

# Sicherer Zugang zu privaten Diensten über Virtual Service Networks

Carsten Link, Norbert Luttenberger

Institut für Informatik und Praktische Mathematik  
Christian-Albrechts-Universität zu Kiel  
{cli,nl}@informatik.uni-kiel.de

**Zusammenfassung.** In vielfältigen Situationen benötigen mobile Nutzer Zugang zu Netzwerkdiensten und Daten, die in einem privaten Intranet, z.B. dem Intranet ihres Arbeitgebers, angeboten werden. Ein Zugang über das öffentliche Internet darf sinnvollerweise aber nur dann eröffnet werden, wenn dadurch die Integrität, Authentizität und Vertraulichkeit der ausgetauschten Daten nicht gefährdet wird und das gesamte Intranet nicht verstärkt externen Angriffen ausgesetzt wird. In dieser Arbeit werden *Virtual Service Networks* vorgestellt, in denen diesem Problem auf zweierlei Weise begegnet wird: Zum einen bleiben Netzwerkdienste solange unsichtbar und damit unangreifbar, bis ein Nutzer authentifiziert und autorisiert worden ist, zum anderen werden in der Autorisierungsentscheidung die verschiedenen Nutzungskontexte, die in der Regel mit unterschiedlichen Gefährdungspotentialen verbunden sind, berücksichtigt, so daß dem Nutzer nur die Privilegien erteilt werden, die für die jeweilige Situation angemessen sind. Dienste werden also nicht uniform angeboten, sondern als sog. *Polymorphic Services*, die dynamisch an die jeweilige Situation angepaßt werden.

## 1 Einleitung

Das heutige Internet ist kein homogenes Netz, sondern ein Netz, das in einen öffentlichen Teil und viele private Teile („Intranets“) aufgespalten ist. Netzwerkdienste werden im öffentlichen und in privaten Netzen angeboten; wir bezeichnen solche Dienste entsprechend als öffentliche und private Dienste. Während öffentliche Dienste anonym genutzt werden können, werden private Dienste von Organisationen ausschließlich für ihre Mitglieder bereitgestellt, und damit müssen Benutzer vor der Nutzung privater Dienste authentifiziert und autorisiert werden.

Wachsende Mobilität bringt es mit sich, daß auf private Dienste auch von Orten aus zugegriffen werden soll, die außerhalb des privaten Bereichs liegen. Das bedeutet,

- daß an die Implementierung privater Dienste erhöhte Anforderungen in puncto Sicherheit gestellt werden müssen, damit diese nicht korrumpiert werden können, wenn sie zusätzlich auch über das Internet verfügbar gemacht werden;

- daß Authentifizierungs- und Autorisierungsmechanismen wirksam eingesetzt werden müssen.

Authentifizierungsprotokolle sind gut untersucht, eine Vielzahl solcher Protokolle existiert und ist im praktischen Einsatz. Vor allem kommen hier kryptographische Protokolle zum Einsatz, die auf der *public key*-Kryptographie basieren. Forschungen zum Thema Autorisierung haben sich insbesondere mit rollenbasierten Ansätzen beschäftigt (siehe Übersicht in [5]). Durch die Unterscheidung von verschiedenen Rollen, die eine Person einnehmen kann, soll das *least privilege principle* durchgesetzt werden, d.h. ein Benutzer kann nur auf die Ressourcen zugreifen, für die entsprechende Zugriffsrechte in der Rolle vorgesehen sind. Darüberhinaus kann der Nutzer durch Rollenaktivierung/-deaktivierung mit minimalen Privilegien agieren.

Wirksame Authentifizierungsprotokolle und Autorisierungsverfahren haben Angreifer allerdings nicht davon abgehalten, Angriffe auf Netzwerkdienste durchzuführen. Es gibt *Denial-of-Service*-Angriffe, bei denen Systeme gezielt überlastet werden, oder Angriffe über *Application Bugs and Backdoors*, d.h. über Schwachstellen in Implementierungen, die im Laufe der Zeit bekannt geworden sind und von den Systemadministratoren nicht immer rechtzeitig mit Hilfe entsprechender *Patches* ausgemerzt werden. Beide Arten von Angriffen gehen an den oben erwähnten Authentifizierungs- und Autorisierungsverfahren „vorbei“ und verschaffen ggf. einem Angreifer Zugang zu einem Rechensystem, ohne sich authentifizieren und autorisieren zu lassen. Es stellt sich also die Frage, ob es angesichts der oben festgestellten erhöhten Sicherheitsanforderungen, die aus der mobilen Nutzung von privaten Diensten resultieren, nicht sinnvoll ist, auch über eine veränderte Strategie für die Implementierung von Netzwerkdiensten nachzudenken.

Wir meinen, daß diesbezüglich ein zweifacher Handlungsbedarf besteht:

1. Authentifizierung und Autorisierung sind vom Dienst zu trennen; der Dienst als solcher darf erst nach erfolgreicher Authentifizierung und Autorisierung sichtbar werden. Damit kann das Problem der sicheren Implementierung auf das Authentifizierungs- und Autorisierungsmodul reduziert werden. Dieses Modul stellt dabei das einzige exponierte Element des Systems dar und ist somit alleiniger Angriffspunkt.
2. Mobile Benutzer („Computer-Nomaden“) suchen Zugang zu Netzwerkdiensten aus unterschiedlichen Situationen heraus. Oftmals (aber nicht immer!) sind diese Situationen durch einen Aufenthaltsort charakterisierbar. Ein Anbieter von Netzwerkdiensten wird sich vergewissern wollen, in welche Situation hinein er einen Dienst verfügbar macht: Es ist eben ein Unterschied, ob die Nachfrage nach einem vertraulichen Dokument von einem heimischen Arbeitsplatz kommt, von einem Gastarbeitsplatz bei einem Kooperationspartner oder nachts um 23:30 aus einem Internet-Café. Neben dem rollenbasierten Ansatz sollte für die Autorisierung auch ein *kontextabhängiges Autorisierungsverfahren* bereitgestellt werden.

Gegenwärtige Verfahren und Applikationen gehen nicht auf Verschiedenartigkeit der Nutzungssituationen ein. Sie treffen eine Ja/Nein-Entscheidung und geben dem Nutzer entweder Vollzugriff oder gar keinen Zugriff. Es ist unabdingbar, bei der Bereitstellung von privaten Diensten das *least privilege principle* erweitert zu deuten, so daß nicht nur

die mit Rollen verbundenen Privilegien, sondern auch die Situation berücksichtigt wird, in der sich ein Rolleninhaber befindet.

Dieser Artikel präsentiert einen systematischen Ansatz, in dem die kontextabhängige Autorisierung eine zentrale Rolle spielt. Im Moment der Anfrage eines Dienstes wird die Situation erfaßt und die Mittel zur Durchsetzung der Schutzziele werden bestimmt. Dienste werden daraufhin dynamisch erzeugt und dabei an die Situation, in der die Nutzung stattfinden soll, angepaßt.

Der nächste Abschnitt erläutert das Konzept der *Virtual Service Networks* und der darin dynamisch erzeugten *Polymorphic Services*. In Abschnitt 3 wird die Infrastruktur des VSN-Systems beschrieben. Der Abschnitt 4 erklärt die Ablaufumgebung für *Polymorphic Services*. Darauf folgt eine detaillierte Beschreibung spezieller *Polymorphic Services* (Proxy-basierte *Polymorphic Services*), die die Dienstnutzung überwachen. Der Artikel endet mit einem Vergleich des Ansatzes zu anderen Arbeiten und einem Ausblick.

## 2 Virtual Service Networks

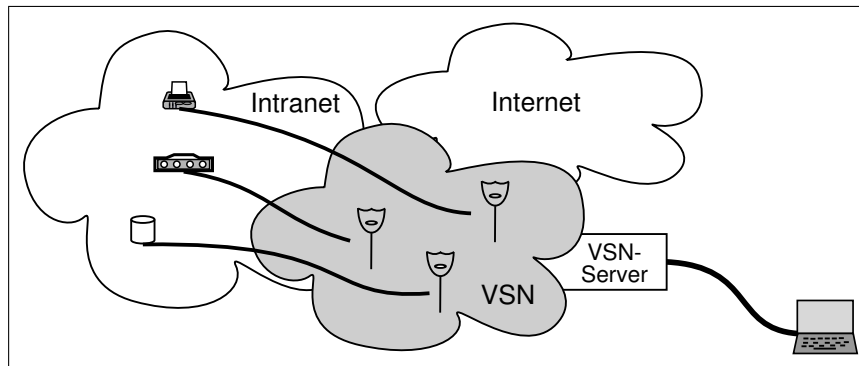
### 2.1 Konzept

Als Umgebung für die kontextabhängige Autorisierung wurde das neue Konzept der *Virtual Service Networks* (VSN) entwickelt. Die Arbeitsweise von VSNs läßt sich durch zwei miteinander verbundene Elemente kennzeichnen:

1. In VSNs werden Netzwerkdienste erst nach Anforderung durch einen Client in objektorientierter Manier dynamisch instanziiert.
2. Bei der Instanzierung eines Netzwerkdienstes wird der Kontext des Client mit berücksichtigt: Es wird eine jeweils an den Kontext des Client angepaßte *Dienstvariante* erzeugt.

Zur Benutzung der in einem VSN vorgehaltenen Dienste muß sich ein Nutzer authentifizieren, z.B. durch den Aufbau eines kryptographischen Tunnels zum VSN-Server, der „Eingangsstation“ eines VSN. Damit erhält er Zugang zu einem vorerst „leeren“ Netz. Die vom Nutzer benötigten Dienste werden erst nach erfolgreicher Authentifizierung und Anforderung erzeugt. Dadurch unterscheiden sich die in VSNs angebotenen Dienste von der herkömmlichen Dienstrealisierung durch Hintergrundprozesse, die in einem System ständig präsent sind. Diese Vorgehensweise hat den Vorteil größerer Sicherheit, da Dienste, die nicht existieren, offensichtlich auch nicht angegriffen oder mißbraucht werden können.

Dienste in VSNs sind nicht statisch konfiguriert, sondern werden bei ihrer dynamischen Erzeugung an die Umstände angepaßt. Es sind dynamisch instanziierte Objekte (im objektorientierten Sinne), bei deren Instanzierung der Kontext des Client mit berücksichtigt wird. Es wird also nicht ein uniformer Netzwerkdienst instanziiert, sondern eine jeweils an den Kontext des Client angepaßte *Dienstvariante*. Dadurch erfolgt implizit eine kontextabhängige Autorisierung des Client. Insgesamt resultiert daraus ein polymorphes Verhalten des Dienstes; entsprechend bezeichnen wir solche Dienste als



**Abb. 1.** Ein Virtual Service Network. Der Rechner des Nutzers ist über einen kryptographischen Tunnel mit dem VSN-Server verbunden.

*Polymorphic Services.* Zu einem VSN gehört ein Regelwerk des Diensteanbieters, mit dem er Kontextinformationen auf das Verhalten von Polymorphic Services abbildet.

In Abbildung 1 ist ein VSN zu sehen, welches Zugriff zu privaten Diensten eines Intranets vermittelt. Polymorphismus wird von den venezianischen Masken symbolisiert: stabiles und bekanntes Interface nach außen, verschiedenartiges Verhalten im Inneren. Zusammengenommen machen die Dynamik bei der Erzeugung von Services und die Trennung von Authentifizierung und Autorisierung VSNs zur Plattform, auf der Dienste abhängig von Situationsattributen (Kontextinformation) kontrolliert angeboten werden und dem Diensteanbieter ein hohes Maß an Sicherheit und Kontrolle über die Dienstnutzung verschafft.

## 2.2 Polymorphic Services

Ein polymorpher Dienst wird durch ein Java-Objekt implementiert und hat daher die aus den Konzepten der objektorientierten Programmierung bekannten Eigenschaften von Objekten. Ein Service

1. durchläuft einen definierten Lebenszyklus,
2. ist Instanz einer Klasse,
3. implementiert ein Interface,
4. hat einen Zustand und
5. zeigt ein Verhalten.

Der Lebenszyklus eines Services beginnt mit der Aktivierungsanforderung eines authentifizierten Benutzers. Der Zyklus endet mit der Deaktivierung oder implizit mit der Abmeldung des Nutzers. Ein Service ist Instanz einer Klasse, welche ein Interface implementiert. Das Interface beschreibt das vom Service verstandene Kommunikationsprotokoll. Das Verhalten des Services wird von seiner Klasse und von seinem Zustand bestimmt und ist daher polymorph. Bei der dynamischen Erzeugung kann zudem die

Initialisierung (Konstruktorparameter) das spätere Verhalten beeinflussen. Innerhalb eines VSN ist ein Service über eine Adresse erreichbar, welche sich aus den Komponenten IP-Adresse, Protokollkennung und Portnummer zusammensetzen kann. Dadurch ist es möglich, dem Benutzer Dienste auf den Netzwerkebenen 3 bis 7 anzubieten. Die Adresse eines Dienstes ist in ihrer Sichtbarkeit auf das VSN beschränkt und nur gültig, solange der Benutzer angemeldet ist.

Es können polymorphe Dienste unterschiedlicher Natur entwickelt werden. Hier einige Beispiele:

1. Autonomer Dienst: Der Service enthält alle nötigen Daten und Funktionen in sich selbst.
2. Personalisierung: Das Erscheinungsbild und die verfügbare Funktionalität ist auf die Belange des Nutzers zugeschnitten. Es können individuelle Vorlieben und bestimmte Benutzereigenschaften (z.B. Alter, Muttersprache, Sehkraft) berücksichtigt werden.
3. Protokolltransformation: Der Service agiert als Gateway und nutzt zur Kommunikation mit dem Client ein anderes Protokoll als zur Kommunikation mit dem Server, zu dem er Zugang verschafft.
4. Redundanz: Um Verfügbarkeit oder Performance zu erhöhen, nutzt der Service mehrere back-end Server.
5. Entkopplung: Clients, die schwach angebunden sind (z.B. über eine Funkverbindung) und zeitweise nicht mit dem Netz verbunden sind, werden von diesem Service unterstützt bzw. vertreten.
6. Weiterleitung: IP-, TCP- oder UDP-Daten werden vom Service sicher weitergeleitet.
7. Transparenter Proxy: Zur Durchsetzung von Sicherheitsbestimmungen oder zu Simulationszwecken kann der Service Daten modifizieren, duplizieren, protokollieren oder verwerfen.

Der Fokus dieses Artikels ist eine Anwendung eines VSN, in der Transparente Proxies benutzt werden, welche als dynamisch instanziierte *Application Level Gateways* agieren und kontrollierten Zugriff zu privaten back-end Servern erlauben. Proxies können in diesem Modell z.B. verhindern, daß vertrauliche Dokumente an den Benutzer gesendet werden, unabhängig davon, ob dies per e-Mail Anhang oder über ein Netzwerkdateisystem geschehen soll. Vorhandene Netzwerkdienste können mit dem hier präsentierten Ansatz mit einer einheitlichen Rechteverwaltung und Zugriffskontrolle versehen werden.

### **3 VSN-Infrastruktur**

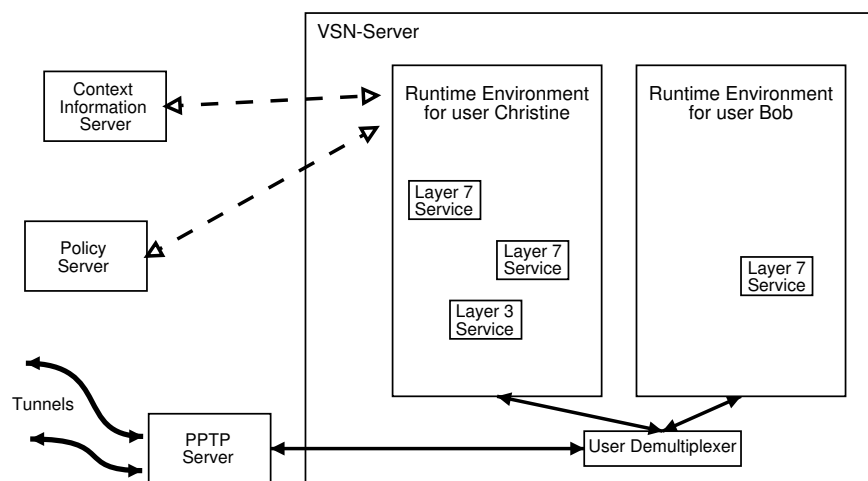
Ein VSN wird von drei Servern gebildet: VSN-Server, Context Information Server und Policy Server. Diese werden in den folgenden drei Abschnitten erläutert.

#### **3.1 VSN-Server**

Das gesamte VSN wird von einem einzigen Java-Programm – dem VSN-Server – erzeugt. Dieser Server emuliert das Virtual Service Network für den Benutzer: Für diesen

hat es den Anschein, als seien im VSN mehrere Hosts mit zugehörigen Serverprozessen. Dennoch werden alle Dienste, die der Nutzer zu sehen bekommen kann, von einem einzigen Java-Programm angeboten.

Um Zugang zum VSN zu erhalten, muß der Benutzer einen verschlüsselten, authentifizierten Tunnel zum VSN-Server aufbauen. Damit wird ein Benutzer authentifiziert, bevor er überhaupt Zugang zu einem VSN bekommen kann. Der gesamte IP-Netzwerkverkehr des kryptographischen Tunnels eines Nutzers endet im VSN-Server. In der derzeitigen Version wird PPTP als kryptographischer Tunnel benutzt. Es sind aber auch andere Tunnelprotokolle wie z.B. L2TP oder IPsec einsetzbar.



**Abb. 2.** Die Komponenten des VSN-Systems: VSN-Server, Context Information Server und VSN-Server. Im VSN-Server wird für jeden Benutzer ein Runtime Environment als Ablaufumgebung für Services erzeugt.

Für jeden Benutzer, der sich bei einem VSN anmeldet, wird vom VSN-Server dynamisch ein eigenes *Runtime Environment* (RTE) erzeugt und mit dem Tunnel des Benutzers verbunden (siehe Abb. 2). Das RTE bildet eine Ablaufumgebung für *Polymorphic Services*. Ohne die Ablaufumgebung existieren keine Dienste für den Benutzer; der Benutzer hätte einen Tunnel in ein leeres Netz. Der VSN-Server beinhaltet einen User-Demultiplexer, der die verschiedenen Benutzer-Datenströme an das jeweils zugehörige RTE weiterleitet. Wird der Tunnel abgebaut, so wird das damit assoziierte RTE mitsamt der darin enthaltenen Services heruntergefahren und aus dem VSN-Server entfernt. Das Zusammenspiel des RTEs und der Polymorphic Services wird in Abschnitt 4 erläutert.

Der VSN-Server muß derart allokiert sein, daß er von allen möglichen Nutzer-Aufenthaltsorten aus erreichbar ist, also z.B. in der Demilitarisierten Zone des Intranet.

### 3.2 Context Information Server

Um dem Benutzer Dienste im VSN maßgeschneidert auf die Situation zu liefern, können Situationsattribute herangezogen werden:

- Aufenthaltsort eines Benutzers
- Eigenschaften des Tunnels zwischen VSN-Server und dem Rechner des Benutzers, z.B. Stärke der Verschlüsselung, Güte der Authentifizierung, Bandbreite
- Systemzustand (Intrusion Detection System)
- ...

Der Aufenthaltsort ist eine wichtige Kontextinformation und kann in den CIS integriert werden. Der Aufenthaltsort kann topologisch (IP-Subnetz), geographisch (Längen- und Breitengrade) oder logisch (Universität) angegeben werden. In [3] werden z.B. digital signierte *location certificates* verwendet, um einem Nutzer den Aufenthalt in einer Nachbarschaft zu einem Location Service zu bescheinigen. Darüberhinaus sind noch weitere Situationsattribute denkbar, die über geeignete Verfahren erfaßt werden müssen.

Der CIS ist ein Frontend für eine Vielzahl von Informationsquellen, die hierdurch eine uniforme Schnittstelle erhalten. Die Komponenten des VSN-Servers, die auf Context Information angewiesen sind, fragen den CIS über das HTTP-Protokoll ab. Kontextinformationen werden daraufhin in Form von XML-Dokumenten geliefert. Das folgende Beispiel zeigt, wie Eigenschaften des Tunnels als Kontextinformation dargestellt werden.

```
GET /TunnelProperties?user=christine HTTP/1.0
```

Die dazugehörige Antwort:

```
HTTP/1.0 200 OK  
Content-type: text/xml
```

```
<?xml version="1.0"?>  
<TunnelProperties>  
  <TunnelType> PPTP</TunnelType>  
  <AuthenticationScheme> Password</AuthenticationScheme>  
  <Encryption>  
    <Algorithm> MPPE</Algorithm>  
    <KeyLength> 40 bit</KeyLength>  
  </Encryption>  
</TunnelProperties>
```

### 3.3 Policy Server

Der Policy Server ist der zentrale Punkt, in dem Autorisierungsentscheidungen getroffen werden. Er umfaßt eine Inferenzmaschine, die Regeln auswerten kann. Der Policy-Service wird als Web-Service mit HTTP-Binding bereitgestellt.

Eine Policy ist ein logisches Programm und definiert die Zugriffsrechte von Subjekten (Nutzer, Gruppen) auf die zu schützenden Objekte. Da der Policy Server eine

Inferenzmaschine enthält, die logische Regeln auswertet, ist es möglich, verschiedene Zugriffskontrollstrategien in einer Policy auszudrücken. Zugriffsrechte können z.B. in der Form von *Access Control Lists* (ACL) oder rollenbasiert (*Role Based Access Control*, RBAC) vergeben werden (siehe [2] und [6]). Im VSN-Ansatz können darüberhinaus die Zugriffsrechte durch den Benutzungskontext weiter eingeschränkt werden. Eine Autorisierungsentscheidung des Policy Server liefert eine Liste von Einschränkungen, der die Dienstonutzung unterliegen soll. Eine Anfrage kann z.B. wie folgt beantwortet werden:

```
HTTP/1.0 200 OK
Content-type: text/xml

<?xml version="1.0"?>
<decision>
  <Skeleton>
    <Type> FTPMonitor</Type>
    <Parameter> backEnd: ftp.comsys.informatik.uni-kiel.de</Parameter>
  </Skeleton>
  <Constraints>
    <pathConstraint>
      <allowed> '/public'</allowed>
    </pathConstraint>
    <fileTypeConstraint>
      <allowed> 'h' '.hpp'</allowed>
    </fileTypeConstraint>
  </Constraints>
</decision>
```

Diese Antwort autorisiert den Nutzer, auf C header im öffentlichen Verzeichnis des FTP Servers zuzugreifen. Alle anderen Zugriffe sind unzulässig. Zugriffe werden von einem Objekt der Klasse *FTPMonitor* (vgl. Abschnitt 5) überwacht. Die Definition einer geeigneten Sprache zur Formulierung der Policy unter Berücksichtigung der Kontextinformation ist derzeit noch Forschungsgegenstand. Eine erste Version findet sich in [1].

## 4 Runtime Environment

Das *Runtime Environment* (RTE) erlaubt es, Dienste verschiedener Ebenen dynamisch zu erzeugen. Es sorgt dafür, daß die Dienste instanziiert werden, leitet den Netzwerkverkehr an diese weiter und bietet den Diensten komfortable Schnittstellen zur Interaktion mit den anderen Komponenten des VSN-Systems.

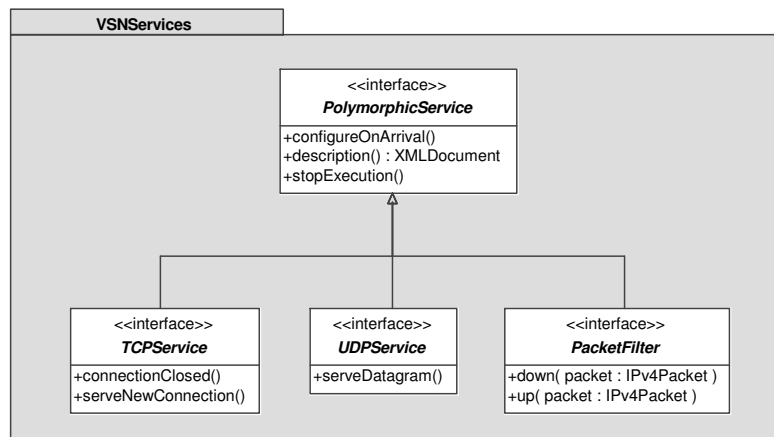
Der User-Demultiplexer des VSN-Servers sorgt dafür, daß die IP-Pakete aus dem kryptographischen Tunnel eines Nutzers in das dazugehörige RTE geleitet werden. Im RTE ist ein eigener TCP/IP-Protocol Stack enthalten. Es wurde bewußt nicht auf den bereits im Kernel vorhandenen Stack zurückgegriffen, um die Sicherheit des VSN-Systems durch Isolierung des VSN vom Wirtsbetriebssystem zu erhöhen. Durch diese Designentscheidung bildet das VSN-System eine abgeschottete Einheit, so daß aus

dem isolierten VSN keine Angriffe auf das Wirtsbetriebssystem und die darauf laufenden Prozesse möglich sind. Ebenso sind die im VSN instanziierten Services ausschließlich über den VSN-Server bzw. den kryptographischen Tunnel erreichbar und folglich nicht von außen angreifbar. Die entwickelte TCP/IP-Implementierung bietet den Polymorphic Services Objekte der Klassen `InputStream` und `OutputStream` als Schnittstelle an.

Das Interface `RuntimeEnvironment` enthält Methoden, die von den Services benötigt werden:

```
public interface RuntimeEnvironment {
    // general service interface
    ContextServer    contextInformationServer();
    PolicyServer    PolicyServer();
    // service registration
    void registerPacketFilter(PacketFilter filter);
    void registerUDPService(UDPService service, int port);
    void registerTCPService(TCPService service, int port);
    // other Methods ...
}
```

Services können den Policy Server und den Context Information Server bei Bedarf über das RTE-Interface abfragen. Die register-Methoden werden von Services während ihrer Initialisierung zur Anmeldung benötigt. Weitere Methoden wurden weggelassen, um die Übersichtlichkeit zu erhöhen.



**Abb. 3.** Die Interfacehierarchie der Polymorphic Services.

Polymorphic Services implementieren ein Interface aus dem Java-Package `VSNServices` (Abbildung 3). Die darin deklarierten Interfaces `TCPService`, `UDPService`

und `PacketFilter` spezialisieren das Interface `PolymorphicService` und bieten ihre Dienste auf Transport- bzw. Netzwerkschicht an. Die Interfaces deklarieren im wesentlichen callback Methoden, die vom RTE aufgerufen werden.

Das RTE hat neben der bereits erwähnten Funktionalität zwei weitere Aufgaben: Zunächst bietet es eine Web-Oberfläche, auf der die für den jeweiligen Benutzer zugänglichen Dienste angeboten und beschrieben werden. Die Dienstbeschreibung wird einem XML-Dokument entnommen, das zu jedem `Polymorphic Service` hinzugehört; `Polymorphic Services` sind damit also selbstbeschreibende Dienste.

Nach Auswahl durch den Benutzer erzeugt das RTE die angewählten `Services`. Das RTE ist das einzige Objekt, das über diese Fähigkeit verfügt. Wenn der Benutzer auf der Web-Oberfläche einen Dienst anfordert, wird eine Autorisierungsprozedur angestoßen, in der der Policy Server mit der Kontextinformation des CIS nach einer Autorisierungsentscheidung gefragt wird. Das RTE setzt aufgrund der Antwort des Policy-Servers einen Service zusammen, der auf die Situation maßgeschneidert ist und die vom Policy Server bestimmten Zugriffskontrollen durchführt. Das RTE verfügt über zwei Pools, aus denen es `Protocol Skeletons` und `Constraint Plugins` (s. Abschnitt 5) entnehmen und zu einem `Services` assemblieren kann. Die Antwort des Policy Server bestimmt, welches Skeleton mit welchen Plugins versehen wird und wie diese parametrisiert werden. Der zusammengesetzte Service wird instanziiert und die Methode `configureOnArrival()` wird aufgerufen. In dieser Methode wird initialisiert sich der Service und meldet sich beim RTE durch Aufruf einer `register`-Methode an. Der Nutzer kann daraufhin über diesen dynamisch erzeugten Proxy auf die von ihm benötigten back-end Server zugreifen.

## 5 Proxy-basierte Polymorphic Services

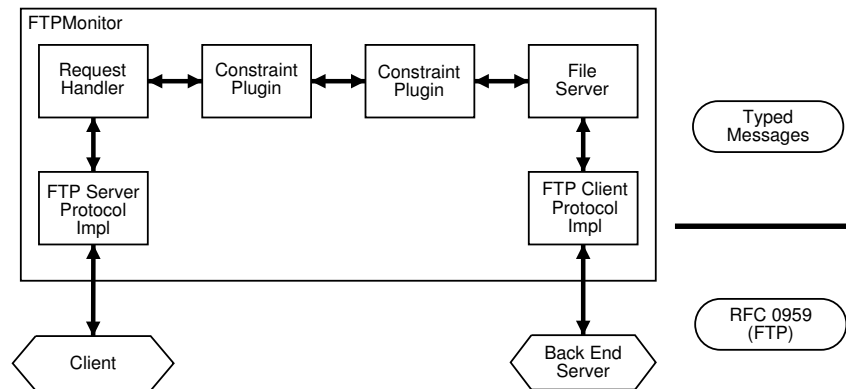
Die Schutzziele, um die es uns hier geht, lassen sich am besten auf der Applikationsebene durchsetzen, da man sich dort von den konkreten Protokollen und den darin ausgetauschten Einzelnachrichten loslösen kann. Um die Kerndienste nicht komplett neu entwickeln zu müssen, wird die geforderte Dienstpolymorphie durch den Einsatz von Proxies herbeigeführt: Diese überwachen auf Applikationsebene den Datenfluß zwischen Client und Server.

Die große Anzahl der eingesetzten Protokolle und die Verschiedenartigkeit der Schutzziele machen allerdings die manuelle Erstellung von Proxies nahezu unmöglich. Daher muß die Proxygenerierung automatisiert werden, was hier mit *Protocol Skeletons* und *Constraint Plugins* erreicht wird. Der hier vorgeschlagene Ansatz beruht auf der Tatsache, daß auf abstrakter Ebene unabhängig vom verwendeten Protokoll Daten bekannter Typen ausgetauscht werden.

*Protocol Skeletons* sind Objekte, die ein (Applikations-)Protokoll implementieren und auf eine abstraktere, objektorientierte Schnittstelle abbilden. So gibt es z.B. `Protocol Skeletons` für FTP und POP3. Bei beiden Protokollen können Dateien mit bekannten Eigenschaften ausgetauscht werden. Ein Skeleton kann nun bei der Instanziierung mit `Constraint Plugins` versehen werden, die den Datenverkehr überwachen können.

Ein *Constraint Plugin* ist ein Java-Objekt mit einer einfachen Schnittstelle. Der Datenverkehr wird vom `Protocol Skeleton` durch die angebundenen `Plugins` geschleift. Da

die Plugins die auf der entsprechenden Schicht ausgetauschten Daten kennen, können sie in den Datenstrom eingreifen und diesen filtern bzw. verändern.



**Abb. 4.** Ein Objekt der Klasse `FTPMonitor`, welches als Application Level Gateway den Datenfluß zwischen Client und back-end Server überwacht. Im rechten Teil wird deutlich, wie der rohe TCP-Datenstrom in getypte Nachrichten umgewandelt wird.

In Abbildung 4 ist zu sehen, wie der Datenfluß zwischen Client und Server innerhalb des FTP-Protocol Skeletons in getypte Nachrichten transformiert und durch die Plugins geschleift wird. Durch das Tysystem auf der abstrakten Ebene sind die ausgetauschten Daten leicht auf Policykonformität hin überprüfbar. Ein einfaches Constraint Plugin könnte z.B. durchsetzen, daß Dateien nur ausgetauscht werden dürfen, falls der Tunnel ins VSN stark verschlüsselt ist. Dazu würde das Plugin vom Context Information Server die Tunneleigenschaften abfragen und den Policy-Server fragen, ob die verwendete Verschlüsselung in die Kategorie stark fällt. Ist das nicht der Fall, so würde das Plugin Dateiaustausch unterbinden, indem die Anfragen negativ beantwortet werden, bzw. vom Server gelieferte Dokumente abgeblockt werden.

## 6 Verwandte Arbeiten

Am ehesten vergleichbar mit dem VSN-Konzept ist die OSGi-Plattform [4]. Dort werden Dienste – sog. Bundles – installiert. Ziel ist es, dem Service Provider die Möglichkeit zu geben, die Dienste zu administrieren, die einem privaten Nutzer in seinem Residential Gateway angeboten werden. Dienste können von einem Content-Provider oder von den im Heim des Benutzers installierten Geräten kommen. Die OSGi-Plattform soll uniformen Zugang zu allen benötigten Diensten erlauben. Das Dienstmodell der OSGi ist nicht so allgemein wie das des VSN. So können Dienste nur auf TCP oder UDP aufgesetzt werden. Insbesondere fehlt bei diesem Ansatz die kontextabhängige Autorisierung.

In einer vorherigen Arbeit ([1], [3]) wurde ein anderer Ansatz verfolgt, sicheren Zugang zu privaten Diensten zu ermöglichen. Dienste werden in diesem Konzept als Jini-Services modelliert und kontextabhängig instanziiert. Die Lösung ist aber nicht für das typische Szenario geeignet, in dem auf Dienste hinter einer Firewall zugegriffen werden muß.

In zahlreichen Firewallinstallationen werden Application Level Gateways eingesetzt. Unser Ansatz steht nicht in Konkurrenz zu diesem Konzept, sondern erweitert dieses um die Berücksichtigung von Situationsattributen und die dynamische Instanzierung.

## 7 Ausblick

Kontextabhängige Autorisierung und die darauffolgende Überwachung der Dienstnutzung sind mächtige Mittel, um sicheren Zugang zu privaten Diensten zu erlauben. Ein Prototyp des VSN-Server wurde auf der CeBIT 2002 ausgestellt. Ein VSN-Server wird gerade in der eigenen Umgebung installiert, um dort Mitarbeitern und Studenten den Zugriff auf Dienste über ein Wireless-LAN zu erlauben.

Derzeit wird ein Toolkit entwickelt, das die semi-automatische Erzeugung von Polymorphic Services erlaubt. Das Toolkit ist so angelegt, daß Programmierer damit Protocol Skeletons und Constraint Plugins für proprietäre Software aufwandsarm realisieren können.

## Literatur

1. Markus Bradtke. *Sicherheitsdienste für Jini-Netze zur Unterstützung des Nomadic Computings*, 2001. Diplomarbeit am Institut für Informatik und praktische Mathematik der Universität Kiel.
2. S. Jajodia, P. Samarati, and V. S. Subrahmanian. A logical language for expressing authorizations. *Proceedings of the IEEE Symposium on Security and Privacy*, pages 31–42, 1997.
3. Norbert Luttenberger. Nomadic Computing mit bestätigten Aufenthaltsorten. *Angenommener Beitrag für die 32. Jahrestagung der GI, Dortmund, 2. Okt. 2002*, October 2002.
4. Dave Marples and Peter Kriens. The Open Services Gateway Initiative: An Introductory Overview. *IEEE Communications*, 39(12):110–114, December 2001.
5. Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, 1996.
6. Thomas Y. C. Woo and Simon S. Lam. Authorizations in distributed systems: A new approach. *Journal of Computer Security*, 2(2-3):107–136, 1993.