Instant Message Whispering via Covert Channels

Qualitätssicherungsdokument

Gruppe NR: Jan Simon Bunten <mail@xyz.de>
Simon Kadel <mail@xyz.de>
Martin Sven Oehler <mail@xyz.de>
Arne Sven Stühlmeier <mail@xyz.de>

Teamleiter: Philipp Plöhn <mail@xyz.de>

Auftraggeber: Titel Carlos Garcia <carlos.garcia@cased.de>
FG Telekooperation
FB 20 - Informatik

Abgabedatum: 15.2.2014

TECHNISCHE UNIVERSITÄT DARMSTADT

Bachelor-Praktikum WS 2013/2014
Fachbereich Informatik

Inhaltsverzeichnis

1 Einleitung

Ziel des Projekts ist es, eine Bibliothek zu entwickeln, die die unentdeckte Kommunikation zwischen zwei oder mehreren Teilnehmern erlaubt. Um dies zu ermöglichen, verwendet die Bibliothek Covert Channels. Dabei handelt es sich um Kommunikationskanäle, die von außen nicht als solche erkennbar sind.

Im Unterschied zur Kryptographie, die nur die Daten eines Kanals verbirgt, wird so der ganze Kanal verborgen. Dadurch wird es Dritten erschwert, die Verkehrsdaten der Verbindung (Zeitpunkt, Dauer) auszuwerten. Anfangs werden wir dazu ein Framework implementieren, das die Basisfunktionalitäten zur Verfügung stellt. Darauf aufbauend werden wir unterschiedliche Covert Channels entwinkeln, die dem Framework als Plugin hinzugefügt werden.

Abschließend soll das Projekt als Open-Source veröffentlich werden, um es anderen zu ermöglichen die Bibliothek in ihren Projekten zu verwenden oder die Bibliothek weiterzuentwinkeln. Außerdem soll die Bibliothek dazu dienen, Maßnahmen zum Aufdecken von Covert Channels zu entwickeln.

2 Qualitätsziele

2.1 Zuverlässigkeit

Der Benutzer einer Bibliothek verlässt sich darauf, dass sie korrekt funktioniert und macht, was in der Dokumentation festgehalten ist. Deshalb ist Zuverlässigkeit für eine Bibliothek unbedingt notwendig.

Die Zuverlässigkeit kann durch Testen verbessert werden. Deshalb benutzen wir die boost.test Bibliothek, die automatische Tests in C++ ermöglicht. Für jede Methode wird mindestens ein Test geschrieben und für jede Aufgabe (siehe Testbarkeit) mindestens 2 Tests. Diese werden mindestens einmal pro Woche auf der aktuellen Version der Software komplett ausgeführt. Fehler werden im Ticketsystem unseres SCM- Servers eingetragen.

Außerdem führen wir Code Reviews durch. Nach Abschluss eines Use Cases wird der Code von einem an diesem Use Case unbeteiligten Teammitglied anhand einer Checkliste überprüft. Mögliche Fehler werden schnellstmöglich von den Entwicklern behoben. Dann wird der Vorgang wiederholt, bis die Kriterien erfüllt sind.

2.2 Testbarkeit

Aus der Zuverlässigkeit ergibt sich ein weiters Qualitätsmerkmal. Wenn man die Software Testen will, muss sie auch testbar sein. Dabei geht es nicht nur um Unittests, sondern vorallem um Integrations- und Systemtests. Damit es einfach ist, diese durchzuführen, wird eine klare Beschreibung und Trennung der Aufgaben benötigt (Seperations of Concerns). Das erreichen wir, indem wir während des Designs alle Aufgaben unserer Software festhalten und einem unserer Module zuweisen.

Es ist sehr schwierig, Testbarkeit durch Werkzeuge sicher zu stellen. Eine gute Testbarkeit ergibt sich durch eine Architektur der Sofware, die dieses Qualitätsmerkmal beachtet. Es muss beim Entwurf berücksichtigt und während der Entwicklung stets überprüft werden.

Die einfachste Möglichkeit, die Testbarkeit zu überprüfen, ist es, Tests zu schreiben und dabei festzustellen, wie gut dies möglich ist. Wenn dabei auffällt, dass manche Funktionalitäten nur schwer oder nicht zu testen sind, werden wir dies in unseren Teamtreffen besprechen und eine Lösung durch Anpassen der Architektur ausarbeiten.

A Anhang

(Am Ende des Projekts nachzureichen)
Beleg für durchgeführte Maßnahmen, bzw. falls nicht durchgeführt eine Begründung wieso die Durchführung nicht möglich oder nicht erfolgt ist.
Weitere Anforderungen sind den Unterlagen und der Vorlesung zur Projektbegleitung zu entnehmen.