REIHE INFORMATIK
TR-2005-10

Kapazitätsmessung eines verdeckten Zeitkanals über HTTP

Hans-Georg Eßer, Felix C. Freiling

Universität Mannheim
Praktische Informatik I
A5, 6
D-68159 Mannheim, Germany

Kapazitätsmessung eines verdeckten Zeitkanals über HTTP

Hans-Georg Eßer Felix C. Freiling
Universität Mannheim, Praktische Informatik I, A5, 6, D-68159 Mannheim

Abstract: Wir beschreiben die Implementierung eines einfachen verdeckten Zeitkanals über HTTP und evaluieren dessen Kapazität im Internet. Im Experiment kommunizierte ein leicht modifizierter Apache-Webserver mit einem selbst geschriebenen HTTP-Proxy auf der Seite des Clients. Optimiert man den Kanal auf Fehlerfreiheit, können 3 Bit/s übertragen werden; akzeptiert man bis zu 10 % Fehler, sind 14 Bit/s möglich. Die einfache Machbarkeit demonstriert erneut die Gefährlichkeit verdeckter Kanäle auch für Heimanwendungen.

1 Einleitung

Es ist seit langem bekannt, dass selbst ein perfekter Zugriffsschutzmechanismus keine vollständige Vertraulichkeit erreichen kann. Die Ursache hierfür sind die so genannten *verdeckten Kanäle*. Ein Kommunikationskanal gilt als verdeckt, wenn er ursprünglich nicht für Kommunikation eingerichtet wurde. Als Beispiel für das Vorhandensein eines verdeckten Kanals wird häufig das Beispiel eines fiktiven Krankenhausinformationssystems herangezogen. In diesem System hat jeder Patient vollen Zugriff auf seine Krankenakte, die alle ärztlichen Diagnosen und medizinischen Messwerte enthält. Um die Psyche der Patienten zu schützen, sollen tödliche Diagnosen (wie etwa Krebs) den Behandelten „schonend“, also im persönlichen Gespräch mit einem Arzt, beigebracht werden. Umgesetzt wird dies, indem das System allen Patienten mit einer solchen Diagnose den Zugriff auf ihre Daten verweigert. Mit dem Wissen über diese Systemeigenheit kann ein Patient jedoch allein aus der Tatsache, dass ihm der Zugriff verweigert wurde, auf eine tödliche Diagnose schließen. Der Zugriffsschutzmechanismus wird zum verdeckten Kanal.

Der Begriff des verdeckten Kanals geht auf eine Arbeit von Lampson [Lam73] zurück. Man unterscheidet zwischen *Ressourcen-* und *Zeitkanälen*. Während erstere durch den gemeinsamen Zugriff der beteiligten Prozesse auf eine Systemressource (wie Plattenplatz, Rechenzeit, Hauptspeicher) geprägt sind, arbeiten Zeitkanäle mit manipulierten zeitlichen Abständen zwischen zwei Systemereignissen. Das in diesem Zusammenhang immer wieder heraufbeschworene Szenario enthält ein Trojanisches Pferd innerhalb einer besonders geschützten Systemumgebung, das mit einem Spion außerhalb dieser Umgebung über einen verdeckten Kanal kommuniziert. Zeitkanäle leiden oft unter Rauschen: Das Zeitverhalten des Systems ist nicht nur vom sendenden Prozess auf dem verdeckten Kanal abhängig, sondern auch vom restlichen System- und Netzwerkverhalten.

Verdeckte Kanäle stellen eines der am schwersten beherrschbaren Phänomene in der IT-Sicherheit dar. Während frühere Arbeiten die Gefahren im Wesentlichen im (militärischen) Hochsicherheitsbereich sahen [Eck03], ist heute aufgrund der steigenden Verbreitung von Viren, Würmern und Trojanischen Pferden die Gefährdung von Firmen und Privatpersonen unabstreitbar. In der industriellen Praxis wird versucht, verdeckte Kanäle durch Designmethoden [Kem83] oder komplexe Schleusentechnologien [HEM00] zu erkennen und zu vermeiden. In privaten Netzen verhindern Techniken wie *Network Address Translation* (NAT) oder eine Firewall ungewollten Datenverkehr. Wie diese Arbeit zeigt, bleiben die Gefahren durch verdeckte Kanäle dort jedoch grundsätzlich bestehen.

Der Ausgangspunkt für die vorliegende Arbeit war eine Adaption des oben beschriebenen Szenarios: Ein Nutzer unterhält einen abgesicherten Rechner mit vertraulichen Daten. Auf dem Rechner läuft als einziger Dienst ein Webserver, der einen gewöhnlichen HTTP-Zugang über den TCP-Netzwerkport 80 anbietet. Auf dem Server wurde jedoch ein Trojanisches Pferd platziert, welches mit einem Nutzer im Internet kommunizieren möchte. Als Kommunikationskanal steht ausschließlich ein verdeckter Zeitkanal über Port 80 zur Verfügung. Die Fragestellung, die hier untersucht wird, lautet: Wie einfach ist es, einen verdeckten Zeitkanal über HTTP zu realisieren und welche Kapazität (d. h., wie viele Bits pro Sekunde) kann man über diesen Kanal kommunizieren?

Um den Punkt der „Einfachheit“ zu erfüllen, wurden folgende Anforderungen gestellt:

1. Die Kommunikation soll ausschließlich über den Zeitkanal auf einer HTTP-Verbindung durchgeführt werden; der eigentliche Inhalt der Verbindung darf nicht verändert werden.

2. Der Zeitkanal soll ausschließlich das Kommunikationsverhalten der Webanwendung ausnutzen und beispielsweise nicht zusätzlich noch Verzögerungen von TCP- oder IP-Paketen verwenden.

3. Insgesamt sollen die Modifikationen des Webservers durch den Trojaner minimal bleiben.

Die Anforderungen 1 und 2 wurden besonders restriktiv gewählt, um Evidenzen für die *Mindestkapazität* eines solchen Kanals zu sammeln. Anforderung 3 ist in gewissem Sinne konkurrierend zu Anforderungen 1 und 2 und sollte die Herausforderung verstärken, maximale Kapazität mit minimalen Veränderungen zu erreichen.

Um die Fragestellung dieser Arbeit zu beantworten, wurde eine HTTP-Verbindung zwischen einem standardmäßigen Apache-Webserver [Apa] und einem HTTP-Proxy als Trägerkanal für einen Zeitkanal verwendet. Der Webserver verzögert beim Übertragen einer größeren Datei einzelne Blöcke auf der TCP-Verbindung abhängig davon, ob auf dem verdeckten Kanal der Wert 0 oder 1 übertragen werden soll. Die auf diese Weise maximal erreichbare relative Kapazität ist offensichtlich 1 Bit pro Block. Die Blockgröße, die vom Apache-Webserver verwendet wird, beträgt 4 KByte. Mit Messungen in verschiedenen Szenarien wurde untersucht, wie gut der optimale Wert erreicht wird und damit wie klein das Rauschen auf dem Kanal ist. Zusätzlich wurde der Nutzen von einfachen Fehlerkorrekturverfahren mit Hamming-Codes evaluiert.

Die Maximalkapazität des hier vorgestellten Zeitkanals hängt direkt von der gewählten Verzögerungszeit einzelner Bits ab und schwankt in den Messungen zwischen 4 und 28 Bit/s; für Fehlerraten unter 5 % liegt die Kapazität zwischen 4 und 9 Bit/s.

Die Mehrzahl der Arbeiten zu verdeckten Kanälen nutzt vorhandene (legitime) Kanäle, um durch Veränderungen des regulären Kanals verdeckte Botschaften zu übertragen: Dyatlov und Castro [DC03] verwenden bei HTTP-Übertragungen Zusatz-Header in den Bodies von HTTP-Anfrage und -Antwort sowie die URL selbst. Infranet [FBH+02] ist ein HTTP-Tunnel mit der speziellen Zielsetzung, Internet-Zensur zu umgehen: Die Anfrage wird über die URL kodiert, für die Antwort greifen die Autoren auf Steganographie zurück. Auch generische HTTP-Tunnel-Programme (etwa GNU httptunnel [Htt]) erzeugen verdeckte Kommunikation, da die Nutzung einer HTTP-Verbindung für andere Zwecke als den Zugriff auf Webserver unerwartet ist. Rutkowska [Rut04] und das LOKI-Projekt [Dae96, Dae97] verändern redundante Inhalte von TCP-Paketen und ungenutzte Bereiche in ICMP-Echo-Paketen zur Implementierung eines verdeckten Kanals. Alle genannten Arbeiten erzeugen somit Ressourcenkanäle, während die hier beschriebene Arbeit einen Zeitkanal aufbaut. Verwandte Arbeiten mit dem Ziel der Verschleierung von Kommunikation sind so genannte MIX-Netze [Cha81, Bau03]: Dort tauscht eine große Anzahl Teilnehmer parallel Informationen miteinander aus; eventuell auch dann, wenn gerade keine Daten zu übertragen sind. Insbesondere in den *stop-and-go* MIX-Netzen [KEB98] geht es aber anders als in dieser Arbeit gerade darum, Zeitkanäle durch das Einbringen zufälliger Verzögerungszeiten zu eliminieren.

Abschnitt 2 beschreibt die Software-Komponenten, die für die Tests entwickelt wurden, die eingesetzte Hardware und die Netzwerkanbindung der Komponenten. In Abschnitt 3 werden die Messergebnisse präsentiert und diskutiert. Schließlich bietet Abschnitt 4 eine Zusammenfassung, Verweise auf ähnliche Arbeiten und einen Ausblick. Die hier präsentierten Ergebnisse sind zusammen mit einigen weiteren Messungen an anderem Ort ausführlicher dokumentiert [Eße05].

2 Experiment

In diesem Abschnitt beschreiben wir den Versuchsaufbau, also die eingesetzte Soft- und Hardware sowie die verwendete Netzwerkanbindung.

2.1 Software

Der Aufbau der Experiments ist in Abbildung 1 (auf der folgenden Seite) dargestellt. Der Rechner des „Spions“ ist ganz links abgebildet. Er kommuniziert über die HTTP-Verbindung mit dem Apache-Webserver (Mitte).

Die Software-Architektur auf dem Webserver ist rechts skizziert. Um den verdeckten Kanal zu erzeugen, wurden drei Komponenten eingesetzt: (1) Apache-Server, (2) Kontroll-Daemon „cc daemon“ und (3) HTTP-Proxy (der auf dem Rechner ganz links läuft). Das

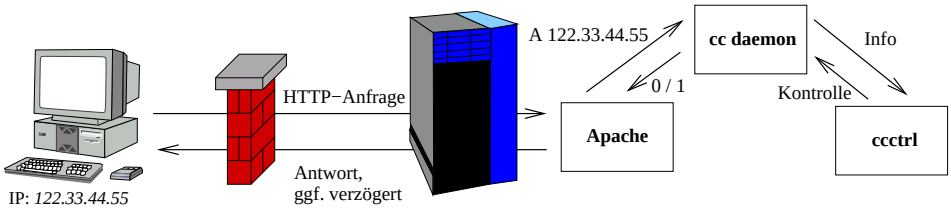

Abbildung 1: Kommunikation zwischen Client und Server sowie lokal auf dem Server.

Modul „ccctrl“ ist das eigentliche Trojanische Pferd, welches über den Kontroll-Daemon den Inhalt der versendeten Nachrichten steuern kann. In der vorliegenden Implementierung kann es über zahlreiche Kommandozeilenparameter gesteuert werden [Eße05].

Die Kommunikation läuft wie folgt ab: Zunächst stellt ein Webclient über den HTTP-Proxy eine Anfrage an den Webserver. Der Webserver fragt bei jedem Zugriff beim Kontroll-Daemon an, ob eine verdeckte Nachricht gesendet werden soll. Der Kontroll-Daemon prüft in seiner Datenbank, ob die IP-Adresse in der Liste der geheimen Kommunikationspartner enthalten ist. Falls ja, schickt er das nächste Bit einer für diese IP-Adresse gespeicherten verdeckten Nachricht als 0 oder 1 kodiert an den Apache-Server. Abhängig von der Antwort des Kontroll-Daemons verschickt der Apache-Server den aktuellen Block verzögert (1) oder unverzögert (0). Der HTTP-Proxy misst die Übertragungszeit (die Zeit zwischen dem Aufruf von `read()` und dem Empfang des Blocks) und schreibt diese Zeiten in eine Protokolldatei. Außerdem reicht er den eigentlichen Inhalt (also die Daten des Trägerkanals) an den Webclient weiter. Nachdem die Datei komplett übertragen wurde, wird die Protokolldatei des HTTP-Proxys mit statistischen Methoden analysiert, um die verdeckte Nachricht zu rekonstruieren.

Um eine möglichst geringe Menge an Bits zu übertragen, wurde der Bereich der zulässigen Zeichen auf ASCII-Zeichen zwischen 32 und 95 (Großbuchstaben, Ziffern und einige Sonderzeichen) eingeschränkt. Durch die Beschränkung auf 64 mögliche Zeichen ist ein Zeichen in 6 Bit kodierbar. Jede Nachricht erhält – zur besseren Erkennung – Anfangs- und Endmarkierungen.

2.2 Veränderungen am Apache

Der Apache-Webserver ist nur geringfügig modifiziert, um den Kontakt zu einem Daemon zu erlauben, der die verdeckte Kommunikation steuert; die Patch-Datei für Apache 1.3.31 besteht aus 222 Zeilen. Verändert wurden im Wesentlichen die Funktionen `ap_send_fd_length()` und `ap_send_mmap()`, die den Versand von Datenblöcken regeln; außerdem die Variablen `MMAP_SEGMENT_SIZE` und `IOBUFSIZE`, welche die Größe der zu versendenden Blöcke festlegen. Der Daemon für die Steuerung der verdeckten Kanäle ist ein 439 Zeilen langes Python-Programm, das Analyse-Tool besteht aus 265 Zeilen Python-Code. Insgesamt mussten also nur äußerst geringe Modifikationen an Standardsoftware vorgenommen werden. Der Quellcode ist im Internet frei verfügbar [Eße05].

2.3 Hardware und Netzwerk

Für die Tests wurden zwei Rechner verwendet: Der Server (Intel Celeron, 2 GHz, 256 MByte RAM, Debian Linux 3.0) steht im Rechenzentrum eines Hosting-Anbieters, der Client (Intel Pentium IV, 2 GHz, 512 MByte RAM, Suse Linux 9.0) ist über eine DSL-Leitung (768 MBit/s *up-stream*, 128 MBit/s *down-stream*) mit dem Internet verbunden.

Der Apache-Webserver versendet die angeforderten Webseiten in Blöcken der Größe 4 KByte. Da die Aufgabenstellung vorsah, ausschließlich das Kommunikationsverhalten der Applikation auszunutzen, konnte pro solchem 4-KByte-Block nur jeweils ein Bit kodiert werden: verzögert (1) bzw. unverzögert (0). Die Verzögerung, mit welcher der Apache-Server den Bitwert 1 kodiert, ist ein Parameter der Installation und wurde in den Tests variiert, um die Kapazität des Kanals zu bestimmen. Für 15 Verzögerungswerte zwischen 0,05 s und 0,5 s wurden je fünf Testläufe ausgeführt, bei denen eine 5 MByte große Testdatei auf dem Trägerkanal übertragen wurde. Die vollständig unverzögerte Übertragung der Testdatei benötigte über diese Verbindung 42 s; bei Verzögerung von durchschnittlich jedem zweiten 4-KByte-Block um 0,5 s erhöhte sich die Übertragungszeit auf 322 s. Diese Zeit liegt im Rahmen der Erwartungen: Die 5 MByte große Datei besteht aus 1280 Blöcken zu je 4 KByte; wird jeder zweite (also 640 Blöcke) um 0,5 s verzögert, ergibt sich eine erwartete Übertragungszeit von 42 s + $640 \times 0,5$ s $= 362$ s.

Der experimentelle Aufbau, der auf dem Herunterladen einer einzelnen großen Datei basiert, ist natürlich unrealistisch, denn in der Praxis sind die versendeten Dateien im Durchschnitt viel kleiner. Ziel des Experimentes war jedoch nicht, ein praktisch verwendbares Hacker-Werkzeug zu entwickeln, sondern eine *worst case*-Abschätzung der Kapazität eines praktischen verdeckten Zeitkanals zu erhalten.

3 Ergebnisse

In diesem Abschnitt präsentieren wir die Messergebnisse. Gemessen wurden die Fehlerraten bei verschiedenen Verzögerungszeiten. Es war zu erwarten, dass kürzere Verzögerungen zu mehr Fehlern führen. Zunächst präsentieren wir einige Einzelergebnisse, um den Charakter der Übertragung zu beschreiben. Danach folgen eine Übersicht aller Messungen und eine Untersuchung der Frage, inwieweit der Versuchsaufbau die Messungen beeinflusst.

3.1 Übersicht

Abbildungen 2, 3 und 4 zeigen die gemessenen Übertragungszeiten für verschiedene Verzögerungen v. Auf der horizontalen Achse sind die Nummern der 4-KByte-Blöcke aufgezeichnet.

Bei einer Verzögerung von $v = 0,4$ s war die Übertragung zu 0,64 % fehlerbehaftet (siehe Abbildung 2 auf der folgenden Seite). Eine deutliche Trennung verzögerter und

unverzögerter Pakete in zwei Messwertblöcke ist optisch nicht erkennbar. Zwar gibt es knapp oberhalb von 0,4 s eine Häufung von Messwerten, aber ein Großteil der Werte ist über den gesamten Bereich knapp oberhalb von 0,03 s bis 0,6 s verteilt. Die Klassifizierung der Übertragungszeiten in „verzögert“ (1) und „unverzögert“ (0) wird auf Empfängerseite durch eine statistische Analyse bestimmt. Dabei geht das Analyseprogramm zunächst davon aus, dass sich die Menge der gemessenen Übertragungszeiten in einen oberen und einen unteren Block unterteilen lässt, so dass der Medianwert die Grenze darstellt. Einzelne Werte werden danach noch auf Basis ihrer Abweichung vom Block-Durchschnitt in den jeweiligen anderen Block verschoben.

***v* = 0,4 s**	
Gesamtwerte	
Median	0.0329700000
Mittelwert	0.1344422891
Abweichung	0.1570459346
Unterer Block	
Minimum	0.0219700000
Maximum	0.1325300000
Mittelwert	0.0361714355
Abweichung	0.0109453373
Oberer Block	
Minimum	0.0329800000
Maximum	0.6235500000
Mittelwert	0.3108148690
Abweichung	0.1423531659
Bitfehler	5
Fehlerrate	0,6 %

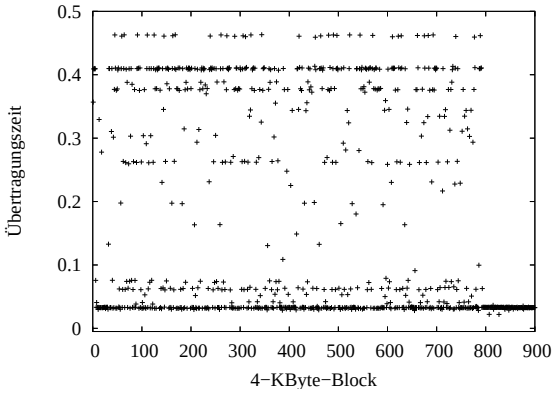

Abbildung 2: Messwerte und Grafik für den Test mit $v = 0,4$ s.

Bei $v = 0,1$ s Verzögerung ergaben sich die Werte aus Abbildung 3. Die Fehlerrate von 17,1 % macht den Kanal nahezu unbrauchbar.

***v* = 0,1 s**	
Gesamtwerte	
Median	0.0328300000
Mittelwert	0.0454305625
Abweichung	0.0272215441
Unterer Block	
Minimum	0.0220300000
Maximum	0.0451800000
Mittelwert	0.0329082048
Abweichung	0.0020199164
Oberer Block	
Minimum	0.0328400000
Maximum	0.3264800000
Mittelwert	0.0719072749
Abweichung	0.0356072184
Bitfehler	133
Fehlerrate	17,1 %

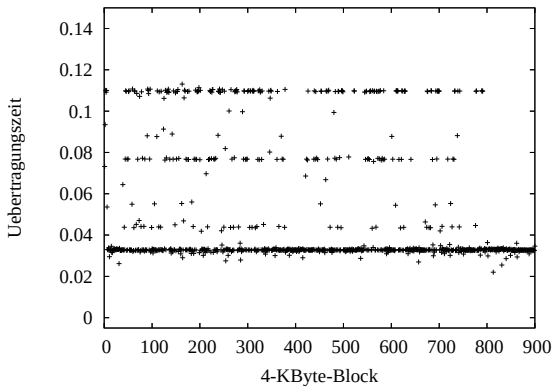

Abbildung 3: Messwerte und Grafik für den Test mit $v = 0,1$ s.

Eine Verzögerung von $v = 0,05$ s führte zur völligen Unbrauchbarkeit des Kanals: Mit 50,64 % Fehlerrate war der Kanal vollständig gestört. Abbildung 4 auf der folgenden Seite

zeigt, dass sich fast alle Messwerte in einem kleinen Intervall um 0,0033 s befinden – die erwartete Aufteilung in zwei Blöcke gab es nicht.

***v* = 0,05 s**	
Gesamtwerte	
Median	0.0327600000
Mittelwert	0.0329584219
Abweichung	0.0069252043
Unterer Block	
Minimum	0.0273100000
Maximum	0.0329800000
Mittelwert	0.0326012435
Abweichung	0.0004558330
Oberer Block	
Minimum	0.0327700000
Maximum	0.2757000000
Mittelwert	0.0340166873
Abweichung	0.0137249371
Bitfehler	395
Fehlerrate	50,64 %

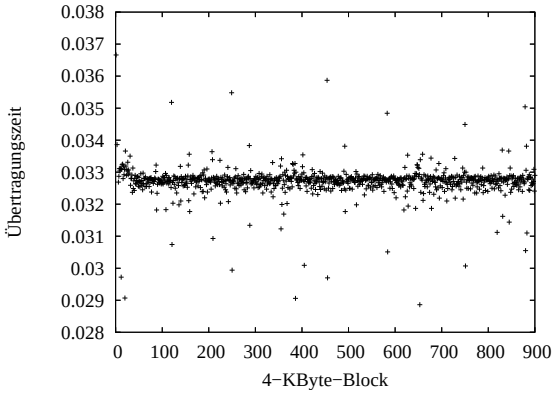

Abbildung 4: Messwerte und Grafik für den Test mit $v = 0,05$ s.

3.2 Ergebnisse ohne Fehlerkorrektur

Abbildung 5 zeigt die Fehlerquoten für die verschiedenen Verzögerungszeiten im linken Graphen mit linearer y-Achse und im rechten Graphen mit logarithmischer y-Achse. Zum Vergleich wurde in beide Graphen die Exponentialfunktion hineingelegt. Man kann hier feststellen, dass die Fehlerquote exponentiell steigt, wenn die Verzögerung reduziert wird.

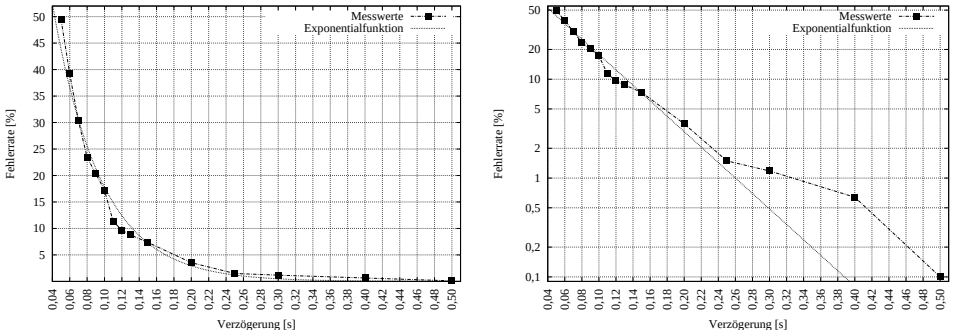

Abbildung 5: Zusammenhang zwischen Verzögerung und Fehlerrate (links: lineare y-Achse, rechts: logarithmische y-Achse; Verbindungslinien nur zur besseren Veranschaulichung).

Für $v = 0,05$ s (und kleinere v) lag die Fehlerrate bei 50 % – hier war keine Unterscheidung zwischen verzögerten und unverzögerten Blöcken mehr möglich, und das Analyseprogramm auf der Empfängerseite interpretierte alle Blöcke gleich. Da die Nullen und Einsen in der verdeckten Nachricht annähernd gleich verteilt sind, ist 50 % das schlechtestmögliche Ergebnis.

Bei den Tests mit 0,06 s $\leq v \leq$0,15 s wurden 39,18 % bis 7,36 % der Bits fehlerhaft übertragen. Ab $v = 0,2$ s lag die Fehlerquote unter 3,54 %, so dass die Hoffnung bestand, dass der Einsatz eines Fehlerkorrekturverfahrens für meist korrekte Übertragungen sorgen könnte. (Mehr dazu im folgenden Unterkapitel.) Selbst bei einer halben Sekunde Verzögerung traten noch Fehler auf – im Mittel waren 0,1 % der Bits falsch.

3.3 Ergebnisse mit Fehlerkorrektur

Im Anschluss an die Testreihe wurde ein einfaches Fehlerkorrekturverfahren auf Basis von Hamming-Codes in die Kommunikation integriert. Jeweils sechs Code-Bits wurden um vier Hamming-Bits erweitert. Ein solcher Code kann bekanntermaßen einen 1-Bit-Fehler korrigieren und zwei 1-Bit-Fehler entdecken [RF89]. Ohne Rückkanal (wie im vorliegenden Fall) ist die Fehlerentdeckung uninteressant und wird im Folgenden nicht weiter betrachtet.

Wir betrachten zunächst analytisch den Nutzen der Fehlerkorrektur: Wir fragen, mit welcher Wahrscheinlichkeit ein Zeichen (6 Bit) vollständig korrekt übertragen wird. Dazu sei p die Fehlerwahrscheinlichkeit für die Übertragung eines Bits.

Im Fall ohne Fehlerkorrektur werden sechs Bits übertragen. Die Wahrscheinlichkeit, dass alle sechs Bits korrekt den Empfänger erreichen, ist $p_1 := (1-p)^6$. Im Fall mit Fehlerkorrektur werden die sechs Bits um vier zusätzliche Bits ergänzt. Die Fehlerkorrektur kann einen Fehler korrigieren, ab zwei Fehlern ist das Ergebnis falsch. Damit ist die Wahrscheinlichkeit für die korrekte Übertragung die Summe zweier Teilwahrscheinlichkeiten: Die erste Teilwahrscheinlichkeit (Übertragung aller zehn Bits ohne Fehler) ist analog zum oberen Fall $(1-p)^{10}$. Die zweite Teilwahrscheinlichkeit (Übertragung der zehn Bits mit einem Fehler) ist $\sum_{i=1}^{10} p(1-p)^9$. (Jeder der zehn Summanden in diesem Ausdruck steht für die Wahrscheinlichkeit, dass Bit Nummer i falsch übertragen und alle anderen Bits korrekt übertragen werden.) Da die beiden Teilwahrscheinlichkeiten disjunkte Ereignismengen beschreiben, kann man sie summieren. Die Wahrscheinlichkeit, dass ein Zeichen mit Fehlerkorrektur korrekt übertragen wird, ergibt sich zu:

$$p_2 := (1-p)^{10} + \sum_{i=1}^{10} p(1-p)^9 = (1-p)^{10} + 10p(1-p)^9$$

Für die Fehlerraten p aus den Tests ohne Fehlerkorrektur wurden die Werte p_1 und p_2 berechnet; Tabelle 1 auf der folgenden Seite listet die Fehlerraten $(1-p_i)$ auf.

Mit steigender Bitfehlerquote sinkt also der Nutzen der Fehlerkorrektur: Sie kann nur bei sehr kleinen Fehlerraten deutliche Verbesserungen bewirken.

Die analytisch gewonnenen Einsichten wurden nun experimentell überprüft. Abbildung 6 auf der übernächsten Seite zeigt den interessanten Bereich mit Verzögerungen zwischen 0,1 s und 0,5 s: Für 0,4 s und 0,5 s war die Übertragung mit aktivierter Fehlerkorrektur fehlerfrei, während ohne Fehlerkorrektur Übertragungsfehler auftraten. Für Verzögerungen von 0,15 s, 0,2 s und 0,3 s gab es trotz Fehlerkorrektur Übertragungsfehler, die Quote der korrekt übertragenen Daten war aber besser als ohne Fehlerkorrektur.

Verz.	p	$1-p_1$	$1-p_2$	$(1-p_2)/(1-p_1)$
–	0,01 %	0,060 %	0,000 %	0,001
–	0,05 %	0,300 %	0,001 %	0,004
0,50 s	0,10 %	0,599 %	0,004 %	0,007
0,40 s	0,64 %	3,779 %	0,178 %	0,047
0,30 s	1,18 %	6,874 %	0,588 %	0,086
0,25 s	1,49 %	8,614 %	0,923 %	0,107
0,20 s	3,54 %	19,447 %	4,668 %	0,240
0,15 s	7,36 %	36,789 %	16,455 %	0,447
0,12 s	9,69 %	45,748 %	25,191 %	0,551
0,11 s	11,31 %	51,332 %	31,487 %	0,613
0,10 s	17,23 %	67,846 %	53,493 %	0,788
0,09 s	20,33 %	74,428 %	63,407 %	0,852
0,08 s	23,41 %	79,815 %	71,824 %	0,900
0,07 s	30,35 %	88,584 %	85,606 %	0,966
0,06 s	39,18 %	94,939 %	94,846 %	0,999
0,05 s	49,49 %	98,339 %	98,833 %	1,005

Tabelle 1: Analytische Berechnung der Fehlerraten mit und ohne Fehlerkorrektur.

Bei 0,11 s und 0,12 s Verzögerung waren die Ergebnisse mit Fehlerkorrektur sogar schlechter als ohne. Hierbei ist allerdings zu beachten, dass die Datenbasis mit jeweils fünf Messungen (im Fall ohne Fehlerkorrektur) bzw. vier Messungen (im Fall mit Fehlerkorrektur) recht klein ist. Tabelle 2 führt neben den Fehlerraten auch die Übertragungszeiten (für die Übertragungen ohne Fehlerkorrektur) auf.

Verzögerung	Fehler (mit Korr.)	Fehler (ohne Korr.)	Zeit (min)
0,05	48,33	49,49	0:45
0,06	40,38	39,18	0:46
0,07	32,91	30,35	0:55
0,08	24,40	23,41	1:00
0,09	20,04	20,33	1:04
0,1	16,67	17,23	1:10
0,11	15,68	11,31	1:16
0,12	13,98	9,69	1:31
0,15	6,54	7,36	1:41
0,2	2,22	3,54	2:12
0,25	1,92	1,49	2:44
0,3	0,64	1,18	3:15
0,4	0,00	0,64	4:19
0,5	0,00	0,10	5:22

Tabelle 2: Übersicht der Ergebnisse mit und ohne Fehlerkorrektur.

Geht man davon aus, dass die übertragene Botschaft in natürlicher Sprache verfasst ist, kann man etwa 10 % der Zeichen falsch übertragen, ohne einen semantischen Verlust davonzutragen. Nach den gemessenen Werten sollte man in diesem Fall eine Verzögerung von knapp unter 0,25 s (ohne Fehlerkorrektur) oder von knapp unter 0,2 s (mit Fehlerkorrektur) wählen. Zu beachten ist jedoch, dass mit eingeschalteter Fehlerkorrektur die Kapazität des Kanals wegen der zusätzlich übertragenen Hamming-Bits eingeschränkt wird.

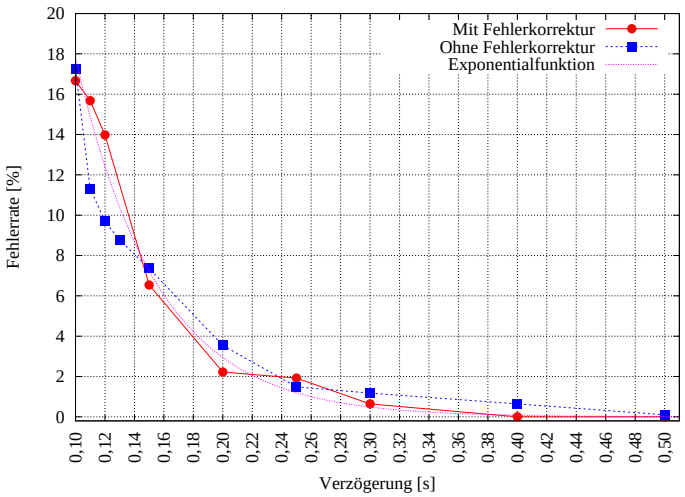

Abbildung 6: Ergebnisse mit und ohne Fehlerkorrektur im Vergleich; Verbindungslinien nur zur besseren Veranschaulichung.

Die Kapazität des verdeckten Kanals hängt einerseits von der Verwendung oder vom Verzicht auf Fehlerkorrektur, andererseits von der Verzögerungsrate ab. Aus den Übertragungszeiten in Tabelle 2 (für 1280 übertragene Bits) ergibt sich beispielsweise für v = 0,4 s eine Kapazität von $\frac{1280}{259} \approx 5$ Bit/s im Fall ohne Fehlerkorrektur. Bei Verwendung der Fehlerkorrektur sinkt die Kapazität um 40 % auf ca. 3 Bit/s. Für v = 0,12 s sind die entsprechenden Werte $\frac{1280}{91} \approx 14$ Bit/s (ohne Fehlerkorrektur) und ca. 8,4 Bit/s.

3.4 Entdeckbarkeit des verdeckten Kanals

Die verdeckte Kommunikation ist unsichtbar in dem Sinne, dass keine Daten verändert werden – allerdings führen die Verzögerungen zu einem im Vergleich zu einer normalen Übertragung auffälligen Zeitverhalten. Das lässt sich leicht mit einem Netzwerk-Sniffer beobachten. Für eine Verzögerungszeit von 0,3 s, die eine geringe Fehlerquote erlaubt, wurde der Netzwerkverkehr mit dem Hilfsprogramm Ethereal [eth] untersucht; dem wurden Vergleichsmessungen ohne Verzögerung gegenübergestellt.

Wird ein verdeckter Zeitkanal vermutet und mit Ethereal danach gesucht, fallen Unterschiede beim Datendurchsatz auf. Abbildung 7 auf der folgenden Seite zeigt links die Datendurchsatzstatistik (*throughput graph*) bei einer unverzögerten Übertragung. Deutlich erkennbar liegt die Rate während des gesamten Transfers einigermaßen konstant um 125 000 Byte/s. Die rechte Abbildung zeigt den von Ethereal ermittelten Datendurchsatz bei der Übertragung mit verdecktem Kanal – hier ergibt sich ein anderes Bild, der Maximalwert von 125 000 Byte/s wird nur in wenigen Fällen angenommen, die Werte sind zu einem großen Teil über den Bereich von 15 000 bis 85 000 Byte/s verschmiert.

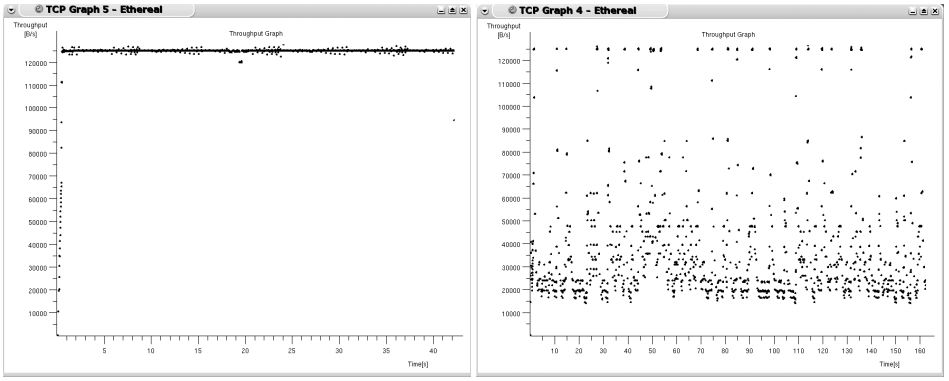

Abbildung 7: Von Ethereal beobachteter Datendurchsatz – links: unverzögert, rechts: verzögert.

4 Zusammenfassung

In dieser Arbeit wurde die Frage untersucht, ob und in welchem Umfang es möglich ist, einen verdeckten Zeitkanal auf einer Standard-HTTP-Verbindung aufzubauen. Es wurde nur der Fall betrachtet, dass Client und Server über das Internet kommunizieren. Ergänzende Untersuchungen im lokalen Netzwerk und völlig lokal (Server und Client auf dem gleichen Rechner) [Eße05] wurden hier nicht weiter behandelt, denn das Szenario mit einem Rechner im Internet ist das praxisnaheste.

Für die Entwicklung der Software waren nur wenige Tage nötig; die größte Aufgabe lag dabei in der Analyse des Apache-Quellcodes: Nachdem die für den Blockversand zuständigen Funktionen isoliert waren, konnte die Grundfunktionalität in wenigen Stunden integriert werden. Probleme, die sich aus der TCP-Paketsegmentierung ergaben, erforderten kleinere Korrekturen am Client-Proxy. Insgesamt ist es für einen erfahrenen Programmierer keine Herausforderung, einen verdeckten Kanal in einen vorhandenen Server einzubauen.

Die Kapazität des implementierten Zeitkanals ist bei hinreichend großer Verzögerung ausreichend, um kleinere Informationsmengen zu übertragen. Beispielsweise kann eine Kreditkartennummer in 54 Bit kodiert werden ($10^{16} < 2^{54}$) und bei Verzögerung $v = 0,5$ s in 14 s übertragen werden. Beim Einsatz des simplen Fehlerkorrekturverfahrens verlängert sich die Übertragungszeit, da statt 54 Bit nun 90 Bit nötig sind, auf 23,3 s.

Die Untersuchung hat gezeigt, dass vorhandene Software mit geringem Aufwand um einen verdeckten Kanal ergänzt werden kann, der keine inhaltlichen Veränderungen an den übertragenen Daten vornimmt und somit durch eine Überwachung dieser Daten nicht aufzufinden ist – das ist lediglich durch Beobachtung der zeitlichen Auffälligkeiten möglich. Die Machbarkeit – und damit auch die Gefährlichkeit – verdeckter Kanäle wurde damit experimentell nachgewiesen.

Literatur

[Apa] Apache Web-Server. `http://www.apache.org`.

[Bau03] Matthias Bauer. New Covert Channels in HTTP: Adding Unwitting Web Browsers to Anonymity Sets. In *Proceedings of the Workshop on Privacy in the Electronic Society (WPES 2003)*, Washington, DC, USA, Oktober 2003.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 4(2), Februar 1981.

[Dae96] Daemon9. Project Loki: ICMP tunneling. *Phrack*, 7(49), 1996. `http://www.phrack.org/show.php?p=49&a=6`.

[Dae97] Daemon9. LOKI2 (the implementation). *Phrack*, 7(51), 1997. `http://www.phrack.org/show.php?p=51&a=6`.

[DC03] Alex Dyatlov und Simon Castro. Exploitation of data streams authorized by a network access control system for arbitrary data transfers: tunneling and covert channels over the HTTP protocol. Juni 2003. `http://gray-world.net/projects/papers/html/covert_paper.html` (12.06.2004).

[Eck03] Claudia Eckert. *IT-Sicherheit*. Oldenbourg Wissenschaftsverlag, 2. Auflage, 2003.

[eth] Ethereal-Homepage. `http://www.ethereal.com/`.

[Eße05] Hans-Georg Eßer. *Ausnutzung verdeckter Kanäle am Beispiel eines Web-Servers*. Diplomarbeit, RWTH Aachen, Mai 2005. `http://privat.hgesser.com/docs/`.

[FBH+02] Nick Feamster, Magdalena Balazinska, Greg Harfst, Hari Balakrishnan und David Karger. Infranet: Circumventing Web Censorship and Surveillance. In *11th USENIX Security Symposium*, San Francisco, CA, August 2002.

[HEM00] E.-G. Haffner, Thomas Engel und Christoph Meinel. Integration der Schleusentechnologie "Lock-Keeper" in moderne Sicherheitsarchitekturen. In M. Schumacher und R. Steinmetz, Hrsg., *Sicherheit in Netzen und Medienströmen. Tagungsband des GI-Workshops "Sicherheit in Mediendaten"*, Informatik Aktuell, Seiten 17–26, 2000.

[Htt] Httptunnel-Homepage. `http://www.nocrew.org/software/httptunnel.html` (29.12.2004).

[KEB98] Dogan Kesdogan, Jan Egner und Roland Büschkes. Stop-and-Go MIXes Providing Probabilistic Security In An Open System. In David Aucsmith, Hrsg., *Information Hiding: Second International Workshop*, Jgg. 1525 of *Lecture Notes in Computer Science*, Seiten 83–98, Portland, Oregon, U.S.A., 1998. Springer-Verlag, Berlin, Germany.

[Kem83] Richard A. Kemmerer. Share resource matrix methodology: An approach to identifying storage and timing channels. *ACM Transactions on Computer Systems*, 1(3):256–277, August 1983.

[Lam73] Butler W. Lampson. A note on the confinement problem. *Commun. ACM*, 16(10):613–615, 1973.

[RF89] T. R. N. Rao und E. Fujiwara. *Error-control coding for computer systems*. Prentice-Hall, 1989.

[Rut04] Joanna Rutkowska. Passive covert channels in the Linux kernel. In *21st Chaos Communication Congress*, Berlin, 2004.