Bachelorarbeit

Network Sniffer

Ein Modul für Verinec

vorgelegt von
Patrick Aebischer
patrick.aebischer@unifr.ch

betreut von
Dominik Jungo und David Buchmann

Universität Freiburg, Schweiz
Departement für Informatik
Telecommunications, Networks and Security Research Group
Prof. Ulrich Ultes-Nitsche

Inhaltsverzeichnis

1 Einleitung

Diese Bachelorarbeit wurde im Rahmen des Projekts VERINEC der Universität Freiburg erstellt. VERINEC steht für "Verified Network Configuration".

Ziel von VERINEC ist es, ein Computer Netzwerk erst einmal abstrakt zu definieren und zu konfigurieren. Dabei sollen die einzelnen Netzwerk Komponenten und die angebotenen Dienste simuliert werden, um zu sehen, wie sich solch ein Netzwerk in der Realität verhalten wird. Diese Netzwerk Komponenten, genannt *nodes*, können jegliche an einem Netzwerk betriebenen Geräte sein, also Computer, Router, Drucker, usw. Die abstrakte Konfiguration kann anschliessend für verschiedene existierende Systeme übersetzt werden.

Der Netzwerk Sniffer dient dazu, aus einem schon bestehenden Netzwerk eine Konfiguration zu erstellen, welche in der VERINEC Umgebung verwendet werden kann. Dies wird durch abhören und analysieren des Netzverkehrs erreicht. Ansonsten müsste man eine solche Konfiguration 'von Hand' erstellen, was für grössere Netzwerke sehr zeitraubend sein muss.

Andere Module von VERINEC befassen sich zum Beispiel damit, eine abstrakte Konfiguration in Systemkonfigurationen zu übertragen und so die Einstellungen eines *nodes* auf ein Gerät zu übertragen. Weitere Informationen zu VERINEC sind auf der Homepage von VERINEC zu finden.

Eine kleine Anmerkung bezüglich der in dieser Dokumentation verwendeten Schriftarten ist an dieser Stelle vielleicht angebracht. Wörter in Grossbuchstaben wie VERINEC lassen sich im Literaturverzeichnis nachschlagen, wobei sich kursiv geschriebene Wörter, wie zum Beispiel *nodes*, auf einen speziellen Ausdruck beziehen.

2 Allgemeines

Der Begriff *sniffen* steht hier für das Abhören von Computer Netzwerken. Ein solches Programm, ein Netzwerk Sniffer[1], befasst sich damit, die gesamte Netzwerkkommunikation abzuhören und auszuwerten. Damit lassen sich für einen Netzwerk Administrator legitime Ziele verfolgen, wie etwa das Auffinden einer Schwachstelle im Netzwerk oder das Überwachen der Netzwerklast. Andererseits können jene Programme auch von Personen verwendet werden, um Netzwerke abzuhören oder Passwörter von anderen Benutzern zu sammeln. Die auf diese Art und Weise ergaunerten Daten lassen sich dann eventuell auch böswillig einsetzen.

Zum Abhören von Netzwerken existieren bereits eine Menge von Programmen, wie zum Beispiel Ethereal oder EtherApe, welche diese Aufgabe auch hervorragend lösen. Diese Programme analysieren alle Pakete, welche sie über eine angeschlossene Netzwerkkarte empfangen. Mit Ethereal ist es somit möglich, die einzelnen Protokolle bis auf alle Details zu überprüfen.

Ein neues Programm zum Abhören von Netzwerken muss sich also irgendwie rechtfertigen. Verinec benutzt einen solchen Sniffer um einerseits die benutzten Geräte in einem Netzwerk ausfindig zu machen, andererseits um die dargebotenen Dienste dieser Geräte aufzuzeigen. Es wird nicht analysiert, was ein einzelner Benutzer auf dem Netz tut, sondern nur die verwendeten Dienste und Geräte. Diesbezüglich unterscheidet sich der Sniffer stark von den oben genannten Programmen. Des weiteren sollte solch ein Sniffer in der Lage sein, die gefundenen Daten im *XML*-Format weiterzugeben, da bei Verinec versucht wird, ein einheitliches Format einzuhalten. Aus diesem Grund befinden sich alle Konfigurationsdateien in diesem Format.

Ich habe versucht, für die Arbeit das Rad nicht komplett neu zu erfinden und suchte nach bereits vorhandenen und wenn möglich in *Java* verwendbaren Programmen. Dabei stiess ich auf die im nächsten Kapitel ausführlich erklärten Programme, beziehungsweise *Libraries*.

[1] Ich entschloss mich, für diese Arbeit die englischen Begriffe *sniffen* und *Sniffer* zu benutzen, da diese den Sachverhalt präziser beschreiben als der deutsche Ausdruck *abhören*.

3 Aufbau

3.1 Netzverkehr abhören unter Java

Die meiste Netzwerkkommunikation erfolgt über TCP-Ströme. Es ist jedoch nicht möglich, diese Ströme direkt mitzuschneiden. Grund dafür ist, dass das Betriebssystem diesen Strom filtert und nur jene Pakete weiter gibt, die für ein gewisses Programm gedacht sind.

Um dies zu umgehen, wird die Library LIBPCAP für Linux oder WINPCAP für Windows benötigt, welche jenen Datenstrom direkt an ein Programm weiterleiten können. Um dies zu erreichen, muss die Netzwerkkarte in einen Betriebszustand gebracht werden, welchen man *promiscuous mode* nennt. Dies bezeichnet das gegenteilige Verhalten wie oben erläutert; es werden also alle Pakete, welche die Netzwerkkarte empfängt, ungefiltert weitergeleitet.

Jedoch sind die Libraries in C geschrieben und somit nicht direkt aus Java brauchbar. Um von Java auf die Funktionen dieser Libraries zuzugreifen, wurde ein Java Wrapper namens JPCAP entwickelt. Der von mir geschriebene Netzwerk Sniffer benötigt diesen Wrapper und die JPCAP-Libraries um auf einem Netzwerk Pakete sniffen zu können.

Mit Hilfe von JPCAP ist es möglich, die empfangenen Pakete genau zu erkennen. Es kann zum Beispiel herausgefunden werden, ob ein gewisses Paket ein *IP*-Paket oder ein *ARP*-Paket ist. Zu jedem Typ von Paket stehen einem auch protokollentsprechende Methoden zur Verfügung[2].

3.2 Hosts scannen mit NMAP

Durch das passive Mithören und Analysieren der Pakete, erhält man zu jedem Host eine Liste mit den Ports, auf welchen Dienste angeboten werden. Aus diesen Portnummern lässt sich dann vermuten, welche Dienste von einem gewissen Computer angeboten werden[3].

Jedoch gibt es keine Garantie dafür, dass nicht irgendein Dienst auf einer nicht dafür vorgesehenen Portnummer läuft. Man könnte zum Beispiel auf der Portnummer 80 ein Telnet laufen lassen, anstatt eines Webservers. Um sicherzustellen, was für Dienste auf welchen Ports angeboten werden, kann man die im ersten Schritt gefundenen Host mit Nmap scannen. Damit erhält man sehr genaue Daten über welche Dienste auf welchen Ports sind

[2]Die zur Verfügung gestellten Methoden und Klassen sind in der Javadoc von JPCAP unter `http://jpcap.sourceforge.net/javadoc/` zu finden.

[3]Dies bezieht sich auf TCP / IP. Siehe TANENBAUM *Computer Networks*.

(inklusive Name und Versionsnummer des Dienstes)[4]. Des weiteren werden auch solche Dienste entdeckt, welche während dem Abhören im ersten Schritt nicht gebraucht wurden.

Da es für den direkten Aufruf von NMAP keinen Javawrapper gibt, exportiert NMAP die vom Portscan erhaltenen Resultate in eine XML-Datei. Diese wird anschliessend vom Network Sniffer eingelesen und in die interne Darstellung von VERINEC umgewandelt. NMAP wird dazu von Java aus mit *verinec.util.ProgramExec* aufgerufen.

3.3 Aufbau des Programms

Im Hauptverzeichnis des Programms gibt es folgende Ordnerstruktur (dabei sind die Namen der Unterordner auf der linken Seite und auf der Rechten eine Beschreibung des Inhalts):

Ordner	Beschreibung
bin	Binäre Dateien für NMAP.
doc	Die Dokumentation zum Network Sniffer.
install	Die Installationsroutine um WINPCAP für Windows zu installieren.
java	Der Inhalt ist weiter unten ausführlich erklärt.

Der Ordner java seinerseits ist wie folgt unterteilt:

Ordner	Beschreibung
doc	Ordner mit der Java Dokumentation des Netzwerk Sniffers.
jars	Darin befinden sich die für das Programm benötigten *jar*-Dateien und Libraries.
lib	Die erzeugte *jar*-Datei des Network Sniffers.
output	Die kompilierten Klassen.
src	Der gesamte Quellcode des Sniffers.

Zur Funktionsweise des Programms:

Der Netzwerk Sniffer wird als Modul von *verinec.networkanalyser.Analyser* gestartet. Analyser ist der zentrale Teil vom VERINEC, es wird eine graphische Oberfläche angeboten, von wo aus alle weitere Module gestartet werden können. Die Klasse *SnifferModule* implementiert dabei die verlangten

[4]Zur Funktionsweise von Nmap siehe FYODOR *The Art of Port Scanning* und FYODOR *Remote OS detection via TCP/IP Stack FingerPrinting*.

Methoden aus *IVerinecModule* (diese Methoden müssen von allen Modulen implementiert werden). Beim Laden des Moduls wird eine GUI gestartet[5], um die Einstellungen zum Sniffen vorzunehmen. Dies wird durch die Klasse *ConfigPanel* bewerkstelligt.

Die Klasse *ConfigPanel* benötigt intern eine Struktur, um die vorgenommenen Einstellungen der Netzwerkkarten abzuspeichern. Diese wird durch *Configure* bereitgestellt. Um dann auch das Speichern und Laden von einer Configdatei zu ermöglichen, wird von *Configure* eine *Document*-Struktur gebraucht, welche recht einfach in eine *XML*-Datei übertragen werden kann. Dazu wird JDOM verwendet, welches Methoden um *XML* zu handhaben bereitstellt.

Wird der Sniffer durch das Drücken des Knopfes *Run* beim *ConfigPanel* gestartet, so wird das von der GUI erstellte *Configure*-Objekt *StartSniffer* übergeben. Dies erstellt für jede aktive Netzwerkkarte ein *Sniffer*-Objekt. Solch ein Objekt ist für die ganze Snifferaktivität verantwortlich. Als erstes wird ein Thread gestartet, welcher sich um das eigentliche sniffen, also empfangen und analysieren der Pakete kümmert.

Solch ein *SnifferThread* interagiert direkt mit JPCAP. Dies wird dadurch erreicht, dass *SnifferThread* die Methode *packetArrived* aus *PacketListener* implementiert. Damit jpcap diese Methode aufrufen kann, muss im *SnifferThread* ein *PacketCaptureCapable*-Objekt erstellt werden. Dieses Objekt ist auch nötig, um JPCAP die nötigen Parameter über die Art des Sniffens (mehr dazu im Kapitel Gebrauch) mitzuteilen.

Falls ein Paket an der Netzwerkkarte ankommt, gibt dies JPCAP über die Methode *packetArrived* an *SnifferThread* weiter. Diese Methode nimmt schon mal die grobe Analyse eines Paketes vor und entscheidet dabei, ob es sich um ein *TCP-, UPD-, ICMP-* oder ein *ARP*-Paket handelt. Die Pakete werden danach den entsprechenden Methoden für die weitere und genaue Analyse weitergegeben. Die meist gebrauchte Methode ist dabei wohl jene, welche *TCP*-Pakete analysiert. Dabei wird versucht, aus einem Paket den Server und den benutzten Port zu bestimmen, um so die laufenden Dienste auf einem Server zu erhalten. Die Schwierigkeit besteht dabei, dass man aus Absender und Empfänger eines Paketes entscheiden muss, wer einen Dienst anbietet. Wenn man dazu nicht jedes einzelne Protokoll analysieren will, kann dies nicht in jedem Fall eindeutig bestimmt werden.

Die Entwickler von JPCAP raten daher, den Netzverkehr erst dann zu beginnen, nachdem der Sniffer läuft. Somit kann dann angenommen werden, dass ein erstes *Request*-Paket eine Anfrage eines Clients an einen Server ist. Ich habe versucht, dies ein bisschen zu verbessern. Bei einem *Request* wird

[5]Genaueres dazu lässt sich im Kapitel Gebrauch finden

auch aufgrund der benutzten Portnummer entschieden, wer den Dienst anbietet. Dies geschieht dadurch, dass für eine Anfrage auf Serverseite einen niederen Port (z.B. Nummer 80 für ein *HTTP-Request*) benutzt wird. Dies trifft wohl für die meist gebrauchten Protokolle zu.

Für jeden im *SnifferThread* gefundenen Host wird ein *HostCommRenderer*-Objekt erstellt. In solch einem Objekt werden alle wichtigen Daten über einen Host gespeichert. Dazu zählen einerseits die genauen Daten wie IP-Adresse oder Hostname, aber auch alle gefundenen Dienste (falls der Host ein Server ist). All diese Objekte werden als eine Art Datenbank in einer *HashMap* gespeichert und sind so bei Bedarf auch sehr schnell abrufbar. Falls ein weiteres Paket von einem schon erkannten Host eintrifft, so wird das entsprechende *HostCommRenderer*-Objekt gesucht und die neuen Werte übergeben.

Die *HashMap* ist im Snifferobjekt zu finden, da sie zwar vom *SnifferThread* erstellt, aber später noch weiterverwendet wird. Ist das Sniffen abgeschlossen, so kann vom Snifferobjekt noch ein *HostScanThread* gestartet werden, welcher die Informationen über die gefundenen Hosts mit Hilfe von NMAP überprüft und die Objekte in der *HashMap* ergänzt.

Das Package *verinec.networksniffer.simulator* wurde komplett von JPCAP übernommen. Zweck dieses Packages ist es, den Netzverkehr einer Netzwerkkarte zu simulieren. Dies kann zum Beispiel zum Testen des Programms sehr hilfreich sein. Es gibt dazu eine Konfigurationsdatei welche sich unter *java/jars/simulator.properties* befindet. Darin lassen sich jegliche Einstellungen bezüglich des Simulators vornehmen (i.e. was für Pakete mit welcher Wahrscheinlichkeit generiert werden sollen).

4 Benutzung

4.1 Voraussetzungen

Um den Network Sniffer gebrauchen zu können, müssen folgende Programme oder Dateien auf dem Computer installiert sein:

- Um Java Programme ausführen zu können, muss eine *Java Runtime Environment* (Version 1.4.2 oder höher) vorhanden sein.
- Um Pakete sniffen zu können, muss eine entsprechende Library verfügbar sein. Auf Windows ist dies WinPcap, beziehungsweise libpcap für Linux.
- Falls der Network Sniffer auf Linux laufen soll, so sollte Nmap installiert sein, oder zumindest alle von Nmap benötigten Libraries. Windowsbenutzer müssen sich nicht um Nmap kümmern, da dies als ausführbare Datei im Network Sniffer enthalten ist.

4.2 Installation

Wenn Sie den Network Sniffer unter Windows benutzen wollen, so installieren Sie bitte folgende Programme (die entsprechende URL steht jeweils daneben).

- Java Runtime Environment, `http://java.sun.com`
- WinPcap, `http://winpcap.polito.it`. Die Installationsroutine befindet sich auch im Ordner *install*.

Falls Sie Linux benutzen, so ist die Installation ein wenig anders. Sie benötigen hierfür nicht das oben genannte WinPcap, sondern die Library libpcap[6]. Falls Sie auch aktiv die Ports der gefundenen Hosts überprüfen wollen, so empfiehlt es sich, Nmap zu installieren. All dies installieren Sie am besten auf die Art und Weise, wie es Ihre Distribution empfiehlt[7].

Es ist nicht nötig, jpcap noch separat zu installieren, da dies schon als *jar*-Datei im Projekt enthalten ist. Zum Starten des Sniffers stehen die Dateien *run.bat* für Windows, beziehungsweise *run.sh* für Linux im Ordner *NetworkSniffer* auf der CD zur Verfügung.

[6]Sollte es mit dieser Library Probleme geben, so besteht die Möglichkeit sie neu zu kompilieren. Mehr dazu auf Seite 13.

[7]Jede Distribution sollte ein Paketmanagment haben, falls Sie sich damit nicht auskennen, so finden Sie in der Dokumentation Ihrer Distribution weitere Informationen. Unter Debian nennt sich das *apt-get install* oder Yast für Suse.

4.3 Installation zu bestehendem Analyser

Falls Sie den Network Sniffer nachträglich zu einem bereits installierten Analyser installieren möchten, so kopieren Sie einfach den Ordner des Sniffers neben jenen des Analysers.

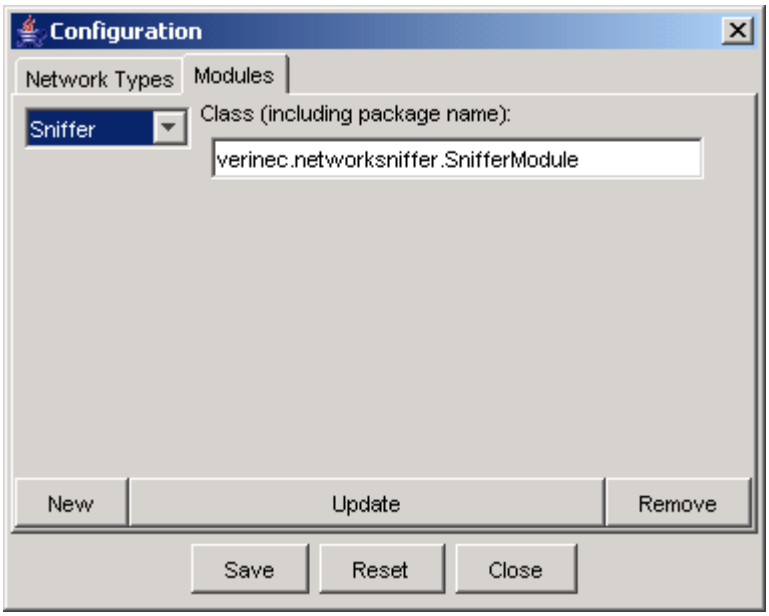

Abbildung 1: Die Konfiguration von Modulen

Falls Sie danach den Analyser starten, so klicken Sie im Menüpunkt *Preferences* auf *Options*. Danach erscheint das Fenster, welches in Abbildung 1 zu sehen ist. Unter *Modules* kann durch klicken auf *Update* nach neuen Modulen gesucht werden. Falls das Modul gefunden wurde, so sollte dies wie in der obigen Abbildung dargestellt werden. Bitte starten Sie den Sniffer so wie unten erklärt wird.

4.4 Gebrauch

Leider können Sie das Programm nicht ausführen, wenn Sie nicht über Administrator Rechte verfügen, da es nicht möglich ist, JPCAP und NMAP als normaler Benutzer auszuführen. Dies dient dazu, die im Kapitel Allgemeines angesprochenen Sicherheitsprobleme einzudämmen. Falls Sie trotzdem das Programm mit eingeschränkten Benutzerrechte starten, so erscheint eine Fehlermeldung. Es ist in einem solchen Fall immer noch möglich, Einstellungen vorzunehmen, aber es ist nur möglich Pakete zu simulieren.

Den Network Sniffer starten Sie, indem sie die Datei *run* aus dem Ordner *NetworkSniffer* ausführen. Dabei wird eine allgemeine VERINEC-Oberfläche gestartet von welcher *Sniffer* als Modul gestartet werden kann. Im Menüpunkt *Sniffer*, finden Sie *Start Sniffer*. Wenn Sie dies auswählen, so sollten Sie ein Fenster erhalten, wie in Abbildung 2 zu sehen ist.

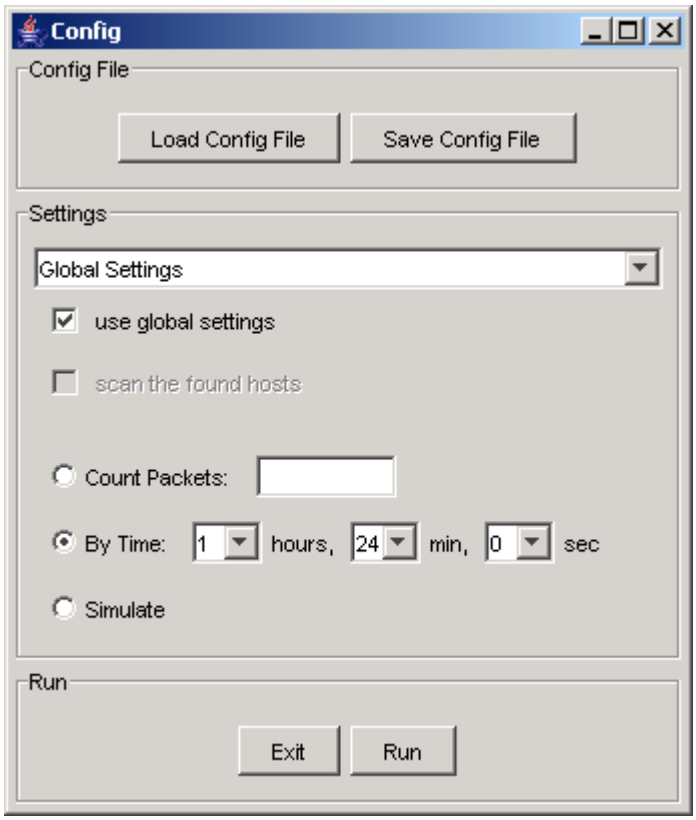

Abbildung 2: Das ConfigPanel – allgemeine Einstellungen

In diesem Fenster werden die genauen Angaben erstellt, wie lange das Programm die Pakete sniffen soll. In der Mitte sind erst einmal die Allgemeinen Einstellungen (*Global Settings*) zu sehen. Dort kann angegeben werden, ob für jede Netzwerkkarte die gleichen Einstellungen gelten (dies ist der Fall, wenn das Kästchen *use global settings* aktiviert ist). Dabei kann man wählen, ob man eine gewisse Anzahl Pakete, oder eine bestimmte Zeit lang Pakete analysieren will. Des weiteren gibt es die Möglichkeit, Pakete zu simulieren. Dies ist eine Funktion um zu testen, wie das Programm funktioniert. In Abbildung 2 sind die Einstellungen so, dass der Verkehr auf allen Netzwerkkarten eine Stunde und 24 Minuten lange abgehört wird.

Im oberen Teil des Fensters können die gemachten Einstellungen abgespeichert (zum Beispiel um bei einem nächsten Start des Programms nicht wieder von Vorne zu Beginnen) oder geladen werden.

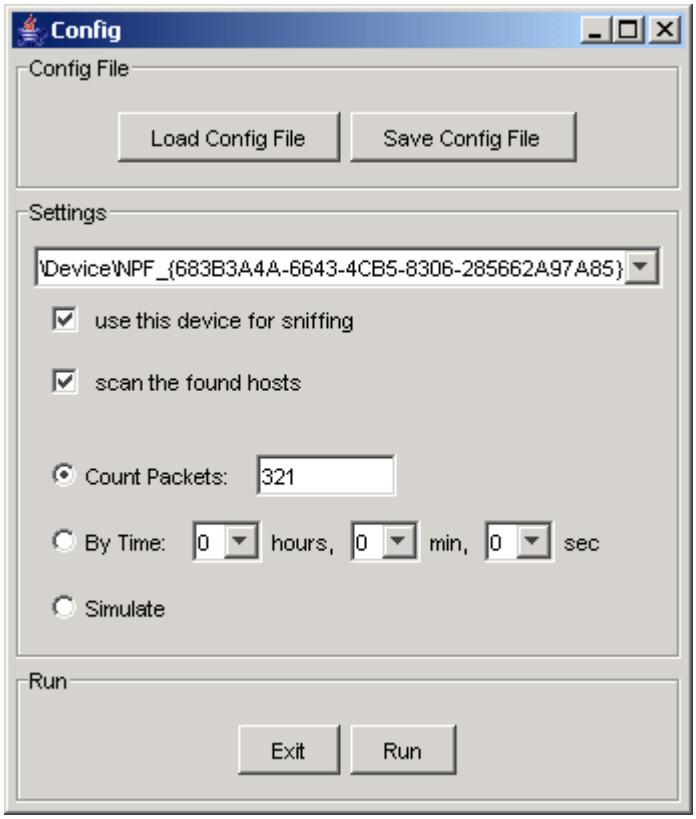

Abbildung 3: Das ConfigPanel – Einstellungen einer Netzwerkkarte

Neben diesen Einstellungen kann auch noch jede vorhandene Netzwerkkarte individuell konfiguriert werden, wie dies in Abbildung 3 zu sehen ist. Falls keine globalen Einstellungen erwünscht sind, so muss für jede einzelne Netzwerkkarte angegeben werden, wie lange gesnifft werden soll.

Ob nun globale oder einzelne Einstellungen vorgenommen werden, es ist immer möglich, die beiden Kontrollkästchen *use this device for sniffing* und *scan the found hosts* zu verändern. Dabei kann angegeben werden ob diese Netzwerkkarte gebraucht werden soll oder nicht. Falls das zweite Kästchen markiert ist, so werden alle gefundenen Hosts durch einen Portscan[8] überprüft.

[8]Dies wird mit Hilfe von Nmap erreicht. Bitte beachten Sie dazu die Installationshinweise auf Seite 8.

Wenn die gefundenen Hosts nicht überprüft werden, so erhält man eine Liste mit Diensten, die auf dem entsprechenden Computer laufen. Dies sind jedoch nur die Portnummern, es gibt also keine Garantie, dass auf einem gewissen Port auch genau der Dienst läuft, der laufen sollte. Mit Hilfe von Nmap kann genau ermittelt werden, welcher Dienst auf welchem Port wirklich läuft und es werden auch die Dienste gefunden, welche während dem Sniffen nicht gebraucht wurde.

Nachdem alle Einstellungen wie gewünscht vorgenommen wurden, kann mit dem Sniffen begonnen werden, indem Sie einfach auf den Knopf *Run* drücken.

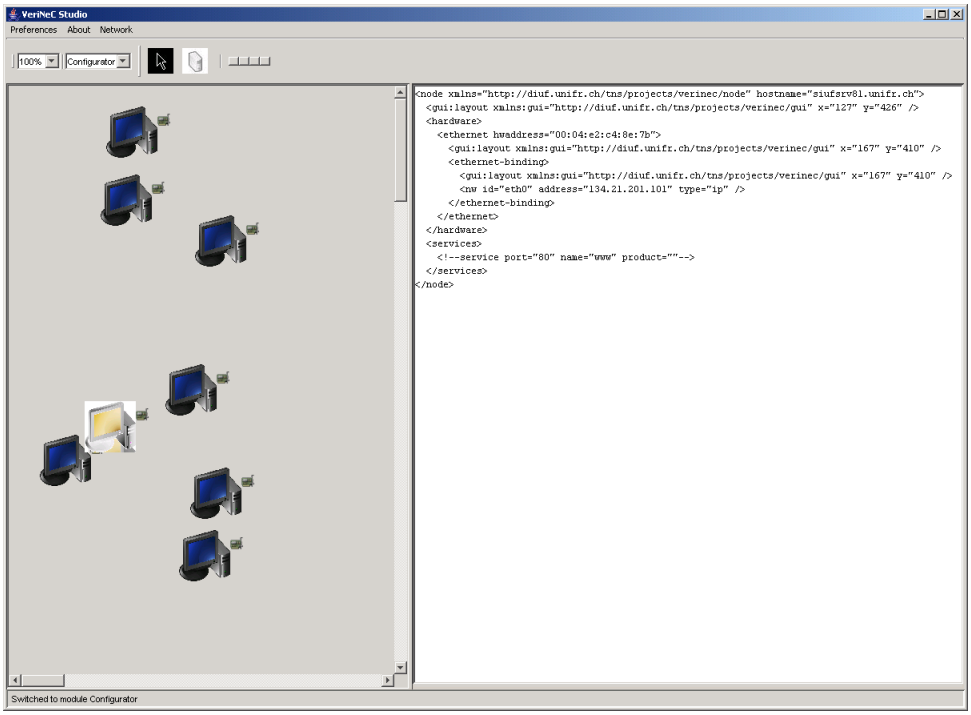

Abbildung 4: Analyser – Resultate des Sniffers

Nach dem Sniffen, werden alle Resultate im *Analyser* dargestellt. Wie in Abbildung 4 zu sehen ist, werden die *nodes* auf der linken Seite dargestellt. Wenn ein *node* ausgewählt wird, so werden auf der rechten Seite die genauen Spezifikationen dargestellt. Unter dem Eintrag *services* werden die gefundenen Dienste angezeigt. Zur Zeit werden jene aber nur als Kommentar eingefügt, da VERINEC noch nicht mit diesen umgehen kann. Die Zeichenkette für *product* ist hier noch leer. Dies weist darauf hin, dass NMAP nicht verwendet wurde.

Wie gewohnt lassen sich die gefundenen *nodes* nach belieben verändern. Sollte ein wichtiges Netzwerk Device fehlen, so lässt sich dies ohne Probleme noch anfügen.

Es gibt zur Namensgebung der Netzwerkkarten im Network Sniffer noch eine kleine Anmerkung. Von JPCAP erhält man für eine Netzwerkkarte immer zwei Devices; ein generic und eine spezifische. Um zu sniffen, sollte nicht ein generic device ausgewählt werden, da auf diesen keine Pakete zu empfangen sind.

Nachdem der Sniffer seine Arbeit erledigt hat, werden die Resultate an den *Analyser* zurückgegeben. Es werden dann auf der linken Seite des Fensters die Geräte und auf der rechten Seite die Spezifikationen angezeigt.

4.5 jpcap für Linux

Unter Linux wird die Library *libjpcap.so* welche im Verzeichnis *java/jars* zu findet ist, für den Gebrauch von JPCAP verwendet. Falls man mit dieser Library nicht zufrieden ist, oder sie auf einem System nicht laufen sollte, so besteht die Möglichkeit, diese neu zu kompilieren. Die Quelldateien sind im oben genanntem Verzeichnis unter dem Namen *jpcap-0.01.15.tar.gz* zu finden.

Um JPCAP neu zu kompilieren, muss man diese Datei entpackt werden. Danach müssen folgende Befehle ausgeführt werden:

```
$ ant clean
$ ant
```

Wichtig ist jedoch, dass davor überprüft wird, ob die dazu erforderlichen Libraries *libpcap* und *libpcap-dev* installiert sind. Des weiteren muss die Umgebungsvariable *JAVA_HOME* auch korrekt gesetzt sein.

5 Ausblick

5.1 Limiten

Das Ziel des Network Sniffers war es, eine Konfiguration zu einem bestehenden Netzwerk aufgrund vom Aufzeichnen des Netzverkehrs zu erstellen. Leider ist es nicht möglich dies ohne Einschränkungen zu erreichen.

Da die Hosts nur durch passives mithören ermittelt werden, ist es nicht möglich, Hosts zu ermitteln, welche in der Zeit, in der der Sniffer aktiv war, nicht kommunizierten. Dies ist jedoch auch nicht so tragisch, wenn über längere Zeit gesnifft wurde, denn in diesem Falle könnte man annehmen, dass dieser Host nicht relevant ist und zukünftig nicht mehr gebraucht wird. Das ist jedoch nicht ein Problem der Implementierung sondern liegt in der Natur des Sniffens.

Leider kann ich nicht garantieren, dass das Programm einwandfrei läuft unter hoher Netzwerklast, da ich keine Möglichkeit hatte, dies zu testen. Ich nehme jedoch an, dass ein gewisser Paketverlust auftreten muss, da Java wohl nicht schnell genug sein wird (zum Beispiel wenn man den Sniffer zwischen zwei Gigabyte Uplinks zweier Switches schaltet). Diese Annahmen bezieht sich auf herkömmliche Computer, dies kann durch den Einsatz von spezieller Hardware welche zur traffic-Analyse konzipiert ist, umgangen werden.

Ein nicht ganz unwichtiger Fehler ist die Genauigkeit der Zuordnung von Client und Server[9]. Es war mir nicht möglich, aus den empfangenen Paketen zu sagen, wer genau den Dienst anbietet. Dies ist kein Problem, wenn die gesamte Netzwerkaktivität erst beginnt, nachdem der Sniffer gestartet wurde. Wird dies nicht eingehalten und zudem noch relativ exotische Protokolle benutzt (zum Beispiel mit einer hohen Portnummer serverseitig) so ist die Fehlerquote am höchsten. Andererseits ist die Zuordnung recht genau bei üblichen Protokollen und den genannten Benutzungseinschränkung[10].

Ein weiteres Problem besteht darin, dass Router welche sich zwischen Hosts befinden, nicht erkannt werden. Zudem werden *nodes*, welche über mehrere Netzwerkkarten verfügen, als mehrere Geräte erkannt, da die Identifikation über die Adresse stattfindet.

Beim Testen der Arbeit auf einem anderen Computer, musste mit Schrecken festgestellt werden, dass JPCAP in Verbindung mit Windows XP und installiertem Service Pack 1, leider nicht läuft. Dabei stellt sich die Frage, ob dies ein Fehler von Microsoft oder den Entwickler der Library ist. Vielleicht wird dieser Fehler mit der nächsten Version von JPCAP behoben.

[9] Siehe dazu auch die Problembeschreibung auf Seite 6.

[10] Einen Hinweis auf dieses Problem ist auch in den Quelldateien von JPCAP zu finden.

5.2 Verbesserungen

Eine Verbesserung könnte durch das Einsetzen von *traceroute* stattfinden. Somit könnte der Sniffer aktiv versuchen, genaueres über die Struktur des Netzwerkes herauszufinden. Dies könnte sehr nützlich sein, um im Analyser die genaue Anordnung der Nodes darzustellen und Informationen über Router welche sich im Netzwerk befinden zu erhalten.

Wird der Siffer auf einem Computer mit mehreren Netzwerkkarten ausgeführt, so könnte ein Test ausgeführt werden, welcher nach *nodes* sucht, die doppelt vorkommen. Das wären *nodes* welche von mehreren Netzwerkkarten erkannt wurden. Dieser Test könnte eventuell noch überprüfen, ob es *nodes* gibt, welche mehrere Netzwerkkarten besitzen (das könnte der Fall sein, wenn mehrere *nodes* den gleichen Namen aufweisen).

Was für das Optische nicht schlecht wäre, ist, wenn es möglich wäre, die nach dem Sniffen im Analyser dargestellten *nodes* auch korrekt zu verbinden, um so die exakte Netzwerktopologie darzustellen.

Referenzen

EtherApe: A graphical network monitor. `http://etherape.sourceforge.net/`, November 2004.

Ethereal: A network protocol analyzer. `http://www.ethereal.com`, November 2004.

Fyodor: The Art of Port Scanning. `http://www.phrack.org/show.php?p=51&a=11`, November 2004.

Fyodor: Remote OS detection via TCP/IP Stack FingerPrinting. `http://www.insecure.org/nmap/nmap-fingerprinting-article.txt`, November 2004.

jdom: A Java-based solution for accessing, manipulating, and outputting *XML* data from Java code. `http://www.jdom.org`, November 2004.

jpcap: Network packet capture library for applications written in Java. `http://sourceforge.net/projects/jpcap`, November 2004.

libpcap: A system-independent interface for user-level packet capture. `http://sourceforge.net/projects/libpcap/`, November 2004.

Nmap: A free open source utility for network exploration or security auditing. `www.insecure.org/nmap/`, November 2004.

Tanenbaum, Andrew S.: Computer Networks. 4. Auflage. Amsterdam, 2003.

Verinec: Verified Network Configuration - A TNS Research Project. `http://diuf.unifr.ch/tns/projects/verinec/`, November 2004.

WinPcap: The free packet capture library for Windows. `http://winpcap.polito.it`, November 2004.