug their USB-based e-books into their computers and have the filesystem mounted automatically, the author of Calibre wrote a "helper" program. To mount filesystems, of course, you need root privileges, so the helper program was setuid root. Unfortunately, half a dozen vulnerabilities were found in it [4], including the ability to execute any program as root (oops).

Additional vulnerabilities were found allowing any su'ers to mount arbitrary filesystems in arbitrary locations. For example, by mounting on top of `/etc` or `/usr`, an attacker could trivially compromise the system and take control of it. Fortunately, a resolution was found to this problem, and the helper program was removed (well, it was replaced by a stub program that simply exits). The lesson here is that a seemingly simple task can have extreme security consequences.

Other People's Code

Finally, I will mention one of the most serious examples of code reuse leading to security incidents. Last year, I wrote about plugin security [5], and one of the programs I specifically mentioned was WordPress. Unfortunately, what has come to pass is far worse than any worst-case scenario I imagined. WordPress plugins contain PHP code that is executed, but they are easy to identify and relatively easy to upgrade. WordPress themes can also contain PHP code and helper applications in the form of PHP scripts. One such application was the `TimThumb.php` program, which provided image resizing capabilities in WordPress blogs. This program was shipped with both traditional plugins and with a wide number of themes (Google indicates more than 350,000 instances of "timthumb.php" in URLs).

Unfortunately, the program contains a trivially exploitable flaw [6] that can be used to execute arbitrary PHP code on servers with `timthumb.php` installed. The attacker simply tells TimThumb to fetch a remote URL, which it does and then executes. As far as I know very few plugins and themes shipping TimThumb have updated their internal copies. My advice is to run `locate timthumb.php` across all your servers and ensure that you replace all instances with an up-to-date copy [7]. Then, you should check for signs of intrusion (check your `wp-config.php` for strange `include` directives).

Conclusion

Developers face some difficult decisions. Re-inventing the wheel is slow and error prone. My advice is to stick to packages that ship as standard (Debian [8], Fedora [9], etc.) because they are already in the system and usually are well maintained. If you need an external library that isn't available, please package it separately so that it is very obvious which version is included. If that's not possible (because you need to modify the library significantly or use an older version for compatibility reasons), make it abundantly clear that it's there. Include the library in the package metainfo (e.g., the `rpm -q --provides`), document it, and ideally keep the file names the same so that people can find it easily. Also, make sure to document where you got the source code. And, please – I implore you – let the upstream vendor know that you're using it and sign up for any security or notification list they have. ■■■

INFO

[1] Silvio Cesare – PackageCloneDetection: *https://github.com/silviocesare/PackageCloneDetection*

[2] Debian – Source Packages: *http://www.debian.org/doc/debian-policy/ch-source.html*

[3] Fedora – No Bundled Libraries: *https://fedoraproject.org/wiki/Packaging:No_Bundled_Libraries*

[4] Calibre – SUID Mount Helper has 5 Major Vulnerabilities: *https://bugs.launchpad.net/calibre/+bug/885027*

[5] "Reduce Your Risk" by Kurt Seifried, *Linux Magazine*, December 2010: *http://www.linuxpromagazine.com/Issues/2010/121/Reduce-Your-Risk/%28kategorie%29/0*

[6] WordPress TimThumb.php Vulnerabilities: *http://www.exploit-db.com/exploits/17872/*

[7] TimThumb.php latest version: *http://timthumb.googlecode.com/svn/trunk/timthumb.php*

[8] Debian Packages: *http://www.debian.org/distrib/packages*

[9] Fedora Package Database: *https://admin.fedoraproject.org/pkgdb*