UNIVERSITÄT OSNABRÜCK

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. Introduction

Part of the IT security lecture was to create a proxy client-server system that redirects TCP connections to bypass firewalls rules and allows to hide personal informational of the clients. The structure is displayed in fig. 1.
Server and client both use a common tunneling class. It is based of a *SocketServer.TCPServer*, which already handles the underlaying socket connection and does not block. Each base-tunnel instance can have a custom handler that implements the required logic and gets called for every request. These handlers handle the initial handshake between client and server and on success switches to redirect mode. Both, server and client can handle multiple requests at the same time.

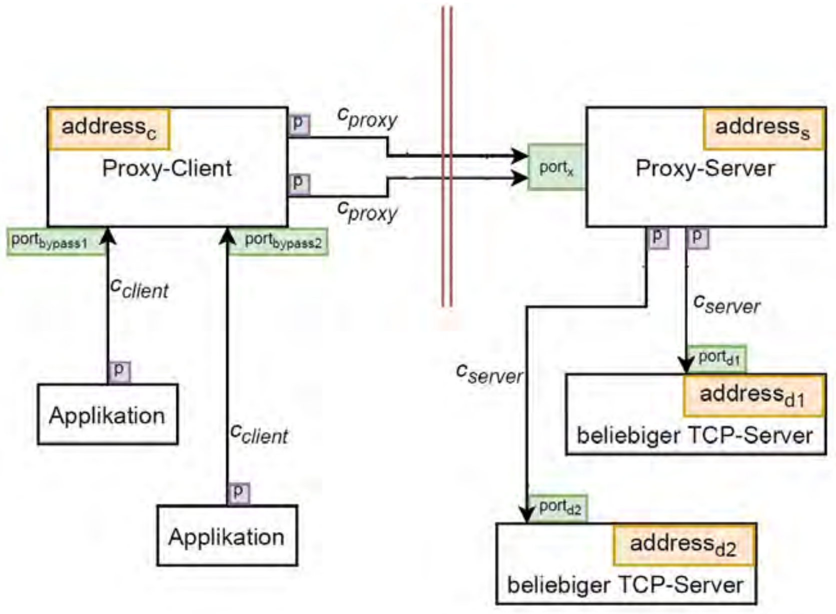


Figure 1: Netzwerk-Architektur to bypass a firewall

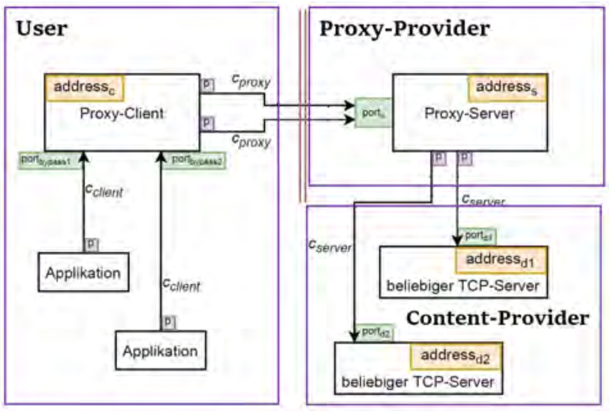
2. Security

At the end of task 1 the proxy system parses a configuration file, accepts multiple requests at a time and in general is fully functional. Nerveless it still lacks some basic requirements in terms of security and availability. The following section discusses these and all involved entities.

2.1. Entities

The following entities are involved by using the proxy:

- User:
  - Application
  - ProxyClient
- Proxy-Provider
  - ProxyServer
- Content-Providers
  - E-Mail
  - WebServices
  - PrivateServices
  - ...



(a) Entities of the proxy client-server setup

Each entity runs one or more technical entities. Technical entities are the processes that an entity operates on.
The following section discusses the relation between these entities as well as the security risks that exists in-between them.

2.2. Security risks and restrictions

User & ProxyProvider: Any internet connected device can perform requests to the proxy. The server can not be sure that the client that connects is indeed a valid client instead of a malicious client that spoofs its identity. The same problem persists on the client side. Attackers may fake the proxy server and convince users to connect to their server to perform further malicious actions. There is no way the entities can be sure about each others identity and thereby it violates the principle of **Authentication**.
Some parts of the communication infrastructure are owned by third party entities. Every one that has access to a part of the communication infrastructure can capture all confidential information on that infrastructure and use it against the recipient. This violates the principle of **Confidentiality**.

Beside capturing information it is also possible to alter the information which violates the principle of **Integrity**.

ProxyProvider & Content-Providers: The proxy server allows strangers to access his services and also redirects any request to other services. So that the server provideer may be responsible for attacks on third-party services that are caused by abusive usage of his server.
The proxy infrastructure itself can be a potential target. Too many request or bad requests (like a Denial of Service (DoS) attack) could crash the proxy server and make it unavailable for all users.

User & Content-Providers: As already described in 2.2, in the current configuration there is no confidentiality between any of the entities. The technical cause for that is that the currently used sockets don't support the Transport Layer Security (TLS) protocol. Even if the application or the destination support TLS it won't work because the authentication and key exchange between proxy and endpoint fails. With this restriction the user is limited to unencrypted services such as HTTP.

2.3. Possible solutions

Confidentiality, Integrity and Authentication To ensure that the user connects to a valid proxy server one could introduce certificates for authentication. The server identifies itself with a certificate that is verifiable by the client. This solves the previously nonexistent **Authentication**. The same principle could be applied to also allow the authentication of the client to the server. For this the server must own all CA-certificate that signed the clients certificates or all clients must be distributed with pre-generated certificates.
The certificate exchange and technical authentication would happen within the TLS handshake. With these certificates and their keys it is also possible to encrypt messenges with TLS. This would ensure the **Confidentiality** of any message. The encryption algorithm depends on the version and configuration of TLS.
To ensure **Integrity**, it is possible to hash the message, encrypt the hash and append it to message as signature. The recipient can compare the hash he calculated with the decrypted hash of the sender. If both hashes match, it guarantees the integrity of the message.
The technical implementation of TLS is done in the Secure Sockets Layer (SSL) python module.

Availability and abuse protection In addition to the security of communication, the availability of the services must also be ensured. This is feasible with an access control to specific users or with resource limits for requests and compute time. Loadbalancers and backup systems could improve availability even further.
These actions and an additional blacklist for specific services could also help to protect the server against abusive or illegal usage.

Limitations This approach allows to secure the communication between proxy client and proxy server. It does not provide end-to-end encrypted usage between application and destination service. Even if the application has the destination certificate, the application does not know the host of the destination (only the client proxy knows) and therefore can not validate its certificate.

3. Evaluation

In the context of this evaluation, only the throughput of the proxy is considered, other metrics such as delay and packet loss are neglected.

3.1. Testsetup

The throughput measurements where made with the `iperf` tool. All nodes (client(Vogon), TestServer1(diggory), TestServer2(bones)) are running Linux (Kali 2020.3 on Vogon, Ubuntu 16.04 on diggory/bones) and version 2.0.14a of `iperf` on the client and version 2.0.5 on the server. The `iperf` server and client is running mostly in standard configuration except for more verbose logging (0.5s interval) and a fixed amount of data to transmit instead of a fixed test duration of 10 seconds. An exemplary call for client and server can be found in listing 1. Results with the fixed data setting makes it easier to approximate download times for in use cases and is also easier interpretable for non technical readers.
The test only covers the download throughput. Because the client is behind a NAT and has now way to expose its IP it was not possible to perform a bidirektional test.
The baseline throughput test is a direct connection between the `iperf` client (Vogon) to diggory without any proxy. Now diggory runs the proxy server and Vogon starts the proxy client. Iperf now connects to the proxy client which redirects the connection to bones and for there to the destination `iperf` server on diggory. This setup with these three nodes represent a realistic usage of the proxy in production. A systematic structure of this setup is shown in fig.3.

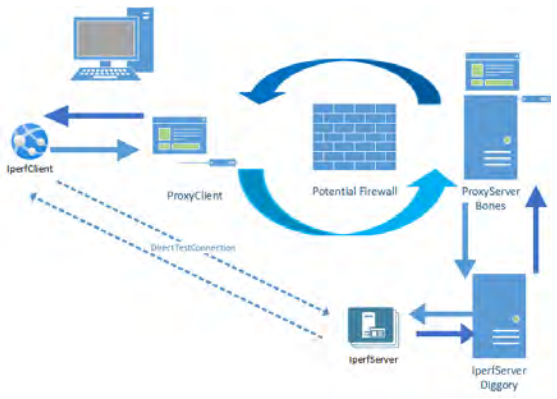


Figure 3: Throughput test setup

The proxy was tested with a SSHProxy, noSSL, server-authentication, client-server-authentication and client-server-authentication configuration with Access Control List (ACL). Each test runs 15 iterations of every configuration. In total the test was performed eight times each with 6 hours interval. The script for the exact test can be found with the resources of this paper.

3.2. Analysis of the results

Each generated log was parsed with a custom script wich also generated the following grafics.
Figure 4 shows the results of the throughput test performed with `iperf`. Contrary to expectations, the direct connection (vogon to diggory) without a proxy, delivers the worst throughput with an average of 35.45 Mbit/s. To eliminate as many variations as possible the test was also performed with the `-N` flag for `iperf` to disable kernel package buffering. This did not change the results.
During the tests, if the test were canceled, it was found that the data throughput was very high (Gbit/s). This happend when `iperf` assumes that the packages where already acknowledged when in fact the server did not receive anything. This did not happen during valid tests, as one could see from the monitored server logs. Lowering the receive buffer size in the proxy resulted in fact in a decreased throughput, after this change and additional checks with Wireshark, the possibility of an implementation error was rejected. The results form the SSHTunnel, that did not run self implemented code, support this assumption. It is presumed that `iperf` could already use the acknowledged of the proxy client to calculate the throughput and thus does not represent the real speed. After all this behavior needs further investigation to allow for final statements.
All other proxy configurations provide an average throughput between 70.63 and 75.92 Mbit/s. Noticeable is a slight deterioration in SSL configurations due to additional SSL overhead. This is still 2x the throughput without any proxy at all. The variation in speeds, per configuration also provides some findings. The direct test without any proxy and the test with the SSHTunnel provide the most consistent throughput. Table 1 shows the most significate results for the evaluation with expected value and standard deviation and variance.
This throughput evaluation shows that the proxy throughput is more than enough for simple web-browsing. Unsupervised practical tests also show usability for video streaming. The additional latency caused by the proxy, which could influence the usability of real-time applications, was not measured in this test. The additional latency caused by the proxy, which could influence the usability of real-time applications, was not measured in this test. This should be investigated at a later date.

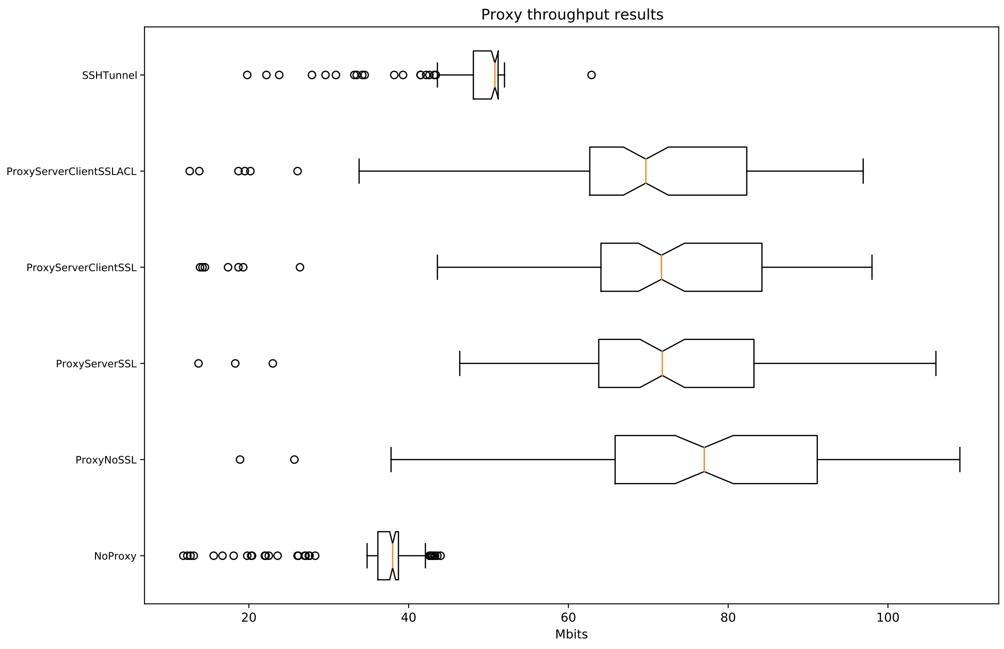


Figure 4: Throughput test results

	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` throughput results

4. SOCKS

The SOCKS protocol is another design of a proxy system. Unlike the current implementation, SOCKS does not need a dedicated client application. The SOCKS client is in general integrated within the application that uses the proxy. This reduces setup complexity. Another advantage is that the client can dynamically change the destination host here. This means that the application also knows the host address and can check the server certificate for the desired host name. This allows the usage of https connections via SOCKS.

5. Conclusion

The implemented proxy setup allows it to bypass a firewall and thereby fulfills its purpose. With the use of SSL, it can also be used practically without running the risk of disseminating all of its information unprotected in the network. ClintSide authentication introduces a lot of additional configuration work and should be only be used in embedded applications. ServerSide authentication is recommended and is much easier to setup.
Submitting a config file for one specific goal also makes usability more difficult. This shows the advantages that the SOCKS protocol provides. Web-browsing and applications that require a moderate bandwidth are, however, possible without any proxy related problems. Realtime application like Voice over IP (VoIP) or gaming may suffer from the additional latency and the less consistent connection.
Depending on use case this proxy can provide an acceptable solution accessing resources that would have been inaccessible otherwise.

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

```bash
#!/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