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 Reife

Begründung: Unser Projekt soll in anderen Programmen als Bibliothek verfügbar sein. Insbesondere soll es zum Testen eines Tools, mit dem Covert Channels entdeckt werden können, verwendet werden. Der Programmierer verlässt sich darauf, dass die Bibliothek nicht der Grund für Fehlverhalten oder Abstürze ist, sondern fehlerfrei funktioniert.

Maßnahmen: Um die Reife unseres Projekts zu erhöhen, müssen wir versuchen möglichst viele Fehler zu finden und zu beheben. Das geschieht bei uns durch zum einen durch Testen, zum anderen durch Code Reviews.

Zum Schreiben von automatischen Tests in C++ benutzen wir die Boost.test Bibliothek. Wir schreiben sowohl Tests für einzelne Methoden (Unittests), als auch für Funktionen der Bibliothek (Systemtests). Mit den Tests soll mindestens Statement Coverage erreicht werden, wobei Getter/Setter und einfache Wrapperfunktionen ausgenommen sind.

In unseren Code Reviews untersuchen wir den Codes unter bestimmten Aspekten auf typische Fehlerquellen. Dabei wird der Code anhand einer Checkliste (siehe Anhang) zum Beispiel auf Punkte wie Endlosschleifen oder Parameterüberprüfung hin überprüft.

Prozess: Die Unittests werden vor oder während der Entwicklung vom Entwickler geschrieben und unseren automatischen Tests hinzugefügt. Nach der Implementierung einer User Story schreibt der Entwickler die Systemtests für die in der User Story beschriebene Funktion. Dabei stellt er sicher, dass die geforderte Abdeckung erreicht wird. Das Durchlaufen der Tests und die Testabdeckung werden bei den Code Reviews überprüft.

Zusätzlich werden die automatischen Tests am Ende eines Sprints auf der aktuellen Version der Software komplett ausgeführt. Auftetende Fehler werden im Ticketsystem unseres SCM-Servers eingetragen und bei der Planung der Sprints mit berücksichtigt. Das Durchführen wird mit Datum und Betriebssystem, auf dem die Tests durchgeführt wurden, in einer Liste festgehalten. Ein Teammitglied wurde dafür ausgewählt zu überprüfen, ob die Tests durchgeführt wurden.

Die Code Reviews werden am Ende jeder User Story durchgeführt. Dafür wird ein neues Ticket erstellt, wodurch alle Teammitglieder benachrichtigt werden, dass ein Code Review aussteht. Code Reviews haben Priorität und müssen innerhalb von drei Tagen durchgeführt werden. Für das Code Review trägt sich ein an der User Story unbeteiligtes Teammitglied ein. Es geht allen in der User Story geschriebenen Code durch und überprüft dabei die Punkte auf der Checkliste. Wenn ein Punkt nicht erfüllt ist, wird das entsprechend vermerkt und an den Entwickler weitergeleitet. Dieser behebt die Fehler im Code und erstellt dann wieder ein Ticket für ein Review. Dieser Prozess wird solange wiederholt, bis alle Punkte der Checkliste abgehakt sind. Erst dann gilt die User Story als abgeschlossen.

2.2 Testbarkeit

Begründung: Außerdem ist die Testbarkeit einiger Funktionen des Projekts komplex, weshalb sie in den Projektzielen vom Auftraggeber als Qualitätsziel hervorgehoben wurde.

Maßnahmen: 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. Außerdem müssen die Schnittstellen und Interfaces gut dokumentiert sein.

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

Konkrete Maßnahme?

Prozess: 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. Beleg?

2.3 Erweiterbarkeit

Begründung: Unser Framework soll nach Abschluss als Open Source Bibliothek veröffentlicht werden, damit andere ihre eigenen Covert Channel schreiben oder die Bibliothek ihren Bedürfnissen anpassen können. Außerdem soll es möglicherweise in weiteren Bachelorpraktika erweitert werden. Um das zu Vereinfachen ist es wichtig, dass unser Code und unser Aufbau der Sofware verständlich ist, und das wir beim Entwurf beachten, dass er modifizierbar ist. Diese beiden Punkte lassen sich unter Erweiterbarkeit zusammenfassen.

Maßnahme: Die Verständlichkeit möchten wir durch zwei Maßnahmen erhöhen. Einerseits soll durch die Dokumentation der Funktionen und Klassen der Bibliothek die Funktionsweise verdeutlicht werden. Andererseits sollen einheitliche Bezeichner und eine einheitliche Struktur des Codes die Lesbarkeit verbessern. So sollen Nutzer sich leichter in der Bibliothek zurecht finden, wenn sie Änderungen oder Erweiterungen vornehmen möchten.

Auch die Modifizierbarkeit wird von diesen beiden Maßnahmen verbessert. Zusätzlich wollen wir durch Modularität und abstrakte Klassen ermöglichen, dass Benutzer Teile der Software leicht austauschen können. Die Aufgaben und Schnittstellen der Module und die Methoden der abstrakten Klassen sind in der Dokumentation festgehalten.

Prozess: Die Dokumentation der Funktionen und Klassen wird durch den Einsatz der Software Doxygen und entsprechend formatierten Kommentaren im Quellcode automatisch erstellt. Gegen Ende des Projekts (Anfang März) sind zwei Wochen vorgesehen, in denen die erzeugte Dokumentation überprüft und erweitert wird.

Eine einheitliche Bezeichung und Formatierung stellen wir durch Code Conventions sicher. Diese wurden zu Beginn des Projekts auf Grundlage der relativ weit verbreiteten Google-Code-Conventions für C++ von uns festgelegt und in unserem Wiki festgehalten. Die Einhaltung der Code Conventions ist ein Punkt auf der Checkliste für die Code Reviews und wird entsprechend mit überprüft. Falls sie nicht eingehalten werden muss, wie bei den anderen Punkten des Code Reviews auch, der Code vom Entwickler entsprechend angepasst und dann erneut überprüft werden.

!!Dokumentation Module!!