This document describes what the vsftpd code trusts, what it doesn't trust, and the reasoning behind any trust decisions.

The importance of trust and trust relationships

Imagine a largely well written and secure piece of code. Now imagine that this piece of code delegates a task to an external program, perhaps in the name of code reuse. Now, if this external program is sloppily coded and insecure, we've wasted a lot of effort making our original program secure; our erroneous trust of the buggy external program means we have a security leak, even though we were careful in _our_ code.

There is a very similar situation with buggy library APIs. Imagine our secure program calling some complex library function which lets the side down by containing a security hole.

Lets put some concrete examples on the two similar above considerations. We can even give examples in the context of FTP daemons.

1) External /bin/ls helper

A very common operation asked of FTP servers is to provide a directory listing. Unfortunately, convention seems to be to emit the directory listing in UNIX "/bin/ls -l" format. Even the Microsoft FTP service can be observed to do this. When writing an FTP server for the UNIX platform, then, this leads to the temptation to reuse /bin/ls as a child process, to avoid having to rewrite a load of code to handle directory listings.

Even more unfortunately, FTP server writers seem to want to adopt the versatility of the average /bin/ls implementation. This means they allow clients to specify arbitrary parameters to /bin/ls.

By using an external /bin/ls command, we would tie the security of our FTP server to that of the /bin/ls code. Be careful not to underestimate the amount of code paths in /bin/ls which are explorable by a remote malicious user. GNU /bin/ls has a myriad of options. Some of these options are complex such as -I or the various formatting options. All it takes is a single coding flaw in the handling of one of these options, and your FTP security is in trouble.

By using an external /bin/ls, you also inherit the risk of any dangerous or complex APIs it uses. For example, calls to libc's complex fnmatch() or glob() functions, which will get given arbitrary malicious user controlled data as the search patterns. Also remember that users (and sometimes remote users) can upload/create files, and filenames are a very prominent input to /bin/ls.

To conclude: vsftpd has no intention of using an external /bin/ls program because of the risks outlined above. Even if I were to audit e.g. GNU fileutils /bin/ls, and also important parts of glibc, this would still leave security in an unknown state on other platforms. The solution I have employed is to write a minimal internal implementation of a /bin/ls listing generator; it's hardly difficult. As a happy side effect, this will boost performance by avoiding unneccesary fork()s and exec()s!

Here's some quick data about FTP servers which tend to use external ls programs:

```
ftp.wuftpd.org:
ftp> ls --version
227 Entering Passive Mode (x.x.x.x.x.x)
150 Opening ASCII mode data connection for /bin/ls.
ls (GNU fileutils) 3.16
226 Transfer complete.

ftp.digital.com:
ftp> ls -v
227 Entering Passive Mode (x.x.x.x.x.x)
150 Opening ASCII mode data connection for /bin/ls.
/bin/ls: illegal option -- v
usage: ls [ -1ACFLRabcdfgilmnopqrstux ]  [files]
226 Transfer complete.
```

Note that /bin/ls is not the only external program invoked by common FTP servers such as wu-ftpd. wu-ftpd also has the ability to invoke "tar" and "gzip" on the fly, so there are trust relationships there too.

2) Complex library APIs

vsftpd is very careful to avoid using library calls which are potentially dangerous. I would typically classify calls as dangerous if they interact with the network non-trivially, or take malicious user supplied data and start parsing it in a major way.

Some examples are clearly required (vsftpd avoids using any of the following):

```
1) fnmatch(). This is the libc glob pattern matcher. The danger comes
from the fact that the user supplies the glob pattern - "ls *.mp3" would
be a simple example. Furthermore, glob pattern matching is complex and
involves a lot of string handling.

2) gethostbyaddr(). This is a libc call to resolve an IP address to a hostname.
Unfortunately, doing this is quite complicated. When you call gethostbyaddr(),
a lot of work goes on under the covers. This usually involves making a network
call out to the DNS server, and, dangerously, parsing the response.

For clarity (and clarity is a very important part of security), all external
APIs used by vsftpd are encapsulated within two "system interaction" files,
named "sysutil.c", and "sysdeputil.c" (for the more variable/system dependent
calls). This provides a convenient audit point for ascertaining which calls
vsftpd trusts.

vsftpd-2.0.0 introduces SSL / TLS support using OpenSSL. OpenSSL is a massive
quantity of code which is essentially parsing complex protocol under the full
control of remote malicious clients. SSL / TLS is disabled by default, both
at compile time and run time. This forces packagers and administrators to make
the decision that they trust the OpenSSL library. I personally haven't yet
formed an opinion on whether I consider the OpenSSL code trustworthy.


Summary
=======

Be very aware of what APIs and/or programs you are trusting, or you might end
up creating a trust relationship which makes your program exploitable --
through no direct fault of your own.
```