

http und https

- Verschlüsselung im Alltag

Eine Dokumentation des Seminarkursprojekts von Björn-Kristoffer Seibt und Florian Semen

Autor: Florian Semen

Inhaltsverzeichnis

1. Projekt.....	3
1.1 Idee	3
1.2 Aufbau.....	4
1.2.1 Hardware	4
1.2.2 Software	4
2. Verschlüsselung	5
2.1 https	5
2.2 Vorteile des SSL-Zertifikats	6
3. Probleme und Meilensteine.....	6
3.1 Zwei Websites auf einem ESP32	6
3.2 Linux	6
3.3 Zertifikatgeneration	7
3.4 WIFI Adapter & Wireshark.....	7
4. Code Übersicht.....	8
5. Durchführung.....	13
5.1 Vorbereitung	13
5.2 Ausführung.....	14
6. Ergebnis.....	15
7. Quellen.....	16

1. Projekt

1.1 Idee

Wann sind wir heute nicht mit dem Internet verbunden? Wo wir früher nur in unseren eigenen vier Wänden die Vorzüge eines W-Lan Zugangs erfahren durften, sind wir heute doch zu jedem Zeitpunkt in der Nähe von mindestens einem W-Lan Netzwerk. Von Starbucks bis Deutsche Bahn, alle versuchen sie im Zuge der Digitalisierung ihren Kunden den konstanten unbegrenzten Zugang zum Internet zu gewähren. Diese offenen Netzwerke sind dabei nicht nur für jede Person in Reichweite zugänglich, sondern mangeln außerdem jegliche Verschlüsselung.

Aber wieviel Sicherheit sind wir bereit für diesen Luxus aufzugeben? Und wie sehr kann diese Kombination aus öffentlicher Zugänglichkeit und fehlender Verschlüsselung uns nun wirklich schaden?

Mit dieser Leitfrage als Grundlage haben wir unser Projekt konzipiert. Um eine direkte Gegenüberstellung einer Verschlüsselten und Nicht-Verschlüsselten Webseite zu ermöglichen, sollten beide sonst identischen Webseiten auf dem gleichen ESP32 gehostet werden. So wäre es für uns möglich, den Verkehr eines möglichen Opfers innerhalb eines ungesicherten, öffentlichen Netzwerkes mit einer Verschlüsselten und Nicht-Verschlüsselten Seite zu simulieren.

Die über diese kabellose Verbindung gesendeten Pakete sollen derweil mit dem Programm Wireshark abgefangen werden und anschließend auf Unterschiede verglichen werden. Somit hätten wir nicht nur die Möglichkeit einer tiefgründigen, forensischen Analyse der gesendeten Datenpakete, sondern auch ein interessantes Projekt für unsere Seminararbeit.

1.2 Aufbau

1.2.1 Hardware

Die Kombination aus dem recht simplen Aufbau des Projekts und der verhältnismäßig komplexen Software ist sicher auch einer der „Reize“ die unser Projekt ausmachen. Denn so erhöht sich die Reproduzierbarkeit des Projektes enorm, da der Code sich unabhängig von seiner Komplexität, sehr einfach auf den eigenen ESP32 überspielen lässt.

Um unser Projekt nachzubauen, bedarf es nämlich nur der folgenden Dinge:

- Einen ESP32
- Einen Laptop mit Linux & ein „Opfer“
- Einen Netzwerkadapter, der den sog. „Monitor Mode“ unterstützt (Bsp. CSL 300 Mbit/s WLAN-Stick)
- Einen offenen Access Point (Handy, weiterer ESP, ...)

Auch der Zusammenbau des Projekts gestaltet sich aufgrund der sehr reduzierten Anzahl der Einzelteile als unaufwendig.

Für den eigentlichen Versuchsaufbau werden schließlich W-Lan Adapter und ESP32 mit dem Laptop verbunden. Der ESP32 kann nach einmaliger Programmierung auch durch jede beliebige Stromquelle mit Strom versorgt werden.

1.2.2 Software

Die Software-Komponente des Projekts beinhaltet sowohl den Code des ESP32, sowie die auf dem Linux Rechner installierten Tools zur Überwachung des Netzwerk-Verkehrs.

So bedarf es für den Nachbau des Projekts folgender Software:

- Ein Linux Betriebssystem
- Die Arduino IDE + Sketch file
- Wireshark

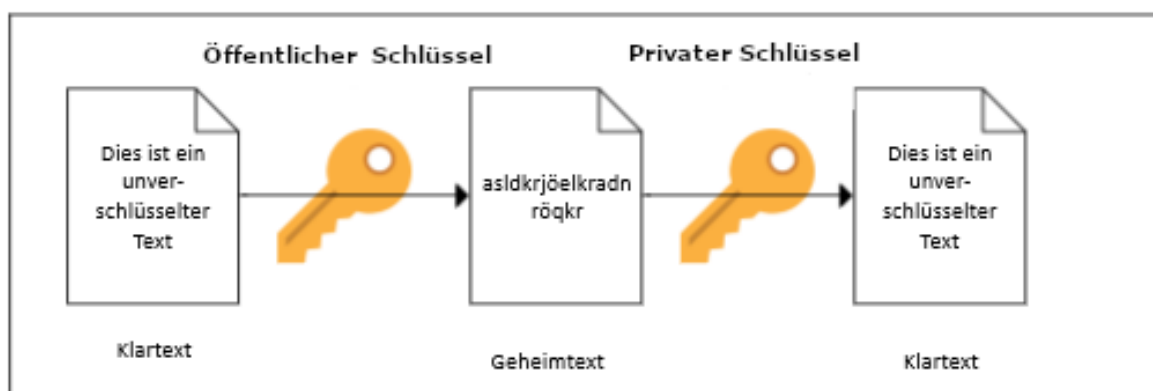
Aufgrund ihrer besseren Kompatibilität wurde für unser Projekt die Linux Distribution „Kali Linux“ als Betriebssystem für den Rechner gewählt. Dieses vereinfacht nicht nur die Modifizierung des W-Lan Adapters, sondern ermöglicht auch die unkomplizierte Integration mit Wireshark selbst.

Zunächst muss der Programmcode auf den ESP32 gespielt, und dieser gestartet werden. Anschließend kann auf dem Laptop das Programm „Wireshark“ gestartet, und die im Netzwerk gesendeten Pakete abgefangen werden.

2. Verschlüsselung

2.1 https

https ist kurz für Hyper Text Transport Protocol **Secure**. Dabei handelt es sich um ein Asymmetrisches Verschlüsselung Verfahren, durch welches ein sicherer Tunnel zwischen Client und Server hergestellt wird. Dieses besteht aus einem Öffentlichen und Privaten Schlüssel. Der Öffentliche Schlüssel dient hierbei ausschließlich der Verschlüsselung der Daten und ist für jeden zugänglich. Die Nachrichten können allerdings nur mit dem Privaten Schlüssel entschlüsselt werden, weshalb der Private Schlüssel geheim gehalten werden muss.



Nun wollen wir eine https Verbindung etablieren. Zunächst sendet der Client dem Server eine sogenannte „ClientHello“ Nachricht. Diese enthält alle nötigen Informationen (darunter der Öffentliche Schlüssel des Clients) für den Server, um eine SSL Verbindung zu dem Client aufzubauen. Der Server antwortet schließlich mit einem „ServerHello“ in der Beispielsweise die Version der Verschlüsselung festgelegt wird.

⇒ **Eine Sichere Verbindung wurde erstellt.**

Nach dem Erstellen einer sicheren Verbindung, muss der Server nun seine Identität preisgeben. Das bewerkstelligt er durch die Bereitstellung seines **SSL-Zertifikats**. Dieses lässt sich gut als „Personalausweis des Servers“ vorstellen. Dieses lässt sich durch verschiedene kryptographische Signaturen nicht kopieren und enthält Informationen über Besitzer, Domain, Signatur, usw. . Der Client prüft nun das Zertifikat auf seine Echtheit. .

⇒ **Die Identität des Servers wurde verifiziert.**

Nach der Überprüfung des Zertifikates wird nun ein Symmetrischer Schlüssel zwischen Server und Client ausgetauscht. Der folgende Austausch wird nun also auf der Grundlage einer Symmetrischen Verschlüsselung basieren.

⇒ **Ein verifizierter, sicherer Austausch wird gewährleistet.**

2.2 Vorteile des SSL-Zertifikats

Aber für was brauchen wir denn überhaupt das Zertifikat für unsere Verschlüsselte Webseite? Die Verwendung des SSL-Zertifikats birgt die folgenden Vorteile:

1. **Verschlüsselung:** Einer der offensichtlichen Gründe für die Verwendung eines Zertifikates ist die Verschlüsselung der Nutzerdaten. Benutzername, Passwort und Kreditkartendaten sind gegen die Angriffe von Hackern gesichert.
2. **Verifizierung:** Ein SSL-Zertifikat prüft außerdem auch die Echtheit der Website. Dieses negiert damit die Möglichkeit für Angreifer, eine bereits bestehende Website zu klonen und sich als vertrauenswürdiges Unternehmen auszugeben.
3. **Vertrauen:** Das durch die Verwendung gewonnene Vertrauen ist ebenfalls eins der Hauptgründe für die Verwendung von SSL. Das steigert das Gefühl von Sicherheit bei den Nutzern, insbesondere bei Zahlungsinformationen und ähnlichen sensiblen Daten.

3. Probleme und Meilensteine

3.1 Zwei Websites auf einem ESP32

Die erste große Hürde die es für unser Projekt zu überwinden galt, war der simultane Betrieb von zwei Webseiten zur gleichen Zeit auf einem ESP32. Dabei konnten wir auch nicht einfach eine einzelne Website mit mehreren Seiten zurückgreifen, da eine der Webseiten für den direkten Vergleich auf das http, die andere auf das https Protokoll zurückgreifen musste.

Für die Lösung des Problems konnten wir auf das ausgiebige Wissen des GitHub Users fhessel [<https://github.com/fhessel>] zurückgreifen. Seine Lösung bestand darin, beide Webseiten auf verschiedenen Nodes und damit auch verschiedenen Ports laufen zu lassen. So ermöglicht es den Zugriff auf die jeweilige Website durch eine Spezifizierung des Ports.

3.2 Linux

Das Linux Betriebssystem, und dabei insbesondere die für Penetrationstest sowie digitale Forensik erstellte Distribution „Kali Linux“ der IT-Security Firma Offensive Security, spielten für den Erfolg unseres Projekts eine wichtige Rolle, und war aufgrund einiger Unterschiede zum gewohnten Windows 10 eine Hürde, die es zunächst zu bezwingen galt. Doch um mögliche Fehlerquellen für unsere Forensik Software ausschließen zu können, wurde Linux als Betriebssystem auf dem Hauptrechner installiert. Nicht nur wurden die meisten dieser

Programme für Linux konzipiert und erstellt, auch die präzisere und tiefgründigere Manipulation der einzelnen Komponenten machten das Betriebssystem zur ersten Wahl

Und auch wenn insbesondere die häufige Integration der Konsole für einfache sowie komplexe Aufgaben zunächst einschüchtert, ist Linux und insbesondere die Community dahinter stark gewachsen, und ermöglichen in Kombination mit den verschiedensten Erklärungen im Internet ein schnelles Verständnis der Linux Features.

3.3 Zertifikatgeneration

Für die Erstellung einer https Website brauchten wir nun also ein SSL-Zertifikat. Den ersten Schritt in die richtige Richtung hatten wir bereits mit der Wahl unseres Betriebssystems hinter uns, da der Prozess des Zertifikatgeneration auf Linux deutlich unkomplizierter wurde.

Für ein Verständnis dieser sehr theorielastigen Thematik, war eine lange und weit gefächerte Recherche der Schlüssel zum Erfolg. Verschiedenste Webseiten und Blogs leisten hervorragende Arbeit, derartige Themen für Interessenten ohne tiefgreifendes Vorwissen greifbar zu machen. Wurde das Grundkonzept erst einmal verstanden, besteht auch weiter noch die Option, sich über die technischen Aspekte der Thematik zu informieren.

3.4 WIFI Adapter & Wireshark

Die Masse der Empfangen Pakete hat uns trotz zu dem Zeitpunkt bereits vorhandenem Hintergrundwissen überrascht, und dabei die Suche und Auswertung der von uns gesendeten Pakete verkompliziert. Denn ungefiltert wurden wir selbst in einem etwas abgelegenen Ort wie Althütte von einer gigantischen Menge an Datenpaketen überrumpelt, die die Implementierung eines Paket-Filters unumgänglich machten.

Und auch das ausführliche Interface des Programms Wireshark wirkt bei erstmaliger Nutzung des Programms überrumpelnd. Die vielen fortgeschrittenen Einstellungsmöglichkeiten tragen auf den ersten Blick ebenfalls nicht dazu bei, sich in der Paket-Masse zurecht zu finden. Doch auch hier war die Dokumentation der Entwickler ebenso hilfreich wie die Unmengen an YouTube Videos, die für jeden erdenklichen Anwendungszweck eine Schritt-für-Schritt Anleitung bieten

4. Code Übersicht

```
/**
 * Seminarprojekt Https/Https
 * Autoren: Björn-Kristoffer Seibt & Florian Semen
 * Fertigstellung: [15.07.2021]
 * Beschreibung: Das Programm wurde für die Verwendung mit einem ESP32 konzipiert.
 * Parallel werden eine verschlüsselte und unverschlüsselte Webseite mit einem Login Form aufgespannt.
 * Ursprüngliches Script von Github User fhessel
 * Modifiziert von Björn-Kristoffer Seibt
 *
 * Wichtig:
 * Damit das Script funktioniert, muss man:
 * 1) W-Lan SSID und Passwort eintragen
 * 2) Sicher stellen, dass ein Zertifikat vorhanden ist. Erklärung hierfür finden sich in den Bibliotheken des Scripts unter Extras.
 *
 * Dieses Script installiert einen Http Server auf Port 80 und einen Https Server auf Port 443 des ESP32.
 *
 * Für genauere Informationen kann die in der Dokumentation hinterlegte Quelle des Programms aufgerufen werden.
 */

// TODO: WiFi konfigurieren
#define WIFI_SSID "TestOffen"
#define WIFI_PSK "" // bei offenem Wlan String frei lassen

// Zertifikate einbinden (siehe oben)
#include "cert.h"
#include "private_key.h"

// Wir benutzen WiFi
#include <WiFi.h>

//Wichtige weitere Bibliotheken
//Hinweis: Hier werden http und https verwendet
#include <HTTPServer.h>
#include <HTTPServer.h>
#include <SSLCert.h>
#include <HTTPRequest.h>
#include <HTTPResponse.h>
#include <ResourceNode.h>
```



```

String IP;

// Der Https Server wird unter einem anderen "namespace" adressiert.
using namespace httpserver;

// Ein SSL Zertifikat wird mit den Dateien von oben erstellt
SSLCert cert = SSLCert(
    example_crt_DER, example_crt_DER_len,
    example_key_DER, example_key_DER_len
);

// Zuerst wird der Https-Server mit dem Zertifikat erstellt
HTTPSServer secureServer = HTTPSServer(&cert);

// Nun wird ein unverschlüsselter Http-Server erstellt
HTTPServer insecureServer = HTTPServer();

// Es werden Handler-Funktionen für alle URLs des Webservers erstellt
void handleRoot(HTTPRequest * req, HTTPResponse * res);
void handle404(HTTPRequest * req, HTTPResponse * res);
void handlelogin(HTTPRequest * req, HTTPResponse * res);
void handlelogins(HTTPRequest * req, HTTPResponse * res);
void handlecheckout(HTTPRequest * req, HTTPResponse * res);
void handlecheckouts(HTTPRequest * req, HTTPResponse * res);

void setup() {
    // Für die Ausgabe im Seriellen Monitor
    Serial.begin(115200);

    // Mit dem WiFi verbinden
    Serial.println("Setting up WiFi");
    WiFi.begin(WIFI_SSID, WIFI_PSK);
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    Serial.print("Connected. IP=");
    Serial.println(WiFi.localIP());
    IP = WiFi.localIP().toString();

    // Für jede Ressource muss eine "ResourceNode" erstellt werden
    // Die "ResourceNode" verbinden URL und HTTP Methoden zu einer Handler-
    Funktion
    ResourceNode * nodeRoot = new ResourceNode("/", "GET", &handleRoot);
    ResourceNode * node404 = new ResourceNode("", "GET", &handle404);
    ResourceNode * nodelogin = new ResourceNode("/login", "GET", &handlelogin);
    ResourceNode * nodelogins = new ResourceNode("/logins", "GET", &handlelogin
s);

```

```

    ResourceNode * nodecheckout = new ResourceNode("/checkout", "GET", &handlec
checkout);

    // Die Notes werden registriert und es wird definiert ob sie auf dem Http- o
der Https-Server laufen
    secureServer.registerNode(nodeRoot);
    insecureServer.registerNode(nodeRoot);
    insecureServer.registerNode(nodelogin);
    secureServer.registerNode(nodelogins);
    insecureServer.registerNode(nodecheckout);
    secureServer.registerNode(nodecheckout);
    secureServer.setDefaultNode(node404);
    insecureServer.setDefaultNode(node404);

    Serial.println("Starting HTTPS server...");
    secureServer.start();
    Serial.println("Starting HTTP server...");
    insecureServer.start();
    if (secureServer.isRunning() && insecureServer.isRunning()) {
        Serial.println("Servers ready.");
    }
}

void loop() {
    // Damit die Server dauerhaft laufen müssen beide Loop-
Funktionen benutzt werden.
    secureServer.loop();
    insecureServer.loop();

    delay(1);
}

//Die einzelnen Handler-Funktionen werden definiert.

void handleRoot(HTTPRequest * req, HTTPResponse * res) {
    res->setHeader("Content-Type", "text/html");

    res->println("<!DOCTYPE html>\n");
    res->println("<html>\n");
    res->println("<title>Seminararbeit IoT</title>\n");
    res->println("<meta charset=\"UTF-8\">\n");
    res->println("<meta name=\"viewport\" content=\"width=device-width, initial-
scale=1\">\n");
    res-
>println("<link rel=\"stylesheet\" href=\"https://www.w3schools.com/w3css/4/w3
.css\">\n");

```

```

res->println("<body class=\"w3-content\" style=\"max-width:2560px\">\n");
res->println("\n");
res->println("<!-- First Grid: Logo & About -->\n");
res->println("<div class=\"w3-row\">\n");
res->println("  <div class=\"w3-half w3-black w3-container w3-
center\" style=\"height:1440px\">\n");
res->println("    <div class=\"w3-padding-64\">\n");
res->println("      <h1>Seminararbeit Http/Https</h1>\n");
res->println("    </div>\n");
res->println("    <div class=\"w3-padding-64\">\n");
if (req->isSecure()) {
  res-
>println("<p>Du befindest dich auf dem <strong>HTTPS</strong> Server.</p>");
} else {
  res-
>println("<p>Du befindest dich auf dem <strong>HTTP</strong> Server.</p>");
}
  res->print("<p>Dein Server läuft schon ");
res->print((int)(millis()/1000), DEC);
res->println(" Sekunden.</p>");
res->println("<meta http-equiv=\"refresh\" content=\"10\" >");
res->println("  <a href=\"login\" class=\"w3-button w3-black w3-
block w3-hover-blue-grey w3-padding-16\">Zum Login</a>\n");
res->println("  </div>\n");
res->println(" </div>\n");
res->println("  <div class=\"w3-half w3-blue-grey w3-
container\" style=\"height:1440px\">\n");
res->println("    <div class=\"w3-padding-64 w3-center\">\n");
res->println("      <h1>Ein Seminarprojekt von:</h1>\n");
res->println("      <h1>Björn-Kristoffer Seibt</h1>\n");
res->println("      <h1>Florian Semen</h1>\n");
res->println("      <div class=\"w3-left-align w3-padding-large\">\n");
res-
>println("        <p>Eine direkte Gegenüberstellung einer Verschlüsselten und
Nicht-
Verschlüsselten Webseite mit der Aufgabe den Verkehr eines möglichen Opfers in
nerhalb eines ungesicherten, öffentlichen Netzwerkes mit einer Verschlüsselten
und Nicht-Verschlüsselten Seite zu simulieren.</p>\n");
res->println("      </div>\n");
res->println("    </div>\n");
res->println("  </div>\n");
res->println("</div>");
}

void handle404(HTTPRequest * req, HTTPResponse * res) {
  req->discardRequestBody();
  res->setStatusCode(404);
  res->setStatusText("Not Found");
}

```

```
res->setHeader("Content-Type", "text/html");
res->println("<!DOCTYPE html>");
res->println("<html>");
res->println("<head><title>Not Found</title></head>");
res-
>println("<body><h1>404 Not Found</h1><p>The requested resource was not found
on this server.</p></body>");
res->println("</html>");
for(int i=0; i<1 ; i++){
res->println("<meta http-equiv=\"refresh\" content=\"0.01\" >");

}

}
```

5. Durchführung

5.1 Vorbereitung

Für die Durchführung des in dieser Dokumentation beschriebenen Projekts werden die in Absatz „1.2 Aufbau“ bereits erwähnten Komponenten benötigt.

Für die eigentliche Ausführung müssen zunächst einige Vorbereitungen getroffen werden.

1. Anschließen

Der ESP32 muss für das flashen der Software an den Linux-Hauptrechner angeschlossen werden. Auch der W-Lan Adapter muss Platz in einem der USB-Ports des Hauptrechners finden. Zu guter Letzt muss auch der Access Point mit einer Stromquelle verbunden werden.

2. Software Start

Ist der ESP32 nicht bereits aus vorherigen Versuchen mit der entsprechenden Software beschrieben, so muss das von uns geschriebene Programm mithilfe der Arduino IDE auf den ESP geflashed werden. Daraufhin wird das Programm ausgeführt und der ESP verbindet sich mit dem offenen Netzwerk und spannt beide Webseiten auf, deren Adressen nun aus dem seriellen Monitor abgelesen werden können. Anschließend muss der Computer des „Opfers“ auf diese Adressen navigiert werden.

3. W-Lan Adapter

Um gesendete Pakete abfangen zu können, müssen wir unseren Linux-Rechner „monitorfähig“ machen. Dazu muss der verbundene W-Lan Adapter mit den folgenden Befehlen in der Linux Konsole in den sogenannten „Monitor mode“ geschaltet werden:

```
$ iwconfig  
$ sudo airmon-ng check kill  
$ sudo airmon-ng start wlan1  
$ sudo airodump-ng wlan1mon --channel 1
```

(**!Wichtig:** einige der Befehle benötigen root-Berechtigungen. Das ‚\$‘ Symbol kennzeichnet den Zeilenanfang und darf nicht in die Konsole eingetragen werden)

4. Wireshark



Wurde der Adapter erfolgreich in den „Monitor mode“ geschaltet, kann anschließend das Programm Wireshark auf dem Linux System gestartet werden. Für die zu überwachende Schnittstelle muss nun die in Schritt 3 erstellte Schnittstelle ausgewählt werden. Damit sind die Vorbereitungen für das Projekt abgeschlossen.

5.2 Ausführung

Nun kann der Login Vorgang für beide Webseiten simuliert werden. Sowohl auf der http, als auch der https Seite können über das Login Form beliebige Benutzernamen und Passwörter eingegeben werden, wie man es Beispielsweise bei einem vorhandenen Account auf Amazon machen würde.

Alle Pakete, die zwischen dem Rechner des „Opfers“ und dem ESP32 ausgetauscht wurden, haben wir mithilfe unseres Wireshark Setups abgefangen und protokolliert. Wurden beide Logins einmal angesprochen, kann das Monitoring innerhalb der Wireshark Software gestoppt werden. Schnell wird man feststellen, dass nicht nur die Pakete des Zielrechners, sondern auch alle anderen Pakete der Umgebung von uns abgefangen wurden.

In der Suchleiste des Programms können wir nun Pakete mit von uns festgelegten Kriterien filtern. Hier gehen wir nur auf diesen Fall ein, für weitere Informationen bezüglich der Filtermöglichkeiten ist die Dokumentation der Entwickler sehr aufschlussreich. Die für uns relevanten Pakete können mit den folgenden Filtereinstellungen ausgemacht werden:

http: `http.request.method == POST`  Ein Paket anklicken  Option „folgen - TCP“ auswählen

https: TLS

Also haben wir jetzt den Verkehr beider Seiten nicht nur abgefangen, sondern zusätzlich diese noch aus einer Vielzahl an Paketen herausgefiltert. Die Inhalte der beiden Pakete können nun einfach verglichen werden.

Die Login-Daten der http-Webseite sind unverschlüsselt gesendet worden, und Nutzerdaten können daher auch als Klartext abgefangen werden.

Die https-Webseite und ihre verschlüsselten Pakete sind zwar ebenfalls von uns abgefangen worden, deren Inhalt allerdings ist für uns aufgrund der Verschlüsselung nicht ersichtlich.

6. Ergebnis

Bevor wir die Ergebnisse unserer spannenden und rechercheintensiven Seminararbeit vorstellen, müssen wir zunächst noch einmal den Grundgedanken / die Leitfrage des Projekts aufgreifen.

Wie gefährlich kann ein Mangel an Verschlüsselung auf Webseiten gefährlich für uns sein?

Mit unserem Projekt konnten wir die Probleme und Gefahren dieser doch recht alltäglichen Situation aufzeigen. Diese bieten besonders im Umgang mit sensiblen Daten ein erhebliches Sicherheitsleck. Doch diese Problematik war bereits kurz nach der Entstehung des Internets präsent.

Heute setzten alle seriösen Seitenbetreiber auf die Verschlüsselung, um sowohl Verschlüsselung sensibler Nutzerdaten, als auch eine Verifikation der Seite zu gewährleisten. Auch weißt heutzutage jeder Browser durch ein Schloss in der Adresszeile auf eine sichere Verbindung mit der gewünschten Seite hin, und informiert bei einem Mangel derselben darüber, keine sensiblen Daten in die Webseite einzutragen.

Problematisch für die Nutzung öffentlicher Netzwerke ist heutzutage nur die Verwendung von Apps auf dem Smartphone, bei denen anders als im Browser eine fehlende Verschlüsselung nicht sofort ersichtlich wird.

Sobald man sich über die „Ecken und Kanten“ öffentlicher Netzwerke im Klaren ist und über die Datenerfassung derselben hinwegsehen kann, bieten diese fast überall einen stabilen Zugang in das World Wide Web – eine Entwicklung, der wie uns nicht so einfach entziehen können.

7. Quellen

<https://love2dev.com/blog/how-https-works/> [15.07.21]

<http://www.steves-internet-guide.com/ssl-certificates-explained/> [15.07.21]

https://github.com/fhessel/esp32_https_server [15.07.21]

https://github.com/fhessel/esp32_https_server/blob/master/examples/HTTPS-and-HTTP/HTTPS-and-HTTP.ino [15.07.21]

<https://blog.mozilla.org/security/2012/11/01/preloading-hsts/> [15.07.21]

<https://support.mozilla.org/en-US/kb/https-only-prefs> [15.07.21]

https://httpd.apache.org/docs/trunk/ssl/ssl_howto.html#ocspstapling [15.07.21]

<https://web.archive.org/web/20070202031218/http://www.cabforum.org/forum.html> [15.07.21]

<https://www.howtogeek.com/181767/htg-explains-what-is-https-and-why-should-i-care/> [15.07.21]

<https://www.eff.org/pages/tor-and-https> [15.07.21]

<https://datatracker.ietf.org/doc/html/rfc3546#section-3.1> [15.07.21]

https://www.smartlife-online.de/blog/beitraege/ssl_erklaerung/ [15.07.21]

<https://robertheaton.com/2014/03/27/how-does-https-actually-work/> [15.07.21]

<https://www.advantageservices.net/blog/40/Why-Websites-and-Servers-Need-SSL-Certificates> [15.07.21]

<https://www.clickssl.net/blog/what-why-and-how-of-ssl-certificate> [15.07.21]

<https://letsencrypt.org/how-it-works/> [15.07.21]

<https://www.sslforfree.com/> [15.07.21]

<https://technicalwall.com/blogging/free-ssl-certificate/#tab-con-9> [15.07.21]

<https://youtu.be/ZYdNTZ174-A> [15.07.21]

<https://stackoverflow.com/questions/46178398/can-a-hacker-sniff-http-packets-transmitted-via-wifi> [15.07.21]

<https://superuser.com/questions/1576227/sniffing-packets-over-a-wifi-network-with-or-without-monitor-mode-using-wiresha> [15.07.21]

<https://superuser.com/questions/503907/sniffing-and-logging-http-traffic-through-my-wireless-router> [15.07.21]

<https://stackoverflow.com/questions/29884654/button-that-refreshes-the-page-on-click>
[15.07.21]

<https://stackoverflow.com/questions/33483960/how-to-reload-a-page-on-submit>
[15.07.21]