ZESZYTY NAUKOWE POLITECHNIKI RZESZOWSKIEJ – ELEKTROTECHNIKA
RZESZOW UNIVERSITY OF TECHNOLOGY JOURNAL OF ELECTRICAL ENGINEERING
39 (2022), s. 51-70

Bartłomiej WILCZKIEWICZ[1]

BEZPIECZEŃSTWO W BEZPRZEWODOWYCH SIECIACH SENSOROWYCH NA PRZYKŁADZIE WYBRANYCH ROZWIĄZAŃ

W ramach artykułu przedstawiono podstawowe informacje na temat Internetu Rzeczy oraz bezprzewodowych sieci czujników. Następnie zaprojektowano prototypowy system IoT, opisano ogólnie jego działanie oraz przedstawiono dostępne zabezpieczenia tego typu systemów. Na przykładzie wybranych rozwiązań sprzętowo programowych przeanalizowano możliwości i narzędzia służące do zdalnego zarządzania, integracji oraz organizacji wymiany danych. Na tej podstawie przedstawiono opis wybranych, przykładowych rozwiązań praktycznych i zaproponowano metodykę selekcji, projektowania i programowania tego typu systemów. Zaprezentowano schemat połączeń dla układu pojedynczego węzła sieciowego, omówiono pokrótce usługi uruchomione na serwerze oraz scharakteryzowano ogólnie protokół MQTT (wymiana danych za pośrednictwem sieci komputerowej). W opracowanym systemie zaimplementowano podstawowe zabezpieczenia. Całość komunikacji odbywała się bez szyfrowania informacji (dane przesyłane jawnym tekstem). Przedstawiono również wyniki podstawowych badań wykonanych z użyciem programu Wireshark oraz zwrócono uwagę na potencjalne formy ochrony systemów IoT. Głównym celem poniższego artykułu było pokazanie jak ogromne znaczenie ma bezpieczeństwo transmisji danych w tego typu systemach.

Słowa kluczowe: internet rzeczy, sensory, moduły bezprzewodowe, node-red, mqtt, raspberry pi

[1] Autor do korespondencji: Bartłomiej Wilczkiewicz, Politechnika Rzeszowska, Katedra Systemów Elektronicznych i Telekomunikacyjnych, W. Pola 2, 35-959 Rzeszów, tel. 178651239, b.wilczkiewi@prz.edu.pl

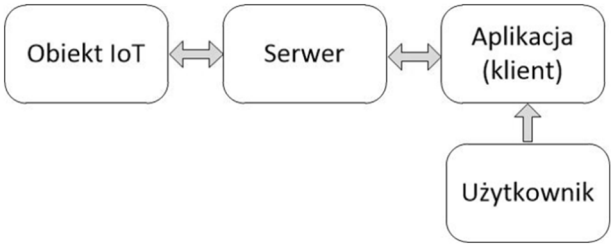
1. Wprowadzenie

Internet Rzeczy jest to koncepcja sieci obiektów, które posiadają elektroniczne układy odpowiadające za odczyt danych z podłączonych do nich czujników (temperatura, wilgotność itd.) i/lub sterowanie elementami wyjściowymi (oświetlenie, przekaźniki itd.) oraz wymianę danych z serwerem za pośrednictwem sieci teleinformatycznej (Internet lub mniejszej obszarowo sieci Intranet) [2]. W świecie IoT komunikacja między jego elementami występuje na różnych poziomach. Komunikacja z różnego typu sensorami odbywa się najczęściej z użyciem protokołów komunikacji: 1-Wire, SPI, I2C, UART. Jeśli chodzi o protokoły wyższych warstw modelu TCP/IP do często stosowanych protokołów najwyższej warstwy modelu TCP/IP zaliczyć można: MQTT, HTTP, Modbus [1]. Poruszając tematykę IoT warto byłoby też wspomnieć o WSN (ang. *Wireless Sensor Network*), czyli bezprzewodowej sieci sensorów.

Bezprzewodowe sieci czujników są to sieci, które składają się z niezależnych, rozproszonych urządzeń (modułów). Każdy z takich węzłów łączy w sobie urządzenie pomiarowe, układ mikroprocesora z pamięcią, zasilanie oraz zawiera dodatkowo nadajnik/odbiornik radiowy (przesył danych własnym protokołem transmisji, za pomocą WLAN itp.). Węzeł za pomocą czujnika monitorującego, może mierzyć różne wielkości fizyczne (temperatura, ciśnienie, wilgotność itd.) w ustalonych miejscach. Pierwotnie tego typu rozwiązania były stosowane i rozwijane wyłącznie w zastosowaniach militarnych, zważywszy jednak na ich funkcjonalność, znalazły także zastosowanie w przemyśle i różnych dziedzinach codziennego życia. Przykładowe zastosowania tego typu mikrosystemów są następujące [3]:

- branża medyczna,
- wojsko,
- kontrola ruchu drogowego,
- inteligentne budynki,
- automatyzacja/kontrola przemysłu,
- wykrywanie zagrożeń,
- edukacja.

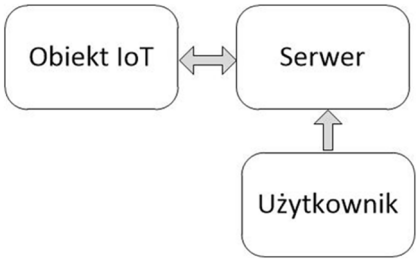
Istnieją różne rozwiązania dotyczące wymiany danych i interakcji w systemach IoT/WSN. Sterowanie obiektem czy też odczyt danych odbywa się najczęściej poprzez aplikację kliencką zainstalowaną na komputerze bądź smartfonie. W architekturze klient-serwer aplikacja kliencka łączy się podczas jej uruchomienia przez użytkownika bezpośrednio z serwerem (rys. 1.).



Rys. 1. Schemat blokowy przepływu danych (opcja I)

Fig. 1. Data flow block diagram (option I)

Zdarza się też tak, że serwer sam w sobie posiada graficzny interfejs użytkownika i można się z nim połączyć podając odpowiedni adres IP, bądź adres mnemoniczny w przeglądarce internetowej. Nie jest wymagana do tego osobna aplikacja kliencka (rys. 2.).



Rys. 2. Schemat blokowy przepływu danych (opcja II)

Fig. 2. Data flow block diagram (option II)

Innym podejściem może byś sytuacja, w której obiekt IoT ma uruchomioną usługę serwera WWW i nie potrzebuje żadnych serwerów pośredniczących. Wtedy (o ile dany obiekt posiada interfejs graficzny) po podaniu odpowiedniego adresu mnemonicznego (bądź adresu IP) w przeglądarce internetowej możliwy jest dostęp do obsługi elementu (rys. 3.).

Rys. 3. Schemat blokowy przepływu danych (opcja III)
Fig. 3. Data flow block diagram (option III)

Sposób interakcji z obiektem IoT przedstawiony na rysunku 3 może też odbywać się za pośrednictwem dedykowanej aplikacji (gdy np. nie ma uruchomionego interfejsu graficznego w danym module), która wymienia dane poprzez sieć Internet z serwerem uruchomionym bezpośrednio we wspomnianym wcześniej elemencie, za pośrednictwem zaimplementowanego protokołu obsługi transmisji.

Sercem systemów IoT są urządzenia elektroniczne, ale do ich działania potrzebne jest też odpowiednie oprogramowanie zarządzające wszystkimi procesami. Do programowania urządzeń systemów wbudowanych w obszarze IoT najczęściej używanym językiem programowania są języki C/C++. W przypadku programowania z użyciem bibliotek udostępnionych dla danego środowiska programistycznego i konkretnych mikrokontrolerów, przygotowanie oprogramowania znacznie się upraszcza i sprowadza do wywoływania zaprogramowanych w bibliotece funkcji. Urządzenia systemów wbudowanych to jedna część systemu IoT, drugą stanowią urządzenia zbierające i przetwarzające dane (komputery/urządzenia brzegowe). Do programowania urządzeń brzegowych (np. Raspberry Pi) w świecie IoT używa się często języków Python, Java, JavaScript, itp. W przypadku systemów operacyjnych urządzeń brzegowych opartych na Linuksie często korzysta się także ze skryptów w języku Bash. Oprócz wymienionych języków programowania w urządzeniach brzegowych często wykorzystuje się też liczne aplikacje sieciowe, które działają jako serwer na urządzeniu i umożliwiają zmianę jego ustawień.

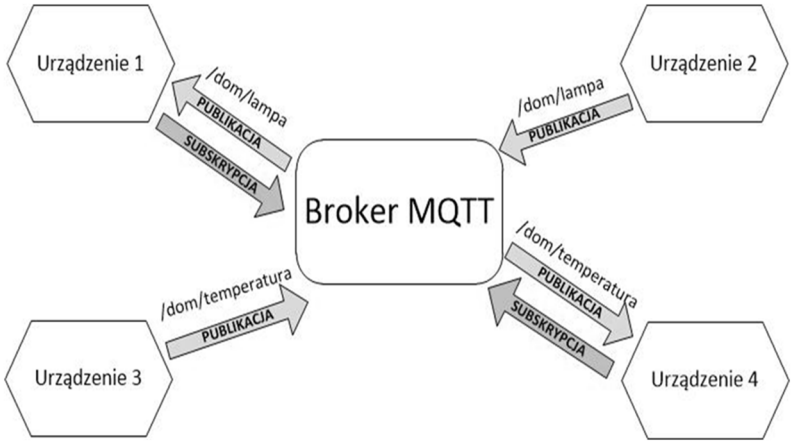
W dzisiejszych czasach powszechnym zainteresowaniem cieszą się graficzne środowiska programowania, które świetnie wizualizują tworzone na serwerze algorytmy sterujące przepływem danych. Istnieje szereg środowisk do programowania wizualnego. Do nich zaliczyć można chociażby LabView czy też rozwiązania bardziej wyspecjalizowane w kierunku IoT takie jak Domoticz i Node-Red, które są edytorami przepływu opartymi na przeglądarkach internetowych i uruchamiane jako usługa serwerowa na komputerze. We wszystkich

wspomnianych przypadkach programowanie odbywa się poprzez łączenie bloków funkcjonalnych. Każdy z nich wykonuje odpowiednie zadanie. Ponadto występują specjalne bloki odpowiadające za wizualizacje wszystkich procesów (wykresy, kontrolki, itd.), wspomagające szybkie opracowywanie graficznego interfejsu użytkownika (GUI – ang. *graphical user interface*) do obsługi tworzonego systemu.

2. Koncepcja przykładowego systemu IoT/WSN

W proponowanym rozwiązaniu jako urządzenie serwera wymiany/akwizycji danych zastosowano minikomputer Raspberry Pi z systemem operacyjnym Raspberry Pi OS opartym na systemie Debian. Na serwerze tym zainstalowano narzędzie do tworzenia przepływów (wybrano narzędzie Node-Red) oraz Broker MQTT, który odpowiada za komunikację między poszczególnymi węzłami i narzędziem Node-Red. Natomiast do realizacji węzła pomiarowego użyto modułu rozwojowego Wemos D1 mini (bazującego na układzie ESP8266), który umożliwia komunikację zarówno z podłączonymi do niego elementami (czujniki, przyciski, diody) jak i wymianę danych z serwerem MQTT za pośrednictwem bezprzewodowej sieci lokalnej (WLAN - *ang. Wireless Local Area Network*). Układ Wemos D1 mini, posiadający bezprzewodowy interfejs sieciowy, zaprogramowano w środowisku programistycznym Arduino IDE z użyciem języka C. W środowisku tym dostępny jest szereg bibliotek do obsługi konkretnych czujników i innych elementów składowych, co bardzo przyspiesza czas realizacji oprogramowania.

Jak wspomniano, w projekcie wykorzystano protokół MQTT (rys. 4).



Rys. 4. MQTT – zasada działania

Fig. 4. MQTT - principle of operation

MQTT jest prostym protokołem przesyłania wiadomości, przeznaczonym przede wszystkim dla urządzeń o niskiej przepustowości. Protokół ten umożliwia wysyłanie poleceń do sterowania wyjściami, odczytywanie i publikowanie danych z węzłów czujników i wiele więcej[2]. Cała istota tego protokołu skupia się wokół wysyłania komunikatów do serwera (Broker MQTT) na dany temat lub nasłuchiwania komunikatów z serwera na dany temat. W przypadku realizowanego systemu Broker MQTT zostanie zainstalowany na Raspberry Pi (program klient-serwer Mosquitto). Wszelakie komunikaty nadawane przez urządzenia klienckie (w tym przypadku Wemos D1 mini), jak i te wysyłane do poszczególnych węzłów (z poziomu środowiska Node-Red), będą kierowane do serwera a pośrednictwem sieci komputerowej. Broker, przejmując te komunikaty, będzie umożliwiał ich odczyt dla urządzeń, które nasłuchują na dany temat. Komunikaty/rozkazy to wiadomości nadawane po znaku „/", co zobrazowano na rysunku nr 4. Protokół ten umożliwia również ustawienie odpowiednich flag w niektórych typach nagłówków ramki danych. Flagi te funkcjonalność tego protokołu (np. poprzez definiowanie nazwy użytkownika, hasła, jakości usług itp.)

Pojęcia związane z MQTT to [4]:

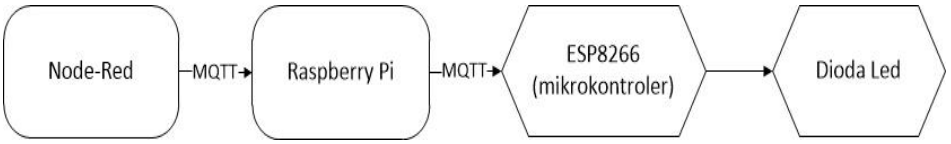
- **Publisher/Subscriber** (publikujący/subskrybujący) – wybrane urządzenie publikuje informację na dany temat, z kolei inne nasłuchuje na ten temat i odczytuję tę wiadomość z serwera;
- **Wiadomości** – komunikaty/rozkazy wymieniane pośród urządzeń;
- **Tematy** – to słowa lub znaki oddzielone ukośnikami (np. /biuro/pokoj/wilgoc). Temat ma format łańcucha znaków i może posiadać on wiele poziomów (słów, znaków oddzielonych ukośnikami). Serwer używa tematu w celu filtrowania wiadomości od różnych podłączonych do niego urządzeń. Zapis tematów jest łudząco podobny do ścieżek plików w systemach operacyjnych;
- **Quality of Service (jakość usług**) – zdefiniowane są trzy poziomy QoS (0,1,2). Zależnie od wyboru poziomu jakości usług, czas transmisji może się wydłużyć w celu zwiększenia pewności wysłania komunikatu;
- **MQTT Broker** – serwer obsługi protokołu MQTT uruchamiany na komputerze/urządzeniu w postaci programu.

Projektowany system, oprócz serwera MQTT potrzebuje też odpowiedniego interfejsu sterującego wydawaniem rozkazów i odczytem danych. W roli tego narzędzia świetnie sprawdzi się wspomniana wcześniej aplikacja Node-Red[3]. Narzędzie to używając protokołu MQTT umożliwi wymianę danych między węzłem a serwerem (rys. 5., 6.) [4].

---

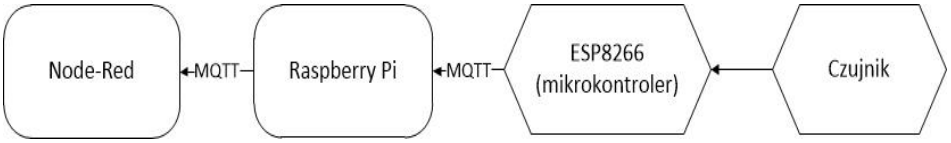
[2] Opis protokołu MQTT: https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/ (dostęp: 25.10.2021)

[3] Opis czym jest Node-Red: https://onixarts.pl/blog/2018/10/daria-node-red/ (dostęp: 25.10.2021)



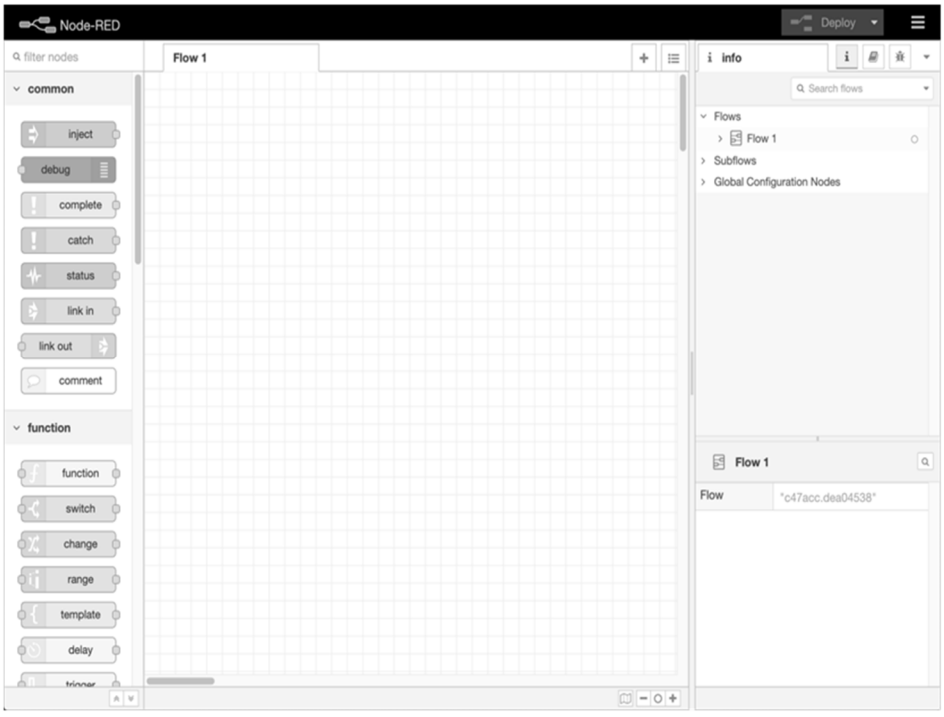
Rys. 5. MQTT - realizacja rozkazu

Fig. 5. MQTT - command execution



Rys. 6. MQTT - realizacja odczytu

Fig. 6. MQTT - implementation of reading



Rys. 7. Node-Red - edytor[4]

Fig. 7. Node-Red - editor[4]

[4] Podręcznik użytkownika dla Node-Red: https://nodered.org/docs/user-guide/editor/ (Dostęp: 25.10.2021)

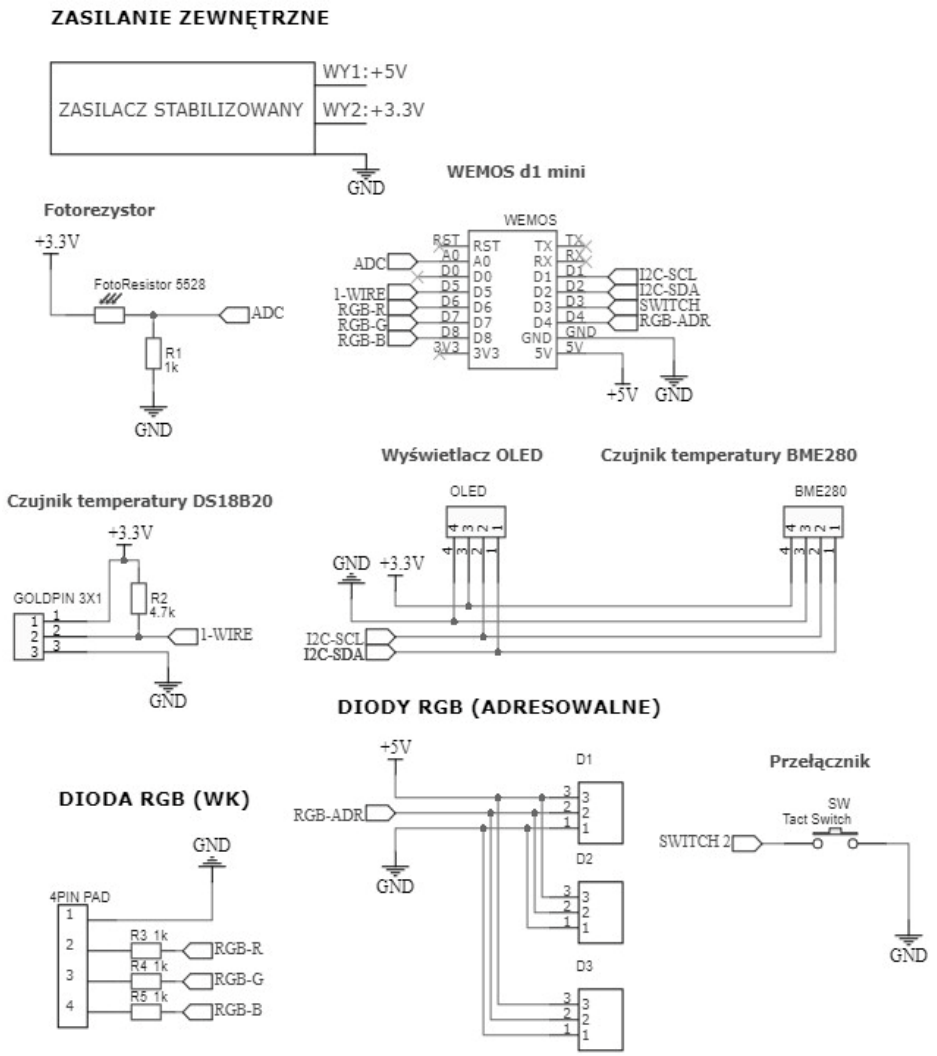
Node-Red to środowisko do modelowania procesów, bazujące na wizualnym tworzeniu kodu programu. Node-Red pracuje w środowisku Node.js i jest niejako graficzną nakładką z wybranymi funkcjami dla środowiska Node.js. Node.js jest środowiskiem uruchomieniowym dla programów pisanych w języku JavaScript. Dzięki środowisku Node.js można tworzyć własne tzw. wtyczki (ang. plug-in) do narzędzia Node-Red oraz tworzyć autonomiczne projekty. Tworzenie przepływów w Node-Red (rys. 7.) odbywa się poprzez łączenie bloków funkcjonalnych, które spełniają określone zadania.

Połączenie możliwości jakie daje protokół MQTT, narzędzie Node-Red oraz środowisko Node.js jest świetnym wariantem dla tworzenia systemów IoT.

3. Realizacja przykładowego systemu IoT/WSN

Tworzenie systemu rozpoczęto od przygotowania schematu połączeń wszystkich komponentów systemu z modułem WEMOS D1 mini (rys. 8.). W tym celu wykorzystano darmowe oprogramowanie EasyEDA. Wszelkie połączenia opracowanego systemu zrealizowano na prototypowej płytce stykowej, która umożliwia swobodny montaż oraz demontaż elementów oraz wszystkich połączeń pomiędzy elementami. Dzięki wykorzystaniu płytki stykowej w procesie tworzenia systemu możliwa była natychmiastowa zmiana ułożenia elementów, wykonanie niezbędnych połączeń, itd.

W kolejnym kroku zaprogramowano moduł WEMOS D1 mini w języku C za pośrednictwem środowiska Arduino IDE, wykorzystując przy tym ogólnodostępne biblioteki umożliwiające obsługę poszczególnych elementów podłączonych do modułu WEMOS. Następnie zrealizowano instalację oraz konfigurację potrzebnych komponentów i usług na serwerze (Raspberry Pi). Projekt został tworzony również pod kątem późniejszego wykorzystania w dydaktyce systemów IoT, w związku z czym zastosowano wiele różnorodnych elementów (adresowalne diody RGB obsługiwane cyfrowo przez 1-Wire, tradycyjna dioda RGB, wyświetlacz OLED, przełącznik, czujniki).



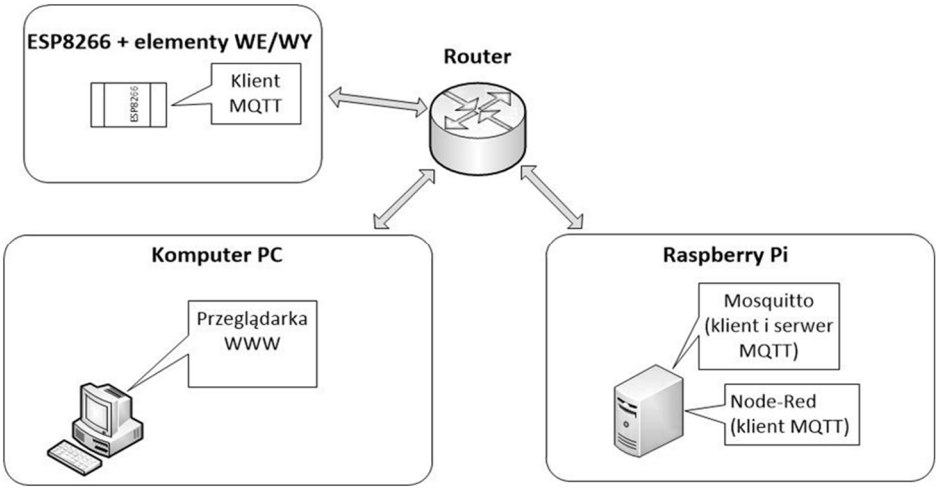
Rys. 8. Schemat połączeń

Fig. 8. Wiring diagram

4. Weryfikacja funkcjonowania systemu

W tej części niniejszego artykułu przeprowadzono testy poprawności działania prototypowego systemu IoT, a także odniesiono się do jego zabezpieczeń.

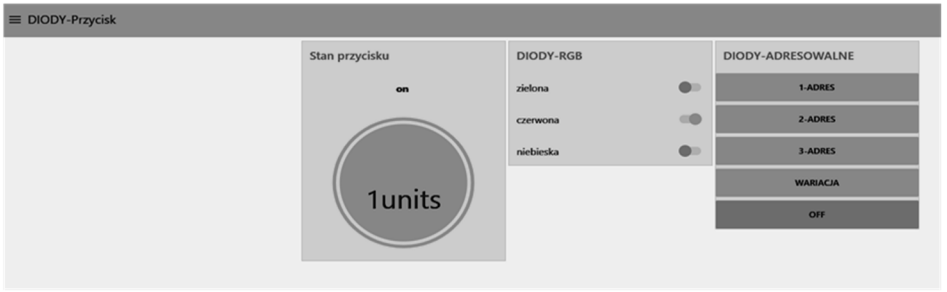
Logowanie do interfejsu użytkownika na serwerze przewidziano z poziomu PC z użyciem przeglądarki internetowej (rys. 9.).



Rys. 9. Architektura działania

Fig. 9. Architecture of operation

W pierwszym kroku sprawdzono poprawność działania interfejsu „DIODY_Przycisk" (rys. 11.). Interfejs ten poprawnie komunikował się z przygotowanym układem węzła, umożliwiał odbiór informacji o stanie przycisku podłączonego do mikrokontrolera, a także sterowanie dostępnymi diodami.



Rys. 11. Testowanie interfejsu „DIODY-Przycisk"

Fig. 11. Testing of the "DIODY-Przycisk" interface

W kolejnym kroku sprawdzono poprawność działania interfejsu „CZUJNIKI" (Rys. 10.). Interfejs ten poprawnie komunikował się z przygotowanym układem węzła, umożliwiał odbiór danych z czujników, zapis danych do pliku ".txt" i sterowanie załączaniem wyświetlacza.

Rys. 10. Testowanie interfejsu „CZUJNIKI”

Fig. 10. Testing of the "CZUJNIKI" interface

Podczas realizacji testów okazało się, że również wielkość liter subskrybowanego/publikowanego tematu ma znaczenie (przykładowo „/dom/salon” oraz „/Dom/salon” to już dwa różne tematy).

W trakcie sprawdzania poprawności działania systemu stwierdzono, że warto byłoby zabezpieczyć edytor Node-Red przed obcymi użytkownikami. W tym celu skonfigurowano odpowiednie pliki systemowe powiązane z Node-Red na serwerze Raspberry Pi [5].

Zaszyfrowano hasło dedykowanym szyfratorem dostarczonym ze środowiskiem Node-Red, a następnie dodano wytworzony ciąg znaków do odpowiednich plików konfiguracyjnych tejże usługi. Całość operacji wykonywano z poziomu terminala systemu Raspberry Pi. W kolejnym kroku wykonano restart usługi Node-Red (rys. 12.).

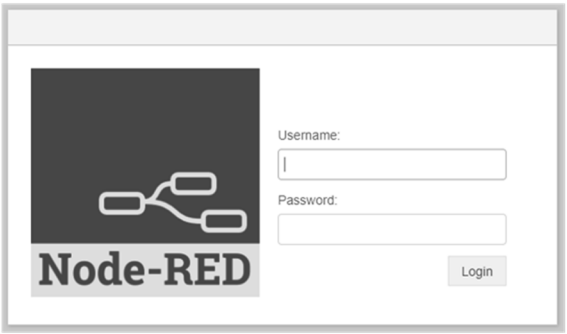
Podczas ponownego uruchomienia pojawił się już komunikat o podanie hasła dostępu.

```
pi@raspberrypi:~/.node-red $ cd
pi@raspberrypi:~ $ systemctl restart nodered.service
==== AUTHENTICATING FOR org.freedesktop.systemd1.manage-units ===
Wymagane jest uwierzytelnienie, aby ponownie uruchomić jednostkę „nodered.service”.
Authenticating as: ,,, (pi)
Password:
==== AUTHENTICATION COMPLETE ===
pi@raspberrypi:~ $
```

Rys. 12. Restart środowiska Node-Red w celu implementacji hasła

Fig. 12. Restart of the Node-Red environment to implement the password

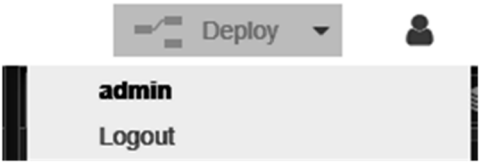
Po restarcie usługi sprawdzono, czy działają poprawnie zabezpieczenia dla edytora przepływów (rys. 13.).



Rys. 13. Edytor przepływów - logowanie

Fig. 13. Flow editor - login

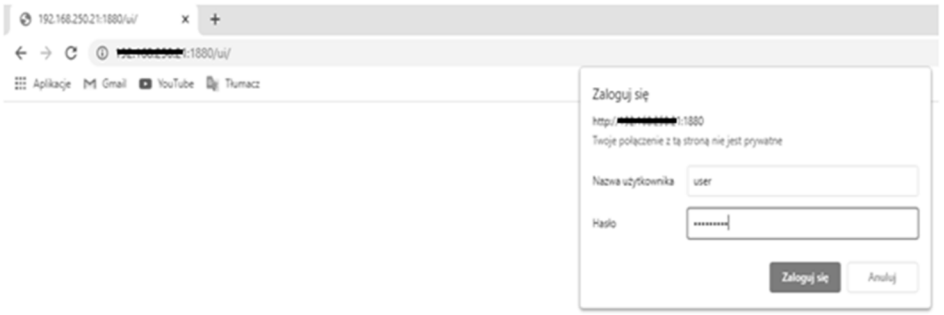
Po zalogowaniu się do edytora klient pozostaje zalogowany domyślnie przez 7 dni (okres czasu zalogowania można ustalić w pliku). Aby wylogować się z edytora należy w tym przypadku wybrać opcję "wylogowanie", pokazaną na rysunku 14., gdyż zamknięcie przeglądarki nie jest wystarczające.



Rys. 14 Zalogowany użytkownik

Fig. 14. Logged in user

Opisane powyżej zabezpieczenie obejmuje tylko edytor przepływów, ale nie interfejs użytkownika. Stwierdzono, że w przypadku poufnych danych należy także zabezpieczyć UI [6]. W celu generacji szyfru hasła można było skorzystać z dostępnego narzędzia dla edytora. Przygotowano odpowiednią konfigurację z poziomu terminala. Po konfiguracji zrestartowano usługę Node-Red. Podczas próby wejścia do GUI trzeba było już podać login i hasło (rys. 15.). W celu zakończenia sesji, należało zamknąć przeglądarkę (przeglądarka zapisuje dane o sesji do przydzielonej pamięci, po jej wyłączeniu ten rejestr pamięci jest resetowany).



Rys. 15. Zabezpieczenia interfejsu użytkownika

Fig. 15. User interface security

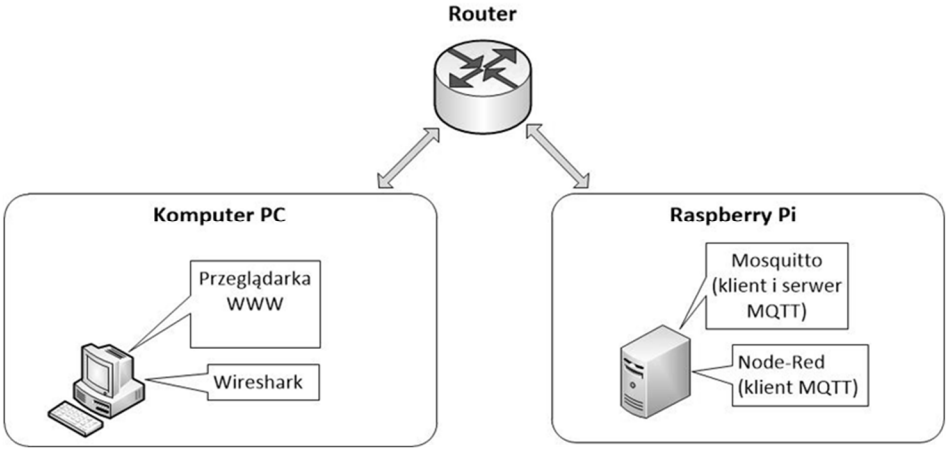
Po powtórnym uruchomieniu przeglądarki ponownie pojawiła się prośba o podanie nazwy użytkownika ze stosownym hasłem.

Kolejnym etapem było sprawdzanie widoczności informacji przesyłanych w lokalnej sieci. Przy podstawowej konfiguracji narzędzia Node-Red i serwera MQTT (Mosquitto) dane przesyłane między serwerem a klientem nie są szyfrowane, a także nie jest wymagane uwierzytelnienie (konkretny użytkownik oraz hasło do publikacji/subskrypcji).

W pierwszym kroku sprawdzono, czy uda się przechwycić pakiet danych z informacjami o logowaniu do edytora przepływów uruchomionego na mini komputerze Raspberry Pi. Do tego celu wykorzystano program Wireshark[5]. Wspomniany program umożliwia przechwytywanie pakietów na wybranym interfejsie danego komputera oraz ich analizę. Ułatwia również rozpoznawanie różnych protokołów komunikacyjnych. Dzięki dostępnym filtrom istnieje możliwość zawężania wyszukiwań (konkretne adresy IP, protokoły, porty itp.). Jego główną zaletą jest przyjazny, graficzny interfejs użytkownika. Program sam w sobie, nie posiada opcji umożliwiających podszywanie się pod inne urządzenia w sieci i przechwytywanie płynącego do nich ruchu. Do tego celu należałoby wykorzystać inne oprogramowanie, a w programie Wireshark przeprowadzać już samą analizę przechwyconych danych

Ideę postępowania w przypadku przechwytywania ruchu na danym interfejsie przedstawiono na rysunku 16.

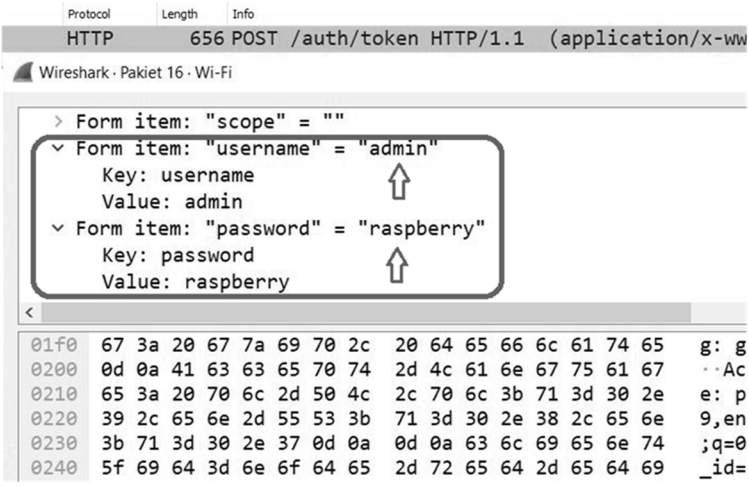
[5] Informacje na temat programu Wireshark: https://www.wireshark.org (Dostęp: 26.10.2021)



Rys. 16. Architektura działania – przechwytywanie danych logowania do edytora

Fig. 16. Architecture of operation – capturing login credentials to the editor

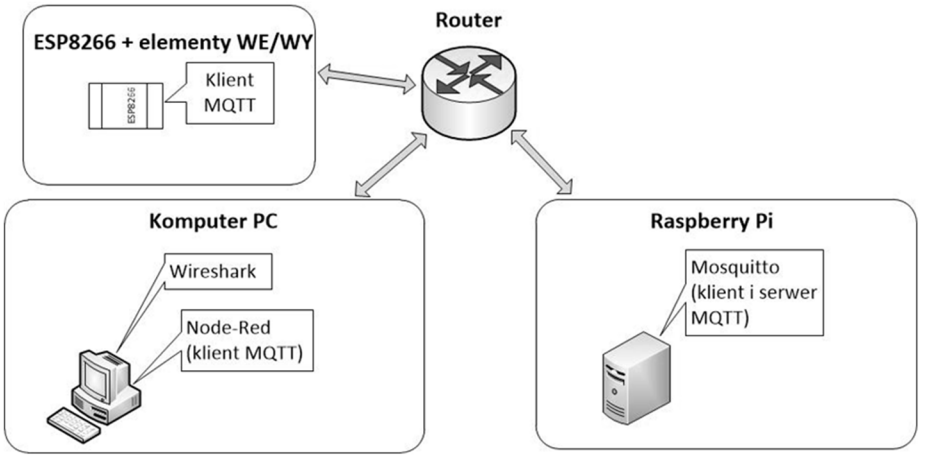
Dane te wysyłane były protokołem HTTP, przez co nie były szyfrowane (w przeciwieństwie np. do protokołu HTTPS). Udało się przechwycić pakiet danych (rys. 17.).



Rys. 17. Przechwycenie logowania do edytora przepływów

Fig. 17. Interception of login to the flow editor

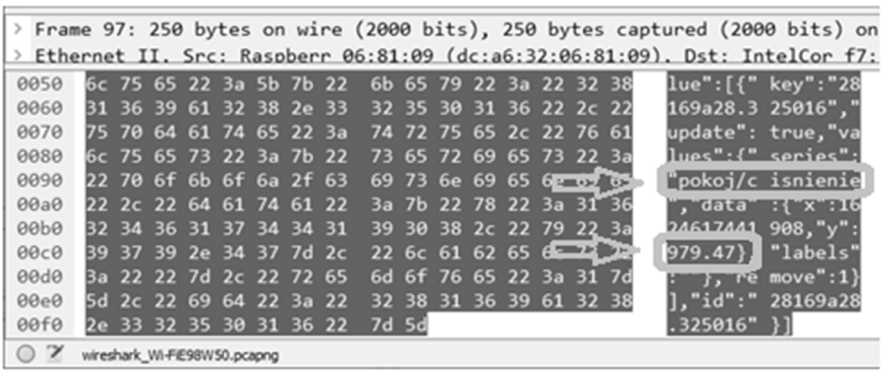
Następnie w innym narzędziu Node-Red, uruchomionym na komputerze PC, dodano adres IP serwera oraz temat do subskrypcji. Ideę postępowania przedstawiono na rys. 18.



Rys. 18. Architektura działania - przechwytywanie danych na konkretny temat

Fig. 18. Architecture of operation - capturing data on a specific topic

Po przechwyceniu ruchu na interfejsie komputera PC z powodzeniem można było odczytać treść przechwyconych pakietów, które przedstawione zostały jawnym tekstem. Na rysunku 19. zaprezentowano przechwycone dane dotyczące wartości ciśnienia.



Rys. 19. Odczyt pakietów danych w programie Wireshark

Fig. 19. Reading packets in Wireshark

Ruch filtrowany jest na konkretnym interfejsie urządzenia. Aby móc obserwować treść publikowanych wiadomości w tym przypadku, należy wykorzystać jedną z podanych możliwości:

- zalogować się na serwerze (Raspberry Pi) i skanować ruch na jego interfejsie sieciowym dowolnym oprogramowaniem;
- znać konkretne tematy, na które jest możliwość subskrypcji/publikacji oraz uruchomić klienta MQTT na swoim urządzeniu podając adres serwera MQTT (tak było w tym przypadku) i wtedy śledząc ruch na własnym interfejsie sieciowym skanować dane publikowane na konkretny znany wcześniej temat (bez znajomości tematu nie da się w tym przypadku nic zobaczyć);
- skanować ruch na interfejsie urządzenia sieciowego, do którego podłączony jest serwer (proste, najczęściej spotykane w użytku domowym urządzenia nie mają takiej opcji);
- próbować przeprowadzić atak typu Spoofing (podszywać się pod inne urządzenie w sieci i przechwytywać ruch doń kierowany).

Istnieją oczywiście różne zaawansowane możliwości wdrożenia szyfrowania informacji. Jedną z takich opcji jest skonfigurowanie serwera MQTT oraz każdego z podłączonych klientów do korzystania z protokołu bezpiecznej transmisji zaszyfrowanego strumienia danych (protokół SSL bądź TLS). Po analizach dokumentacji usługi Mosquitto, Node-Red oraz Wemos okazało się, że było by to wykonalne. Na potrzeby zaprojektowanego systemu IoT zdecydowano się na prostsze zabezpieczenie, które w wystarczy dla domowego systemu IoT.

Wybranym, dodatkowym zabezpieczeniem systemu, oprócz wiadomego zabezpieczenia edytora przepływów oraz UI, było uruchomienie uwierzytelniania (poprzez użytkownika oraz hasło) na serwerze Mosquitto [7]. W takim wypadku każdy klient chcący subskrybować bądź publikować dane dla odpowiedniego tematu musi wykonać uwierzytelnienie z serwerem (użytkownik/hasło). Aby dodać taką opcję do serwera Mosquitto należało utworzyć plik ".txt", do którego wygenerowano szyfr hasła z użyciem dedykowanego szyfratora dostępnego z usługą Mosquitto. Następnie odpowiednio ustawiono plik konfiguracyjny usługi serwera MQTT (który uprzednio stworzono), uruchomionego na Raspberry Pi i wykonano restart usługi Mosquitto.

W przypadku węzła opartego na module Wemos D1 mini należało dodać argumenty do już działającej w kodzie programu funkcji obsługującej uwierzytelnianie. W celu uruchomienia poświadczeń dla serwera MQTT należało uzupełnić wspomnianą funkcję o argumenty (użytkownik oraz hasło). Po takiej konfiguracji podczas działania programu klient zaczął komunikację z serwerem MQTT podając identyfikator użytkownika (np: „KlientWemos”), następnie nazwę użytkownika stworzonego na serwerze Mosquitto (serwer MQTT) oraz hasło. Dane do uwierzytelniania należało też wpisać w odpowiednie ustawienia narzędzia Node-Red, który także jest klientem MQTT.

Po konfiguracji wszystkich elementów sprawdzono poprawność komunikacji. Wszystkie elementy interfejsu użytkownika działały poprawnie. Oznacza to że zarówno Wemos D1 mini, jak i narzędzie Node-Red poprawnie realizują uwierzytelnianie. Wykonano szybki test publikacji za pomocą klienta MQTT z poziomu terminala minikomputera Raspberry Pi (działa on jako "serwer", aczkolwiek może też wysyłać dane sam do siebie z poziomu "klienta"). W przypadku podania błędnego hasła lub nazwy użytkownika do serwera MQTT pojawił się komunikat o błędzie (rys. 20.).

```
pi@raspberrypi:/etc/mosquitto/conf.d $ mosquitto_pub -t "/home/button" -m "on" -u "student" -P "student"
pi@raspberrypi:/etc/mosquitto/conf.d $ mosquitto_pub -t "/home/button" -m "off" -u "student" -P "student"
pi@raspberrypi:/etc/mosquitto/conf.d $ mosquitto_pub -t "/home/button" -m "on" -u "student" -P "student2"
Connection Refused: not authorised.
Error: The connection was refused.
pi@raspberrypi:/etc/mosquitto/conf.d $
```

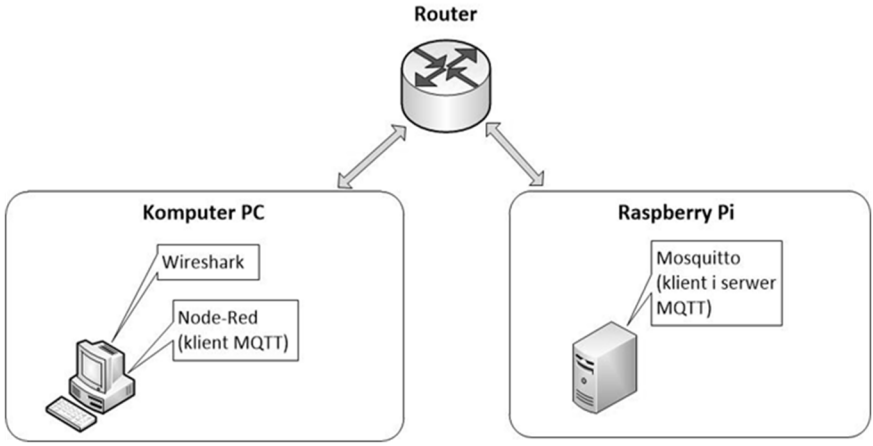
Rys. 20. Próby publikacji z poziomu terminala

Fig. 20. Attempts to publish from the terminal

Sprawdzono też widoczność przeprowadzanego uwierzytelniania (klienta z serwerem) w sieci z poziomu komputera z zainstalowanym klientem MQTT. Ruch badany był na bezprzewodowym interfejsie komputera PC, z którego próbowano połączyć się z serwerem z poziomu narzędzia Node-Red (rys. 21). Narzędzie Node-Red posiada bowiem odpowiednie, bloki umożliwiające subskrypcję/publikację na dany temat i dodatkowe ustawienia odnośnie dodawania zabezpieczeń (certyfikaty poświadczeń, hasła).

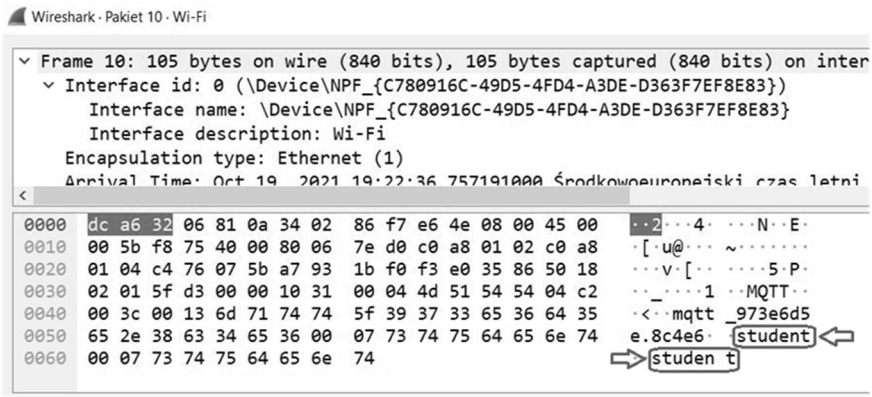
W tym przypadku wysłanie danych do serwera odbyło się tylko raz (rys. 22.).

W sytuacji, w której nieautoryzowany użytkownik byłby w stanie subskrybować/publikować znając adres serwera i wymagane tematy, to i tak nie będzie mógł odbierać/publikować tych danych dopóki nie poda nazwy użytkownika oraz hasła dostępu. Jeśli nawet chciałby przechwycić dane bez podawania tematów przesyłanych wiadomości, bądź też same dane do logowania, musiałby mieć zdalny dostęp do serwera i na nim zainstalować program do przechwytywania pakietów (np. TCPDUMP), ewentualnie mieć dostęp do urządzenia sieciowego, przez które przechodzi cały ruch kierowany z/do serwera. Zawsze istnieje ryzyko ataku typu "Spoofing", podczas którego istnieje duże ryzyko pomyślnego przechwycenia danych (wystarczy wtedy by obcy użytkownik znalazł się w sieci lokalnej, czy to fizycznie, czy też z użyciem VPN). Dlatego ważne jest odpowiednie zabezpieczenie serwera oraz urządzenia sieciowego oraz wyłączenie wszelkich kanałów dostępu (ssh, telnet, itp.), jeśli nie jest konieczne ich użycie do zdalnej konfiguracji serwera, a także ograniczenie dostępu osób postronnych



Rys. 21. Architektura działania – przechwytywanie uwierzytelniania MQTT

Fig. 21. Architecture of operation – capturing MQTT authentication



Rys. 22. Przechwycenie uwierzytelniania MQTT

Fig. 22. Intercepting MQTT authentication

do własnej sieci lokalnej. Dobrze zabezpieczony serwer oraz sieć, do której jest podłączony, to większa gwarancja na bezpieczeństwo poufnych danych nawet jeśli takowe byłyby szyfrowane.

5. Podsumowanie

W pracy przedstawiono projekt oraz wyniki badań przykładowego systemu IoT, implementującego typowe elementy takiego rozwiązania.

Dokonano ogólnej analizy działania oraz przeprowadzono badanie modelowego systemu IoT z wykorzystaniem narzędzia do monitorowania ruchu danych w sieci komputerowej. Podczas realizacji części badawczej stwierdzono, że szczególną uwagę należy zwrócić na możliwe opcje późniejszego zabezpieczenia tworzonego systemu oraz wsparcie ze strony producenta i innych źródeł. Pierwotna koncepcja realizowanego systemu zmieniała się bardzo dynamicznie, gdyż możliwych było wiele różnych konfiguracji od strony programowej, jak i zastosowanie różnego rodzaju urządzeń. Wybrany do realizacji projektu moduł bazujący na ESP8266 okazał się być wydajnym i działał niezawodnie obsługując wiele zadań na raz. Testowanie bezpieczeństwa usług uruchomionych na serwerze pokazało, na co należy zwrócić uwagę podczas korzystania z tego typu systemów. W pierwotnej wersji usługi uruchomione na serwerze nie posiadały żadnych zabezpieczeń. Po analizie możliwych schematów włamań do tego systemu uruchomiono zabezpieczenia narzędzia Node-Red (edytora przepływów oraz interfejsu użytkownika).

Przy okazji późniejszego badania okazało się, że mając dostęp do serwera (zdalnie bądź stacjonarnie), można filtrować pakiety i sprawdzać ich zawartość. Stwierdzono, że znając adres serwera oraz tematy publikowanych przez niego wiadomości, można odbierać te dane i przetwarzać swobodnie na innym serwerze, a także wysyłać rozkazy (np. znając treść wiadomości oraz tematu potrzebnego do uruchomienia oświetlenia). Po przeglądzie dokumentacji dostawcy usługi „Mosquitto" okazało się, że istnieją różne rodzaje zabezpieczeń możliwe do wdrożenia. Wybrano zabezpieczenie, które wykorzystywało mechanizmy uwierzytelniania (podanie nazwy użytkownika oraz hasła podczas publikacji/subskrypcji do serwera na dany temat). Po dokonaniu odpowiedniej konfiguracji serwera skonfigurowano również urządzenia klienckie. Okazało się, że w połączeniu z zabezpieczeniami narzędzia Node-Red pozwoliło to na podstawowe ograniczenie możliwości ingerencji osób postronnych w działanie systemu oraz przechwytywanie poufnych danych. Jeżeli serwer będzie odpowiednio zabezpieczony, to wprowadzone konfiguracje w powinny wystarczyć. W przypadku bardzo poufnych danych należałoby wdrożyć obsługę protokołu SSL/TLS (szyfrowanie danych).

Podziękowania

Projekt finansowany w ramach programu Ministra Edukacji i Nauki pod nazwą „Regionalna Inicjatywa Doskonałości" w latach 2019-2022 nr projektu 027/RID/2018/19 kwota finansowania 11 999 900 zł.

Literatura

[1] Culic I., Radovici A., Rusu C.: Komercyjne i przemysłowe aplikacje Internetu Rzeczy na Raspberry Pi. APN Promise, Warszawa 2020

[2] Dominique Guinard, Vlad Trifa.: Internet Rzeczy, Helion, Gliwice, 2017
[3] Opis WSN: http://www.wsn.agh.edu.pl/?q=pl/node/164 (dostęp: 26.10.2021)
[4] Opis protokołu MQTT: https://randomnerdtutorials.com/what-is-mqtt-and-how-it-works/ (dostęp: 25.10.2021)
[5] Ochrona narzędzia Node-Red: https://nodered.org/docs/user-guide/runtime/securing-node-red (dostęp: 26.10.2021)
[6] Zabezpieczanie Node-Red (forum): https://discourse.nodered.org/t/adding-login-to-ui-dashboard-as-flow-page/37770/7 (dostęp: 26.10.2021)
[7] Zabezpieczanie serwera MQTT: https://www.vultr.com/docs/how-to-install-mosquitto-mqtt-broker-server-on-ubuntu-16-04 (dostęp: 26.10.2021)

SAFETY AND RELIABILITY IN WIRELESS SENSOR NETWORKS ON THE EXAMPLE OF SELECTED SOLUTIONS

Summary

The article presents basic information on the Internet of Things and wireless sensor networks. Then a prototype IoT system was designed, its operation was described in general and the available security measures for such systems were presented. On the example of selected hardware and software solutions, the possibilities and tools for remote management, integration and organization of data exchange were analyzed. On this basis, a description of selected, exemplary practical solutions was presented and a methodology for selecting, designing and programming such systems was proposed. The connection diagram for a single network node is presented, the services running on the server are briefly discussed and the MQTT protocol (data exchange via a computer network) is generally characterized.

Basic security has been implemented in the developed system. All communication took place without information encryption (data sent in clear text). The results of basic research carried out with the use of the Wireshark program are also presented, as well as the potential forms of protection of IoT systems. The main purpose of the article below was to show the great importance of data transmission security in such systems.

Keywords: internet of things, sensors, wireless modules, node-red, mqtt, raspberry pi

DOI: 10.7862/re.2022.4

Submitted/Tekst złożono w redakcji: listopad 2021 r.
Accepted / Przyjęto do druku: month.year
Published/Tekst opublikowano: month.year