Chapter 2.1 - Pidgin

Pidgin Post-mortem

The Deliverable #2 work below was written for our September 27th Testing Plan. Our task in revising this test plan was to get much more specific in terms of writing user requirements and associated test cases, but we weren't able to get that far with Pidgin and have decided to switch projects! This section is a record of our efforts to run Pidgin functions at the command line.

We were originally quite optimistic about creating a testing framework for Pidgin, because the source files came with a "test-driver" and associated test files. We weren't able to figure out on our own how to run the test-driver, because it required a number of command line arguments that we didn't understand and for which we couldn't find documentation. We emailed the development team on September 9th requesting some direction on using the test-driver but never heard back. We concluded that the test-driver is probably used within the application, by the application, for the purposes of testing encryption protocols while the application is running.

The next step was to identify methods that we would be able to run from the command line, but we were unable to do so. Our strategy was to search in the simplest possible c files (those with the fewest #include dependencies) for methods that required simple inputs that we could replicate. Most of the methods we found usually took some type of complex data structure as an argument, usually related to the user's buddy list and other social and communication connections, but we did find some simple functions with straightforward arguments. The major problem was that we couldn't find any c files that we were actually able to compile due to complex dependencies. We tried to manually compile dependencies and we also tried to use the built-in makefile for our own purposes, but in both cases the composition of files was too complex for us to break down into smaller independent components that we could call from a command line.

Our final step was to try working just with the finch package which is the underlying communication platform without the extra pidgin bells and whistles. Finch has its own makefile, and we tried the same approach of compiling only finch and isolating finch methods, but we were stymied by the same complexity issues of the whole pidgin program.

In retrospect, it's not too surprising that Pidgin proved too difficult a problem to reduce into smaller pieces. Probably any instant messaging platform is going to have integrated encryption that is always running and is going to always be connected to a dynamic contact list data structure as well as other profile information stored online.

Deliverable #2 (Original)

The Testing Process

The ultimate goal of our team term project is to develop an automated testing framework for Pidgin and implement the framework with twenty-five test cases, five injection faults, and plenty of documentation. The project is incremental, as we work toward our completed test suite through several chapters described below. While Pidgin is an active and well supported open-source project, our challenge as a team will be to identify tests that are manageable for us to implement given our limited experience with the C programming language, communication protocols, and encryption algorithms.

1. Deliverable #2: Overall test plan and identification of our first five test cases.
2. Deliverable #3: Automated testing framework with all necessary documentation on how it works and how to use it.
3. Deliverable #4: Implementation of our automated testing framework using our 25 test cases.
4. Deliverable #5: Fault injection - 5 faults that cause some but not all of our 25 test cases to fail.
5. Deliverable #6: Final Report

Requirements traceability

In the broadest sense, the user requirements for Pidgin involve secure access to a wide range of instant messaging protocols with a high level of availability. The major tasks of Pidgin then are to securely handle user authentication, protect against intrusions, and quickly connect users to the networks and channels within networks that they wish to access. Secondary requirements include user profile management, buddy management, and GUI experience.

Tested items

Methods that will be tested (identify by directory/filename/linenumber-methodname):
Nicholas:
* libpurple/core.c/289-purple_core_get_version(); Simple method, when called is supposed to return the version, we can run it and make sure that it is returning the correct version. *update: I used our test driver to try and run this method and i got this message...
pidgin-2.11.0/libpurple/internal.h:102:21: fatal error: gmodule.h: No such file or directory. This looks like a bug in their code. I'll have to look into it.
* libpurple/version.c/ *purple_version_check(guint required_major, guint required_minor, guint required_micro)
* libpurple/version.h/ shows us the correct version numbers.

* libpurple/idle.c/291-purple_idle_get_ui_ops(void); Looks to be a simple get method. Ran tests and got the same issue with idle.c, no Gmodule.h which is supposed to be included in internal.h.
* libpurple/idle.c/291-purple_idle_get_handle(void); Another get method.

Walter:
* libpurple/buddyicon.c/248-purple_buddy_icon_data_new(guchar *icon_data, size_t icon_len, const char *filename); glib dependencies noted above are solved by including a flag that tells the compiler to look for glib library. Additional dependencies throw errors, I believe they are other pidgin files that need to be compiled.
* libpurple/imgstore.c/70-purple_imgstore_new_from_file(const char *path)

Matthew:
* libpurple/proxy.c
After solving glib.h and glibconfig.h dependencies, trying to include the proxy.c leads to a lot of undefined references.
* libpurple/tests/test_cipher.c
Including this in the test driver does not give undefined references, but has a dependency on an include interal to the test_cipher.c file.

Testing schedule

Deadline	Deliverable	Resources
Sept 27th	Test Plan & five test-cases	Pick first test cases and begin work on test driver.
Oct 18th	Automated Testing Framework	Resolve compiler dependencies, finish test driver, get used to IDE C unit testing features, organize team github repo directory
Nov 10th	Testing Implementation with twenty-five test-cases	TBD
Nov 22nd	Fault Injection	TBD
Nov 29th	Final Report	TBD

We have been working on the automated testing framework. We have worked out how to write a script that calls a C test driver whose main invokes an external function. We have uploaded a sample script that runs the test driver for a separate hello world function to document our progress. We believe that these three files represent the framework by which we will test specific pidgin functions with our test driver.

We do not yet understand how to invoke a pidgin function. We run into errors when trying to compile our test driver and pidgin file that contains the function we want to test. The compiler originally threw errors related to the glib library that it was not finding on our computers, but adding a flag to the compiler invocation solved these glib dependencies. Additional dependencies relate to other pidgin files and will presumably be solved by calling the test driver in a fully compiled program rather than the raw source code where all of the dependencies of our test function have not yet been compiled.

Test recording procedures

All test results will be pushed to /TestAutomation/temp with a structured name format (TBD) by COB on the day the test was run.

Hardware and software requirements

Test suites will be small and therefore will not require a great deal of hardware resources. The Eclipse C/C++ IDE with CDT unit testing feature will be used for software test development.