Instant Message Whispering via Covert Channels

Qualitätssicherungsdokument

Gruppe 35:	Jan Simon Bunten <jan_simon.bunten@stud.tu-darmstadt.de> Simon Kadel <simon.kadel@stud.tu-darmstadt.de> Martin Sven Oehler <martin_sven.oehler@stud.tu-darmstadt.de> Arne Sven Stühlmeier <arne_sven.stuehlmeier@stud.tu-darmstadt.de>
Teamleiter:	Philipp Plöhn <philipp.ploen@stud.tu-darmstadt.de>
Auftraggeber:	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 es ermöglicht, unentdeckt Kommunikationskanäle zu einem oder mehreren anderen Teilnehmern zu öffnen. Um die Kommunikation vor Dritten zu verstecken, werden sogennante Covert Channels verwendet.

Covert Channels

Covert Channels sind Kommunikationskanäle, die von Außen nicht als solche erkennbar sind. In der Literatur sind viele unterschiedliche Covert Channels bekannt.

Im Unterschied zur Kryptographie, die nur die Daten eines Kanals verbirgt, wird der ganze Kanal verborgen. Dadurch wird es Dritten erschwert, die Verkehrsdaten der Verbindung (Zeitpunkt, Dauer) auszuwerten. Ist es möglich die Pakete einer bestehenden Verbindung anderer Teilnehmer des Netzwerks zu verändern, kann damit auch die Identität der Nutzer verborgen werden.

Wie bei offenen Kanälen ist es auch bei Covert Channels von großer Bedeutung, wie groß der Datendurchsatz ist und wie zuverlässig die Informationen übertragen werden. Vor allem der Datendurchsatz ist bei Covert Channels üblicherweise stark beschränkt.

Implementierung

Das Hauptziel ist, ein Framework zu implementieren, das notwendige Funktionen für die Covert Channels bereit stellt. Dazu gehören das Öffnen und Schließen von Covert Channels, das Senden von selbst erstellten Paketen, das Empfangen von Paketen und das Anzeigen von Statistiken der geöffneten Kanäle. Die eigentlichen Covert Channels können als Plugins hinzugefügt werden. So soll sichergestellt werden, dass die Bibliothek für unterschiedliche Covert Channels genutzt werden kann.

Neben dem Framework werden wir im Rahmen des Projekts drei unterschiedliche Covert Channels implementieren. Zuerst ein einfacher Channel, der darauf beruht Informationen im Header eines TCP oder UDP Pakets zu verstecken. Als zweites einen komplizierterer Covert Channel aus der Literatur. Wenn möglich soll dann noch ein dritter von uns entwickelter Covert Channel implementiert werden.

Das Projekt soll als Open-Source veröffentlich werden, um es anderen zu ermöglichen die Bibliothek in ihren Projekten zu verwenden oder die Bibliothek weiterzuentwinkeln. Wir werden eine Beispielanwedung in der Art eines Instant Messagers entwickeln, um die Funktionen zu demonstrieren.

2 Qualitätsziele

2.1 Zuverlässigkeit

Der Benutzer einer Bibliothek verlässt sich darauf, dass sie korrekt funktioniert und sich wie in der Dokumentation beschrieben verhält. 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 zwei 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 einer User Story 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 vor allem 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.