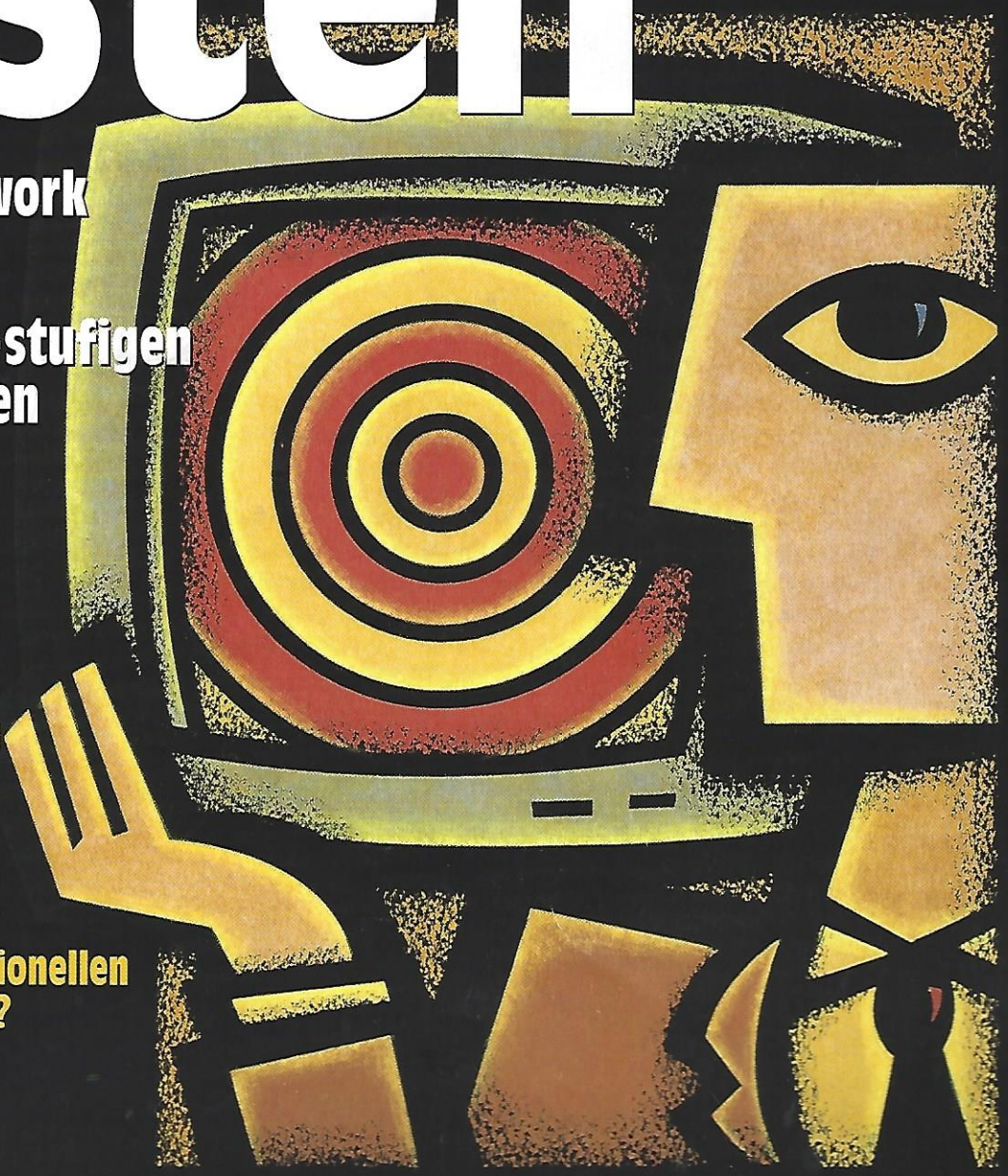
SIGS AUSGABE 5 • SEPTEMBER/OKTOBER '98 • NR.15 • DM 10,- • ÖS 77,- • SFR 10,- http://www.sigs.de

Java™ SPEKTRUM

Java™ SPEKTRUM

DAS COMPUTERMAGAZIN RUND UM JAVA

Testen



JUnit: Ein Framework für das Testen

Das Testen von n-stufigen Java-Anwendungen

Enterprise JavaBeans

Das neue JDK 1.2– Ein Überblick

Java und SSL– Sichere Kommunikation im Internet

Terminalemulation in Java-Umgebungen

Ist XML für den professionellen Einsatz effizient genug?

Blick hinter die Kulissen: Oracle's Java-Strategie

A SIGS PUBLICATION • 30325 • ISSN 1431-4436

05
4 394156 110000

Java auf dem Server

Wer die diversen Softwarekonferenzen und Pressemeldungen in diesem Jahr aufmerksam verfolgt hat, dem dürfte ein allgegenwärtiger Trend nicht entgangen sein. Sowohl Microsoft als auch die auf Java und Unix fokussierte Konkurrenz haben die Schlacht um die letzte Bastion eröffnet. Objekt der Begierde ist nicht der Desktop, sondern die verteilte Unternehmens-EDV, also zu neudeutsch die sogenannten Enterprise-Applikationen. Hier lassen sich noch Geld und Lorbeeren verdienen, ganz im Gegensatz zum Desktop-Bereich, der bereits eine gewisse Sättigungsschwelle erreicht hat.

Bei näherer Betrachtung sind Gemeinsamkeiten der miteinander konkurrierenden Lager nicht zu übersehen. Im Fokus stehen die Multitier-Architekturen, die eine Verteilung von Funktionalität über mehrere Rechner adressieren. Die Devise lautet „weg von monolithischen Strukturen und hin zu einem komponentenbasierten Ansatz". Grafische Benutzungsoberflächen kommunizieren dabei mit Geschäftsobjekten, deren Zustand sich mit Hilfe von Datenbanken oder anderen Speichermedien persistent halten läßt. Vernetzt sind dabei nicht nur Computer im herkömmlichen Sinne, sondern auch alle anderen Arten von Endgeräten, wie etwa Mobilgeräte, Haushaltsgeräte oder Unterhaltungselektronik. Zudem haben moderne Internet/Intranettechnologien die Welt kleiner gemacht. Nicht zu vergessen die wachsende Bedeutung der Integration von Computer- und Telekommunikationstechnologien.

Die Welt mutiert zu einem gigantischen Computernetzwerk. Softwareentwickler stehen somit einer komplexen Integrationsaufgabe gegenüber, nicht zuletzt aufgrund der Heterogenität heutiger IT-Landschaften. Was für Entwickler und Anwender ein Problem darstellt, eröffnet den Herstellern von Betriebssystemen und anderen Basistechnologien neue Geschäftsmöglichkeiten. Kein Wunder, daß Firmen wie IBM, Microsoft oder Sun die Gunst der Stunde nutzen wollen. Microsoft hat die eigene Vision unter dem Namen DNA (Distributed interNet Applications Architecture) subsumiert und will mit Betriebssystemplattformen, Werkzeugen und Enterprise-Applikationen alle Bereiche der Unternehmens-EDV beglücken. Mehr als nur eine Alternative zum Redmonder Windows-Only-Lösungsansatz stellen Java und CORBA dar. Deren Karten sollten eigentlich gar nicht schlecht stehen, zumal diese Technologien von erfahrenen Anbietern für Unternehmenslösungen stammen. Genannt seien an dieser Stelle so renommierte Firmen wie IBM, Sun oder Novell.

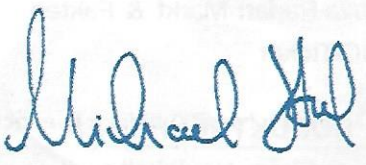
Ansätze wie Enterprise JavaBeans, JDBC, JNDI, CORBA verdecken die Heterogenität vorhandener Softwareplattformen statt einen Umstieg auf neue Technologien zu erzwingen. Dadurch lassen sich einerseits Plattformabhängigkeiten vermeiden und andererseits Produktivität und Qualität steigern. Enterprise JavaBeans sind schließlich sowohl auf Windows NT5 unter dem Microsoft Transaction Server als auch auf einem IBM-Mainframe unter Verwendung von CICS lauffähig.

Warum also ist die ganze Welt nicht schon längst ins Java/CORBA-Lager gewechselt? Nicht Microsoft trifft hier die Schuld. Auch die angebliche Unreife der Java-Technologie taugt längst nicht mehr als Argument. Aus meiner Sicht sollte sich die Microsoftkonkurrenz an die eigene Nase fassen. Wer Java als Hebel für Plattformabhängigkeit und Enterprise Computing propagiert, muß auch die eigene Betriebssystemstrategie überdenken. Ein einheitliches Betriebssystem mit integrierter Java-Technologie, einfacher Bedienung, integrierter Admininistrationslösung, integriertem CORBA und weiteren Enterprise-Technologien ist aus meiner Sicht die längst überfällige Antwort. Das oft zitierte „Write Once, Run Anywhere"-Motto macht in einer Welt von NT-Rechnern nur wenig Sinn. Notwendig in diesem Zusammenhang ist Pragmatismus, nicht Durchhalteparolen oder ein Rückzug auf immer weniger Marktnischen. Noch ist offen, wer aus dem Kampf um die Enterprise-Technologien als Sieger hervorgehen wird. Das Schicksal von Java und CORBA ist mit dieser Schlacht jedenfalls eng verwoben. Aber nicht vergessen: Auch wir Entwickler und Entscheidungsträger tragen Verantwortung. Genau genommen sitzen wir sogar am längeren Hebel, nur hat das bisher keiner bemerkt.

Doch zurück zur Gegenwart. Auch in der vorliegenden Ausgabe haben wir uns bemüht, ein breites Spektrum an wichtigen Themen abzudecken. In einem Artikel von Markus Speier geht es um die bereits eingangs erwähnten vielschichtigen Architekturen. Der Beitrag von Erich Gamma und Kent Beck behandelt das inkrementelle Entwickeln und Testen von Java-Anwendungen. Wunibald Vogl erläutert die sichere Kommunikation im Internet. Mit der neuen brandneuen JDK-Version 1.2 hat sich Hannes Heckner beschäftigt. Unter der Rubrik „Produkte" stehen diesmal Terminalemulationen mit Java im Vordergrund. JavaSPEKTRUM hatte des weiteren die Gelegenheit, ein Interview mit Jeremy Burton von Oracle zu führen. Das ist aber nur ein Ausschnitt aus unserem Angebot. Wir hoffen, auch diesmal wieder die geeignete Mixtur für Sie zusammengestellt zu haben.

An dieser Stelle ein Aufruf in eigener Sache: Wenn Sie selbst zu dem Kreis von Java-Spektrum-Autoren gehören wollen, oder falls Sie Kommentare, Kritik und Empfehlungen artikulieren möchten, können Sie die Redaktion jederzeit unter der E-Mail-Adresse javaspektrum@sigs.com kontaktieren. Wir freuen uns über jede Nachricht, über jedes Lob, aber auch über deftige Kritik.

In diesem Sinne verbleibe ich im Namen des gesamten Teams



Ihr Michael Stal

Inhalt

Java™ SPEKTRUM
DAS COMPUTERMAGAZIN RUND UM JAVA

Kolumnen

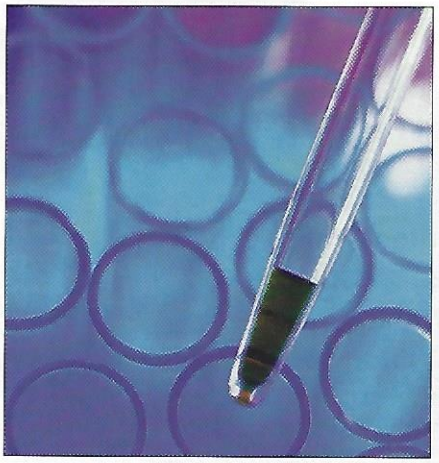
Menschen und Meinungen

Produkte

Ein Hinweis in eigener Sache: Wir sind dabei, die Web-Site für Java-Spektrum und OBJEKTspektrum unter der URL www.sigs.de aufzubauen. Dort werden Sie ab sofort einen Überblick über die Zeitschriften, z. B. die Inhaltsverzeichnisse sämtlicher bisher erschienener Hefte, sowie ausgewählte Artikel finden.

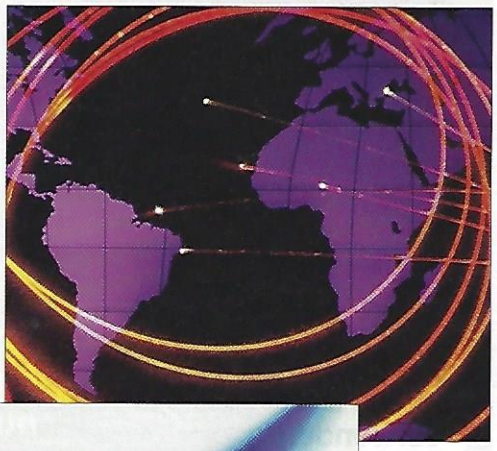
Rubriken

TITELTHEMA: TESTEN



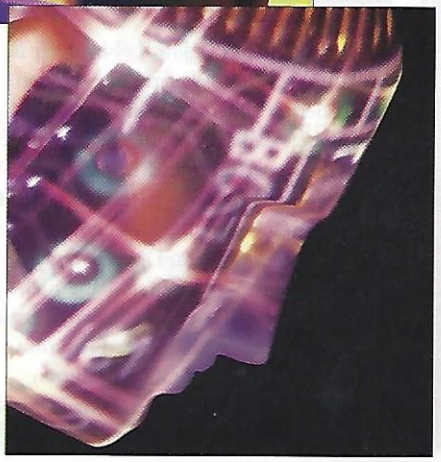
S. 22

FACHTHEMEN



S. 40

S. 46



S. 56

Auf Nummer Sicher

Java und „Secure Socket Layer“

WUNIBALD VOGL

Bei einer Kommunikation vertraulicher Daten über das Internet ist Verschlüsselung angesagt. Hierzu hat sich das „Secure Socket Layer“-Protokoll als De-facto-Standard durchgesetzt. Am Beispiel des Java-Anschlusses für die Client-Seite eines Transaktionsmonitors wird der Einsatz von SSL erläutert und gezeigt, daß heute schon sichere Verfahren für eine kommerzielle Nutzung des Internets zur Verfügung stehen.

Java bietet hervorragende Möglichkeiten zur Softwareverteilung. Daß diese Vorteile zunehmend auch für geschäftskritische Anwendungen interessant werden, liegt auf der Hand. Falls die Kommunikation zwischen Java-Client und Java-Server-Anwendung nicht mehr nur über das Intranet, sondern auch über das Internet stattfindet, müssen entsprechende Vorkehrungen getroffen werden. Verschlüsselung ist angesagt. Aufgrund seiner Einfachheit und Kompatibilität hat sich das Protokoll SSL (Secure Socket Layer) als De-facto-Standard durchgesetzt. SSL kann mit jedem Internet-Anwendungsprotokoll zusammenarbeiten, das auf TCP/IP aufsetzt. Zwischen der Anwendungs- und Transportschicht wird die SSL-Funktionalität integriert. Die Anwendungsschicht muß dabei nur marginal angepaßt werden. Verschlüsselung ist ein Aspekt, doch SSL leistet mehr.

Am Beispiel des Java-Anschlusses für die Client-Seite des Transaktionsmonitors openUTM von Siemens Nixdorf wird der Einsatz von SSL erläutert. Ein Einsatzszenario ist beispielsweise die Buchung einer Urlaubsreise über das Internet inklusive Bezahlung via Kreditkarte. Die Transaktionsfunktionalität von openUTM wird benötigt, um sicherzustellen, daß entweder *alle* Buchungen (Flug, Hotel, Mietauto) erfolgen oder *alle* Buchungen unterbleiben, SSL ist nötig für die Sicherheit der Online-Bezahlung der Reise.

Welche Gefahren drohen bei einer Kommunikation über das Internet

Knacken der Schlüssel

SSL stützt sich auf mehrere kryptografische Verfahren. Das RSA-Publickey-Verfahren wird eingesetzt zum Austausch der Sessionkeys und der Client-/Server-Authentifizierung. Für die Verschlüsselung werden bekannte Verfahren wie RC4, IDEA, DES etc. verwendet. Sobald eines dieser Verfahren geknackt werden würde, wäre auch SSL nicht mehr sicher. Soviel nur zur Vollständigkeit.

Leichter als das Knacken des Verfahrens wäre dagegen, eine Kommunikation zwischen einem Client und einem Server aufzuzeichnen und viel Zeit und Geld zu investieren, um die verwendeten Schlüssel zu knacken. Der Aufwand für das Knacken eines 128-bit-Schlüssels ist immens hoch und beansprucht soviel Zeit, daß ein möglicher Nutzen in keinem Verhältnis zum Aufwand steht. Nebenbei sei bemerkt, daß die Schlüssellänge auch abhängig ist von den aktuell verfügbaren Rechnerleistungen. Je schneller die Rechner werden, um so länger müssen dann auch die Schlüssel gewählt werden.

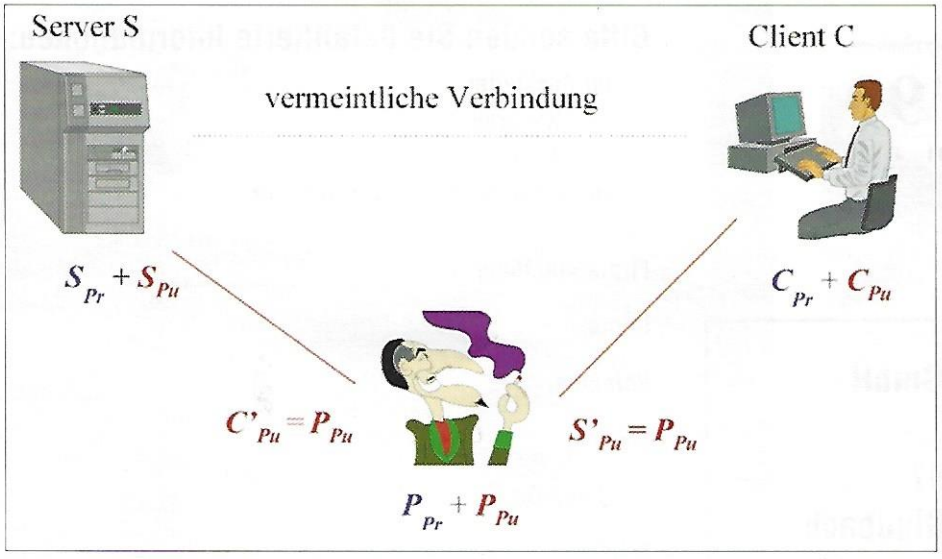


Abb. 1: Man-in-the-middle“-Attacke

„Known Plaintext“-Attacke

Es wäre denkbar, daß aufgrund der Kenntnis eines Teils der zu übertragenden Daten die Entschlüsselung erleichtert wird. Die in SSL verwendeten Chiffren und deren Betriebsmodi sind gegen „Known Plaintext“-Attacken resistent, d. h. das Wissen über den verschlüsselten Text erleichtert die Suche nach dem Schlüssel in keiner Weise.

Blockchiffren werden im CBC-Modus betrieben. Das Ergebnis der Verschlüsselung eines Blocks hängt dabei von einem zufälligen Initialisierungsvektor und den vorangegangenen Blöcken ab, so daß identische Blöcke zu verschiedenen Verschlüsselungsergebnissen führen (s. [applcrypt]).

„Replay"-Attacke

Die Sitzung zwischen einem Client und dem Server wird von einer Person P abgehört und aufgezeichnet. Zu einem späteren Zeitpunkt wird dann von dieser Person P eine Verbindung zu diesem Server aufgebaut und gemäß den aufgezeichneten Daten die Sitzung wiederholt ohne Kenntnis der Daten, aber wohlwissend, daß die Daten korrekt verschlüsselt sind.

Wenn die Session noch nicht geschlossen wurde, nimmt der Server die Verbindung an. Der Server erkennt jedoch anhand von Sequenz-Nummern, daß die ihm geschickten Pakete nicht gültig sind.

„Man-in-the-middle"-Attacke

Diese Attacken sind vor allem deswegen sehr gefährlich, weil hier nicht nur die Nachrichten entschlüsselt, sondern auch verfälscht werden könnten, ohne daß die Partner etwas bemerken.

Damit eine dritte Person P sich zwischen den Client C und den Server S schieben kann und von beiden nicht erkannt wird, ist es notwendig, daß die Person P seine eigenen Schlüssel ins Spiel bringen kann. Gegenüber dem Client tritt die Person P als Server auf, gegenüber dem Server als Client. Der Client bekommt dabei den Publickey von P als den Publickey von S und der Server den Publickey von P als Publickey von C untergeschoben (s. Abb. 1).

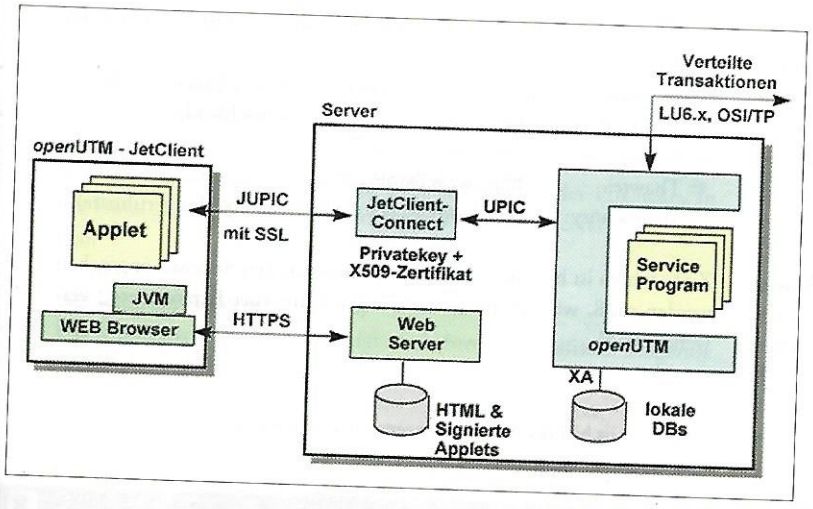


Abb. 2: Architektur des openUTM-JetClient

P reicht die Daten vom Client zum Server und umgekehrt. Dadurch daß P seine Schlüssel ins Spiel gebracht hat, kann P die Sessionkeys und damit alle Daten entschlüsseln und beliebig modifizieren.

Beim SSL-Protokoll wird dies dadurch unterbunden, daß der Server sich gegenüber dem Client durch ein Zertifikat ausweist. Dieses Zertifikat enthält den Publickey des Servers. Durch Überprüfung dieses Zertifikats kann der Client sicher sein, daß er auch mit dem Server verbunden ist.

Erläuterung einer sicheren Kommunikation am Beispiel openUTM-JetClient

Abbildung 2 stellt die Architektur des openUTM-JetClient dar. Von einem Applet aus soll über einen Verteilerbaustein, genannt openUTM-JetClientConnect, auf mehrere openUTM-Anwendungen zugegriffen werden können. Der Fokus wird nun auf die Kommunikation zwischen dem Applet und JetClientConnect gelegt werden, denn genau zwischen diesen Partnern findet eine SSL-Kommunikation statt.

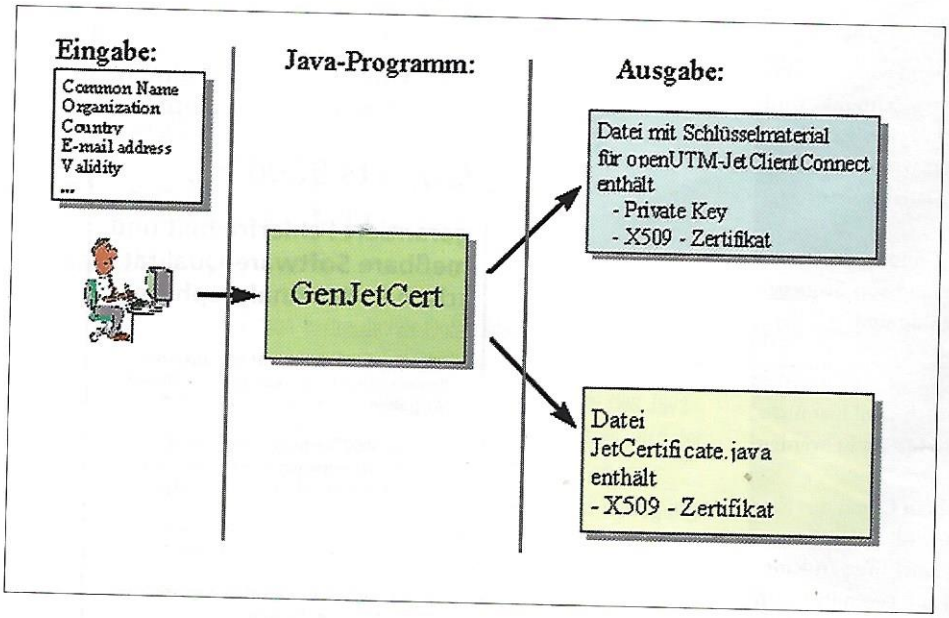


Abb. 3: Erzeugung des Schlüsselmaterials mit GenJetCert

Ein openUTM-JetClient-Applet besteht neben den JDK-1.1.x-Klassen aus:

- den openUTM-JetClientClasses: Das sind Klassen, die die Kommunikation mit JetClientConnect ermöglichen. In diesen Klassen ist sowohl der Socket-Anschluß integriert als auch die Methoden für den Zugriff auf die openUTM-Anwendungen.
- den kryptografischen Java-Klassen: Hierzu gehören sowohl die Klassen zum Verschlüsseln, Entschlüsseln etc. als auch die Klassen für das SSL-Protokoll.
- den Anwenderklassen: Diese Klassen müssen vom Anwender erstellt werden. Sie beinhalten die Logik des Applets und die grafische Darstellung am Bildschirm.

Der Verteilerbaustein JetClientConnect ist ein reines C-Programm, dessen Aufgabe darin besteht, eine Verbindung zur gewünschten UTM-Anwendung aufzubauen. Dieser Baustein kann als Namensdienst für das Routing der Client-Requests dienen und ermöglicht die örtliche Trennung von Web-Server und Anwendungs-Server. Für den JetClientConnect wird eine in C implementierte SSL-Bibliothek verwendet.

Die Schnittstelle der JetClientClasses zu den Anwenderklassen ist sehr einfach gehalten und besteht im wesentlichen aus folgenden Methoden:

- Adressieren einer UTM-Anwendung,
- Auswählen eines aufzurufenden Services,
- Setzen einer Benutzerkennung (optional),
- Setzen eines Kennworts (optional),
- Senden einer Nachricht an einen Service,
- Empfangen einer Nachricht von einem Service,
- Abbau der Verbindung und Endebehandlung.

Vorbereitungen

Erzeugen der Schlüssel für den Server JetClientConnect

Um überhaupt über SSL kommunizieren zu können, braucht der Server ein Zertifikat und einen Privatekey. Das Zertifikat ist notwendig für die Server-Authentifizierung. Der Privatekey ist notwendig, um Nachrichten entschlüsseln zu können, die der Client mit dem im Zertifikat enthaltenen Publickey verschlüsselt versendet.* Für die Erzeugung des Schlüsselmaterials wird im openUTM-JetClient das Java-Programm GenJetCert zur Verfügung gestellt, das sowohl ein Zertifikat samt Privatekey für den Server erzeugt als auch dieses Server-Zertifikat in einer Java-Klasse ablegt (s. Abb. 3). Die Java-Klasse enthält das Zertifikat als Bytearray und dient dem Client zur Überprüfung des Server-Zertifikats, näheres dazu später.

Eingaben für das Werkzeug sind z. B. Name des Servers, Firma, E-Mail-Adresse, Land etc.

Besorgen eines Schlüssels zum Signieren der Applets

Einen Schlüssel zum Signieren erhält man u. a. von folgenden Stellen:

- Trustcenter: http://www.trustcenter.de (Deutschland),
- Verisign: http://www.verisign.com,
- Thawte: http://www.thawte.com,
- Trustfactory: http://www.secude.com/trustfactory (Testzertifikate).

Zu beachten ist hier, daß unter den oben genannten Adressen angegeben werden muß, welcher Browser (Navigator, Internet Explorer etc.) verwendet wird.

* Gemeint ist hier der Austausch sogenannter Sessionkeys.

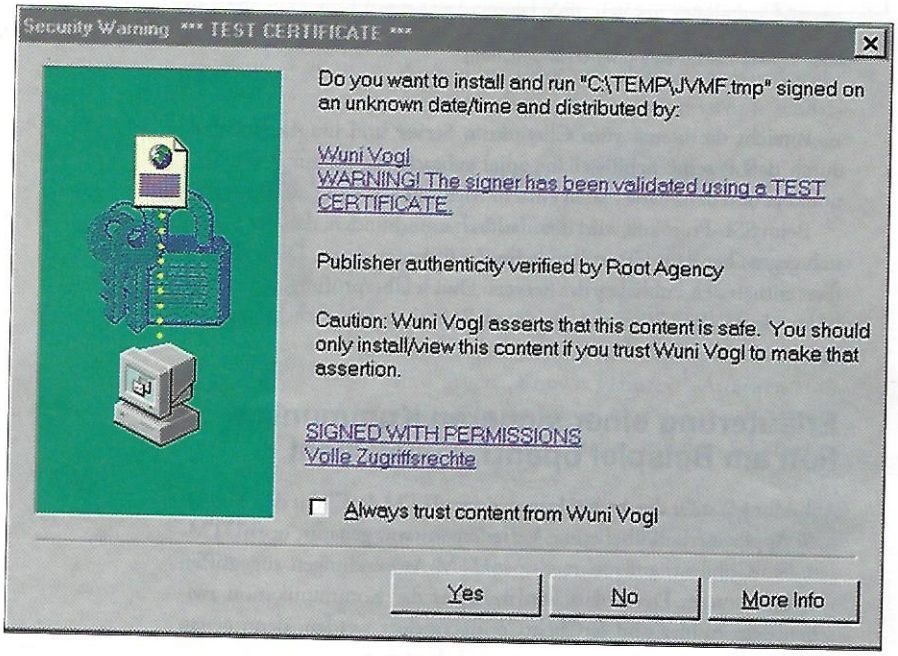


Abb. 4: Anzeige eines Zertifikats im Internet Explorer

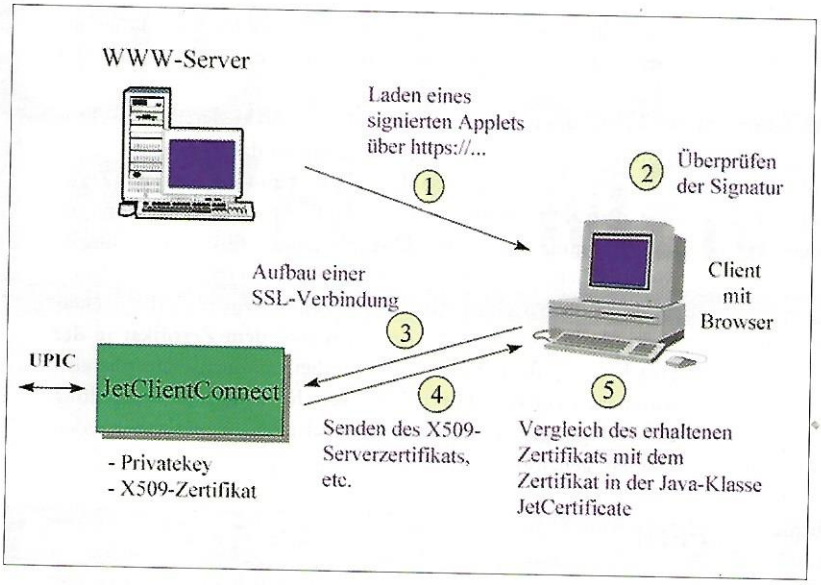


Abb. 5: SSL-Kommunikation

Im ersten Schritt generiert der Browser ein Schlüsselpaar, das in der browserspezifischen Datenbank abgelegt wird. Im zweiten Schritt wird mit diesem Schlüsselpaar ein X509-Zertifikat erzeugt, das an die Zertifizierungsstelle geschickt wird. Nach Erhalt des von der Zertifizierungsstelle signierten Zertifikats wird es mit Hilfe des Browser ebenfalls in dieser Datenbank abgelegt. Im Falle Netscape ist der Privatekey in der Datei „key3.db" und das Zertifikat in der Datei „cert7.db" zu finden.

Signieren der Applets

Das Signieren der Applets erfolgt auch browserspezifisch. Folgende Werkzeuge stehen zur Verfügung:

- SDK von Microsoft (SignCode): Internet Explorer,
- zigbert: Netscape Navigator,
- javakey Java Activator.

Beispielprozedur zum Signieren mit dem Netscape-Werkzeug zigbert:

Die Dateien „key3.db" und „cert7.db", die das Zertifikat „Wunibald Vogl's GMD ID" und den dazugehörigen Privatekey enthalten, wurden zu diesem Zweck in das Verzeichnis c:\bin\crypto kopiert:

```
zigbert -k „Wunibald Vogl's GMD ID" -d"c:\bin\crypto" sources

cd sources
zip -r0 ..\testSSL.jar *
cd ..
```

Beispielprozedur zum Signieren mit Microsofts SDK:

```
cabarc -p -r n testssl.cab *
makecert -sk vogl -n „CN=Wunibald Vogl" vogl.cer
cert2spc vogl.cer vogl.spc
signcode -j javasign.dll -jp low -spc vogl.spc -k vogl testssl.cab
```

Ablauf einer SSL-Sitzung

Download der Software

Das entscheidende Kriterium für eine sichere und verschlüsselte Kommunikation im Java-Umfeld ist, daß die Original-Software auf den Client-Rechner heruntergeladen wird. Denn in dieser Software ist festgelegt, ob dem Partner zu trauen ist, welche Verschlüsselungsverfahren verwendet werden und welche Schlüssellängen benutzt werden. Wenn es einem möglichen Angreifer gelingen würde, diese Software zu verfälschen, wäre jegliche Manipulation möglich.

Zum sicheren Laden der Software von einem WWW-Browser stehen zwei Möglichkeiten zur Verfügung:

- Signieren der Applets: Mittels der Signatur kann überprüft werden, ob die Software verfälscht wurde.
- Herunterladen der Software über eine SSL-Verbindung zwischen dem Browser und dem WWW-Server: Daß eine SSL-Verbindung verwendet wird, erkennt man am URL, der mit https://... beginnt.

Die erste Variante stellt sicher, daß die Original-Software heruntergeladen wurde. Es ist zwar nicht sichergestellt, daß die Software vom angewählten Rechner heruntergeladen wird, aber aufgrund der Signatur kann sichergestellt werden, daß die Software unverfälscht und die Sicherheit damit nicht beeinträchtigt wird.

Die zweite Variante stellt lediglich sicher, daß die Software auch vom angewählten Rechner heruntergeladen wird. Sicherheit wird hier nur erreicht, wenn Unbefugte keinen Zugriff auf den Rechner haben, d. h. daß Unbefugte auch keine Dateien, insbesondere Software, auf diesen Rechner ablegen können.

Die erste Variante wäre der zweiten vorzuziehen, doch leider gibt es ein Problem. Wenn ein Applet keine Rechte beansprucht außer den Rechten, die ein nichtsigniertes Applet auch besitzt, ermöglichen die beiden Standard-Browser Netscape Navigator und Microsofts Internet Explorer kein Überprüfen der Signatur. Die Signatur zu verwenden für die Authentizität der heruntergeladenen Software wäre ein hervorragendes Hilfsmittel, wenn dies von den Browsern unterstützt würde. Momentan ist zu empfehlen, daß die beiden Varianten kombiniert werden.

Die Alternative, dem Applet provisorisch zusätzliche Rechte einzuräumen, um die Signatur überprüfen zu können, ist nur mit Vorbehalt zu empfehlen, da zumindest für den Navigator browserspezifisches Coding notwendig ist. Für den Internet Explorer gibt es die Möglichkeit, verschiedene Sicherheitsstufen bei der Signierung anzugeben (hoch, mittel, niedrig). Hoch entspricht dem Sandkastenmodell, niedrig bedeutet, daß das Applet z. B. auch auf die Festplatte zugreifen darf. Abbildung 4 zeigt ein Fenster, das vom Internet Explorer angezeigt wird, wenn ein mit Sicherheitsstufe „niedrig" signiertes Applet gestartet werden soll.

SSL-Kommunikation

Abbildung 5 zeigt die Schritte im einzelnen:

1. Der Benutzer gibt im URL-Feld des Browsers die Adresse des JetClient-Applets ein. Da der WWW-Server eine SSL-Verbindung voraussetzt, muß der URL mit https:// beginnen. Ist im Browser das Server-Zertifikat nicht als vertrauenswürdig hinterlegt, erscheint ein Fenster mit dem Inhalt des Zertifikats, damit das Zertifikat überprüft werden kann.
2. Falls das Applet das „Sandkastenmodell" verlassen will, besteht die Möglichkeit, die Signatur zu überprüfen.
3. Das Applet wird gestartet. Nachdem der Benutzer seine Benutzerkennung, sein Kennwort und weitere Daten eingegeben hat, wird versucht eine SSL-Verbindung aufzubauen. Was nun im einzelnen zwischen dem Applet und dem JetClientConnect abläuft, ist durch die SSL-Software abgedeckt, d. h. der JetClient-Programmierer braucht sich darum nicht zu kümmern, ausgenommen Zertifikatsüberprüfung, siehe Punkt 5.
4. Während dieser Initialisierungsphase schickt das (Server-)Programm JetClientConnect sein Zertifikat an das Applet.
5. Das Applet muß nun der SSL-Bibliothek mitteilen, ob das Zertifikat akzeptiert werden kann und damit eine SSL-Verbindung zustande kommen soll. Die Klasse, die das Zertifikat überprüft, ist vom JetClient-Programmierer zu implementieren und beinhaltet im wesentlichen einen Abgleich zwischen dem über die Socket-Verbindung empfangenen Zertifikat und dem Zertifikat in der Java-Klasse JetCertificate. Falls die beiden Zertifikate übereinstimmen, steht eine mit 128 bit verschlüsselte SSL-Verbindung zur Verfügung, über die Daten gesendet und empfangen werden können, genauso wie ohne Verschlüsselung.

Fazit

Trotz der Browserabhängigkeiten stehen auch heute schon sichere Verfahren für eine kommerzielle Nutzung des Internets zur Verfügung. Zu hoffen bleibt allerdings, daß die Browserabhängigkeiten mit dem Erscheinen des JDK 1.2 und insbesondere den JDK-1.2-kompatiblen Browsern endgültig der Vergangenheit angehören. Wenn dieser Artikel erscheint, ist es vielleicht schon so weit ...

Links

JCE- und SSL-Packages sowohl für kommerzielle als auch für private Zwecke: http://jcewww.iaik.tu-graz.ac.at/

Trustcenter (Zertifizierungsstelle in Deutschland): http://www.trustcenter.de/

Testzertifikate (Gültigkeit: 30 Tage): http://www.secude.com/trustfactory/

Signieren von Applets: http://www.suitable.com/Doc_CodeSigning.shtml

Netscape Signing Tool: http://developer.netscape.com/docs/manuals/signedobj/zigbert/index.htm

Netscape System Targets: http://developer.netscape.com/docs/manuals/signedobj/targets/index.htm

Netscape Capabilities Classes: http://developer.netscape.com/docs/manuals/signedobj/capabilities/contents.htm

Microsoft's Signiertool: http://www.microsoft.com/java/sdk/20/tools/signcode.htm

Outside the sandbox: http://www.sigs.com/jro/features/9801/sommers.html

Literatur

[applcrypt] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, Handbook of applied cryptography, CRC Press, Boca Raton New York London Tokyo

WUNIBALD VOGL arbeitet bei der Siemens Nixdorf Informationssysteme AG und ist zuständig für die Java-Anbindung des Transaktionsmonitors openUTM.
E-Mail: Wunibald.Vogl@bingo.baynet.de.