

OS-Fingerprinting Projektbericht

Projekt zur Vorlesung: Introduction to Internet and Security

Alexander Luyten | Elias Arnold

4. Mai 2021

Inhaltsverzeichnis

1	Was ist OS-Fingerprinting	2
2	Warum OS-Fingerprinting?	3
3	Methoden	3
4	Setup	4
4.1	Konfiguration	4
4.2	Erstversuch	5
5	Schutz vor OS Fingerprinting	5
6	Resultate	6
6.1	Endprodukt	6
6.2	Probes	7
6.3	Packet Parser	8
6.4	Interpretation der Informationen	8
6.5	Resultate	8
6.6	Lessons Learned	10
A	Anhang	12
A.1	Screenshot erster Versuch	12
A.2	Sequenzdiagramm erster Versuch	13
A.3	Screenshot des Fakers	14
A.4	UML-Klassendiagramm von FingerBerry	15
A.5	Screenshot des Probesenders	16
A.6	Screenshot des Parsers	16

1 Was ist OS-Fingerprinting

Unter dem Begriff 'OS-Fingerprinting' versteht man das Erkennen des Betriebssystems, das von einem Zielhost verwendet wird. Um dies zu bewerkstelligen schaut man sich die Datenpakete dieses Hosts genauer an. Damit ein solches Paket das gewünschte Ziel erreichen kann und nicht irgendwo in den Tiefen des (heutzutage) riesigen Netzwerk von Netzwerken, dem Internet, verschwindet, müssen sich die Sender solcher Pakete an strenge Vorschriften halten. Eine ganze Palette von weiteren Informationen müssen in der richtigen Reihenfolge und Länge an die Nachricht angefügt werden (sog. "Header"). Der Inhalt einer Nachricht kann natürlich beliebig sein (In der Fachsprache "Payload" genannt). Man kann bei dem eben erklärten Prinzip Analogien zu vielen anderen Sender-/Empfänger-Prozessen erkennen. Zum Beispiel funktioniert die Briefpost nach einem ganz ähnlichen Prinzip. Der beliebige Inhalt eines Datenpaketes kann mit dem individuellen Text im Nachrichtenfeld einer Postkarte verglichen werden. Die streng vorgeschriebenen Mehrinformationen entsprechen hier der Angabe einer Empfängeradresse und dem Frankieren mit einer Briefmarke genügenden Werts. Üblicherweise kann der Empfänger eines Briefes/Postkarte den Absender eindeutig identifizieren, weil der Inhalt des Briefes vom Absender gekennzeichnet ist. Im Internet funktioniert das nicht. Fordert man ein Server dazu auf, Informationen zukommen zu lassen (z.B. durch Aufrufen einer Webseite), kann man zwar seine Adresse bestimmen, mit der er im Internet kommuniziert (mit Hilfe des DNS-Servers), jedoch kann man daraus keine Informationen über den Server selbst ableiten. Da dieser Server aber eine Antwort über das Internet an den Sender zurückschickt, muss auch er zusätzliche Informationen an die eigentliche Antwort anhängen. Diese Aufgabe wird von TCP/IP-Stack übernommen, der je nach Betriebssystem etwas anders aussieht, und dementsprechend leicht andere Pakete verschickt. Diese minimalen Unterschiede macht man sich beim OS-Fingerprinting zu Nutzen und versucht daraus trotzdem Rückschlüsse über den Server, insbesondere über das verwendete Betriebssystem, zu ziehen. Bei der Briefpost kann man das mit der Handschrift des Senders vergleichen, die zwar je nach Sender unterschiedlich ist, aber (falls leserlich geschrieben) trotzdem vom Empfänger gelesen werden kann.

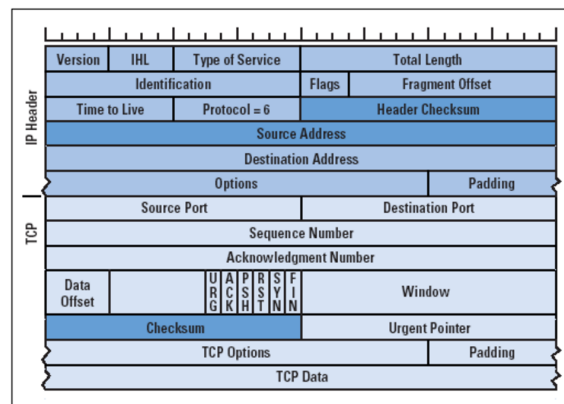


Abbildung 1: Der IP-Header der bei jedem Datenpaket vorhanden sein muss. Sowie der TCP-Header, welcher für eine stabile Verbindung zwar benötigt, aber nicht zwingend vorhanden sein muss. [1]

2 Warum OS-Fingerprinting?

Wie im vorherigen Kapitel bereits erläutert, kann man durch gezieltes Auslesen der Headerinformationen von Paketen im Internet, Informationen über das Betriebssystem eines Senders gewinnen. Doch welchen Nutzen ergibt sich aus den Informationen, die man mittels OS-Fingerprinting gewinnt? Der sicherheitsbewusste Leser wird nun gleich an einen massiven Einschnitt in die Privatsphäre sowie an Spionage denken. Den ersten Punkt, den Einschnitt in die Privatsphäre, kann man grösstenteils verwerfen, da der Markt nur eine begrenzte Palette an Betriebssystemen bereitstellt, und nicht alle Menschen ein individuelles, privates OS besitzen. Natürlich gibt es hier Ausnahmen. Jedoch muss die Person, die ihr Betriebssystem umkonfiguriert oder sogar komplett neu geschrieben hat, sehr weitreichende Informatikkenntnisse gehabt haben, und musste den Auswirkungen und Folgen Ihrer Tat vollumfänglich bewusst gewesen sein. Unter dem Strich benutzt jeder Mensch ein von Experten entwickeltes und OS, dem man mit OS-Fingerprinting keine persönlichen Informationen entlocken kann. Darum kann man hier nicht von einer Verletzung der Privatsphäre sprechen.

Ein weiteres Vorurteil gegenüber OS-Fingerprinting stellt die Tatsache dar, dass es darum geht, das Verhalten eines Zielsystems auszuspionieren. Diesen Kritikpunkt kann man nicht abstreiten. Für Hacker ist dies eine wichtige Information, um ihre Shellskripte auf dem Zielrechner ausführen zu können [2]. Grosse Firmen heuern sogar Hacker (sog. Pentester) an, welche versuchen, in das Firmennetz einzudringen, um die eigene IT-Infrastruktur auf Herz und Nieren zu prüfen [3]. Egal ob der Hacker böse oder gute Absichten hat, er braucht, um seine 'Arbeit' verrichten zu können, zuverlässige Informationen über das jeweilige Betriebssystem.

Jedoch müssen die gewonnen Informationen nicht unbedingt dazu verwendet werden, leichter in ein fremdes System einzudringen. Manchmal reicht es völlig aus, zu wissen, welche Systeme im eigenen Netzwerk verwendet werden, oder ob diese das tun, was man von ihnen erwartet. So kann eine Firma zum Beispiel testen, ob alle Mitarbeiter mit der aktuellen Version eines Betriebssystems ausgerüstet sind und es bei ihnen reibungslos läuft.

3 Methoden

Alle Methoden, die für OS-Fingerprinting verwendet werden, funktionieren nach dem Prinzip, möglichst viele Informationen über Eigenheiten und Charakteristiken eines Zielsystems zu gewinnen, und diese nachher mit bereits bekannten Informationen über das Verhalten verschiedener Betriebssysteme zu vergleichen. Um das zu bewerkstelligen gibt es verschiedene Ansätze [4]. Dabei kann man die benötigten Informationen von Headern auslesen, die unterschiedlich tief im TCP/IP-Stack an die Nachricht angefügt werden. Wir haben uns für Header entschieden, die im Transport- oder Network-Layer angefügt werden (TCP, ICMP, IP). Wenn man Pakete des Zielhosts auslesen will, dann unterscheidet man zwischen zwei Methoden: Der Aktiven- und der Passiven-Methode. Beim aktiven OS-Fingerprinting generiert man Anfragepakete und verschickt diese an den Zielhost in der Hoffnung, darauf eine Antwort zu bekommen. Bei der passiven-Methode wird kein eigener 'traffic' verursacht, daher können nur Pakete des Zielhosts ausgelesen werden, wenn dieser aus irgendeinem Grund zu senden beginnt. Ein weiteres Unterscheidungsmerkmale der Aktiven Methode ist die Zuverlässigkeit. Man darf davon ausgehen, dass die individuellen Anfragen vom Zielhost beantwortet werden, jedoch fällt man durch das aktive Senden viel eher auf, weshalb die Anonymität beeinträchtigt wird.

4 Setup

4.1 Konfiguration

Ziel des Projektes war es, selber ein Tool zu entwickeln, um OS-Fingerprinting zu betreiben. Die erste Frage die wir uns stellen mussten war, ob wir die Aktive- oder die Passive-Methode verwirklichen sollen. Da in absehbarer Zeit ein Resultat/eine Vermutung über das verwendete Betriebssystem vorliegen soll, haben wir uns für die zuverlässige, Aktive-Methode entschieden. Wie im obigen Kapitel 'Methoden' bereits betont, liegt der Vorteil dieser Methode nicht nur im aktiven Senden von Anfragen an den Zielhost, sondern auch in der individuellen Gestaltung dieser Anfragepakete.

Zu Beginn des Projektes haben wir den Fokus weniger auf den Zusammenbau dieser 'Massgeschneiderten' Anfragen gelegt, sondern viel mehr auf den groben Ablauf eines Tests und die dafür notwendigen Klassen und Methoden. Dabei haben wir unsere Überlegungen mit der Programmiersprache Java in die Tat umgesetzt (Erstversuch). Wie die bereits existierenden Sniffer-Programme wollten auch wir das Pcap Interface implementieren. Unter Windows haben wir dazu die Bibliotheken WinPcap und jNetPcap (Java-Wrapper für WinPcap) verwendet. Das einbinden in Java war ein Kinderspiel [8]. Jedoch hatten wir erhebliche Schwierigkeiten, dasselbe unter Linux-Distributionen wie zum Beispiel Ubuntu zu tun, weshalb wir uns schlussendlich der Programmiersprache Python zuwendeten. Die aussagekräftigsten Informationen mit denen sich Rückschlüsse auf das verwendete Betriebssystem ziehen lassen, befinden sich im IP- und TCP-Header. In allen unseren Versuchen wollten wir mit dem Zielsystem eine TCP-Verbindung aufbauen, um an diese Header zu kommen und daraus die aussagekräftigen Daten zu extrahieren.

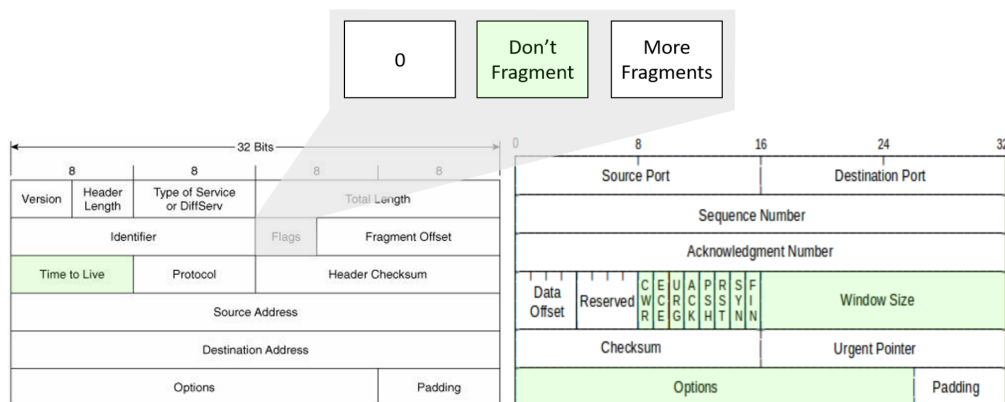


Abbildung 2: Der TCP-Header(rechts) und der IP-Header(links). Datenfelder sind in Hellgrün markiert:

- Time to Live (TTL): Anzahl Hops die ein Paket maximal zurücklegen darf.
- Don't Fragment: Ein Bit das festlegt, ob eine Nachricht in kleinere Stücke unterteilt werden darf. Je nach OS unterschiedlich bei kleineren Paketen.
- TCP-Control-Flags: Diese Zustände werden benötigt, um wichtige Kontrollpakete zu kennzeichnen.
- Window Size: Anzahl der Pakete, die der Empfänger am Stück empfangen möchte
- TCP-Options: Unterschiedlich grosse Zusatzinformationen

4.2 Erstversuch

Die erste Idee war, dem Zielhost ein simples ICMP-Ping-Paket zu schicken, sowie mit ihm eine TCP-Verbindung aufzubauen. Unabhängig davon ob die TCP-Verbindung zustande kommt (3-Wege-Handshake) oder nicht, werden so TCP-Header in beide Richtungen ausgetauscht. In Java blockt die `'connect()'`-Methode aber solange, bis die Verbindung aufgebaut oder abgelehnt wurde. Um dieses Problem zu umgehen, mussten wir mit Threads arbeiten, um gleichzeitig zum Verbindungsversuch die ausgetauschten Datenpakete abfangen zu können. Jedoch ergab sich durch die Arbeit mit Threads wiederum ein neues Problem: Da man nicht voraussagen konnte welcher Thread wie lange für den Verbindungsaufbau braucht, mussten wir die Ausgaben der einzelnen Threads in eine Datencontainer-Klasse umleiten, um sie nach den Tests geordnet ausgeben zu können. Ein Ablaufdiagramm sowie ein Screenshot unseres ersten Versuchs in Java finden Sie im Anhang (A.1 und A.2).

5 Schutz vor OS Fingerprinting

Wie im einleitenden Kapitel bereits angetönt, sind Informationen über das OS auf einem Zielrechner eine sehr wichtige Information für Hacker. So liegt natürlich der Gedanke nicht weit, mit kleinen Veränderungen am TCP/IP Stack potentiellen OS-Fingerprintern ein falsches Betriebssystem vorzugaukeln. Da jeder durch OS-Fingerprinting gewonnenen Information eine bestimmte Relevanz zugeordnet werden muss, reicht es meistens schon aus, einen Parameter des eigenen TCP/IP Stacks zu verändern. Unser Endprodukt verwendet einfache `if` Statements, um das Betriebssystem zu bestimmen. Liegen die erforderlichen Headerfelder erstmal vor, verwendet FingerBerry die TTL des IP-Headers (Dieses Feld ist in jedem Netzwerkpaket präsent) als erste Instanz, um die Suche zwischen den Betriebssystemklassen einzugrenzen. Um sich jetzt vor OS-Fingerprinting zu schützen, oder zumindest um den Angreifer in die Irre zu führen, ist eine Option die eigene TTL (Time to Live) zu verändern, die jedem Netzwerkpaket das den eigenen Host verlässt im IP-Header mitgegeben wird.

Diese Idee verfolgten wir mit unserem Tool: 'Faker' (Screenshot im Anhang A.3). Da FingerBerry auch nur auf Linuxartigen Betriebssystemen läuft, funktioniert der Faker auch nur auf Linux-Derivaten. Jedoch kann man das auch manuell machen, indem man (unter Linux) folgende Datei verändert:

`/proc/sys/net/ipv4/ip_default_ttl`. Auf Linux Betriebssystemen ist Standardmässig der TTL-Wert auf 64 gesetzt. Da diese Zahl aber bei jedem Hop zwischen Ihnen und dem OS-Fingerprinter um eins verringert wird, ist es eine schlechte Wahl, diesen Wert nur gering zu verändern (z.B auf 63 zu setzen). Sie können diesen Wert natürlich frei wählen. Wollen Sie jedoch ein bestimmtes Betriebssystem simulieren, hilft ihnen folgende Liste weiter:

Operating System (OS)	IP Initial TTL
Linux (kernel 2.4 and 2.6)	64
Google's customized Linux	64
FreeBSD	64
Windows XP	128
Windows 7, Vista and Server 2008	128
Cisco Router (IOS 12.4)	255

Abbildung 3: Die Standard TTL Werte nach Betriebssystem [9]

6 Resultate

6.1 Endprodukt

Nach dem 'Fehltritt' mit Java, haben wir uns entschieden, mit Python weiter zu arbeiten. Unser Endprodukt trägt den Namen 'FingerBerry' und ist eine interaktive Konsolenanwendung.

Um die Applikation zu installieren, muss man die benötigten Quelldateien herunterladen, ausserdem muss Python3 auf dem Computer installiert sein. Die Datei `Application.py` muss mit Administratorrechten ausgeführt werden, damit man die benötigten Rechte bekommt, um die Netzwerkinterfaces abzuhören und selber Raw Packets zu verschicken.

Nach Aufstarten der Applikation wird der Benutzer mit dem FingerBerry ASCII Logo begrüsst. Dem Benutzer stehen verschiedene Befehle zur Verfügung. Falls man mit der Syntax nicht vertraut ist, gibt man einfach 'help' ein, und alle möglichen Befehle werden aufgelistet. Das sieht dann etwa wie folgt aus:

- `fingerprint <interface> <target-ip>` – guess of the target OS
- `port <target-ip> <from portNr.> <to portNr.>` – performs a Portscan
- `ip <interface>` – prints current ip of interface
- `interfaces` – lists all Network-interfaces of your computer

Die grundlegende Funktionsweise von 'FingerBerry' ist stark an das Sequenzdiagramm im Anhang angelehnt. Jedoch geht unser Endprodukt weit über die dort vorgesehene Funktionalität hinaus. Ein UML-Klassendiagramm von 'FingerBerry' finden sie ebenfalls im Anhang (A.4).

6.2 Probes

Probes sind gefälschte TCP oder ICMP Pakete, die dem Zielhost gesendet werden. Nach Versenden der verschiedenen Probes wartet man auf eine Antwort von dem Zielhost. Die Antwortpakete werden dann analysiert.

Verschiedene Betriebssysteme antworten anders auf die verschiedenen Probes. In der Regel sind die Probes nicht sinngemässe TCP oder ICMP Pakete. In unserem FingerBerry Fingerprinting Tool verwenden wir drei verschiedene Probes:

- TCP Probe: Sinngemässes TCP Packet, dass dem Zielhost auf einem offenen Port gesendet wird. Dient lediglich zum Auslesen verschiedener Informationen, wie z.B. TTL, Window Size und andere TCP Optionen die gesetzt sind.
- ICMP Probe: Dient zum Auslesen der ICMP Optionen.
- FIN Probe: Ist ein nicht sinngemässes TCP Packet. Dem Zielhost wird ein TCP Packet gesendet mit dem FIN Bit gesetzt. Da keine Verbindung initialisiert wurde, also der 3-Way Handshake garnie stattgefunden hat sollte der Zielhost gemäss RFC 793 TCP ein Antwortpaket mit dem RST Bit gesetzt zurücksenden. Was wirklich passiert, variiert von Betriebssystem zu Betriebssystem. Ein Screenshot des Probesenders finden sie im Anhang (A.5)

In unserer Applikation versenden wir zuerst eine FIN-Probe danach eine TCP- und eine ICMP-Probe. Danach evaluieren wir die Antwortpakete nach verschiedenen Eigenschaften.

6.3 Packet Parser

Ein Grossteil der Arbeit an unserer Applikation ging in das Auslesen der verschiedenen Antwortpakete, die wir vom Zielhost empfangen haben. Beim Auslesen der Daten hatten wir viele Schwierigkeiten, da wir nicht sehr gut vertraut waren mit dem TCP oder IP Header. Unsere Applikation liest die Antwortpakete als Bytes aus und interpretiert die verschiedenen Bytes anders. Dabei werden die verschiedenen IP, TCP oder ICMP Header anders interpretiert. Die Bytes werden mit der Python `struct.unpack` Funktion in Datenformate umgewandelt, die man interpretieren kann, um das Betriebssystem zu erkennen. Ein Screenshot davon finden sie im Anhang (A.6).

6.4 Interpretation der Informationen

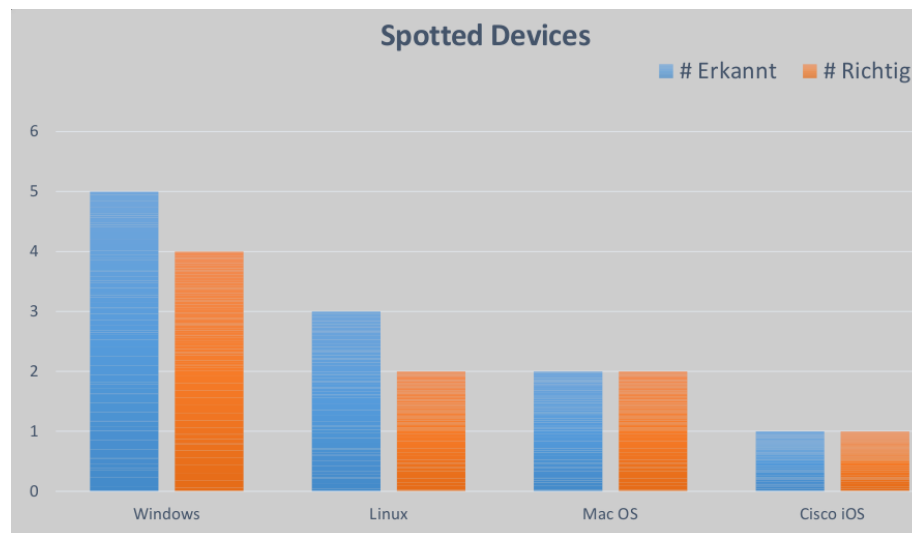
Nach Versenden von ICMP-/TCP-/FIN-Probes und Empfangen von Antwortpaketen können wir alle erhaltenen Informationen interpretieren. Für die Erkennung der verschiedenen Betriebssysteme haben wir am meisten Gewicht auf die Time to Live Headerinformation gesetzt. Diese Informationen haben uns am einfachsten eine Grobeinteilung der grössten Betriebssysteme gegeben und sind für praktisch alle Betriebssysteme per Default gleich gesetzt. Nach dem wir eine Grobeinteilung von Linux, Mac OS und Windows erhalten, unterteilen wir diese Betriebssysteme weiter, indem wir die Window Size betrachten. Dies ermöglicht es uns verschiedene Windows Betriebssysteme zu unterscheiden.

Zwischen Mac OS und Linux entscheiden wir, indem wir betrachten, ob eine Antwort erhalten wurde von der FIN-Probe oder nicht.

6.5 Resultate

Unser Fingerprinting Tool haben wir innerhalb dem Uni-Netzwerk an verschiedenen Zielhosts getestet. Die Resultate, die wir bekamen, waren ziemlich erfreulich. Bei 11 Zielsystemen haben wir 9 von 11 richtig erkannt. Dies entspricht einer richtigen Zuordnung von fast 80%.

Was jedoch zu beachten ist, dass unsere Applikation nur die grossen Betriebssysteme: Linux, Mac OS, Cisco IOS, Windows 2000, Windows XP, Windows 7 und höher unterscheiden kann. Dies ist in der Praxis nicht allzu nützlich, da man mehr Informationen erhalten will über die genaue Version des Betriebssystems, aber unsere Applikation macht eine zuverlässige Grobunterteilung der verschiedenen Betriebssystemen.



6.6 Lessons Learned

Während der Arbeit am Projekt haben wir viel über die einzelnen Header und den Sinn und Zweck ihrer Felder gelernt. Die verschiedenen Optionen des TCP-Headers haben unser Interesse besonders geweckt. An dieser Stelle möchten wir ihnen kurz die 'Window-Scale-Option' vorstellen:

Die 'Window-Scale-Option' ist, wie der Name schon vermuten lässt, eine optionale Zusatzinformation für den TCP-Header. Im standardisierten Teil dieses Headers mit fixer Länge (20 Bytes), ist im 'Window-Size' Feld bereits angegeben, wie gross das Empfangsfenster (Menge an Daten die empfangen werden können, ohne deren Erhalt dem Sender zu bestätigen) gewählt werden soll. Da sich im Laufe der Zeit die Übertragungstechniken stets verbessert haben (wenige Bitfehler) und vor allem viel schneller geworden sind, muss das Empfangsfenster vergrößert werden, da es sonst viel zu wenig ausgelastet ist. Diese Aufgabe probiert die 'Window-Scale-Option' zu lösen:

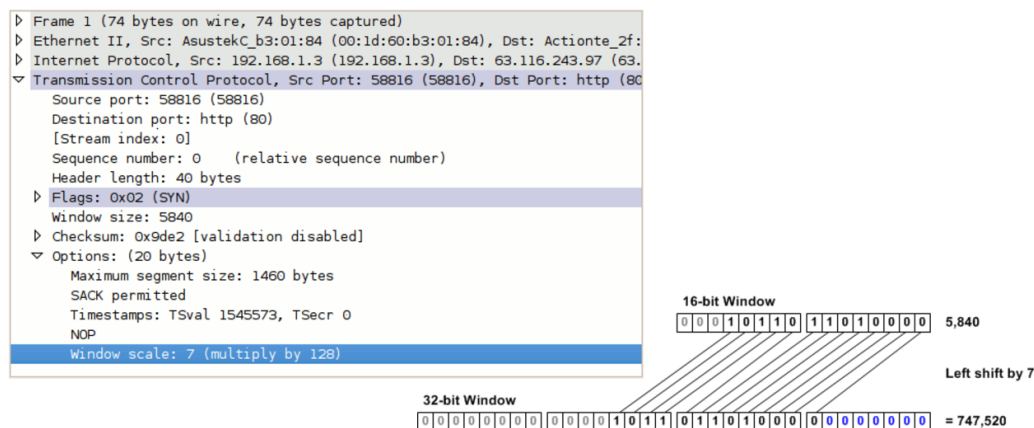


Abbildung 4: Ein Screenshot aus Wireshark (links), sowie eine illustrative Grafik der Veränderung der Empfangsfenstergrösse mittel Bitshifting um sieben Bits nach links (Entspricht Multiplikation mit $2^7 = 128$) [10].

Während des Projektverlaufs mussten wir uns insbesondere mit Kommunikationsproblemen innerhalb des Teams herumschlagen. Wir hätten uns den ganzen Aufwand der mit dem wechseln der Programmiersprache oder dem Starten eines neuen Versuches sparen können, wenn wir uns in der Gruppe besser abgesprochen hätten. Wir arbeiteten einfach drauflos, ohne sich vorher mit dem Teammitglied über das Vorhaben auszutauschen. Wenn wir das Projekt nochmals machen könnten, würden wir uns zuerst über die Ziele des Projekts und die Vorgehensweise Gedanken machen, und erst dann mit dem Code-Schreiben beginnen.

Jedoch gab es aber auch die technischen Probleme, mit denen wir konfrontiert wurden. Beispielsweise gestaltete sich das Auslesen der Control-Flags als ziemlich anspruchsvoll, da in dem Bit-Array das die Zustände der Flags repräsentiert, auch führende Nullen eine Bedeutung haben. Python schneidet diese Nullen weg, wodurch sich das Auslesen des restlichen Headers als extrem schwierig gestaltet. Die Lösung dieses Problems war Python-Methode `zfill()`, die ein Array bis zu einer gewünschten Länge mit Nullen auffüllt (Bei uns `zfill(8)` (8 Bit = 1 Byte)).

Literatur

- [1] Erle Robotics S.L:
https://erlerobotics.gitbooks.io/erle-robotics-introduction-to-linux-networking/content/introduction_to_network/tcp_and_packets.html
- [2] Gordon Lyon, NMAP Network Scanning, Reasons for OS Detection:
<https://nmap.org/book/osdetect.html#osdetect-reasons>(Stand: 12.06.2017)
- [3] Handelsblatt GmbH - ein Unternehmen der Verlagsgruppe Handelsblatt GmbH & Co. KG
<http://www.wiwo.de/erfolg/trends/it-sicherheit-wo-hacker-ihr-unheimliches-handwerk-lernen-seite-4/5156098-4.html>(Stand: 12.06.2017)
- [4] Forensicswiki - OS-Fingerprinting
http://forensicswiki.org/wiki/OS_fingerprinting(Stand: 12.06.2017)
- [5] Gordon Lyon, NMAP Network Scanning, Service and Version Detection:
<https://nmap.org/book/man-version-detection.html>(Stand: 12.06.2017)
- [6] wikipedia.org - nmap:
<https://de.wikipedia.org/wiki/Nmap>(Stand: 12.06.2017)
- [7] Gordon Lyon, NMAP Network Scanning, Port Scanning Basics:
<https://nmap.org/book/man-port-scanning-basics.html>(Stand: 12.06.2017)
- [8] Capturing network packets using jNetPcap API:
<https://compsciplab.wordpress.com/2013/02/17/capturing-network-packets-through-java/>
(Stand: 24.06.2017)
- [9] Common TTL Values by Netresec:
<http://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting>(Stand: 25.06.2017)
- [10] Jeremy Stretch - TCP Windows and Window Scaling:
<http://packetlife.net/blog/2010/aug/4/tcp-windows-and-window-scaling/>(Stand: 28.06.2017)

A Anhang

A.1 Screenshot erster Versuch

```
- Überwachen des Netzwerks mit Wireshark und SecurityVollkrisis: java -jar ProjektVoll.jar
Gefundene Netzwerkkarten:
#0: \Device\NPF_{907B9C87-87F5-4218-8FB6-78B733C26420} [Microsoft]
#1: \Device\NPF_{0EACD475-7C1A-40A3-BAE7-AE9ADB31C2CA} [Realtek PCIe GBE Family Controller]
#2: \Device\NPF_{2E11B0C4-4C9C-41AD-802A-2656CA3411FD} [Microsoft]

Wählen Sie eine Netzwerkkarte aus der obenstehenden Liste aus, indem Sie deren Index eingeben: 1
Wähle 'Realtek PCIe GBE Family Controller' für die Aufzeichnung:
Geben Sie den Typspezifischen Bezeichner des Netzwerkes(A, B oder C) ein, in dem Sie sich befinden: c
Geben Sie eine IPv4 Adresse, gefolgt von der gewünschten Portnummer ein. E.g. <192.168.1.125:6665>: 192.168.1.125:6665
Geben Sie eine Portnummer ein, von der aus eine TCP-Verbindung zum gewählten Ziel (192.168.1.125:6665) aufgebaut werden soll: 80
Bitte warten Sie während die Tests ausgeführt werden

Test: Baue TCP Verbindung mit 192.168.1.125:6665 auf!
Test: Sende ICMP-Ping Paket an 192.168.1.125!
##### TCP-PAKET #####

Ip: ***** Ip4 - "ip version 4" - offset=14 (0x1) length=20 protocol suite=NETWORK
Ip:
Ip:     version = 4
Ip:    hlen = 5 [5 * 4 = 20 bytes, No Ip Options]
Ip:     diffserv = 0x0 (0)
Ip:     0000 00.. = [0] code point: not set
Ip:     .... 0.. = [0] ECN bit: not set
Ip:     .... 0.. = [0] ECE bit: not set
Ip:     length = 92
Ip:     id = 0x0 (0)
Ip:     flags = 0x2 (2)
Ip:     0.. = [0] reserved
Ip:     .1. = [1] DF: do not fragment: set
Ip:     .0 = [0] MF: more fragments: not set
Ip:     offset = 0
Ip:     ttl = 64 [time to live]
Ip:     type = 6 [next: Transmission Control]
Ip:     checksum = 0x8683 (46723) [correct]
Ip:     source = 192.168.1.125
Ip:     destination = 192.168.1.115
Ip:

Tcp: ***** Tcp offset=34 (0x22) length=32
Tcp:
Tcp:     source = 6665
Tcp:     destination = 80
Tcp:     seq = 0xE8D65298 (3956691608)
Tcp:     ack = 0xE8CA223 (1317839395)
Tcp:     hlen = 8
Tcp:     reserved = 0
Tcp:     flags = 0x12 (18)
Tcp:     0... .. = [0] cwr: reduced (cwr)
Tcp:     .0.. .... = [0] ece: ECN echo flag
Tcp:     ..0. .... = [0] ack: urgent, out-of-band data
Tcp:     ...1 .... = [1] ack: acknowledgment
Tcp:     .... 0... = [0] ack: push current segment of data
Tcp:     .... 0.. = [0] ack: reset connection
Tcp:     .... .1. = [1] ack: synchronize connection, startup
Tcp:     .... ..0 = [0] fin: closing down connection
Tcp:     window = 5840
Tcp:     checksum = 0xBA7B (35451) [correct]
Tcp:     urgent = 0
Tcp:
Tcp: + NoOp: offset=28 length=1
Tcp:     code = 1
Tcp:     length = 1 [implied length from option type]
Tcp:
Tcp: + MSS: offset=20 length=4
Tcp:     code = 2
Tcp:     length = 4
Tcp:     mss = 1460
Tcp:
Tcp: + WindowScale: offset=20 length=3
Tcp:     code = 3
Tcp:     length = 3
Tcp:     scale = 4
Tcp:
Tcp: + SACK_PERMITTED: offset=26 length=2
Tcp:     code = 4
Tcp:     length = 2
Tcp:
##### ARRAYS FÜR DIE ANALYSE #####
[64, 5840, 1460, 4]
[false, true, true]
```

Abbildung 5: Die Ausgabe unseres ersten Versuches mit Java. Das Bild zeigt lediglich den TCP-Test. Daneben wurde auch noch eine Ping Anfrage an den Zielhost geschickt (hier nicht im Bild). Man konnte jedes Byte des Antwortpaketes einzeln ansprechen und dessen Inhalt auslesen. Im obigen Bild wurde jede Information aus dem TCP-Paket (IP- und TCP-Header) ausgelesen und in einer Liste aufgeführt.

A.2 Sequenzdiagramm erster Versuch

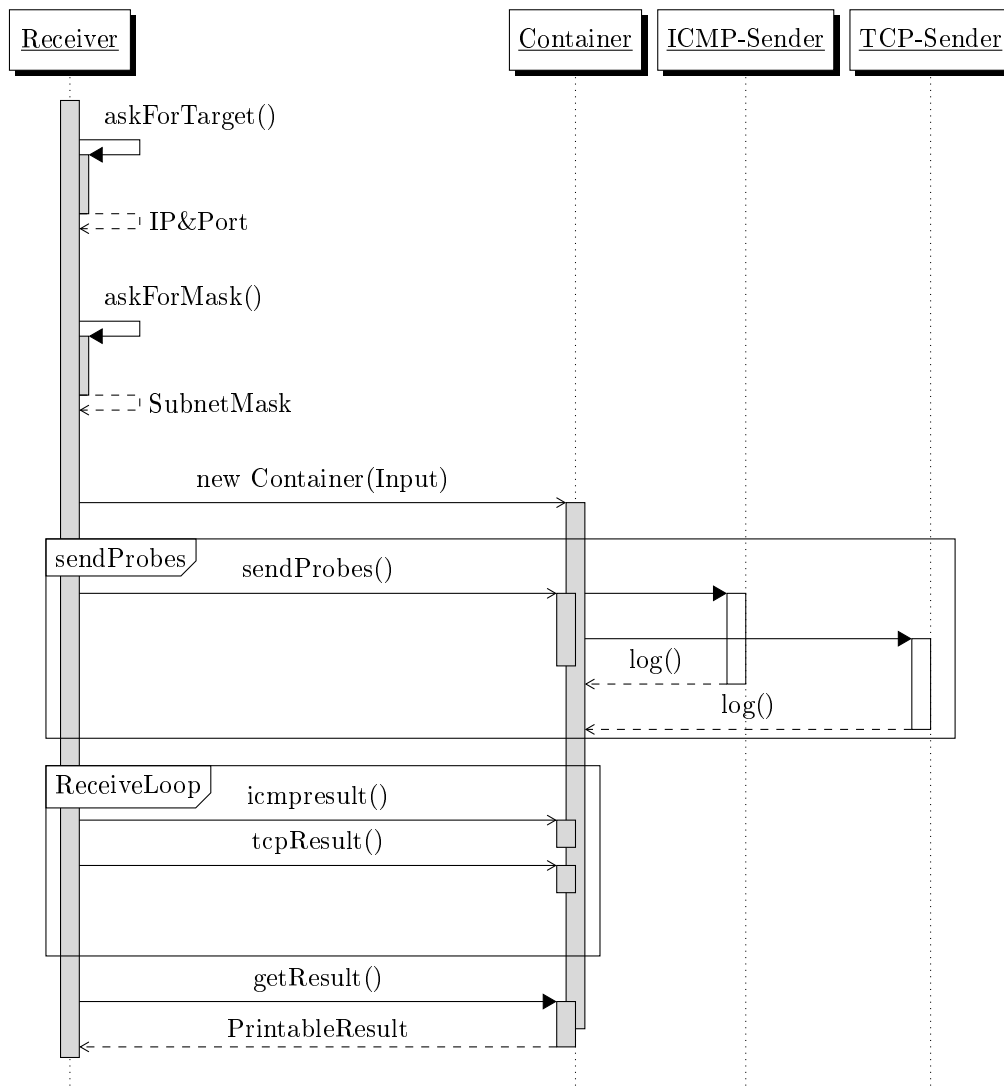
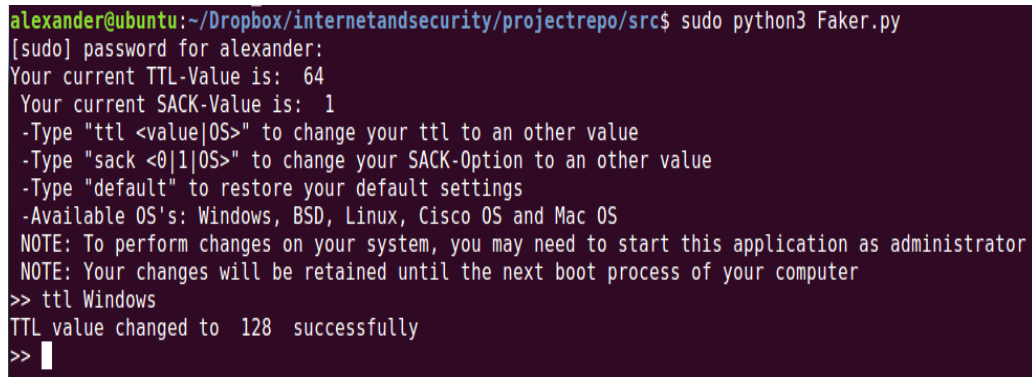


Abbildung 6: Das Sequenzdiagramm für den ersten Versuch. Dieses Konzept ist aber (bis auf kleine Änderungen) auch für die weiteren Versuche verwendet worden.

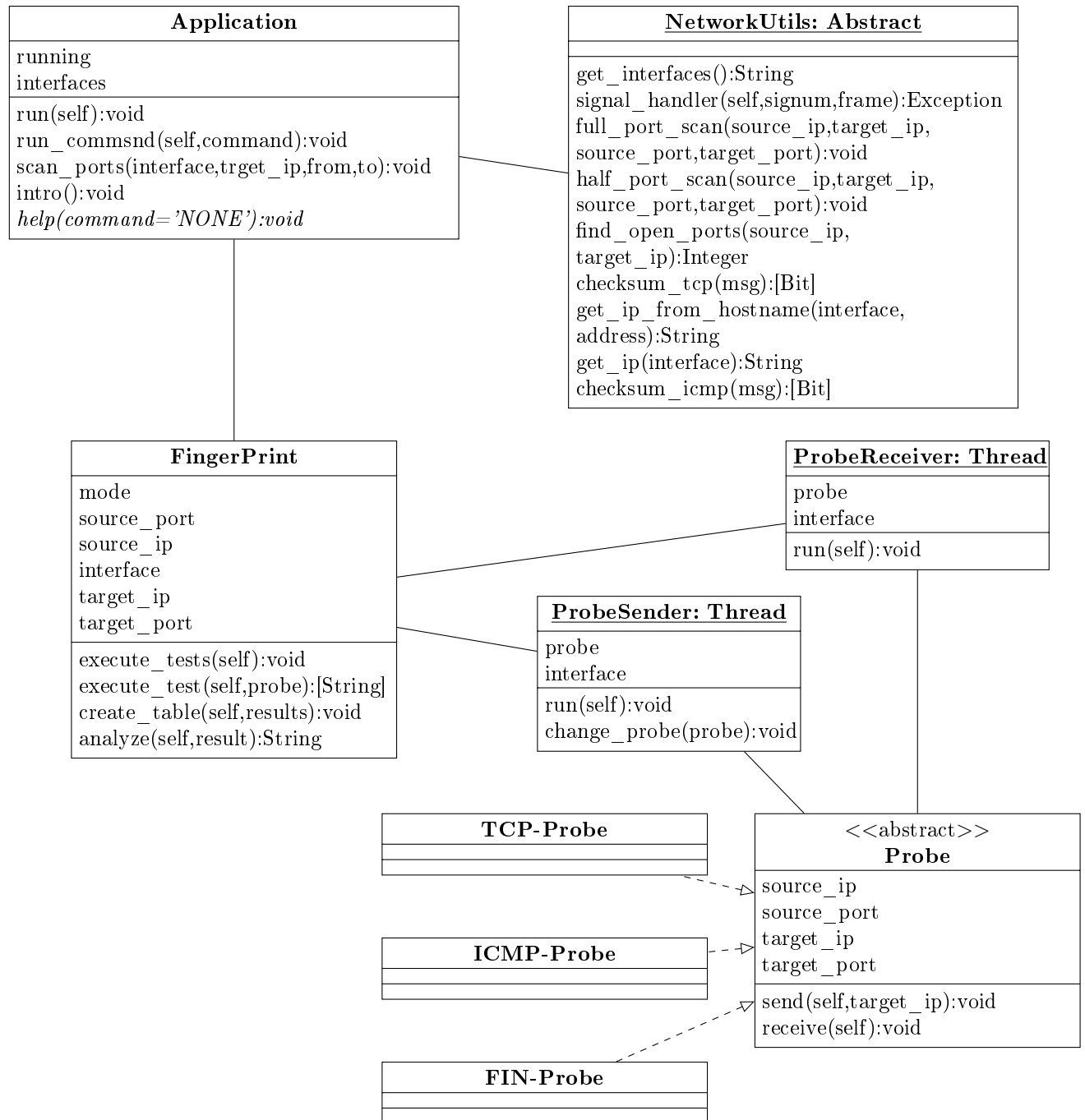
A.3 Screenshot des Fakers



```
alexander@ubuntu:~/Dropbox/internetandsecurity/projectrepo/src$ sudo python3 Faker.py
[sudo] password for alexander:
Your current TTL-Value is: 64
Your current SACK-Value is: 1
-Type "ttl <value|OS>" to change your ttl to an other value
-Type "sack <0|1|OS>" to change your SACK-Option to an other value
-Type "default" to restore your default settings
-Available OS's: Windows, BSD, Linux, Cisco OS and Mac OS
NOTE: To perform changes on your system, you may need to start this application as administrator
NOTE: Your changes will be retained until the next boot process of your computer
>> ttl Windows
TTL value changed to 128 successfully
>> █
```

Abbildung 7: Ein Screenshot von dem Fingerprint Faker. In der Abbildung wird die TTL auf den Windows Default-Wert gesetzt.

A.4 UML-Klassendiagramm von FingerBerry



A.5 Screenshot des Probesenders

```
class ICMPProbe(Probe):
    def send(self, target_ip):
        # print(target_ip)
        my_socket = socket.socket(socket.AF_INET, socket.SOCK_RAW, socket.IPPROTO_ICMP)
        try:
            host = socket.gethostbyname(target_ip)
        except socket.gaierror:
            print('error')

        icmp = struct.pack('>BBHHH', 8, 0, 0, 0, 0)
        icmp = struct.pack('>BBHHH', 8, 0, 0, NetworkUtils.checksum_icmp(icmp), 0)
        sent = my_socket.sendto(icmp, (target_ip, 1))
```

Abbildung 8: Ein Screenshot von der ICMPProbe Klasse. Die `send` Methode, ist zuständig fuer die Erstellung eines Raw Sockets und dem Versand des ICMP Packets.

A.6 Screenshot des Parsers

```
def receive(self):
    try:
        my_socket = socket.socket(socket.AF_PACKET, socket.SOCK_RAW, socket.ntohs(0x0800))
    except socket.error as msg:
        return msg

    has_ip_header = False

    my_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    my_socket.setsockopt(socket.SOL_SOCKET, socket.SO_BROADCAST, 1)
    my_socket.bind((self.interface, 0x0800))
    result = [0, 0]
    while True:
        packet = my_socket.recvfrom(100)
        # Unterscheiden zwischen den einzelnen Headern
        ethernet_header = packet[0][0:14]
        ip_header = packet[0][14:34]

        # Die Bytes in Strings umwandeln
        ethernet_information = struct.unpack('16s6s1h', ethernet_header) # 0: Destination_mac, 1: Source_mac, 2: Type
        ip_information = struct.unpack('11s1s2s2s1s1s1s1s2s4s4s', ip_header)
        # Informationen aus den Strings auslesen
        ip_address_source = socket.inet_ntoa(ip_information[9])
        # ip_address_destination = socket.inet_ntoa(ip_information[10])
        ip_ttl = ord(ip_information[6]) # ord(<str>)" wandelt den <str>-Wert in ein <int>-Wert um (nach der ASCII-Codierung)

        if int(ethernet_information[2]) == 2048 and ip_address_source == self.target_ip:
            # print("-----ICMP-TEST-----")
            # "\nFrom ", ip_address_source,
            # "\n-ttl: ", ip_ttl
            result = [ip_address_source, ip_ttl]

    return result
```

Abbildung 9: Ein Screenshot von der ICMP Klasse. Die `receive` Methode erstellt zuerst einen Raw Socket und empfängt Pakete auf diesen. Diese Pakete werden mit `struct.unpack` entpackt und anschliessend wird überprüft, ob überhaupt das gewünschte Paket empfangen wurde. Anschliessend werden die relevanten Informationen weitergegeben.