CERT/CC Blog

Vulnerability Insights

Mining Ubuntu for Interesting Fuzz Targets

POSTED ON AUGUST 15, 2013 BY JONATHAN FOOTE [/AUTHOR/JONATHAN-FOOTE] IN RESEARCH [HTTPS://INSIGHTS.SEI.CMU.EDU/CERT/RESEARCH/]

Hello, Jonathan Foote here. In this post I'll explain how to use information from databases in stock Ubuntu systems to gather the parameters needed to perform corpus distillation [http://googleonlinesecurity.blogspot.com/2011/08/fuzzing-at-scale.html] (gathering of seed inputs) and fuzzing against the installed default file type handlers in Ubuntu Desktop 12.04. This technique applies to most modern versions of Ubuntu.

Default file type handlers in Ubuntu are of particular interest to security as they are the applications that are generally invoked by users when they double-click a file in their web browser, email client, or file browser. Thus, bugs in these applications could be leveraged by attackers to perform spear phishing or other remote code execution attacks.

Figure 1. Could be trouble

Background

I'm the primary investigator on two Software Engineering Institute (SEI) R&D projects [http://www.sei.cmu.edu/research/] in vulnerability discovery. Our research team works with the Carnegie Mellon University (CMU) Department of Electrical and Computer engineering to research using black-box [http://www.sei.cmu.edu/reports/12tn019.pdf] and concolic fuzzing, automatic exploit generation [http://users.ece.cmu.edu/%7Esangkilc/papers/oakland12-cha.pdf], and other vulnerability discovery techniques to secure software.

One of the challenges in validating application vulnerability discovery and remediation techniques is scaling

systems to apply to large volumes of applications. While not applicable to all of the binaries in Debian wheezy [http://lists.debian.org/debian-devel/2013/06/msg00720.html], this blog post describes a technique for garnering the information necessary to apply basic vulnerability discovery techniques (like black box fuzzing) to a large number of applications.

An Example Workflow for Vulnerability Discovery

For the purposes of this blog post, I'll assume we plan to perform simple black box mutational fuzzing against applications in Ubuntu using a tool such as CERT BFF [http://www.cert.org/vuls/discovery/bff.html] or Peach Fuzzer [http://peachfuzzer.com/]. Before we can fuzz an application using one of these tools, we need the following information, at a minimum:

1. The application invocation string
2. A set of seed files to use as a basis for mutational fuzzing

An application invocation string is the text command line used to launch a target application, such as 'firefox somefile.html'. In our case a set of seed files is a representative set of files that exercises diverse functionality in an application when parsed respectively. Imagine something like 'a.html, foo.gif, bar.jpg' for firefox, but more extensive. A fuzzing tool mutates these seed files and commands a target application to consume them to find interesting and perhaps exploitable states in the application.

As an aside, note that in certain cases it might be wise to fuzz the codecs used to parse particular file types (such as libjpeg, for example) directly as a starting point for vulnerability discovery. I am going to gloss over this detail for now.

As I'll describe later, we can determine application invocation strings using information in the Ubuntu desktop entry database. We can use a common, highly parallelizable, technique known as corpus distillation (example here [http://googleonlinesecurity.blogspot.com/2011/08/fuzzing-at-scale.html]) to filter a large set of potential seed files to a smaller set that is more tractable for some of the seed file selection techniques used by BFF and Peach. So, our example vulnerability discovery workflow for this blog post is:

1. Determine the set of applications to analyze
2. Determine command line invocation strings for each application
3. Gather "minsets" of seed files of the types of files that each application consumes, respectively *Conventionally, a "minset" is the set containing the minimum number of files that maximize (or minimize) some metric (such as code coverage) when consumed by the target application
4. Fuzz applications using command line invocation strings and seed files
5. Analyze results

This post will focus on using information in databases in a stock Ubuntu Desktop 12.04 system to support the first three steps of this workflow.

Determining the Set of Applications to Analyze

As I mentioned in the introduction, in this post we will be focusing on the default file type handlers in Ubuntu

Desktop 12.04.

There are actually multiple systems for determining which application handles a given file type in Ubuntu. For example, an older, command-line system based on the mailcap conventions exists in many versions of Ubuntu. We, however, will be focusing on the default handlers used by the Gnome desktop system. This is the system that launches well-known applications that run on Gnome, such as firefox, evince, rhythmbox, and so on, when a file is double-clicked.

Gnome uses a file at `/etc/gnome/defaults.list` to determine default file type handlers. The file acts as an ordered list of associations between "`.desktop`" file names and MIME types (more on these later). Here is a snippet from the file.

```
$ head /etc/gnome/defaults.list
[Default Applications]
application/csv=libreoffice-calc.desktop
application/excel=libreoffice-calc.desktop
application/msexcel=libreoffice-calc.desktop
application/msword=libreoffice-writer.desktop
application/ogg=rhythmbox.desktop
```

There is an effort to standardize how default file type handlers are defined across Gnome and KDE. You can learn a good deal about how these systems work at freedesktop.org [http://freedesktop.org/wiki/Specifications/]. For Gnome, the Free Desktop Entry Specifications at freedesktop.org [http://standards.freedesktop.org/desktop-entry-spec/latest/index.html] and the gnome.org developer documentation for adding applications to the desktop menu [https://developer.gnome.org/integration-guide/stable/desktop-files.html.en] are helpful. However, these specifications are a work in progress [http://www.gnome.org/news/2013/04/report-from-the-freedesktop-summit/], so to get the whole story I had to search some forums and take a look at the systems in my particular instance of Ubuntu. The xdg-mime and gvfs-* commands are good starting points for this analysis.

Regardless, we can uniquify `/etc/gnome/defaults.list` to determine a list of default file handlers for a given Ubuntu distribution. Here is a simple example Python script that prints the default handlers to stdout:

```
#! /usr/bin/env python

seen = []
for line in file("/etc/gnome/defaults.list", "rt").readlines():
line = line.strip()
if "=" not in line:
continue
mt, filename = line.split("=")
if filename not in seen:
```

```
seen.append(filename)
print filename
```

Here is a list of the ***installed*** default file handler applications in the stock Ubuntu Desktop 12.04 nightly build Amazon EC2 virtual machine [http://cloud-images.ubuntu.com/locator/ec2/]:

```
libreoffice-calc
libreoffice-writer
rhythmbox
evince
totem
libreoffice-draw
libreoffice-math
libreoffice-impress
file-roller
ubuntu-software-center
firefox
gedit
eog
nautilus
rhythmbox-device
shotwell
thunderbird
```

Determining the Command Line Invocation Strings for Each application

Each of the "`.desktop`" filenames that we gathered from defaults.list corresponds to something called a desktop entry, which Gnome uses to launch applications. Again, I recommend looking at the details of how your instance of Ubuntu works, but the gnome.org developer documentation for adding applications to the desktop menu [https://developer.gnome.org/integration-guide/stable/desktop-files.html.en] is quite helpful. It turns out that the default file type handler desktop entries for installed applications are located in the `/usr/share/applications` directory on my instance of Ubuntu. Note that, beyond the default handler entries, there are many more desktop entries that can be explored in this directory, and even more that can be gathered from software repositories.

Each desktop entry includes some useful information - in particular, the command line invocation string and the MIME types that this application can handle (this will come in handy later). Fortunately, freedesktop.org supplies an open-source licensed Python library for handling desktop entry files [http://freedesktop.org/wiki/Software/pyxdg/], so we don't have to write our own parser. Here is an extension of our simple Python script that will print the command line invocation strings for each of the default file type handler applications:

```
#! /usr/bin/env python
```

```
import os, sys
from xdg_app import App

seen = []

for line in file("/etc/gnome/defaults.list", "rt").readlines():
line = line.strip()
if "=" not in line:
continue
mt, filename = line.split("=")
if filename not in seen:
seen.append(filename)
filepath = "/usr/share/applications/" + filename
if not os.path.exists(filepath):
sys.stderr.write("desktop file not found: %s\n" % filepath)
continue
if not os.path.exists(filepath):
errors[filename] = errors.get(filename, []) + [mt]
continue
print App(filepath).getExec()
```

Note that this script has a condition for when desktop entry files don't exist in `/usr/share/applications` on the filesystem. This is due to the fact that the `/etc/gnome/defaults.list` file specifies default file type handlers for both installed and not-installed applications. You can read more about this convention at gnome.org [https://developer.gnome.org/integration-guide/stable/desktop-files.html.en] . For this blog post, we will ignore applications that aren't included in the stock Ubuntu Desktop 12.04 installation.

The script above will print command line invocation strings, but be warned that these strings may invoke scripts rather than launching the target application binaries. Depending on your purposes, you may need to mine the binary invocation path from the script file printed by our Python script.

Gathering "Minsets" for Each Application

In our research, we experiment with multiple techniques for gathering and distilling a corpus of seed inputs for use in black box mutational fuzzing and other vulnerability discovery techniques. In general, the steps we use for corpus distillation are:

For each application,

1. Determine the extensions of the file types used by the application
2. Crawl a large set of files for files matching those extensions
3. For each file,

- gather information from dynamic analysis of the target application when run with the file as input

4. Select files for the application's seed input corpus based on analysis results

While there are many interesting details to discuss in corpus distillation, the remainder of this post will focus on mining information from databases in Ubuntu.

Mining the extensions of the file types used by the application

As I mentioned above, the Ubuntu desktop entries specify the MIME types that can be handled by each application, respectively. Your mileage may vary, but some common techniques [http://fuzzinginfo.files.wordpress.com/2012/05/ben-nagy-prospecting-for-rootite-2010.pdf] for crawling sets of files require querying by file extensions rather than mime types.

Fortunately, we can use the same technique of reviewing the freedesktop.org standards [https://developer.gnome.org/shared-mime-info-spec/] and the details of our particular instance of Ubuntu to mine the Gnome MIME type database for associations between MIME types and file extensions. In my instance of Ubuntu Desktop 12.04, I can find the mime type information I need in the files at `/usr/share/mime/globs2` and `/usr/share/mime/aliases`. I will leave parsing these files as an exercise to the reader.

Next Steps: Distilled Corpuses for All?

At this point we have mined enough useful information out an instance of Ubuntu Desktop 12.04 to begin fuzzing all of the default handlers in the system, and to get started with finding some interesting bugs. These activities would be handled in steps 4 and 5 from our example vulnerability discovery workflow above ("Fuzz applications using command line invocation strings and seed files" and "Analyze results"), but they are outside the scope of this blog post.

While bug hunting in default file handlers is interesting, there may be some other ways to apply this technique. Note that some well-known fuzzing campaigns [http://fuzzinginfo.files.wordpress.com/2012/05/cmiller-csw-2010.pdf] have built distilled file corpuses of a given type (such as application/pdf) using a full-featured representative application (such as Adobe Reader) and then re-used the corpuses against other applications (such as Foxit Reader). The technique described in this blog post allows one to build corpuses against a large number of applications that consume many media types ...hmm...

If you are interested in this area of work, have questions, or would like to learn more about our research in vulnerability discovery at CMU SEI CERT feel free to drop me a line via the feedback link to the right. Thanks for reading.

About the Author

Jonathan Foote

Contact Jonathan Foote [https://www.sei.cmu.edu/contact.cfm]
View other blog posts by Jonathan Foote [/author/jonathan-foote]