IT- und Netzwerksicherheit
WiSe 2020/21

Abschlussaufgabe: Hausarbeit

Nils Aschenbruck

Leonhard Brüggemann, Alexander Tessmer, Stefanie Thieme

Till Zimmermann

Ausgabe 14.01.2021
Abgabe 28.02.2021

Allgemeine Informationen zur Hausarbeit:

- Lesen Sie sich das Aufgabenblatt **aufmerksam von vorne bis hinten** durch, bevor Sie mit der Bearbeitung beginnen!
- Diese Hausarbeit ist die **Prüfungsleistung** für das Modul. Sie müssen diese **alleine** anfertigen.
- **(Code-)Plagiate werden nicht toleriert!** Gleiches gilt, falls Code von anderen Quellen ohne Referenz oder ungewöhnlich viel Code (auch mit Referenz) übernommen wird. Dies betrifft alle beteiligten Teilnehmer.
  - Machen Sie sich bewusst, dass eine Verletzung der hier aufgeführten Regeln Konsequenzen über das Nicht-Bestehen dieser Veranstaltung hinaus haben wird.
  - Weil es sich bei dieser Aufgabe um die Prüfungsleistung handelt, müssen Sie eine Erklärung zur selbstständigen Abfassung dieser Hausarbeit abgeben.
  - Unterschreiben Sie die Erklärung (letzte Seite) handschriftlich und reichen Sie einen Scan oder ein gut lesbares Foto mit dem Dateinamen **Erklaerung.pdf**, **Erklaerung.png** oder **Erklaerung.jpg** davon ein.
  - Ohne diese Erklärung wird die Hausarbeit nicht korrigiert und gilt damit als nicht bestanden (5.0).
- Die Abgabe der Aufgabe muss am **28.02.2021 bis spätestens 23:59 Uhr** via **StudIP** in der extra dafür angelegten Veranstaltung **Prüfung IT- und Netzwerksicherheit** erfolgt sein. Ihre Ausarbeitung sowie die zugehörigen Dateien sind in ein ZIP Archiv zu packen und im Aufgabenbereich der Veranstaltung hochzuladen. Der Name der Datei muss nach folgendem Schema gewählt sein:

  ITS_WiSe_202021_Hausarbeit_*MatrNo*

  wobei ***MatrNo*** durch die jeweilige Matrikelnummer ersetzt werden muss. Beim Entpacken des Archivs muss automatisch ein **Unterordner nach dem gleichen Namensschema** erzeugt werden (siehe S. 12).
- **Befolgen Sie alle aufgeführten Konventionen** für Dateinamen, Ordnerstruktur, Programmparameter, etc. Ansonsten verlängern Sie unnötig die Korrekturdauer und verzögern somit die Ergebnisverkündung.
- Im Gegensatz zu den praktischen Aufgaben wird es keine Tutoren-Fragestunde geben. Stattdessen werden zwei Besprechungen an den Übungsterminen stattfinden (21.01.21, 14:00 und 04.02.21, 14:00). Hier können aufgekommene Fragen für alle beantwortet werden. Bitte besuchen Sie diese oder lassen Sie sich von Kommiliton*innen vertreten, da hier u.U. wichtige Fragen beantwortet werden. Senden Sie umfangreichere Fragen bitte im Voraus an den Tutor (**tzimmermann@uos.de**).
- Bei Fragen von allgemeinem Interesse nutzen Sie bitte die **Mailingliste** zur Vorlesung. Bitte beachten Sie, dass diese ab Ausgabe der Hausarbeit moderiert wird und sich die Mailzustellung daher verzögert.

Vorwort

In öffentlichen Netzwerken (Hotel, Zug, Internet-Cafe) werden oft Firewalls eingesetzt, um sicherzustellen, dass User ausschließlich erwünschte Anwendungen wie etwa Web-Browsing nutzen können. Auch in den Netzwerken bestimmter Internet Service Provider wird die Nutzbarkeit verschiedener Layer 7-Protokolle eingeschränkt, etwa um den Abruf lokal illegaler Inhalte zu unterbinden [1]. Um trotzdem beliebige TCP-Dienste nutzen zu können, kann mithilfe eines TCP-basierten VPN-Protokolls oder eines Proxys beliebiger Traffic die von der Firewall akzeptierte Form annehmen. Ihre **1. Aufgabe** im Rahmen dieser Hausarbeit ist es, solch eine Proxy-Software, bestehend aus Client und Server zu implementieren. Somit kann mithilfe Ihres Proxys beispielsweise eine SSH-Verbindung über den meist erlaubten HTTP-Port 80 aufgebaut werden. Um dabei die IT-Sicherheits-Grundregeln zu beachten, soll Ihre Software die Nutzung des TLS-Protokolls erlauben **(2. Aufgabe)**, um Server und Nutzer zu authentifizieren sowie den Datenverkehr zu verschlüsseln. Im Rahmen der **3. Aufgabe** sollen Sie die Performance Ihrer Software evaluieren, um eine Einschätzung der praktischen Nutzbarkeit zu erhalten.

Neben Ihren Programmen erstellen Sie zu den Aufgaben 2 und 3 eine schriftliche Ausarbeitung mit dem Dateinamen **ITS_WiSe_202021_Hausarbeit_*MatrNo*.pdf** (MatrNo ersetzt durch Ihre Matrikelnummer). Das Layout sollte dem dieser Aufgabenstellung entsprechen, insbesondere die Schrift (Times oder Computer Modern, 12pt., 1-facher Zeilenabstand). Wir empfehlen die Verwendung von LaTeX [2] mit der Vorlage `\documentclass [12pt, a4paper]{scrartcl}` zum Satz der Ausarbeitung.

Ihre Programme müssen in der in PA0 beschriebenen Kali-Linux-Umgebung lauffähig sein.

1 Proxy zur Firewall-Umgehung

In Abbildung 1 ist eine Architektur für einen TCP-Proxy-Server aufgezeigt. Statt einer normalen Verbindung einer Client-Applikation (*Applikation*: bspw. Web-Browser, FTP-Client o.Ä.) zu *beliebigen TCP-Servern* kommen ein Proxy-Server und -Client zum Einsatz. Ihre Aufgabe ist es im Folgenden, diese Software zu entwickeln. Zur Vereinfachung der Bearbeitung beschränken Sie sich auf IPv4, schalten Sie also in Ihrer virtuellen Maschine die IPv6-Unterstützung vollständig ab[1]. Der Proxy-Client akzeptiert eine lokale TCP-Verbindung (c_{client}) von der Client-Applikation unter $address_{bypass} : port_{bypass_i}$. Daraufhin wird eine Verbindung (c_{proxy}) zum Proxy-Server unter $address_s : port_x$ aufgebaut. Von diesem wird wiederum zum gewünschten Ziel-TCP-Server unter $address_{d_i} : port_{d_i}$ eine

[1] `sysctl -w net.ipv6.conf.all.disable_ipv6=1`
`sysctl -w net.ipv6.conf.default.disable_ipv6=1`

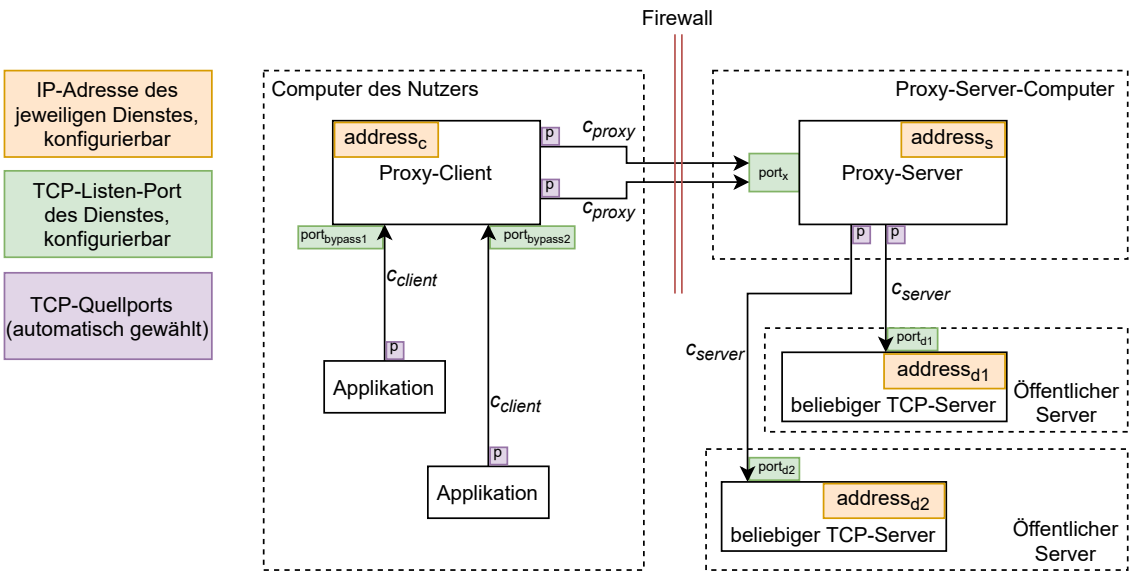

Abbildung 1: Netzwerk-Architektur zur Firewall-Umgehung

Verbindung (c_{server}) aufgebaut. Nach dem Verbindungsaufbau wechseln sowohl Proxy-Client als auch -Server in einen Weiterleitungsmodus, in dem die TCP-Payload beider aufgebauter Verbindungen jeweils an die entsprechende andere Verbindung weitergeleitet wird (Client: $c_{client} \rightleftarrows c_{proxy}$, Server: $c_{proxy} \rightleftarrows c_{server}$).

1.1 Grundgerüst

Um generisch einsetzbar zu sein, soll der Client eine Liste der zur Verfügung gestellten Proxy-Verbindungen über eine Konfigurationsdatei erhalten. Die Konfigurationsdatei ist mit dem Kommandozeilen-Argument `-f FILE` anzugeben. Verwenden Sie hierfür z.B. das Python-Modul *argparse*[2]. Implementieren Sie das Gerüst eines Proxy-Clients, der alle Einträge (respektive Zeilen) der Konfigurationsdatei einliest und jeweils eine Client-Instanz startet[3]. Eine Zeile der Konfigurationsdatei hat dabei das Format:

```
address_d_i:port_d_i:address_s:port_x:port_bypass_i
```

Implementieren Sie weiterhin eine Kommentar-Funktion, d.h. Ihr Programm sollte Zeilen ignorieren, deren erstes Zeichen eine Raute (#) ist.

[2] `https://docs.python.org/3/library/argparse.html`
[3] Es bietet sich an, hier mit Threads oder Prozessen zu arbeiten.

1.2 Basis-Funktionalität mittels SSH

Das SSH-Protokoll unterstützt das Tunneln einzelner TCP-Verbindungen [3]. Nutzen Sie dieses, um die gewünschte Funktionalität des späteren Proxys zu testen, sowie die Funktionsfähigkeit Ihres Grundgerüstes zu überprüfen. Zur Vorbereitung stellen Sie sicher, dass ein Public-Key-basierter SSH-Zugriff auf einen der öffentlichen Test-Server (`bones.informatik.uni-osnabrueck.de`; `diggory.informatik.uni-osnabrueck.de`) funktioniert. Eine Anleitung hierfür finden Sie beispielsweise im Debian-Wiki[4,5].

Implementieren Sie auf Basis Ihres Grundgerüstes zuerst einen Client (*ssh_proxy_client.py*), der für jeden Eintrag der Konfigurationsdatei (also in o.g. Client-Instanzen) einen SSH-Client mit den entsprechenden Optionen startet. Ein Start des SSH-Tunnels hat dabei die Syntax `ssh -L port_bypass_i:address_d_i:port_d_i user@address_s`. Hierbei ist $address_s$ einer der von uns zur Verfügung gestellten Testserver und *user* Ihr RZ-Benutzername. Mit dieser Syntax ist $address_{bypass}$ implizit auf *localhost* festgelegt. Implementieren Sie das Kommandozeilenargument `-u` zur Angabe des verwendeten *user*s. Prüfen Sie nun, ob Sie entsprechende getunnelte Verbindungen aufbauen können. Hierzu bietet sich beispielsweise ein Web-Browser an, wobei Sie jedoch anstatt der gewünschten Ziel-Webseite `http://localhost:port_bypass_i` eintragen. Hinterlegen Sie hierfür die Ziel-Webseite sowie den HTTP-Port 80 in der Konfigurationsdatei als $address_{d_i}$ und $port_{d_i}$. Beachten Sie, dass nur HTTP-Verbindungen, jedoch keine TLS/HTTPS-Verbindungen möglich sind. Um zu prüfen, ob Ihre Verbindung tatsächlich getunnelt ist, prüfen Sie beispielsweise Ihre öffentliche IP-Adresse, die bei aktivem Proxy die des genutzten Test-Servers (bones: 131.173.33.211, diggory: 131.173.33.209) sein sollte. Per HTTP (Port 80) können Sie hierfür den Dienst der Seite `http://icanhazip.com` verwenden (und entsprechend in Ihrer Proxy-Konfiguration hinterlegen).

1.3 Proxy-Server

Entwickeln Sie nun einen Proxy-Server (*proxy_server.py*), welcher die TCP-Tunnel-Funktion von SSH ersetzt. Dazu sollte Ihr Server Verbindungen von Ihrem später entwickelten Proxy-Client annehmen und auf eine Angabe des Ziel-Servers warten. Nach dem Verbindungsaufbau zum angegebenen Ziel-Server wechselt der Server in einen Weiterleitungsmodus. Der Server sollte die Kommandozeilenargumente `-l` für die zu bindende Adresse/den Hostnamen sowie `-p` für den Listen-Port ($port_x$) verwenden. Entwickeln Sie ein einfaches Signalisierungs-Protokoll für die beschriebene Angabe des Ziel-Servers, welches der Client nutzt um Adresse und Port des Ziel-Servers ($address_{d_i} : port_{d_i}$) an den Proxy-Server zu senden. Hierzu kann es sinnvoll sein, das Python-Modul *struct*[6] zu

[4] `https://wiki.debian.org/SSH#Using_shared_keys`
[5] verwenden Sie keine Passphrase und verwenden Sie die standard-identity.
[6] `https://docs.python.org/3/library/struct.html`

verwenden. Alternativ können Sie auch einen String übermitteln, der mit entsprechenden Begrenzungszeichen zusammengesetzt ist[7]. Zu diesem Ziel-Server soll daraufhin eine Verbindung aufgebaut werden. Ist diese erfolgreich, werden die Inhalte dieses Verbindungspaars an die jeweils andere Verbindung weitergeleitet. Ist dies nicht erfolgreich, wird die Verbindung zum Proxy-Client geschlossen. Auch wenn im weiteren Verlauf eine Verbindung geschlossen wird, sollte die jeweils andere geschlossen werden. Achten Sie darauf, dass sich mehrere Clients verbinden können[8]. Stellen Sie weiterhin sicher, dass auch in der ersten Nachricht keine Inhalte verloren gehen, die direkt nach der Signalisierungs-Nachricht gesendet werden.

1.4 Proxy-Client

Entwickeln Sie nun den in 1.2 entwickelten Client mit dem neuen Namen *proxy_client.py* weiter, sodass nicht SSH-Clients gestartet werden, sondern TCP-Verbindungen auf (localhost : $port_{bypass_i}$) akzeptiert werden. Sobald ein lokaler TCP-Client (*Applikation*) sich hierzu verbindet, sollte eine Verbindung zu Ihrem Proxy-Server aufgebaut werden und entsprechend Ihres Protokolls der Ziel-Server ($address_{d_i}$: $port_{d_i}$) an den Proxy-Server übermittelt werden. Daraufhin wird für dieses Verbindungspaar ebenfalls der Weiterleitungsmodus aktiviert (analog zum Server). Auch hier sollte auf ein Schließen einer der beiden Verbindungen entsprechend durch Schließen der anderen reagiert werden. Prüfen Sie, ob Sie durch sinnvolle Modularisierung Code-Teile für Client und Server gleichzeitig verwenden können.

1.5 Proxy-Test

Stellen Sie sicher, dass Ihr Server und Client zusammen als Proxy für TCP-Verbindungen agieren. Für HTTP-Dienste testen Sie dies mindestens mit dem o.g. öffentlichen Dienst unter `icanhazip.com:80` sowie `sys.cs.uos.de:80`. Ihren Proxy-Server können Sie auf einem der o.g. Test-Server laufen lassen. Um Ihnen allen das gleichzeitige Testen zu ermöglichen, verwenden Sie ausschließlich die Ports 1..9 + [2., 4., & 6. Stelle Ihrer MatNr.] [9]. Achten Sie weiterhin darauf, Ihren Server stets nach dem Testen zu beenden.[10]

[7] Achten Sie darauf, dass Ihr gewähltes Begrenzungszeichen niemals in Hostnamen, IPs oder Ports vorkommen darf.

[8] Es sollte keine fest definierte Anzahl möglicher Clients geben, sondern dynamisch zusätzliche gestartet werden. Die konkrete Anzahl hängt damit von der Leistungsfähigkeit des Rechners ab.

[9] Studierende mit der MatNr 424711 könnten somit die Ports 1271, 2271, 3271, ..., 9271 verwenden

[10] Aus Sicherheitsgründen werden auf den Test-Servern alle Prozesse, die TCP-Verbindungen akzeptieren, nach 10 Minuten beendet.

2 Reale Sicherheitsprobleme

Diskutieren Sie als Teil Ihrer Ausarbeitung, inwiefern der bisher entwickelte Proxy grundlegene Fehler in der Sicherheitsarchitektur aufweist. Gehen Sie dabei auf folgende Punkte explizit ein:

- Erarbeiten Sie die involvierten Entitäten sowohl technischer, organisatorischer sowie juristischer Art.
- Überlegen Sie, durch welche Entitäten hierbei gegen welche Grundprinzipien der IT-Sicherheit verstoßen wird.
- Legen Sie für diese Fehler dar, wie diese abstrakt zu lösen wären (welche Mechanismen wären nötig, gegen welches Szenario richten sie sich konkret, was bieten sie jeweils nicht).
- Sie können zur Inspiration überlegen, welche Sicherheitsmechanismen wir bei der Nutzung der Testserver ergreifen.
- Erklären Sie, warum es mit Ihrem Proxy nicht ohne Weiteres möglich ist, HTTPS-Webseiten aufzurufen. Legen Sie dabei den Fokus auf die Rollen und Mechanismen der verschiedenen Software-Teile (Browser, Webserver, HTTPS-Protokoll).

Der Umfang dieser Diskussion sollte ein bis zwei Seiten betragen. Gehen Sie in Ihrer Einleitung kurz auf die Programmieraspekte Ihrer Bearbeitung ein, sodass ein außenstehender Leser die Umgebung nachvollziehen kann. Sie sollten dabei jedoch keine Erläuterungen zu den Details Ihrer Implementierung verfassen, weswegen der Umfang dieser Einleitung nicht mehr als $\frac{1}{4}$ Ihrer Diskussion betragen sollte.

2.1 PKI

Um diese Sicherheitsprobleme zu lösen, bereiten Sie eine Public-Key-Infrastructure (PKI) vor. Alle Dateien für diese Aufgabe müssen in dem Ordner *pki/* liegen. Die PKI sowie alle Zertifikate, Schlüssel usw. müssen durch das Bash- oder Python-Skript *initialize_pki.py* bzw. *initialize_pki.sh* initialisiert werden.

Nutzen Sie das *openssl*-Kommandozeilenwerkzeug [4][11] um Schlüssel, Zertifikate und Signaturen zu erzeugen. Beim Erarbeiten der nötigen Befehle können Sie openssl manuell und interaktiv verwenden, das resultierende Skript muss jedoch ohne weitere Eingabe funktionieren. Ein Satz aller erzeugten Zertifikate, Schlüssel usw. muss in der Abgabe im Unterordner *certificates/* enthalten sein.

[11] Aufgrund der Komplexität von OpenSSL und der wenig praxisgerichteten Dokumentation kann es sinnvoll sein, erst durch Anleitungen o.Ä. die nötigen Befehle herauszufinden und im Anschluss das OpenSSL-Manual zum vollständigen Verständnis der Befehle hinzuzuziehen.

Im ersten Schritt erstellen Sie bitte einen Schlüssel[12] und ein Zertifikat für Ihre Certificate Authority (CA), wobei der Common Name *authority-MatrNo* lauten sollte (MatrNo entsprechend Ihrer Matrikelnummer). Alle anderen Angaben, wie Organisation oder Postadresse sind irrelevant und können daher leer gelassen werden. Erstellen Sie im Anschluss mindestens ein Zertifikat, welches von der CA signiert ist und dessen Common Name *example* lautet. Hierbei müssen Sie zunächst ein Certificate Signing Request erstellen. Erst durch dessen Signatur durch die CA entsteht das eigentliche Zertifikat. Erstellen Sie (mindestens!) folgende Dateien:

- `ca.key` Privater Schlüssel der CA
- `ca.pem` Zertifikat/Öffentlicher Schlüssel der CA
- `example.key` Privater Schlüssel des Beispiel-Schlüsselpaars
- `example.csr` Certificate Signing Request des Beispiel-Schlüsselpaars (unsignierter, öffentlicher Schlüssel)
- `example.pem` Zertifikat/Öffentlicher Schlüssel des Beispiel-Schlüsselpaars inkl. Signatur der CA.

Kommentieren Sie die einzelnen Schritte Ihres Skriptes und gehen Sie hierbei auf alle relevanten Kommandozeilenargumente ein.

2.2 Serverseitiges TLS

Erweitern Sie nun Ihren Proxy um serverseitige TLS-Authentifizierung[13] und Verschlüsselung. Erstellen Sie dafür zuerst ein von der CA signiertes Zertifikat für Ihren Server. Da pro Zertifikat nur ein Common Name möglich ist, wird heutzutage meist nur noch das Feature der *Subject Alternative Names* verwendet. Hierbei ist es möglich, ein Zertifikat für mehrere IP-Adressen bzw. DNS-Namen zu nutzen. Machen Sie hiervon Gebrauch und übernehmen Sie alle sinnvollen Hostnames und IP-Adressen in Ihr Zertifikat, also insb. auch die der beiden Test-Server. Der Common Name hat damit nur noch informativen Charakter, setzen Sie diesen daher auf *server-MatrNo* (MatrNo ersetzt durch Ihre Matrikelnummer).

Erweitern Sie Ihren Proxy-Server um die Kommandozeilenoptionen `--certificate` und `--key`. Wenn diese angegeben sind, verwenden Sie das *ssl*-Python-Modul [5] um serverseitiges TLS zu aktivieren. Das Argument `--certificate` legt dabei das TLS-Zertifikat des Servers fest, das Argument `--key` den zugehörigen privaten Schlüssel. Der Proxy-Client soll die zusätzliche Option `--ca` erhalten, die das öffentliche Zertifikat

[12] Verzichten Sie im Rahmen dieser Aufgabe auf eine passwortbasierte Verschlüsselung aller Schlüssel.
[13] Der Begriff Authentifizierung wirkt hier unintuitiv, ist aber korrekt. Wie bei HTTPS authentifiziert sich der Server gegenüber dem Client.

der CA angibt. Mithilfe von *ssl* soll diese Datei als (einzige) erlaubte CA definiert werden. Achten Sie darauf, dass Server und Client bei Auslassen der entsprechenden Kommandozeilenargumente unverändert ohne TLS funktionieren. Geben Sie mindestens folgende Dateien zusätzlich ab:

- `server.key` Privater Schlüssel des Server-Zertifikats
- `server.csr` Certificate Signing Request des Server-Schlüsselpaars
- `server.pem` Zertifikat des Server-Schlüsselpaars inkl. Signatur der CA.

2.3 Clientseitiges TLS

Um sicherzustellen, dass nur berechtigte Nutzer Ihren Proxy verwenden können, erweitern Sie Ihren Proxy-Server und -Client nun um clientseitige TLS-Authentifizierung. Hierzu soll Ihr Server ein vom Client präsentiertes TLS-Zertifikat prüfen und erzwingen, dass dieses von Ihrer CA signiert ist. Der Server soll über die zusätzliche Kommandozeilenoption `--ca` den Pfad des CA-Zertifikats erhalten, der Client über die Optionen `--key` und `--certificate` das Client-Zertifikat und den Client-Schlüssel. Die zu erstellenden Zertifikate sind analog zu denen aus 2.1 aufgebaut, die Common Names werden jedoch als eindeutige Benutzernamen verwendet. Erstellen Sie mindestens Zertifikate für die „Benutzer“ *client1* und *client2*, d.h. geben Sie min. die zusätzlichen Dateien `client1.key`, `client1.csr`, `client1.pem`, `client2.key`, `client2.csr` und `client2.pem` ab. Wie bei den bisherigen Aufgabenteilen soll auch die Client-Authentifizierung optional sein, also bei nicht-Angabe der entsprechenden Optionen nicht verwendet werden. Da Ihr Server mit dieser Infrastruktur eine kryptographisch gesicherte Übermittlung eines Benutzernamens erhält, implementieren Sie eine einfache Benutzer-Datenbank in Form einer Access Control List-Datei. Diese Datei soll dem Server über Option `--acl` angegeben werden und einen Benutzernamen pro Zeile enthalten. Nur wenn ein Benutzername (respektive Common Name des Client-Zertifikats) in dieser Datei vorkommt, soll eine Verbindung möglich sein, andernfalls wird diese vom Server wieder geschlossen[14]. Wird keine ACL angegeben, akzeptiert der Server Verbindungen von allen Clients, die ein korrekt signiertes Zertifikat präsentieren.

2.4 Test des sicheren Proxys

Prüfen Sie nun erneut, dass der von Ihnen entwickelte Proxy vollständig funktioniert und die erwünschten Sicherheitsmechanismen bietet. Das Zusammenspiel aus Server und Client sollte folgende Varianten unterstützen:

- SSH-Tunnel-Proxy-Client ohne eigenen Server

[14] Tipp: Recherchieren Sie die Methode `getpeercert`.

- Selbst entwickelter Client und Server,
  - vollständig unverschlüsselt
  - Server-Zertifikat mit korrektem Hostname, keine Benutzer-Authentifizierung
  - Server- und Benutzerauthentifizierung ohne ACL
  - Server- und Benutzerauthentifizierung mit ACL.

Prüfen Sie insbesondere für jeden Fall, ob das Fehlen des entsprechenden Authentifizierungsmerkmals auch wirklich sicherstellt, dass die Verbindung abgelehnt wird. Verwenden Sie hierfür:

- Ein nicht signiertes Server-Zertifikat
- Ein nicht signiertes Client-Zertifikat
- Ein Server-Zertifikat, welches den zum Verbindungsaufbau verwendeten Hostnamen nicht enthält
- Ein Client-Zertifikat mit einem Common-Name, der nicht in der ACL enthalten ist.

In allen Fällen einer fehlerhaften Client-Authentifizierung sollte der Server trotz Ablehnen einer Verbindung weiterhin neue Verbindungen akzeptieren, jedoch stets eine Fehlermeldung ausgeben. Der Client kann sich in Fehlerfällen mit einer Fehlermeldung beenden. Varianten der falschen Benutzung (beispielsweise ein Client mit Authentifizierung bei einem Server ohne CA-Angabe) dürfen Sie ignorieren.

3 Evaluation

Führen Sie eine exemplarische Leistungsbewertung Ihrer Proxy-Software durch und stellen Sie die Ergebnisse im Umfang von ein bis zwei Seiten sinnvoll dar. Verwenden Sie für Ihre Ergebnisse eine geeignete Darstellungsform. Gehen Sie auf Ihre Evaluationsumgebung ein und interpretieren Sie die Ergebnisse. Geben Sie zusätzlich zur Ausarbeitung einen mittels tcpdump [6] erstellten Mitschnitt des Datenverkehrs Ihrer Evaluation ab und achten Sie dabei darauf, alle notwendigen Header, jedoch nicht die vollständigen Nutzdaten aufzuzeichnen. Sollten Sie graphische Darstellungen erstellen, empfehlen wir hierfür die Nutzung einer geeigneten Statistik-Software wie etwa Matplotlib [7], Gnuplot [8] oder R [9]. Wenn Sie die Evaluation mithilfe eines Skriptes durchführen, geben Sie dieses zusätzlich ab, damit Ihre Ergebnisse schneller nachzuvollziehen sind.

Nutzen Sie für alle im Rahmen der Evaluation anfallenden Dateien den Ordner *evaluation/*, ggf. mit einer passenden README-Datei.

3.1 Durchsatz

Neben den Sicherheitsvorteilen, die der erweiterte Proxy bietet, sind auch negative Auswirkungen auf die praktische Nutzbarkeit möglich. Um diese quantifizieren zu können, evaluieren Sie im Folgenden die Leistungsfähigkeit Ihres Proxys. Da der Proxy auf der Transportebene ansetzt, ist es weniger sinnvoll, klassische Link-Layer Metriken wie Round-Trip-Time oder Delay zu messen. Sinnvoller, gerade für Anwendungen mit größeren Datenmengen (Streaming, Dateiaustausch), ist jedoch der erzielbare Netto-Datendurchsatz (Goodput).
Messen Sie mithilfe von iPerf [10][15] den Durchsatz einer normalen TCP-Verbindung zu einem der öffentlichen Testserver (s.o.). Vergleichen Sie diese Ergebnisse mit der Benutzung Ihres Proxys in den verschiedenen Varianten. Erstellen Sie dazu eine entsprechende Proxy-Konfiguration für iperf. Achten Sie darauf, diese Vergleiche unter weitestgehend gleichen Rahmenbedingungen durchzuführen (gleicher Internetanschluss, ähnliche Uhrzeit, ähnliche Testserver-Auslastung, ähnliche Rechnerauslastung). Beachten Sie sowohl bei der Durchführung Ihrer Evaluation, als auch bei der Beschreibung der Ergebnisse Kap. 7 der Vorlesung.

Bonusaufgaben

Die folgenden Aufgaben sind optionale Bonusaufgaben. Sie können zum Ausgleich von Abzügen in den ersten Aufgaben dienen.

SOCKS-Kompatiblität

Wie Ihnen möglicherweise aufgefallen ist, unterscheidet sich Ihre Proxy-Software nicht fundamental von verbreiteter Proxy-Software, wie etwa SOCKS [11]. Fügen Sie eine Kommandozeilenoption (`--socks`) hinzu, bei deren Nutzung Ihr Server anstelle des selbst entworfenen Signalisierungsprotokolls das SOCKS Protokoll der Version 4a [12] verwendet. Wenn Sie dieses erfolgreich umgesetzt haben, sollte es möglich sein anstelle Ihres Clients normale Anwendersoftware zur Nutzung Ihres Servers zu konfigurieren. Auch wenn der TLS-Teil Ihres Servers in diesem Fall nicht genutzt werden kann, kann bspw. über Firefox der Proxy noch einfacher verwendet werden. In diesem Fall ist es auch möglich, HTTPS-Webseiten abzurufen. Erläutern Sie auf maximal einer zusätzlichen Seite, warum dies mit dieser Variante möglich ist.

[15] Verwenden Sie die Version 2 von iperf. Das entsprechende Paket heißt lediglich `iperf`.

Evaluation: Time to First Byte

Auch wenn Sie in 3.1 bereits einen ersten Eindruck über die Leistungsfähigkeit Ihres Proxys bekommen haben, lässt sich daraus nur eingeschränkt die wahrgenommene Qualität[16] beurteilen. Bei Anwendungen mit häufigen Nutzerinteraktionen ist die wahrgenommene Latenz, im Falle von Web-Browsing-Anwendungen also die Gesamtdauer vom Klick eines Links bis zum Seitenaufbau, erheblich aussagekräftiger. Hier bietet sich die Time to First Byte-Metrik an, die die Dauer vom Beginn des TCP-Verbindungsaufbaus im Client bis zum Eintreffen des ersten Nutzdaten-Bytes ebenda bezeichnet. Da es für diese Metrik keine verbreitete Standard-Mess-Software gibt, finden Sie auf den Testservern im Ordner `/public/` eine rudimentäre Mess-Software. Der Server-Teil der Software ist auf den öffentlichen Testservern bereits kompiliert und lässt sich mithilfe von `/public/ttfb_server` ausführen. Kopieren Sie den Client in ihre virtuelle Maschine und kompilieren Sie diesen ggf. selbst. Erweitern Sie entweder den Client derart, dass dieser die Zeit bis zum 1. empfangenen Nutzdaten-Byte misst, oder messen Sie die gesamte Laufzeit des Programms mit einer definierten Anzahl Replikationen. Achten Sie darauf, dass Ihre Messungen nicht die Verfügbarkeit der Testserver beeinträchtigen. Interpretieren Sie Ihre Messergebnisse und stellen Sie insbesondere Unterschiede zur Durchsatzmessung auf maximal einer zusätzlichen Seite dar.

Sollten sich - zum Beispiel bei einer sehr guten Internetverbindung Ihrerseits - keine Unterschiede feststellen lassen, können Sie künstlich eine schlechtere Internetverbindung emulieren. Beachten Sie in diesem Fall jedoch, dass sich Messungen (egal welcher Art) nur vergleichen lassen, wenn diese Parameter unverändert sind.
Mithilfe des Befehls `tc qdisc add dev eth0 root netem loss 5%` in Ihrer virtuellen Maschine werden ausgehende Pakete mit einer Wahrscheinlichkeit von 5% verworfen. Einen höheren Delay können Sie (optional auch zusätzlich) mit dem Befehl `tc qdisc add dev eth0 root netem delay 20ms 10ms` emulieren, der ausgehende Pakete um 20 ± 10 Millisekunden verzögert. Das Beenden der Emulation erreichen Sie durch `tc qdisc del dev eth0 root`. In allen Fällen müssen Sie `eth0` durch das jeweilige Netzwerkinterface ersetzen.

Happy coding, writing and evaluating!

[16] Diese Fragestellungen werden in Fachveröffentlichen als Quality of Experience bezeichnet.

Abgabe:

- ZIP-Archiv (*ITS_WiSe_202021_Hausarbeit_MatrNo.zip*), das beim Entpacken mit *unzip* automatisch den Ordner *ITS_WiSe_202021_Hausarbeit_MatrNo* anlegt.
- In diesem Ordner müssen sich alle Quelldateien sowie die Hausarbeit und die Erklärung zur selbstständigen Abfassung der Hausarbeit, mindestens also die folgenden Dateien befinden:

```
ITS_WiSe_202021_Hausarbeit_MatrNo
├── ITS_WiSe_202021_Hausarbeit_MatrNo.pdf
├── Erklaerung.pdf oder Erklaerung.png oder Erklaerung.jpg
├── proxy_server.py
├── proxy_client.py
├── ssh_proxy_client.py
├── pki/
│   ├── certificates/
│   │   ├── ca.key
│   │   ├── ca.pem
│   │   ├── example.key
│   │   ├── example.csr
│   │   ├── example.pem
│   │   ├── server.key
│   │   ├── server.csr
│   │   ├── server.pem
│   │   ├── client1.key
│   │   ├── client1.csr
│   │   ├── client1.pem
│   │   ├── client2.key
│   │   ├── client2.csr
│   │   └── client2.pem
│   └── initialize_pki.py oder initialize_pki.sh
└── evaluation/
    └── evaluation.pcap sowie ggf. README und Evaluationsskript(e)
```

Abgabevariante:

- Upload des ZIP-Archives in **StudIP** (Unter der **neuen Veranstaltungen** *Prüfung IT- und Netzwerksicherheit* → *Aufgaben* → *Hausarbeit* die Datei hinzufügen)
- **Ihr ZIP-Archiv muss einen Scan oder ein Foto der handschriftlich unterschriebenen „Erklärung zur selbstständigen Abfassung der Hausarbeit“ enthalten!**
- Sollte Ihre Abgabe eine Dateigröße von 10MB überschreiten, teilen Sie die Zip-Datei bitte mithilfe des folgenden Befehles auf und geben Sie die Einzelteile ab: `split -d -b 9M ITS_WiSe_202021_Hausarbeit_MatrNo.zip ITS_WiSe_202021_Hausarbeit_MatrNo.zip`

Referenzen

[1] D. Anderson. Splinternet behind the great firewall of china. ACM Queue, 10(11):40–49, Nov 2012. https://doi.org/10.1145/2390756.2405036

[2] https://www.latex-project.org

[3] https://man.openbsd.org/ssh#L

[4] https://www.openssl.org/docs/manmaster/man1/openssl.html

[5] https://docs.python.org/3.8/library/ssl.html

[6] https://wiki.ubuntuusers.de/tcpdump/

[7] https://matplotlib.org

[8] http://www.gnuplotting.org

[9] https://www.r-project.org

[10] https://sourceforge.net/projects/iperf2/

[11] https://www.openssh.com/txt/socks4.protocol

[12] https://www.openssh.com/txt/socks4a.protocol

Erklärung zur selbstständigen Abfassung der Hausarbeit

Name: _____

Geburtsdatum: _____

Matrikelnummer: _____

Ich versichere, dass ich die eingereichte Hausarbeit und den zugehörigen Programmcode selbstständig und ohne unerlaubte Hilfe verfasst habe. Anderer als der von mir angegebenen Hilfsmittel und Schriften habe ich mich nicht bedient. Alle wörtlich oder sinngemäß den Schriften anderer Autoren entnommenen Stellen habe ich kenntlich gemacht.

_____ _____

Ort, Datum Unterschrift