

Analysis of FileZilla codebase

Ravibabu Matta, Martin Ukrop, Jiří Weiser

December 8, 2014

1 Introduction

FileZilla is a free, cross-platform FTP application software for multiple platforms. The source code is open and licensed as GNU GPLv2. The most recent stable version as of now is 3.9.0.6, which was used for the purposes of all below-mentioned analyses. We inspected the client-side application, compiled either on Windows (for *CppCheck* and *PreFast* analyzers) or Linux (for *Valgrind* analysis).

2 Static analysis using CppCheck

Static analysis of FileZilla using CppCheck reported the following issues:

- In *TinyXML*, array index used before limits check: `for(int i=0; p[i] && i< *length; ++i)` and copy-constructor is written as `TiXmlHandle operator=()` instead of `TiXmlHandle &operator=()`.
- Expression is always false because “else-if” condition matches previous condition (it’s a false positive): `if(c==0) ...; else if(...|| (c<expr2>)) ... else if(c==0)...`
- 134 style issues for reducing the scope of the 62 variables and for 72 C-style pointer casting.
- 9 portability issues for reading integers using `scanf` without field limit (crashes in *libc* older than 2.13-25). False positives, as the input was read from fixed sized buffer using `sscanf`.

3 Static analysis using PreFast

Static analysis using *PreFast* requires the FileZilla to be compiled using Visual Studio. *Filezilla-3.9.0.6* depends on *gnutls-3.3.9-w32*, *wxWidgets-3.0.0*, *sqlite-amalgamation-3080702* for compiling. Project property file *Dependencies.props* is required to be created for defining the user macros, which gives information about the path to the dependent include files and libraries to link. The Visual Studio solution, which comes with FileZilla was used for build. *PreFast* reported the following issues:

- Read overrun or write overrun: difficult to understand, requires in-depth understanding.
- Dereferencing a null pointer:
 - Before dereferencing there is an assert function call, so these are false positives.
 - Dereferencing a pointer value returned from iterator.
 - A function is called with a pointer variable as a parameter – if the parameter is null then the function returns null. The return value from function is checked and if it is not null, the pointer variable is accessed. False positive reported by *PreFast* that the pointer variable could be null.
- Using uninitialized memory: only in platform dependent code (in `#ifdef __WXMSW__`).
- Return value ignored: function calls are annotated with `_Check_return` but alternate mechanism is used for checking i.e., checking the output parameters.

4 Dynamic analysis using Valgrind

Running FileZilla with Valgrind *memcheck* tool resulted in several leaks in the program. We inspected a GNOME version of the software, thus we employed some common suppression patters to try to exclude leaks caused by external libraries. The suppression files used can be obtained from <https://github.com/dtrebbien/GNOME.sup>.

The inspected run consisted of just opening the program and correctly closing it again (*File > Exit*). This resulted in:

- 6.5 kiB of definitely lost bytes,
- 33.3 kiB of indirectly lost bytes,
- 51.9 kiB of possibly lost bytes,
- 2.4 MiB of bytes still reachable at exit and
- 13.4 kiB of suppressed bytes.

The blocks still reachable at exit are of little interest to us, since the optimization strategy of GTK is to leave some object allocated for cleanup during program termination. However, those 91.7 kiB of (possibly or indirectly) lost data should not be there. It may be (at least partly) due to leaks in supporting libraries (mainly GTK). Furthermore, this is only a result for plain opening and closing of the program – with more interaction the number of leaks may rise.

5 Issues regarding password processing

From version 3 onward, FileZilla saves all stored usernames and passwords in an unencrypted form in the file *sitemanager.xml*. The file is located in application’s configuration folder in user’s profile. In GUI, nothing indicates plain storage (the password is hidden by bullets). Although a bit unorthodox solution, showing the password in clean in GUI would more properly reflect the real storage type.

When connecting to servers not explicitly saved, the application asks once, if it should save passwords for convenience of the future use. This, if agreed on, again results in plain storage in a similar location. The choice in the mentioned dialog window is applied as a global setting to (not) save passwords (except those explicitly input in the site manager).

6 Conclusions

The static and dynamic analysis tools did not find any severe flaws in FileZilla. This was expected, as it is a well-established software with some active developers (most recent commit within the last month). There were, however, some less important style and memory issues. The underlying libraries are also not bug-free and don’t use the memory properly.

There is an important issue with plaintext password storage. However, there’s no simple and correct solution here – it’s a part of a wider discussion concerning offline password management having pros and cons on both sides.

No issues were reported back to the developer, since all found were of low severity and either already known (password storage), too vague without in-depth analysis (read overrun, lost bytes) or debatable (style issues).