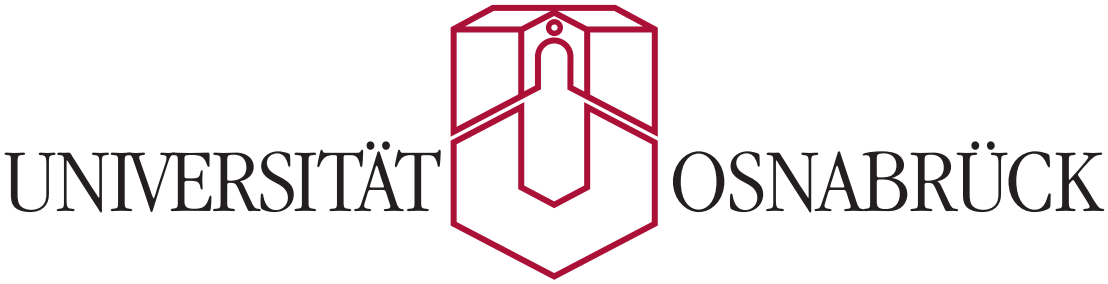


Institut für Informatik
Arbeitsgruppe Verteilte Systeme

IT - Sicherheit

ITS - Klausurersatzleistung

Henrik Gerdes
MatNr: 969272

Wintersemester

February 26, 2021

Contents

ITS - Klausurersatzleistung

Henrik Gerdes

February 26, 2021

1. Einleitung

Im Rahmen der Klausurersatzleistung für das Modul IT-Sicherheit wurde eine Proxy Client-Server Architektur erstellt, mit der TCP Verbindungen umgeleitet werden können, um z.B. die Filterregeln einer Firewall zu umgehen und um blockierte Anwendungen dennoch zu nutzen. Der strukturelle Aufbau ist in fig. 1 zu sehen.
Die Client-Server Architektur nutzt eine gemeinsame Tunnelklasse. Diese besteht aus einem *SocketServer.TCPServer*, der bereits die unterliegenden sockets und den nicht blockierenden Programmablauf regelt. Server und Client haben einen anwendungsspezifischen Handler der bei eintreffenden Anfragen aufgerufen wird. Dieser steuert die Kommunikation zwischen einander, sowie zu den Endpunkten. Es sind mehrere gleichzeitige Verbindungen möglich.

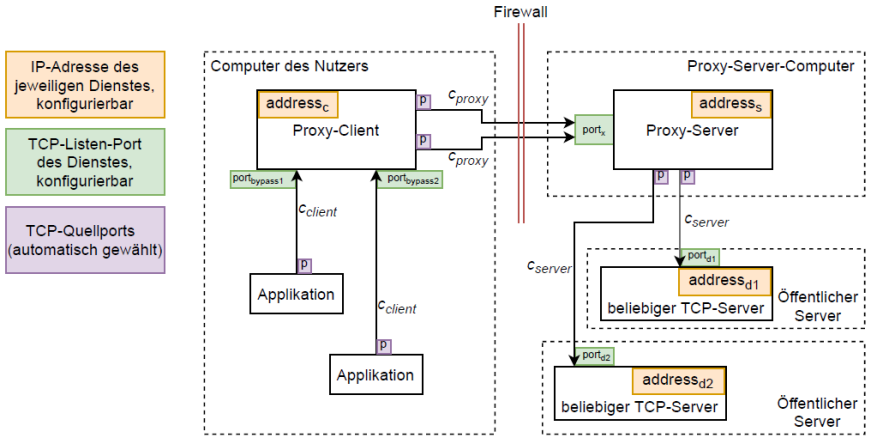


Figure 1: Netzwerk-Architektur zur Firewall-Umgehung

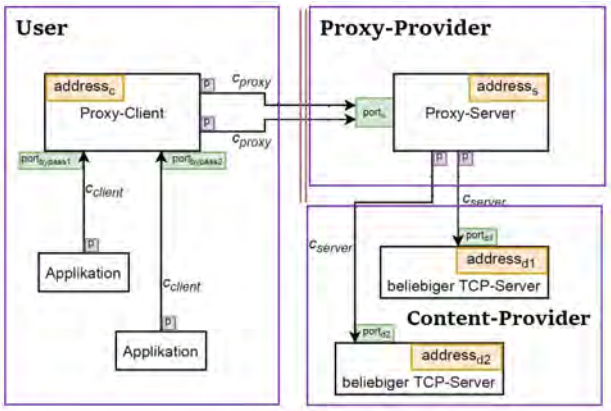
2. Security

Nach Abschluss der Aufgabe 1 besteht eine funktionstüchtige Client-Server-Proxy Architektur, die mehrere Anfragen gleichzeitig bearbeiten kann. Diese unterliegt jedoch bestimmten Einschränkung hinsichtlich sicherer Nutzung. Diese Einschränkungen werden nachfolgend bearbeitet.

2.1. Entitäten

Die Nutzung kann auf folgende Entitäten aufgeteilt werden:

- Anwender:
  - Anwendung
  - ProxyClient
- Proxy-Betreiber
  - ProxyServer
- Inhaltsanbieter
  - E-Mail
  - WebDienste
  - PrivateDienste
  - ...



(a) Entitäten der ProxyClient-Server Architektur

Jede Entität betreibt eine oder mehrere technische Entitäten. Diese stellen die Prozesse da, die jede Entität betreibt. Der nachfolgende Absatz beschäftigt sich mit der Beziehung zwischen Anwender und Proxy-Betreiber

2.2. Sicherheitsanfälligkeiten und Einschränkungen

Anwender & Betreiber: Der ProxyServer ist frei aus dem Internet erreichbar und nimmt von jedem Endgerät Datenpakete entgegen. Dabei kann der Server nicht verifizieren, dass der anfragende Dienst tatsächlich der von dem Server erwartete ProxyClient ist. Das gleiche gilt für die Authentizität des Servers gegenüber des Clients. Beides verstößt gegen den Grundsatz der **Authentication**.
Die Kommunikation zwischen ProxyClient und ProxyServer läuft über fremde Kommunikationswege. Jeder der Zugriff auf einen Teil dieses Kommunikationsweges hat, kann die übertragenen Informationen mitschneiden und möglicherweise gegen die Kommunikationspartner verwenden. Die Möglichkeit zum Belauschen und Lesen der Kommunikation verstößt gegen die **Confidentiality**.
Neben dem Mitlesen können die übertragenden Informationen auch verändert werden, was wiederum gegen die **Integrity** verstößt.

Betreiber & Inhaltsanbieter: Der ProxyServer Anbieter ermöglicht es fremden Entitäten seine Dienste zu nutzen, sowie durch diese fremden Entitäten Anfragen an Dritte zu richten. Dabei trägt der Server Anbieter die Verantwortung für Angriffe oder andere illegale Handlungen, die über seine Infrastruktur gegenüber Dritte getätigt werden.
Auch die eigene Infrastruktur kann Ziel eines Angriffs werden, indem z.B. Angreifer so viele Anfragen an der Proxy stellen, dass dieser zusammenbricht und nicht mehr durch andere genutzt werden kann, wie bei einem klassischen Denial of Service (DoS) angriff.

Anwender & Inhaltsanbieter: Wie oben in 2.2 bereits erwähnt, ist das Prinzip der Vertraulichkeit bei der aktuellen Konfiguration zwischen keinen der Entitäten gegeben. Technisch ist dies dadurch bedingt, dass der Proxy zu diesem Zeitpunkt keine sockets unterstützt, die das Transport Layer Security (TLS)-Protokoll für TCP Verbindungen nutzen. Aufgrund dessen schlägt die Authentifizierung, sowie der Schlüsselaustausch für nachfolgende Verschlüsselung mit dem WebServers gegenüber des Proxys und somit auch der Anwendung (Browser) fehl. Anwendungen sind somit auf nicht verschlüsselte Dienste, wie HTTP beschränkt.

2.3. Lösungsansetze

Vertraulichkeit, Integrität und Authentizität Damit Anwender sich sicher sein können, dass sie sich mit einem vertrauenswürdigen ProxyServer verbinden, kann man eine Server Authentifizierung einführen, bei der sich der Server mit einem Zertifikat ausweist und der Client dieses Zertifikat Zertifikat überprüft. Dies löst das Problem der **Authentication** des Servers gegenüber des Clients. Gleiches kann der Client gegenüber dem Servers machen. Organisatorisch wäre dafür notwendig, dass der Server alle CA-Zertifikate besitzt, mit denen die Clients signiert wurden, was in der realen Welt eine unvorstellbare logistische Herausforderung bedeuten würde.
Technisch gesehen würden diese Zertifikate während des TLS-Handshakes ausgetauscht werden. Ebenfalls möglich ist es mittels TLS Nachrichten zu verschlüsseln, um dessen **Confidentiality** sicherzustellen. Die genauen Verschlüsselungsverfahren hängen von der genutzten Version und Erweiterungen von TLS ab.
Um die **Integrity** einer Nachricht sicherzustellen wird über die Nachricht ein Hash berechnet, anschließend ebenfalls verschlüsselt und als Signatur angehängt. Der Empfänger kann den selbst berechneten Hash mit dem entschlüsseltem Hash aus der Signatur vergleichen, um dessen Integrität zu überprüfen.
Die technische Umsetzung ist mit dem Secure Sockets Layer (SSL)-Modul möglich, welches das TLS-Protokoll vollständig implementiert.

Verfügbarkeit und Missbrauchsschutz Neben der Sicherheit der Kommunikation ist auch die Verfügbarkeit, sowie die Nutzung des Proxys für ausschließlich legale Zwecke sicherzustellen. Dies kann eben einer Eingrenzung der Nutzer durch Filter gewährleistet werden, die bestimme Dienste blockieren und Ressourcen wie Datennutzung, Anzahl der

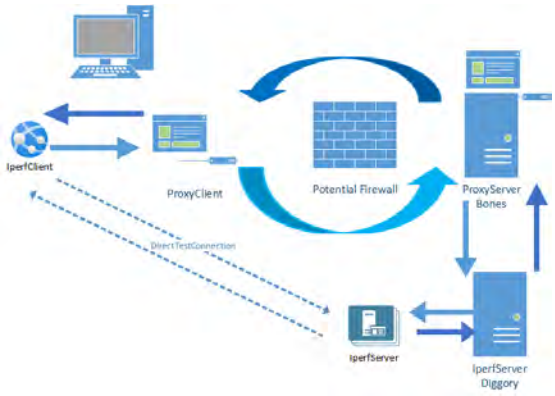


Figure 3: Bandbreiten Testaufbau

Anfragen und Rechenleistung limitieren. Loadbalancer und Backup-Systeme würden die Verfügbarkeit zusätzlich steigern.

3. Evaluation

Im Rahmen dieser Evaluation wird nur der Datendurchsatz des Proxys betrachtet, andere Metriken wie Delay und Paketverlust werden vernachlässigt.

3.1. Testaufbau

Für die Durchführung der Durchsatzmessungen wurde auf das Werkzeug `iperf` zurückgegriffen. Dieses erlaubt es mittels einer Client-Server Architektur den Datendurchsatz zwischen Endpunkten zu bestimmen.
Alle Rechner (Client(Vogon), TestServer1(diggory), TestServer2(bones)) werden mit Linux (Kali 2020.3 auf Vogon, Ubuntu 16.04 auf diggory/bones) betrieben und nutzen Version 2.0.14a von `iperf` auf dem Client und Version 2.0.5 auf dem Server. Abseits der standard Konfiguration wurde auf die einheitliche Testdauer von 10s verzichtet und stattdessen eine Datenmenge von 16MB für die Übertragung festgelegt. So kann z.B. einfach die Übertragungsdauer bestimmter WebSeiten abschätzen werden. Aufgrund des natürlichen Einschätzungsvermögen für Zeit sind die Ergebnisse so auch für Leser ohne technischem Hintergrund einfacher einzuschätzen. Zusätzlich wurde das Logging-interval auf 0.5s gesetzt um genauer Intervalle extrahieren zu können. Es wurde nur der Download vom Server gemessen, ein bidirektionaler Test war Aufgrund der nicht öffentlich zugänglichen IP des Clients nicht möglich.
Als Basisfall wurde ein direkte Messung ohne Proxy zu einem der System diggory durchgeführt. Um einem realistischem Einsatz zu simulieren, wurde anschließend der ProxyServer auf dem zweitem Testrechner (bones) gestartet, welcher dann eine Verbindung zur iperf Instanz auf diggory herstellt. So erhält man eine Topologie von drei Rechnern, dem Client, der ProxyServer und der Zielserver, wie sie auch in der Realität anzufinden wäre.

Der ProxyServer wurde dann in den Konfiguration SSHTunnel, KeinSSL, ServerAuthentification, ClientServerAthentification und ClientServerAthentificationACL getestet. Ein Testlauf besteht aus 15 Iterationen über diese Konfigurationen. Insgesamt wurden acht Testläufe zu unterschiedlichen Uhrzeiten, alle sechs Stunden, durchgeführt. Ein automatisiertes Script befindet sich im Anhang.

3.2. Analyse der Ergebnisse

Die generierten Log-Dateien wurden mit dem beiliegendem Script ausgelesen und erzeugen alle nachfolgenden Grafiken. Abbildung 4 zeigt die Messergebnisse für den Datendurchsatz aller verschiedener Proxy Konfigurationen. Entgegen der Erwartungen liefert die direkte Verbindung (ohne Proxy) den geringsten Durchsatz mit einer durchschnittlichen Geschwindigkeit von 35.45 Mbit/s. Um potentiell ungewollte Variationen auszuschließen wurde der Test mit dem `-N` Flag von `iperf` wiederholt, was die Kernel-Pufferung deaktiviert. Die Testergebnisse blieben jedoch unverändert.
Bei abgebrochenen Tests fiel auf, dass `iperf` sehr hohe Durchsätze lieferte. Das ist dadurch begründet, dass `iperf` annimmt das Daten im Sendefenster bereits am empfangen wurden obwohl der Server noch nichts empfangen wurde. Dieses Phänomen ist den Serverlogs entsprechend nicht während gültiger Tests aufgetreten. Durch das Verkleinern des Proxy Empfangspuffers konnte ein systematisch verringerter Datendurchsatz über alle Konfigurationen gemessen werden. Nach dieser Beobachtung und weiteren Nachforschungen mit Wireshark wurde ein Implementierungsfehler im Proxy ausgeschlossen. Die Tatsache, dass der SSHTunnel, der keinen eigenständig implementierten Code enthält ebenfalls performanter ist die direkte Proxyverbindung unterstützt diese Annahme. Zu vermuten ist, dass `iperf` bereits die Bestätigungen zu dem lokalen Proxy-Clients nutzt für die Durchsatzmessungen. Dieses verhalten bedarf jedoch weiterer Nachforschungen.
Alle anderen Proxy Konfiguration lieferten einen durchschnittlichen Durchsatz von 70.63 bis 75.92 Mbit/s. Mit steigenden Sicherheit Funktionen ließ der Durchsatz jeweils um ca. 2 Mbit/s nach. Dies ist durch den Zusätzlichen Datenverkehr von SSL zu begründen. Dennoch ist dies der doppelte Durchsatz ohne Proxy und ca. 20Mbis/s mehr als die Leistung des SSHTunnels. Eine weitere interessante Tatsache ist die Variation des Durchsatzes über alle Konfigurationen hinweg. Die eigene Proxyanwendung schwankt um ein vielfaches mehr als die direkte Verbindung und der SSHTunnel. Tabelle 1 zeigt mit dem Durchschnitt, der Standartabweichung und der Varianz die signifikantesten Werte für die Durchsatzmessung. Die Ergebnisse zeigen jedoch, dass der Durchsatz mehr als ausreichend für einfaches Web-Browsing ist. In nicht überwachten praktischen Tests ließen sich auch Videos ohne Probleme abspielen. Die nicht evaluierte zusätzliche Latenz durch den Proxy könnte jedoch Einfluss auf Echtzeit Anwendungen haben. Dies sollte in einem späteren Zeitpunkt untersucht werden.

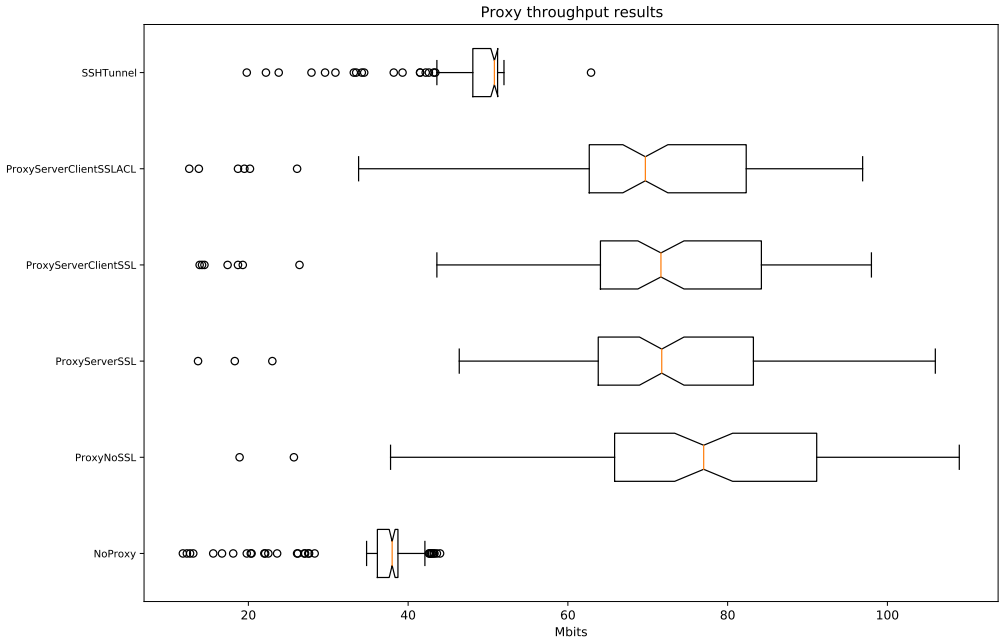


Figure 4: Throughput test results

	AVG	Std	Var
NoProxy	35.45	7.26	52.75
SSHTunnel	47.87	6.60	43.54
ProxyNoSSL	75.92	18.86	355.88
ProxyServerSSL	73.19	15.23	231.88
ProxyServerClientSSL	71.78	17.85	318.56
ProxyServerClientSSLACL	70.87	17.72	313.88

Table 1: `Iperf` Durchsatz Ergebnisse

4. SOCKS

Das SOCKS Protokoll ist ein weiteres design zur Umsetzung eines Proxy-Systems. Anders als die aktuelle Implementation braucht SOCKS keine separate Client Anwendung. Diese ist im allgemeinem bereits in der Anwendung die den Proxy nutzen will integriert. Dies verringerter den Konfigurationsaufwand und erlaubt es der Anwendung dynamisch den Zielserver zu ändern. Das bedeutet außerdem, dass die Anwendung den Namen des Zielservers kennt und mittels dessen Zertifikat den Names des Hosts überprüfen kann. Damit ist SOCKS auch in der Lage `https` Verbindungen aufzubauen.

5. Fazit

Die erstellte Proxy-Architektur erfüllt seinen Zweck einen durch die Firewall blockierten Dienst dennoch zu benutzen. Mit der Nutzung von SSL laufen Nutzer auch nicht Gefahr all ihre Informationen unverschlüsselt über fremde Kommunikationswege zu übertragenen. Clientseitige Authentifizierung erfordert erheblichen zusätzlichen Aufwand und sollte für Endnutzer nur in eingebetteten Systemen genutzt werden um fehlerhafte Bedienung zu vermeiden. Die Serverseitige Authentifizierung ist jedoch dringend zu empfehlen und deutlich einfacher zu konfigurieren.
Das Erstellen eines Konfigurationseintrags für jeden Zielserver beeinträchtigt zusätzlich die Nutzbarkeit des Proxys und zeigt erneut welche Vorteile SOCKS bietet. Web-browsing und Anwendungen mit moderaten Bandbreitenanforderungen sind ohne Einschränkungen nutzbar. Echtzeitanwendungen wie Voice over IP (VoIP) oder Spiele könnten durch die zusätzliche Latenz und inkonsistentere Verbindung negativ beeinflusst werden.
Je nach Anwendungsscenario stellt der Proxy jedoch eine akzeptable Lösung für das Nutzen von Diensten da, die anderseits nicht nutzbar wären.

Appendix

A. Abbreviations

DoS	Denial of Service
TLS	Transport Layer Security
HTTP	Hypertext Transfer Protocol
ACL	Access Control List
SSL	Secure Sockets Layer
VoIP	Voice over IP

B. Iperf test script

```
#!/bin/bash
trap "kill 0" EXIT

ITERATIONS=15
PACKAGE_SIZE=16MB
SERVER="diggory"
LOG_FILE="iperf_${SERVER}_new_1.log"

# Starting clients in background & example iperf call"
python3 ../proxy_client.py -f ../conf/config_iperf_bones.txt -t &
python3 ../ssh_proxy_client.py -f ../conf/config_ssh.txt &
# iperf -s -i 0.5 -N -p 2622

for iteration in $(seq 1 $ITERATIONS); do
    echo "NoProxy" >> $LOG_FILE
    iperf -c ${SERVER}.informatik.uni-osnabrueck.de -i 0.5 -p 2622 -n
        $PACKAGE_SIZE -N >> $LOG_FILE

    echo "ProxyNoSSL" >> $LOG_FILE
    iperf -c 127.0.0.1 -i 0.5 -p 8000 -n $PACKAGE_SIZE -N >> $LOG_FILE

    echo "ProxyServerSSL" >> $LOG_FILE
    iperf -c 127.0.0.1 -i 0.5 -p 8001 -n $PACKAGE_SIZE -N >> $LOG_FILE

    echo "ProxyServerClientSSL" >> $LOG_FILE
    iperf -c 127.0.0.1 -i 0.5 -p 8002 -n $PACKAGE_SIZE -N >> $LOG_FILE

    echo "ProxyServerClientSSLACL" >> $LOG_FILE
    iperf -c 127.0.0.1 -i 0.5 -p 8003 -n $PACKAGE_SIZE -N >> $LOG_FILE

    echo "SSHTunnel" >> $LOG_FILE
    iperf -c 127.0.0.1 -i 0.5 -p 2222 -n $PACKAGE_SIZE -N >> $LOG_FILE
done
```

Listing 1: Iperf test script

List of Figures

Declaration of Authorship

I hereby declare that the paper submitted is my own unaided work. I assure that I wrote this paper without using any other means and sources than those specified. As well as the sources used literally or analogously taken from the sources identified as such.

Signature (Henrik Gerdes)

Osnabrück, the February 26, 2021