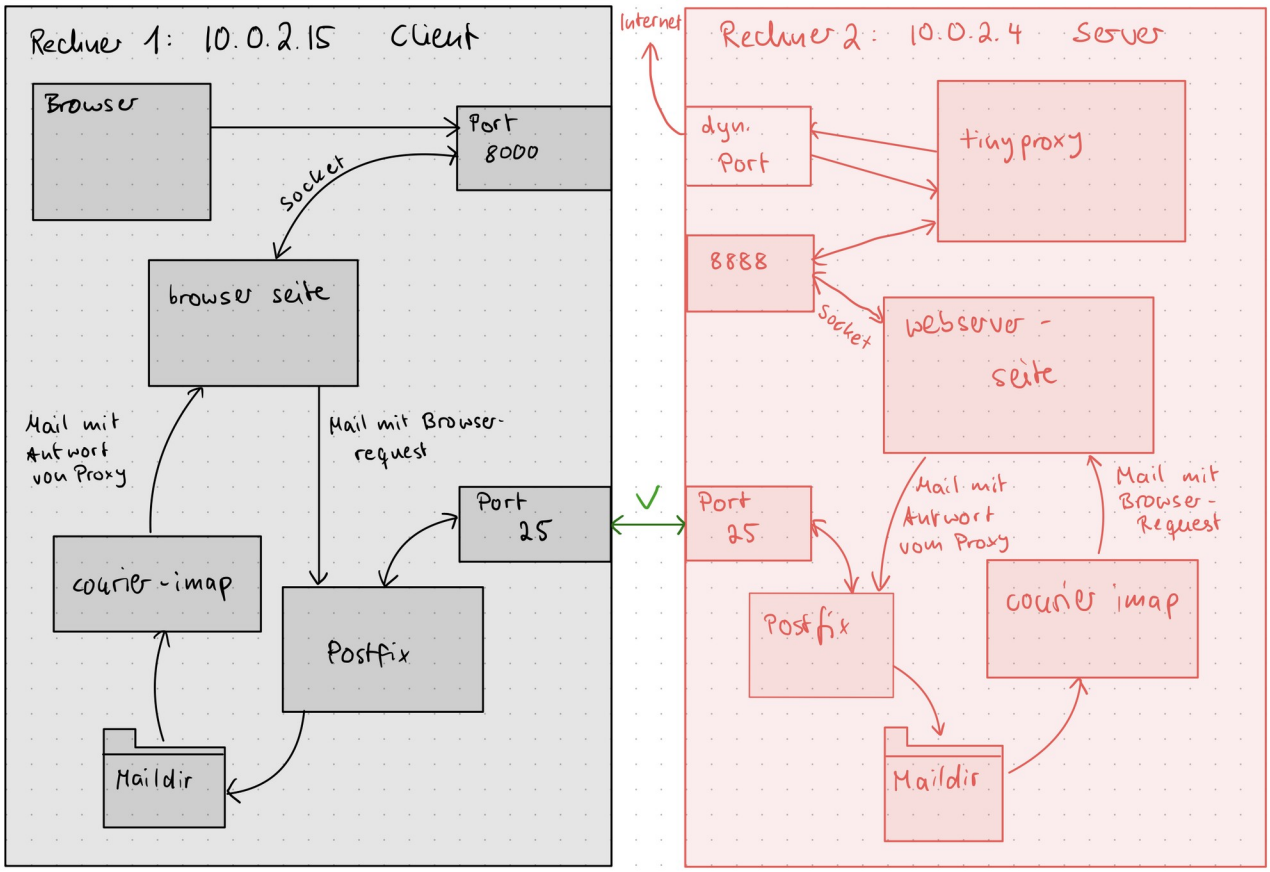
Modul 63585 Fachpraktikum IT-Sicherheit, IT-Forensik und Datenschutz - SoSe 2023

Dokumentation:

Gruppe:
Herzog, Richard
Lubrich, Vanessa

Schematische Darstellung der Konfiguration



Auswahl und Konfiguration der verwendeten Tools

Mailserver:
Für unseren Mailserver haben wir uns als MTA für Postfix entschieden, da Postfix leicht zu konfigurieren ist und eine gute Performance bietet.
Bei der Installation über apt haben wir folgende Einstellungen vorgenommen:

- General type of mail configuration? - Internet Site
- System mail name und root mail recipient festgelegt
- Standardeinstellungen wurden für die meisten Parameter beibehalten

In der Konfigurationsdatei „main.cf“ haben wir folgende Einstellungen vorgenommen:

- als Relayhost die IP-Adresse des jeweils anderen beteiligten Rechners eingetragen, damit alle Mails automatisch an die richtige Adresse weitergeleitet werden, ohne einen Domainnamen registrieren zu müssen → dies stellt den SMTP-Tunnelaspekt dar

– „Maildir/“ als home_mailbox eingetragen

Um die „Maildir/“-Struktur zu initialisieren wurden folgende Befehle verwendet:
$ echo 'export MAIL=~/Maildir' | sudo tee -a /etc/bash.bashrc | sudo tee -a /etc/profile.d/mail.sh

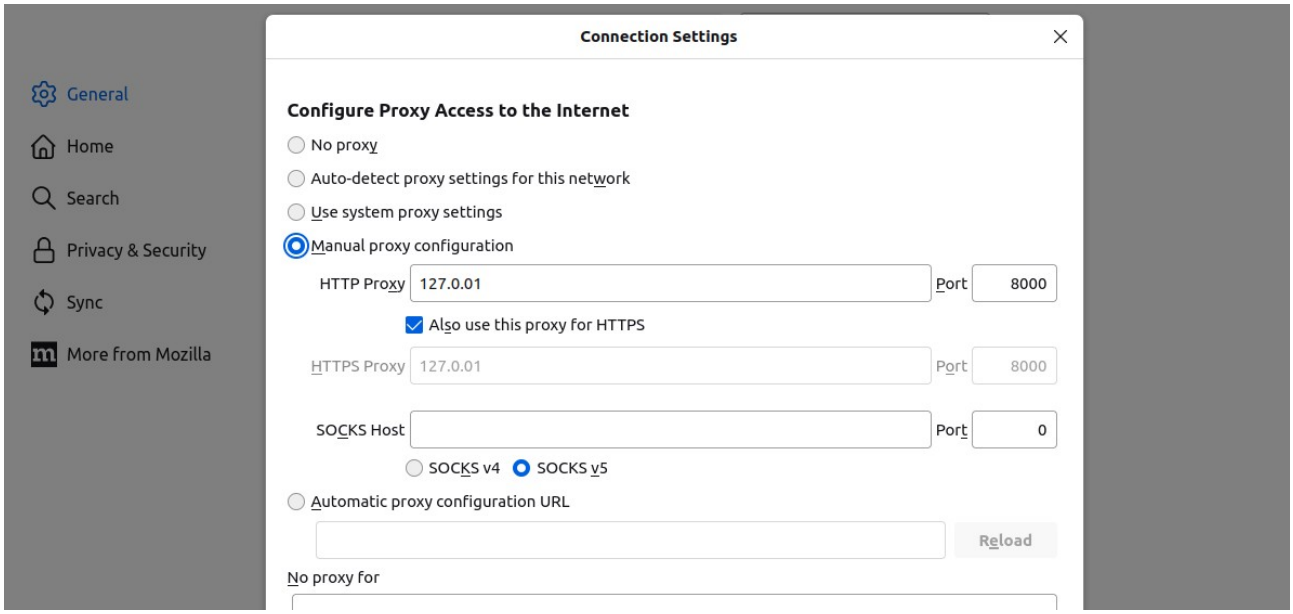
$ source /etc/profile.d/mail.sh

Im ersten Schritt wird festgelegt, dass die Umgebungsvariable „MAIL“ auf die „Maildir/“-Struktur zeigen soll. Im zweiten Schritt wird die neu zugewiesene Umgebungsvariable in die aktuelle Sitzung geladen.

Um die E-Mails abrufen zu können, haben wir uns für Courier IMAP als IMAP-Server entschieden, da er schnell ist und mit Maildirs arbeitet.

Proxyserver:
Auf dem Rechner, den wir als Webserver verwenden, haben wir tinyproxy (Release 1.11.1) für die Verbindung zum Internet installiert. Die Installation haben wir ebenfalls über apt durchgeführt. An der Konfiguration von tinyproxy mussten wir nichts ändern, da wir unsere Browserrequests an den Standard-Listeningport (8888) von tinyproxy weiterleiten.

Browser:
In unserer Konfiguration verwenden wir als Browser Firefox in der Version 113.0.2, dieser ist bei Linux Mint bereits vorinstalliert und bietet eine einfache Möglichkeit, in den Einstellungen einen Proxyserver zu definieren.
Unsere Einstellung sieht wie folgt aus:



Dokumentation des Programmcodes

Zum Schreiben des Codes wurde PyCharm 2022.3.3 (Community Edition) verwendet.
Als Interpreter wurde Python 3.11 verwendet.

Der Code umfasst die folgenden selbstgeschriebenen Module:

- **browser_side.py:**

Dieses Modul muss auf dem Rechner auf dem der Browser läuft, ausgeführt werden. Das Modul enthält eine Main-Methode, in der eine Kommunikation mit der HTTPS-Proxy-Seite gestartet wird, sobald der Browser dies initiiert.

- **communication.py:**
  Dieses Modul enthält die Methoden, die die eigentliche Kommunikation zwischen Browser und HTTPS-Proxy übernehmen.

- **connection.py:**
  Dieses Modul enthält Methoden, mit denen man einem Browser oder einem HTTPS-Proxy einen Socket für die Kommunikation zur Verfügung stellt. Ein Socket wird sowohl für das Senden als auch für das Empfangen verwendet.

- **https_proxy_side.py:**
  Dieses Modul muss auf dem Rechner auf dem der HTTPS-Proxy läuft, ausgeführt werden. Das Modul enthält eine Main-Methode, in der eine Kommunikation mit der Browser-Seite gestartet wird, sobald die Browser-Seite dies anstößt.

- **imap_mail_receiving.py:**
  Dieses Modul ermöglicht das Abholen der Mails bei einem Mail-Server und das Auslesen der, in den Mails enthaltenen Daten.

- **mail_subjects.py:**
  Dieses Modul ermöglicht es, für Mails je nach Art der enthaltenen Information einen einheitlichen Betreff festzulegen.
  Der Betreff spielt eine wesentliche Rolle beim Senden, Abholen und Löschen der Mails aus einer Mailbox.

- **smtp_mail_sending.py:**
  Dieses Modul ermöglicht das Verpacken von Daten (bspw. von einem Socket) in Mails und das Senden an einen Mail-Server.

- **unique_id.py:**
  Dieses Modul ermöglicht die eindeutige Zuordnung von Mails zu einer Kommunikation zwischen Browser-Seite und HTTPS-Proxy-Seite.

Starten des Programms über das Terminal

vor dem Programmstart:

- Bei dem Rechner, auf dem der Internetzugriff gemacht werden soll, muss Tinyproxy gestartet werden.
- Bei beiden Rechnern muss Postfix und Courier IMAP gestartet werden.
- Bei dem Rechner, auf dem Firefox läuft, muss bei Firefox in den Netzwerkeinstellungen als Proxyhost der localhost und ein freier Port eingetragen werden. Der Standardport von browser_side.py ist 8000.
- Es kann helfen den Browser-cache von Firefox zu löschen. Die Option dazu findet sich unter “Anwendungsmenü → Hilfe → weitere Informationen zur Fehlerbehebung”.
- Es kann helfen, alle Erweiterungen die zu Firefox hinzugefügt wurden, zu stoppen.
- Die Firewall muss bei beiden Rechnern so eingestellt sein, dass die Kommunikation

ohne Probleme stattfinden kann. Es muss also Postfix mit Port 25 und Courier IMAP mit Port 143 erlaubt sein.

1. Allgemeines Kommando um die HTTPS-Proxy-Seite zu starten:
   - **`python3.11 /pfad/zu/https_proxy_side.py`**
2. Allgemeines Kommando um die Browser-Seite zu starten:
   - **`python3.11 /pfad/zu/browser_side.py`**

verfügbare Kommandozeilen-Parameter für browser_side.py und https_proxy_side.py:

- **`-h, --help:`** Zeigt Informationen über die verfügbaren Parameter an, hauptsächlich Standardwerte, Datentypen und Kurzbeschreibungen.
- **`-sock_port:`** Standardmäßig 8000 bei der Browser-Seite und 8888 bei der Webserver-Seite
- **`-lin_usr:`** Benutzername; Dieser Benutzer hat Zugriff auf die Mailbox an die Postfix Mails weiterleitet.
- **`-lin_usr_pw:`** Passwort des Benutzers. Beim Abholen der Mails ist eine Authentifizierung mit diesem Passwort notwendig.
- **`-send_addr:`** E-Mail Adresse des Senders von dieser Kommunikations-Seite.
- **`-recv_addr:`** E-Mail Adresse des Empfängers.

Design-Entscheidungen

- **Tinyproxy:** Durch die Verwendung eines HTTPS-Proxies entsteht eine massive Erleichterung für unsere Lösung, da die gesamte Kommunikation inklusive Verbindungsaufbau mit dem Webserver nun von eben diesem HTTPS-Proxy übernommen wird und wir somit die Daten nur weiterleiten müssen.
- **Anzahl der Mail-Server:** Die Entscheidung zwei Mail-Server zu verwenden, hat den Vorteil, dass keine der IMAP-Verbindungen über das Internet erfolgen muss, und hier somit nicht leicht ein ungewollter Datenabfluss von E-Mail-Daten entstehen kann.
- **Programmiersprache:** Den Quellcode haben wir mit **Python 3** geschrieben, da uns einerseits Python bekannt ist und andererseits eine gute Dokumentation für die verwendeten Module existiert.
- **Verwendung von Sockets:** Unser Proxy stellt einen **Socket**[1] zur verfügung. Dafür bedarf es nicht mehr als drei Zeilen Code. Ein Browser (Firefox) kann dann schon von diesem Socket Daten empfangen bzw. senden. Ein HTTPS-Proxy wie Tinyproxy kann von so einem Socket direkt die (Byte-) Daten empfangen, die aus den E-Mails geholt wurden und gleichermaßen die (Antwort-) Daten von einem Webserver wieder an diesen Socket übergeben.
- **Verwendung von Threads:** Wir haben uns dazu entschlossen, Threads bei der Kommunikation zu verwenden. Es werden pro Kommunikations-Seite so viele Threads gestartet, wie der Browser neue Verbindungen über den Socket herstellen möchte. Der Grund, einen Thread eine komplette Kommunikation alleine führen zu lassen ist der, dass E-Mails senden, E-Mails empfangen, vom Socket empfangen und zum Socket senden Aufgaben sind, die zu genau festgelegten Momenten erledigt werden müssen. Es ist hervorzuheben, dass keine zwei Aufgaben gleichzeitig erledigt werden sollen bzw. müssen.
- **Nicht-blockierende Sockets:** Es ist während einer Kommunikation leicht möglich, dass ein Thread im Programm an die Stelle kommt, wo er versucht, Daten vom Socket zu empfangen, der Socket aber im Moment keine Daten hat, oder diese noch nicht freigeben möchte. Daher ist es wichtig, dass das Empfangen vom Socket mit der recv-Methode nicht blockierend ist, da sonst ggf. für immer gewartet werden würde, bis Daten ankommen. Aus

1 Instanzen der Klasse “socket” aus dem Modul “socket”

diesem Grund haben wir uns dazu entschlossen, die Sockets beim Empfangen, nach einer gewissen Zeit einen Timeout-Error erzeugen zu lassen, welcher von uns angemessen behandelt wird.

- **smtplib:** Für den Anschluss von unserem Proxy zum Mail-Server haben wir das Modul "**smtplib**" verwendet. Alternativ dazu wäre es möglich gewesen die Daten an einen bestehenden Mail-Client, bspw. Thunderbird, als Anhang zu übergeben. Wir haben uns aber für die Verwendung von smtplib entschieden, da uns das einfacher erschien, als eine zusätzliche Konnektivität für einen Mail-Client zu implementieren.
- **imaplib:** Für den Anschluss von unserem Proxy zum IMAP-Server haben wir das Modul "**imaplib**" verwendet. Die darin enthaltenen Klasse IMAP4 stellt einen Server zur verfügung, mit dem man bspw. E-Mails bei einem Mail-Server abholen kann.
- **Verbindungs IDs:** Die Verbindungen werden mit IDs versehen. Hierzu wird ein Modul "**unique_id.py**" verwendet. Hier speziell ein Objekt der Klasse "**ConnectionID**". Diese Objekt existiert nur einmal pro Programmlauf und stellt IDs für alle Verbindungen bereit. Die Verbindungs-IDs werden eingesetzt, um den Sockets die richtigen Daten zu übergeben und die richtigen Mails beim Mail-Sever abzuholen.
- **E-Mail Betreff:** Da es sich bei den E-Mails um verschiedenartige Signalisierungen handeln kann, etwa E-Mails mit Byte-Daten für die Sockets, IDs für Verbindungen die von der Webserver-Seite an die Browser-Seite propagiert werden, Mails die das Ende einer Verbindung von einer Seite aus kennzeichnen und noch weitere, haben wir uns dazu entschieden, Die E-Mails anhand ihres Betreffs zu behandeln. Dazu werden sowohl beim Senden der Mails als auch beim Abholen der Mails aus der Mailbox die selben Betreff-Phrasen verwendet. Hierzu verwenden wir ein eigenes Modul "**mail_subjects.py**" mit dem Enum-Typ "**MailSubject**". Der Vorteil von dieser Vorgehensweise ist, dass für viele E-Mails garkeine Inhalte abgerufen werden müssen. Es reicht in diesen Fällen alleine die Tatsache, dass eine E-Mail mit einem bestimmten Betreff empfangen wurde.
- **Löschen von E-Mails:** Wenn eine E-Mail mit Nutzerdaten empfangen, geholt und entpackt wurde, dann wird sie anschleißend gleich aus der Mailbox gelöscht um sie nicht erneut an einen Socket zu übermitteln. Dies muss getan werden, da unser Proxy sich nicht merkt welche E-Mails er schon abgerufen hat. Man möchte glauben das könne man lösen indem man eine Liste mit IDs von E-Mails erstellt in der man die IDs der Mails die man abgerufen hat einträgt. Jedoch werden Mails aus dem Postfach immer nur nach einem Teil des Betreffs, der im aktuellen Kontext der zu holenden Mails steht geholt. Dieser Teil des Betreffs wird immer durch ein Attribut eines Objekts vom Enum-Typ "MailSubject", in kombination mit der ID der Verbindung dargestellt und hat nichts mit der ID der E-Mail zu tun (mehr dazu im nächsten Punkt). Es werden bspw. alle Mails mit dem Teil-Betreff "byte_data+Verbindungs-ID" abgerufen, also nicht nur die mit IDs die noch nie abgerufen wurden. Würden bereits abgerufene E-Mails also nicht gelöscht werden, dann würden sie in jedem weitern Durchgang erneut abgerufen werden.
- **Mail IDs:** Mails die Byte-Daten enthalten, haben im Betreff zusätzlich zur ID der Verbindung, auch noch eine fortlaufende ID von einem Objekt der Klasse "**Mail-ID**". Die Klasse Mail-ID ist ebenfalls im Modul "unique_id.py" deklariert. Jede Mail mit Byte-Daten erhält dadurch einen einzigartigen Betreff. Jede Verbindung hat ihr eigenes Mail-ID-Objekt. Da E-Mails mit Nutzerdaten erst durch eine Kombination aus Verbindungs-ID und Mail-ID eindeutig unterscheidbar werden und die E-Mails anhand ihres Betreffs aus dem Postfach gelöscht werden, ist die Mail-ID nur deswegen vorhanden, um eine bereits abgerufene E-Mail anhand ihres Betreffs löschen zu können, ohne das andere E-Mails mit der selben Verbindungs-ID ungewollt gelöscht werden.
- **Eindeutiger E-Mail Betreff:** Da beim Durchsuchen der Mailbox mit der Methode

imaplib.IMAP4.search(criterion) alle Mails abgerufen werden bei denen das Abrufkriterium nur einen Teilstring des Betreffs kleiner dem vollständigen Betreff abdecken muss, sieht die Syntax für einen Betreff von Mails mit Byte-Daten wie folgt aus: **String+Int.Int.** .Der Betreff enthält also noch ein Präfix (**+**), ein Infix (**.**) und ein Suffix (**.**).

- **die Verwendung von threading.Lock:** Da es pro Verbindung zu einem Webserver mindestens einen Thread pro Kommunikations-Seite gibt, verwenden wir an Programmabschnitten an denen die Mailbox “Inbox“ in irgendeiner Weise benutzt wird, ein Objekt der Klasse “Lock“ aus dem Modul “threading“. Wenn das Programm bei einem Thread an eine Stelle mit “with lock“ ankommt, dann wird versucht diesen Abschnitt mit dem threading.Lock-Objekt zu entsperren, sodass man den Programmabschnitt ausführen kann. Falls aber irgendein anderer Thread gerade einen kritischen Programmabschnitt ausführt, müssen alle Threads die ebenfalls so einen Abschnitt ausführen wollen, an einer Stelle mit “with lock“ warten, bis das Lock-Objekt wieder freigegeben wird. Das Lock-Objekt wird automatisch freigegeben sobald ein Thread einen kritischen Programmabschnitt beendet hat. Der Grund warum wir ein threading.Lock-Objekt verwenden ist der, dass wir verhindern wollen, dass sich mehrer Threads beim Manipulieren der gemeinsamen Mailbox gegenseitig behindern.

Quellen

- https://github.com/anapeksha/python-proxy-server/blob/main/src/server.py: Abgerufen am 13.07.2023. Hier wurden wir einerseits von dem Teil inspieriert, der die Konnektivität mit dem Browser realisiert und andererseits von dem Teil der einen neuen Socket für die Gegenseite, also für den HTTPS-Proxy, bereitstellt. Auch das Einlesen der Kommandozeilenargumente haben wir von hier übernommen. Im wesentlichen wird das Wissen von dieser Quelle in den folgenden Modulen eingesetzt: **browser_side.py, https_prox_side.py, connection.py und communication.py**
- https://humberto.io/blog/sending-and-receiving-emails-with-python/: Abgerufen am 13.07.2023. Hier wird beschrieben, wie man mit Hilfe eines IMAP-Server, Mails aus einer Mailbox holt. Auch wird gezeigt, wie man an die Nutzdaten der Mails heran kommt und wie man die Mails anschließend löscht. Im wesentlichen werden diese Kenntnisse im Modul **imap_mail_receiving.py** eingesetzt.
- https://www.youtube.com/watch?v=mWZYn5I_jkY: Abgerufen am 13.07.2023. Wird teilweise im Modul **smtp_mail_sending.py** verwendet
- https://www.digitalocean.com/community/tutorials/how-to-install-and-configure-postfix-on-ubuntu-20-04-de: Abgerufen am 14.07.2023. An dieser Anleitung haben wir uns bei der Installation/Konfiguration von Postfix orientiert.
- https://docs.gitlab.com/ee/administration/reply_by_email_postfix_setup.html#test-the-out-of-the-box-setup: Abgerufen am 14.07.2023: An dieser Anleitung haben wir uns für die Installation von Courier IMAP orientiert.