

Internet Security 2009W

Protokoll

WLAN Relay

Manuel Mausz, Matr. Nr. 0728348

manuel-tu@mausz.at

Aldin Rizvanovic, Matr. Nr. 0756024

e0756024@student.tuwien.ac.at

Wien, am 23. Dezember 2009

Inhaltsverzeichnis

Aufgabenstellung und Vorbereitung.....	3
Paketaufzeichnung und Tools.....	3
Protokoll: HTTP.....	5
Protokoll: HTTPS.....	8
Protokoll: SMTP.....	9
Protokoll: POP3.....	10
Protokoll: IMAP.....	11
Protokoll: AOL.....	12
Protokoll: Jabber.....	13

Aufgabenstellung und Vorbereitung

Teil der Aufgabe ist es, ein „man in the middle“-Szenario aufzubauen, IP-Pakete diverser Protokolle aufzuzeichnen und diese zu analysieren. Fokus soll ins besonders auf Logindaten liegen.

Um einen „man in the middle“-Angriff zu ermöglichen, ist ein zusätzlicher Angreifer-PC notwendig, der als Bridge (oder Router) agiert, der Pakete vom und zum Benutzer-PC empfängt und diese transparent an sein entsprechendes Ziel weiterleitet.

In der Aufgabenstellung ist der Angreifer-PC als Bridge zu konfigurieren. Dies funktionierte jedoch auf unserem Angreifer-PC, dessen Betriebssystem GNU/Linux ist, nicht. Die Verbindung zum Benutzer-PC via WLAN im AdHoc-Modus funktionierte zwar, doch sobald die 2 Endpunkt-Interfaces der Bridge hinzugefügt wurde, war kein Ping mehr möglich. Merkwürdigerweise funktionierte die IP-Zuweisung via DHCP, dessen Pakete ebenfalls durch die Bridge geroutet werden. Nach genauerer Recherche stellte sich heraus, dass alle der Bridge zugewiesenen Interfaces, das dynamische Ändern der MAC-Adresse unterstützen müssen, was unsere Intel WLAN-Karte nicht unterstützte.

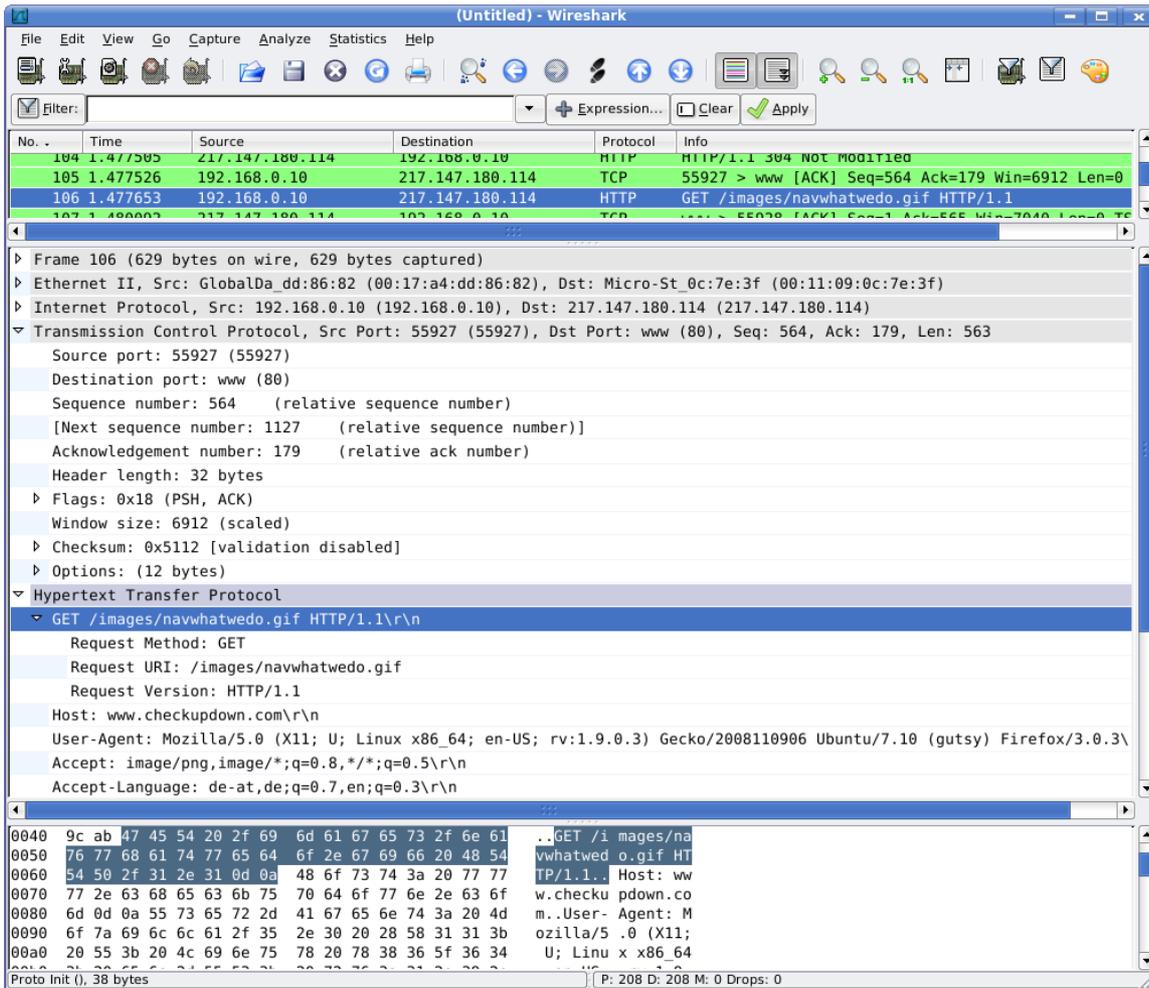
Daher haben wir uns für eine alternative Konfiguration entschieden, bei der der Angreifer-PC als Router dient und ein NAT („*Network Address Translation*“) aufbaut. Somit ist das Szenario, abgesehen von unterschiedlichen Subnetzen der 2 PCs, gegeben. Wie man ein NAT mit *iptables* unter Linux konfiguriert, ist auch Teil der Aufgabe „firewall“ und wird daher nicht näher behandelt.

Paketaufzeichnung und Tools

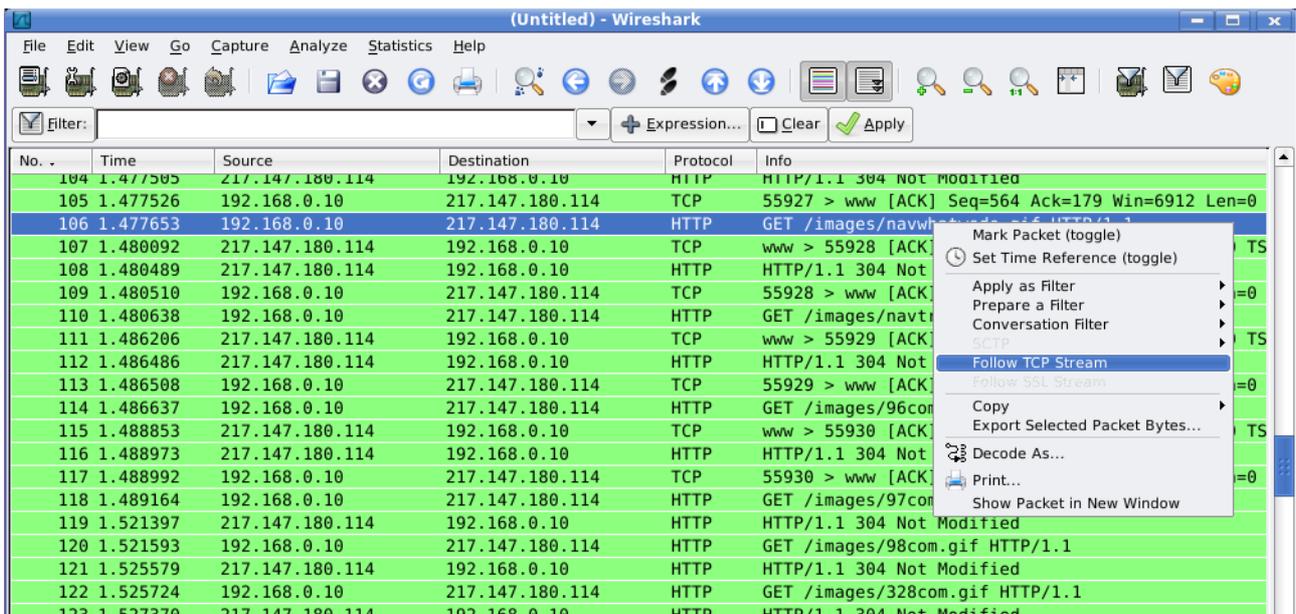
Zur Aufzeichnung der Datenpakete sowie zur Auswertung des TCP/UDP-Streams wurde das OpenSource Tool Wireshark verwendet, das unter der Lizenz GPL entwickelt wird. Hierzu wird Wireshark gestartet, in den Options das entsprechend Ethernet Interface ausgewählt und die Aufzeichnung gestartet.

Alle ein und ausgehenden Pakete dieses Interfaces bekommen eine fortlaufende Nummer zugewiesen und werden direkt angezeigt in einer Tabelle angezeigt. Zudem wird üblicherweise die IP-Adresse der Quelle und des Ziels sowie der Port und einige Informationen aufgelistet. Genauere Details erhält man über das klicken auf das jeweilige Datenpaket. Zudem kann man sich, sofern für das jeweilige Protokoll unterstützt, alle Nutzdaten der einzelnen Pakete in einem eigenem Fenster anzeigen lassen.

Informationen zu dem ausgewählten Paket:



„Follow TCP Stream“ zum Anzeigen der Nutzdaten des Streams:



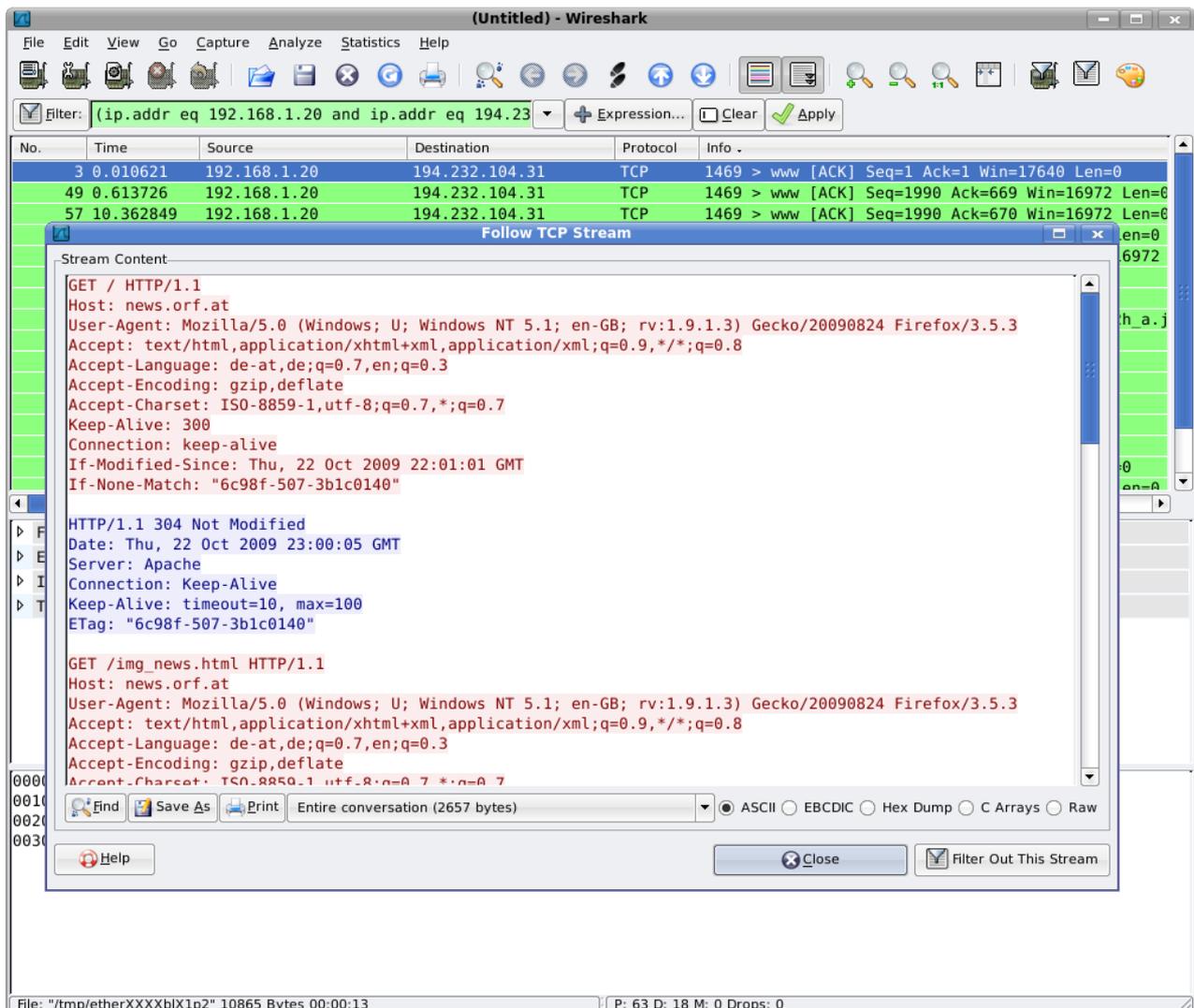
Protokoll: HTTP

Als erstes Protokoll wurde von uns HTTP ausgewählt. Hierbei wurde natürlich, wie bei allen folgenden Protokollen auch, hauptsächlich Pakete erzeugt, anhand deren man benutzerspezifische Daten auswerten kann. Bei einigen Protokollen wurde insbesondere die Verschlüsselung deaktiviert, damit die Pakete überhaupt gelesen werden können.

Das HTTP („*Hypertext Transfer Protocol*“) Protokoll dient zur Übertragung von Daten über ein Netzwerk, wobei es heutzutage hauptsächlich zur Übertragung von Webseiten eingesetzt wird.

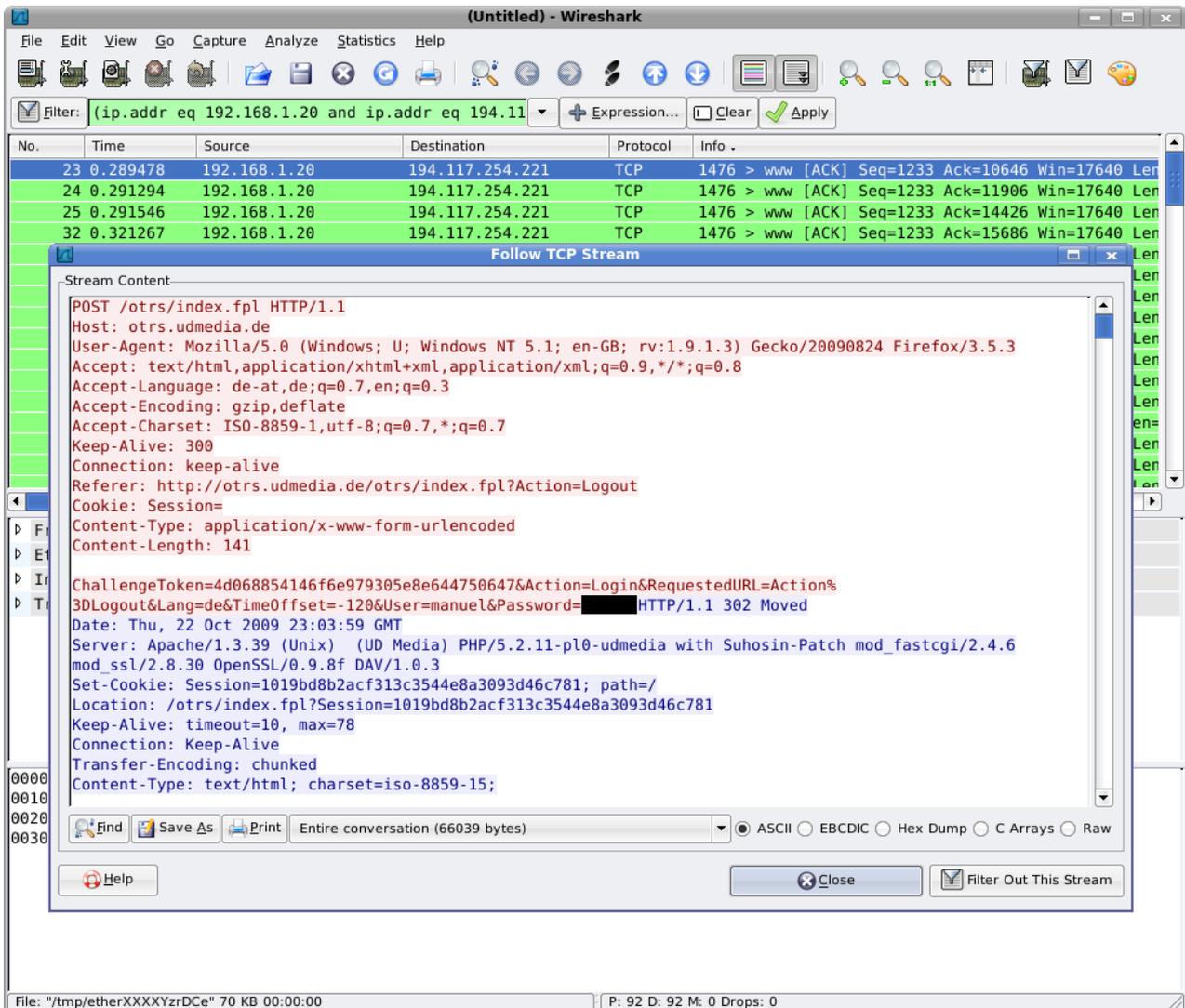
Als erstes Beispiel wurde die Webseite `http://news.orf.at` aufgerufen. Der rot hinterlegte Text kennzeichnet hierbei die Daten des Benutzer-PCs an den Webserver, der blaue die Daten des Webserver an den Benutzer-PC. Im ersten Teil sendet der Benutzer-PC einen „*HTTP GET*“-Request, worauf der Webserver mit einem „*HTTP 304 Not Modified*“ antwortet. Diese Antwort bedeutet, die angefragte Webseite enthält keine neuen Daten und muss somit nicht neu übertragen und kann aus dem Cache gelesen werden. Ob die Webseite neue Daten enthält, kann der Webserver durch das im ersten Request enthalten Datum im „*If-Modified-Since*“-Header feststellen.

Im dritten Teil wird ein „*HTTP GET*“-Request für ein Bild an den Webserver geschickt. Die weiteren Daten sind im Screenshot nicht ersichtlich.



Im zweiten Beispiel wurde ein „*HTTP POST*“-Request durchgeführt. Dies wird üblicherweise bei Formularen verwendet, um meist benutzerspezifische Daten an den Webserver zu übertragen, dabei aber diese Daten nicht in der URL und somit in der Browser History für andere sichtbar zu machen. Meist ist jeder Login ein solche Request, weshalb wir einen solchen auch als Beispiel gewählt haben.

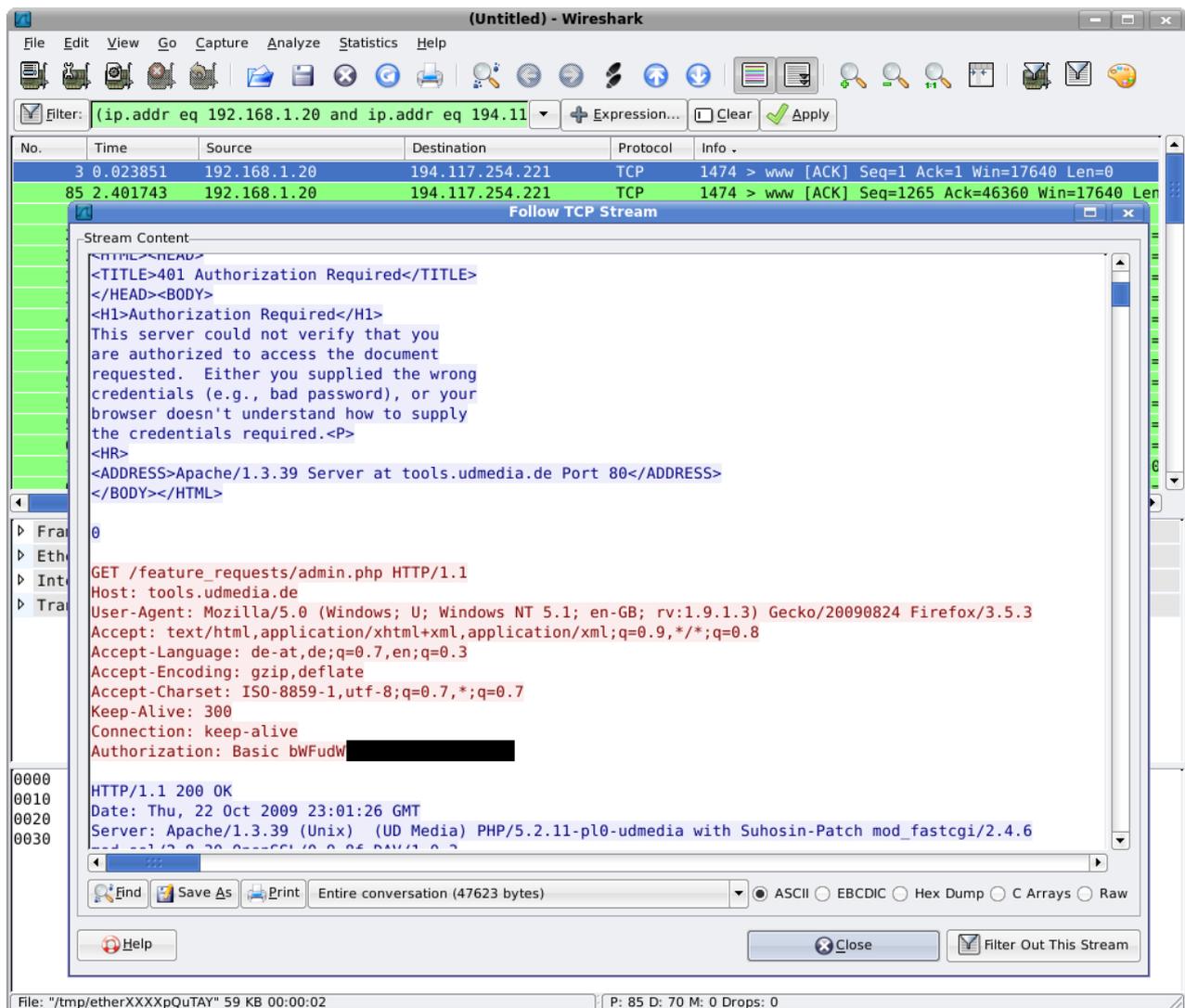
Die an den Webserver übertragenen Daten sieht man am Ende des Requests an den Webserver (roter Teil). Insbesondere Benutzername („*User*“) und Passwort („*Password*“) ist in der Paket-Aufzeichnung deutlich ersichtlich.



Als drittes Beispiel für das HTTP-Protokoll, haben wir uns für die Authentifizierung (*HTTP 401 Authorization Required*) entschlossen. Dies ist ein HTTP-Requesttyp um eine direkte Authentifizierung des Benutzers über den Webbrowser zu erzwingen. Üblicherweise wird dies durch ein schlichtes Popup des Webbrowsers mit Eingabefeldern für Benutzername und Passwort dargestellt.

Auch bei diesem Requesttyp liegen die Benutzerdaten im Klartext vor, sind jedoch zusätzlich noch im Syntax „<benutzername>:<password>“ Base64 kodiert. Die Base64-Kodierung dient hierbei zur Übertragung von Daten, die nicht im ASCII-Zeichensatz enthalten sind, wird also nicht zur Erhöhung der Sicherheit angewendet! - Zur Dekodierung muss also die Zeichenkette zuerst Base64 dekodiert werden und kann dann gelesen werden.

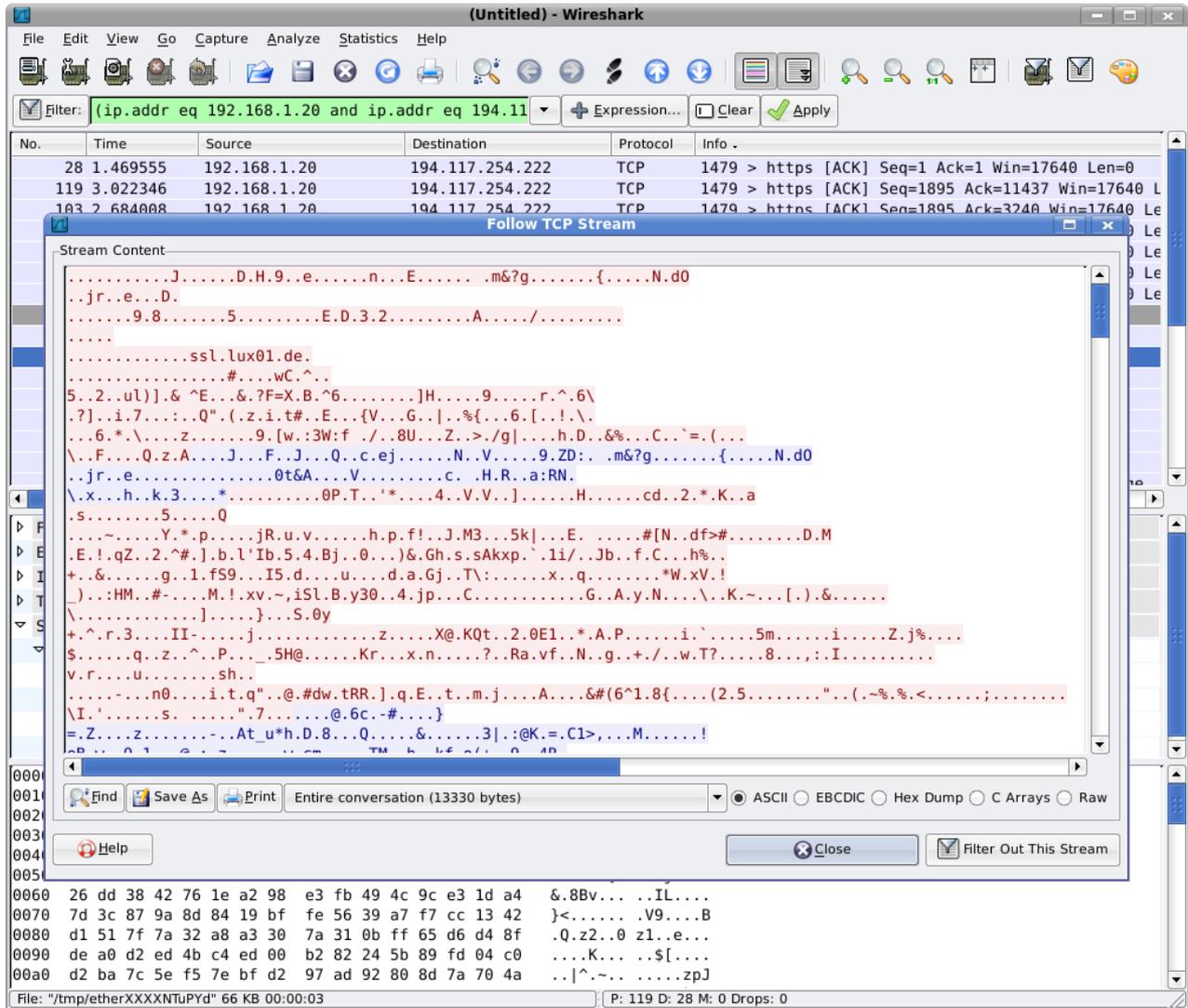
Zur Vollständigkeit sei erwähnt, dass die HTTP-Authentifizierung auch Methoden unterstützt, bei dem nur ein Hash-Wert aus Benutzerdaten und Challenge (vom Webserver erstellt) übertragen wird. Hier sind die Benutzerdaten selbstverständlich nicht lesbar.



Protokoll: HTTPS

Als zweites Protokoll wurde *HTTPS* ausgewählt. Hierbei werden die Daten der HTTP-Übertragung zusätzlich noch mit *SSL/TLS* verschlüsselt. SSL bzw. TLS dient zur verschlüsselten Übertragung von Daten, basiert auf „*asymmetrischen Kryptoverfahren*“ und unterstützt mehrere Algorithmen.

Wie man im Screenshot erkennen kann, ist das Lesen der übertragenen Daten nicht möglich. Grundsätzlich ist aber auch SSL/TLS nicht vor „*man in the middle*“-Attacken geschützt, denn der Angreifer-PC könnte spezifische Pakete an beide Seiten senden, wodurch neue Schlüssel ausgehandelt werden. Damit wäre auch hier das Lesen der Nutzdaten möglich.



Protokoll: SMTP

Das SMTP „Simple Mail Transfer Protocol“ Protokoll dient, wie der Name schon sagt, zum Versenden von E-Mails über einen E-Mail-Server. Damit das Versenden an für den E-Mail-Server fremde Benutzer (nicht lokale Benutzer, üblicherweise als „Relaying“ bezeichnet) möglich ist, muss sich der Benutzer zuerst authentifizieren. Dies ist beim SMTP-Protokoll, wie bei der HTTP-Authentifizierung, ebenfalls Base64 kodiert, jedoch in 2 einzelne Befehle ausgeteilt, wobei zuerst der Benutzername, anschließend das Passwort verschickt wird.

Auch hier sei zur Vollständigkeit erwähnt, dass SMTP eine Vielzahl von Authentifizierungen unterstützt. Neben dem, aus der HTTP Authentifizierung schon bekannten, Übertragung eines Hash-Wertes, ist auch die verschlüsselte Übertragung über SSL/TLS möglich.

Diese Aufzeichnung einer SMTP-Verbindung überträgt übrigens keine E-Mail, sondern beendet die Sitzung vorher. Üblicherweise werden die Daten nach einem „DATA“-Befehl übertragen.

The image shows a Wireshark capture of an SMTP session. The main window displays a packet list with the following entries:

No.	Time	Source	Destination	Protocol	Info
22	18.646571	194.117.254.30	192.168.1.20	SMTP	Response: 334 VXNlcm5hbWU6
12	6.738710	194.117.254.30	192.168.1.20	SMTP	Response: 502 unimplemented (#5.5.1)
4	0.024956	192.168.1.20	194.117.254.30	TCP	[TCP segment of a reassembled PDU]

The 'Follow TCP Stream' window shows the following SMTP conversation:

```
.....220 mail.root.udmedia.de ESMTP
ehlo inetsec
502 unimplemented (#5.5.1)
ehlo inetsec
250-mail.root.udmedia.de
250-STARTTLS
250-PIPELINING
250-8BITMIME
250-AUTH LOGIN PLAIN
250 SIZE 31457280
auth login
334 VXNlcm5hbWU6
dWRfM3Ax
334 UGFzc3dvcnQ6
235 ok, go ahead (#2.0.0)
mail from: root
250 ok
rcpt to: root
250 ok
quit
221 mail.root.udmedia.de
```

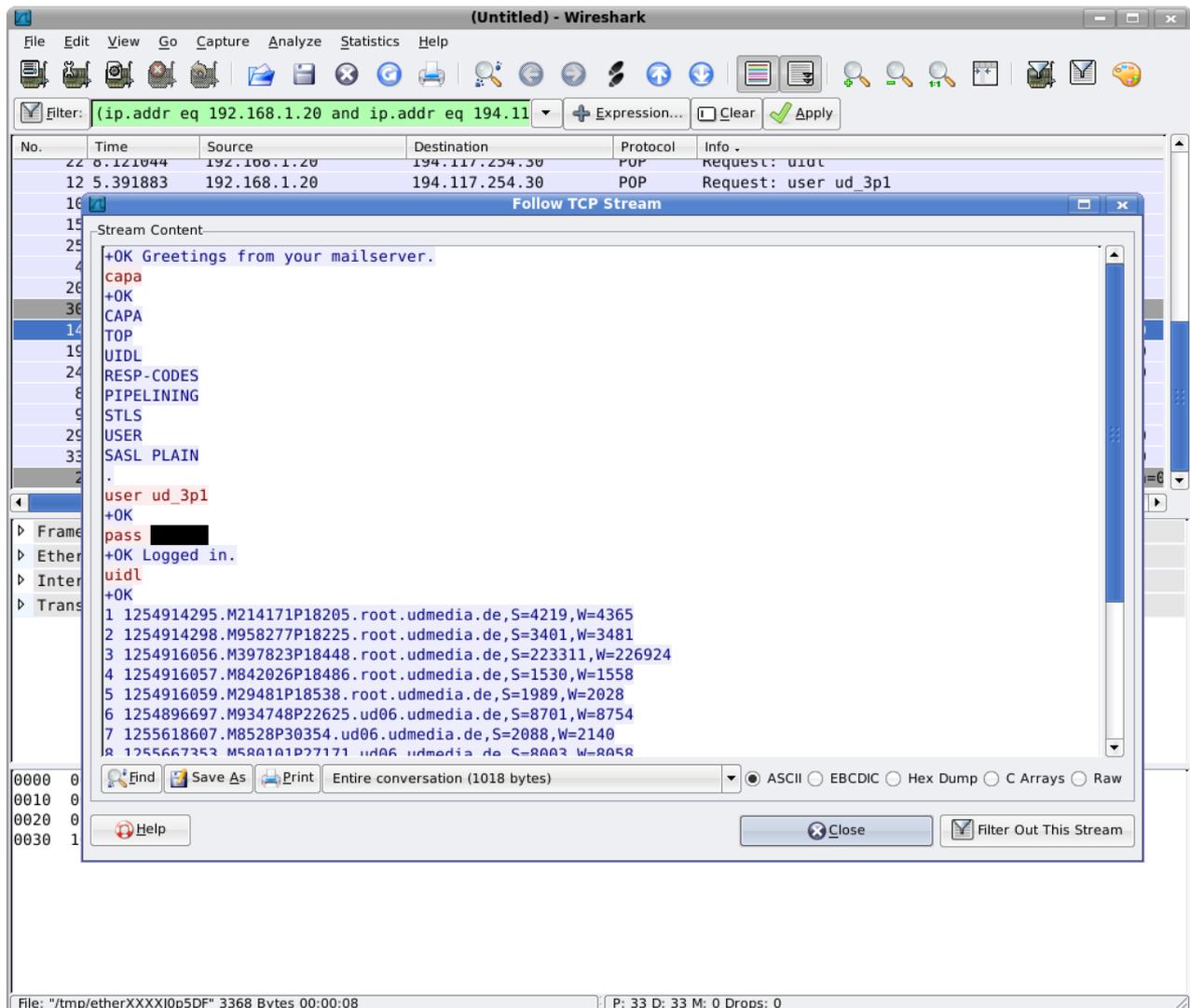
The status bar at the bottom indicates: File: "/tmp/etherXXXXVnHRPV" 4005 Bytes 00:01:09 P: 51 D: 51 M: 0 Drops: 0

Protokoll: POP3

Das POP3 „*Post Office Protocol*“ Protokoll dient zum Abruf von E-Mails in einem Postfach. Es ist ein äußerst simples Protokoll und unterstützt beispielsweise kein verschieben von E-Mails in Ordner oder eine servserseitige Suche.

Die Authentifizierung erfolgt bei POP3 üblicherweise in Klartext, sofern nicht SSL/TLS verwendet wird. Die Authentifizierungsdaten werden hierbei direkt in Klartext übertragen.

In der Aufzeichnung ersichtlich, ist zudem eine Liste von E-Mails, die in dem Postfach enthalten sind. Der Inhalt der E-Mails kann über den Befehl „*RETR*“ abgerufen werden.



The image shows a Wireshark capture of a POP3 session. The main window displays a list of packets with the following columns: No., Time, Source, Destination, Protocol, and Info. The selected packet (No. 12) is a POP3 Request: user ud_3p1. A 'Follow TCP Stream' window is open, showing the stream content in ASCII format. The stream content includes the following text:

```
+OK Greetings from your mailserver.  
capa  
+OK  
CAPA  
TOP  
UIDL  
RESP-CODES  
PIPELINING  
STLS  
USER  
SASL PLAIN  
.  
user ud_3p1  
+OK  
pass ██████████  
+OK Logged in.  
uidl  
+OK  
1 1254914295.M214171P18205.root.udmedia.de,S=4219,W=4365  
2 1254914298.M958277P18225.root.udmedia.de,S=3401,W=3481  
3 1254916056.M397823P18448.root.udmedia.de,S=223311,W=226924  
4 1254916057.M842026P18486.root.udmedia.de,S=1530,W=1558  
5 1254916059.M29481P18538.root.udmedia.de,S=1989,W=2028  
6 1254896697.M934748P22625.ud06.udmedia.de,S=8701,W=8754  
7 1255618607.M8528P30354.ud06.udmedia.de,S=2088,W=2140  
8 1255667252.M580101D27171.ud06.udmedia.de,S=2002,W=2058
```

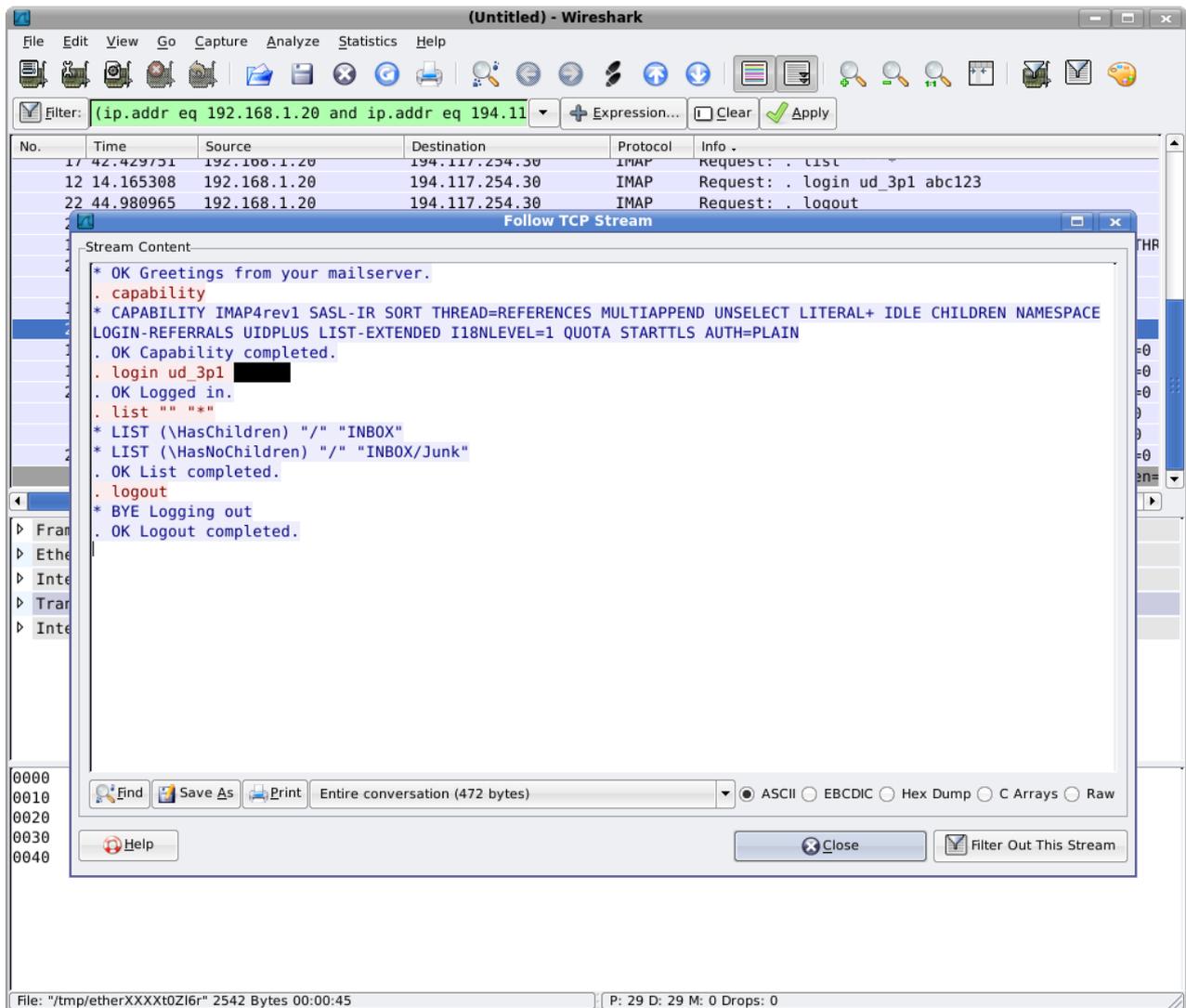
The bottom of the window shows the status bar with the file path: "/tmp/etherXXXXJ0p5DF" 3368 Bytes 00:00:08 and the packet count: P: 33 D: 33 M: 0 Drops: 0.

Protokoll: IMAP

Das IMAP „*Internet Message Access Protocol*“ Protokoll, dient wie das POP3-Protokoll, zur Übertragung von E-Mails, unterstützt jedoch eine Vielzahl an zusätzlichen Funktionen.

Üblicherweise wird es verwendet, um E-Mails am Server zu belassen, gleichzeitig jedoch effizient mit einer hohen Anzahl an E-Mails zu arbeiten. So werden die Befehle meist serverseitig ausgeführt.

Der Login bei IMAP in IMAP erfolgt ebenfalls im Klartext, sofern kein Login über einen Hash-Wert verwendet wird. Ebenfalls möglich ist die sichere Verwendung von SSL/TLS.



The screenshot shows a Wireshark capture of an IMAP session. The main pane displays a list of packets with IMAP requests. A 'Follow TCP Stream' window is open, showing the raw text of the IMAP conversation.

No.	Time	Source	Destination	Protocol	Info
17	42.429751	192.168.1.20	194.117.254.30	IMAP	Request: . list
12	14.165308	192.168.1.20	194.117.254.30	IMAP	Request: . login ud_3pl abc123
22	44.980965	192.168.1.20	194.117.254.30	IMAP	Request: . logout

```
* OK Greetings from your mailserver.
. capability
* CAPABILITY IMAP4rev1 SASL-IR SORT THREAD=REFERENCES MULTIAPPEND UNSELECT LITERAL+ IDLE CHILDREN NAMESPACE
LOGIN-REFERRALS UIDPLUS LIST-EXTENDED I18NLEVEL=1 QUOTA STARTTLS AUTH=PLAIN
. OK Capability completed.
. login ud_3pl [REDACTED]
. OK Logged in.
. list "" "*"
* LIST (\HasChildren) "/" "INBOX"
* LIST (\HasNoChildren) "/" "INBOX/Junk"
. OK List completed.
. logout
* BYE Logging out
. OK Logout completed.
```

Protokoll: AOL

Das AOL- bzw. ICQ-Protokoll ist ein Chatprotokoll und wurde ursprünglich von AOL entwickelt. Der Login hierbei erfolgt heutzutage zwingend über eine Hash-Wert Berechnung, weshalb das nicht Passwort lesbar ist. Die Übertragung der einzelnen Nachrichten erfolgt jedoch nach wie vor im Klartext.

In diesem Screenshot ersichtlich ist eine ausgehende Nachricht an den Benutzer mit der Nummer *15xy00xy1* (zensiert) mit dem Inhalt „sniffer test“.

The screenshot shows a Wireshark capture on the eth1 interface. The filter is set to (ip.addr eq 192.168.1.20 and ip.addr eq 64.12.25.168). The packet list shows an AIM message (No. 89) from 192.168.1.20 to 64.12.25.168. The packet details pane shows the AIM message structure, including the message content: "sniffer test". The packet bytes pane shows the raw data in hexadecimal and ASCII, with the message content visible in ASCII as ".1555000 51.....sniffer test".

No.	Time	Source	Destination	Protocol	Info
80	34.313002	192.168.1.20	64.12.25.168	TCP	1536 > aol [ACK] Seq=1001 Ack=10280 Win=17077 Len=...
81	51.415883	192.168.1.20	64.12.25.168	AIM Mess	AIM Messaging, Outgoing to: 155500051
82	51.540714	64.12.25.168	192.168.1.20	AIM Gene	AIM Generic, Rate Change
83	51.682967	64.12.25.168	192.168.1.20	AIM Mess	AIM Messaging, Acknowledge
84	51.684923	192.168.1.20	64.12.25.168	TCP	1536 > aol [ACK] Seq=1743 Ack=10525 Win=16832 Len=...
85	59.575667	192.168.1.20	64.12.25.168	AIM	Keep Alive
86	59.752853	64.12.25.168	192.168.1.20	TCP	aol > 1536 [ACK] Seq=10525 Ack=1749 Win=16384 Len=...
89	66.725499	192.168.1.20	64.12.25.168	AIM Mess	AIM Messaging, Outgoing to: 155500051
90	66.834584	64.12.25.168	192.168.1.20	AIM Gene	AIM Generic, Rate Change
91	66.836728	64.12.25.168	192.168.1.20	AIM Mess	AIM Messaging, Acknowledge
92	66.837244	192.168.1.20	64.12.25.168	TCP	1536 > aol [ACK] Seq=1822 Ack=10770 Win=16587 Len=...
93	81.476897	192.168.1.20	64.12.25.168	AIM Mess	AIM Messaging, Outgoing to: 155500051
94	81.587586	64.12.25.168	192.168.1.20	AIM Gene	AIM Generic, Rate Change
95	81.589382	64.12.25.168	192.168.1.20	AIM Mess	AIM Messaging, Acknowledge
96	81.589900	192.168.1.20	64.12.25.168	TCP	1536 > aol [ACK] Seq=1894 Ack=11015 Win=17640 Len=...
97	116.579280	192.168.1.20	64.12.25.168	AIM	Keep Alive

AIM Messaging, Outgoing

- ICBM Cookie: 6EE9E04AFE020000
- Message Channel ID: 0x0001
- Buddy: 155500051
- TLV: Message Block
 - Value ID: Message Block (0x0002)
 - Length: 25
 - Message: sniffer test
- TLV: Server Ack Requested
- TLV: Message was received offline

```
0000 00 19 d2 c2 b6 e0 00 21 63 90 44 2d 08 00 45 00 .....! c.D...E.
0010 00 71 fc e7 40 00 80 06 e2 2e c0 a8 01 14 40 0c .q.@... ..@.
0020 19 a8 06 00 14 46 bc a9 55 85 e8 4d cc 1a 50 18 .....F.. U..M..P.
0030 41 c0 af b7 00 00 2a 02 19 10 00 43 00 04 00 06 A.....*...C....
0040 00 00 00 6d 00 06 6e e9 e0 4a fe 02 00 00 00 01 ...m..n..J.....
0050 09 31 35 35 30 30 30 30 35 31 00 02 00 19 05 01 .1555000 51.....
0060 00 01 01 01 01 00 10 00 03 00 00 73 6e 69 66 66 .....sniff
0070 65 72 20 74 65 73 74 00 03 00 00 00 06 00 00 er test. ....
```

Protokoll: Jabber

Das Jabber-Protokoll, dient wie das AOL-Protokoll, zum Chatten, ist jedoch als offener Standard definiert. Das heißt jeder kann einen Jabber-Server sowie -Client entwickeln und man ist nicht an einen einzelnen Anbieter gebunden.

Wie beim AOL-Protokoll ist der Login ebenfalls Hash-Wert-Verschlüsselt, die Nutzdaten jedoch nach wie vor im Klartext, und werden in XML übertragen. In diesem Screenshot ersichtlich ist eine ausgehende Nachricht an den Benutzer „exy-b@jabber.org“ (zensiert) mit dem Inhalt „sniffer test“.

Zur Vollständigkeit sei auch hier erwähnt, dass auch Jabber die Verschlüsselung über SSL/TLS unterstützt.

The screenshot shows the Wireshark interface with a capture of Jabber/X traffic. The main pane displays a list of packets, with packet 52 selected. The packet list shows the following details:

No.	Time	Source	Destination	Protocol	Info
45	5.773120	208.68.163.220	192.168.1.20	Jabber/X	Response: <iq from='christian.fuchs@gmail.com' and
46	5.902920	192.168.1.20	208.68.163.220	TCP	1538 > xmpp-client [ACK] Seq=2359 Ack=4912 Win=16
47	22.045903	208.68.163.220	192.168.1.20	Jabber/X	Response: <presence from='christian.fuchs@gmail.c
48	22.200823	192.168.1.20	208.68.163.220	TCP	1538 > xmpp-client [ACK] Seq=2359 Ack=5313 Win=16
49	41.589489	192.168.1.20	208.68.163.220	Jabber/X	Request: <iq type='get' to='edy-b@jabber.org' id=
50	41.719952	208.68.163.220	192.168.1.20	TCP	xmpp-client > 1538 [ACK] Seq=5313 Ack=2447 Win=13
51	41.720267	208.68.163.220	192.168.1.20	Jabber/X	Response: <iq from='edy-b@jabber.org' to='manuelr
52	41.721188	192.168.1.20	208.68.163.220	Jabber/X	Request: <message type='chat' to='edy-b@jabber.or
53	41.889586	192.168.1.20	208.68.163.220	TCP	1538 > xmpp-client [ACK] Seq=2537 Ack=5554 Win=17
54	41.893476	208.68.163.220	192.168.1.20	TCP	xmpp-client > 1538 [ACK] Seq=5554 Ack=2537 Win=13
55	41.895028	192.168.1.20	208.68.163.220	Jabber/X	Request: <iq type='get' to='edy-b@jabber.org' id=
56	42.029478	208.68.163.220	192.168.1.20	TCP	xmpp-client > 1538 [ACK] Seq=5554 Ack=2645 Win=13
57	42.029750	208.68.163.220	192.168.1.20	Jabber/X	Response: <iq from='edy-b@jabber.org' to='manuelr
58	42.217777	192.168.1.20	208.68.163.220	TCP	1538 > xmpp-client [ACK] Seq=2645 Ack=5782 Win=17
59	60.140865	192.168.1.20	208.68.163.220	Jabber/X	Request: \t
60	60.313790	208.68.163.220	192.168.1.20	TCP	xmpp-client > 1538 [ACK] Seq=5782 Ack=2648 Win=13

The packet details pane for packet 52 shows the following XML structure:

```
<message type='chat' to='edy-b@jabber.org' id='mir_19'>
  <body>
    sniffer test
  </body>
</message>
```

The packet bytes pane shows the raw data in hexadecimal and ASCII:

```
0000 00 19 d2 c2 b6 e0 00 21 63 90 44 2d 08 00 45 00 .....! c.D...E.
0010 00 82 fd 2d 40 00 80 06 c7 6a c0 a8 01 14 d0 44 ...-@... .j....D
0020 a3 dc 06 02 14 66 5b dc 84 85 09 74 17 24 50 18 .....f[. ...t.$P.
0030 40 25 e6 db 00 00 3c 6d 65 73 73 61 67 65 20 74 @%...<m message t
0040 79 70 65 3d 27 63 68 61 74 27 20 74 6f 3d 27 65 ype='cha t' to='e
0050 64 79 2d 62 40 6a 61 62 62 65 72 2e 6f 72 67 27 dy-b@jab ber.org'
0060 20 69 64 3d 27 6d 69 72 5f 31 39 27 3e 3c 62 6f id='mir_19'<bo
0070 64 79 3e 73 6e 69 66 66 65 72 20 74 65 73 74 3c dy>sniff er test<
0080 2f 62 6f 64 79 3e 3c 2f 6d 65 73 73 61 67 65 3e /body></ message>
```